HPE SOA Registry Foundation

Software Version: 10.04

Windows and Linux Operating System

Product Documentation

Document Release Date: July 2017
Software Release Date: July 2017

Hewlett Packard
Enterprise

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

To check for recent updates or to verify that you are using the most recent edition of a document, go to: https://softwaresupport.hpe.com/.

This site requires that you register for an HP Passport and to sign in. To register for an HP Passport ID, click **Register** on the HPE Software Support site or click **Create an Account** on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

Support

Visit the HPE Software Support site at: https://softwaresupport.hpe.com/.

Most of the support areas require that you register as an HP Passport user and to sign in. Many also require a support contract. To register for an HP Passport ID, click **Register** on the HPE Support site or click **Create an Account** on the HP Passport login page.

To find more information about access levels, go to: https://softwaresupport.hpe.com/web/softwaresupport/access-levels.

HPE Software Solutions Now accesses the HPSW Solution and Integration Portal website. This site enables you to explore HPE Product Solutions to meet your business needs, includes a full list of Integrations between HPE Products, as well as a listing of ITIL Processes. The URL for this website is https://softwaresupport.hpe.com/km/KM01702731.

About this PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

Contents

Chapter 1: Read This First

Welcome to HPE SOA Registry Foundation!

HPE SOA Registry Foundation is the leading business service registry, providing discovery and publishing of SOA business services. With full support for version 3 of the UDDI (Universal Description, Discovery and Integration) standard, HPE SOA Registry Foundation 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.

Installation and Deployment Guide This book guides you through installing HPE SOA Registry Foundation, installing and setting up databases, and deploying HPE SOA Registry Foundation to application servers.

User's Guide This book describes how to manually maintain HPE SOA Registry Foundation contents. All basic functions of the Registry Console are discussed here.

Developer's Guide Introduces the basics of creating extensions and client programs in HPE SOA Registry Foundation. The Developer's Guide also documents the HPE SOA Registry Foundation demo suite.

Administrator's Guide Explains HPE SOA Registry Foundation's configuration and management, and introduces the tools and utilities you will need to perform these tasks.

HPE SOA Registry Foundation Features Overview

HPE SOA Registry Foundation 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). HPE SOA Registry Foundation is an easy-to-use, standards-based mechanism for publishing and discovering Web services and related resources like XML Schemas.

HPE SOA Registry Foundation fully implements the OASIS UDDI V3 standard. HPE SOA Registry Foundation 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. HPE SOA Registry Foundation is also easy to customize to support different enterprise user communities.

HPE SOA Registry Foundation 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. HPE SOA Registry clearly defines responsibility for every registered component.
- **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).
- **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.

HPE SOA Registry Foundation 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, ranging from Tomcat to BEA WebLogic, IBM Websphere or JBoss are supported. HPE SOA Registry Foundation also unrivalled support for a broad set of database management systems for storing registrations (such as Oracle, MS SQL Server, Sybase, PostgreSQL and HSQL). Crucially, HPE SOA Registry Foundation also integrates with both LDAP and Microsoft ActiveDirectory.

Release Notes

Known Issues

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 is used for canonicalization of digital signatures. Schema-centric XML Canonicalization is not yet implemented.

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.

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.

- We do not recommend installing HPE SOA Registry Foundation with the HSQL database under IBM Java 1.4.x since the installation may time out.

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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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.
- 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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation configuration using the Setup tool, demo data is always imported the registry database.

Change Log

HPE SOA Registry Foundation 10.0x

- Support for Oracle databases 11g and 12c.
- Support for MS SQL databases 2005 and 2008.
- Support for JDK 1.7

HPE SOA Registry Foundation 6.65

- The configurations in the Database feature enables the simple configuration of cluster deployment. The database can also hold a history of configuration files. The administration console enables you to display differences between current and past configurations and stored configuration collections.
- Replications are improved and more reliable.
- Client certificate authentication (Two Way SSL) is supported.
- IPv6 is supported except for literal addresses.
- Enhance the subscription UI.
- Support for publishing WSDL with empty URL in soap:action location.
- Resolves the following security issues

    - Prevent Published files from residing on the server file system.
    - Closing of cross-site scripting (XSS) security vulnerabilities.
    - Verification of session data integrity.
    - Increase access point length for UDDI V2.
- SAP NetWare Service Registry integration - Setup tool supports for changing hostname, HTTP(S) ports and username to integrate with SAP NetWare Service Registry and enabling the HTTP Basic Authentication.
- Support for changing server URLs.
- Support for Installation on Oracle WebLogic Server 11g with Domain Templates - Installer produces files including a .war file and WebLogic Domain template and places them into the HPE SOA Registry Foundation Home. Upon completion, you can run the Oracle WebLogic Configuration Wizard to deploy to the selected clusters and/or managed servers.
- Bug fixes. Small improvements.

HPE SOA Registry Foundation 6.64

- Tools support for 64-bit Windows (except Itanium).
    - JavaService can run on both 32 and 64 bit Windows.
    - Messages can be logged in the EventLog of 32 and 64 bit Windows.
- Improved SiteMinder group support: users can access the registry with permissions granted via Siteminder Authentication Configuration.
- Database deadlock recovery and prevention.
    - User connects to Registry from multiple threads or computers using any operation (find, get, save, delete). Registry completes all operations even when there is deadlock in the database.
- Bug fixes. Small improvements.

HPE SOA Registry Foundation 6.63

Bug fixes. Small improvements.

HPE SOA Registry Foundation 6.62

Separate Spanish release of 6.61.

HPE SOA Registry Foundation 6.61

- IPv6 support (except for literal IPv6 addresses).
- Removed SSO as newer versions of HPE SOA Systinet (former Systinet 2) do not use it.
- Bug fixes. Small improvements.

HPE SOA Registry Foundation 6.6

- Removed BSC.
- Removed Taxnomoy editor (only viewer left).
- Removed XSLT, XML publising support.

HPE SOA Registry Foundation 6.5.4

Bug fixes. Small improvements.

HPE SOA Registry 6.5.3

- Configuration in Database feature (easy clustering setup, configurations are persisted in case of redeployment).
- Improved replications.
- SSL Certificate authentication. Added sslTool.
- Bug fixes. Small improvements.

Systinet Registry 6.5.2

- Single Sign-On with Systinet 2.
- Bug fixes. Small improvements.

Systinet Registry 6.5.1

Bug fixes. A lot of small improvements.

Systinet Registry 6.5

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

Systinet Registry 6.0

- 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 - Systinet 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 5.5 - 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 - Backup functionality allows you to save the Systinet Registry data and configuration to a filesystem directory. Later the backup data can serve for a full restore of HPE SOA Registry data and configuration.
- Documentation
  - Introduction to HPE SOA Registry Foundation
  - Accessing UDDI from Developer Tools

Systinet Registry 5.5

- Business Service Console - Using the Business Service Console, developers, architects and business users can browse the various perspectives of the Systinet Business Services 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.

- Advanced query capabilities - Range Queries - users can search for UDDI structures using >,< operators when searching by categories.
- Taxonomy management
  - Taxonomy management has been enhanced by drag and drop taxonomy structure editing. You can move a category item in the taxonomy hierarchy without de-associating it with current UDDI entities categorized with this item’s value.
  - Administrators can edit an enterprise taxonomy list. Users can edit their lists of favorite taxonomies.
- Mapping resources. New publishing wizards and APIs. The WSDL2UDDI publishing wizard and API have been enhanced. New wizards and APIs for publishing of resources have be been created.
  - Publish a WSDL document
  - Publish an XML document
  - Publish an XML schema document
  - Publish an XSL Transformation

Systinet Registry 5.0

- UDDI Multi-version Registry
  - UDDI Version 3 Registry - Implementation of the UDDI Version 3 Specification - Committee Specification v3.0.1
  - UDDI Version 2 Registry - Implementation of the UDDI Version 2 Specifications - OASIS Standard
  - UDDI Version 1 Registry - Implementation of the UDDI Version 1 Specifications - contributed
- WSDL Publishing - Implementation of Using WSDL in a UDDI Registry, Version 2.0 for UDDI Version 2 and Version 3
- Access Control - Allows definition of granular access control for registered components. Component publisher can specify find, get, modify, and delete access permissions for every published object.
- Account and Group Management - Allows management of user's account and groups.
- External Accounts Integration - Allows integration of the registry with custom account storages

including three integration scenarios with LDAP.

- Taxonomy Management and Validation - Allows administrator to create, download, upload, browse and manage taxonomies.
- Approval Process - component promotion and approval mechanisms for promoting components between development, staging, and production environments.
- Selective One-way Replication - Replication based on subscription-notification mechanism. An asynchronous subscription listener listens to incoming subscription data from a master registry.
- Registry Console - User-friendly UI enables user to query and publish the registry, manage user's account and provide various administration tasks.
- Administration Tools
  - GUI Setup and Administration Tool - Allows administrator to set up, port, and configure the registry; create and drop the registry database; and migrate data from other registry databases.
  - Web Administration Console - Allows administrator to configure and manage registry permissions, data, and users; configure replications; and view registry access statistics.
- Support for leading database engines including Oracle, MS SQL 2000 or 2005, IBM DB2, PostgreSQL, Sybase, Hypersonic SQL. Systinet Registry contains both a bundled and a pre-configured Hypersonic SQL 1.7.1 database.
- Support for application servers - Systinet Registry supports BEA WebLogic and Apache Tomcat application servers.
- Client Libraries - This distribution includes UDDI Version 1,UDDI Version 2, and UDDI Version 3 account, groups, and permissions management, taxonomy management, approval, administration and configuration clients with generated javadocs.
- Open Server-Side Architecture
  - Registry Integration and Embedding - Developers can directly access instances of registry APIs, run custom classes inside the registry, create custom login modules, and write custom integration with external accounts and groups storages.
  - Registry Extensions - Developers can write their own extension services, create and use external and internal validation services, write custom interceptors to intercept registry messages, customize the approval process, and customize or create their own Registry Console using a supplied JSP Web Framework.

WASP UDDI 4.6

- Evaluation License Enforcement Mechanism - evaluation version of WASP UDDI requires an evaluation license
- Integration with LDAP/MS Active Directory - WASP UDDI; 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 - WASP UDDI 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 WASP UDDI.

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

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 - WebSphere 5.0, JBoss 3.0.4, BEA WebLogic 6.1 SP3, BEA WebLogic 7.0.
- Database installation - Added database installation to WASP UDDI 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 - WASP UDDI can be now run as NT service.

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 WASP UDDI. There are two basic ways how the subscription is used: asynchronous notification and synchronous pull subscription.
- WASP UDDI 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.

WASP UDDI 4.0

- InstallShield - Graphical installation tool, InstallShield added.
- PointBase - Support for PointBase 4.3 database added.
- Oracle 9i - Oracle 9i AS (OC4J) deployment added.
- Disabled Runtime Services - System services removed from WASP UDDI 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 - WASP UDDI now support Sun's implementation of JDK 1.4.
- Deployment - BEA WebLogic, IBM WebSphere, Orion, Tomcat deployment scripts and documentation included.
- Taxonomy and Validation - Additional Taxonomy and Validation services integrated into the web interface.

Supported Platforms

HPE SOA Registry Foundation 10.04 has been tested on the following platforms.

- Operating systems:
  - WindowsServer 2012 R2
  - Windows Server 2008 R2
  - Linux (RedHat 5.6, 6)
  - Linux (Ubuntu 12, 12.4)
  - HPE-UX
  - AIX
  - Solaris
- JDKs
  - Oracle (Sun) JDK 1.7
  - HPE JDK 1.7
- Databases:
  - Oracle 11g
  - Oracle 12c
  - Microsoft SQL 2005(SP2)
  - Microsoft SQL 2008(SP1)
- LDAP:

  - Sun One Directory Server 5.2
  - Sun Java System Directory Server 6.3
  - Microsoft Windows Server 2008 Active Directory
- Application Servers:
  - Oracle WebLogic Server 11g R1 [http://www.oracle.com]
  - Oracle WebLogic Server 10g R3 [http://www.oracle.com]
  - IBM WebSphere 6.1.x and 7.0.0.7 [http://www.ibm.com/software/info1/websphere/index.jsp]
  - JBoss 5.1 [http://www.jboss.org]
  - JBoss EAP 5 [http://www.jboss.org]
- Browsers:
  - Google Chrome (latest version)
  - Microsoft Internet Explorer 10 or newer
  - Mozilla Firefox (latest version)
  - Mozilla Firefox ESR (latest version)

Specifications

HPE SOA Registry Foundation conforms to the following specifications:

- UDDI Specifications
- UDDI Version 1 Specification
- UDDI Version 2 Specification
- UDDI Version 3 Specification
- Technical Note Using WSDL in a UDDI Registry, Version 2.0

Document Conventions

This document uses the following typographical conventions:

`run.bat make`	Script name or other executable command plus mandatory arguments.
[--help]	Command-line option.
either | or	Choice of arguments.
`replace_value`	Command-line argument that should be replaced with an actual value.
{arg1 | arg2}	Choice between two command-line arguments where one or the other is mandatory.
`rmdir /S /Q System32`	User input.
`C:\System.ini`	Filenames, directory names, paths and package names.
`a.append(b);`	Program source code.
`server.Version`	Inline Java class name.
`getVersion()`	Inline Java method name.
Shift+N	Combination of keystrokes.
Service View	Label, word, or phrase in a GUI window, often clickable.
OK	Button in a user interface.
New→Service	Menu option.

Documentation Updates

This guide's title page contains the following identifying information:

- Software version number, which indicates the software version.
- Document release date, which changes each time the document is updated.
- Software release date, which indicates the release date of this version of the software.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HPE Passport and sign-in. To register for an HPE Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please** register link on the HPE Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

Legal

Notices

Legal Notices

Warranty

The only warranties for HPE products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HPE shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HPE required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Third-Party Web Sites

Mercury provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. Mercury makes no representations or warranties whatsoever as to site content or availability.

Copyright Notices

Trademark Notices

Java™ is a US trademark of Sun Microsystems, Inc. Microsoft®, Windows® and Windows XP® are U.S. registered trademarks of Microsoft Corporation. IBM®, AIX® and WebSphere® are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries. BEA® and WebLogic® are registered trademarks of BEA Systems, Inc.

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.

Chapter 2: Installation Guide

HPE SOA Registry Foundation may be installed using the following scenarios:

- **Standalone Registry**

  This is the default installation scenario; under it the HPE SOA Registry Foundation server is installed on a local machine and connects to a local or external registry database. To perform a standalone installation, follow the instructions at “Installation”. For more configuration information, refer to “Server Configuration” and “Database Installation”.

- **Deployed to an Application Server**

  The installed standalone HPE SOA Registry Foundation server may be deployed to several application servers. To deploy HPE SOA Registry Foundation to an application server, perform the standalone installation as described in “Installation” and then follow the instructions in “Deployment to an Application Server”.

- **Standalone registry with data migration**

  In this case, a standalone installation is performed and data is migrated to it from a previous installation of HPE SOA Registry Foundation. Follow the instructions in “Migration”.

- **External Accounts Integration**

  HPE SOA Registry Foundation server may be optionally configured to use external accounts on an LDAP or other account store. It is possible to set up external accounts integration during database installation. For more information, please see “Database Installation” and “External Accounts Integration”

- **Registry cluster**

  A UDDI cluster is a group of UDDI registries deployed on multiple servers possibly with a clustered database in the back-end. Load balancing is used to distribute requests amongst HPE SOA Registry Foundation servers to get the optimal load distribution. Standalone Registry or registry deployed to an application server could be configured to cluster with instructions in “Cluster Configuration”

- **Support for Windows NT service and Unix Daemon**

  HPE SOA Registry Foundation can be run as a service on Windows 2000/XP. Support for NT service installation is installed by default on Windows servers, see instructions in “NT Service

Support”. Also, HPE SOA Registry Foundation can be run as a system daemon on Unix machines, see instructions in “Running in Linux”.

System Requirements

This section explains the requirements that must be met *before* you start installation. "Supported Platforms" on page 31 in "Read This First" summarizes the software platform options in the current release.

You should:

1. Ensure the installation machine meets the requirements that follow in " Hardware" below.
2. Decide which combination of supported platform components will be used.
3. Ensure each component is installed as described in this section.

Then you can proceed with installation.

Hardware

The following table summarizes hardware requirements for the installation machine. The minimum specifications are suitable for experimental use of HPE SOA Registry Foundation on a workstation. Although it may be possible to install the product on a machine with lower specifications, performance and reliability may be severely affected. The requirements of servers in a production environment are greater and depend on patterns of use. See "Supported Platforms" on page 31 in "Read This First" if you need assistance.

Specification	**Minimum**	**Notes**
CPU	Intel Xeon E processor family, 8 cores	Actual requirements depend on the on patterns of use in the target environment.
RAM	16GB	
Disk Space	40GB	This is sufficient if the selected database system is installed on another machine. The database server machine must have sufficient space for the

		selected database system. The requirements for registry data are quite modest. Each GB typically provides for registration of several thousand additional entities. So disk performance is more significant.

Java™ Platform

A supported *Java Development Kit* is required on the installation machine. A Java Runtime Environment is not sufficient because it must be possible to compile JSP pages at runtime.

IBM JDK 1.7 and higher must contain a JCE provider. Bouncy Castle provider is supported, and JCE Unlimited Strength Jurisdiction policy files are required.

1. Copy the file `bcprov-ext-jdk15on-*.jar` from Bouncy Castle provider to `IBMJava2/jre/lib/ext;`
2. Add the following line to the file `java.security` located in `IBMJava2/jre/lib/security:`

   ```
   security.provider.5=org.bouncycastle.jce.provider.BouncyCastleProvider
   ```

Relational Database

Setting up a relational database during installation is optional - you can instead set it up after installation using the setup tool. See "Database Installation" on page 80, and in both cases you can use the pre-configured HSQL database system that comes with HPE SOA Registry Foundation.

The installation process allows you to set up a database using one of the other supported database systems, in which case the database server must be installed and running (not necessarily on the same machine). JDBC driver files must generally be available locally, but some drivers are distributed with HPE SOA Registry Foundation.

Installation

This section describes the standalone installation of HPE SOA Registry Foundation and all settings.

> **Note:** Make sure that the JAVA_HOME environment variable points to your JDK and that your path includes %JAVA_HOME%\bin.

To install the registry, type the following at a command prompt:

java -jar hpe-soa-registry-foundation-10.04.jar

and follow the wizard panels.

If you have associated javaw with `*.jar` files on Windows complete the following step:

- Double-click the icon for the file `hpe-soa-registry-foundation-10.04.jar.`

Command Line Options

Installation can be launched with the following optional arguments:

`java -jar hpe-soa-registry-foundation-10.04.jar` **[[--help] | [-h] | [--gui] | [-g]]**

[[-u *configfile* **] | [--use-config** *configfile* **]]**

[[-s *configfile*] | [--save-config *configfile* **]]**

[[-d] | [--debug]] [[-c] | [--console]] [[-v] | [--version]]

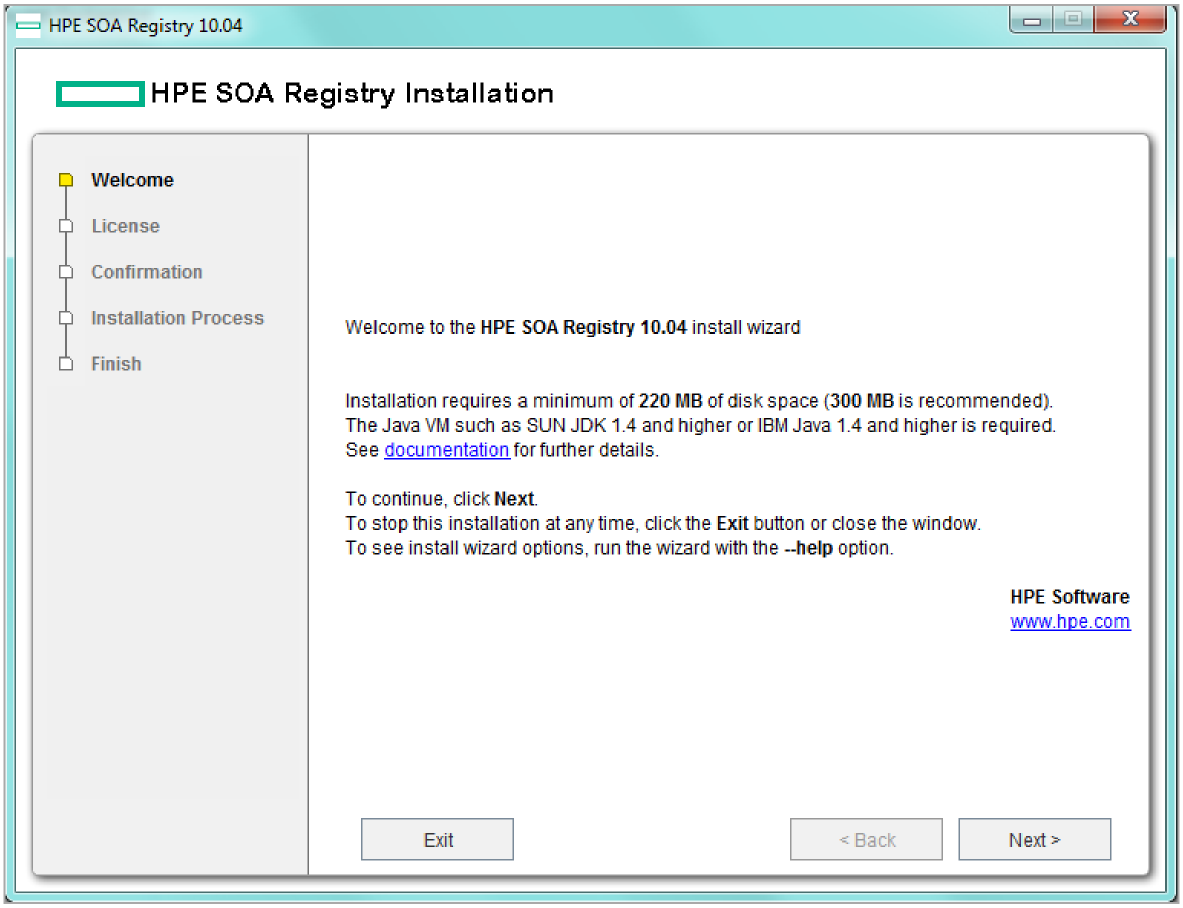
Argument	Description
`-g | [--gui]`	Starts the installation in gui mode (default).
`-c | [--console]`	Runs command-line installation.
`-h | [--help]`	Shows help messages.
`-s configfile | --save-config configfile`	Saves the installation settings into the configuration file without actually installing the registry.
`-u configfile | --use-config configfile`	Installs the registry using the settings contained in the configuration file.
`-d | [--debug]`	The installation produces more information to localize problems or errors.
`-v | [--version]`	Displays version of the product.

Installation Panels

This section discusses the content of the installation wizard. It goes through the following installation panels using default settings.

Welcome Panel

The first panel for the Registry Installation wizard is the Welcome panel. This panel contains links to HPE SOA Registry Foundation documentation and to the Systinet Web site.



Welcome to the **HPE SOA Registry 10.04** install wizard

Installation requires a minimum of **220 MB** of disk space (**300 MB** is recommended).
The Java VM such as SUN JDK 1.4 and higher or IBM Java 1.4 and higher is required.
See documentation for further details.

To continue, click **Next**.
To stop this installation at any time, click the **Exit** button or close the window.
To see install wizard options, run the wizard with the **--help** option.

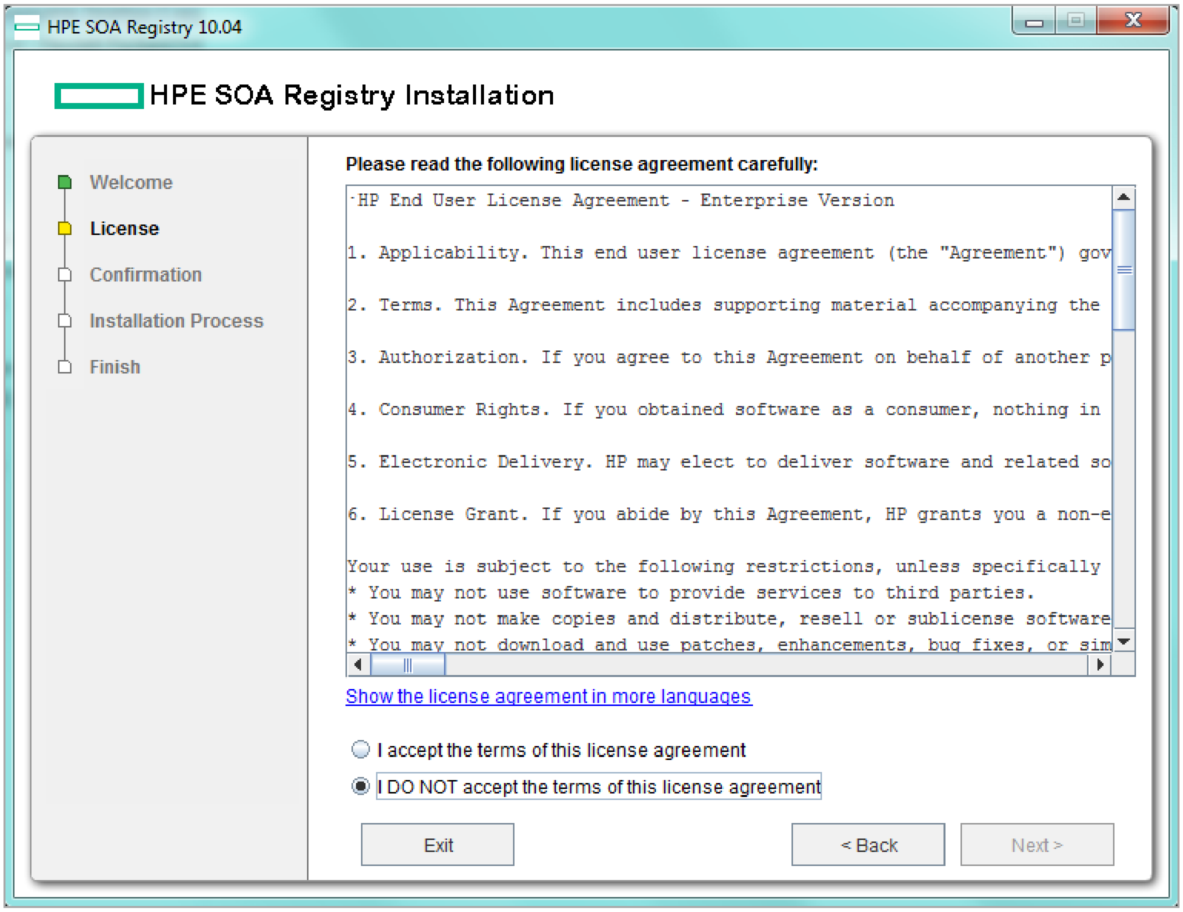
HPE Software
www.hpe.com

License Panel

To continue with the installation of the registry, read the license agreement:

- To accept the license agreement, select **I accept the terms of the license agreement**, and click **Next**.
- If you do not accept the terms of the license agreement, select **I DO NOT accept the terms of the license agreement**, and click **Exit**.

Until you agree to the license, only the **Exit** button is enabled. You cannot proceed with the installation unless you agree to the license.



Please read the following license agreement carefully:

```
·HP End User License Agreement - Enterprise Version

1. Applicability. This end user license agreement (the "Agreement") gov

2. Terms. This Agreement includes supporting material accompanying the

3. Authorization. If you agree to this Agreement on behalf of another p

4. Consumer Rights. If you obtained software as a consumer, nothing in

5. Electronic Delivery. HP may elect to deliver software and related so

6. License Grant. If you abide by this Agreement, HP grants you a non-e

Your use is subject to the following restrictions, unless specifically
* You may not use software to provide services to third parties.
* You may not make copies and distribute, resell or sublicense software
* You may not download and use patches, enhancements, bug fixes, or sim
```

Show the license agreement in more languages

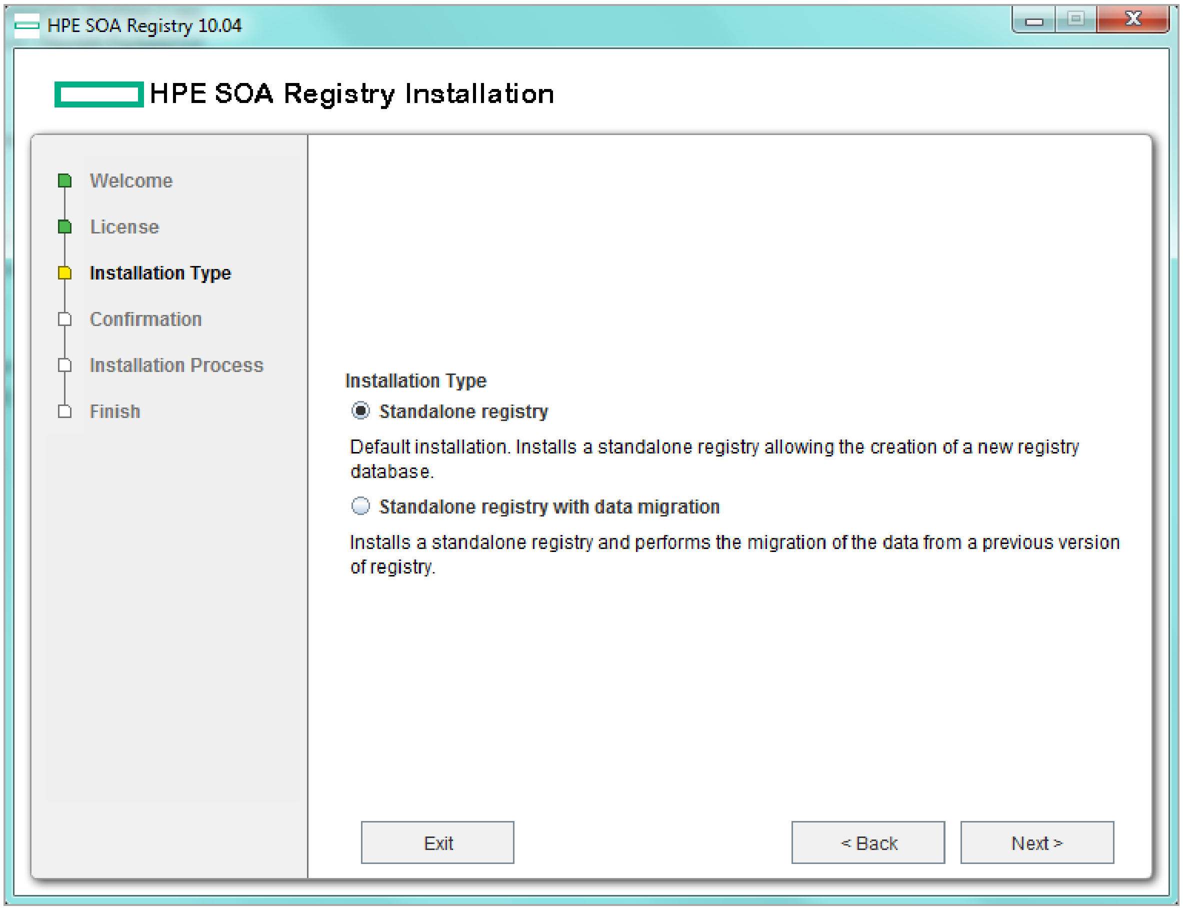
○ I accept the terms of this license agreement

● I DO NOT accept the terms of this license agreement

Installation Type

The Installation Type panel shows two installation scenarios. Select one of the following:

- **Standalone registry**: Default installation. Installs a standalone registry and enables the creation of a new registry database.
- **Standalone registry with data migration**: Installs standalone registry with migration of data from a previous installation of the registry. For more information, see "Migration" on page 177.



Installation Type

● Standalone registry

Default installation. Installs a standalone registry allowing the creation of a new registry database.

○ Standalone registry with data migration

Installs a standalone registry and performs the migration of the data from a previous version of registry.

Installation Directory

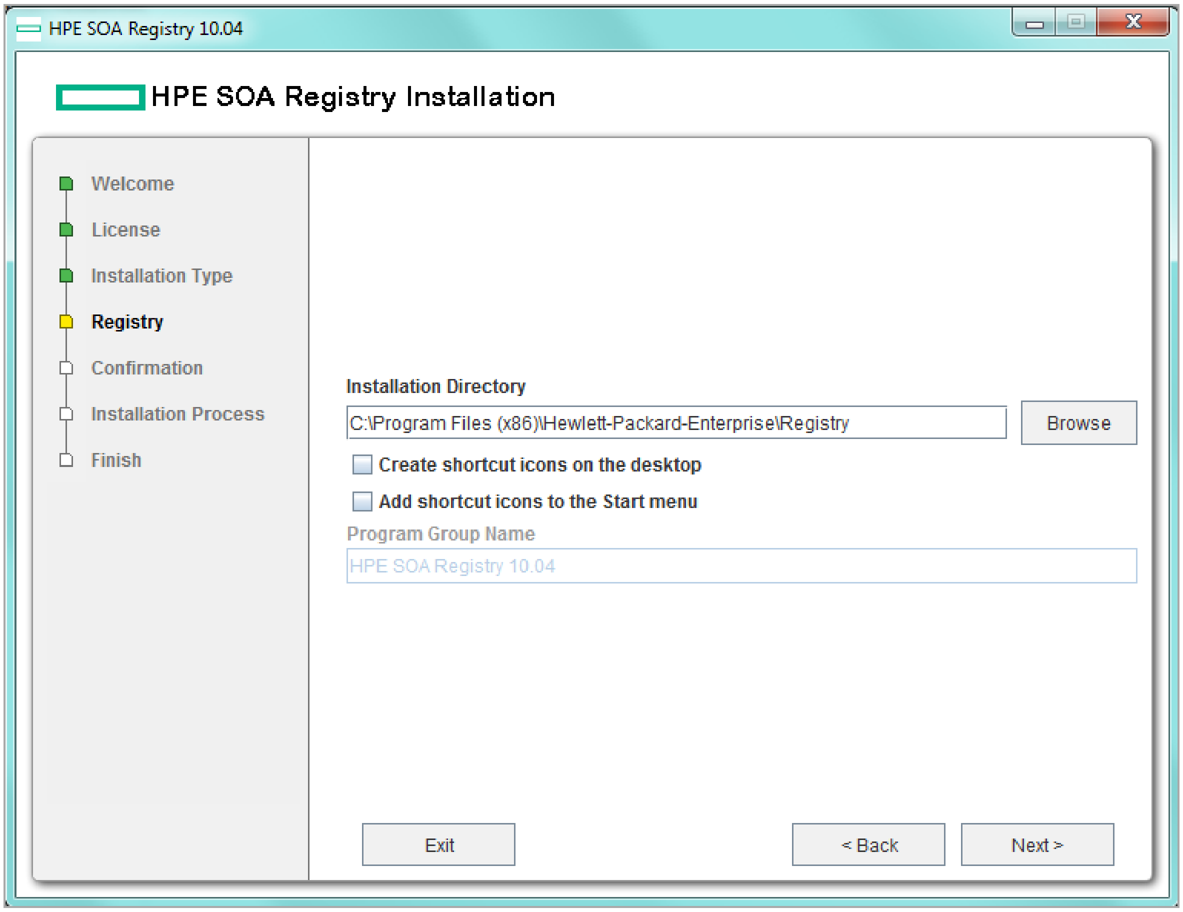
Type the path to the Installation Directory where HPE SOA Registry Foundation will be installed. The default directory is the current working directory.

> **Note:** The Installation directory can consist of ASCII characters. International characters in installation directory path are not supported.

If you install on a Windows platform, you can select from the following:

- **Create shortcut icons on the desktop**: If selected, icons for accessing the Registry Console and for starting and stopping the registry will be created on the desktop.
- **Add shortcut icons to the Start menu**: If selected, the icons noted above are added to the **Start** menu.
- **Program group name**: Group name created in the **Start** menu where shortcut icons will be placed.

> **Note:** You must have read and write permissions on the installation directory.



Installation Directory: C:\Program Files (x86)\Hewlett-Packard-Enterprise\Registry

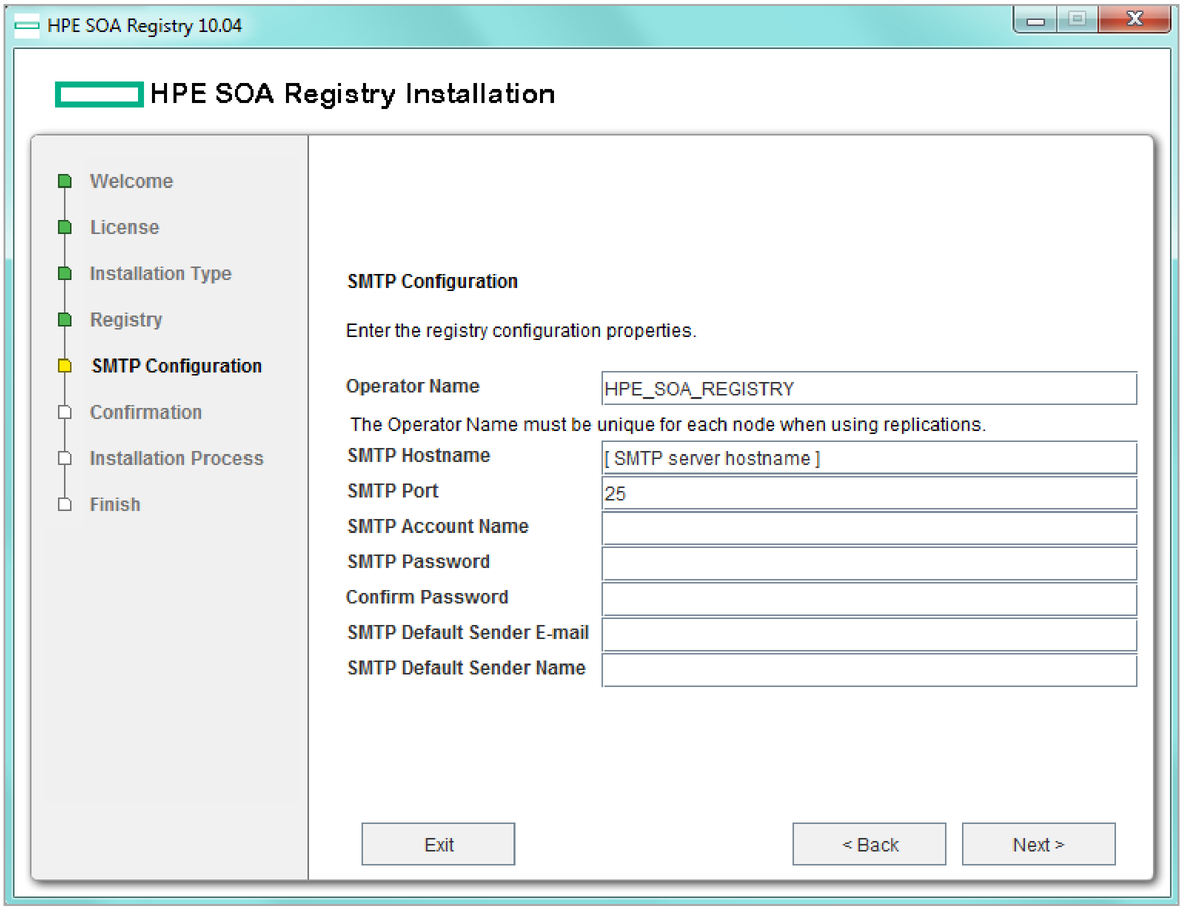
☐ Create shortcut icons on the desktop

☐ Add shortcut icons to the Start menu

Program Group Name: HPE SOA Registry 10.04

SMTP Configuration

The SMTP configuration is important when users need to receive email notifications from subscriptions.



SMTP Configuration

Enter the registry configuration properties.

Operator Name	HPE_SOA_REGISTRY

The Operator Name must be unique for each node when using replications.

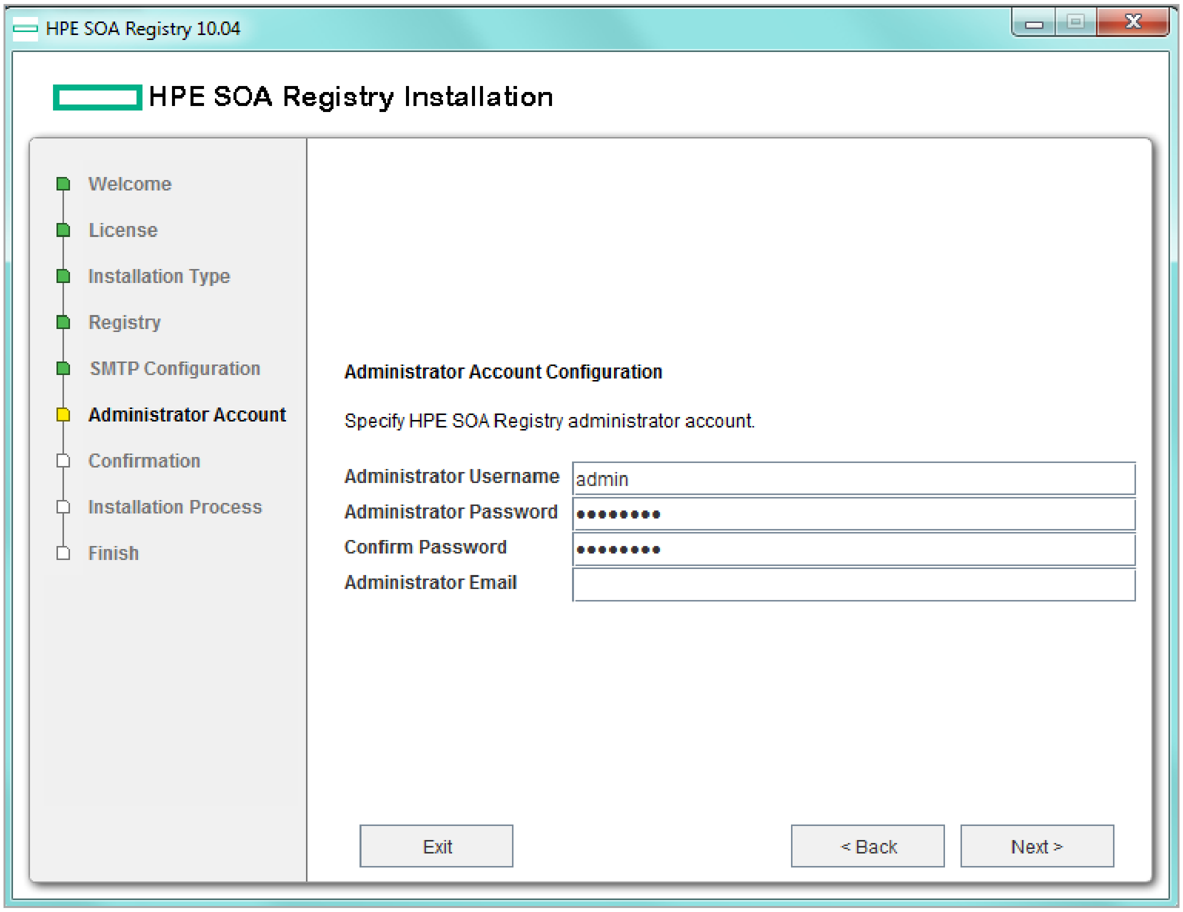
SMTP Hostname	[SMTP server hostname]
SMTP Port	25
SMTP Account Name	
SMTP Password	
Confirm Password	
SMTP Default Sender E-mail	
SMTP Default Sender Name	

Enter the following information:

- **Operator Name:** Name of the operation for example HPE_SOA_Registry.
- **SMTP Host Name**: Host name of the SMTP server associated with this installation of HPE SOA Registry Foundation.
- **SMTP Port**: Port number for this SMTP server.
- **SMTP Password**: Password for SMTP server.
- **Confirm password**: Retype the same password.
- **SMTP Default Sender E-mail, Name**: HPE SOA Registry Foundation will generate email messages with this identity.

Set Up Administrator Account

Enter the HPE SOA Registry Foundation administrator account information.



Administrator Account Configuration

Specify HPE SOA Registry administrator account.

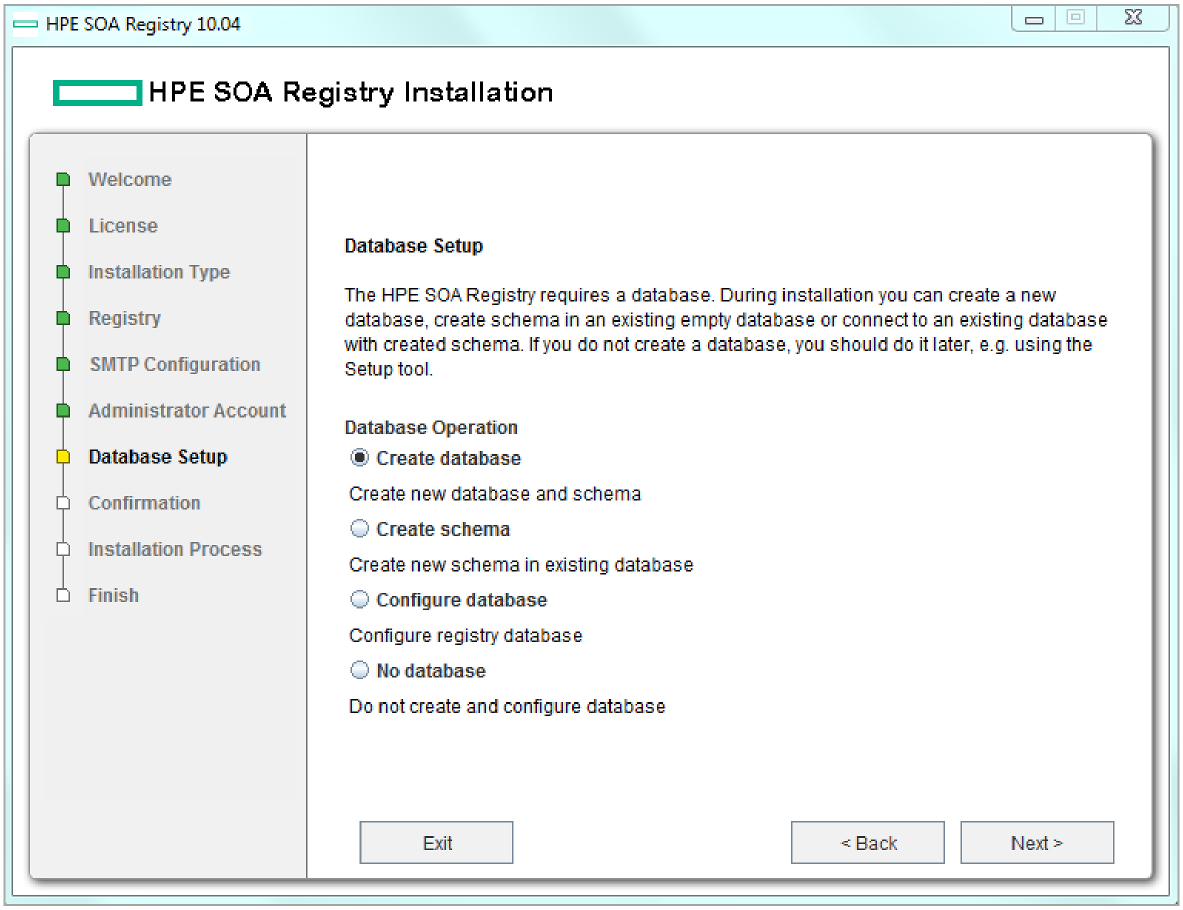
Administrator Username: admin
Administrator Password: ●●●●●●●●
Confirm Password: ●●●●●●●●
Administrator Email:

Database Settings

The registry requires a database, which can be created during installation. During installation you can create a new database, create schema in an existing empty database, or connect to an existing database with created schema. Using the Setup tool, you can also drop the database or database schema.

Database Creation Method

Select your database creation method.



Database Setup

The HPE SOA Registry requires a database. During installation you can create a new database, create schema in an existing empty database or connect to an existing database with created schema. If you do not create a database, you should do it later, e.g. using the Setup tool.

Database Operation

● Create database

Create new database and schema

○ Create schema

Create new schema in existing database

○ Configure database

Configure registry database

○ No database

Do not create and configure database

- **Create database**: Create new database/users/tablespaces (depending on the type of the database server) and database schema. This is the most common method, but please note that you must know the credentials of the database administrator.
- **Create schema**: Create a new schema in an existing database. Select this option if you have access to an existing empty database and the ability to create tables and indexes. This option is suitable when you do not know the administrator's credentials. It is assumed that the admin has already created a new database/users/tablespaces for this option. See "Database Installation" on page 80.

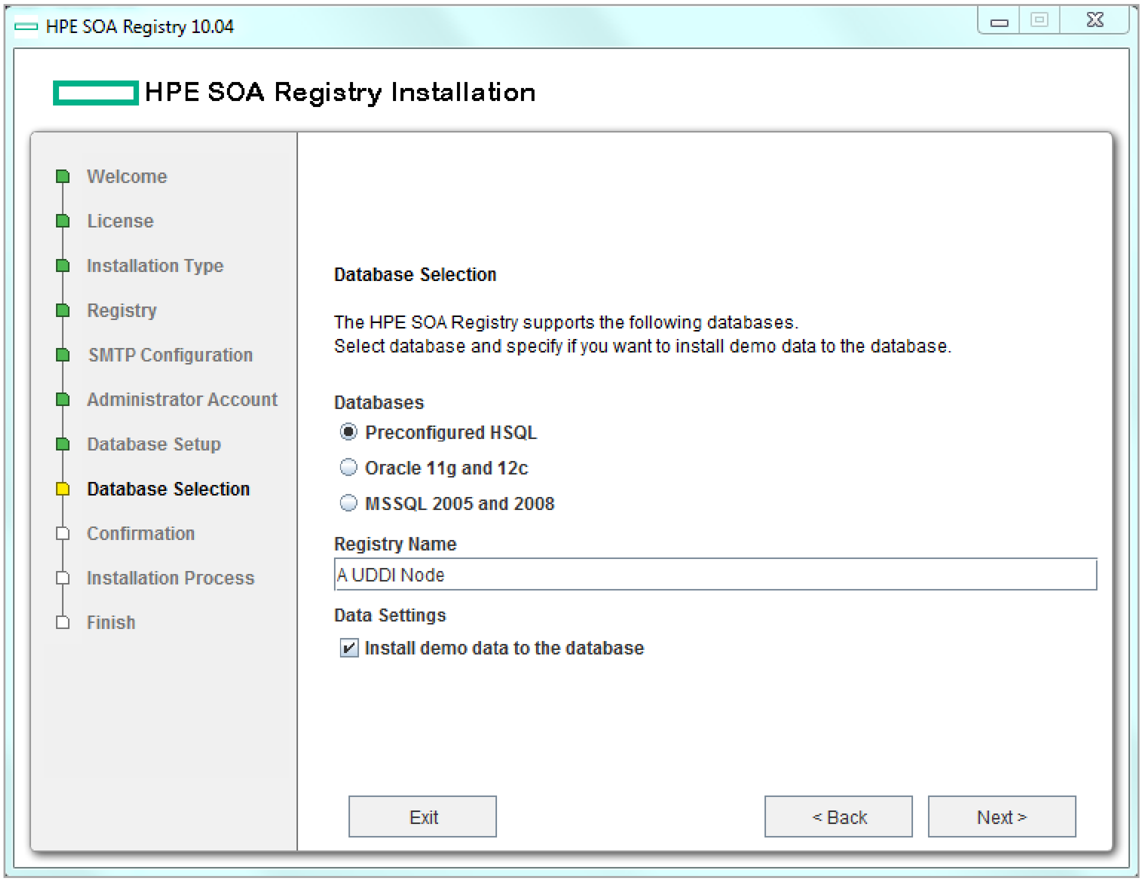
  > **Note:** The cannot be started without a database.

- **Configure database**: Configure registry database. Select this option if the registry database already exists (For example, from a previous installation) and fill in only the connection parameters.

- **No database**: Select this option if you intend to create a registry database later.

Select Database

The Select Database panel shows the supported database engines that can be prepared for HPE SOA Registry Foundation.



Database Selection

The HPE SOA Registry supports the following databases.
Select database and specify if you want to install demo data to the database.

Databases

● Preconfigured HSQL

○ Oracle 11g and 12c

○ MSSQL 2005 and 2008

Registry Name: A UDDI Node

Data Settings

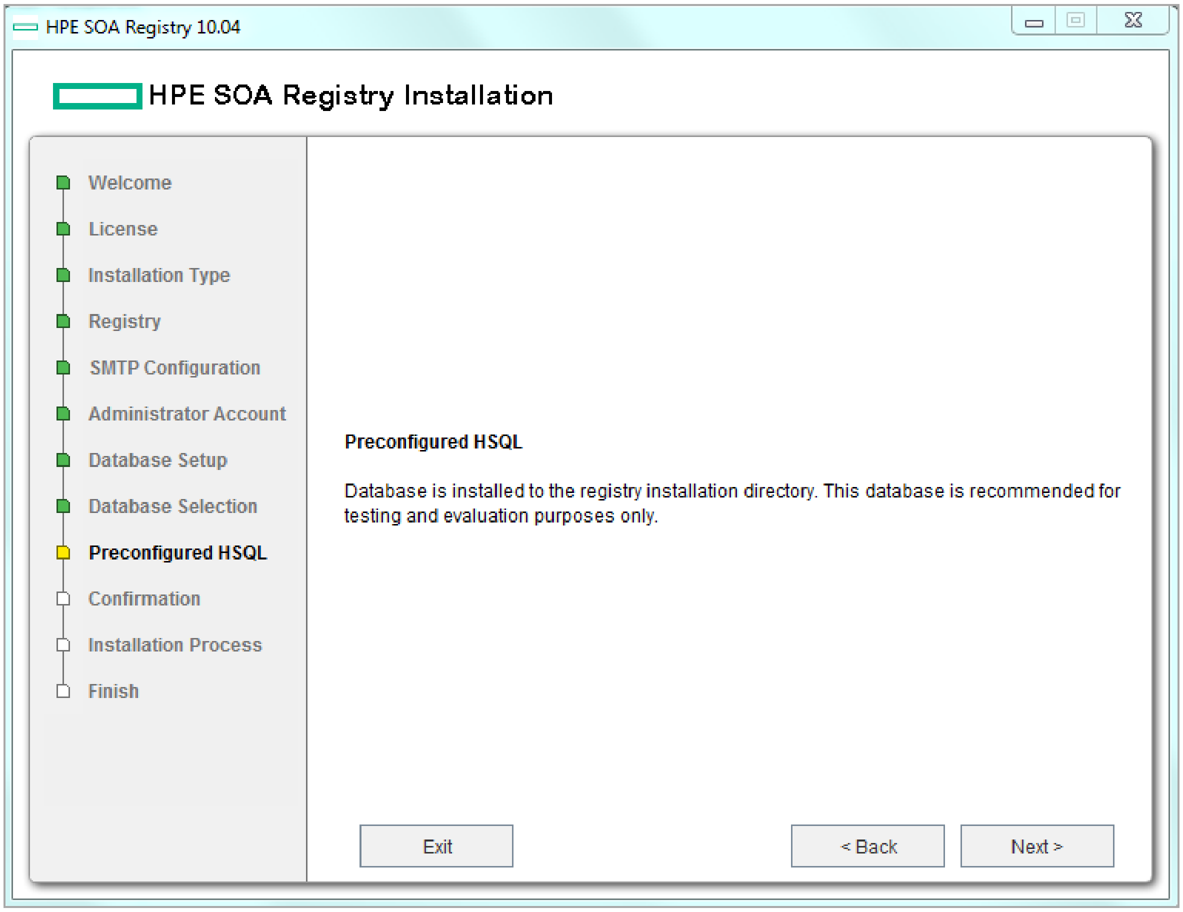
☑ Install demo data to the database

You can specify the name of HPE SOA Registry Foundation installation. The name is saved to the operational business entity. The registry name appears in the upper right corner of Registry Console.

Select **Install demo data** if you want to evaluate the provided HPE SOA Registry Foundation demos after installation.

The default database to create is **Preconfigured HSQL (HSQL)**. This database is recommended for evaluation purposes.

Preconfigured HSQL Panel



Preconfigured HSQL

Database is installed to the registry installation directory. This database is recommended for testing and evaluation purposes only.

The database files will be installed into the `REGISTRY_HOME/hsqldb/uddinode` directory. The database user is `uddiuser` and the password is `uddi`.

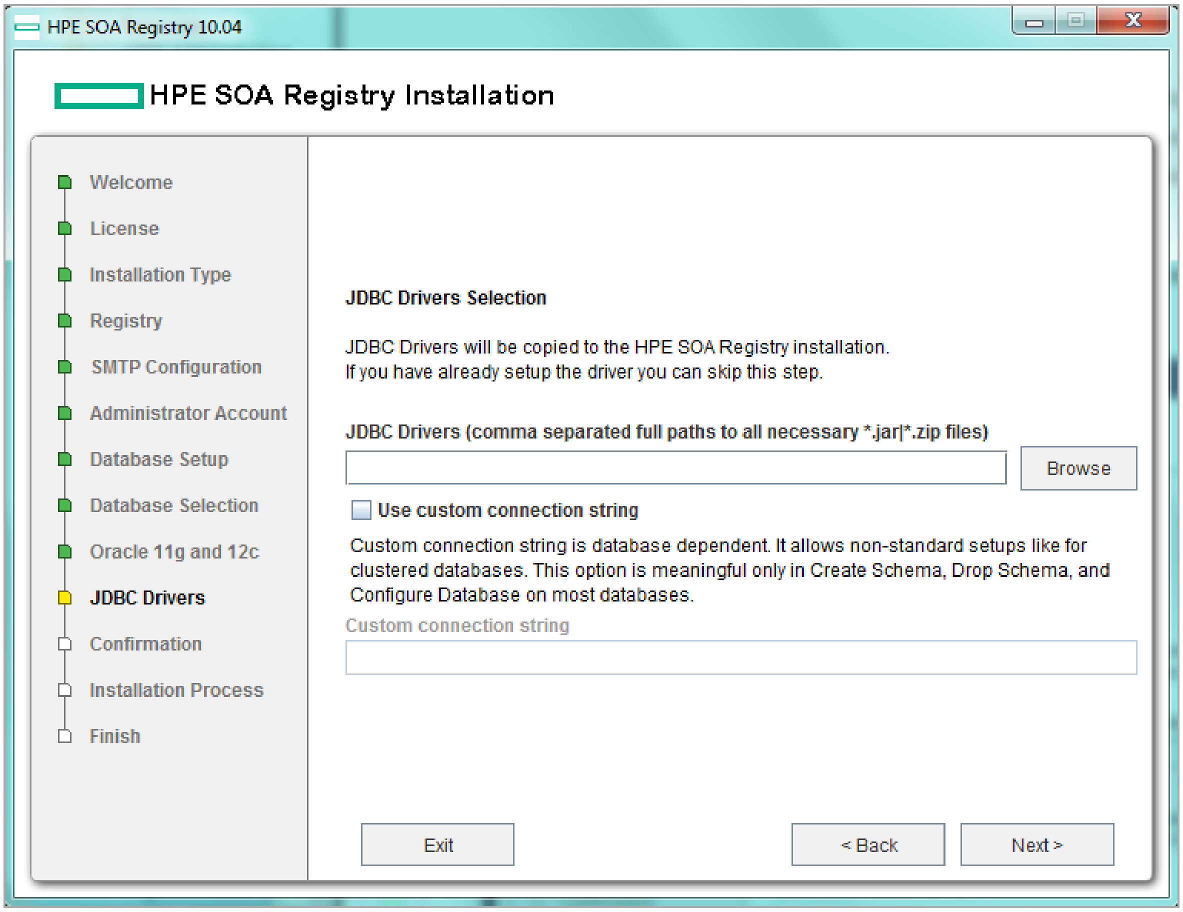
> **Note:** You can use the Setup tool to change the database after installation.

For more information on database installation, see "Database Installation" on page 80.

Optional JDBC Driver

You can specify a custom JDBC connection string. A JDBC string can be useful for special environments, such as for database clusters where a JDBC driver does load-balancing or failover. This setting is useful only in Create Schema, Drop Schema and Configure Database.

> **Note:** HPE recommends that you do not select this option unless you have a clear need to do so.



JDBC Drivers Selection

JDBC Drivers will be copied to the HPE SOA Registry installation.
If you have already setup the driver you can skip this step.

JDBC Drivers (comma separated full paths to all necessary *.jar|*.zip files)

□ **Use custom connection string**

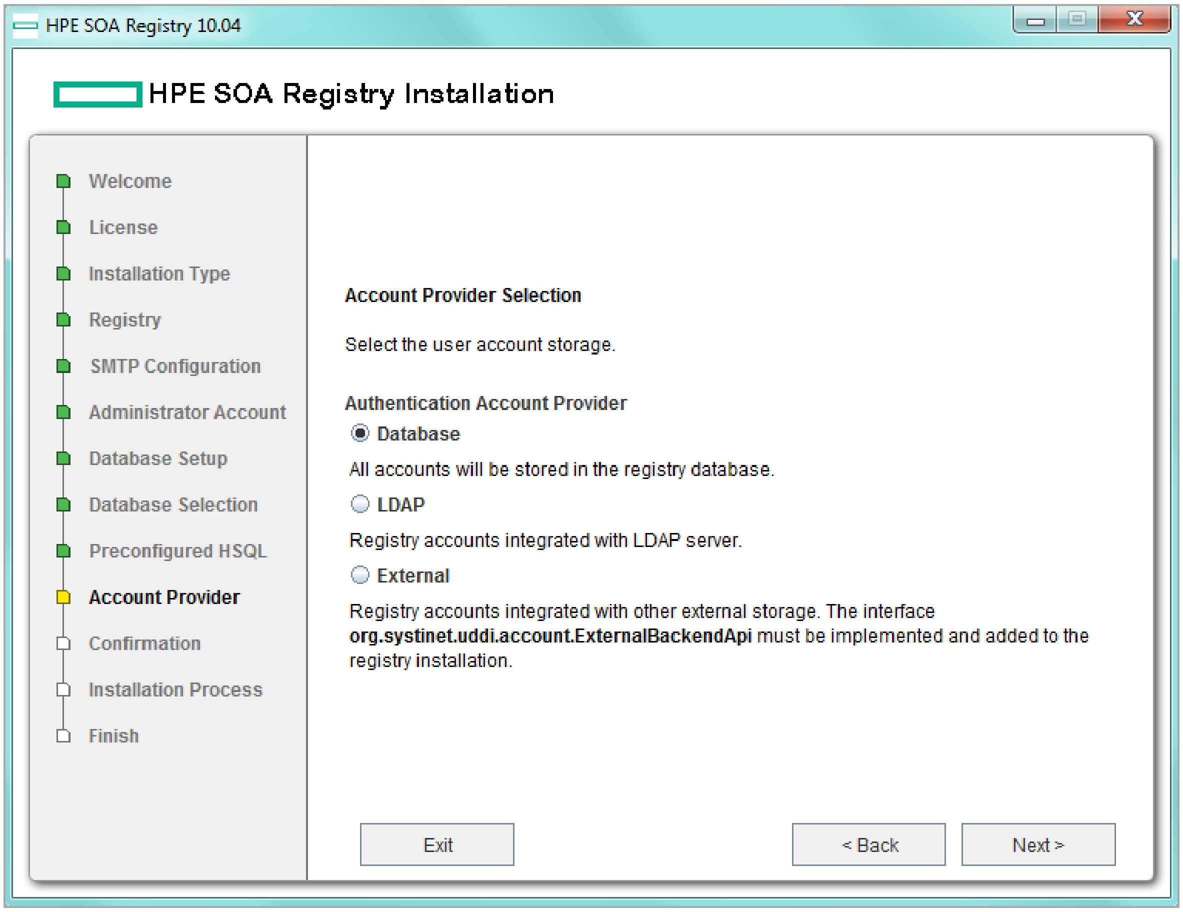
Custom connection string is database dependent. It allows non-standard setups like for clustered databases. This option is meaningful only in Create Schema, Drop Schema, and Configure Database on most databases.

Custom connection string

Enter the path to the JDBC drivers on the **JDBC Drivers** panel. You do not need to configure this path for the HSQL and PostgreSQL databases, because the JDBC drivers for those databases are installed in the distribution.

Authentication Account Provider

You can select an authentication account provider:



Account Provider Selection

Select the user account storage.

Authentication Account Provider

● **Database**

All accounts will be stored in the registry database.

○ **LDAP**

Registry accounts integrated with LDAP server.

○ **External**

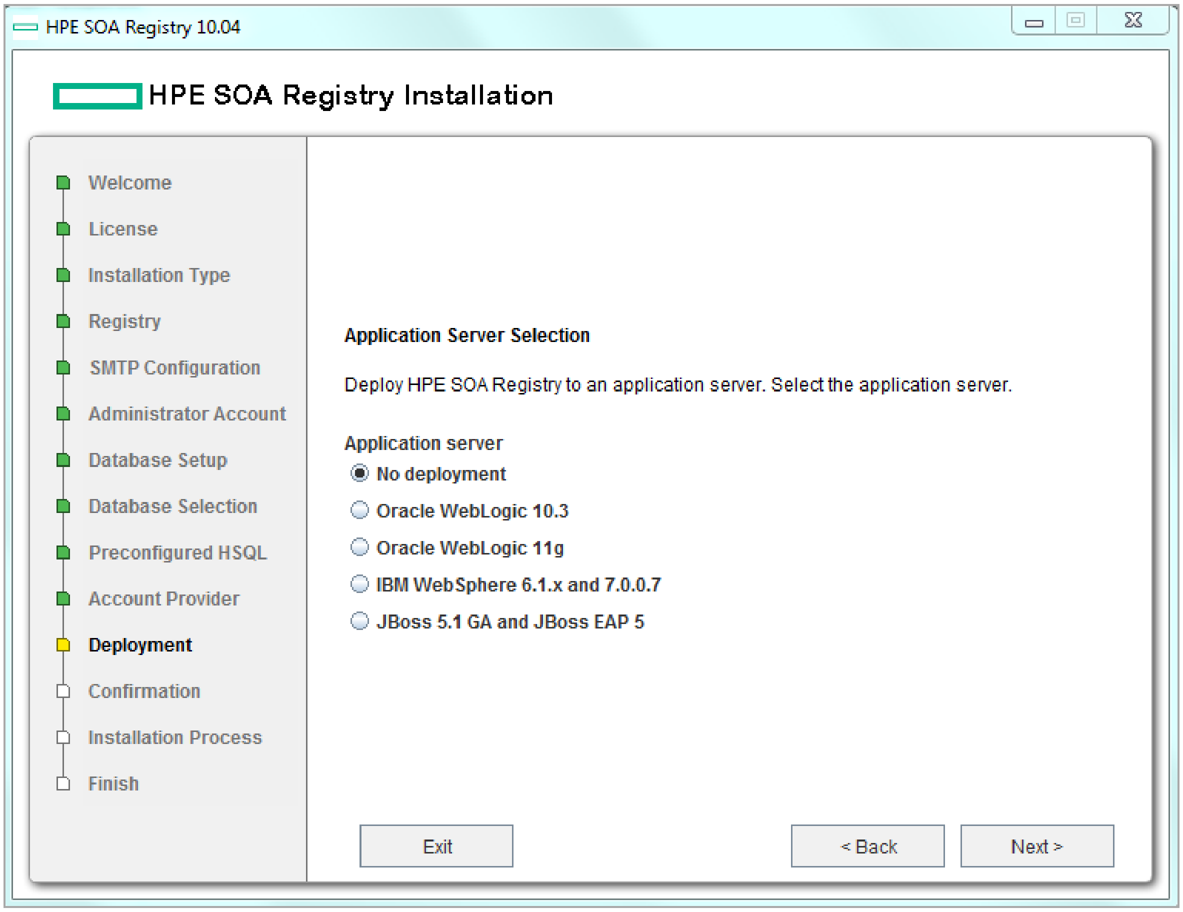
Registry accounts integrated with other external storage. The interface **org.systinet.uddi.account.ExternalBackendApi** must be implemented and added to the registry installation.

Choose one of the following:

- **Database**: All accounts will be stored in the registry database.
- **LDAP**: Registry accounts integrated with LDAP server.
- **External**: Registry accounts integrated with other external storage. The interface `com.systinet.uddi.account.ExternalBackendApi` must be implemented and added to the registry installation.

Direct Deployment

You can use direct deployment to create EAR or WAR files for deployment in an application server directly from the installer. You can also deploy later with Setup (see "Creating a Web Application Archive (WAR,EAR)" on page 120). Deployment from the installer is similar.



Application Server Selection

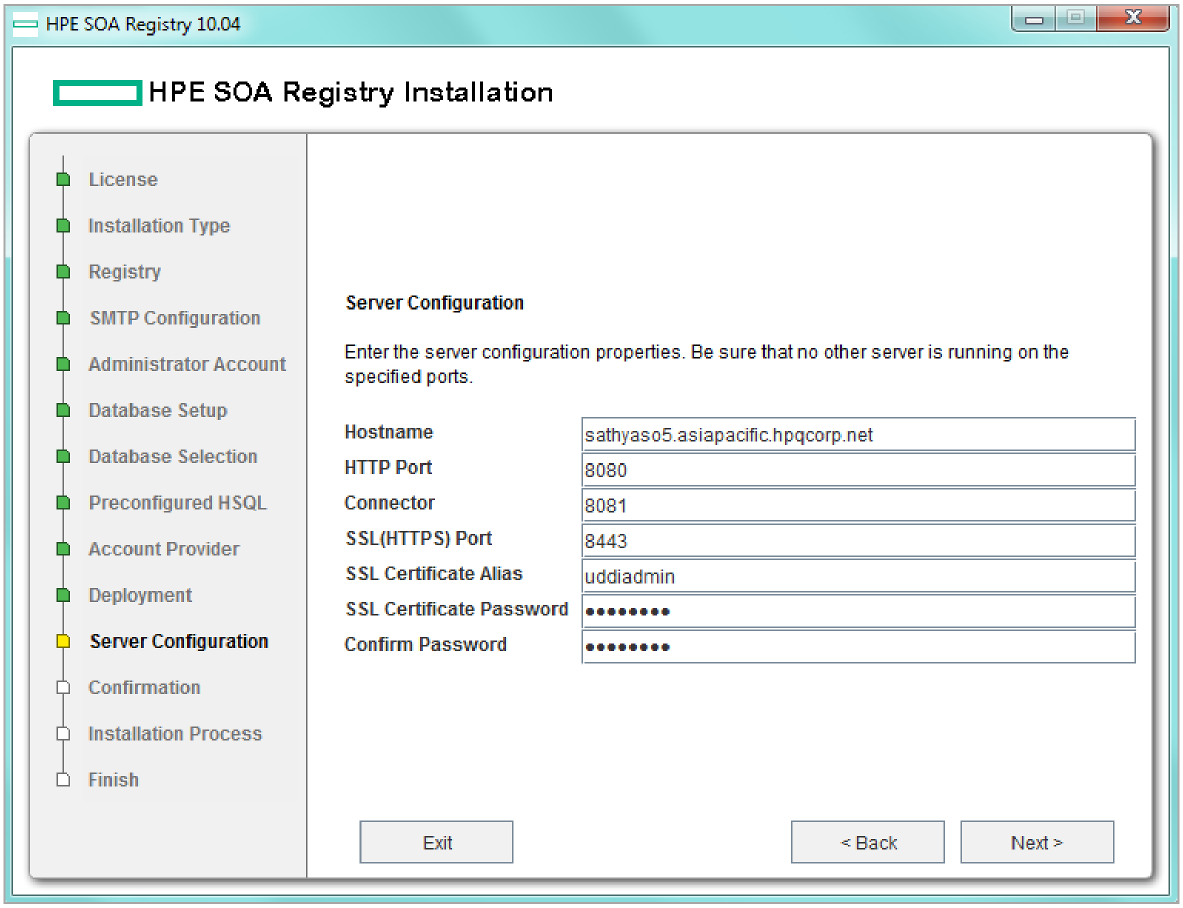
Deploy HPE SOA Registry to an application server. Select the application server.

Application server

● No deployment
○ Oracle WebLogic 10.3
○ Oracle WebLogic 11g
○ IBM WebSphere 6.1.x and 7.0.0.7
○ JBoss 5.1 GA and JBoss EAP 5

Server Configuration Settings

The server settings will be used for the HTTP and HTTPS servers. The default recommended settings are filled in the text fields.



Server Configuration

Enter the server configuration properties. Be sure that no other server is running on the specified ports.

Hostname	sathyaso5.asiapacific.hpqcorp.net
HTTP Port	8080
Connector	8081
SSL(HTTPS) Port	8443
SSL Certificate Alias	uddiadmin
SSL Certificate Password	••••••••
Confirm Password	••••••••

Host name — The host name of this computer; change the auto-completed entry if it is different.

HTTP Port — The nonsecure port for accessing the Registry Console (default value: 8080)

SSL (HTTPS) Port — Secure port for accessing the Registry Console (default value: 8443)

Connector — The connector port is used by standalone server to listen for control signals. Note that no other application may use this port (default value: 8081).

SSL Certificate Alias — Alias used to identify the SSL private key in protected store management. For more information see "PStore Tool" on page 361. (default value: uddiadmin)

SSL Certificate Password — Password to encrypt SSL private key. (default value: changeit)

Confirm Password — Retype the same password. Note that if it is not same as previous, you cannot continue.

The host name, SSL Certificate Alias, and SSL password are used to create a new security identity in the local protected store. It creates a certificate and adds this certificate to REGISTRY_HOME/conf/clientconf.xml, REGISTRY_HOME/conf/pstore.xml, and also exports it to the certificate file REGISTRY_HOME/doc/registry.crt. See "PStore Tool" on page 361 for instructions on how to operate the protected security store.

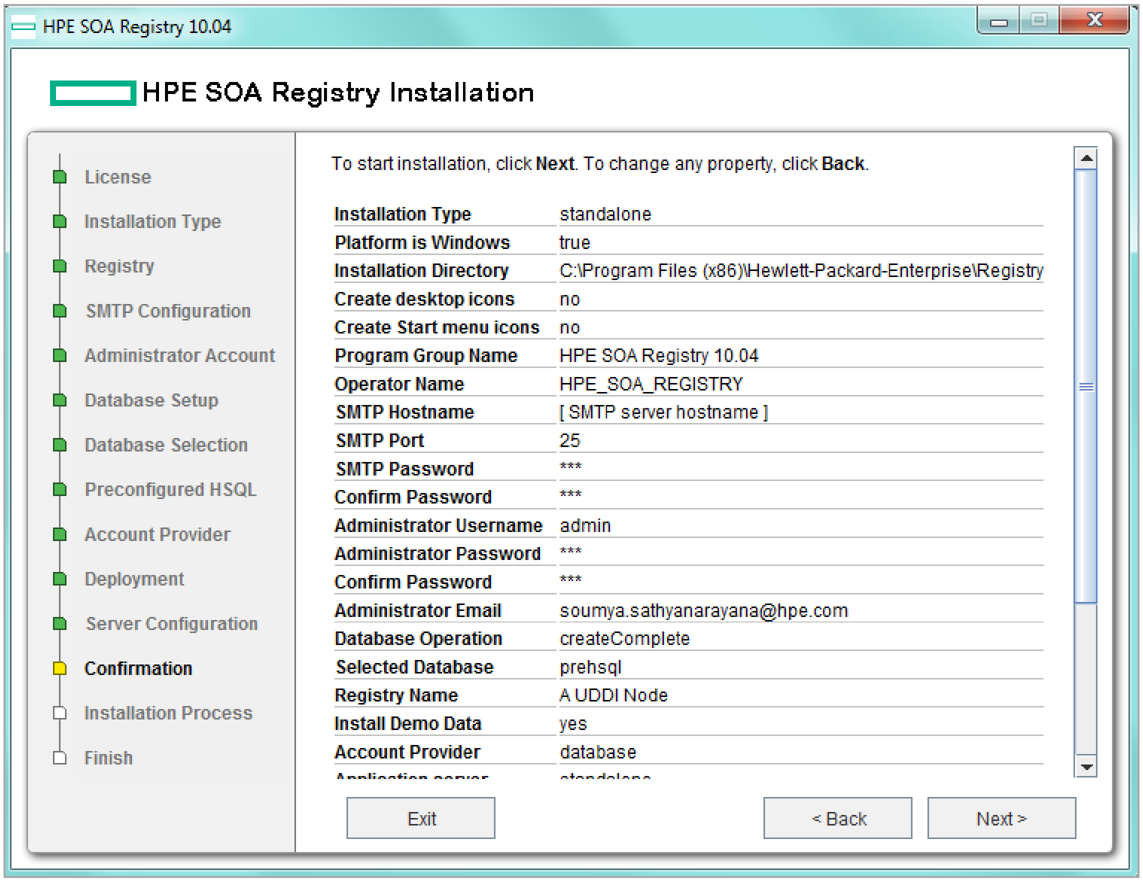
> **Note:** The server configuration may be changed after install.

Confirmation and Installation Process

Confirmation Panel

The Confirmation panel shows a summary of installation information. All required and optional properties are set:

- If you want to continue with the installation, click **Next** and the install process will start.
- If you want to change any property, click **Back**.



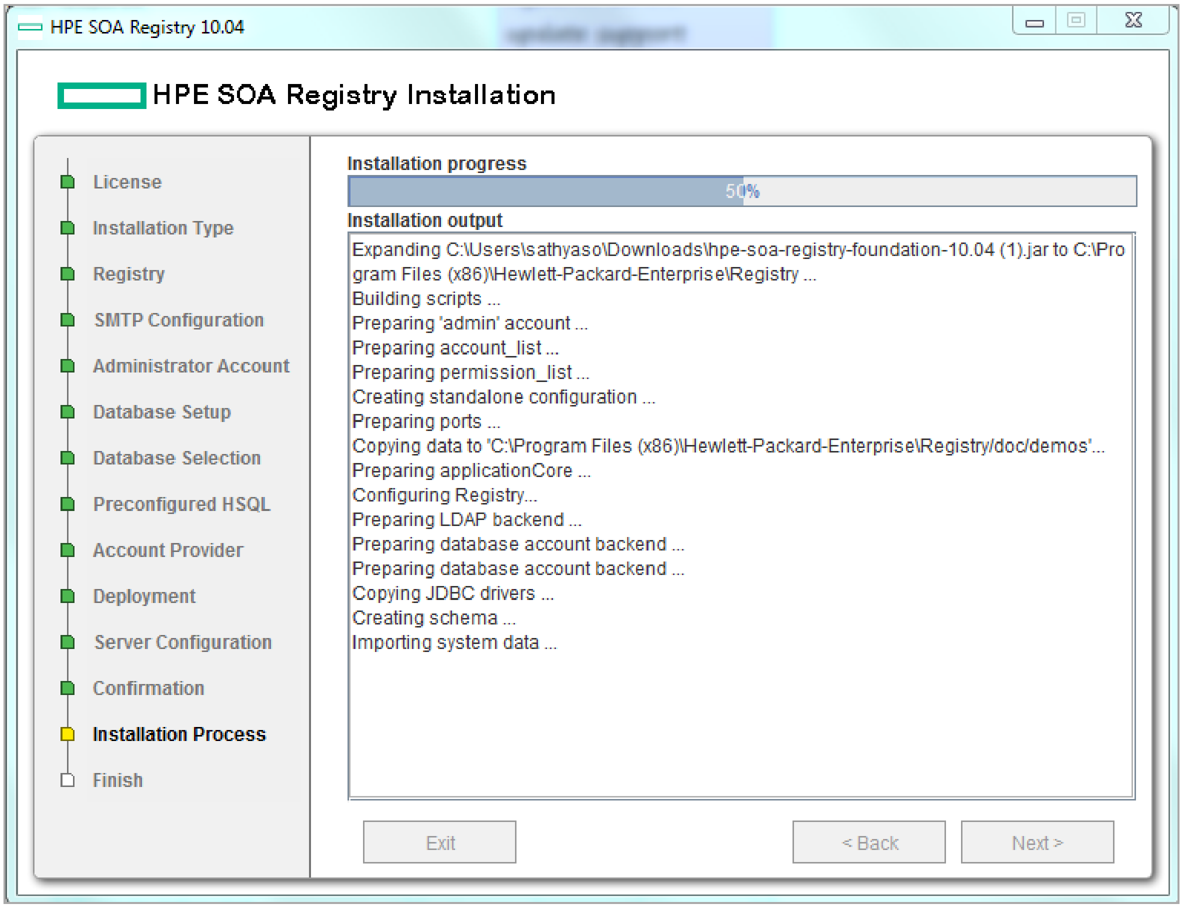
To start installation, click **Next**. To change any property, click **Back**.

Installation Type	standalone
Platform is Windows	true
Installation Directory	C:\Program Files (x86)\Hewlett-Packard-Enterprise\Registry
Create desktop icons	no
Create Start menu icons	no
Program Group Name	HPE SOA Registry 10.04
Operator Name	HPE_SOA_REGISTRY
SMTP Hostname	[SMTP server hostname]
SMTP Port	25
SMTP Password	***
Confirm Password	***
Administrator Username	admin
Administrator Password	***
Confirm Password	***
Administrator Email	soumya.sathyanarayana@hpe.com
Database Operation	createComplete
Selected Database	prehsql
Registry Name	A UDDI Node
Install Demo Data	yes
Account Provider	database
Application server	standalone

Installation Process Panel

The Installation Process panel shows the installation output and progress. Installation consists of copying files, configuring the server and installing the database. When the installation has completed successfully, the **Next** button is enabled. If there is a problem, an error message and a **Recovery** button will appear on the screen.

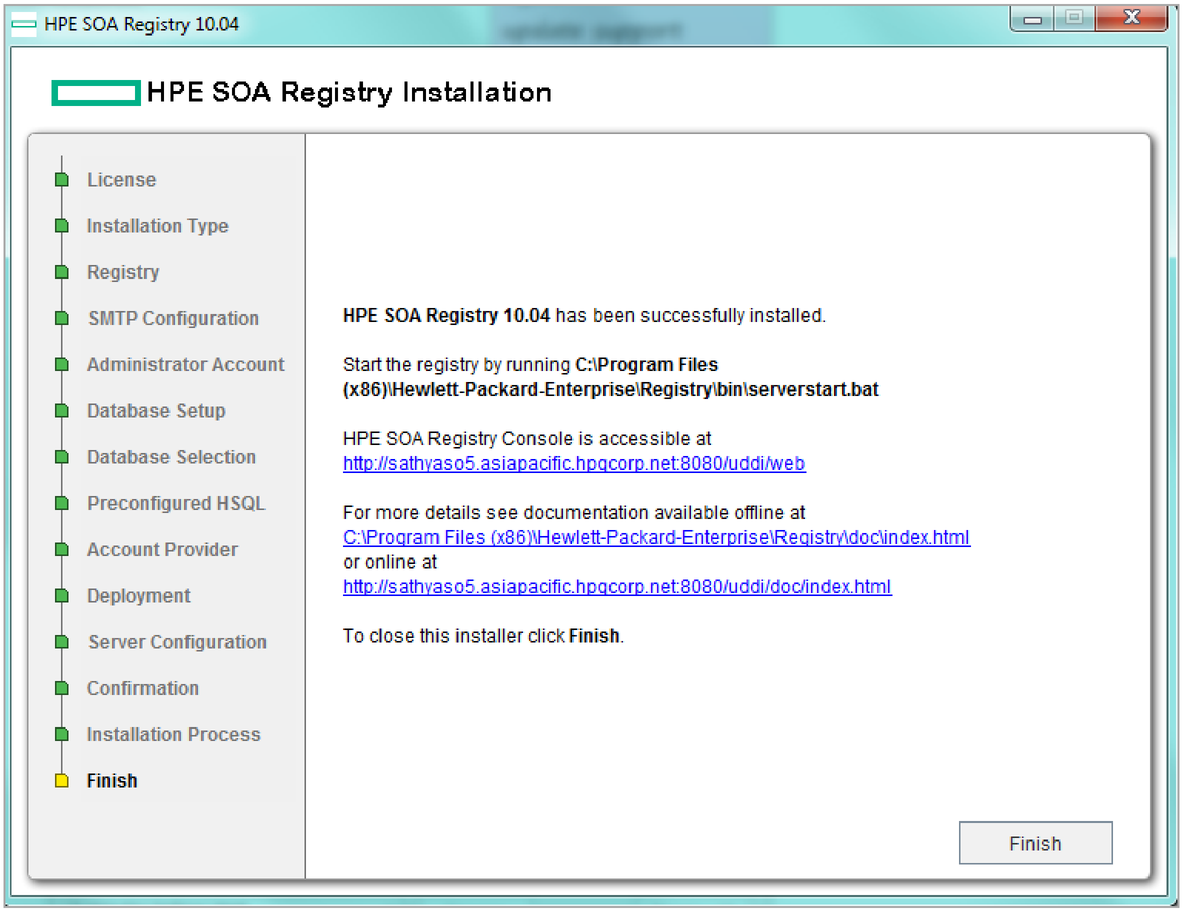
For more information on recovery, see "Troubleshooting" on page 66.



```
Expanding C:\Users\sathyaso\Downloads\hpe-soa-registry-foundation-10.04 (1).jar to C:\Pro
gram Files (x86)\Hewlett-Packard-Enterprise\Registry ...
Building scripts ...
Preparing 'admin' account ...
Preparing account_list ...
Preparing permission_list ...
Creating standalone configuration ...
Preparing ports ...
Copying data to 'C:\Program Files (x86)\Hewlett-Packard-Enterprise\Registry/doc/demos'...
Preparing applicationCore ...
Configuring Registry...
Preparing LDAP backend ...
Preparing database account backend ...
Preparing database account backend ...
Copying JDBC drivers ...
Creating schema ...
Importing system data ...
```

Finish

To complete the installation, click **Finish**.



HPE SOA Registry 10.04 has been successfully installed.

Start the registry by running **C:\Program Files (x86)\Hewlett-Packard-Enterprise\Registry\bin\serverstart.bat**

HPE SOA Registry Console is accessible at http://sathyaso5.asiapacific.hpqcorp.net:8080/uddi/web

For more details see documentation available offline at C:\Program Files (x86)\Hewlett-Packard-Enterprise\Registry\doc\index.html or online at http://sathyaso5.asiapacific.hpqcorp.net:8080/uddi/doc/index.html

To close this installer click **Finish**.

Installation Summary

The following tables contain summary information.

- "Directory Structure" below
- "Registry Endpoints" on the next page
- "Pre-Installed Data" on page 59

Directory Structure

The installation directory structure contains the following directories.

app	HPE SOA Registry Foundation deployed as Web services in SOA Registry Foundation Server for Java.

`bin`	Contains command-line scripts for running HPE SOA Registry Foundation. See "Command Line Scripts" on page 59.
`conf`	HPE SOA Registry Foundation configuration files
`demos`	demos of HPE SOA Registry Foundation functionality. For more information, see "Demos" on page 531.
`dist`	HPE SOA Registry Foundation client packages.
`doc`	HPE SOA Registry Foundation documentation.
`etc`	additional data and scripts.
`hsqldb`	preconfigured HSQL database with registry data.
`lib`	HPE SOA Registry Foundation libraries
`log`	logs of installation, setup, and server output. See "Logs" on page 65.
`work`	This directory is available after the first launch of the server; it is a working image of the app directory.

Registry Endpoints

HPE Systinet is configured as follows. The <`host name`>, <`http port`> and <`ssl port`> are specified during installation. For more information, see "Server Settings" on page 1. For each endpoint you can use either an http or ssl port.

- **Registry Console home page**: `http://<host name>:<http port>/uddi/web`
- **UDDI Inquiry API endpoint** - `http://<host name>:<port>/uddi/inquiry`

See UDDI Version 1, UDDI Version 2, and UDDI Version 3 in "UDDI APIs" on page 378 in the Developer's Guide.

- **UDDI Publishing API endpoint** - `http://<host name>:<port>/uddi/publishing`

See UDDI Version 1, UDDI Version 2, and UDDI Version 3 in "UDDI APIs" on page 378 in the Developer's Guide.

- **UDDI Security Policy v3 API endpoint**:- `http://<host name>:<port>/uddi/security`

See UDDI Version 3 in "UDDI APIs" on page 378 in the Developer's Guide.

- **UDDI Custody API endpoint** - `http://<host name>:<port>/uddi/custody`

See UDDI Version 3 in "UDDI APIs" on page 378in the Developer's Guide.

- **UDDI Subscription API endpoint** - `http://<host name>:<port>/uddi/subscription`

See UDDI Version 3 in "UDDI APIs" on page 378 in the Developer's Guide.

- **Taxonomy API endpoint**:- `http://<host name>:<port>/uddi/taxonomy`

See "Taxonomy" on page 395 in the Developer's Guide.

- **Category API endpoint** - `http://<host name>:<port>/uddi/category`

See "Category" on page 406 in the Developer's Guide.

• Administration Utilities API endpoint - `http://<host name>:<port>/uddi/administrationUtils`

See "Administration Utilities" on page 412 in the Developer's Guide.

• Replication API endpoint - `http://<host name>:<port>/uddi/replication`

See "Replication" on page 417 in the Developer's Guide.

• Statistics API endpoint - `http://<host name>:<port>/uddi/statistics`

See "Statistics" on page 418 in the Developer's Guide.

• WSDL2UDDI API endpoint - `http://<host name>:<port>/uddi/wsdl2uddi`

See Developer's Guide, WSDL Publishing on page 521.

• XSD2UDDI API endpoint - `http://<host name>:<port>/uddi/xsd2uddi`

See "XSD Publishing" on page 435 in the Developer's Guide.

• Extended Inquiry API endpoint - `http://<host name>:<port>/uddi/inquiryExt`

• Extended Publishing API endpoint - `http://<host name>:<port>/uddi/publishingExt`

• Configurator API endpoint - `http://<host name>:<port>/uddi/configurator`

• Account API endpoint - `http://<host name>:<port>/uddi/account`

See "Accounts" on page 453 in the Developer's Guide.

Group API endpoint - `http://<host name>:<port>/uddi/group`

See "Group" on page 460 in the Developer's Guide.

• Permission API endpoint - `http://<host name>:<port>/uddi/permission`

See "Permission" on page 467 in the Developer's Guide.

Pre-Installed Data

HPE SOA Registry Foundation contains the following data:

- Operational business - This entity holds miscellaneous nodes' registry settings such as the validation service configuration.
- Built in tModels - tModels required by the UDDI specification.
- Demo data - Data required by the HPE SOA Registry Foundation demos. For more information, see "Demos" on page 531.

Command Line Scripts

The bin subdirectory contains scripts, including those for launching the server, installing Windows services, and changing configuration.

Serverstart

Windows:	serverstart.bat
UNIX:	./serverstart.sh

Starts the standalone registry server.

serverstop

Windows:	serverstop.bat
UNIX:	./serverstop.sh

Stops the standalone registry server.

server

Windows:	server.bat
UNIX:	./server.sh

Helper script to manipulate the standalone HPE SOA Registry Foundation server. To start and stop the registry, use serverstart or serverstop without parameters instead of server with parameters. For more information, see "Server Properties" on page 63.

Setup

Windows:	setup.bat
UNIX:	./setup.sh

Setup may be launched with the following optional arguments:

`setup.sh (.bat)` [[--help] | [-h] | [--gui] | [-g] | [-u `file`] | [--use-config `file`]] [[-s `file`] | [--save-config `file`]] [--debug]

`-h | --help` shows help message

`-g | --gui` starts the setup wizard. The wizard is the default mode.

`-u | --use-config` file starts setup in non-interactive mode; it reads all properties from the specified file.

`-s | --save-config` file starts the setup wizard. All configuration will be saved into specified file instead of execute configuration. The file may be used later in a non-interactive installation.

`--debug` the setup produces more information to localize problems or errors.

To change the HPE SOA Registry Foundation configuration after installation see "Reconfiguring After Installation" on page 62.

Signer

Windows:	signer.bat
UNIX:	./signer.sh

The Signer is a graphical application that can be used to add, remove, and verify the signatures of UDDI structures you have published. See "Signer Tool" on page 291.

register

Windows:	register.bat
UNIX:	./register.sh

Registers evaluation version of HPE SOA Registry Foundation.

SoapSpy

Windows:	SoapSpy.bat
UNIX:	./SoapSpy.sh

Debugging tool to control low level soap communication. See "How to Debug" on page 526.

PStoreTool

Windows:	PStoreTool.bat
UNIX:	./PStoreTool.sh

Protected security storage manipulation tool. See "PStore Tool" on page 361.

env

Windows:	env.bat
UNIX:	./env.sh

Helper script to set system variables. HPE recommends that you not use it directly.

Reconfiguring After Installation

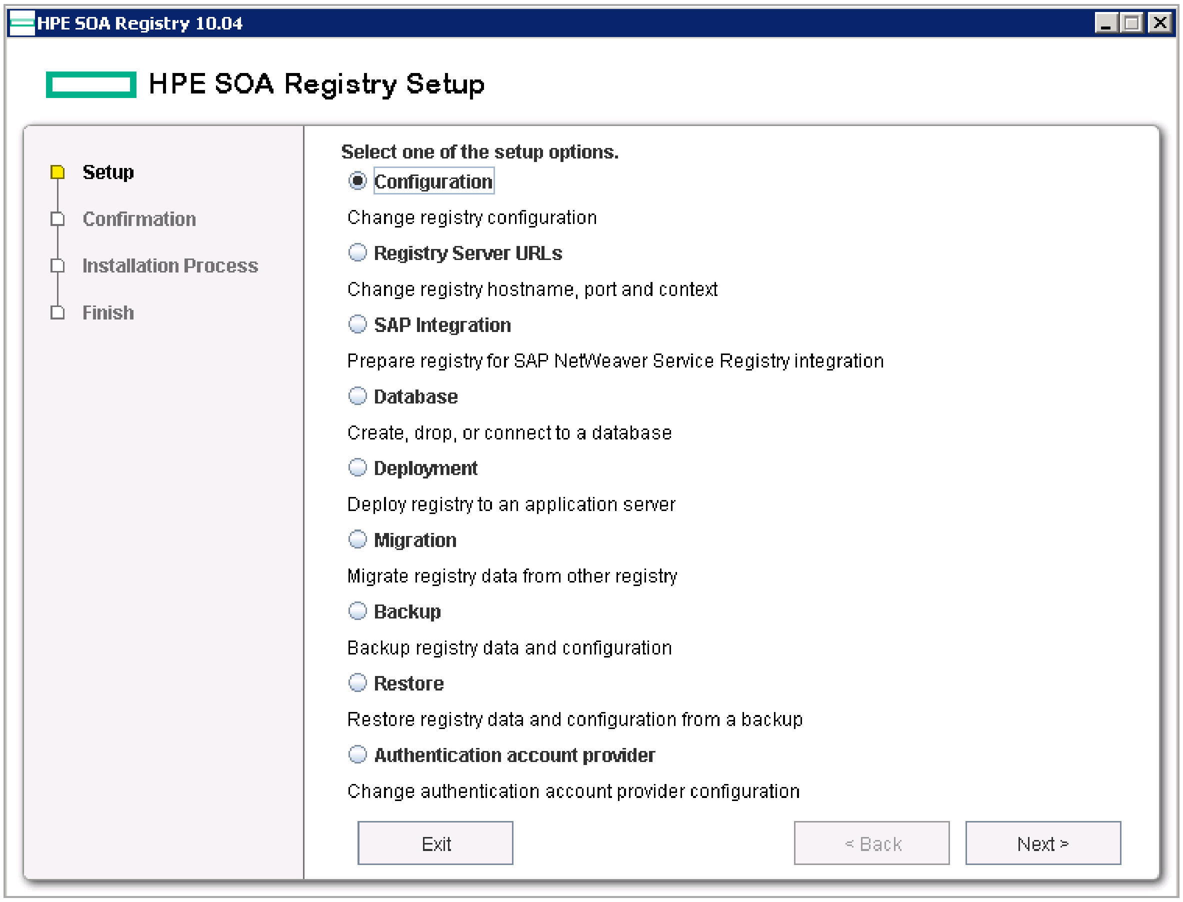
All settings may be changed after installation using the Setup tool.

The Setup tool also facilitates other functions such as deploying to an application server (described in “Deployment to an Application Server”) and data migration from previous installation (described in “Migration”).

The Setup tool contains similar panels to those in the installation tool. To run this tool, execute the following script from the bin subdirectory of your installation:

Windows:	setup.bat
UNIX:	./setup.sh

By default setup starts in wizard mode as shown here:



Select one of the setup options.

● Configuration
Change registry configuration

○ Registry Server URLs
Change registry hostname, port and context

○ SAP Integration
Prepare registry for SAP NetWeaver Service Registry integration

○ Database
Create, drop, or connect to a database

○ Deployment
Deploy registry to an application server

○ Migration
Migrate registry data from other registry

○ Backup
Backup registry data and configuration

○ Restore
Restore registry data and configuration from a backup

○ Authentication account provider
Change authentication account provider configuration

The following topics may be configured:

Configuration: Change server and registry configuration. Follow Server Configuration on page 103.

Registry Server URLs: Change Registry server URLs in database. Follow Change Server URLs on page 107.

SAP Integration: Prepare registry for SAP NetWeaver Service Registry integration. Follow SAP Integration on page 113.

Database: Create, drop, or connect to a database. Follow Database Installation on page 117.

Deployment: Deploy registry to an application server. Follow Deployment to an Application Server on page 175.

Migration: Migrate registry data from other registry. Follow Migration on page 238.

Backup and Restore: Backup and restore HPE SOA Registry Foundation. Follow Backup on page 244

Authentication account provider: Change account backend configuration. Follow External Accounts Integration on page 138.

Server Properties

System properties are the main means of configuring HPE SOA Registry Foundation as deployed into Systinet Server for Java. Default values for these properties are in the resource `META-INF/wasp.properties`, which is located in `lib/runner.jar`.

There are two ways to alter system properties, for the two different types of HPE SOA Registry Foundation installation:

- *Standalone Installation*: Set the property from the command line when starting the server from either the `REGISTRY_HOME/bin/server.bat` or `server.sh` script. The syntax is:

  `server(.sh)` [-D`name of property=value`] { start | stop }

  For example: `server -Didoox.debug.level=4 start`

- *HPE SOA Registry Foundation deployed to an application server*: Default property values can be overridden in the `init-param` elements in the web application deployment descriptor, `web.xml`.

The following properties are checked when HPE SOA Registry Foundation is initialized:

Property	Description
`wasp.location`	This property is mandatory for running a HPE SOA Registry Foundation server. It must point to the directory in which HPE SOA Registry Foundation is installed.
`wasp.config.location`	This is an absolute or `wasp.location`-relative path pointing to the registry configuration file. Setting this property is optional; the default value is `conf/clientconf.xml`.
`wasp.config.include`	Comma-separated list of additional config paths to include. These paths can be either absolute or relative to the working directory. This property is optional.
`wasp.impl.classpath`	Sets a classpath for the registry implementation. This property is optional; if it is not set, registry interfaces and implementation are loaded in the same classloader.
`wasp.shutdownhook`	Set to `true` if HPE SOA Registry Foundation should be automatically destroyed just before JVM is destroyed. Set to `false` if you want to manage the shutdown process yourself. The default setting is `true`.
`idoox.debug.level`	Determines the number of debugging messages produced by HPE SOA Registry Foundation: • 0: none • 1: errors • 2: warnings • 3: infos • 4: debugs This property is optional; the default value is 2 for the client and 3 for the server. The debug level is available in the non-stripped distribution only. The logging level specified by the `idoox.debug.level` property overrides the level specified in the configuration file determined by the `log4j.configuration property`
`idoox.debug.logger`	Specifies which logging system is used, `waspLogger` or `log4j`. Default is `log4j`. Setting the value of this property to waspLogger uses this logger, instead.
`log4j.configuration`	Specifies the location of the configuration (properties file) for log4j. This property can contain a relative (`conf/log4j.config`) or absolute (`/home/waspuser/log4j.config`) path to the configuration file. If it is not set, the default configuration (`ConsoleAppender` with the pattern `%p: %c{2} - %m\n`) will be used. An example configuration file for log4j, `log4j.config`, is located in the

Property	Description
	conf subdirectory of the HPE SOA Registry Foundation installation directory.

Windows Services

Use the following scripts to install, uninstall, start, and stop HPE SOA Registry Foundation as a Windows service:

InstallService - `InstallService.bat`

Installs HPE SOA Registry Foundation into system services

UnInstallService - `UnInstallService.bat`

Uninstalls HPE SOA Registry Foundation from system services.

StartService - `StartService.bat`

Starts the already installed HPE SOA Registry Foundation service.

StopService - `StopService.bat`

Stops the started HPE SOA Registry Foundation service.

See "NT Service Support" on page 189.

Logs

There are four log files in `REGISTRY_HOME/log` directory.

These two log files are produced by the Installation and Setup processes:

`install.log`

This log contains installation output information including all properties set during installation, and output from the installation process. If an error occurs during installation, see this log for details.

`setup.log`

The log of the Setup tool. Any execution of the Setup tool writes the set properties and output from setup processes here. Errors occurring during setup are written to this log.

The default server logs are:

`logEvents.log`

The standard server output contains informative events which occur on the HPE SOA Registry Foundation server.

`errorEvents.log`

This file contains detailed logs of error events which occur on the HPE SOA Registry Foundation server.

replicationEvents.log

Replication process logs can be found in the `REGISTRY_HOME/log/replicationEvents.log` file.

configuratorEvents.log

Cluster configuration events are logged in the `REGISTRY_HOME/log/configuratorEvents.log` file

wasp_NTService.log

Events of the server are written into the REGISTRY_HOME\log\wasp_NTService.log file.

The server logs may be configured by one of two logging systems, the in-house `waspLogger` and `log4j`. By default, `log4j` is used. The default log4j configuration file is located in `REGISTRY_HOME/conf/log4j.config`.

> **Note:** An explanation of using `log4j` is outside the scope of this documentation; see the Apache log4j documentation for more information.

Troubleshooting

If errors occur during the installation process, the installer displays a message and a **Recovery** button.

Execution of Task fails. You can click **Recovery** and correct erroneous selections or click **Exit** to exit the installation.

If you click **Recovery**, the installation returns to the step that should be corrected. For example, if the installation fails during copying files, it will return to the installation type panel. If the process fails during configuring database it will return to the database panels.

If errors occur when using the Setup tool, only the error message is displayed, you can continue by clicking **Next**.

The following general problems may occur:

Installation backend timeout

If the task does not respond for a long time, a timeout error is thrown and the task is stopped. The default timeout is 30 minutes. If you have a slow machine, try to redefine the **`timeout`** system property for a greater value in minutes at a java command line.

For 60 minutes, run installation by following command: **java -Dtimeout=60 -jar hpe-soa-registry-foundation-10.04.jar**

For 60 minutes, edit the `setup.sh` (`setup.bat`) file; add the `-Dtimeout=60` option into the java command line so it looks like:

Windows:	"%JAVA_CMD%" -Dtimeout=60
UNIX:	"$JAVA_CMD" -Dtimeout=60

Cannot find JDBC driver, java.lang.ClassNotFoundException

Some external classes cannot be found. Usually the path to JDBC driver does not contain the needed `*.jar` or `*.zip` files. Another reason this error may be thrown is that the JDBC driver is not supported by HPE SOA Registry Foundation. See "Database Installation" on page 80 for more information about supported databases.

Cannot access database, java.sql.SQLException

This usually happens during the creation of database which already exists. To resolve this error, try to connect or drop this database first.

This error is also thrown when trying to drop a database which is currently in use, or does not exist. Note that some set properties must exist on the database engine and some of them are optional. See "Database Installation" on page 80 for more information about supported databases.

Couldn't create or access important files. Wrong path

This error is displayed when the installation directory specified is bad or the user does not have read and write permissions for it. Try to install to another directory or reset the read and write permissions.

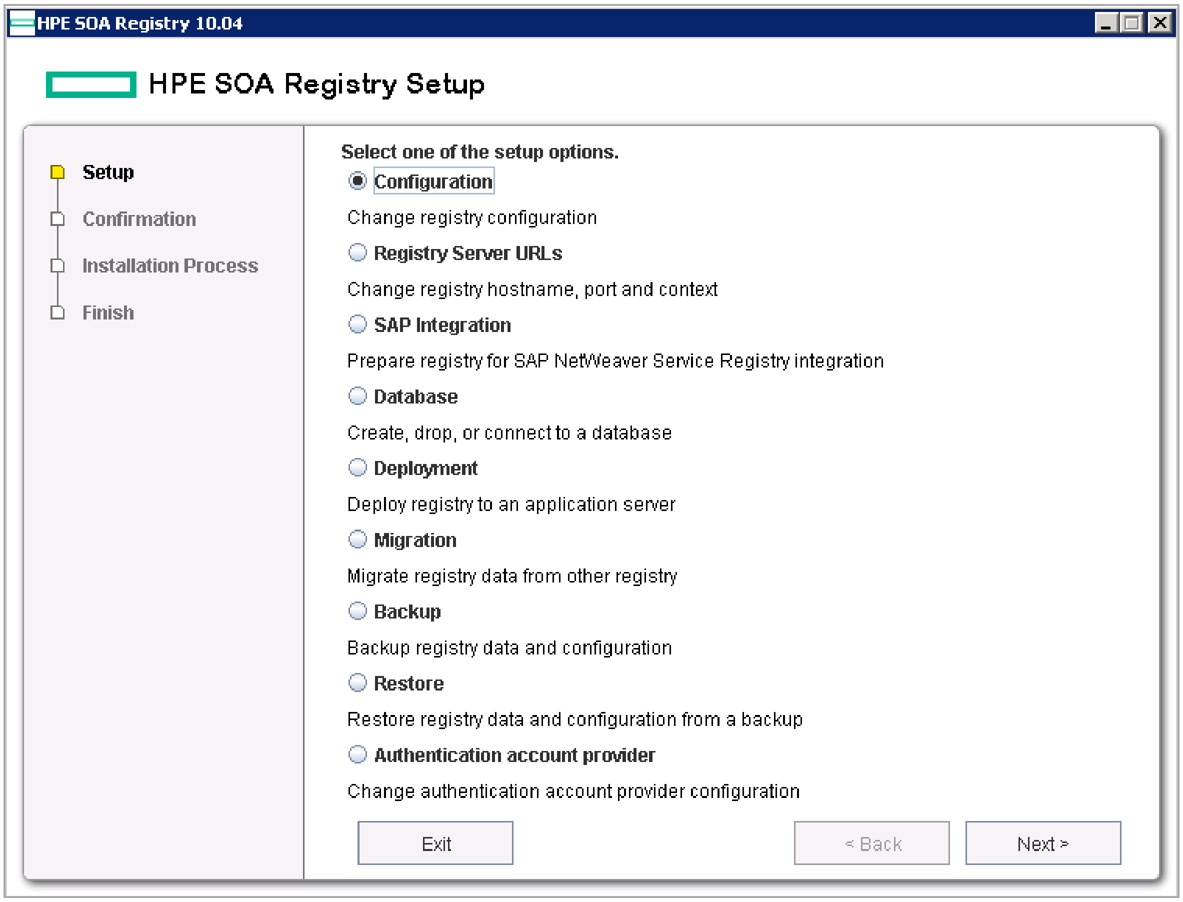
Server Configuration

The server configuration may be set up during installation or by using the Setup tool after installation. Both of these scenarios use the same set of GUI panels shown in this section.

To run the Setup tool, execute the following script from the `bin` subdirectory of your installation:

Windows:	setup.bat
UNIX:	./setup.sh

See command-line parameters in "Setup" in "Command Line Scripts" on page 59.



Select one of the setup options.

● Configuration

Change registry configuration

○ Registry Server URLs

Change registry hostname, port and context

○ SAP Integration

Prepare registry for SAP NetWeaver Service Registry integration

○ Database

Create, drop, or connect to a database

○ Deployment

Deploy registry to an application server

○ Migration

Migrate registry data from other registry

○ Backup

Backup registry data and configuration

○ Restore

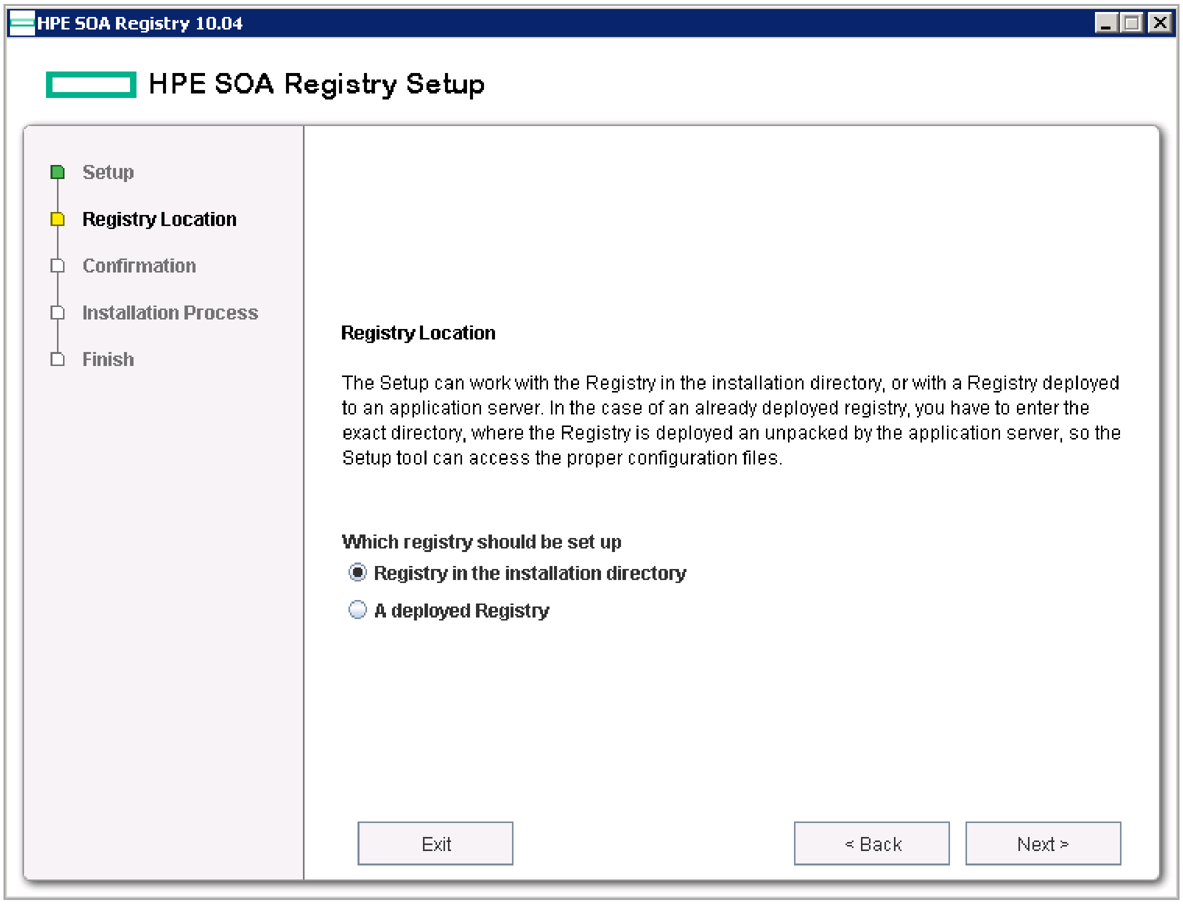
Restore registry data and configuration from a backup

○ Authentication account provider

Change authentication account provider configuration

For more information on the Setup tool, see "Reconfiguring After Installation" on page 62.

Select whether you want to setup HPE SOA Registry Foundation that has been deployed (second choice) or not (first choice).



Registry Location

The Setup can work with the Registry in the installation directory, or with a Registry deployed to an application server. In the case of an already deployed registry, you have to enter the exact directory, where the Registry is deployed an unpacked by the application server, so the Setup tool can access the proper configuration files.

Which registry should be set up

● Registry in the installation directory

○ A deployed Registry

The "SMTP Configuration" panel allows you to configure SMTP. The SMTP configuration is important when users needs to receive email notification from subscriptions.

SMTP Host Name - Host name of the SMTP server, through which all e-mail alerts and notification are sent to administrator and users.

SMTP Port - Port number for this SMTP server

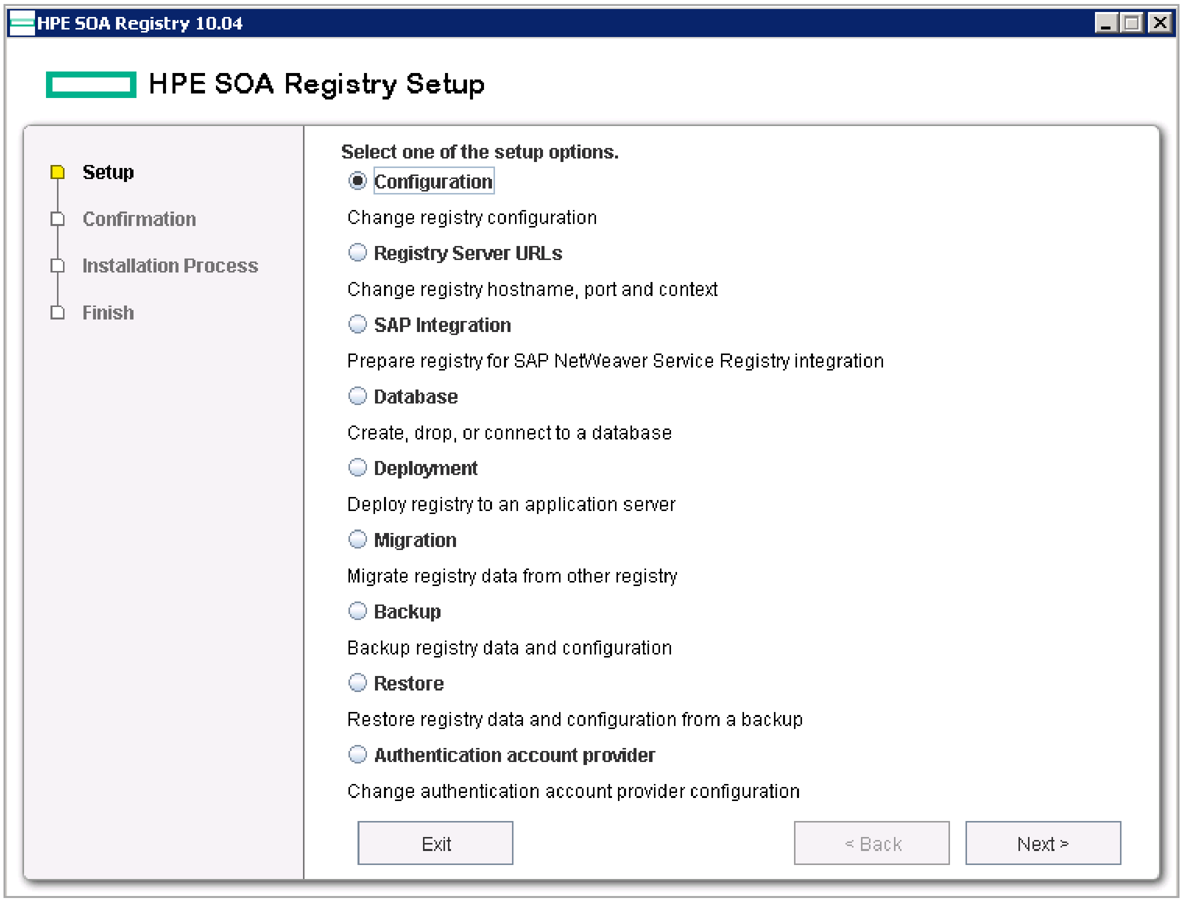
SMTP Password - Password to access SMTP server

Confirm password - Retype the same password. Note that if it is not same as the password in the previous box, you cannot continue.

SMTP Default Sender E-mail, Name - HPE SOA Registry Foundation will generate email messages with this identity.

SMTP Configuration

Select **Configuration** on the first panel.



Select one of the setup options.

● Configuration

Change registry configuration

○ Registry Server URLs

Change registry hostname, port and context

○ SAP Integration

Prepare registry for SAP NetWeaver Service Registry integration

○ Database

Create, drop, or connect to a database

○ Deployment

Deploy registry to an application server

○ Migration

Migrate registry data from other registry

○ Backup

Backup registry data and configuration

○ Restore

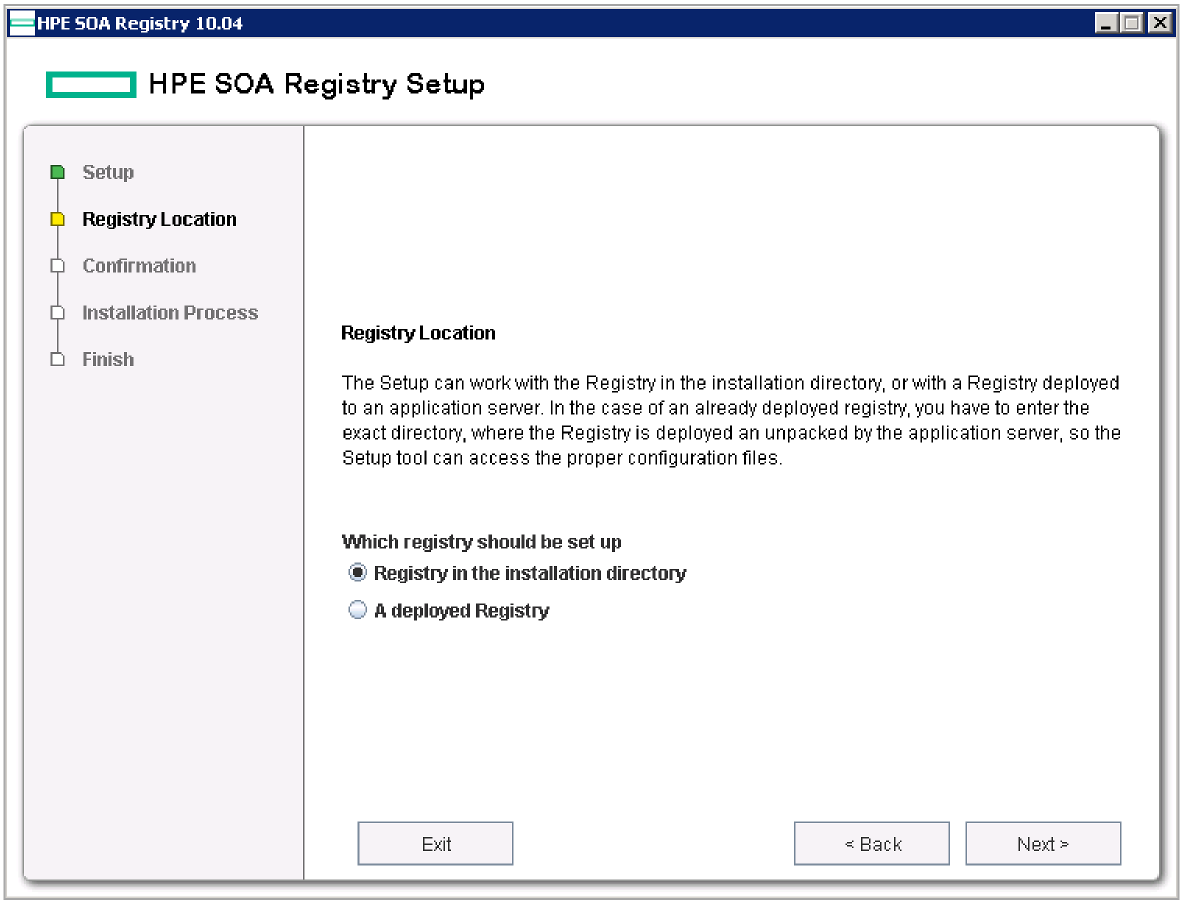
Restore registry data and configuration from a backup

○ Authentication account provider

Change authentication account provider configuration

For more information on the Setup tool, see "Reconfiguring After Installation" on page 62.

Select whether you want to set up HPE SOA Registry Foundation that has been deployed (second choice) or not (first choice).



Registry Location

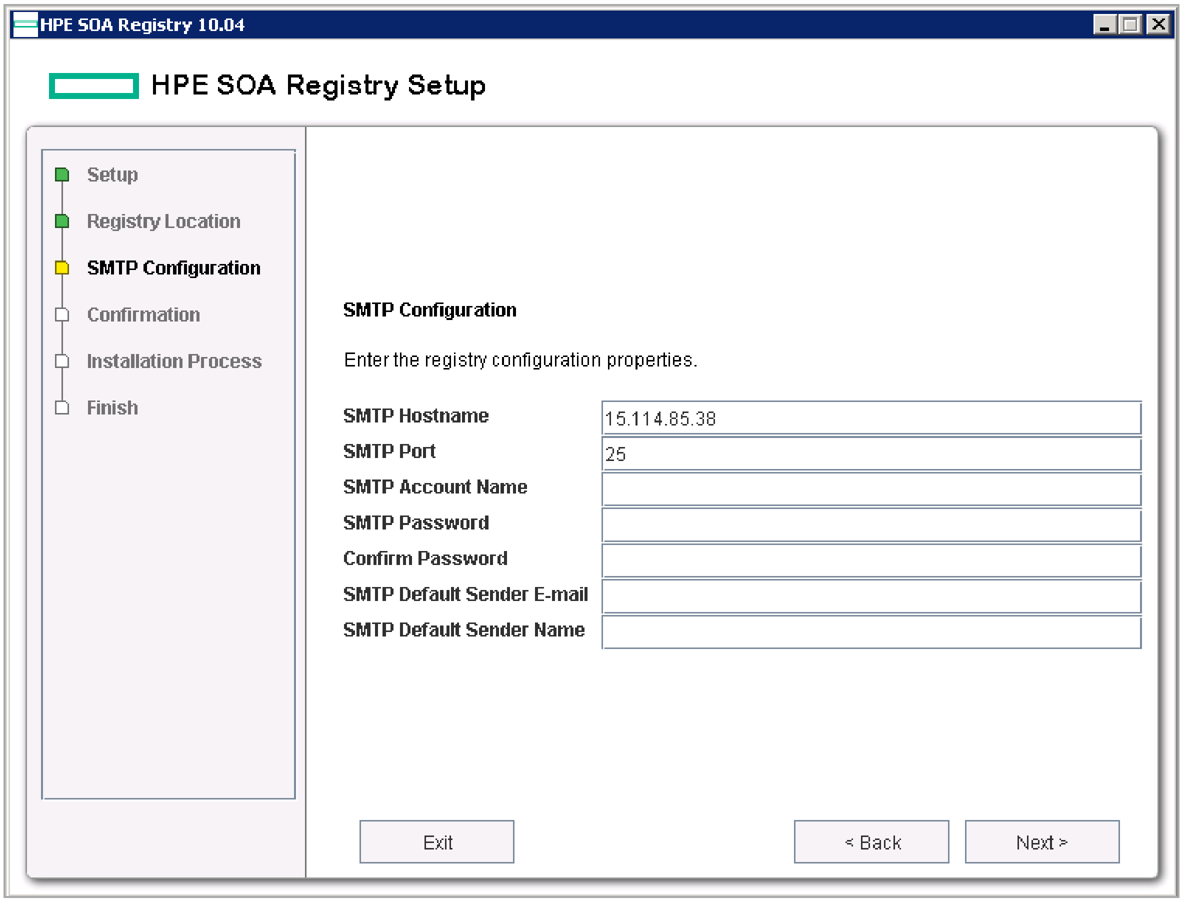
The Setup can work with the Registry in the installation directory, or with a Registry deployed to an application server. In the case of an already deployed registry, you have to enter the exact directory, where the Registry is deployed an unpacked by the application server, so the Setup tool can access the proper configuration files.

Which registry should be set up

● Registry in the installation directory

○ A deployed Registry

The SMTP configuration is important when users needs to receive email notifications from subscriptions.



SMTP Configuration

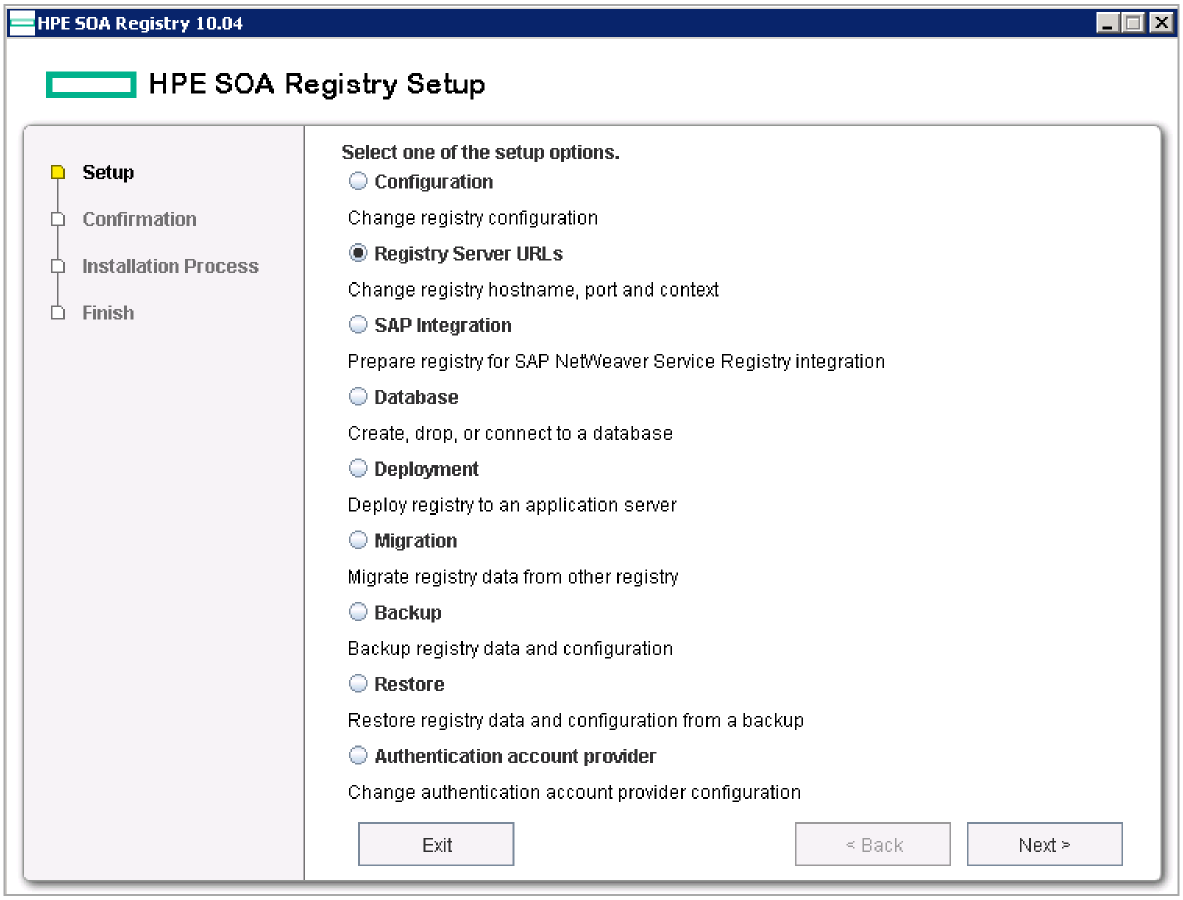
Enter the registry configuration properties.

SMTP Hostname	15.114.85.38
SMTP Port	25
SMTP Account Name	
SMTP Password	
Confirm Password	
SMTP Default Sender E-mail	
SMTP Default Sender Name	

- **SMTP Host Name:** Host name of the SMTP server, through which all e-mail alerts and notification are sent to administrator and users.
- **SMTP Port:** Port number for this SMTP server.
- **SMTP Password:** Password to access SMTP server.
- **Confirm password:** Retype the same password. Note that if it is not same as the password in the previous box, you cannot continue.
- **SMTP Default Sender E-mail, Name:** HPE SOA Registry Foundation will generate email messages with this identity.

Change Server URLs

Select **Registry Server URLs** on the first panel.



Select one of the setup options.

○ **Configuration**

Change registry configuration

● **Registry Server URLs**

Change registry hostname, port and context

○ **SAP Integration**

Prepare registry for SAP NetWeaver Service Registry integration

○ **Database**

Create, drop, or connect to a database

○ **Deployment**

Deploy registry to an application server

○ **Migration**

Migrate registry data from other registry

○ **Backup**

Backup registry data and configuration

○ **Restore**

Restore registry data and configuration from a backup

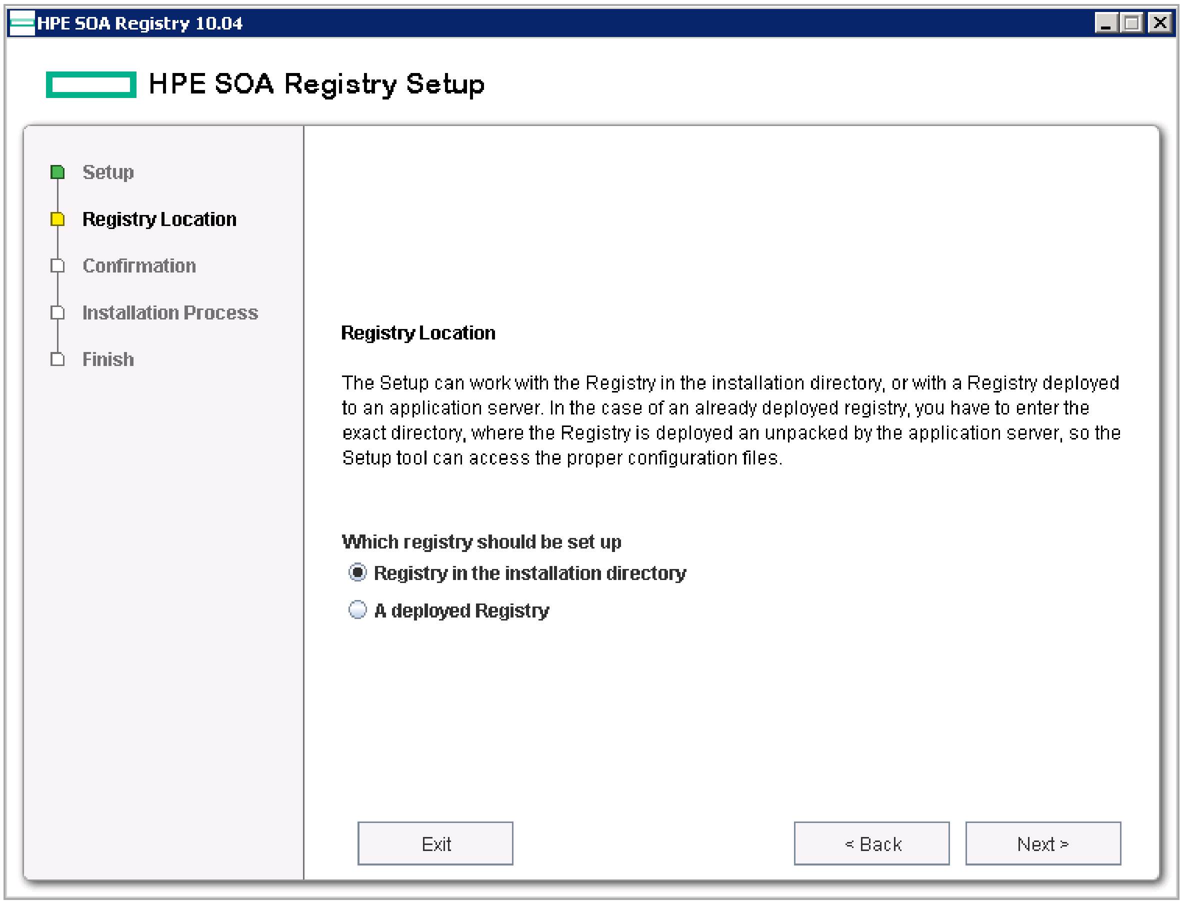
○ **Authentication account provider**

Change authentication account provider configuration

For more information on the Setup tool, see "Reconfiguring After Installation" on page 62.

Select whether you want HPE SOA Registry Foundation to setup only a configuration (first option) or to only alter a earlier deployment.

Setup



Registry Location

The Setup can work with the Registry in the installation directory, or with a Registry deployed to an application server. In the case of an already deployed registry, you have to enter the exact directory, where the Registry is deployed an unpacked by the application server, so the Setup tool can access the proper configuration files.

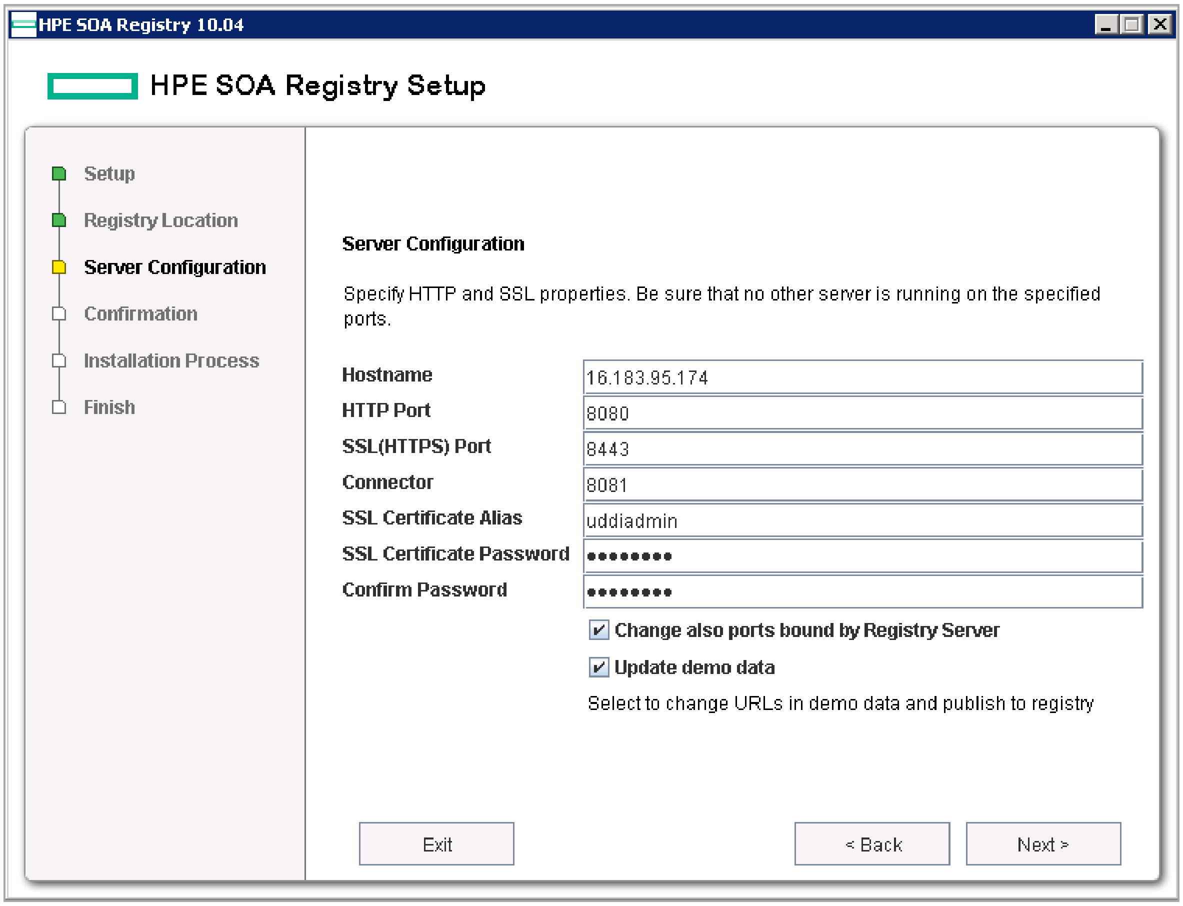
Which registry should be set up

● Registry in the installation directory

○ A deployed Registry

Change Server URLs standalone deployment

This panel allows you to configure server URLs. There is no context field in a standalone deployment.



Server Configuration

Specify HTTP and SSL properties. Be sure that no other server is running on the specified ports.

Hostname	16.183.95.174
HTTP Port	8080
SSL(HTTPS) Port	8443
Connector	8081
SSL Certificate Alias	uddiadmin
SSL Certificate Password	••••••••
Confirm Password	••••••••

☑ Change also ports bound by Registry Server

☑ Update demo data

Select to change URLs in demo data and publish to registry

Host name: Host name of the computer on which HPE SOA Registry Foundation is installed; change the auto-completed entry if it is different.

HTTP Port: The non-secure port for accessing the Registry Console (default value: 8080)

SSL (HTTPS) Port: Secure port for accessing the Registry Console (default value: 8443)

Connector: Connector port is used by standalone server to listen for control signals. No other application may use this port (default value: 8081)

SSL Certificate Alias: Alias used for the identification of an SSL private key used in protected store management. For more information see “PStore Tool”. (default value: uddiadmin)

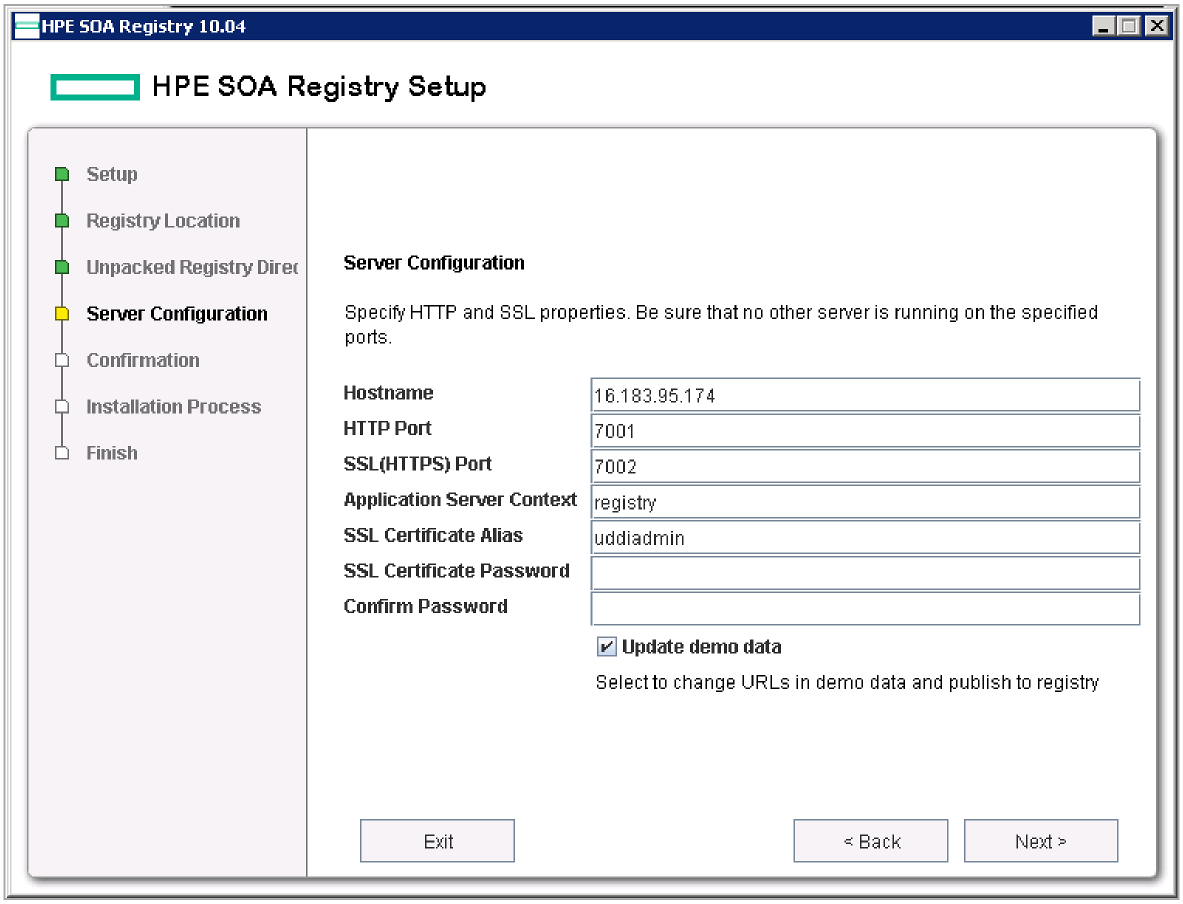
SSL Certificate password: Password to encrypt SSL private key.(default value: changeit)

Confirm password: Retype the same password. Note that if it is not identical to the first one, you will not be able to continue

Change also ports bound by Registry Server: Select this option to change the ports bound to the Jetty server. When 'Change also ports bound by Registry Server' is selected, the connector port field will be enabled otherwise it will remain disabled.

Update demo data: When the option Update demo data is selected, demo data will be deployed to the registry. It is not imported during installation.

Change Server URLs Oracle WebLogic deployment



Server Configuration

Specify HTTP and SSL properties. Be sure that no other server is running on the specified ports.

Hostname	16.183.95.174
HTTP Port	7001
SSL(HTTPS) Port	7002
Application Server Context	registry
SSL Certificate Alias	uddiadmin
SSL Certificate Password	
Confirm Password	

☑ Update demo data

Select to change URLs in demo data and publish to registry

Application Server Context: Context part of the URL, used to access Oracle Service Registry ported to an application server.

The host name, SSL Certificate Alias, and SSL password are used to create a new security identity in the local protected store. It creates a certificate and adds this certificate to `REGISTRY_HOME/conf/clientconf.xml`, `REGISTRY_HOME/conf/pstore.xml`, and also exports it to the certificate file `REGISTRY_HOME/doc/registry.crt`. See "PStore Tool" on page 361 for instructions in how to operate the protected security store.

> **Note:** The certificate generated by Registry is signed by our Demo Certification Authority. This enables HPE SOA Systinet 10.04 to access HPE SOA Registry Foundation without additional trust setup when deployed to JBoss. Using the generated certificate for production is not recommended.

After setting these properties, the server will be available at `http://[host name]:[HTTP Port]/[Context of URL]`. For example, in the "Change Server URLs standalone deployment" above, the server is available at `http://mydomain.mycompany.com:8080/uddi` and at `https://mydomain.mycompany.com:8443/uddi`. Note that communication could be spied upon by using the SoapSpy tool, see "How to Debug" on page 526.

SAP Integration

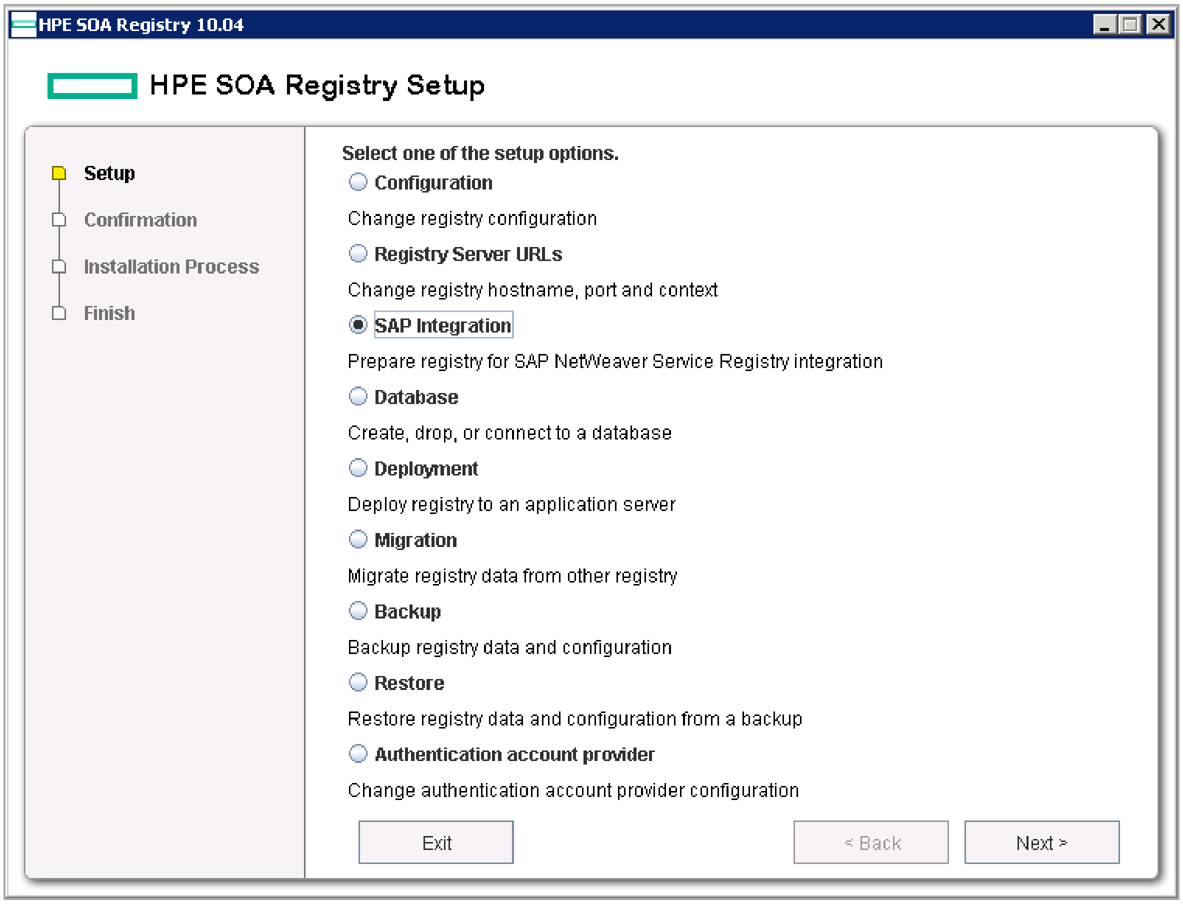
SAP Integration is set by using the Setup tool after installation. This scenario uses the set of GUI panels for SAP integration shown in this section.

To run the Setup tool, execute the following script from the bin subdirectory of your installation:

Windows:	setup.bat
UNIX:	./setup.sh

See command-line parameters in "Setup" in "Command Line Scripts" on page 59.

Select **SAP Integration** on the first panel.



Select one of the setup options.

○ Configuration

Change registry configuration

○ Registry Server URLs

Change registry hostname, port and context

● SAP Integration

Prepare registry for SAP NetWeaver Service Registry integration

○ Database

Create, drop, or connect to a database

○ Deployment

Deploy registry to an application server

○ Migration

Migrate registry data from other registry

○ Backup

Backup registry data and configuration

○ Restore

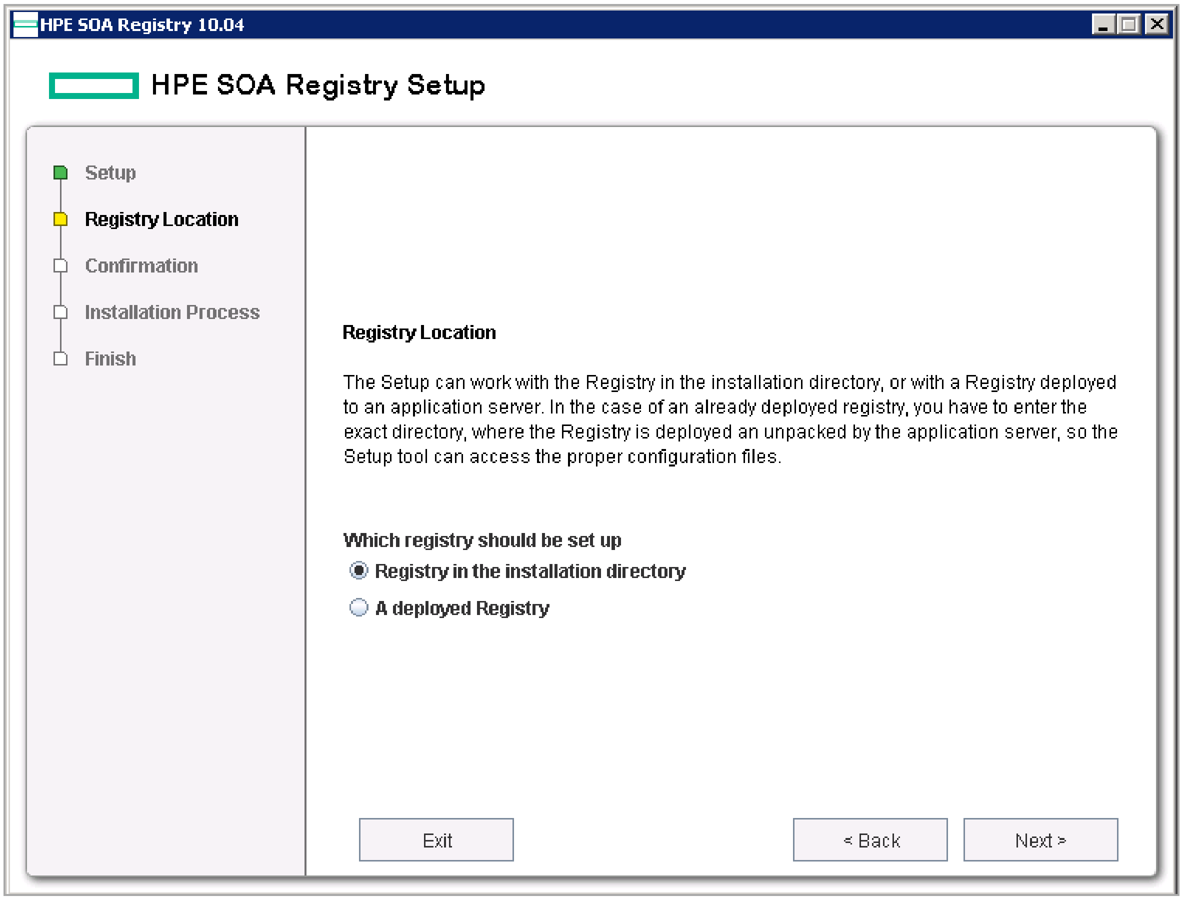
Restore registry data and configuration from a backup

○ Authentication account provider

Change authentication account provider configuration

For more information on the Setup tool, see "Reconfiguring After Installation" on page 62.

Select whether you want HPE SOA Registry Foundation to setup only a configuration (first option) or to only alter a earlier deployment (second option).



Registry Location

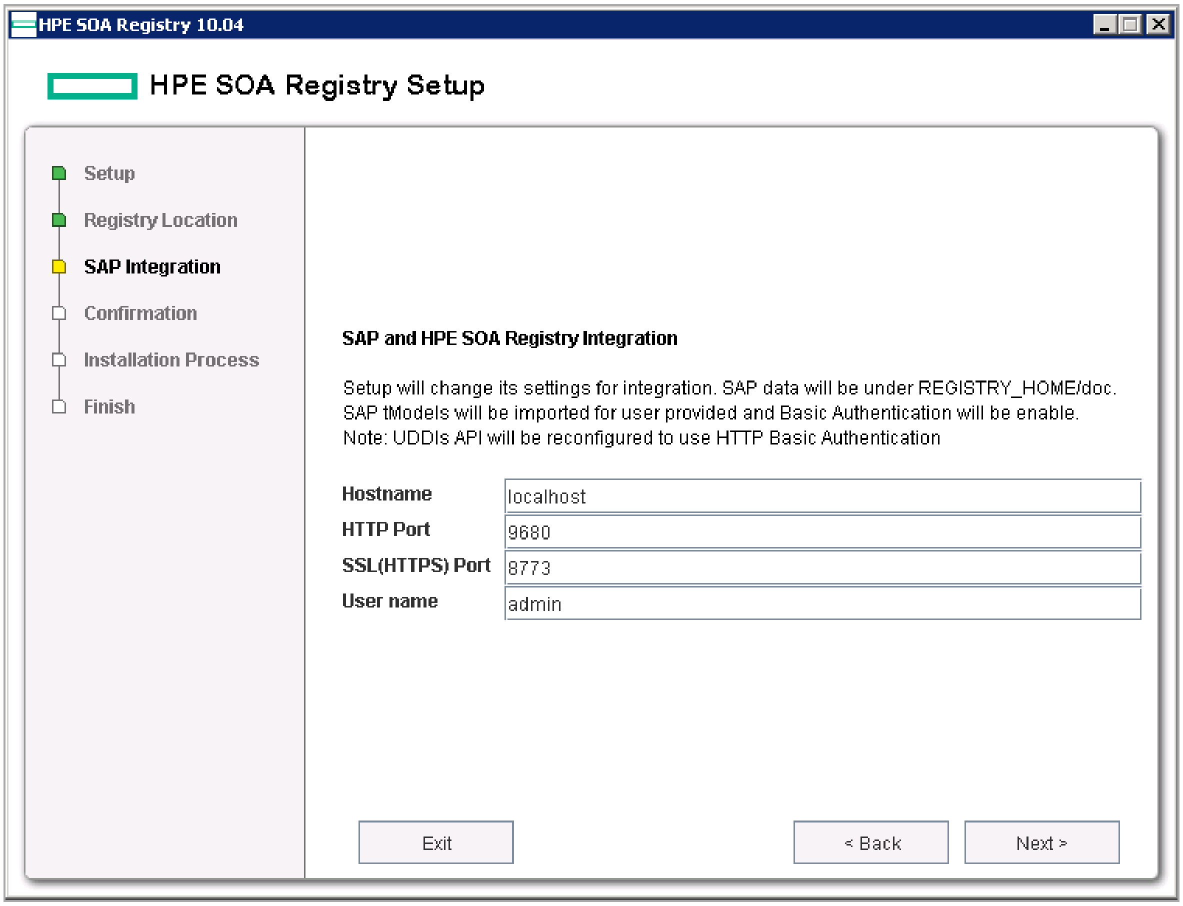
The Setup can work with the Registry in the installation directory, or with a Registry deployed to an application server. In the case of an already deployed registry, you have to enter the exact directory, where the Registry is deployed an unpacked by the application server, so the Setup tool can access the proper configuration files.

Which registry should be set up

● Registry in the installation directory

○ A deployed Registry

The SAP integration settings are used for the host name, HTTP(S) and user name.



SAP and HPE SOA Registry Integration

Setup will change its settings for integration. SAP data will be under REGISTRY_HOME/doc.
SAP tModels will be imported for user provided and Basic Authentication will be enable.
Note: UDDIs API will be reconfigured to use HTTP Basic Authentication

Hostname: localhost
HTTP Port: 9680
SSL(HTTPS) Port: 8773
User name: admin

Host name: Host name of the computer on which HPE SOA Registry Foundation is installed; change the autocompleted entry if it is different.

HTTP Port: The non-secure port for accessing the Registry Console (default value: 8080)

SSL (HTTPS) Port: Secure port for accessing the Registry Console (default value: 8443)

User name: User name for logging on the Registry Console (default value: admin)

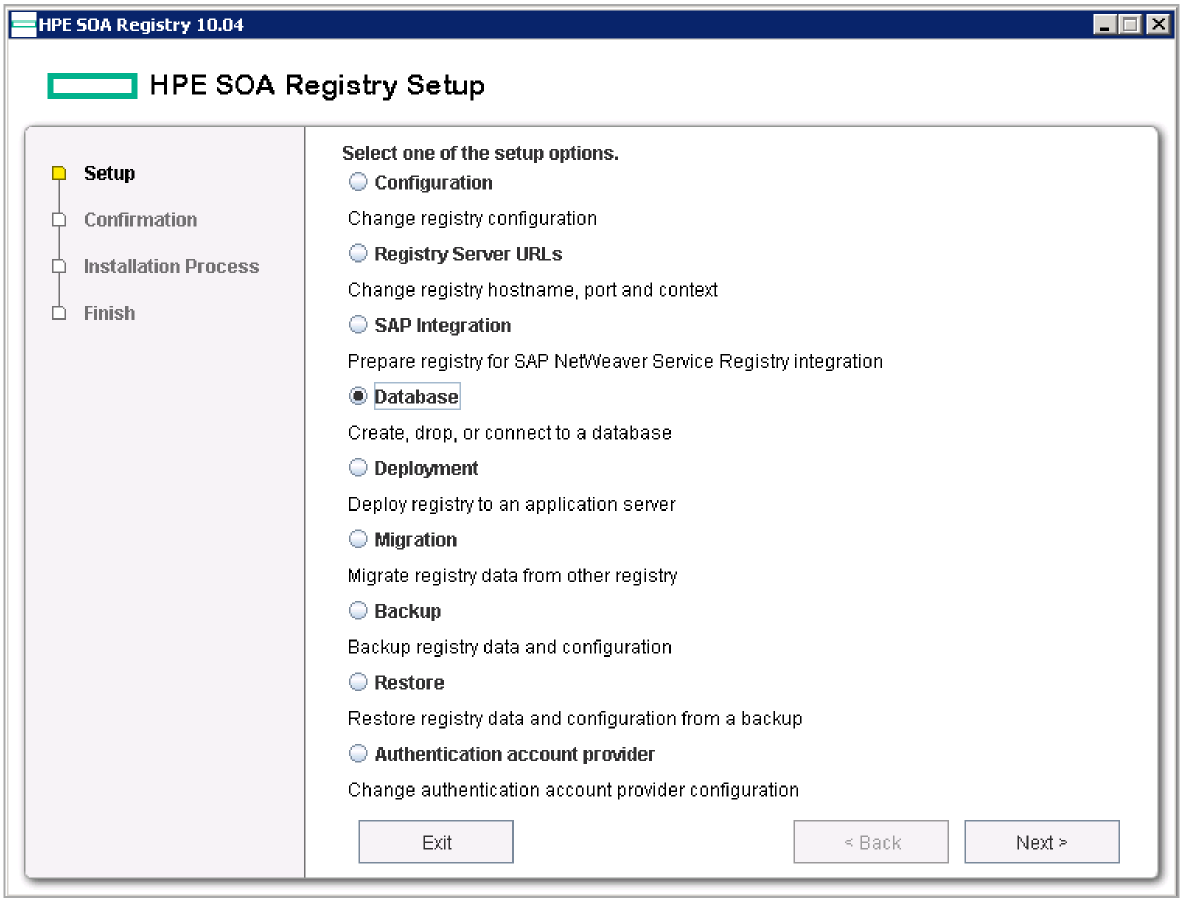
Database Installation

The database may be set up during installation or by using the Setup tool after installation. Both of these scenarios use the same set of GUI panels shown in this section.

To run the Setup tool, execute the following script from the `bin` subdirectory of your installation:

Windows:	setup.bat
UNIX:	./setup.sh

See command-line parameters in "Setup" in "Command Line Scripts" on page 59.



Select one of the setup options.

○ Configuration
Change registry configuration

○ Registry Server URLs
Change registry hostname, port and context

○ SAP Integration
Prepare registry for SAP NetWeaver Service Registry integration

● Database
Create, drop, or connect to a database

○ Deployment
Deploy registry to an application server

○ Migration
Migrate registry data from other registry

○ Backup
Backup registry data and configuration

○ Restore
Restore registry data and configuration from a backup

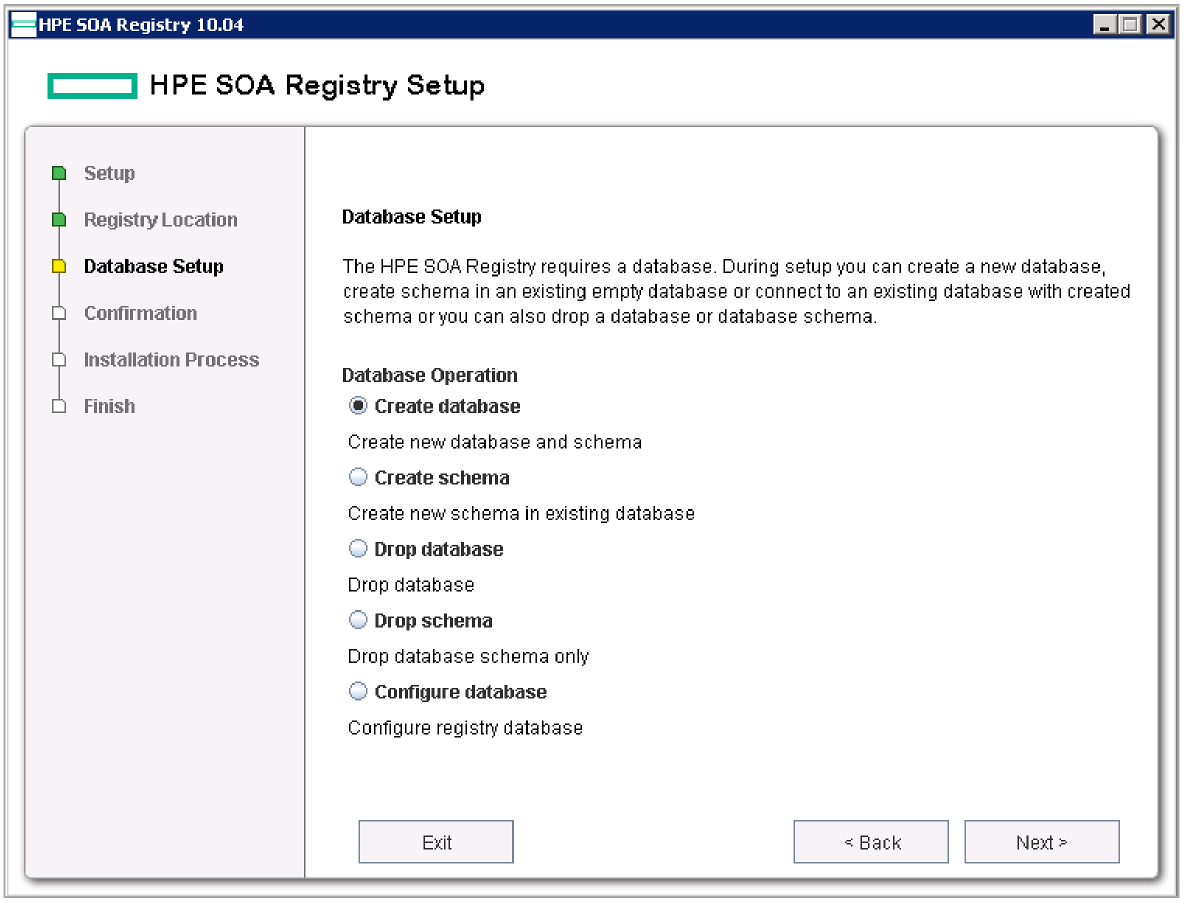
○ Authentication account provider
Change authentication account provider configuration

Select your database. For more information on the Setup tool, see "Reconfiguring After Installation" on page 62.

Database Creation Method

The registry requires a database. During installation you can create a new database, create schema in an existing empty database or connect to an existing database with created schema. Using the Setup tool, you can also drop a database or database schema.

Select your database operation on the following panel:



Database Setup

The HPE SOA Registry requires a database. During setup you can create a new database, create schema in an existing empty database or connect to an existing database with created schema or you can also drop a database or database schema.

Database Operation

● Create database

Create new database and schema

○ Create schema

Create new schema in existing database

○ Drop database

Drop database

○ Drop schema

Drop database schema only

○ Configure database

Configure registry database

Select a method from those shown in the above figure.

Create database: Create new database/users/tablespaces (depending on the type of database server) and database schema. This is the easiest way to attach the required database to HPE SOA Registry Foundation. Note that you must have the credentials of the database administrator.

Create schema: Create a new schema in existing database. Select this method if you have access to an existing empty database with the ability to create tables and indexes. This option is suitable when you does not know the administrator's credentials. We assume the administrator has already created a new database/users/tablespaces for this option.

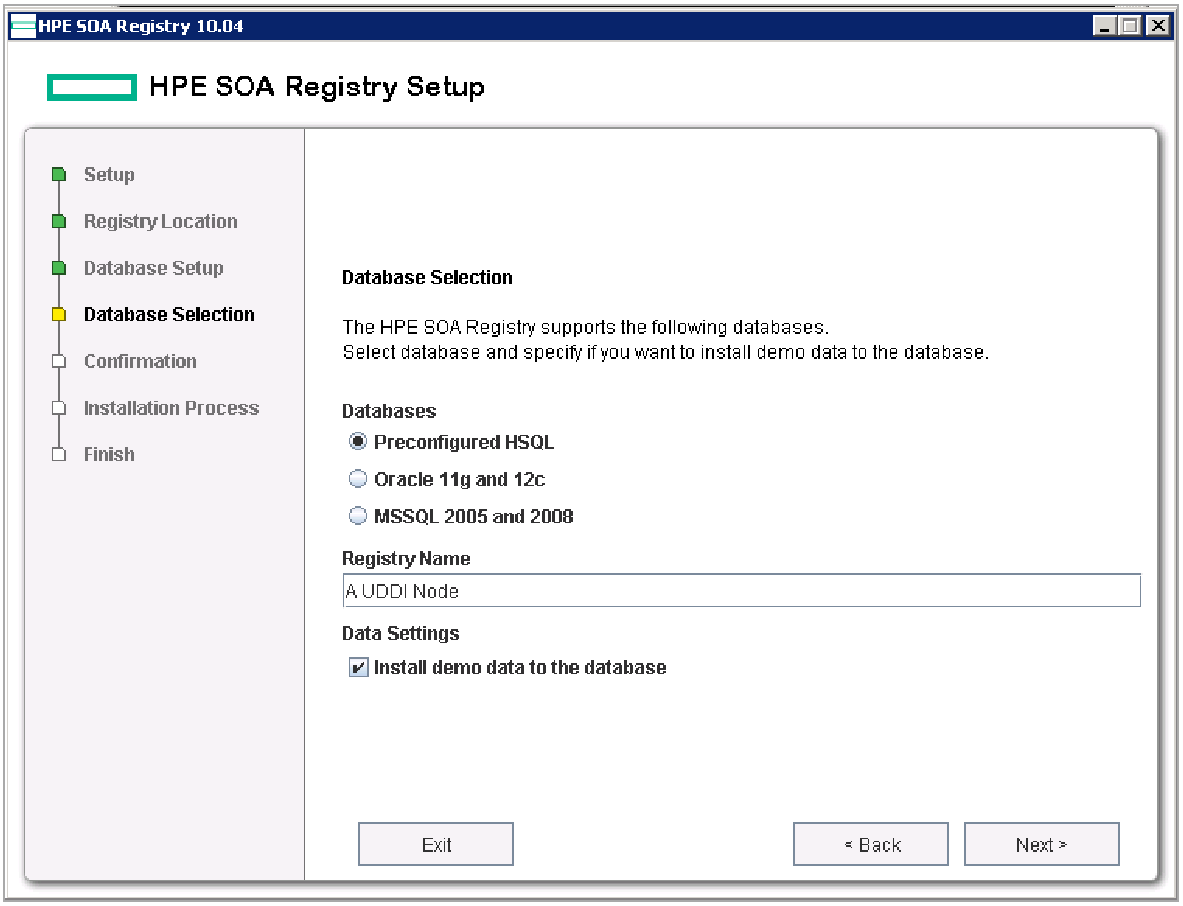
Drop database: Drop databaseDrops the whole database/users/tablespaces. Note that this option depends on the type of database server.

Drop schema: Drops all tables in the database but leave the empty database.

Configure database: Configure registry database. Use this method if the registry database already exists, for example, from a previous HPE SOA Registry Foundation installation of the same release number, and fill in only the connection parameters.

Select Database Type

The “Select Database Type” panel shows the supported database engines that can be prepared for HPE SOA Registry Foundation. The panel may differ if another method was selected in the previous step.



Database Selection

The HPE SOA Registry supports the following databases.
Select database and specify if you want to install demo data to the database.

Databases

● Preconfigured HSQL

○ Oracle 11g and 12c

○ MSSQL 2005 and 2008

Registry Name

A UDDI Node

Data Settings

☑ Install demo data to the database

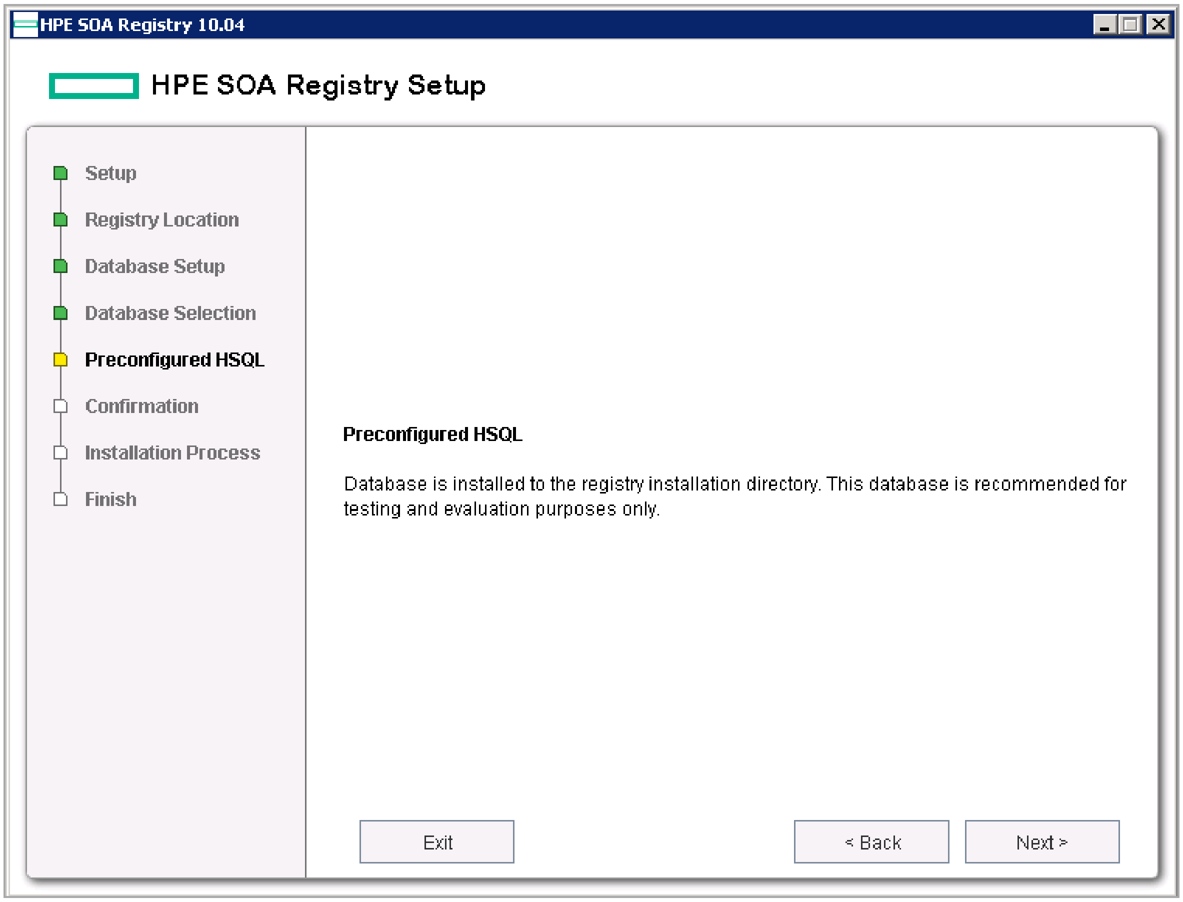
Follow these links for the selected database:

- "Preconfigured HSQL" on the next page
- "Oracle" on page 85
- " MSSQL 2005 or 2008" on page 86

Preconfigured HSQL

The default database is the preconfigured HSQL. The installer or Setup tool creates a database named ***REGISTRY_HOME/hsqldb/uddinode*** with the user account ***uddiuser*** with the password ***uddi*** in the database. Note that all database files can be found in `REGISTRY_HOME/hsqldb directory`.

> **Note:** This database is recommended for evaluation and testing purposes only.
>
> If you use HSQL then the user credentials are stored in the HSQL database files in plain text. You must protect these files from unauthorized reading using the appropriate filesystem access rights. The files are located in the directory `REGISTRY_HOME/hsqldb/` by default.

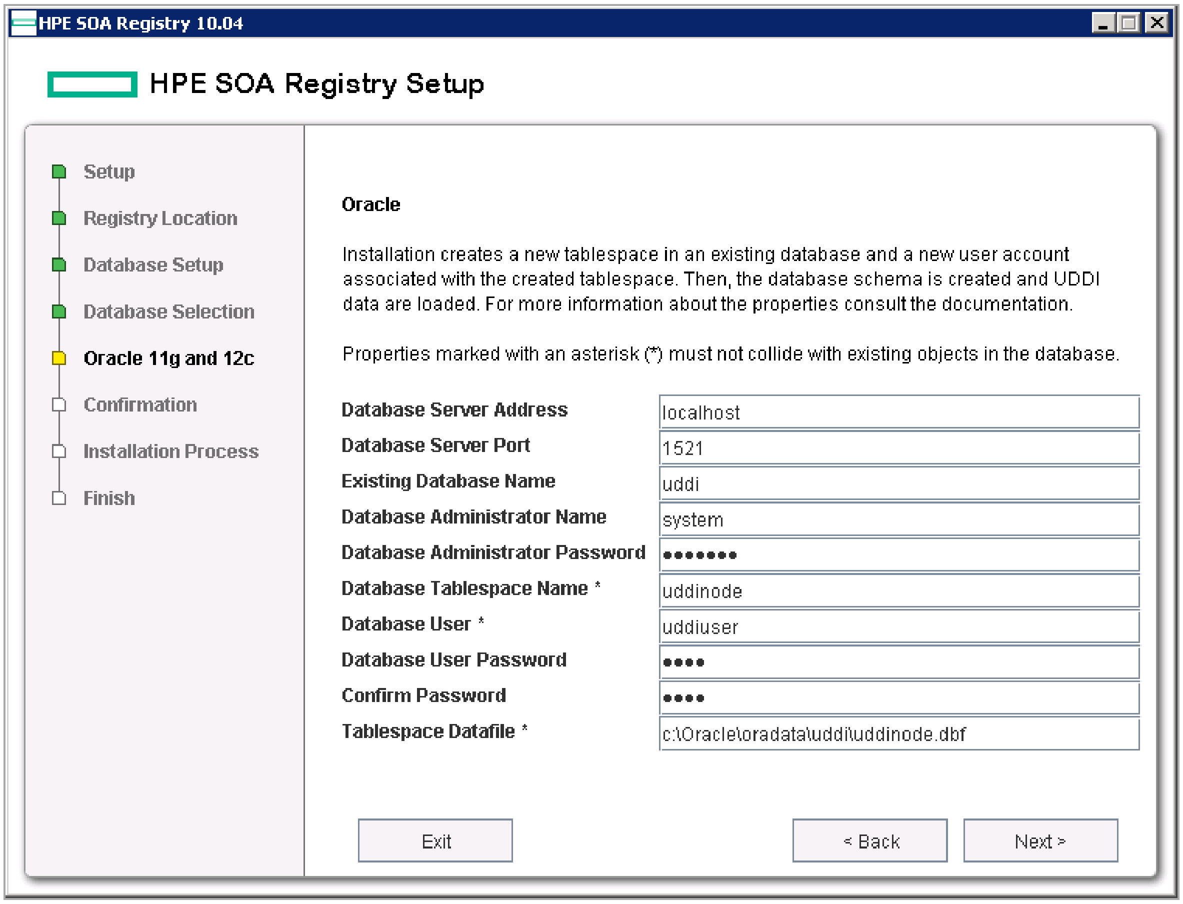


Preconfigured HSQL

Database is installed to the registry installation directory. This database is recommended for testing and evaluation purposes only.

Oracle

The **Create database** option on the installer/Setup tool does not mean to create a new physical database. The installation process only creates a new tablespace in an existing database and a new user of the default tablespace is set up on the created one. Then a database schema is created and UDDI data are loaded. Because relational tables are created in the schema of the specified user, if you want to create more UDDI databases, you must create UDDI databases with different database users.



Oracle

Installation creates a new tablespace in an existing database and a new user account associated with the created tablespace. Then, the database schema is created and UDDI data are loaded. For more information about the properties consult the documentation.

Properties marked with an asterisk (*) must not collide with existing objects in the database.

Database Server Address	localhost
Database Server Port	1521
Existing Database Name	uddi
Database Administrator Name	system
Database Administrator Password	•••••••
Database Tablespace Name *	uddinode
Database User *	uddiuser
Database User Password	••••
Confirm Password	••••
Tablespace Datafile *	c:\Oracle\oradata\uddi\uddinode.dbf

Oracle database creation requires the following properties. To connect or create a schema requires a subset of these properties. Please note that properties marked with an asterisk (*) must not collide with existing objects in the database.

Database Server Address: Usually the host name or IP address of the computer where the database server is accessible.

Database Server Port: Port on which the database listens for a connection

Existing Database Name: Name of a database that already exists into which the HPE SOA Registry Foundation tablespace will be created.

Database Administrator Name: User name of the administrator of the database; required to create a new tablespace on the existing database

Database Administrator Password: Password for the administrator account specified in the previous text box.

Database Tablespace Name *: Name of the tablespace to be created in the existing database and which will store UDDI data structures.

Database User *: A new user account which will be created to connect to the tablespace.

Database User Password: Password for the user account specified in the previous text box.

Confirm password: Again, if it is not the same as in the previous text box, you cannot continue.

Tablespace Datafile *: Enter the path to the tablespace data file. If you use Oracle with Automatic Storage Management (ASM) you may want to specify the disk group in the Tablespace Datafile field in format: "+DATA" (the DATA is default group name). If you do not use this extension, specify the filename of the datafile (including the file extension and the full path on the database server).

Continue with "JDBC Driver" on page 88.

MSSQL 2005 or 2008

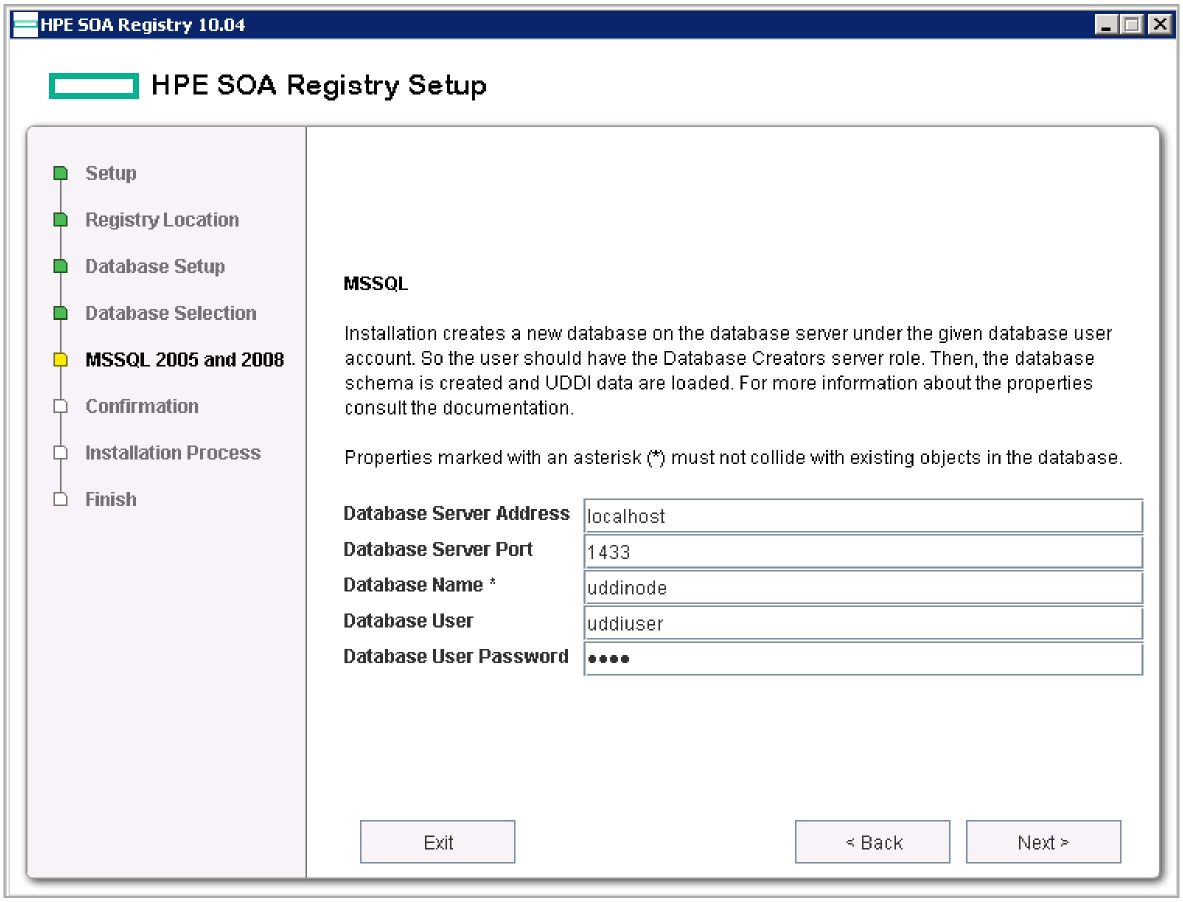
You have to select right version of MSSQL. Either MSSQL 2005 or MSSQL 2008 can be selected in panel shown in figure Select Database Type. The options that follow are same for both but the versions differ in connection string and JDBC class name so that the selected version must match the version of database.

The installation process creates a new database on the database server under the given user name. The database schema is created and UDDI data are loaded. This user should have the Database Creators server role.

> **Note:** Make sure your database server has case-sensitive collation, otherwise all comparisons will be case insensitive, even if the caseSensitiveMatch findQualifier is set. Alternatively, you can create a database with case-sensitive collation manually and use the **create schema** option.

> **Note:** If you selected the option **Create database** in the installation/Setup panel shown in figure Database Creation Method, you need a database user account with the `Database creators` server role. To create such account, you can use the `SQL Server Enterprise Manager`:

1. Select the **Console Root** > **Microsoft SQL Servers** > **SQL Server Group** > *server name* > **Security** > **Logins**.
2. Right-click on **Logins** and select the **New Login** from the context menu.
3. Enter the account name, click on the **SQL Server Authentication** option and fill in the password.
4. Select **Server Roles** tab, mark the **Database Creators**, click **OK**, and retype the password.



MSSQL

Installation creates a new database on the database server under the given database user account. So the user should have the Database Creators server role. Then, the database schema is created and UDDI data are loaded. For more information about the properties consult the documentation.

Properties marked with an asterisk (*) must not collide with existing objects in the database.

Database Server Address	localhost
Database Server Port	1433
Database Name *	uddinode
Database User	uddiuser
Database User Password	••••

MSSQL database creation requires the following properties. To connect or create schema requires a subset of these properties. Please note that properties marked with an asterisk (*) must not collide with existing objects in the database.

Database Server Address: Usually the host name or IP address where the database server is accessible.

Database Server Port: Port on which the database listens for a connection.

Database name *: Name of the database that will hold UDDI data structures.

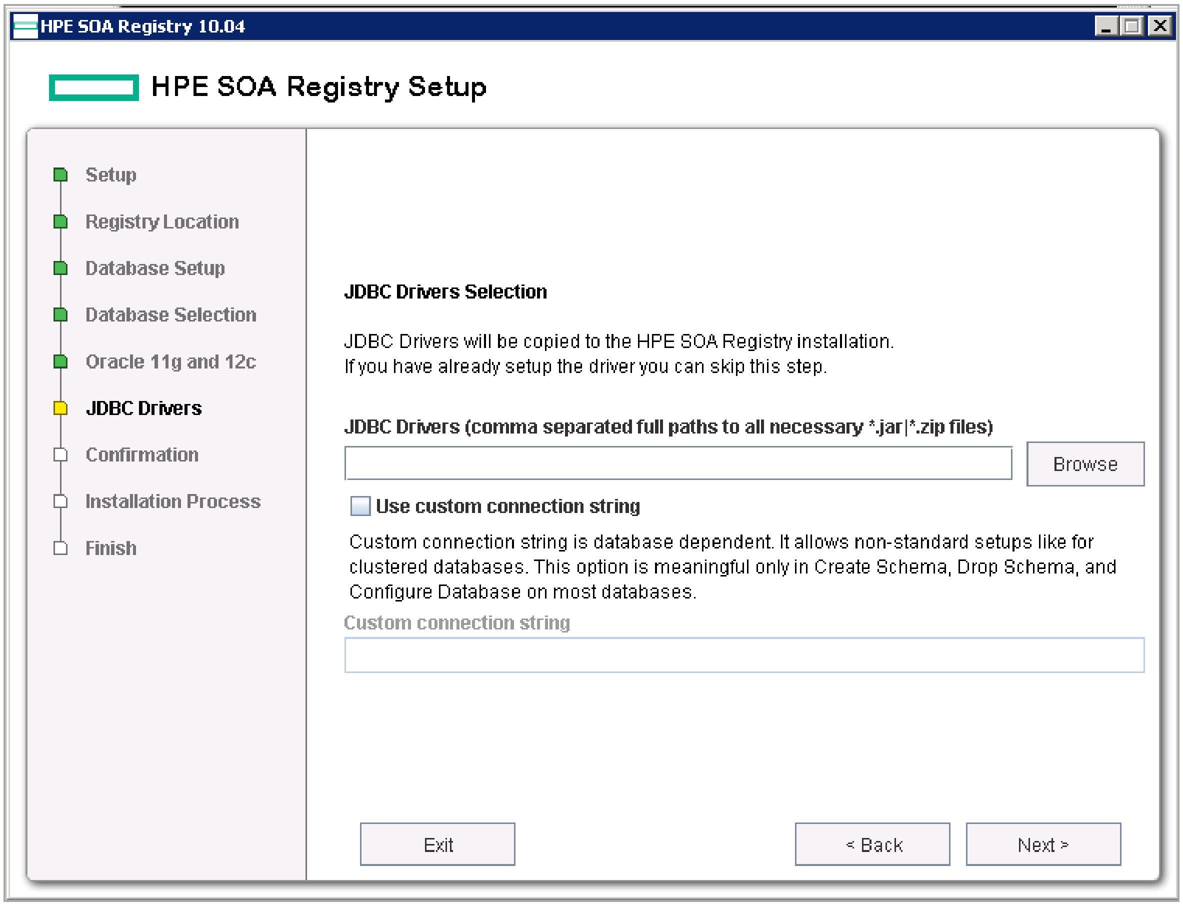
Database user: User name of a user who is able to create a new database.

Database User Password *: Password for the user specified above.

Continue with "JDBC Driver" below.

JDBC Driver

Select the JDBC Driver as shown in following figure, “Optional JDBC Driver”. It is not necessary to configure this path for the HSQL database as the JDBC drivers for this database are installed in the distribution. It is also not necessary if you have already configured this path previously for the selected database. The JDBC drivers are usually supplied by database vendors.



JDBC Drivers Selection

JDBC Drivers will be copied to the HPE SOA Registry installation.
If you have already setup the driver you can skip this step.

JDBC Drivers (comma separated full paths to all necessary *.jar|*.zip files)

☐ Use custom connection string

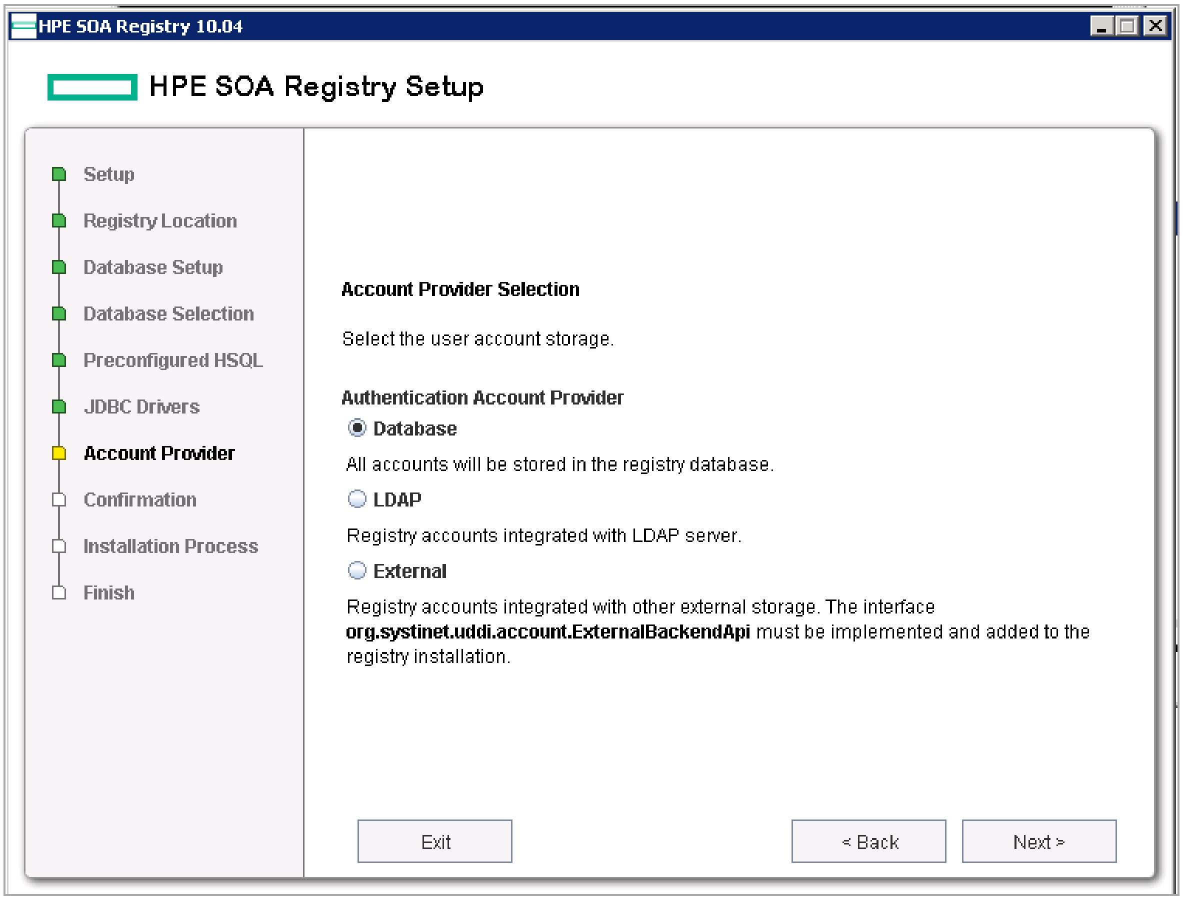
Custom connection string is database dependent. It allows non-standard setups like for clustered databases. This option is meaningful only in Create Schema, Drop Schema, and Configure Database on most databases.

Custom connection string

You can also specify custom JDBC connection string. Such string may be useful for special environments like database clusters where JDBC driver does load-balancing or failover. This setting is useful only in Create Schema, Drop Schema and Configure Database. We do not recommend to use this option unless there is special need to do so.

Account Backend

If you created a database or schema, you can configure an authentication account provider.



Account Provider Selection

Select the user account storage.

Authentication Account Provider

● Database

All accounts will be stored in the registry database.

○ LDAP

Registry accounts integrated with LDAP server.

○ External

Registry accounts integrated with other external storage. The interface **org.systinet.uddi.account.ExternalBackendApi** must be implemented and added to the registry installation.

Figure “Authentication Provider” allows you to select the authentication account provider.

Database: All accounts will be stored in the registry database. This is the recommended backend.

LDAP: Registry accounts integrated with LDAP server.

External: Registry accounts integrated with other external storage. To integrate HPE SOA Registry Foundation, with an external backend, you must implement the interface `com.systinet.uddi.account.ExternalBackendApi` and add it to the registry installation.

For more information about LDAP and External account backends, see "External Accounts Integration" on page 92.

Multilingual Data

This section describes how HPE SOA Registry Foundation supports the storage of UDDI structures in the multilingual data format.

There are two types of text fields in UDDI structures: Unicode fields and ASCII fields.

Unicode fields:intended for human readable information, the field length is measured in number of characters as follows**:**

Field Name	**Max Length (in chars)**
name of businessEntity and businessService	255
keyName	255
keyValue	255
useType	255
description	255
addressLine	80
personName	255

ASCII fields: intended for machine processing, such as URIs. The length is measured in bytes. ASCII fields can typically hold multilingual data. Its length is limited by the number of bytes of its serialized form in UTF-8 encoding. For example, the name of a tModel can carry 85 Japanese characters, because Japanese characters are encoded into three bytes each under UTF-8 encoding (255/3=85).

Field Name	**Max Length (in chars)**
name of tModel	255
overviewURL	4096
discoveryURL	4096
sortCode	10
email	255
phone	50
accessPoint	4096
instanceParms	8192

MSSQL

MSSQL supports Unicode characters only in Unicode fields. Unicode characters are stored successfully to ASCII fields only if they match with the server collation, otherwise are converted to question marks (?). For example, Japanese characters are stored correctly if the Japanese_Unicode_CI_AS collation is default to the server. If the English collation is set up, Japanese characters are converted to ? characters.

Oracle

Oracle database supports Unicode characters in both types (Unicode and ASCII) of fields.

JDBC Drivers

HPE SOA Registry Foundation requires by default the following classes for connection to the database. Please ensure that your downloaded JDBC JAR(s) includes them:

Database	Driver class
DB2	`com.ibm.db2.jcc.DB2Driver`
MSSQL	`com.microsoft.jdbc.sqlserver.SQLServerDriver`
Oracle	`oracle.jdbc.driver.OracleDriver`

Alternative JDBC Drivers

This section describes the use JDBC drivers other than the default drivers mentioned above. Suppose you downloaded `FooJDBC.jar`, where the driver class is `foo.jdbc.Driver` and the connection string is `jdbc:foo:....`

If you want to use an alternative JDBC driver while you already installed the registry and set up database with the default JDBC driver, edit the file `REGISTRY_HOME/app/uddi/conf/database.xml` as follows:

1. Add

   ```
   <universalDriver name="fooDriver">
   ```

```
<JDBC_driver>foo.jdbc.Driver</JDBC_driver>
<URI_pattern>jdbc:foo:...</URI_pattern>
</universalDriver>
```

at the end of <databaseMappings/> element

You can use following parameters in the <URI_pattern> element

- `${hostname}` - hostname or IP address of the database server
- `${port}` - Port where the database server listens for requests
- `${dbName}` - Name of the database
- `${userName}` - Name of database account
- `${userPassword}` - Password of the account

Replace the parameters with corresponding values using the Setup tool or the Registry Console.

2. Replace the className attribute of the interfaceMapping element with `fooDriver` value for your database. Determine the right `databaseMapping` element by value of `type` attribute.)

If you want to create a database with the alternative JDBC driver (without needing to use the default driver):

1. Install the HPE SOA Registry Foundation without the database.
2. Modify `REGISTRY_HOME/app/uddi/conf/database.xml` as described above.
3. Replace the driver class and connection string in the installation scripts in `REGISTRY_HOME/etc/db/ <database_type>/installXXX.xml`
4. Run the Setup tool to create database.

External Accounts Integration

During database installation or by employing the Setup tool, you may choose to use accounts from external repositories. This chapter describes how to integrate accounts from an LDAP server and from non-LDAP user stores into HPE SOA Registry Foundation.

An LDAP server can be integrated with HPE SOA Registry Foundation with these scenarios:

- LDAP with a single search base - The scenario is very simple. There is only one LDAP server in this scenario. All identities are stored under a single search base.

- LDAP with multiple search bases - In this scenario there is also only one LDAP server, but it has multiple search bases mapped to a domain. The domain is a specified part of the user's login name (that is, `DOMAIN/USERNAME`). All users must specify the domain name in the login dialog. When managing accounts or groups, we recommend using the `DOMAIN/USERNAME` format for performance reasons. If no domain is set, searches are performed across all domains.

Multiple LDAP services - More than one LDAP service is used in this scenario. The correct LDAP service is chosen via DNS. As in the previous scenario, users must specify a domain name during login. When managing accounts or groups, users have to set domain name. If the domain name is not specified, then no domain is processed.

This chapter also contains the following configuration examples:

- Sun One with a single search base
- Sun One with multiple search bases

Active Directory with a single search base

> **Note:** HPE SOA Registry Foundation treats external stores as read-only. User account properties stored in these external stores cannot be modified by HPE SOA Registry Foundation.

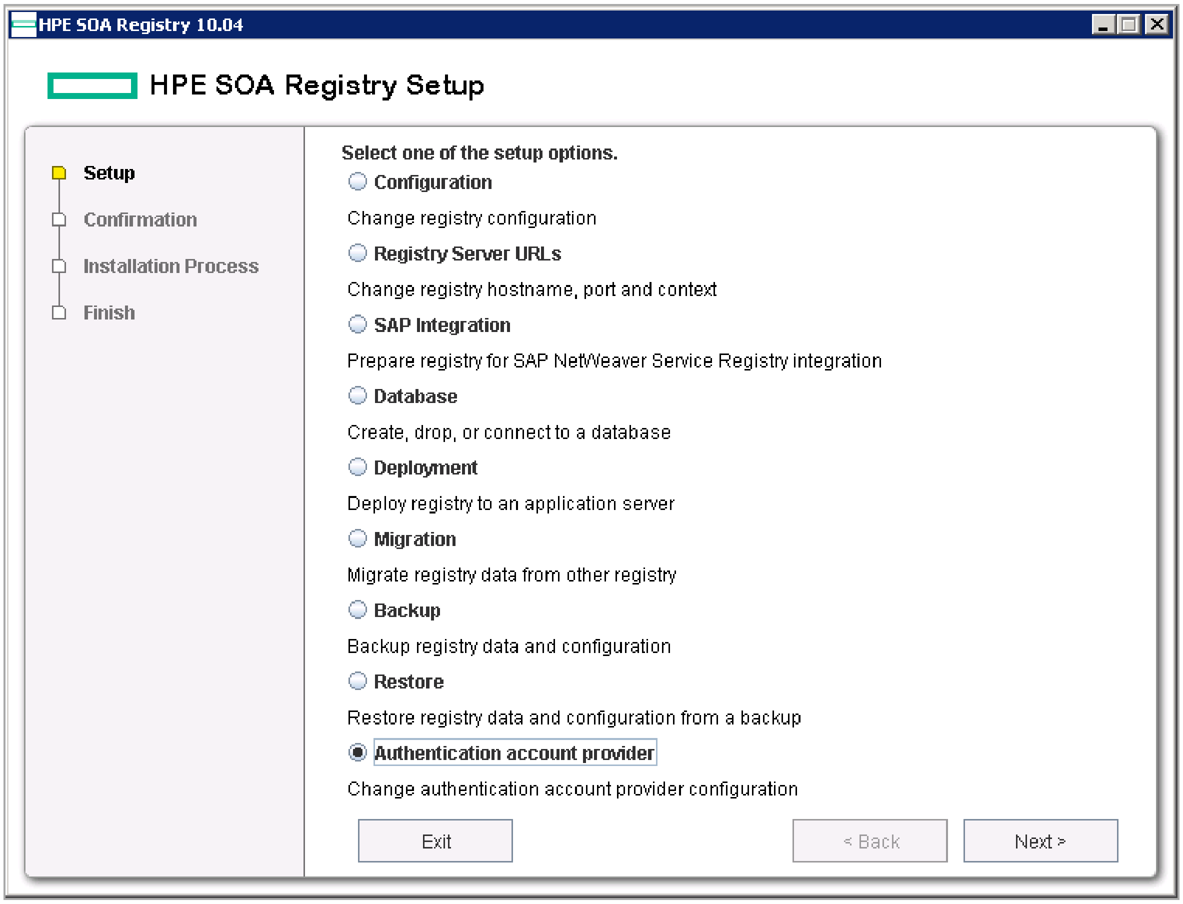
> **Note:** The Administrator account must not be stored in the LDAP. We strongly recommend that users stored in `account_list.xml` (by default, only administrator) should not be in the LDAP. If you really need to have users from LDAP in the file `account_list.xml`, delete password items from the file and change of all the accounts' properties according to the LDAP. The `account_list.xml` file contains a list of users that can be logged into a registry without connection to the database.
>
> Sometimes HPE SOA Registry Foundation displays various warnings into logs. We recommend to edit file `directory.xml` and file `group_core.xml` manually in order to suppress warnings related to account / group integration - LDAP (set true for attribute *suppressWarnings*).

To integrate external accounts from another repository, either:

- Create a database or create a new schema on the connected database by following the instructions in " Database Settings" on page 46, or
- Use the Setup tool and choose **Authentication provider**. To run the Setup tool, execute the following script from the `bin` subdirectory of your installation:

Windows:	setup.bat
UNIX:	./setup.sh

See command-line parameters in "Setup" in "Command Line Scripts" on page 59.



Select one of the setup options.

○ Configuration

Change registry configuration

○ Registry Server URLs

Change registry hostname, port and context

○ SAP Integration

Prepare registry for SAP NetWeaver Service Registry integration

○ Database

Create, drop, or connect to a database

○ Deployment

Deploy registry to an application server

○ Migration

Migrate registry data from other registry

○ Backup

Backup registry data and configuration

○ Restore

Restore registry data and configuration from a backup

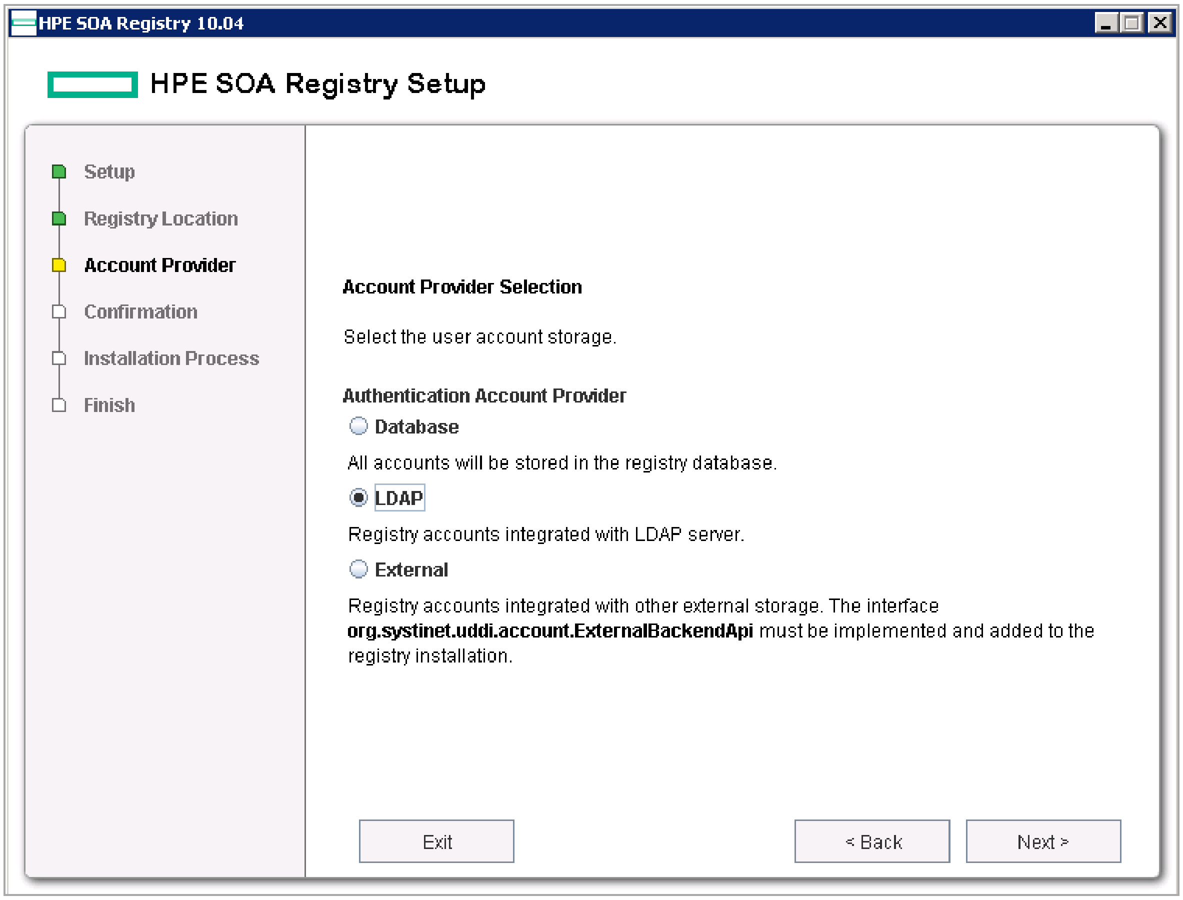
● Authentication account provider

Change authentication account provider configuration

For more information on the Setup tool, see "Reconfiguring After Installation" on page 62.

LDAP

Select **LDAP** on the **Account Provider** panel.



Account Provider Selection

Select the user account storage.

Authentication Account Provider

○ Database

All accounts will be stored in the registry database.

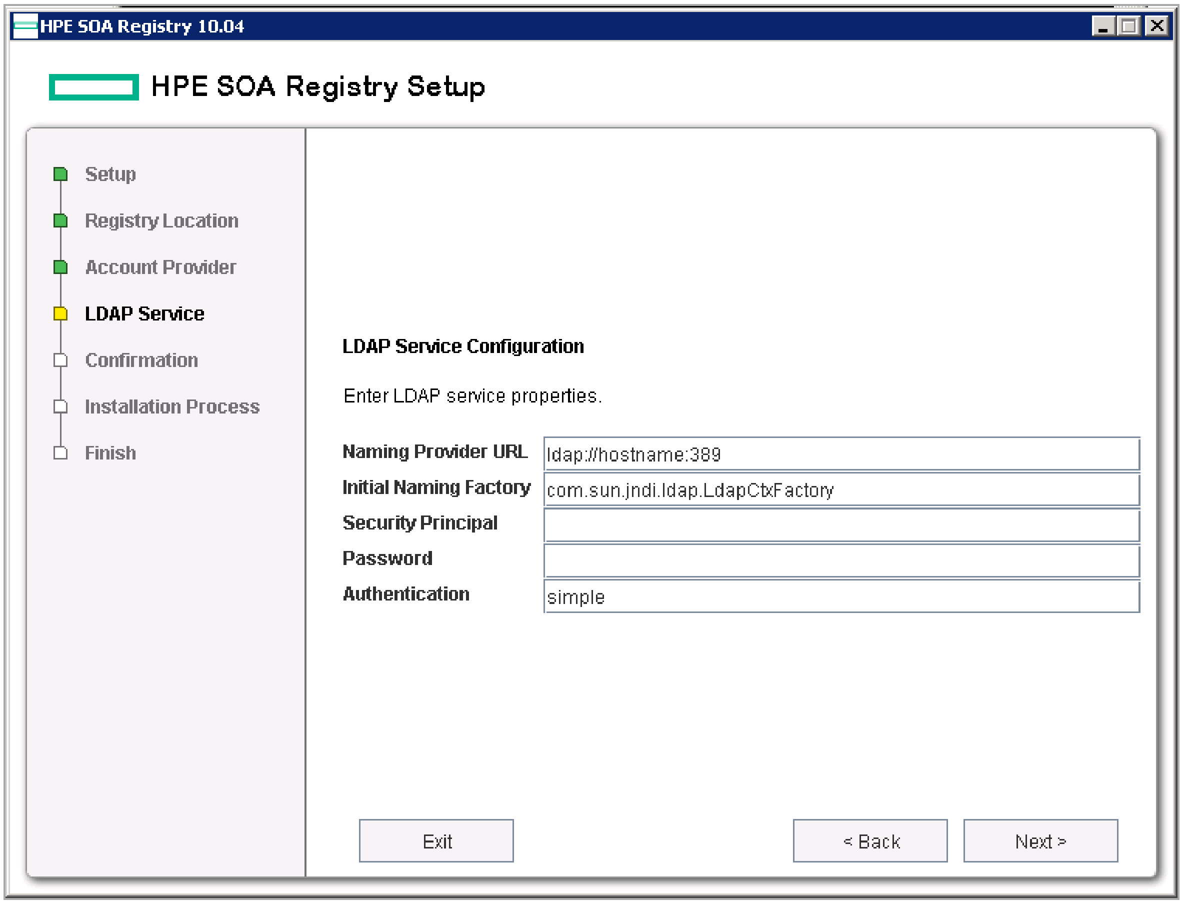
● LDAP

Registry accounts integrated with LDAP server.

○ External

Registry accounts integrated with other external storage. The interface **org.systinet.uddi.account.ExternalBackendApi** must be implemented and added to the registry installation.

Enter the following settings:



LDAP Service Configuration

Enter LDAP service properties.

Naming Provider URL: ldap://hostname:389
Initial Naming Factory: com.sun.jndi.ldap.LdapCtxFactory
Security Principal:
Password:
Authentication: simple

HPE SOA Registry Foundation uses a JNDI interface to connect to LDAP servers. The following JNDI properties must be known to the server. (The default properties are noted in parentheses.)

Java naming provider URL - A URL string for configuring the service provider specified by the "Java naming factory initial" property. (`ldap://hostname:389`).

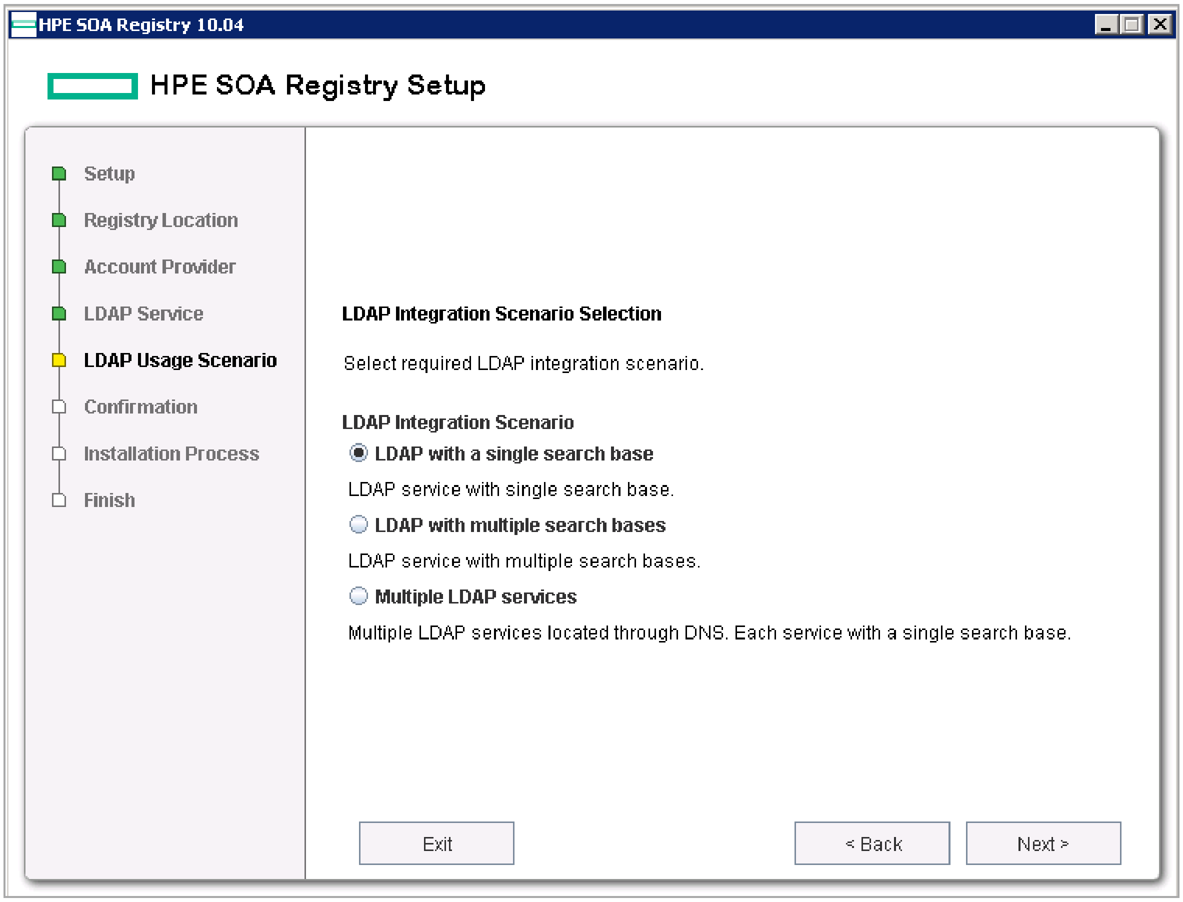
Initial Naming Factory - Class name of the initial naming factory. (`com.sun.jndi.ldap.LdapCtxFactory`).

Security Principal - The name of the principal for anonymous read access to the directory service.

Password - Password of security principal.

Authentication - Security level. (simple)

LDAP Usage Scenarios



LDAP Integration Scenario Selection

Select required LDAP integration scenario.

LDAP Integration Scenario

● **LDAP with a single search base**

LDAP service with single search base.

○ **LDAP with multiple search bases**

LDAP service with multiple search bases.

○ **Multiple LDAP services**

Multiple LDAP services located through DNS. Each service with a single search base.

You can select the following LDAP usage scenarios:

LDAP with a single search base - The scenario is very simple. There is only one LDAP server in this scenario. All identities are stored under a single search base.

LDAP with multiple search bases - In this scenario there is also only one LDAP server, but it has multiple search bases mapped to a domain. The domain is a specified part of user's login name (that is, `DOMAIN/USERNAME`). All users must specify the domain name in the login dialog. During the managing with accounts or groups it is recommended to use `DOMAIN/USERNAME` because of performance. If no domain is set then search is performed across all domains.

Domains can be specified dynamically or statically. For dynamic settings it is necessary to specify, for example, a domain prefix or postfix. Static domains are set during the installation directly and so they must be known in time of installation.

Multiple LDAP services - More than one LDAP service are used in this scenario. The correct LDAP service is chosen via DNS. As in the previous scenario, users must specify a domain name during login. When managing accounts or groups users have to set domain name. If domain name is not specified then no domain is processed.

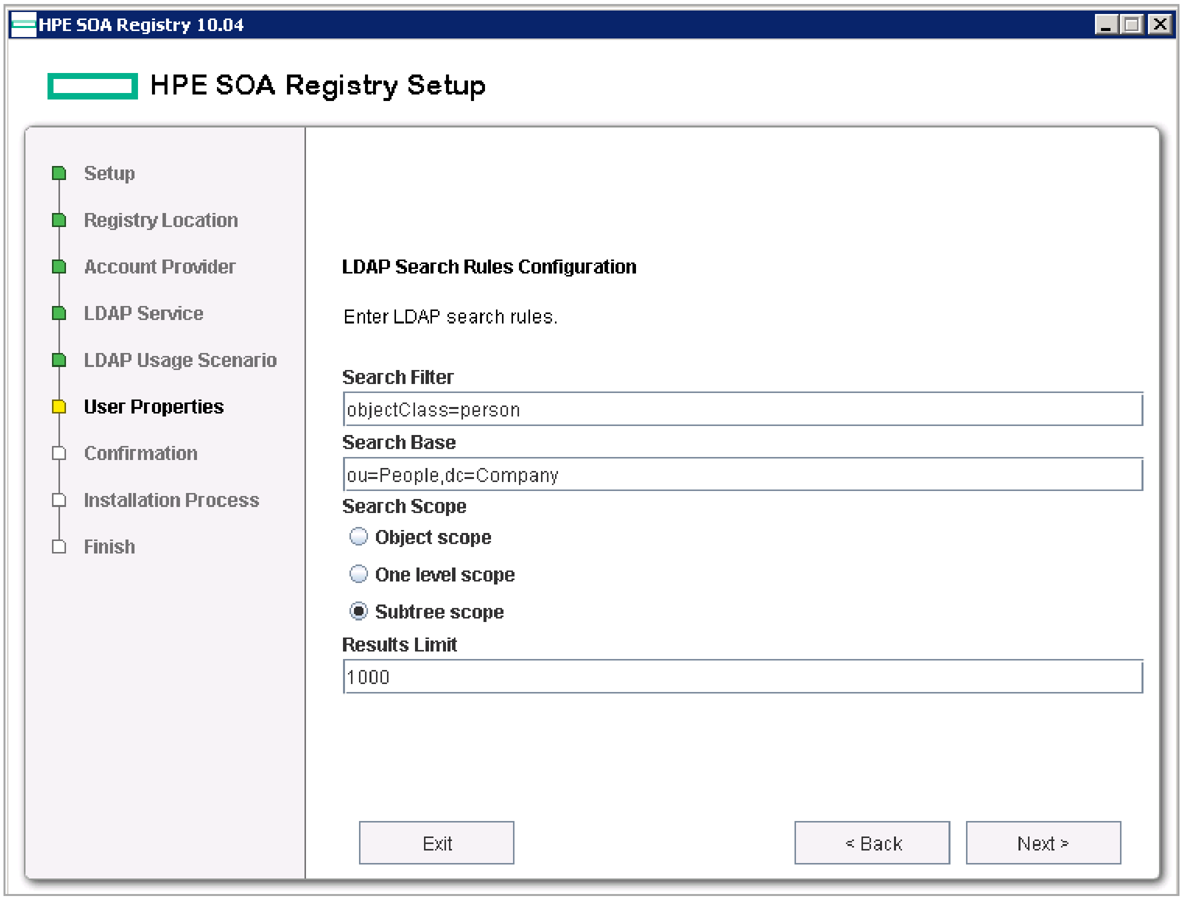
> **Note:** HPE SOA Registry Foundation treats external stores as read-only. User account properties stored in these external stores cannot be modified by HPE SOA Registry Foundation.
>
> The automatic discovery of LDAP servers allows you not to hardwire the URL and port of the LDAP server. For example, you can use `ldap:///o=JNDITutorial,dc=example,dc=com` as a URL and the real URL will be deduced from the distinguished name `o=JNDITutorial,dc=example,dc=com`.
>
> HPE SOA Registry Foundation integration with LDAP uses the JNDI API. For more information, see http://java.sun.com/products/jndi/tutorial/ldap/connect/create.html and http://docs.oracle.com/javase/jndi/tutorial/getStarted/overview/index.html

LDAP with a Single Search Base

The installation consists of the following steps:

1. Specify user/account search properties as shown in the “User Search Properties” figure below.
2. Map Registry user properties to LDAP properties as shown in Figure 39, “User Properties Mapping”.
3. Specify group search properties as shown in Figure 40, “Group Search Properties”.
4. Map Registry group properties to LDAP properties as shown in the “Group Properties Mapping” figure below.

User Search Properties



LDAP Search Rules Configuration

Enter LDAP search rules.

Search Filter: objectClass=person

Search Base: ou=People,dc=Company

Search Scope

○ Object scope

○ One level scope

● Subtree scope

Results Limit: 1000

Field description:

- **Search Filter -** The notation of the search filter conforms to the LDAP search notation. You can specify the LDAP node property that matches the user account.
- **Search Base -** LDAP will be searched from this base including the current LDAP node and all possible child nodes.
- **Search Scope -** Here you can specify how deep the LDAP tree structure's data will be searched.
  - *Object Scope* - Only the search base node will be searched.
  - *One-level Scope* - Only direct sub-nodes of the search base (entries one level below the search base) will be searched. The base entry is not included in the scope.
  - *Subtree Scope* - Search base and all its sub-nodes will be searched.
- **Results Limit** - Number of items returned when searching LDAP.

  If an LDAP search returns more results than the limit then the following warning is returned:

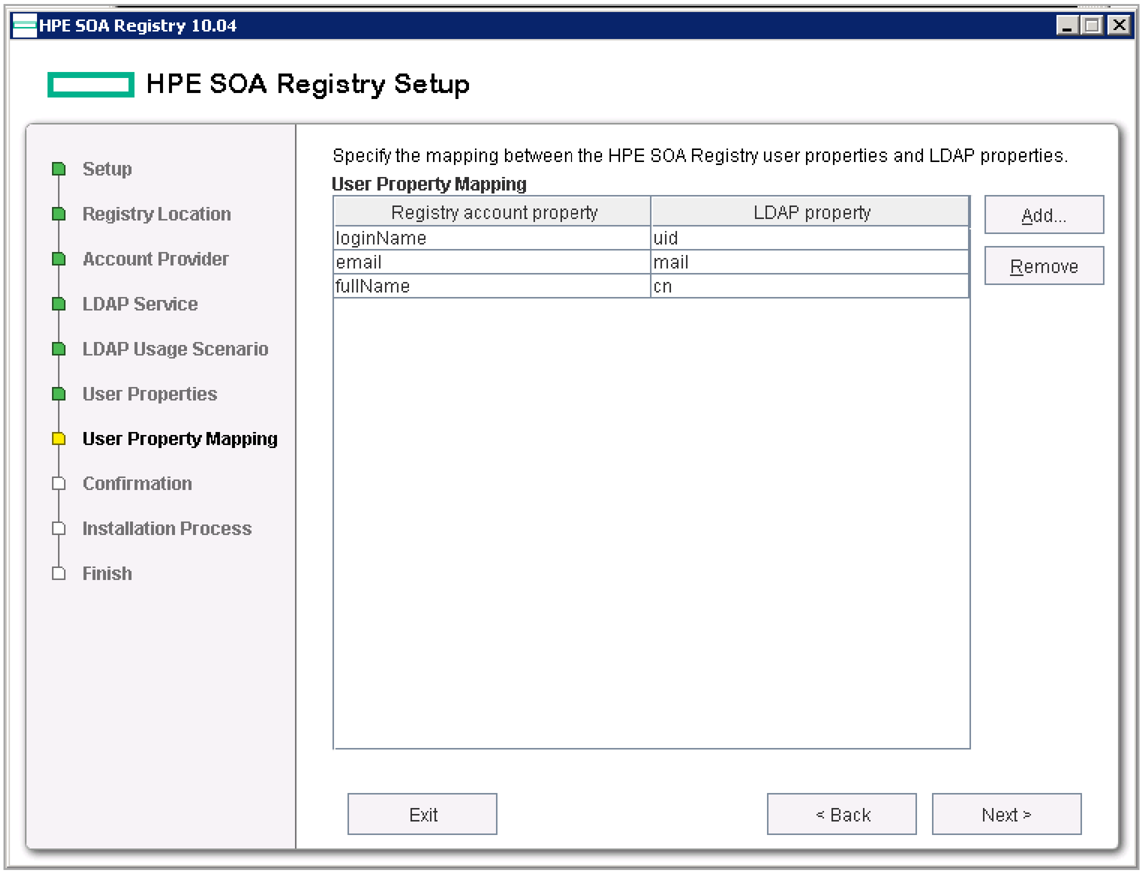
  ```
  WARN: ldap.LdapBackendImpl - The result of LDAP query (searchbase:
  'dc=in,dc=idoox,dc=com', filter:
  ```

```
'(&(uid=*)(objectClass=person))') is truncated by using the count limit search
control which is set to '100'.
```

```
 The query produced too many answers and so please narrow your search filter or
increase default limit count.
```

```
Read the documentation in order to suppress the warning.
```

User Property Mapping



Specify the mapping between the HPE SOA Registry user properties and LDAP properties.

User Property Mapping

Registry account property	LDAP property
loginName	uid
email	mail
fullName	cn

You can specify mapping between HPE SOA Registry Foundation user account properties and LDAP properties. You can add rows by clicking Add. To edit an entry, double click on the value you wish to edit.

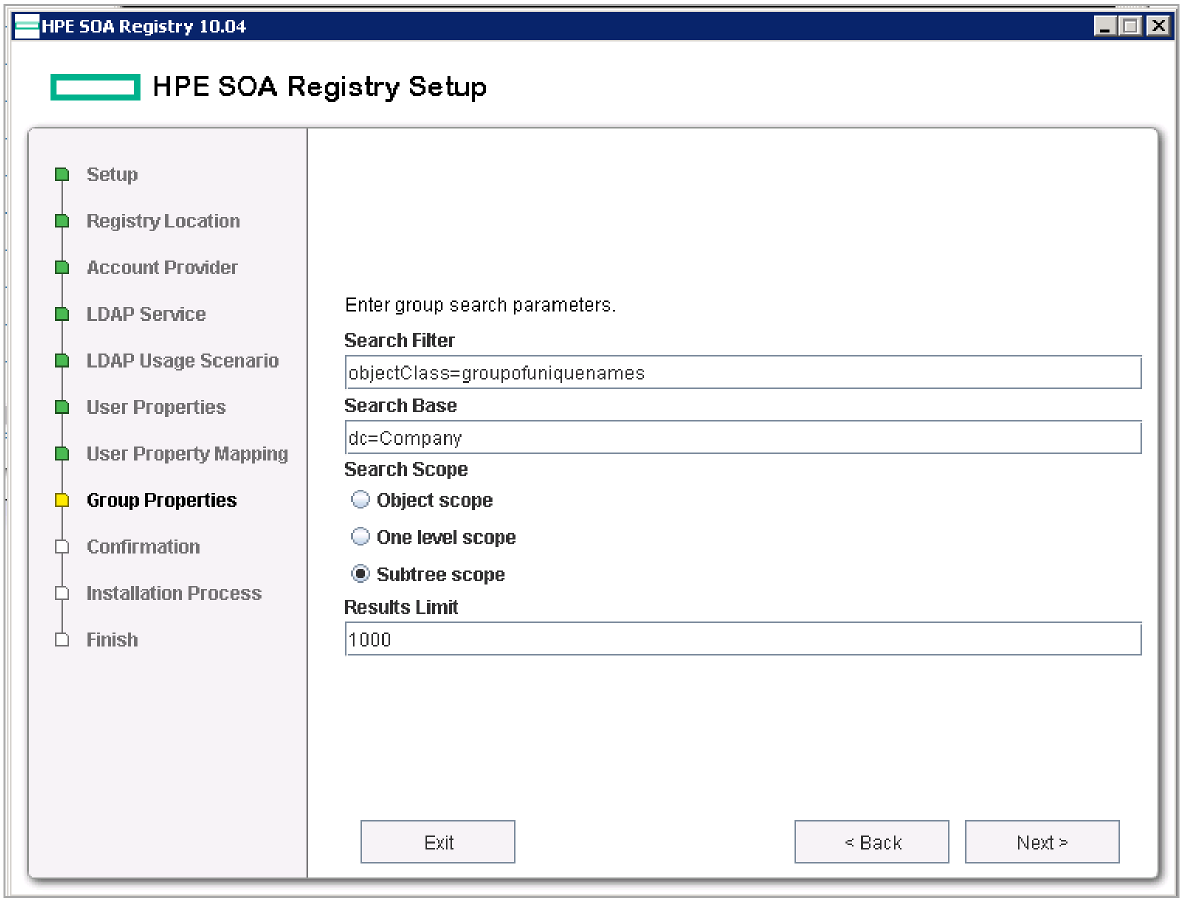
The following user account properties can be mapped from an LDAP server:

```
java.lang.String loginName
java.lang.String email
java.lang.String fullName
java.lang.String languageCode
java.lang.String password
java.lang.String description
java.lang.String businessName
java.lang.String phone
java.lang.String alternatePhone
java.lang.String address
java.lang.String city
java.lang.String stateProvince
java.lang.String country
java.lang.String zip
java.util.Date expiration
java.lang.Boolean expires
java.lang.Boolean external
java.lang.Boolean blocked
java.lang.Integer businessesLimit
java.lang.Integer servicesLimit
java.lang.Integer bindingsLimit
java.lang.Integer tModelsLimit
java.lang.Integer assertionsLimit
java.lang.Integer subscriptionsLimit
```

> **Note:** The Registry account property dn specifies the LDAP distinguished name. The value depends on the LDAP vendor.
>
> - On the Sun ONE Directory Server, the value is entryDN
> - On Microsoft Active Directory, the value is distinguishedName
>
>   If an optional property (such as email) does not exist in the LDAP, then the property's value is set according to the default account. The default account is specified in the config file whose name is account_core.xml.
>
>   User account properties that you specify in the “User Property Mapping” page (shown above) will be treated as read-only from the Registry Console and registry APIs.

For more information, see the "userAccount" data structure in "Accounts" on page 453 in the Developer's Guide.

Group Search Properties



Enter group search parameters.

Search Filter: objectClass=groupofuniquenames

Search Base: dc=Company

Search Scope

○ Object scope

○ One level scope

● Subtree scope

Results Limit: 1000

Field description:

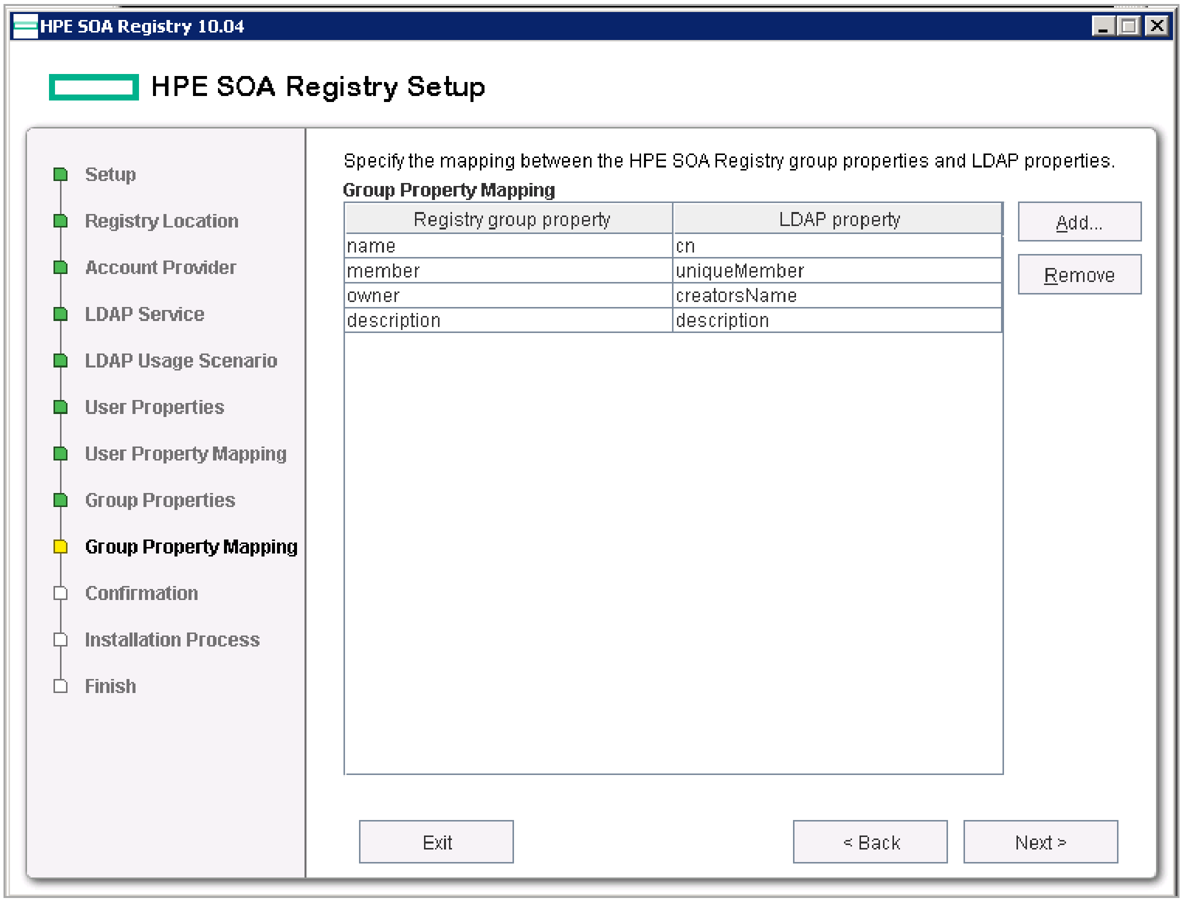
Search Filter - The notation of the search filter conforms to LDAP search notation. You can specify the LDAP node property that matches the group.

Search Base - LDAP, including the current LDAP node and possible all child nodes, will be searched from this base.

Search Scope - Here you can specify how deep the LDAP tree structure data will be searched.

- *Object Scope* - Only the search base node will be searched.
- *One-level Scope* - Search base and its direct sub-nodes will be searched.
- *Subtree Scope* - Search base and all its sub-nodes will be searched.

Group Property Mapping



Specify the mapping between the HPE SOA Registry group properties and LDAP properties.

Group Property Mapping

Registry group property	LDAP property
name	cn
member	uniqueMember
owner	creatorsName
description	description

You can specify mapping between HPE SOA Registry Foundation group properties and LDAP properties. You can add rows by clicking **Add**. To edit an entry, double click on the value you wish to edit.

If a property (such as description) does not exist in the LDAP then property value is set according to the default group. The default group (groupInfo) is specified in the config file whose name is `group.xml`.

For more information, please see *Developer's Guide*, group data structure.

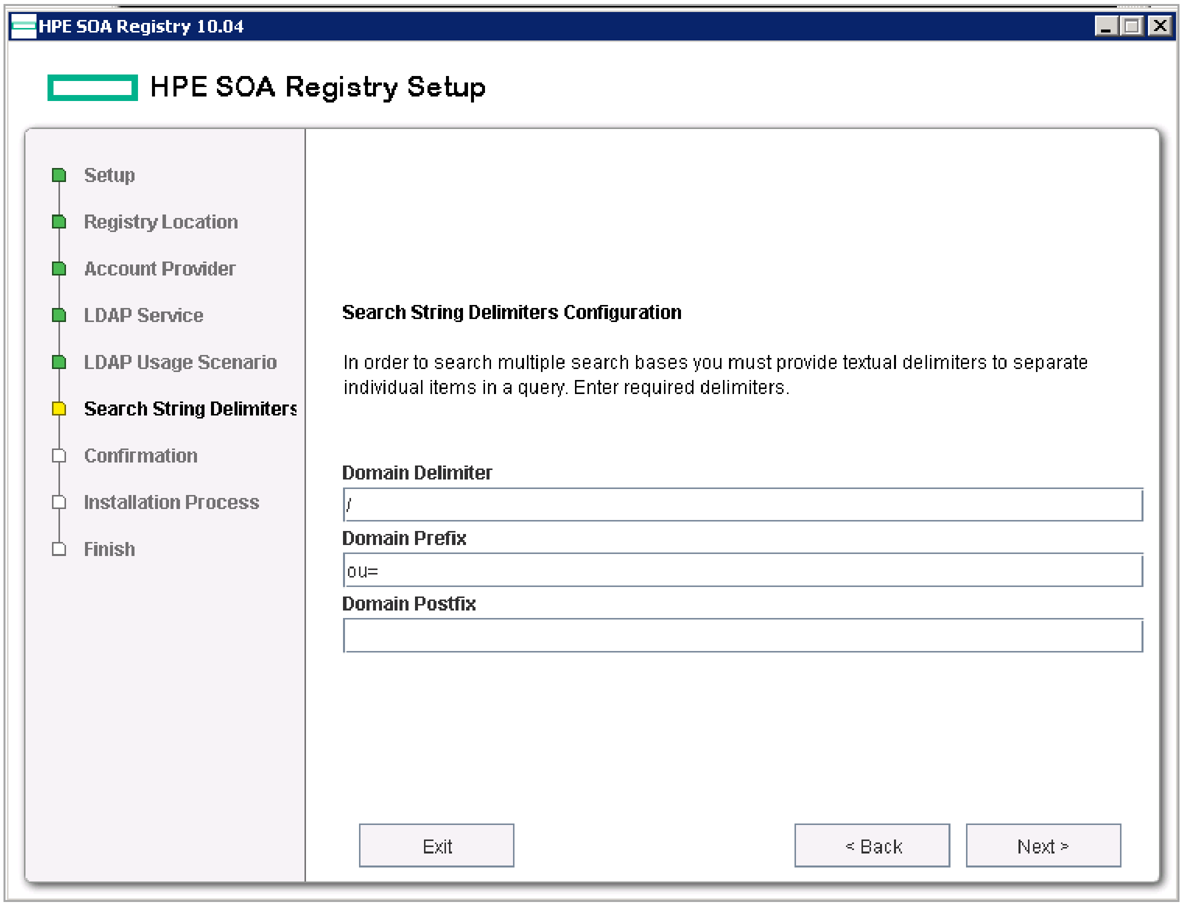
LDAP with Multiple Search Bases

The installation consists of the following steps:

1. Specify the domain delimiter, domain prefix and postfix as shown in figure "Domain Delimiter”.
2. Enable/Disable domains as shown in figure “Enable/Disable Domains”.
3. Specify User Search properties as shown in figure “User Search Properties”.

4. Map Registry user properties to LDAP properties as shown in figure “User Property Mapping”.
5. Specify group search properties as shown in figure “Group Search Properties”.
6. Map Registry group properties to LDAP properties as shown in figure “Group Property Mapping”

Domain Delimiter



Search String Delimiters Configuration

In order to search multiple search bases you must provide textual delimiters to separate individual items in a query. Enter required delimiters.

Domain Delimiter: /

Domain Prefix: ou=

Domain Postfix:

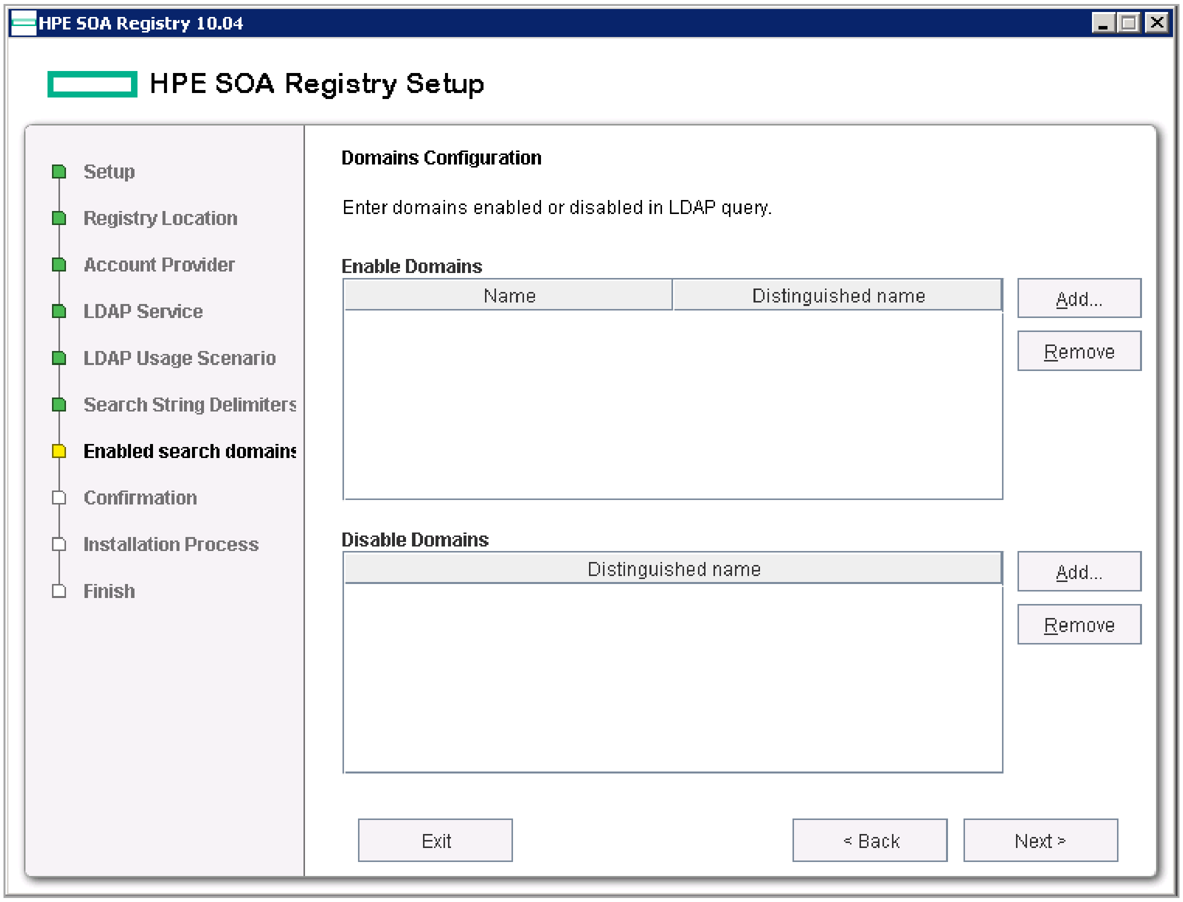
Field descriptions:

Domain Delimiter - Specifies the character that delimits domain and user name. When left empty, users are searched from all domains.

Domain Prefix, Domain Postfix - Domains are searched using the following pattern: `{domain prefix}domain_name{domain postfix}{search base}`

where {domain prefix} is the value of the property called domain prefix, {domain postfix} is the value of the property called domain postfix and {searchbase} is the value of the property called searchbase.

Enable/Disable Domains



Domains Configuration

Enter domains enabled or disabled in LDAP query.

Enable Domains

Name	Distinguished name

Disable Domains

Distinguished name

Enable Domains - Left column: domain name that users will be using during login. Right column: distinguished domain name.

Disable Domains - Enter distinguished domain name of domains you wish to disable.

Multiple LDAP Services

The correct LDAP service is chosen via DNS. The installation consists of the following steps:

1. Specify user/account search properties as shown in figure “User Search Properties”.
2. Map Registry user properties to LDAP properties as shown in figure “User Property Mapping”.
3. Specify group search properties as shown in figure “Group Search Properties”.
4. Map Registry group properties to LDAP properties as shown in figure “Group Property Mapping”.

LDAP Over SSL/TLS

It is only a matter of configuration to setup LDAP over SSL (or TLS) with a directory server of your choice. We recommend that you first install HPE SOA Registry Foundation with a connection to LDAP that does not use SSL. You can then verify the configuration by logging in as a user defined in this directory before configuring use of SSL.

The configuration procedure assumes that you have already installed HPE SOA Registry Foundation with an LDAP account provider. HPE SOA Registry Foundation must not be running.

LDAP over SSL Without Client Authentication

In this case only LDAP server authentication is required. This is usually the case.

Edit the REGISTRY_HOME/app/uddi/conf/directory.xml file in one of the following ways depending on the version of Java used to run HPE SOA Registry Foundation:

- If HPE SOA Registry Foundation will always be running with Java 1.4.2 or later:
    a. Change the java.naming.provider.url property to use the `ldaps` protocol and the port on which the directory server accepts SSL/TLS connections. For example ldaps://sranka.in.idoox.com:636;
- Otherwise, if HPE SOA Registry Foundation may be run with a Java version less than 1.4.2:
    a. Change the java.naming.provider.url property to the appropriate URL using the `ldap` protocol. For example `ldap://sranka.in.idoox.com:636;`
    b. Add a new property, after the *java.naming.provider.url* property, with name *java.naming.security.protocol* and value `ssl`;

This is shown in the following example:

Example 1. Directory configuration

```
<config name="directory" savingPeriod="5000">
<directory>
<!-- LDAP over (SSL/TLS) unprotected connection -->
<!--
<property name="java.naming.provider.url" value="ldap://hostname:47361"/>
-->
<!-- LDAP over SSL/TLS for Java 1.4.2 and later -->
```

```
<!--
<property name="java.naming.provider.url" value="ldaps://hostname:636"/>
-->
<!-- LDAP over SSL/TLS for Java where LDAP over SSL is supported -->
<property name="java.naming.provider.url" value="ldap://hostname:636"/>
<property name="java.naming.security.protocol" value="ssl"/>
...
...
...
</directory>
</config>
```

In both cases, be sure that the hostname specified in the *java.naming.provider.url* property matches the name that is in the directory server certificate's subject common name (CN part of certificate's Subject). Otherwise you will get an exception during startup of HPE SOA Registry Foundation. It will inform you of a hostname verification error. The stacktrace contains the hostname that you must use.

LDAP over SSL With Mutual Authentication

HPE SOA Registry Foundation can be configured to communicate with LDAP server over 2 way SSL. In this case HPE SOA Registry Foundation has to authenticates itself to LDAP server via client certificate.

To enable 2 way SSL communication with LDAP server:

- Specify the client certificate for HPE SOA Registry Foundation via Java system properties.

Property	Description
javax.net.ssl.keyStore	Absolute path to client keystore file. Keystore file must contain keyEntry that identifies the client.
javax.net.ssl.keyStorePassword	Password for the keystore file.

- Ensure trust to LDAP server. For more information see section bellow. Briefly:
  - Get the certificate of LDAP server or the certificate of its CA.
  - Import the certificate to keystore file via keytool.

- Add the following properties to Java system properties.

Property	Description
javax.net.ssl.trustStore	Absolute path of your trust store file.
javax.net.ssl.trustStorePassword	Password for the trust store file.

- Modify REGISTRY_HOME/app/uddi/conf/directory.xml
  - Change the java.naming.provider.url property to LDAPs URL or alternatively add java.naming.security.protocol (for Java version less 1.4.2). More details are described above.
  - Change the value of the java.naming.security.authentication from simple to EXTERNAL. In this case LDAP server does not use principal and his password so properties java.naming.security.principal and java.naming.security.credentials have no sense.

> **Note:** If LDAP server requires client authentication then it is necessary to set uddi.ldap.clientCertificateAuthentication to true. In this case HPE SOA Registry Foundation must be installed in two way SSL mode, in order to check client identity properly

Ensuring Trust of the LDAP Server

The client that connects to the SSL/TLS server must trust the server certificate in order to establish communication with that server. The configuration of LDAPS explained above inherits the default rule for establishing trust from JSSE (the Java implementation of SSL/TLS). This is based on trust stores.

When a trust store is needed to verify a client/server certificate, it is searched for in the following locations in order:

1. The file specified by the javax.net.ssl.trustStore system property, if defined;
2. Otherwise the file `JAVA_HOME\jre\lib\security\jssecacerts` if it exists;
3. Otherwise the file `JAVA_HOME\jre\lib\security\cacerts` if it exists;

It is recommended to use the first option to define a trust store specifically for the application you are running. In this case, you have to change the command that starts the registry (or the JVM environment of the deployed registry) to define the following Java system properties:

Property	Description
javax.net.ssl.trustStore	Absolute path of your trust store file.
javax.net.ssl.trustStorePassword	Password for the trust store file.

To ensure that the server certificate is trusted, you have to:

1. Contact the administrator of the LDAP server and get the certificate of the server or the certificate of the authority that signed it;
2. Import the certificate into the trust store of your choice using the Java keytool:

   ```
   keytool -import -trustcacerts -alias alias -file file -keystore keystore -
   storepass storepass
   ```

   where the parameters are as follows:

   alias

   A mandatory, unique alias for the certificate in the trust store;

   The file containing the certificate (usually with .crt extension);

   The keystore file of your choice;

   A password designed to protect the keystore file from tampering. Java level keystores (cacerts and jssecacerts) usually require the password `changeit`;

   file

   The file containing the certificate (usually with .crt extension);

   The keystore file of your choice;

   A password designed to protect the keystore file from tampering. Java level keystores (cacerts and jssecacerts) usually require the password `changeit`;

   keystore

   The keystore file of your choice;

   A password designed to protect the keystore file from tampering. Java level keystores (cacerts and jssecacerts) usually require the password `changeit`;

   storepass

   A password designed to protect the keystore file from tampering. Java level keystores (cacerts and jssecacerts) usually require the password `changeit`;

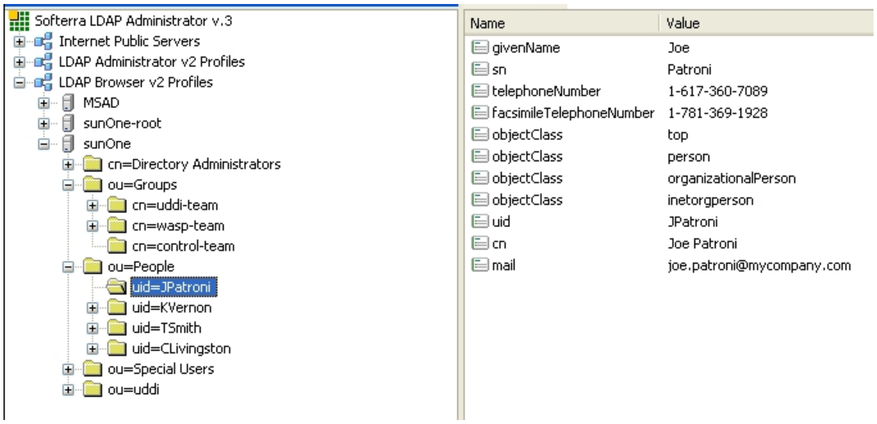
LDAP Configuration Examples

SUN One with Single Search Base

In this example, we show how to configure a Sun One Directory Server 5.2 under the LDAP Single Search Base scenario.

“SUN One with Single Search Base” shows user properties that are stored in the LDAP server.

User Properties in LDAP

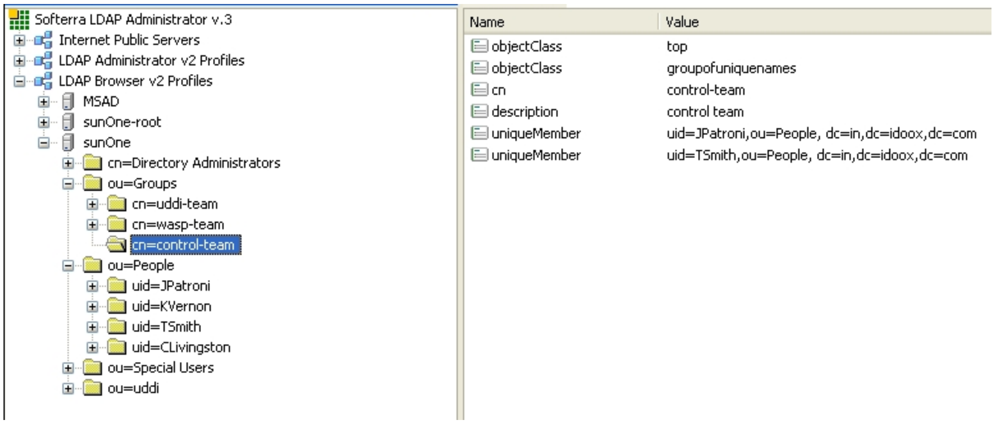


```
Softerra LDAP Administrator v.3
  Internet Public Servers
  LDAP Administrator v2 Profiles
  LDAP Browser v2 Profiles
    MSAD
    sunOne-root
    sunOne
      cn=Directory Administrators
      ou=Groups
        cn=uddi-team
        cn=wasp-team
        cn=control-team
      ou=People
        uid=JPatroni
        uid=KVernon
        uid=TSmith
        uid=CLivingston
      ou=Special Users
      ou=uddi
```

Name	Value
givenName	Joe
sn	Patroni
telephoneNumber	1-617-360-7089
facsimileTelephoneNumber	1-781-369-1928
objectClass	top
objectClass	person
objectClass	organizationalPerson
objectClass	inetorgperson
uid	JPatroni
cn	Joe Patroni
mail	joe.patroni@mycompany.com

“SUN One with Single Search Base” shows group properties that are stored in the LDAP server.

Group Properties in LDAP



```
Softerra LDAP Administrator v.3
  Internet Public Servers
  LDAP Administrator v2 Profiles
  LDAP Browser v2 Profiles
    MSAD
    sunOne-root
    sunOne
      cn=Directory Administrators
      ou=Groups
        cn=uddi-team
        cn=wasp-team
        cn=control-team
      ou=People
        uid=JPatroni
        uid=KVernon
        uid=TSmith
        uid=CLivingston
      ou=Special Users
      ou=uddi
```

Name	Value
objectClass	top
objectClass	groupofuniquenames
cn	control-team
description	control team
uniqueMember	uid=JPatroni,ou=People, dc=in,dc=idoox,dc=com
uniqueMember	uid=TSmith,ou=People, dc=in,dc=idoox,dc=com

The following table shows how to configure HPE SOA Registry Foundation using this scenario.

Config Property	Config Value	See
Java naming provider URL	ldap://localhost:389	Figure 36, “LDAP Service”
Initial Naming Factory	com.sun.jndi.ldap.LdapCtxFactory	Figure 36, “LDAP Service”
Security Principal	uid=JPatroni,ou=people,dc=in,dc=idoox,dc=com	Figure 36, “LDAP Service”
Security Protocol	simple	Figure 36, “LDAP Service”

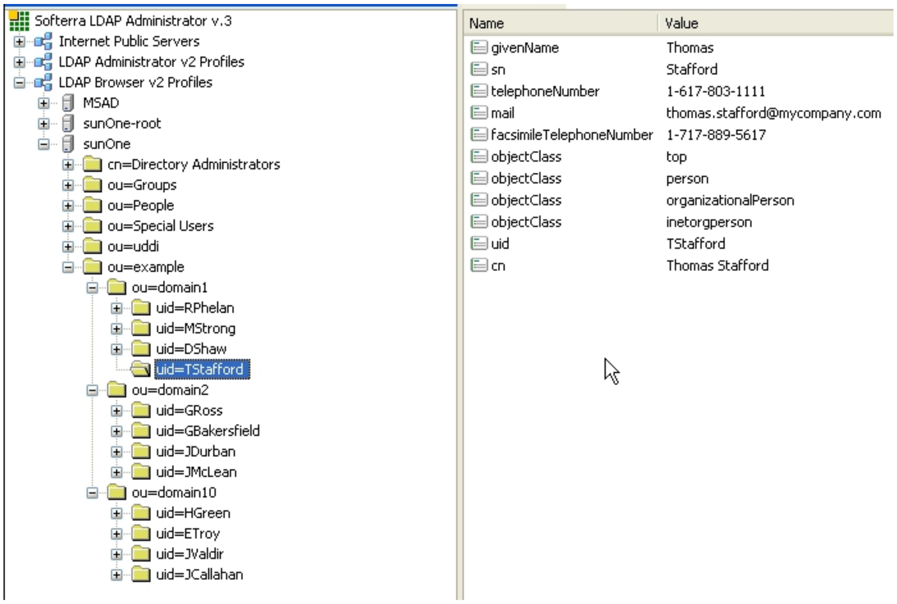
User Properties		
Search Filter	objectClass=person	Figure 38, “User Search Properties”
Search Base	ou=people,dc=in,dc=idoox,dc=com	Figure 38, “User Search Properties”
Search Scope	Subtree Scope	Figure 38, “User Search Properties”
Result Limit	100	Figure 38, “User Search Properties”
telephoneNumber	phone	Figure 39, “User Properties Mapping”
uid	loginName	Figure 39, “User Properties Mapping”
cn	fullName	Figure 39, “User Properties Mapping”
mail	email	Figure 39, “User Properties Mapping”
Group Properties		
Search Filter	objectClass=groupofuniquenames	Figure 40, “Group Search Properties”
Search Base	ou=groups,dc=in,dc=idoox,dc=com	Figure 40, “Group Search Properties”
Search Scope	Subtree Scope	Figure 40, “Group Search Properties”
Result Limit	100	Figure 40, “Group Search Properties”
creatorsName	owner	Figure 41, “Group Properties Mapping”
description	description	Figure 41, “Group Properties Mapping”
uniqueMember	member	Figure 41, “Group Properties Mapping”
cn	name	Figure 41, “Group Properties Mapping”

Sun One with Multiple Search Bases

In this example, we show how to configure Sun One Directory Server 5.2 with multiple search bases. In Figure 47, “Registry Users”, you can see users and domains that are stored on the LDAP server. We want to configure the LDAP integration with HPE SOA Registry Foundation in this way:

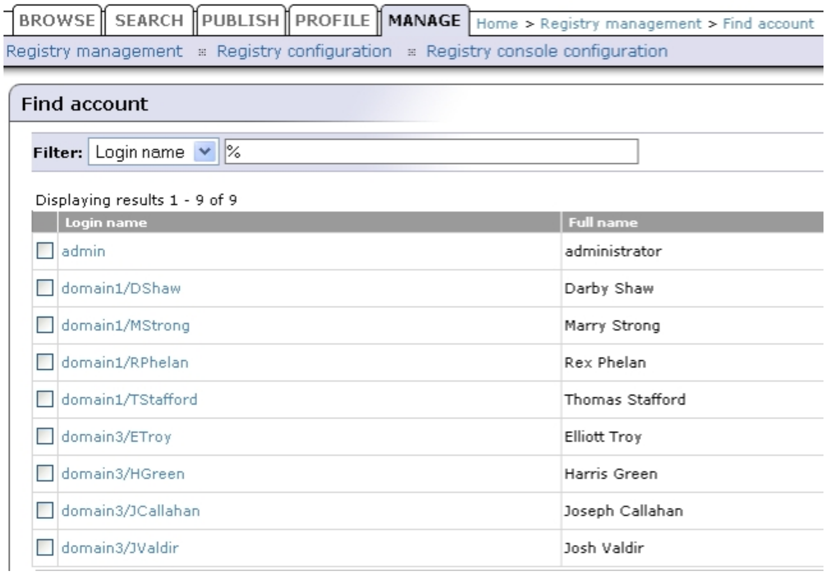
- Only users from `domain1` and `domain10` can log into HPE SOA Registry Foundation. LDAP `domain2` will be disabled.
- LDAP `domain10` will be mapped to the `domain3` user group in HPE SOA Registry Foundation.

Figure 47, “Registry Users” shows how users from LDAP are mapped to HPE SOA Registry Foundation



- Softerra LDAP Administrator v.3
  - Internet Public Servers
  - LDAP Administrator v2 Profiles
  - LDAP Browser v2 Profiles
    - MSAD
    - sunOne-root
    - sunOne
      - cn=Directory Administrators
      - ou=Groups
      - ou=People
      - ou=Special Users
      - ou=uddi
      - ou=example
        - ou=domain1
          - uid=RPhelan
          - uid=MStrong
          - uid=DShaw
          - uid=TStafford
        - ou=domain2
          - uid=GRoss
          - uid=GBakersfield
          - uid=JDurban
          - uid=JMcLean
        - ou=domain10
          - uid=HGreen
          - uid=ETroy
          - uid=JValdir
          - uid=JCallahan

Name	Value
givenName	Thomas
sn	Stafford
telephoneNumber	1-617-803-1111
mail	thomas.stafford@mycompany.com
facsimileTelephoneNumber	1-717-889-5617
objectClass	top
objectClass	person
objectClass	organizationalPerson
objectClass	inetorgperson
uid	TStafford
cn	Thomas Stafford



Find account

Filter: Login name %

Displaying results 1 - 9 of 9

Login name	Full name
□ admin	administrator
□ domain1/DShaw	Darby Shaw
□ domain1/MStrong	Marry Strong
□ domain1/RPhelan	Rex Phelan
□ domain1/TStafford	Thomas Stafford
□ domain3/ETroy	Elliott Troy
□ domain3/HGreen	Harris Green
□ domain3/JCallahan	Joseph Callahan
□ domain3/JValdir	Josh Valdir

The following table shows how to configure HPE SOA Registry Foundation using this scenario.

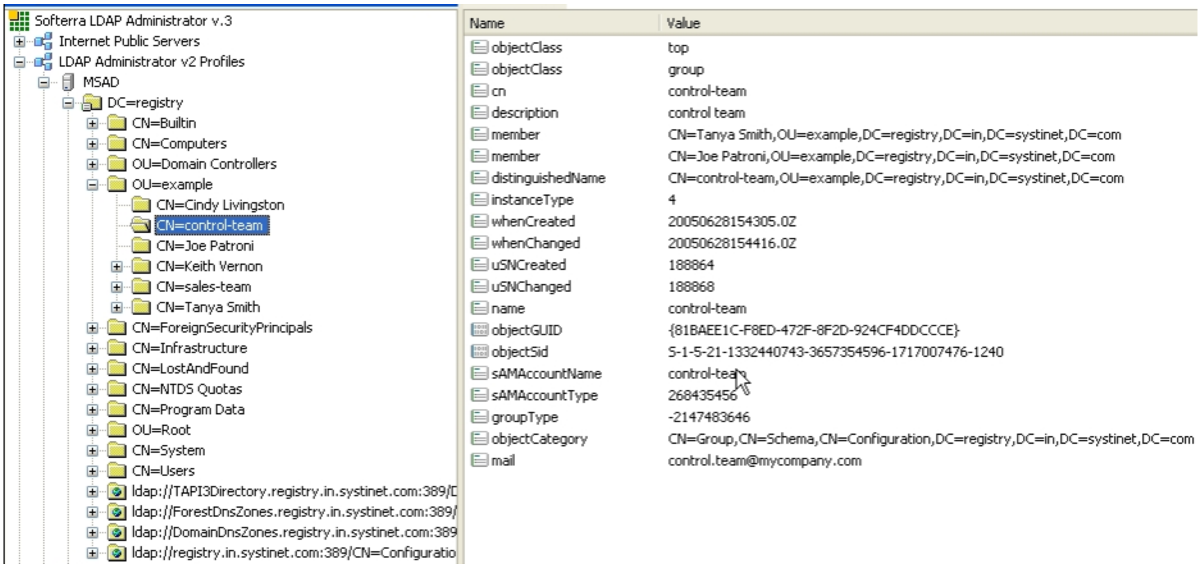
Config Property	Config Value	See
Java naming provider URL	ldap://localhost:1000	Figure 36, “LDAP Service”
Initial Naming Factory	com.sun.jndi.ldap.LdapCtxFactory	Figure 36, “LDAP Service”
Security Principal	uid=JPatroni,ou=people,dc=in,dc=idoox,dc=com	Figure 36, “LDAP Service”
Security Protocol	simple	Figure 36, “LDAP Service”
uddi.ldap.domain.delimiter	/	Figure 42, “Domain Delimiter”
uddi.ldap.domain.prefix	ou=	Figure 42, “Domain Delimiter”
uddi.ldap.domain.postfix	leave empty	Figure 42, “Domain Delimiter”
Enable domains		
domain name	domain3	Figure 43, “Enable/Disable Domains”
Distinguished name	ou=domain10,ou=example,dc=in,dc=idoox,dc=com	Figure 43, “Enable/Disable Domains”
Disable domains		
Distinguished name	ou=domain2,ou=example,dc=in,dc=idoox,dc=com	Figure 43, “Enable/Disable Domains”
User Properties		
Search Filter	ou=people,dc=in,dc=idoox,dc=com	Figure 38, “User Search Properties”
Search Scope	Subtree Scope	Figure 38, “User Search Properties”
Result Limit	100	Figure 38, “User

		Search Properties”
telephoneNumber	phone	Figure 39, “User Properties Mapping”
uid	loginName	Figure 39, “User Properties Mapping”
cn	fullName	Figure 39, “User Properties Mapping”
mail	email	Figure 39, “User Properties Mapping”
Group Properties		
Search Filter	objectClass=groupofuniquenames	Figure 40, “Group Search Properties”
Search Base	ou=groups,dc=in,dc=idoox,dc=com	Figure 40, “Group Search Properties”
Search Scope	Subtree Scope	Figure 40, “Group Search Properties”
Result Limit	100	Figure 40, “Group Search Properties”
creatorsName	owner	Figure 41, “Group Properties Mapping”
description	description	Figure 41, “Group Properties Mapping”
uniqueMember	member	Figure 41, “Group Properties Mapping”

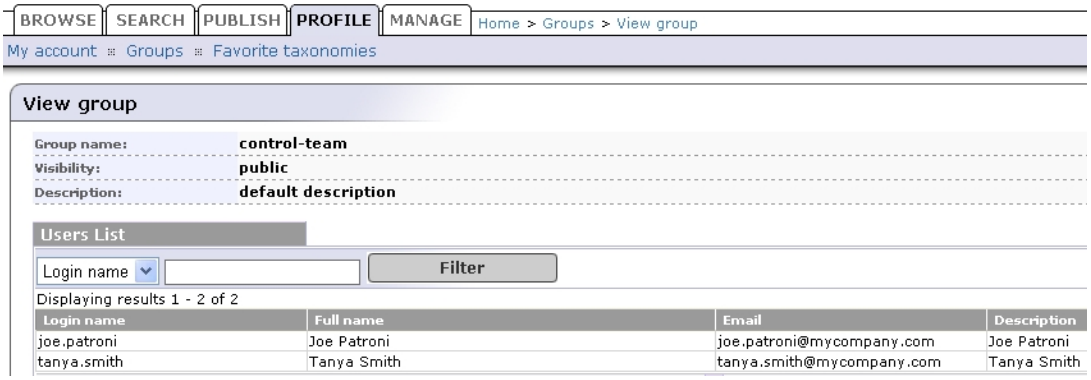
cn	name	Figure 41, “Group Properties Mapping”

Active Directory with Single Search Base

In this example, we show how to configure an Active Directory with a single search base. Figure 48, “LDAP User Group” shows group properties that are stored in the Active Directory. These group properties will be mapped to HPE SOA Registry Foundation as shown in Figure 49, “User Group in HPE SOA Registry Foundation”.


Softerra LDAP Administrator v.3
Internet Public Servers
LDAP Administrator v2 Profiles
MSAD
DC=registry
CN=Builtin
CN=Computers
OU=Domain Controllers
OU=example
CN=Cindy Livingston
CN=control-team
CN=Joe Patroni
CN=Keith Vernon
CN=sales-team
CN=Tanya Smith
CN=ForeignSecurityPrincipals
CN=Infrastructure
CN=LostAndFound
CN=NTDS Quotas
CN=Program Data
OU=Root
CN=System
CN=Users
ldap://TAPI3Directory.registry.in.systinet.com:389/D
ldap://ForestDnsZones.registry.in.systinet.com:389/
ldap://DomainDnsZones.registry.in.systinet.com:389
ldap://registry.in.systinet.com:389/CN=Configuratio

Name	Value
objectClass	top
objectClass	group
cn	control-team
description	control team
member	CN=Tanya Smith,OU=example,DC=registry,DC=in,DC=systinet,DC=com
member	CN=Joe Patroni,OU=example,DC=registry,DC=in,DC=systinet,DC=com
distinguishedName	CN=control-team,OU=example,DC=registry,DC=in,DC=systinet,DC=com
instanceType	4
whenCreated	20050628154305.0Z
whenChanged	20050628154416.0Z
uSNCreated	188864
uSNChanged	188868
name	control-team
objectGUID	{81BAEE1C-F8ED-472F-8F2D-924CF4DDCCCE}
objectSid	S-1-5-21-1332440743-3657354596-1717007476-1240
sAMAccountName	control-team
sAMAccountType	268435456
groupType	-2147483646
objectCategory	CN=Group,CN=Schema,CN=Configuration,DC=registry,DC=in,DC=systinet,DC=com
mail	control.team@mycompany.com



View group

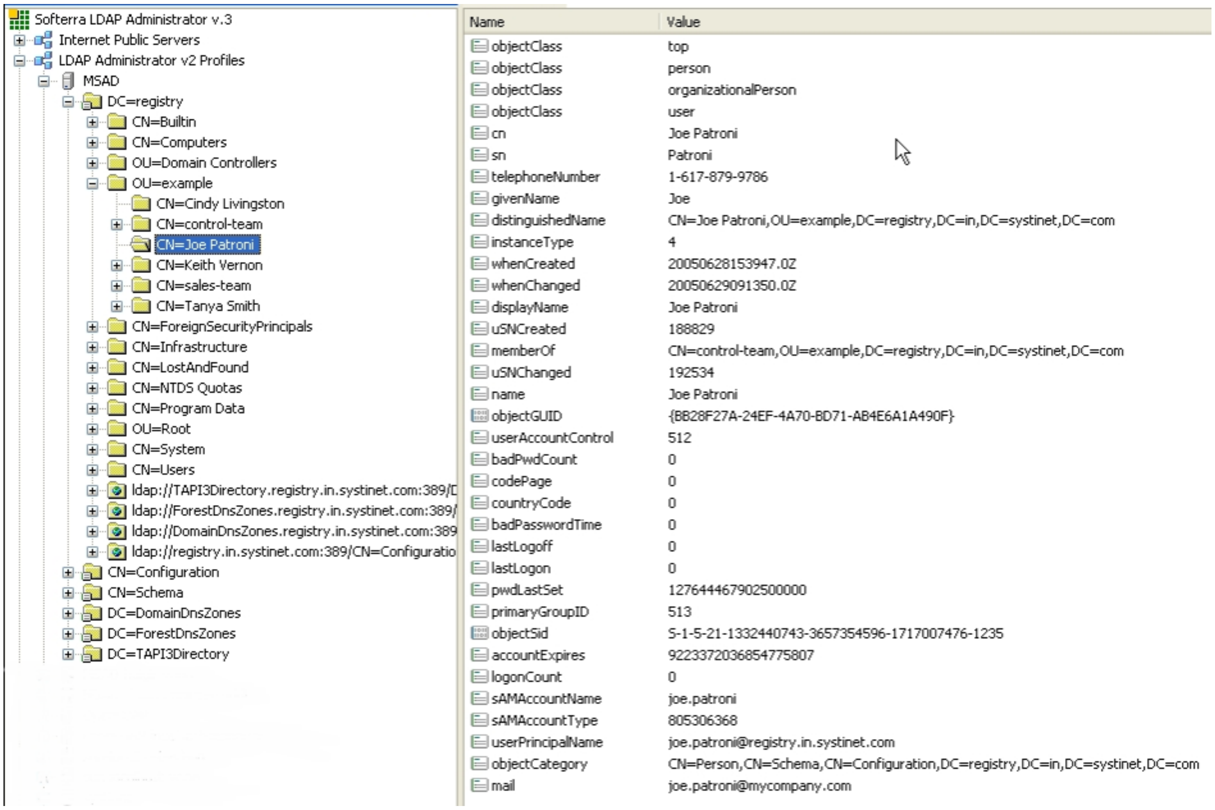
Group name:	control-team
Visibility:	public
Description:	default description

Users List

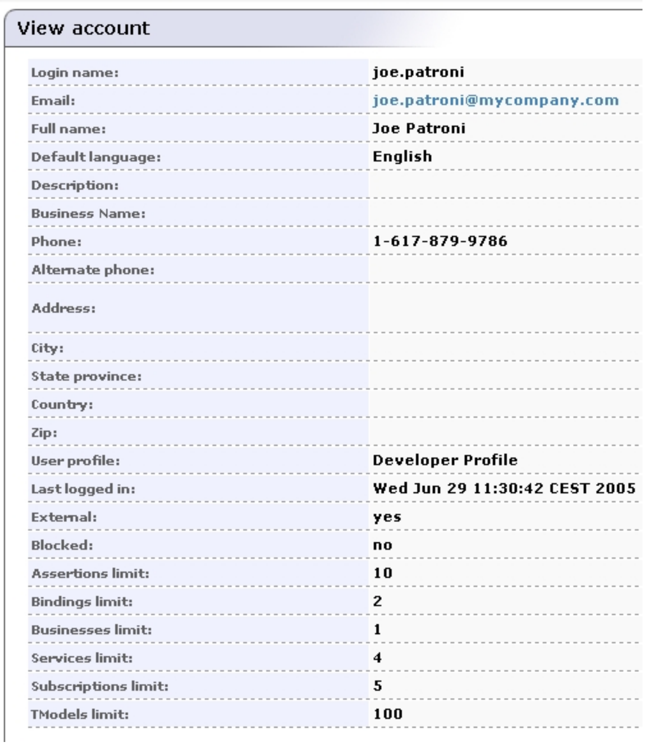
Displaying results 1 - 2 of 2

Login name	Full name	Email	Description
joe.patroni	Joe Patroni	joe.patroni@mycompany.com	Joe Patroni
tanya.smith	Tanya Smith	tanya.smith@mycompany.com	Tanya Smith

Figure 50, “LDAP User Properties” shows user properties that are stored in the Active Directory. These user properties will be mapped to HPE SOA Registry Foundation as shown in Figure 49, “User Group in HPE SOA Registry Foundation”.



Name	Value
objectClass	top
objectClass	person
objectClass	organizationalPerson
objectClass	user
cn	Joe Patroni
sn	Patroni
telephoneNumber	1-617-879-9786
givenName	Joe
distinguishedName	CN=Joe Patroni,OU=example,DC=registry,DC=in,DC=systinet,DC=com
instanceType	4
whenCreated	20050628153947.0Z
whenChanged	20050629091350.0Z
displayName	Joe Patroni
uSNCreated	188829
memberOf	CN=control-team,OU=example,DC=registry,DC=in,DC=systinet,DC=com
uSNChanged	192534
name	Joe Patroni
objectGUID	{BB28F27A-24EF-4A70-BD71-AB4E6A1A490F}
userAccountControl	512
badPwdCount	0
codePage	0
countryCode	0
badPasswordTime	0
lastLogoff	0
lastLogon	0
pwdLastSet	127644467902500000
primaryGroupID	513
objectSid	S-1-5-21-1332440743-3657354596-1717007476-1235
accountExpires	9223372036854775807
logonCount	0
sAMAccountName	joe.patroni
sAMAccountType	805306368
userPrincipalName	joe.patroni@registry.in.systinet.com
objectCategory	CN=Person,CN=Schema,CN=Configuration,DC=registry,DC=in,DC=systinet,DC=com
mail	joe.patroni@mycompany.com


View account

Login name:	joe.patroni
Email:	joe.patroni@mycompany.com
Full name:	Joe Patroni
Default language:	English
Description:	
Business Name:	
Phone:	1-617-879-9786
Alternate phone:	
Address:	
City:	
State province:	
Country:	
Zip:	
User profile:	Developer Profile
Last logged in:	Wed Jun 29 11:30:42 CEST 2005
External:	yes
Blocked:	no
Assertions limit:	10
Bindings limit:	2
Businesses limit:	1
Services limit:	4
Subscriptions limit:	5
TModels limit:	100

The following table shows how to configure HPE SOA Registry Foundation using this scenario.

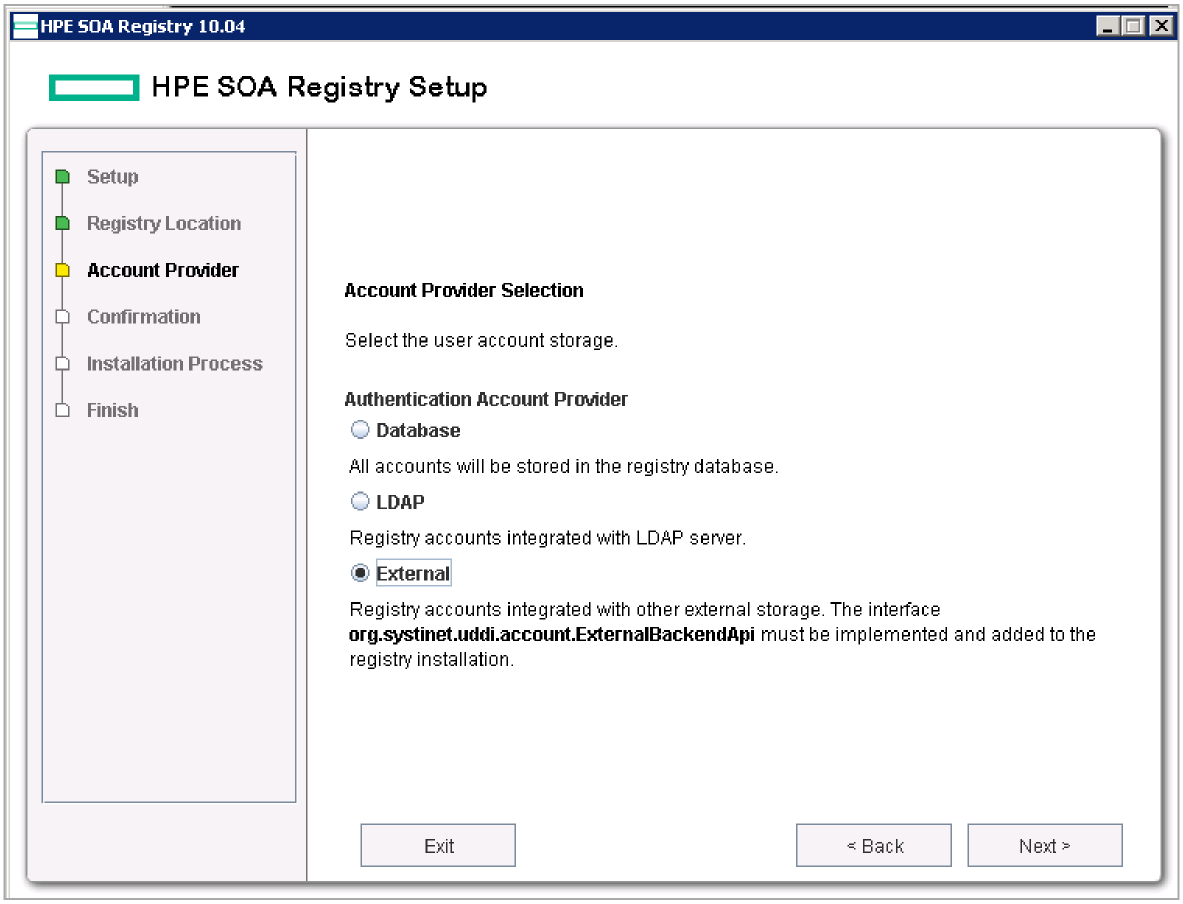
Config Property	Config Value	See
Java naming provider URL	ldap://localhost:389	Figure 36, “LDAP Service”
Initial Naming Factory	com.sun.jndi.ldap.LdapCtxFactory	Figure 36, “LDAP Service”
Security Principal	CN=userx,OU=root,DC=registry,DC=in,DC=mycompany,DC=com	Figure 36, “LDAP Service”
Security Protocol	DIGEST-MD5	Figure 36, “LDAP Service”
User Properties		
Search Filter	objectClass=person	Figure 38, “User Search Properties”
Search Base	ou=example,dc=registry,dc=in,dc=mycompany,dc=com	Figure 38, “User Search Properties”
Search Scope	Subtree Scope	Figure 38, “User Search Properties”
Result Limit	100	Figure 38, “User Search Properties”
sAMAccountName	loginName	Figure 39, “User Properties Mapping”
cn	fullName	Figure 39, “User Properties

		Mapping”
mail	email	Figure 39, “User Properties Mapping”
telephoneNumber	phone	Figure 39, “User Properties Mapping”
Group Properties		
Search Filter	objectClass=group	Figure 40, “Group Search Properties”
Search Base	ou=example,dc=registry,dc=in,dc=mycompany,dc=com	Figure 40, “Group Search Properties”
Search Scope	Subtree Scope	Figure 40, “Group Search Properties”
Result Limit	100	Figure 40, “Group Search Properties”
member	member	Figure 41, “Group Properties Mapping”
cn	name	Figure 41, “Group Properties Mapping”
uniqueMember	member	Figure 41, “Group

		Properties Mapping”
cn	name	Figure 41, “Group Properties Mapping”

Custom (Non-LDAP)

Select **External** on the **Advanced Account Settings** panel.



Account Provider Selection

Select the user account storage.

Authentication Account Provider

○ **Database**

All accounts will be stored in the registry database.

○ **LDAP**

Registry accounts integrated with LDAP server.

● **External**

Registry accounts integrated with other external storage. The interface **org.systinet.uddi.account.ExternalBackendApi** must be implemented and added to the registry installation.

External accounts require implementation of the interface `org.systinet.uddi.account.ExternalBackendApi`.

Deployment to an Application Server

To deploy HPE SOA Registry Foundation to any application server, it must be installed as standalone server, as described in "Installation" on page 38. After installation, use the Setup tool as described in "Creating a Web Application Archive (WAR,EAR)" below to create Web application archive (WAR,EAR) for the specific application server.

The WAR file or EAR file is then prepared for deployment to the application server. You must deploy it into the application server manually, according to your specific application server's instructions:

> **Note:** If you are going to use the HSQL (despite the fact it is recommended only for demo/testing purposes) and deploying the `registry.war` on a different machine, do not forget to copy the database files from the `REGISTRY_HOME/hsqldb` directory to the host where the application server is running. Then, change the database configuration accordingly after the first start of HPE SOA Registry Foundation.

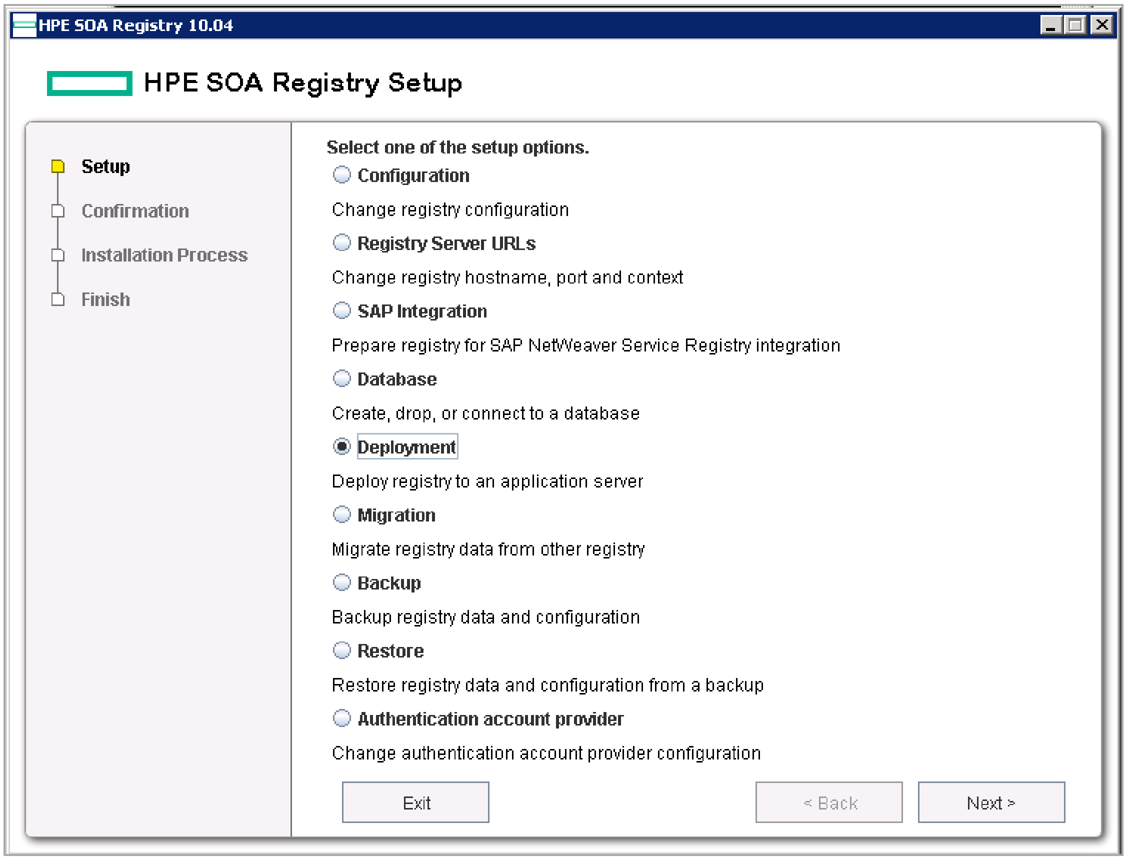
Creating a Web Application Archive (WAR,EAR)

To create a Web application archive:

1. Briefly, launch the Setup tool by executing the following command from the bin directory of your installation:

Windows:	setup.bat
UNIX:	./setup.sh

2. Select Deployment on the first panel:



Select one of the setup options.

○ Configuration

Change registry configuration

○ Registry Server URLs

Change registry hostname, port and context

○ SAP Integration

Prepare registry for SAP NetWeaver Service Registry integration

○ Database

Create, drop, or connect to a database

● Deployment

Deploy registry to an application server

○ Migration

Migrate registry data from other registry

○ Backup

Backup registry data and configuration

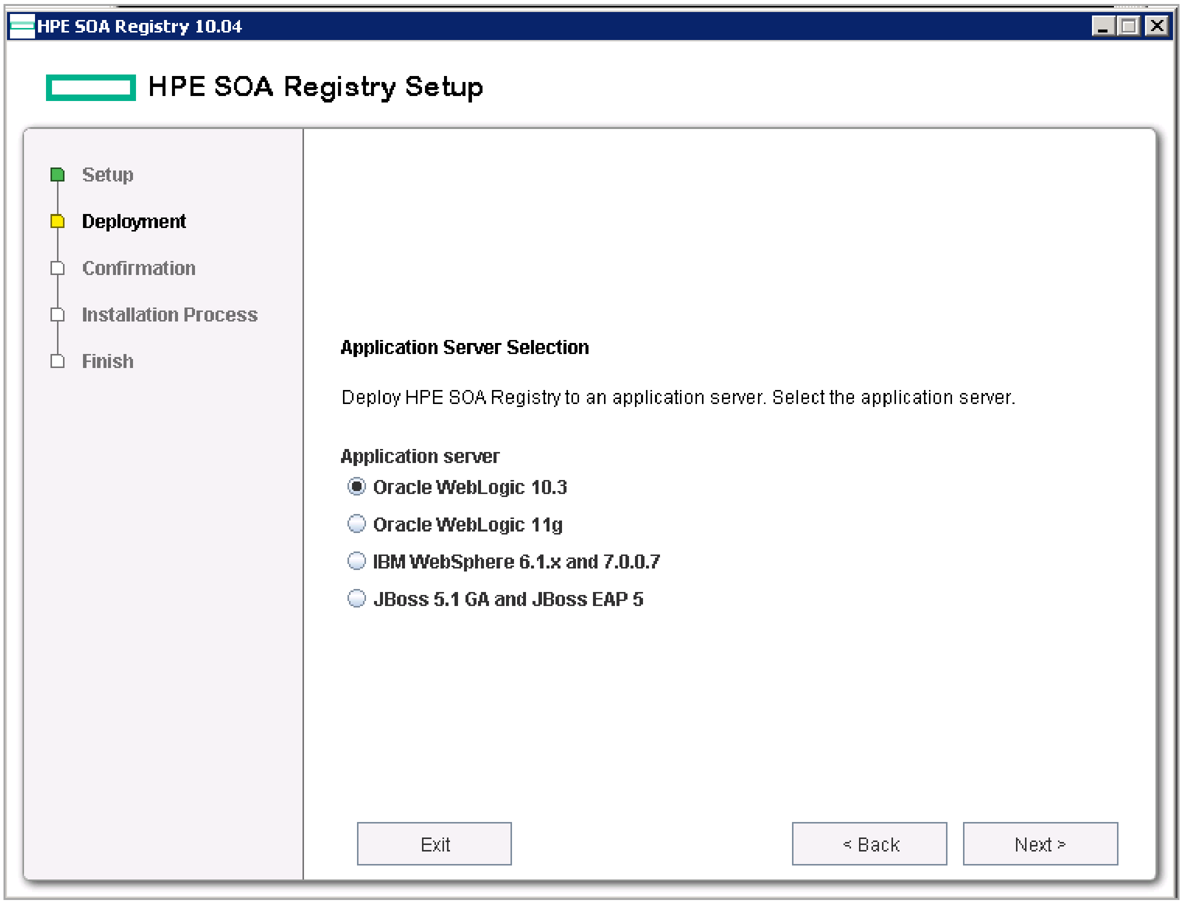
○ Restore

Restore registry data and configuration from a backup

○ Authentication account provider

Change authentication account provider configuration

3. Select the application server on the next panel.



Application Server Selection

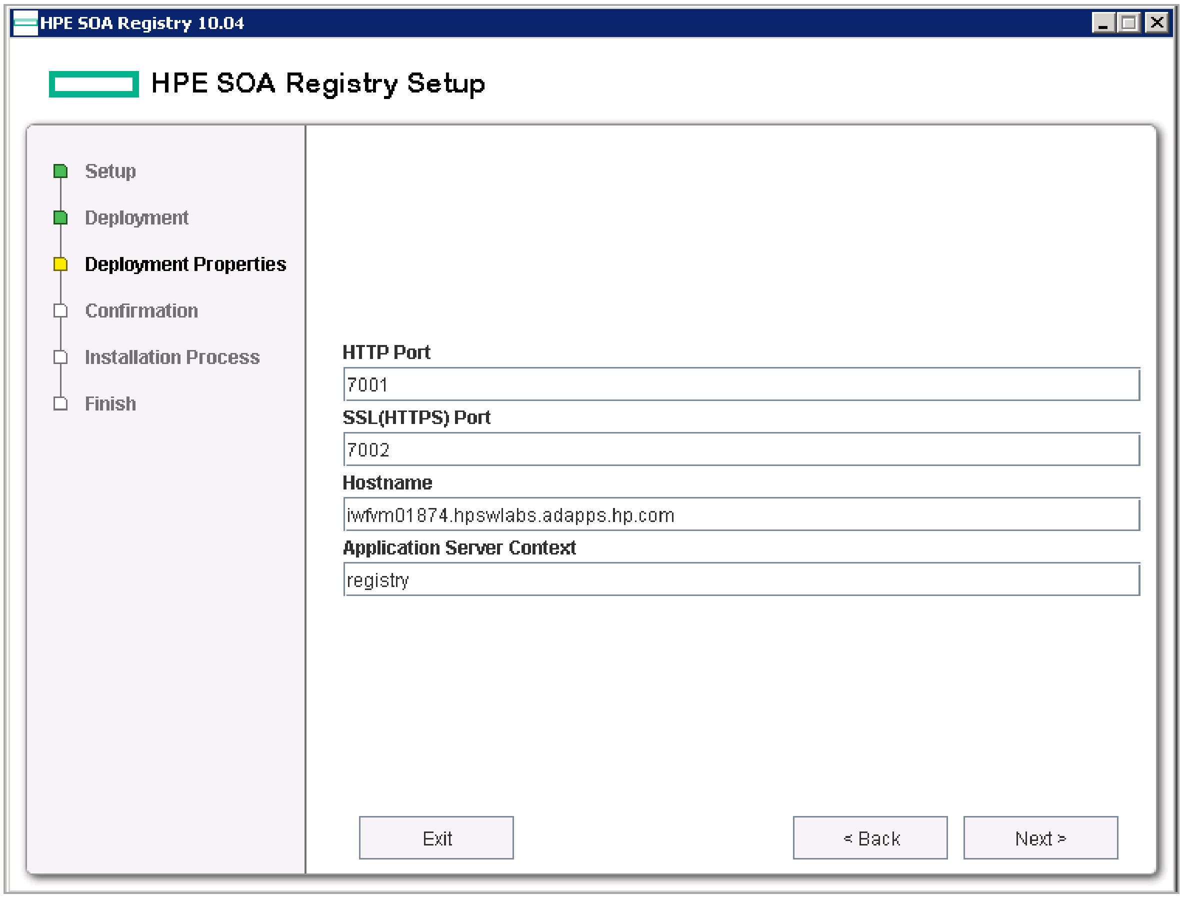
Deploy HPE SOA Registry to an application server. Select the application server.

Application server

- ● Oracle WebLogic 10.3
- ○ Oracle WebLogic 11g
- ○ IBM WebSphere 6.1.x and 7.0.0.7
- ○ JBoss 5.1 GA and JBoss EAP 5

Select the application server to which you want to deploy HPE SOA Registry Foundation.

4. The next panel shows deployment settings on the application server.



HTTP Port: 7001

SSL(HTTPS) Port: 7002

Hostname: iwfvm01874.hpswlabs.adapps.hp.com

Application Server Context: registry

HTTP Port - HTTP port of the application server

SSL(HTTPS) Port - HTTPS port of the application server

Host name - Host name of the application server

Application Server Context - Use the context you will use to deploy on the application server. (default: wasp)

Deployment Process After Confirmation of Settings

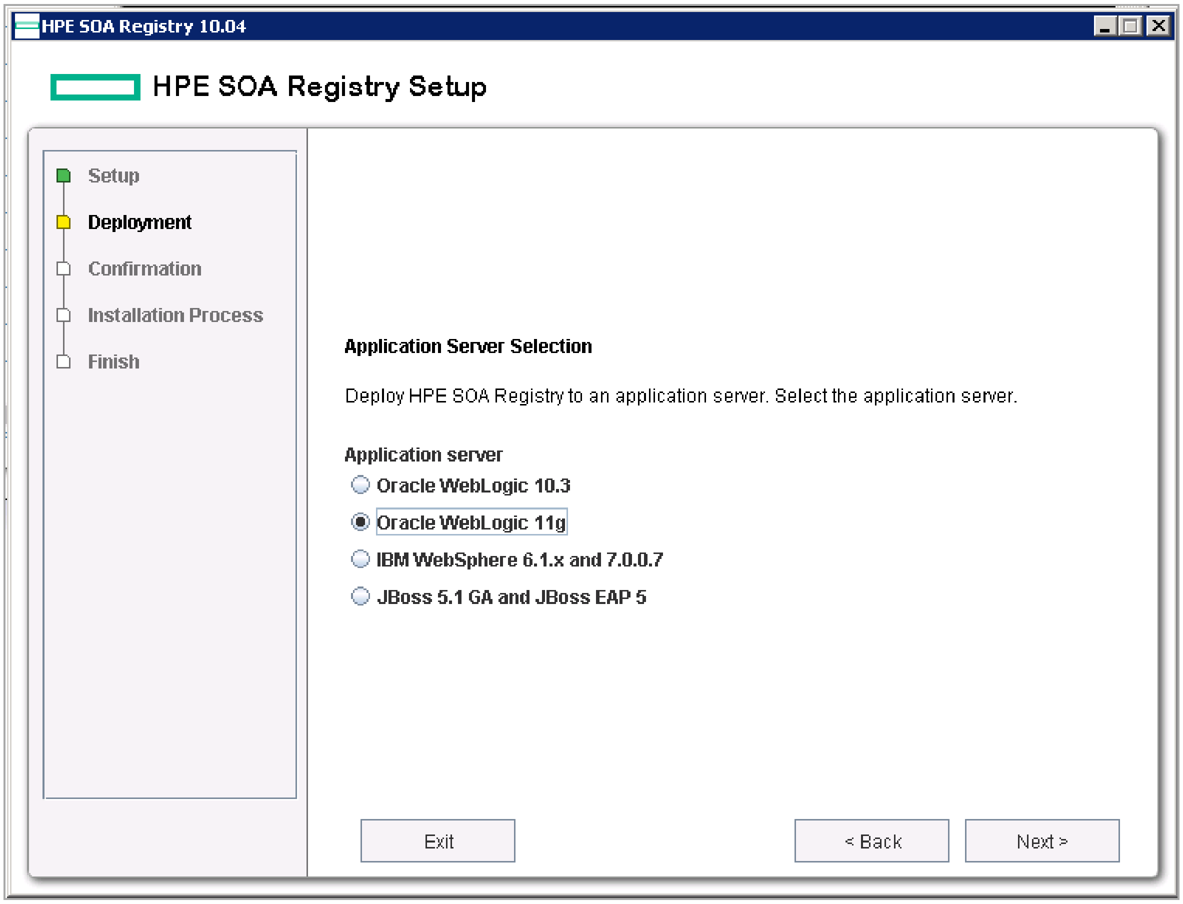
To continue the deployment process, follow the instruction in the log window. For further details, see the instructions in the individual sections below dedicated to the individual application servers.

- "Weblogic" on the next page
- "WebSphere" on page 144
- "JBoss" on page 149

Weblogic

This section describes Oracle WebLogic deployment options. Both Oracle WebLogic 10.3 and Oracle WebLogic 11g are supported.

Invoke Oracle WebLogic installation by selecting WebLogic deployment option in the following installer screen:



Application Server Selection

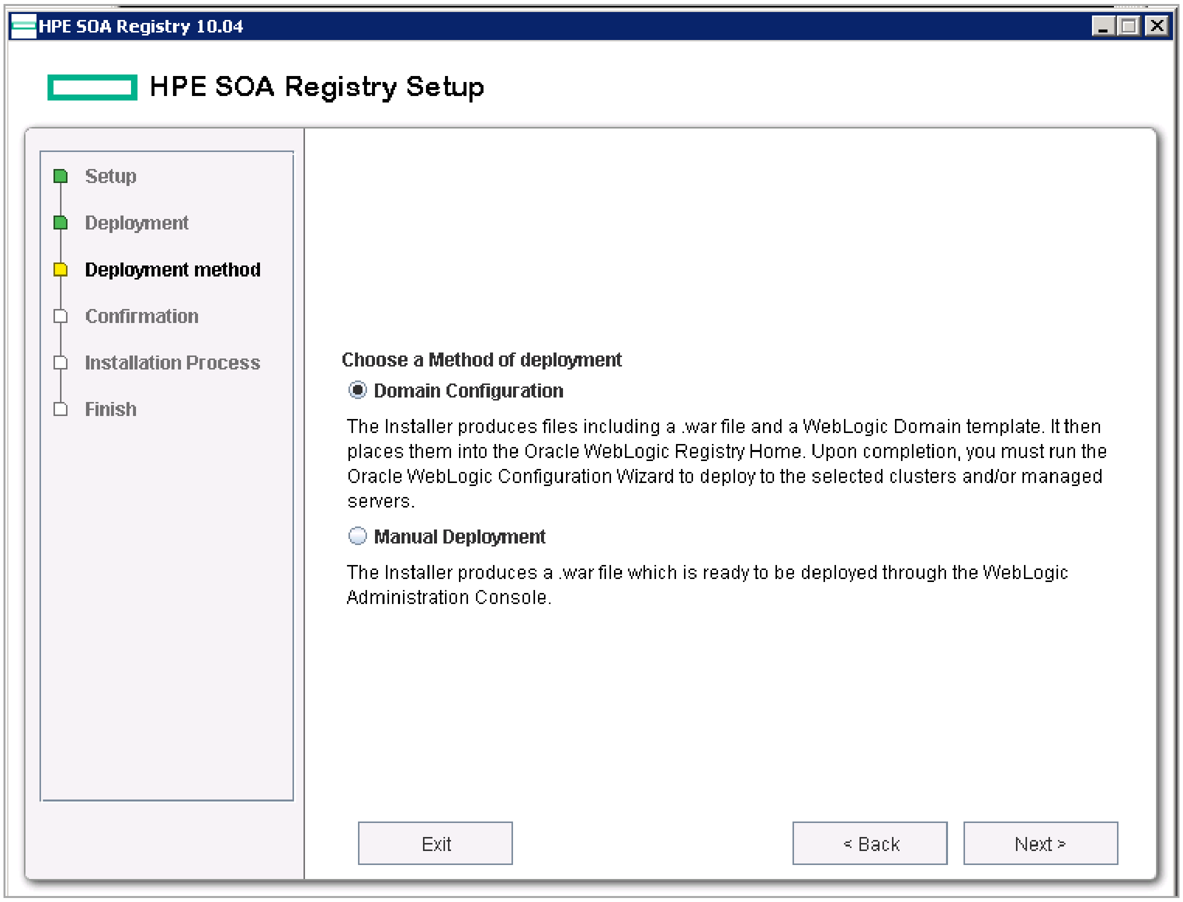
Deploy HPE SOA Registry to an application server. Select the application server.

Application server

- ○ Oracle WebLogic 10.3
- ● Oracle WebLogic 11g
- ○ IBM WebSphere 6.1.x and 7.0.0.7
- ○ JBoss 5.1 GA and JBoss EAP 5

> **Note:** `WL_HOME` refers to the directory where WebLogic is installed.
>
> `REGISTRY_HOME` refers to the directory in which the HPE SOA Registry Foundation distribution is installed.

The next installation screen requests a method deployment.



Choose a Method of deployment

● Domain Configuration

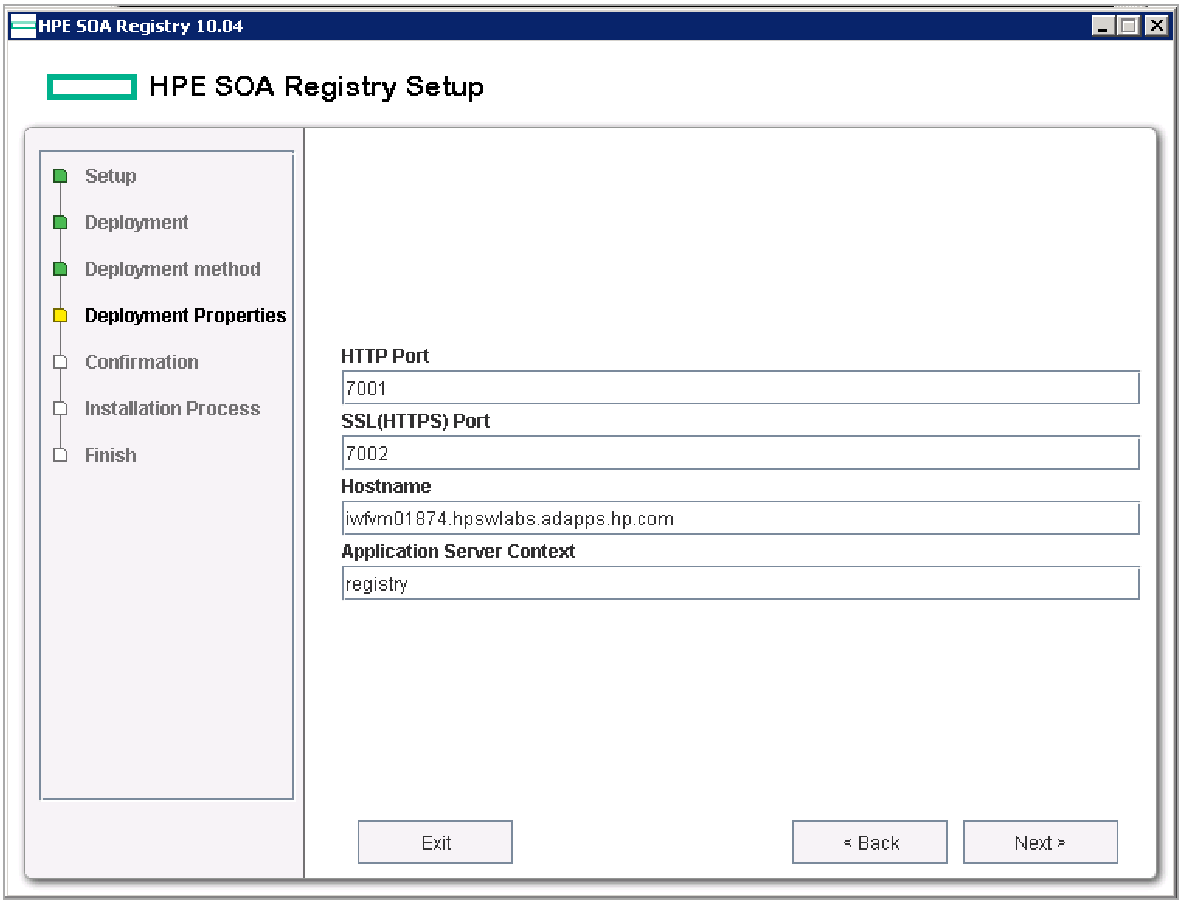
The Installer produces files including a .war file and a WebLogic Domain template. It then places them into the Oracle WebLogic Registry Home. Upon completion, you must run the Oracle WebLogic Configuration Wizard to deploy to the selected clusters and/or managed servers.

○ Manual Deployment

The Installer produces a .war file which is ready to be deployed through the WebLogic Administration Console.

Domain Configuration is supported for Weblogic 11g only. Upon completion, you must run the Oracle WebLogic Configuration Wizard to create WebLogic domains. See more at Creating WebLogic Domains Using the Configuration Wizard. If you select Oracle Weblogic 10.3 on the previous page, Domain Configuration will not be shown.

The next installation screen requests URL components (hostname, port, ssl port, context) that HPE SOA Registry Foundation uses in its web user interface. If you use proxy or load-balancer, the URL components should point to it. The registry administrator can change the URL later in Registry Management. If the URL is incorrect the web UI cannot be accessed until the URL value is corrected in the configuration files. In the case of domain configuration, the hostname and port values in this page must be the same as the values set when running the Configuration tools to create the weblogic domain.



HPE SOA Registry Setup

HTTP Port: 7001

SSL(HTTPS) Port: 7002

Hostname: iwfvm01874.hpswlabs.adapps.hp.com

Application Server Context: registry

If you want to deploy manually you must specify the location of the HPE SOA Registry Foundation .war

file. This file is in `REGISTRY_HOME/conf/porting/weblogic/build/[context_name].war` (if installation succeeded).

Other required changes to complete the integration:

1. Modify the Oracle WebLogic server launch script which is:

   a. `WL_HOME/user_projects/domains/DOMAIN_NAME/startWebLogic.sh or startWebLogic.cmd`

   b. Add the following property to the Java command line for starting the WebLogic server:

   ```
   -Djava.security.auth.login.config=REGISTRY_HOME/conf/jaas.config
   ```

2. Import the SSL certificate of the WebLogic server to the HPE SOA Registry Foundation configuration.

   Obtain the WebLogic SSL certificate. There are two methods:

a. You can get certificate using Internet Explorer 6.0 web browser connected to WebLogic via HTTPS. Select "Properties" in context menu of the page, button "Certificates", tab "Details", button "Copy to file", and then export certificate in Base 64 encoded X.509 .cer format.

b. You can also use `REGISTRY_HOME/bin/sslTool.sh or REGISTRY_HOME\bin\sslTool.bat` to get certificate. Run command:

```
sslTool serverInfo --url https://HOST:9043 --certFile weblogic.cer
```

c. This command will connect to specified host and port using HTTPS and it will store server certificate into specified file.

   To import this certificate use

   PStoreTool located in [registry_home]/bin PStoreTool.sh add -config

   [registry_home]/conf/clientconf.xml -certFile [weblogic.cer]

3. Enable SSL in WebLogic if not yet enabled and (re)start the Oracle WebLogic server.

   Deployment should now be complete. The HPE SOA Registry Foundation URL is

   ```
   http://[hostname]:[http_port]/[context]/uddi/web
   ```

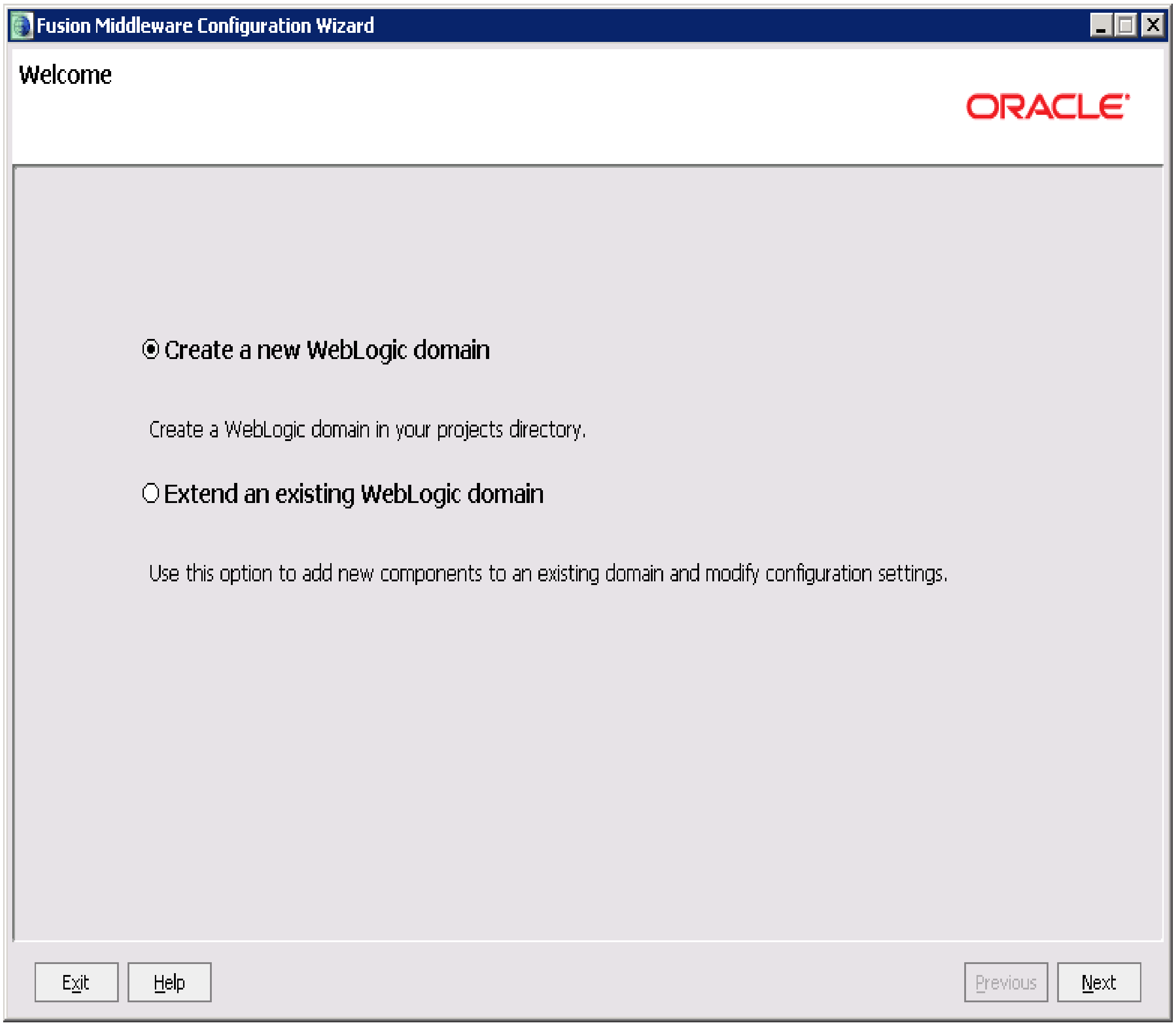
> **Note:** WebLogic 8.x: When "Segmentation fault" problems occur during WebLogic startup on RedHat Enterprise Linux, you have to set environment variable LD_ASSUME_KERNEL to value 2.4.1. Add this line to WebLogic startup script: `export LD_ASSUME_KERNEL="2.4.1"`
>
> **To set up a managed server:** Cause: WebLogic 10g (10.3) node manager uses system variables PATH and CLASSPATH in the server start command. The node manager does not handle these variables if they contain spaces. To avoid this problem, do the following:
>
> 1. On Windows, replace the conflicting parts of the paths with DOS-like 8.3 file names and restart node manager.
> 2. Edit `WL_HOME/common/nodemanager/nodemanager.properties`, and add the parameter `StartScriptEnabled=true`, and then restart node manager.

Creating WebLogic Domains Using the Configuration Wizard

This section provides information and examples for some common domain configuration tasks using the Configuration Wizard.

Create New Domain with default Managed Server

1. Start the Configuration Wizard in graphical mode.
   a. On Windows: **Start—>Programs—>BEA Products—>Tools—>Configuration Wizard** or enter <WEBLOGIC_HOME>/common/bin/config.cmd
   b. On UNIX: enter <WEBLOGIC_HOME>/common/bin/config.sh
2. In the Welcome window select Create a new WebLogic domain



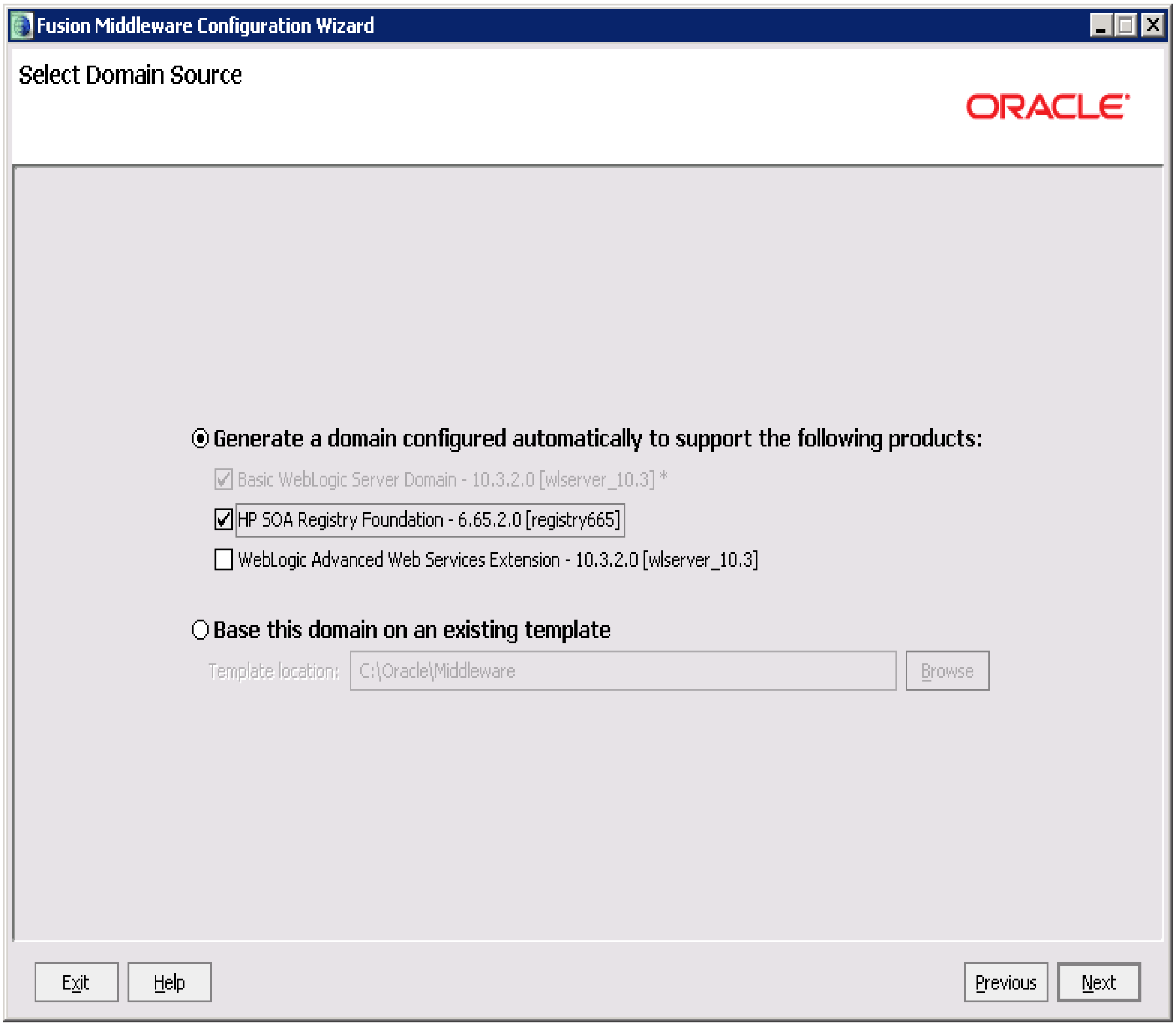
● Create a new WebLogic domain

Create a WebLogic domain in your projects directory.

○ Extend an existing WebLogic domain

Use this option to add new components to an existing domain and modify configuration settings.

3. In the **Select a Domain Source window**, select **Generate a domain configured automatically to support the following BEA products** and the **HPE SOA Registry Foundation** check box is selected.



Select Domain Source

● Generate a domain configured automatically to support the following products:

☑ Basic WebLogic Server Domain - 10.3.2.0 [wlserver_10.3] *

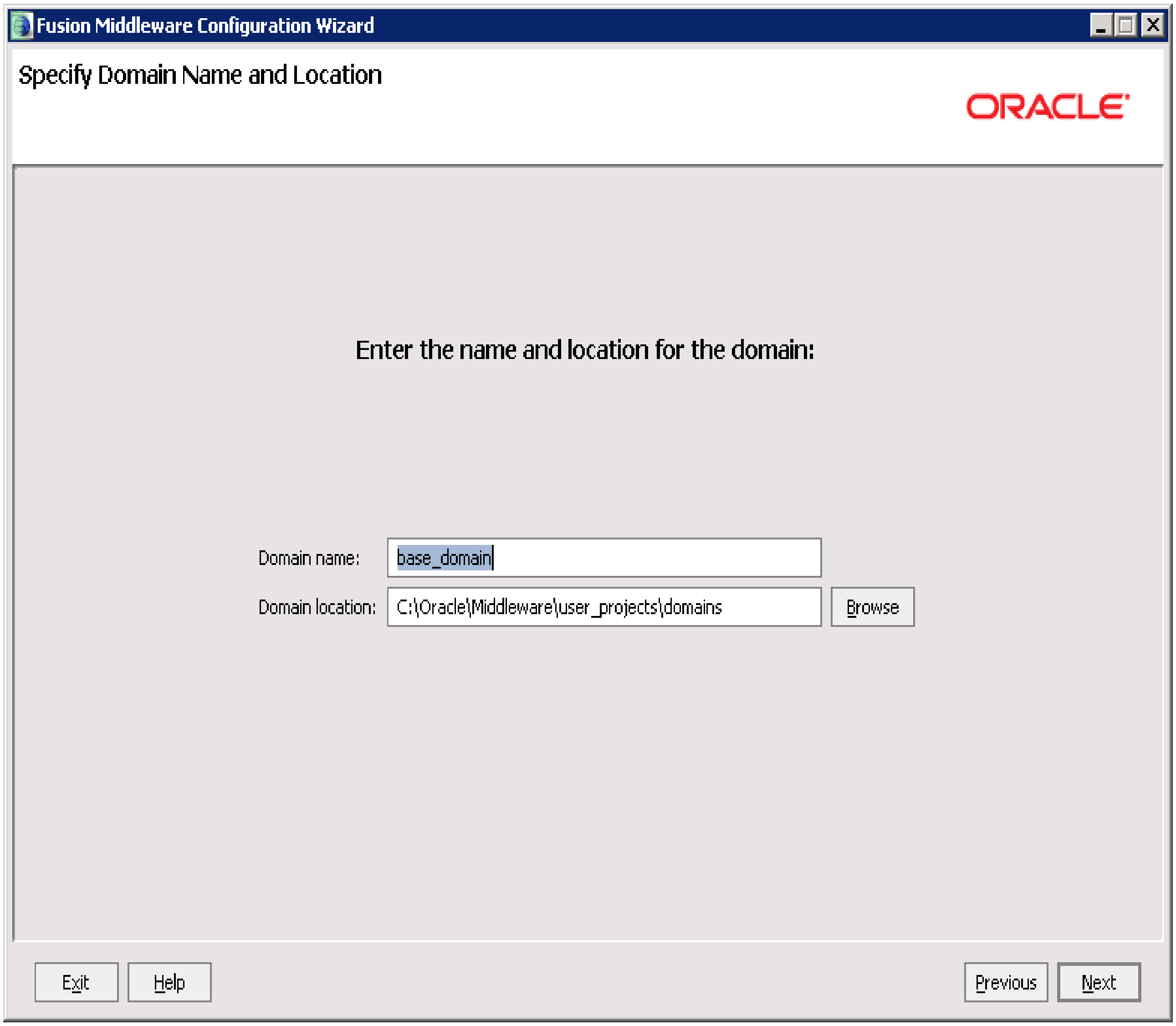
☑ HP SOA Registry Foundation - 6.65.2.0 [registry665]

□ WebLogic Advanced Web Services Extension - 10.3.2.0 [wlserver_10.3]

○ Base this domain on an existing template

Template location: C:\Oracle\Middleware

4. In the **Specify Domain Name and Location** window, enter the name and location for the domain.



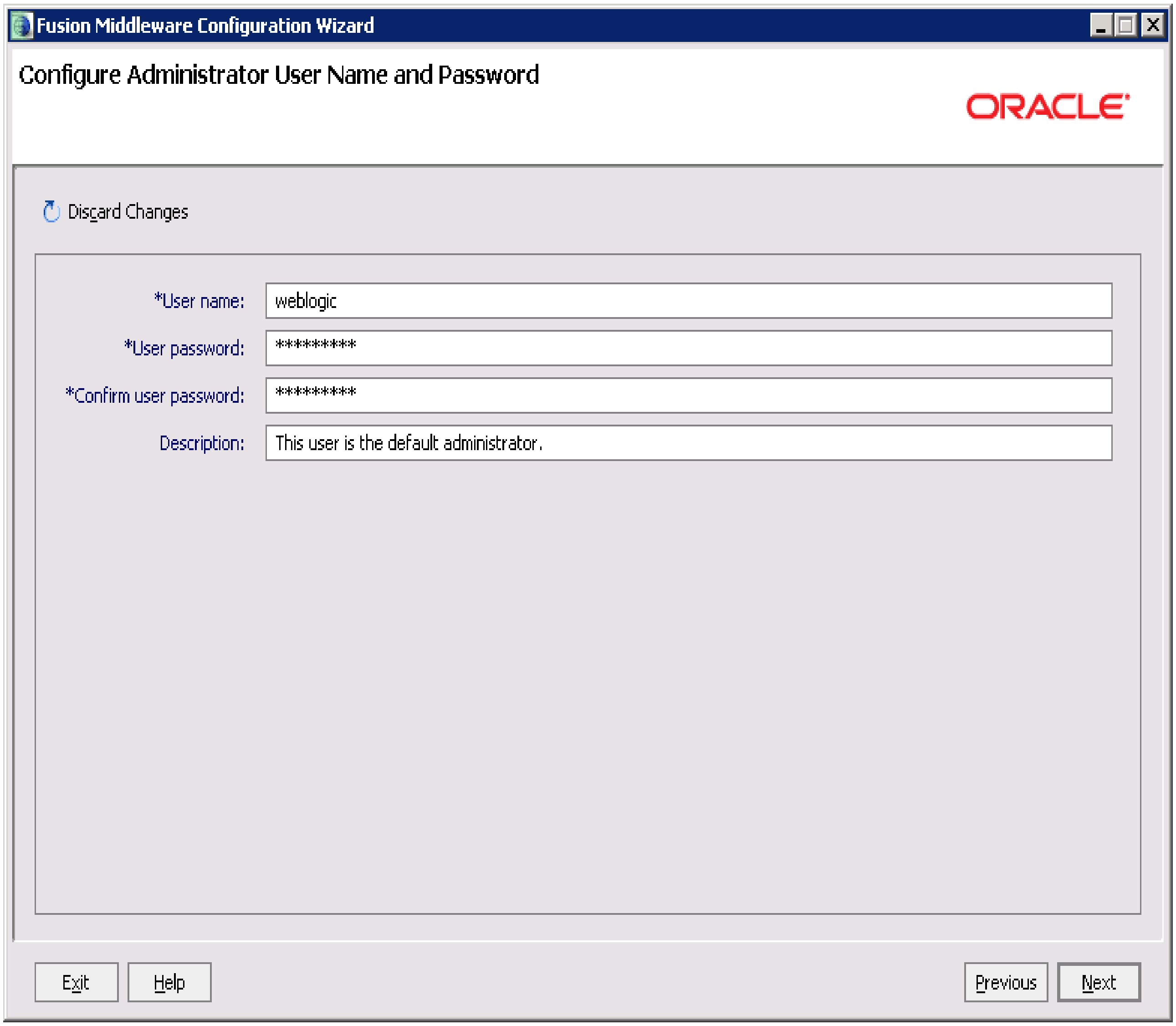
Fusion Middleware Configuration Wizard
Specify Domain Name and Location
ORACLE
Exit
Help
Previous
Next
Browse

Enter the name and location for the domain:

Domain name: base_domain

Domain location: C:\Oracle\Middleware\user_projects\domains

5. In the **Configure Administrator Username and Password** window, enter a valid username and password, and click Next. This username is used to boot the Administration Server and connect to it.



Configure Administrator User Name and Password

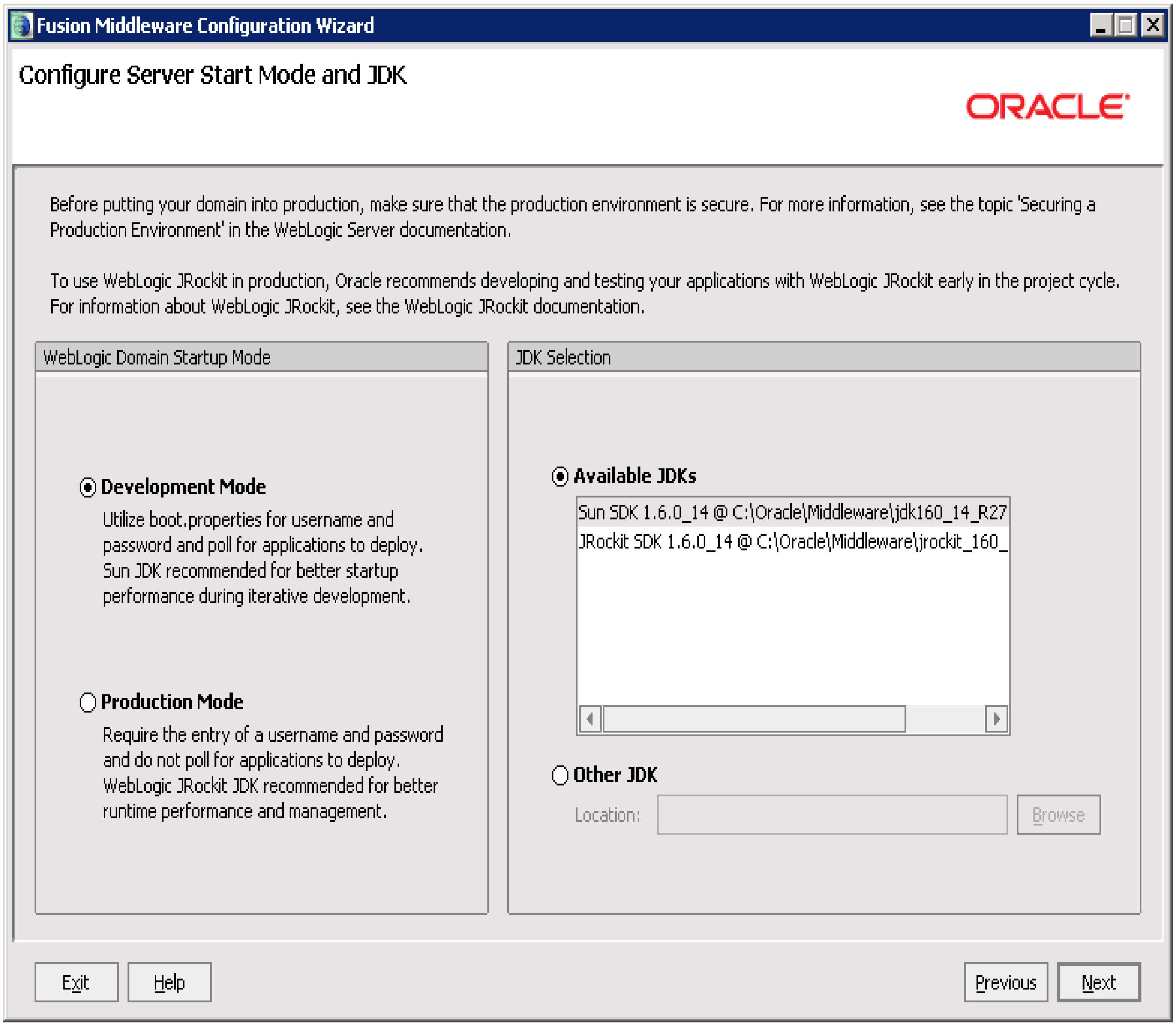
*User name: weblogic

*User password: *********

*Confirm user password: *********

Description: This user is the default administrator.

6. In the Configure Server Start Mode and JDK window, specify whether to start the server in development mode or production mode, and select which JDK to use



Fusion Middleware Configuration Wizard

Configure Server Start Mode and JDK

Before putting your domain into production, make sure that the production environment is secure. For more information, see the topic 'Securing a Production Environment' in the WebLogic Server documentation.

To use WebLogic JRockit in production, Oracle recommends developing and testing your applications with WebLogic JRockit early in the project cycle. For information about WebLogic JRockit, see the WebLogic JRockit documentation.

WebLogic Domain Startup Mode

● Development Mode

Utilize boot.properties for username and password and poll for applications to deploy. Sun JDK recommended for better startup performance during iterative development.

○ Production Mode

Require the entry of a username and password and do not poll for applications to deploy. WebLogic JRockit JDK recommended for better runtime performance and management.

JDK Selection

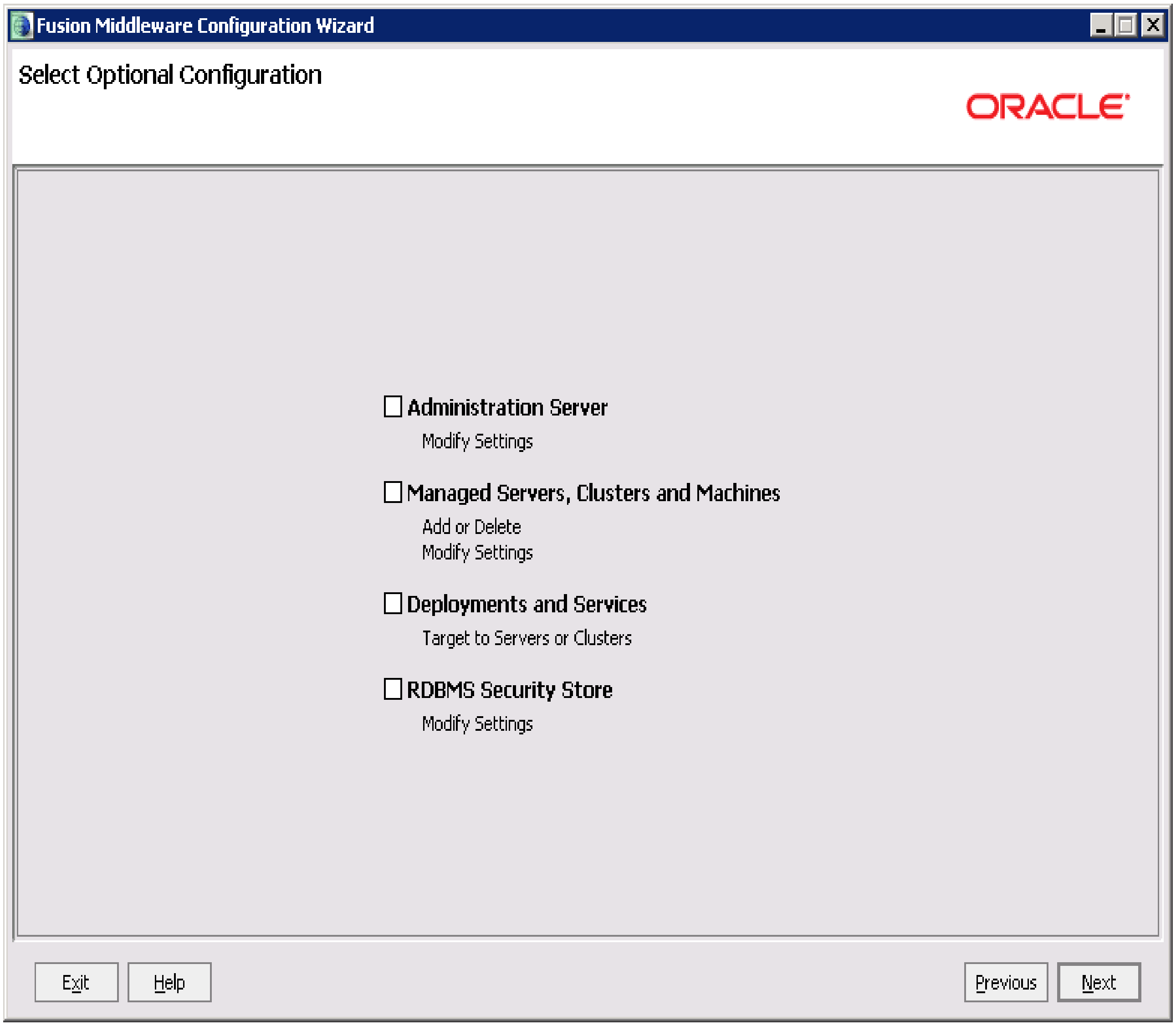
● Available JDKs

Sun SDK 1.6.0_14 @ C:\Oracle\Middleware\jdk160_14_R27
JRockit SDK 1.6.0_14 @ C:\Oracle\Middleware\jrockit_160_

○ Other JDK

Location:

7. In Select Optional Configuration indicate whether you want to change the distribution of your domain across servers, clusters, and machines



Fusion Middleware Configuration Wizard

Select Optional Configuration

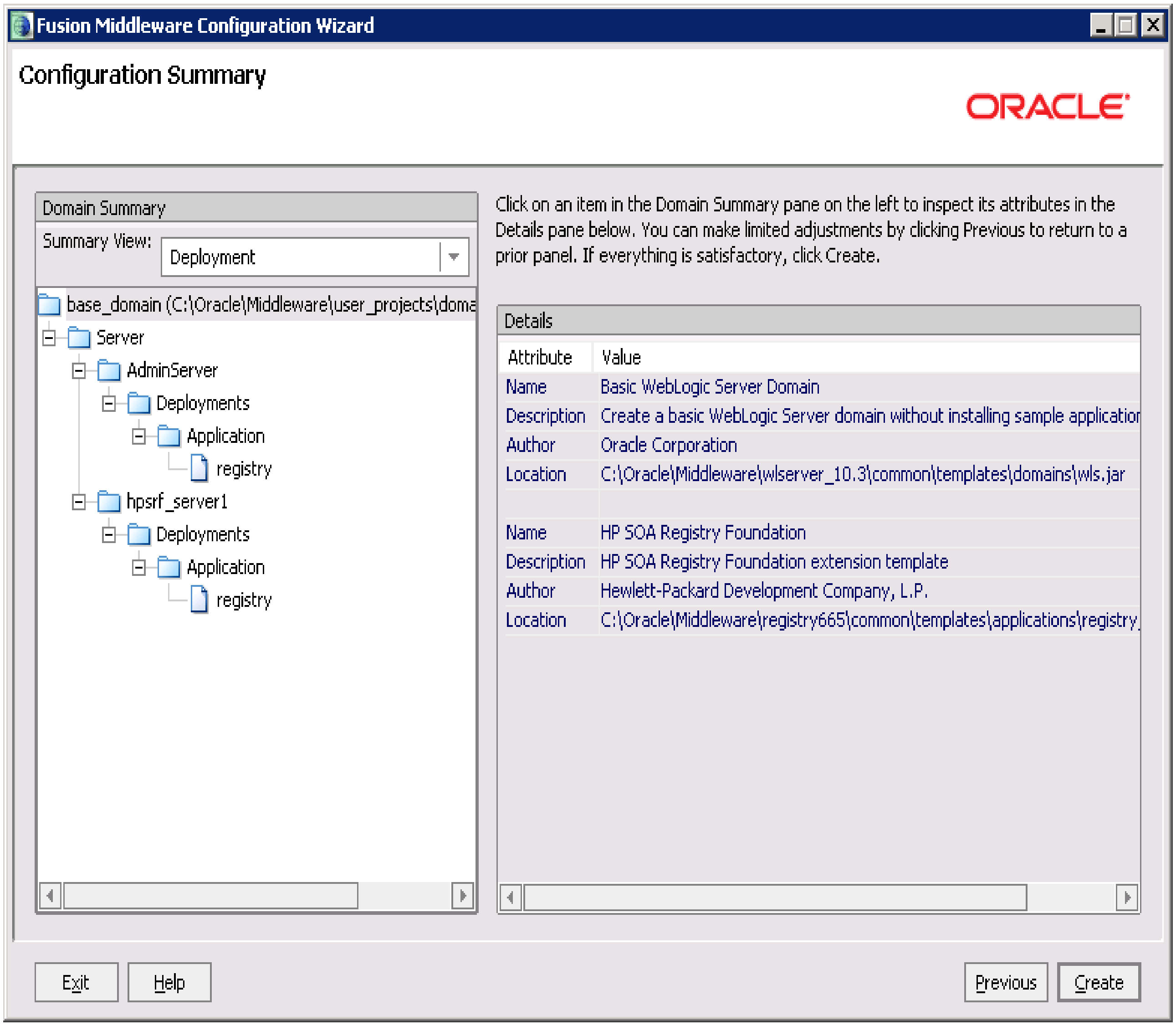
□ Administration Server
Modify Settings

□ Managed Servers, Clusters and Machines
Add or Delete
Modify Settings

□ Deployments and Services
Target to Servers or Clusters

□ RDBMS Security Store
Modify Settings

8. In Configuration Summary, review the values supplied to the configuration and click Create to create the domain.

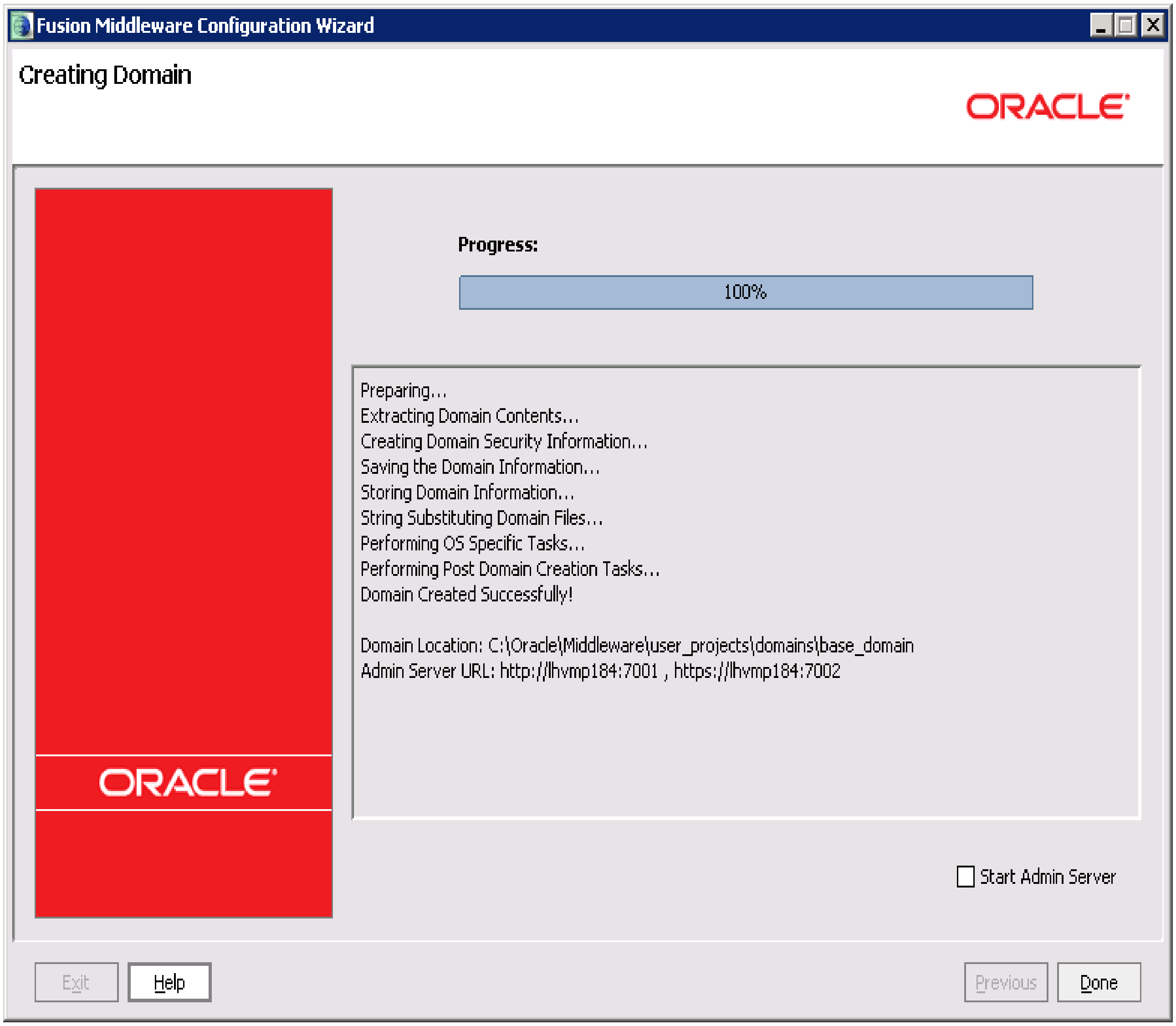


Summary View: Deployment

Click on an item in the Domain Summary pane on the left to inspect its attributes in the Details pane below. You can make limited adjustments by clicking Previous to return to a prior panel. If everything is satisfactory, click Create.

Attribute	Value
Name	Basic WebLogic Server Domain
Description	Create a basic WebLogic Server domain without installing sample applicatior
Author	Oracle Corporation
Location	C:\Oracle\Middleware\wlserver_10.3\common\templates\domains\wls.jar
Name	HP SOA Registry Foundation
Description	HP SOA Registry Foundation extension template
Author	Hewlett-Packard Development Company, L.P.
Location	C:\Oracle\Middleware\registry665\common\templates\applications\registry_

9. The domain will be created so long as there are no erroneous or conflicting values.

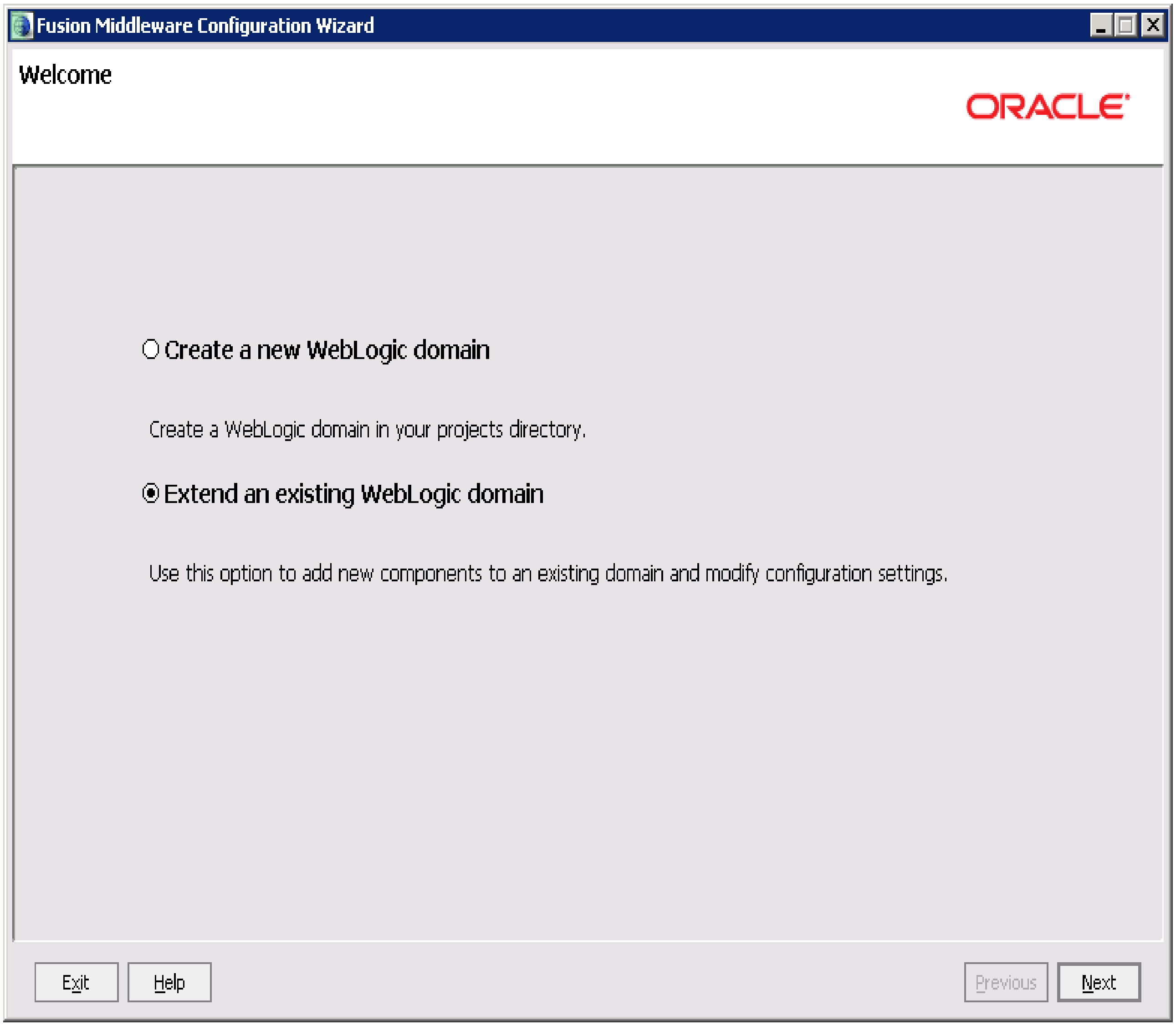


```
Preparing...
Extracting Domain Contents...
Creating Domain Security Information...
Saving the Domain Information...
Storing Domain Information...
String Substituting Domain Files...
Performing OS Specific Tasks...
Performing Post Domain Creation Tasks...
Domain Created Successfully!

Domain Location: C:\Oracle\Middleware\user_projects\domains\base_domain
Admin Server URL: http://lhvmp184:7001 , https://lhvmp184:7002
```

□ Start Admin Server

Extend Existing Domain and Target on an existing Managed Server

1. Start the Configuration Wizard in graphical mode.
   - On Windows: **Start—>Programs—>BEA Products—>Tools—>Configuration Wizard** or enter <WEBLOGIC_HOME>/common/bin/config.cmd
   - On UNIX: enter <WEBLOGIC_HOME>/common/bin/config.sh
2. In the **Welcome** window, select **Extend an existing WebLogic domain**.



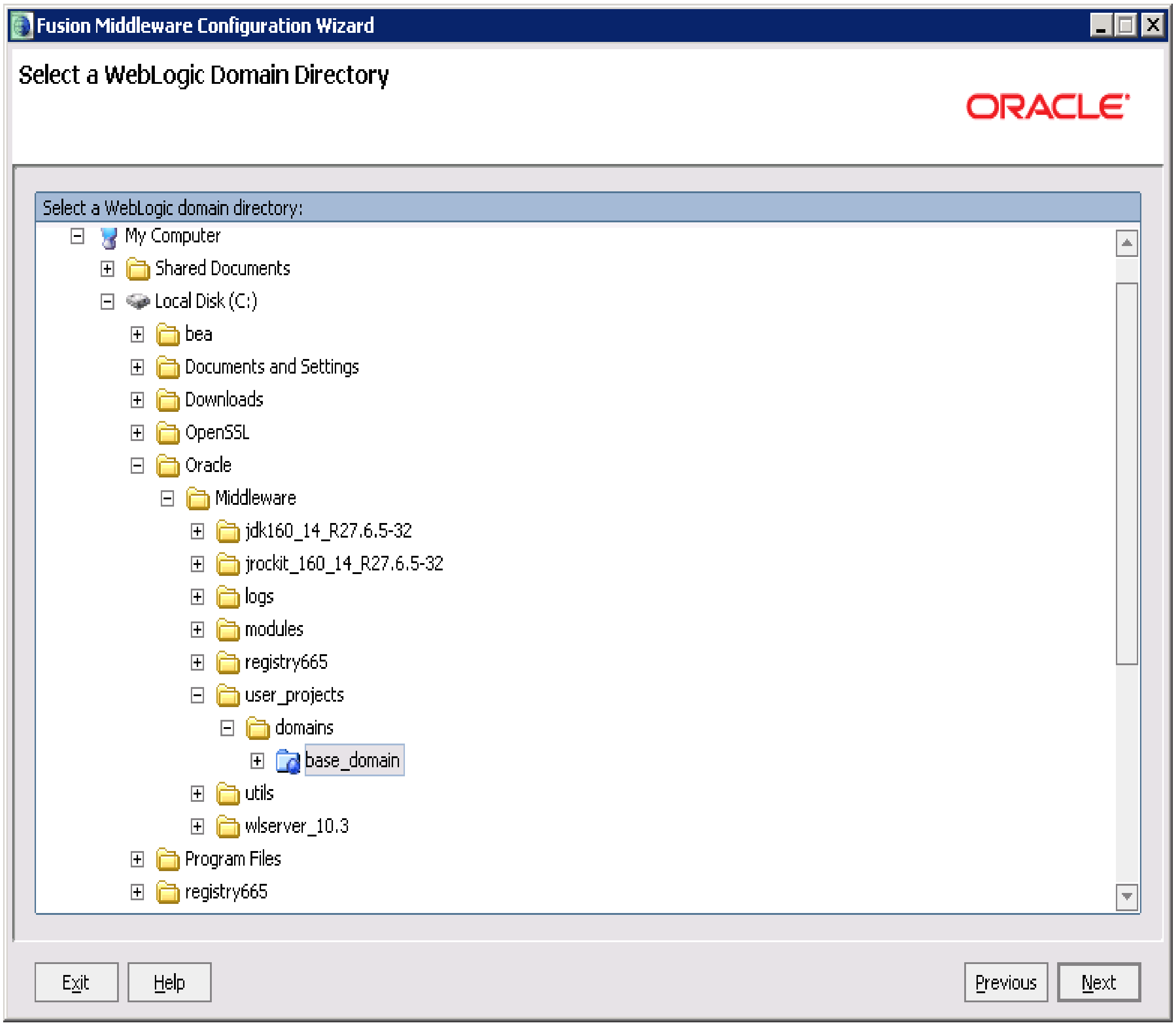
○ Create a new WebLogic domain

Create a WebLogic domain in your projects directory.

● Extend an existing WebLogic domain

Use this option to add new components to an existing domain and modify configuration settings.

3. In the Select a WebLogic Domain Directory window, navigate to the domain directory that you want to extend.

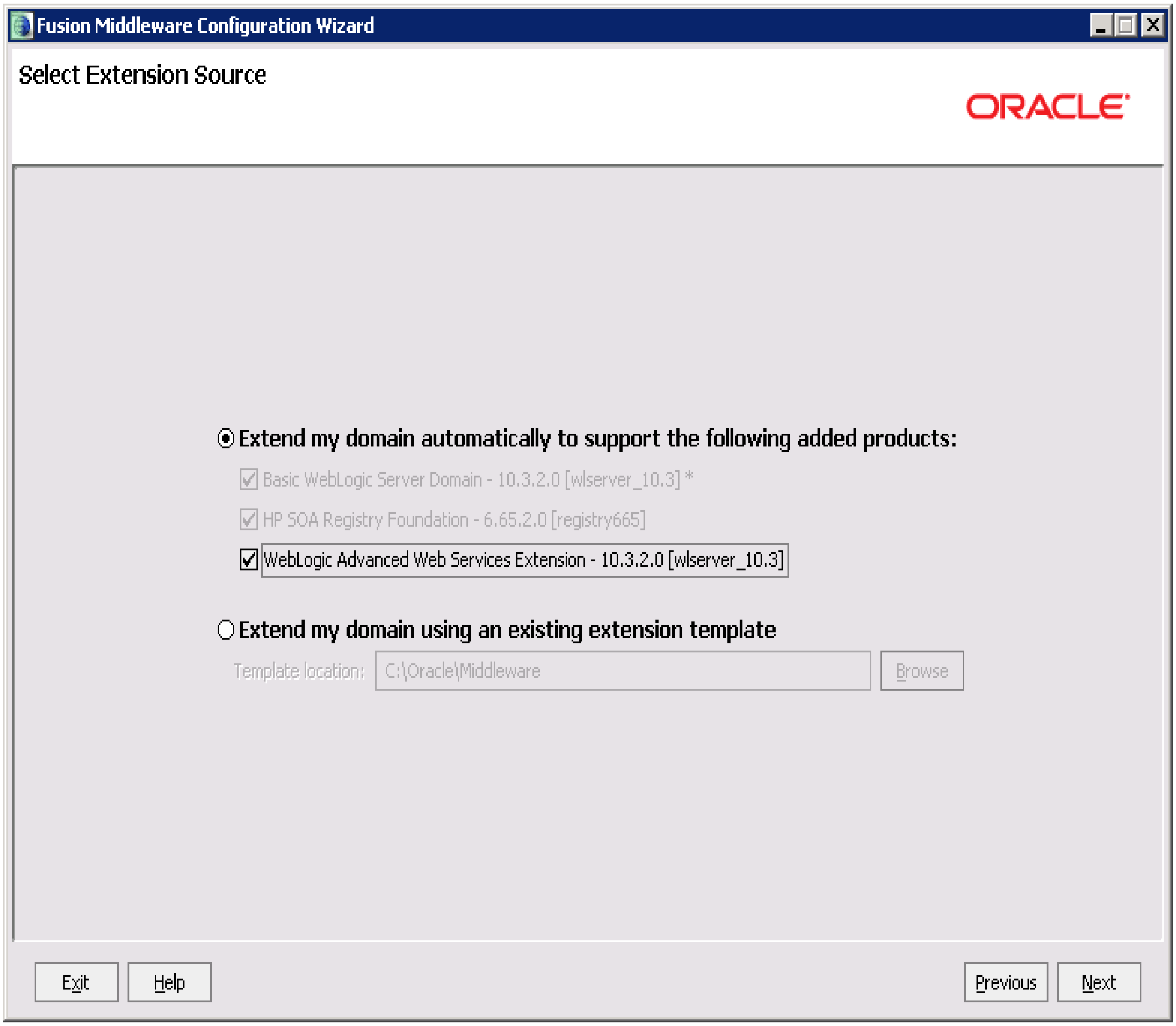


Fusion Middleware Configuration Wizard

Select a WebLogic Domain Directory

Select a WebLogic domain directory:

4. In the Select Extension Source window, you can choose to extend your domain by selecting an HPE SOA Registry Foundation.



● Extend my domain automatically to support the following added products:

☑ Basic WebLogic Server Domain - 10.3.2.0 [wlserver_10.3] *

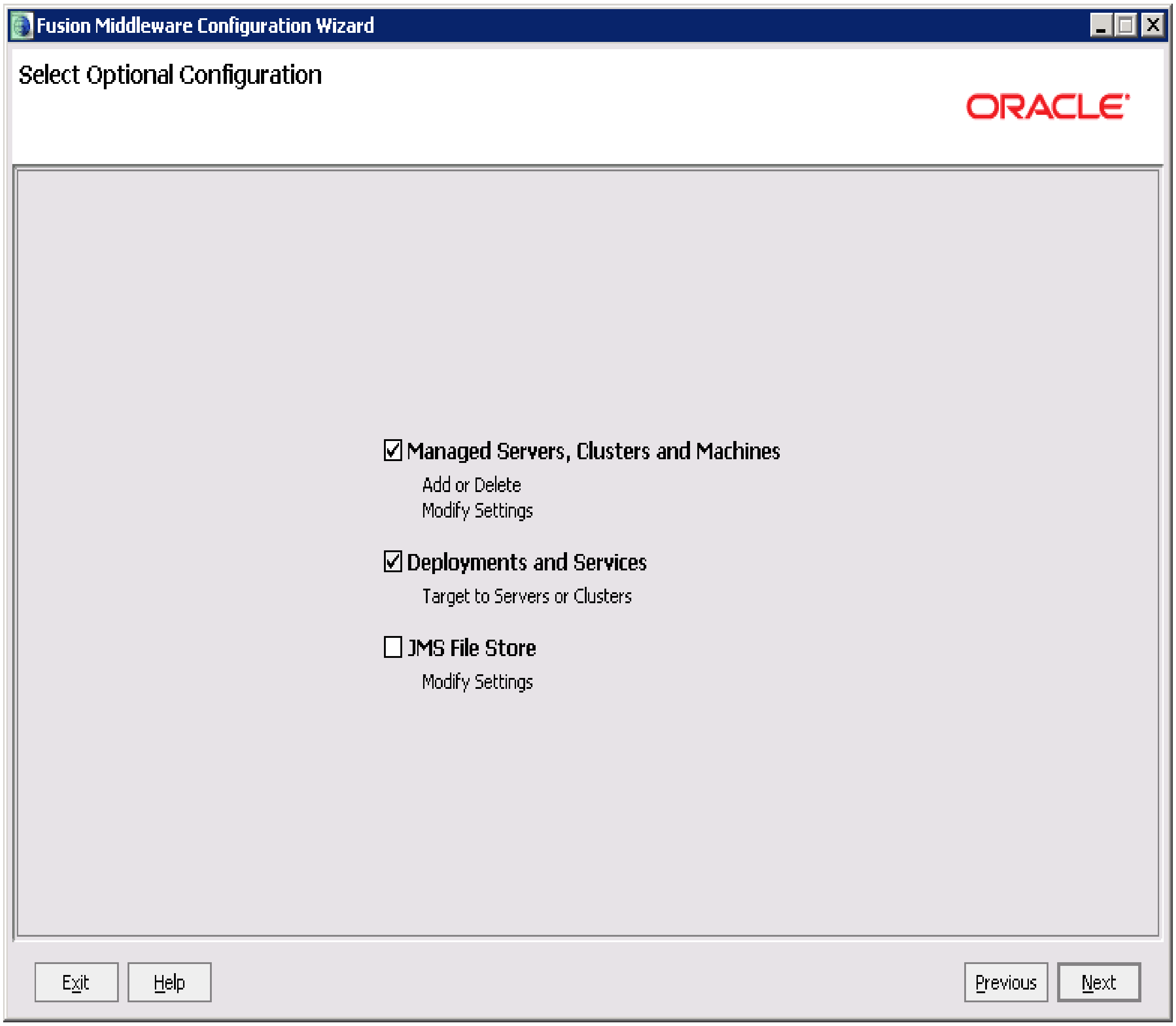
☑ HP SOA Registry Foundation - 6.65.2.0 [registry665]

☑ WebLogic Advanced Web Services Extension - 10.3.2.0 [wlserver_10.3]

○ Extend my domain using an existing extension template

Template location: C:\Oracle\Middleware

5. In Select Optional Configuration, select these options to configure servers, clusters, and machines



Select Optional Configuration

☑ Managed Servers, Clusters and Machines

Add or Delete
Modify Settings

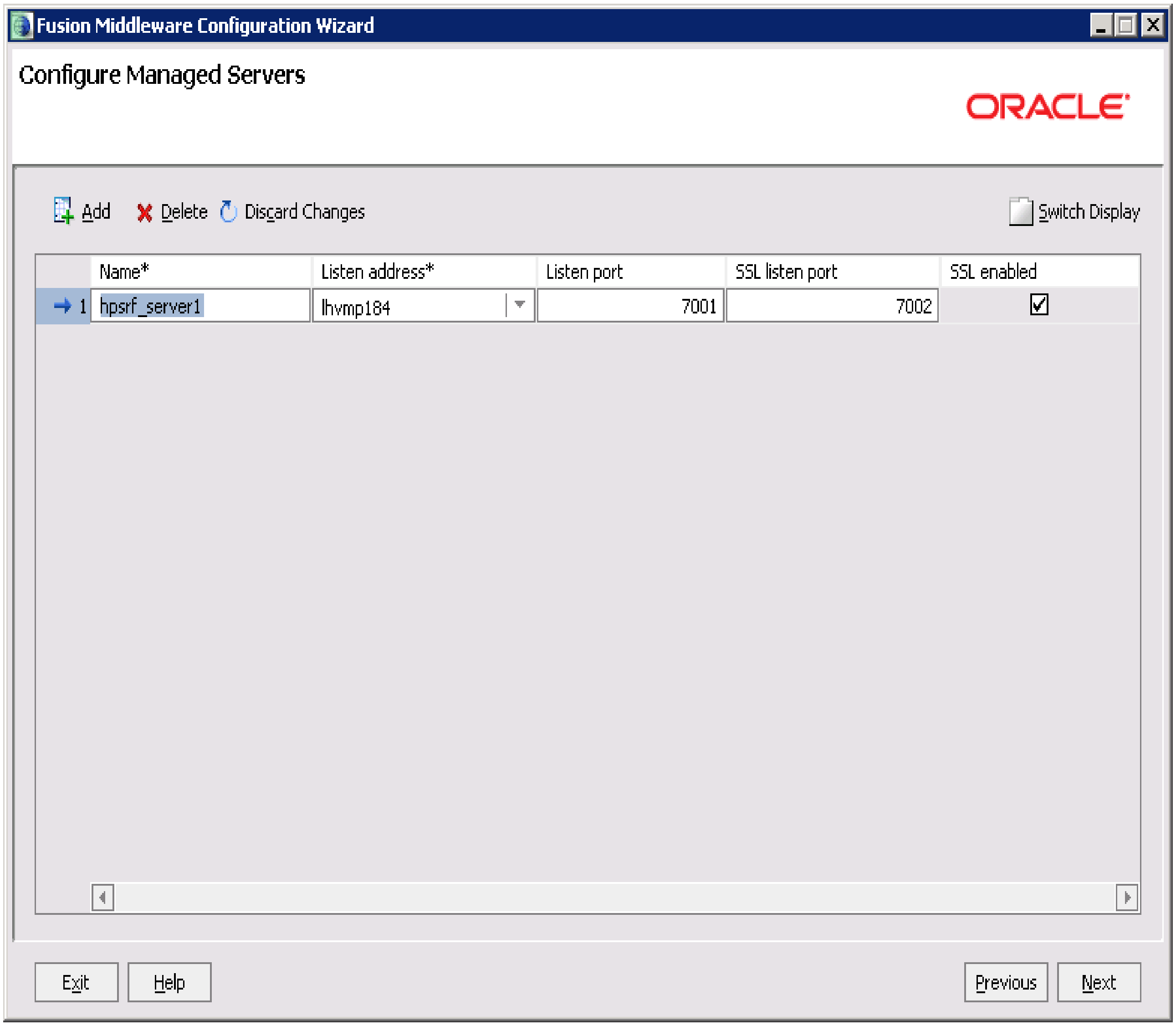
☑ Deployments and Services

Target to Servers or Clusters

□ JMS File Store

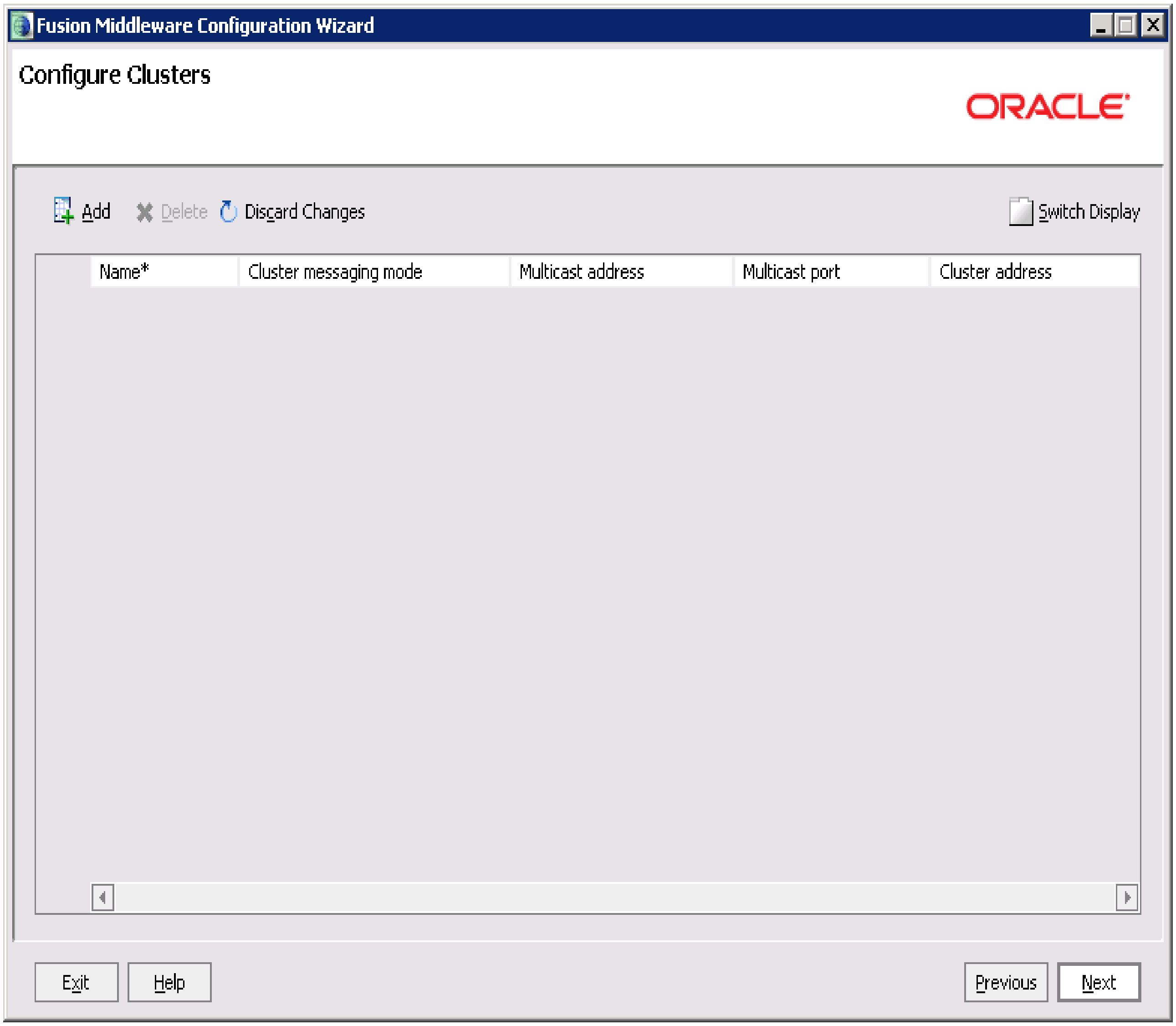
Modify Settings

6. Select the configure managed servers.



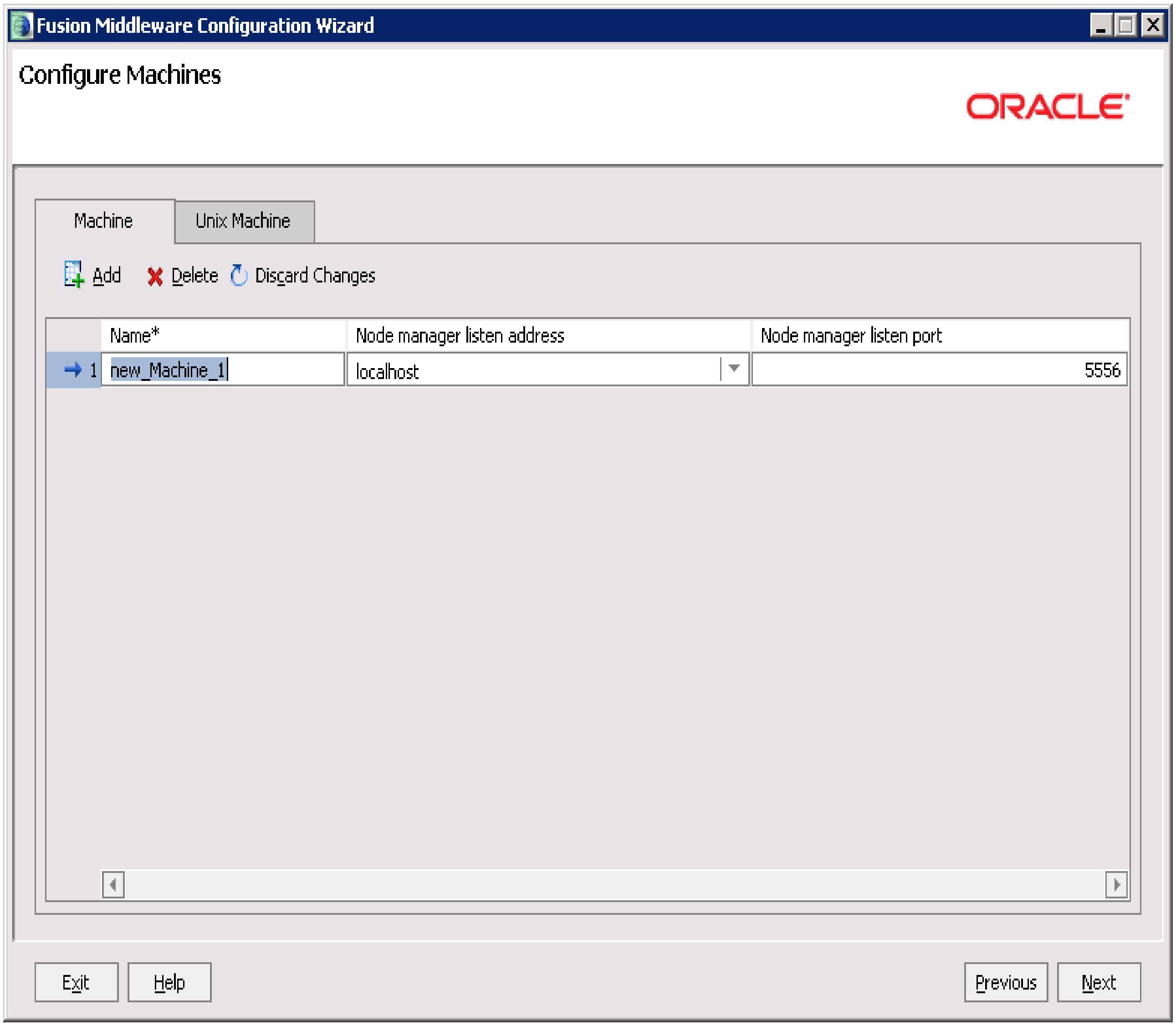
	Name*	Listen address*	Listen port	SSL listen port	SSL enabled
1	hpsrf_server1	lhvmp184	7001	7002	☑

7. Configure the clusters.



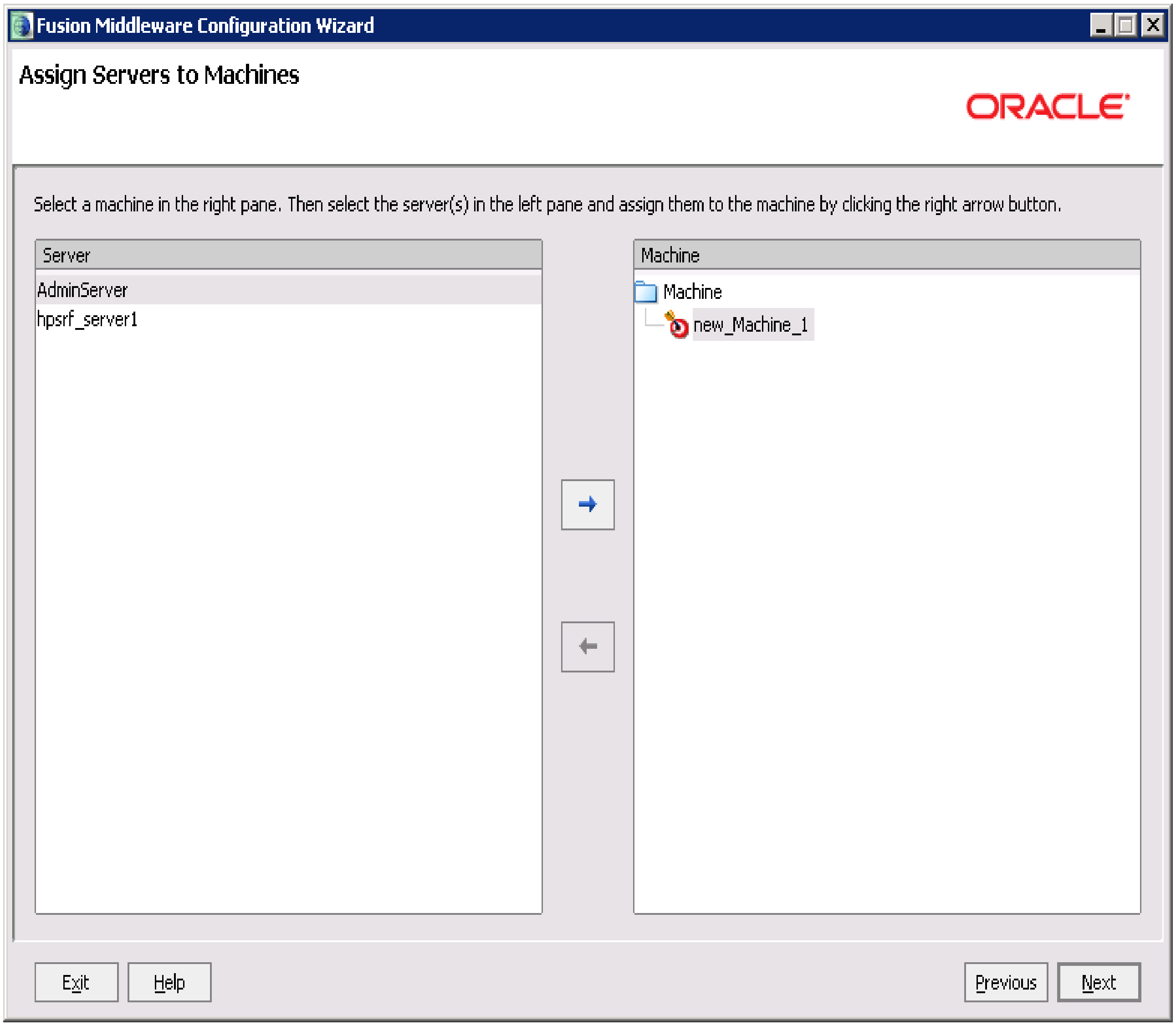
Name*	Cluster messaging mode	Multicast address	Multicast port	Cluster address

8. Configure the machines.



	Name*	Node manager listen address	Node manager listen port
1	new_Machine_1	localhost	5556

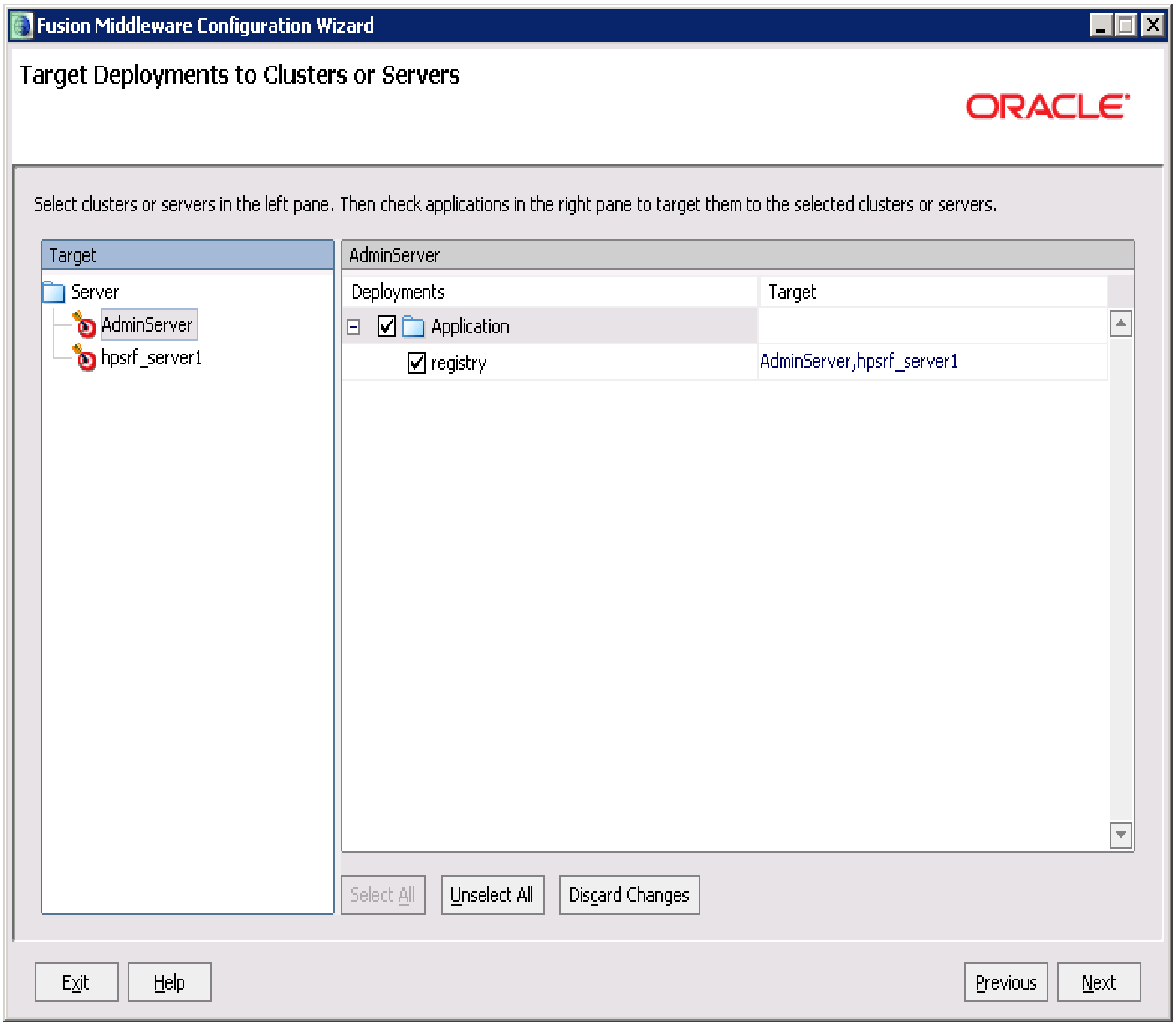
9. Assign servers to the machines.



Assign Servers to Machines

Select a machine in the right pane. Then select the server(s) in the left pane and assign them to the machine by clicking the right arrow button.

10. Select the clusters or servers.



Fusion Middleware Configuration Wizard
Target Deployments to Clusters or Servers
ORACLE

Select clusters or servers in the left pane. Then check applications in the right pane to target them to the selected clusters or servers.

Deployments	Target
⊟ ☑ Application	
☑ registry	AdminServer,hpsrf_server1

11. After configuring these environments you can extend the existing domain.

WebSphere

This process has been tested on WebSphere 6.1.x and 7.0.0.7

> **Note:** `REGISTRY_HOME` refers to the directory in which the HPE SOA Registry Foundation distribution is installed.
>
> `WEBSPHERE_HOME` refers to the directory in which IBM WebSphere is installed.

> PORTING_CONTEXT refers to context under which the HPE SOA Registry Foundation is deployed.

The REGISTRY_HOME/conf/porting/websphere/6.x/build/PORTING_CONTEXT.ear file is ready for deployment. Please follow these steps to complete the integration:

1. The IBM WebSphere server uses IBM java, which is installed in the WEBSPHERE_HOME/java directory. You must set up the security for this IBM JVM. To do so, follow the java security section in "System Requirements" on page 37.

   > **Note:** You should not download and replace the following security jars: US_ExportPolicy.jar and local_policy.jar

2. Modify the file WEBSPHERE_HOME/profiles/default/config/cells/DOMAIN_NAME/security.xml (for version 6.0) by adding the following lines between the tags <applicationLoginConfig> and </applicationLoginConfig>:

   Example: WebSphere Configuration

```
                   <entries xmi:id="WaspCredentials" alias="Credentials">
    <loginModules xmi:id="Credentials"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_1" name="debug" value="true"/>
       <options xmi:id="delegate_property_1" name="delegate"
          value="com.idoox.security.jaas.GSSLoginModule"/>
    </loginModules>
</entries>
<entries xmi:id="WaspReceivedCredentials" alias="ReceivedCredentials">
    <loginModules xmi:id="ReceivedCredentials"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_2" name="debug" value="true"/>
       <options xmi:id="delegate_property_2" name="delegate"
          value="com.idoox.security.jaas.GSSLoginModuleNoAuth"/>
    </loginModules>
</entries>
<entries xmi:id="WaspHttpCredentials" alias="HttpCredentials">
    <loginModules xmi:id="HttpCredentials"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_3" name="debug" value="true"/>
       <options xmi:id="delegate_property_3" name="delegate"
          value="com.idoox.security.jaas.HttpLoginModule"/>
    </loginModules>
</entries>
<entries xmi:id="WaspKrbCredentials" alias="KrbCredentials">
    <loginModules xmi:id="KrbCredentials"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_4" name="debug" value="false"/>
       <options xmi:id="krb_property_1" name="storeKey" value="true"/>
       <options xmi:id="delegate_property_4" name="delegate"
          value="com.sun.security.auth.module.Krb5LoginModule"/>
    </loginModules>
</entries>
<entries xmi:id="WaspCachedKrbCredentials" alias="CachedKrbCredentials">
    <loginModules xmi:id="CachedKrbCredentials"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_5" name="debug" value="false"/>
       <options xmi:id="krb_property_2" name="useTicketCache" value="true"/>
       <options xmi:id="delegate_property_5" name="delegate"
          value="com.sun.security.auth.module.Krb5LoginModule"/>
    </loginModules>
</entries>
<entries xmi:id="WaspNamePasswordNoAN" alias="NamePasswordNoAN">
    <loginModules xmi:id="NamePasswordNoAN"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_6" name="debug" value="true"/>
       <options xmi:id="delegate_property_6" name="delegate"
          value="com.idoox.security.jaas.NamePasswordLoginModuleNoAuth"/>
    </loginModules>
</entries>
<entries xmi:id="UDDINamePasswordAN" alias="NamePasswordAN">
    <loginModules xmi:id="NamePasswordAN"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_7" name="debug" value="true"/>
       <options xmi:id="delegate_property_7" name="delegate"
```

```
          value="com.systinet.uddi.security.jaas.NamePasswordLoginModule"/>
    </loginModules>
</entries>
<entries xmi:id="UDDIAuthTokenAN" alias="AuthTokenAN">
    <loginModules xmi:id="AuthTokenAN"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_8" name="debug" value="true"/>
       <options xmi:id="delegate_property_8" name="delegate"
          value="com.systinet.uddi.security.jaas.AuthTokenLoginModule"/>
    </loginModules>
</entries>
<entries xmi:id="WaspNameDigestAN" alias="NameDigestAN">
    <loginModules xmi:id="NameDigestAN"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
    <options xmi:id="debug_property_9" name="debug" value="true"/>
    <options xmi:id="delegate_property_9" name="delegate"
          value="com.idoox.security.jaas.NameDigestLoginModule"/>
    </loginModules>
</entries>
<entries xmi:id="WaspNameMapping" alias="NameMapping">
    <loginModules xmi:id="NameMapping"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_10" name="debug" value="true"/>
       <options xmi:id="delegate_property_10" name="delegate"
          value="com.idoox.security.jaas.NameLoginModuleNoAuth"/>
    </loginModules>
</entries>
<entries xmi:id="WaspCertsMapping" alias="CertsMapping">
    <loginModules xmi:id="CertsMapping"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_11" name="debug" value="true"/>
       <options xmi:id="delegate_property_11" name="delegate"
          value="com.idoox.security.jaas.CertsLoginModule"/>
    </loginModules>
</entries>
```

```
<entries xmi:id="HttpRequestMapping" alias="HttpRequest">
    <loginModules xmi:id="HttpRequest"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_12" name="debug" value="true"/>
       <options xmi:id="delegate_property_12" name="delegate"
          value="com.systinet.uddi.security.jaas.SmLoginModule"/>
    </loginModules>
</entries>
<entries xmi:id="RegistryIdentityAsserter" alias="IdentityAsserter">
    <loginModules xmi:id="IdentityAsserter"

moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProx
y"
          authenticationStrategy="REQUIRED">
       <options xmi:id="debug_property_13" name="debug" value="true"/>
       <options xmi:id="delegate_property_13" name="delegate"
          value="com.systinet.uddi.security.jaas.IdentityAsserterLoginModule"/>
    </loginModules>
</entries>
```

3. Deploy the file `REGISTRY_HOME/conf/porting/websphere/6.x/build/PORTING_CONTEXT.ear` file using the IBM WebSphere admin console, leaving all the options set at their default values.

4. After you finish the deployment, use WebSphere's admin console to set following properties. They are in "Class loading and update detection" section inside of enterprise application properties (in WebSphere 6.1).

   - mode of the WASP Application's classloader to 'PARENT_LAST' or "Classes loaded with application class loader first" option.
   - WAR Classloader Policy to 'Application' or "Single class loader for application" option

5. Import the SSL certificate of the Websphere server to the HPE SOA Registry configuration. Follow these steps:

   a. Obtain the WebSphere SSL certificate. There are two methods:

      i. You can get certificate using Internet Explorer 6.0 web browser connected to WebSphere via HTTPS. Select "Properties" in context menu of the page, button "Certificates", tab "Details", button "Copy to file", and then export certificate in Base 64 encoded X.509 .cer format.

      ii. You can also use `REGISTRY_HOME/bin/sslTool.sh` or `REGISTRY_HOME\bin\sslTool.bat` to get certificate. Run command:

```
sslTool serverInfo --url https://HOST:9043 --certFile websphere.cer
```

This command will connect to specified host and port using HTTPS and it will store server certificate into specified file.

b. Import this certificate using the PStoreTool located in `REGISTRY_HOME/bin`. The command follows (replace variables with real values):

```
PStoreTool add -config REGISTRY_HOME/conf/clientconf.xml -certFile
websphere.cer
```

HPE SOA Registry Foundation is now running on `http://<hostname>:9080/wasp/uddi/web`.

> **Note:**
>
> - The lines added to `login-config.xml` are an analogy of `jaas.config` expressed in XML.
> - The `PARENT_LAST` option and `Application` ClassLoader policy need to be set because there is a conflict between our implementations of the `saaj`, `jaxm`, `jaxrpc` and `wsdl` interfaces. `PARENT_LAST` assures that the servlet classloader is the first to be asked for the definition of classes.

JBoss

Tested on JBoss 4.3.0, 5.1 GA and JBoss EAP 5

> **Note:** `REGISTRY_HOME` refers to the directory in which the HPE SOA Registry Foundation distribution is installed.
>
> `JBOSS_HOME` refers to the directory in which JBoss is installed.

`REGISTRY_HOME/conf/porting/jboss/build/[context_name].war` is now ready for deployment. Please follow these steps to complete the integration:

1. Unpack the created file into the [context_name].war subdirectory of the JBoss deployment directory, which is usually `JBOSS_HOME/server/[jboss_configuration]/deploy`.
2. Modify the JBoss launch script (usually in JBOSS_HOME/bin/run.sh) as follows:
   a. Add the following jars to the beginning of the JBoss classpath:

   ```
   REGISTRY_HOME/lib/security-ng.jar
   ```

```
REGISTRY_HOME/conf/porting/dist/security3-ng.jar
JBOSS_HOME/server/[jboss_configuration]/lib/log4j.jar
```

3. Enable security: Add the following lines to the file `JBOSS_HOME/server/[jboss_configuration]/conf/loginconfig. xml` between the tags <policy>...</policy>:

Enabling Security - JBoss

```
<application-policy name="Credentials">
<authentication>
<login-module code="com.idoox.security.jaas.GSSLoginModule"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
</application-policy>
<application-policy name="ReceivedCredentials">
<authentication>
<login-module code="com.idoox.security.jaas.GSSLoginModuleNoAuth"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
</application-policy>
<application-policy name="HttpCredentials">
<authentication>
<login-module code="com.idoox.security.jaas.HttpLoginModule"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
</application-policy>
<application-policy name="NamePasswordNoAN">
```

```
<authentication>
<login-module code="com.idoox.security.jaas.NamePasswordLoginModuleNoAuth"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
</application-policy>
<application-policy name="NamePasswordAN">
<authentication>
<login-module code="com.systinet.uddi.security.jaas.NamePasswordLoginModule"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
</application-policy>
<application-policy name="NameDigestAN">
<authentication>
<login-module code="com.idoox.security.jaas.NameDigestLoginModule"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
</application-policy>
<application-policy name="NameMapping">
<authentication>
<login-module code="com.idoox.security.jaas.NameLoginModuleNoAuth"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
```

```
</application-policy>
<application-policy name="CertsMapping">
<authentication>
<login-module code="com.idoox.security.jaas.CertsLoginModule"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
</application-policy>
<application-policy name="AuthTokenAN">
<authentication>
<login-module code="com.systinet.uddi.security.jaas.AuthTokenLoginModule"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
</application-policy>
<application-policy name="HttpRequest">
<authentication>
<login-module code="com.systinet.uddi.security.jaas.SmLoginModule"
flag="required">
<module-option name = "debug">true</module-option>
</login-module>
</authentication>
</application-policy>
<application-policy name="IdentityAsserter">
<authentication>
<login-module
code="com.systinet.uddi.security.jaas.IdentityAsserterLoginModule"
flag="required">
<module-option name = "debug">true</module-option>
```

```
</login-module>
</authentication>
</application-policy>
```

4. Configure log4j for HPE SOA Registry: Add the following lines to the file `JBOSS_HOME/server/[jboss_configuration]/conf/jboss-log4j.xml` after the last tag </appender>:

   Example 4. Log4j Configuration - JBoss

   ```
   <!-- Registry log4j appenders -->
   <appender name="sr_eventLog" class="org.apache.log4j.RollingFileAppender">
   <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
   <param name="File"
   value="${jboss.server.home.dir}/log/HPSOARegistry_logEvents.log"/>
   <param name="MaxFileSize" value="10000KB"/>
   <param name="MaxBackupIndex" value="10"/>
   <layout class="org.apache.log4j.PatternLayout">
   <param name="ConversionPattern" value="(%d) - %m%n"/>
   </layout>
   </appender>
   <appender name="sr_errorLog" class="org.apache.log4j.RollingFileAppender">
   <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
   <param name="File"
   value="${jboss.server.home.dir}/log/HPSOARegistry_errorEvents.log"/>
   <param name="MaxFileSize" value="10000KB"/>
   <param name="MaxBackupIndex" value="10"/>
   <layout class="org.apache.log4j.PatternLayout">
   <param name="ConversionPattern" value="(%d) - %m%n"/>
   </layout>
   </appender>
   <appender name="sr_clusterLog" class="org.apache.log4j.RollingFileAppender">
   <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
   <param name="File"
   ```

```
value="${jboss.server.home.dir}/log/HPSOARegistry_configuratorEvents.log"/>

<param name="MaxFileSize" value="10000KB"/>

<param name="MaxBackupIndex" value="10"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="(%d) - %m%n"/>

</layout>

</appender>

<appender name="sr_replicationLog"
class="org.apache.log4j.RollingFileAppender">

<errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>

<param name="File"

value="${jboss.server.home.dir}/log/HPSOARegistry_replicationEvents.log"/>

<param name="MaxFileSize" value="10000KB"/>

<param name="MaxBackupIndex" value="10"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="(%d) - %m%n"/>

</layout>

</appender>

<appender name="sr_notificationLog"
class="org.apache.log4j.RollingFileAppender">

<errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>

<param name="File"

value="${jboss.server.home.dir}/log/HPSOARegistry_notificationEvents.log"/>

<param name="MaxFileSize" value="10000KB"/>

<param name="MaxBackupIndex" value="10"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="(%d) - %m%n"/>

</layout>

</appender>

<!-- Registry log4j categories -->

<category name="com.idoox.wasp.server.adaptor.RawAdaptor" additivity="false">
```

```
<priority value="ERROR"/>
</category>
<category name="com.systinet.wasp.events" additivity="false">
<priority value="INFO"/>
<appender-ref ref="sr_eventLog"/>
</category>
<category name="com.systinet.wasp.errors" additivity="false">
<priority value="ERROR"/>
<appender-ref ref="sr_errorLog"/>
</category>
<category name="org.apache.xml.security" additivity="true">
<priority value="ERROR"/>
</category>
<category
name="configurator.com.systinet.uddi.configurator.cluster.ConfiguratorManagerAp
iImpl"
additivity="false">
<priority value="INFO"/>
<appender-ref ref="sr_clusterLog"/>
</category>
<category name="replication_v3.com.systinet.uddi.replication.v3.ReplicatorTask"
additivity="false">
<priority value="DEBUG"/>
<appender-ref ref="sr_replicationLog"/>
</category>
<category name="uddi_subscription_v3.com.systinet.uddi.subscription.v3"
additivity="false">
<priority value="DEBUG"/>
<appender-ref ref="sr_notificationLog"/>
</category>
```

5. If you do not have SSL keys and certificate, generate them using the keytool from the JDK distribution as follows:

- Change the directory to the `bin` subdirectory of `JBOSS_HOME` and enter the following command:

  keytool *-keystore JBOSS_HOME/server/[jboss_configuration]/conf/server.keystore -genkey -alias jboss -keyalg RSA -storepass changeit*

- Change your directory to the `bin` subdirectory of `REGISTRY_HOME`.

- Export the certificate to a file using the following command:

  keytool `-keystore JBOSS_HOME/server/[jboss_configuration]/conf/server.keystore -export -file jboss.crt -alias jboss -storepass changeit`

- Import the certificate to `clientconf.xml` in the HPE SOA Registry Foundation distribution using this command:

  PStoreTool.sh (bat) `add -certFile jboss.crt -alias jboss -config REGISTRY_HOME/conf/clientconf.xml`

6. Enable SSL in JBoss.
   - JBoss 4.x: Uncomment the following lines in the file

     ```
     JBOSS_HOME/server/[jboss_configuration]/deploy/jboss-web.deployer/server.xml

     <Connector port="8443" address="${jboss.bind.address}"

     maxThreads="100" strategy="ms" maxHttpHeaderSize="8192"

     emptySessionPath="true"

     scheme="https" secure="true" clientAuth="false"

     keystoreFile="${jboss.server.home.dir}/conf/server.keystore"

     keystorePass="123456" sslProtocol = "TLS" />
     ```

     Change the values of keystoreFile to ${jboss.server.home.dir}/conf/server.keystore and keystorePass to

     changeit.

     > **Note:** Use the actual values you used when invoking the keytool utility if those values differ from the values shown here.

   - JBoss 5.x: Uncomment and edit the following lines in the file

     `JBOSS_HOME/server/[jboss_configuration]/deploy/jbossweb.sar/server.xml`

```
<Connector protocol="org.apache.coyote.http11.Http11Protocol"
SSLEnabled="true"

port="8443" address="${jboss.bind.address}"

maxThreads="100" strategy="ms" maxHttpHeaderSize="8192"

emptySessionPath="true"

scheme="https" secure="true" clientAuth="false"

keystoreFile="${jboss.server.home.dir}/conf/server.keystore"

keystorePass="123456" sslProtocol = "TLS" />
```

Change the values of `keystoreFile` to `${jboss.server.home.dir}/conf/server.keystore` and `keystorePass` to `changeit`.

7. (Re)start the JBoss server

   Installation should be complete. The HPE SOA Registry Foundation URL is

   `http://hostname:8080/[context_name]/uddi/web`.

   > **Note:** The lines added to `login-config.xml` are an analogy of `jaas.config` expressed in XML.

Cluster Configuration

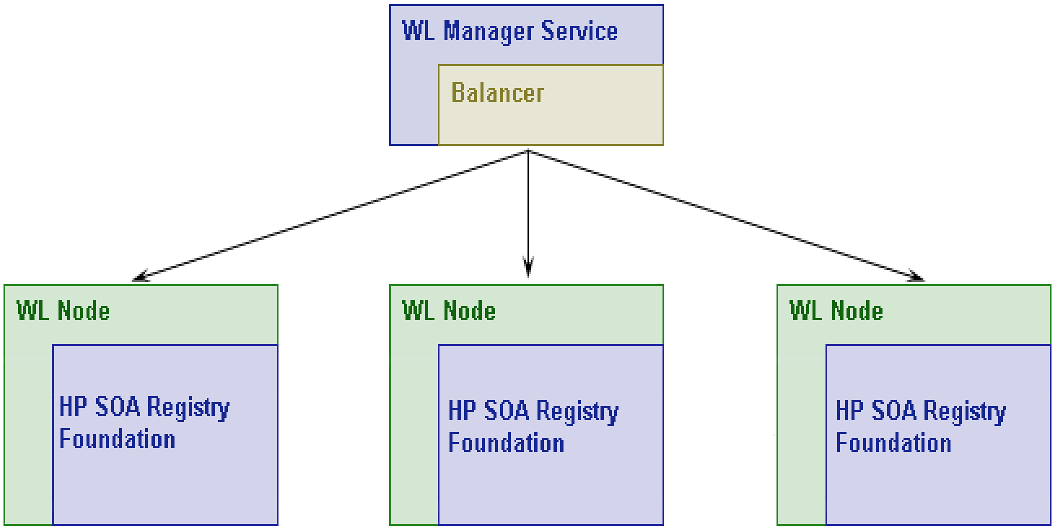
This chapter contains general notes about the synchronized configuration of a HPE SOA Registry Foundation cluster and gives instructions on how to deploy HPE SOA Registry Foundation to a WebLogic Cluster ("WebLogic Specific Configuration for Use with Cluster" on page 162).

Cluster operation

Cluster operation is achieved by running multiple registries and joining their functionality with a load balancer (proxy).

Load balancing is used to distribute requests among registries to get the optimal load distribution. The load balancer should be configured to distribute requests among all physical endpoints of the registry nodes. If using an application server, refer to its documentation for details about configuring load balancing.

HPE SOA Registry Foundation in WebLogic Cluster


WL Manager Service
Balancer
WL Node
HP SOA Registry Foundation
WL Node
HP SOA Registry Foundation
WL Node
HP SOA Registry Foundation

Clients to HPE SOA Registry Foundation access TCP ports on the balancer which forwards the connection to a running cluster node with an actual HPE SOA Registry Foundation. Each HPE SOA Registry Foundation has a connection to a common database so that each HPE SOA Registry Foundation has access to the latest data. This connection also serves as a distribution point for changed configurations and inter-node events.

When a HPE SOA Registry Foundation node fails (there are various reasons for this such as hardware problems, network connection problems or software failure), other nodes can work without it. The intelligent load balancer will detect this and further requests will not be directed there until the node starts to respond.

Every node has a Node ID - a string identifying the node. Each node should have a different ID. Breaking this rule will cause nodes with the same ID to miss some configuration changes and synchronization events.

Node ID can be specified by the administrator in the REGISTRY_HOME\app\uddi\conf\nodeid.xml file. If it is not specified before the initial start of HPE SOA Registry Foundation, it will be generated as a unique UUID string. It is possible to change it later, but node-local configurations under the old ID will be left in the database. Ensure that EAR/WAR file generated for deployment has either:

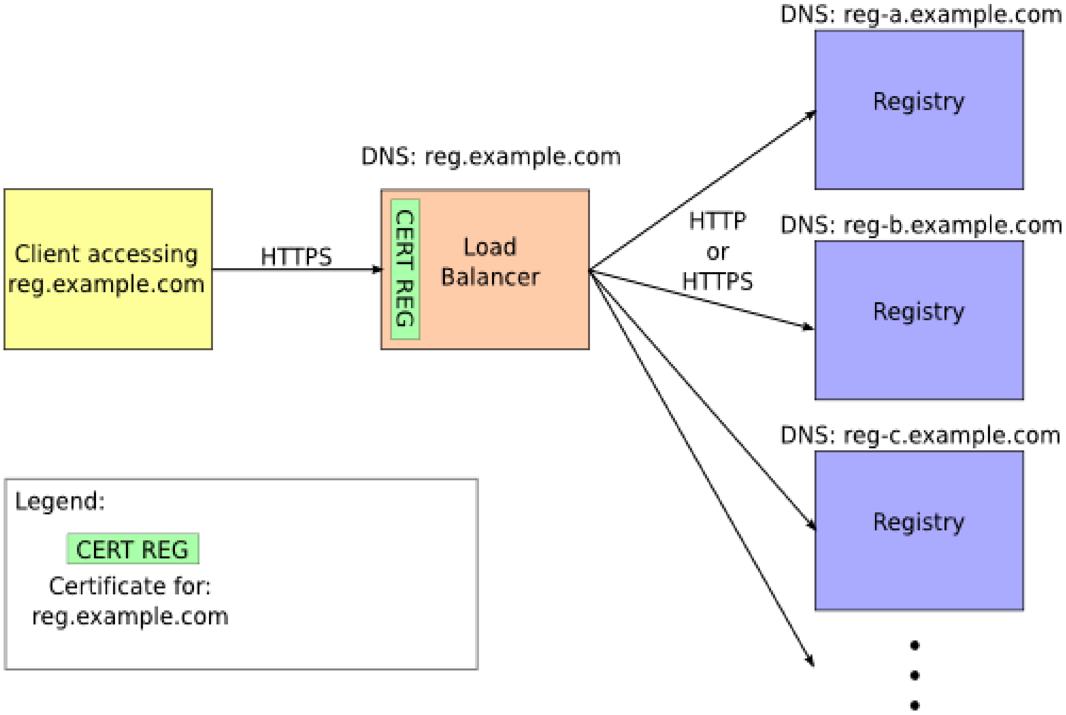
1. Empty Node ID - so that each deployment of the file will generate a unique Node ID on first run and will retain it until deletion or redeployment of EAR/WAR. You can use such EAR/WAR to deploy on all nodes.
2. Specified Node ID - when you deploy the EAR/WAR file to a single node and generate another EAR/WAR file for others. You can choose meaningful names for Node ID this way.

You can set the Node ID in the nodeid.xml file before starting setup to generate EAR/WAR file. If you use generation of EAR/WAR file directly from installer the Node ID will be empty.

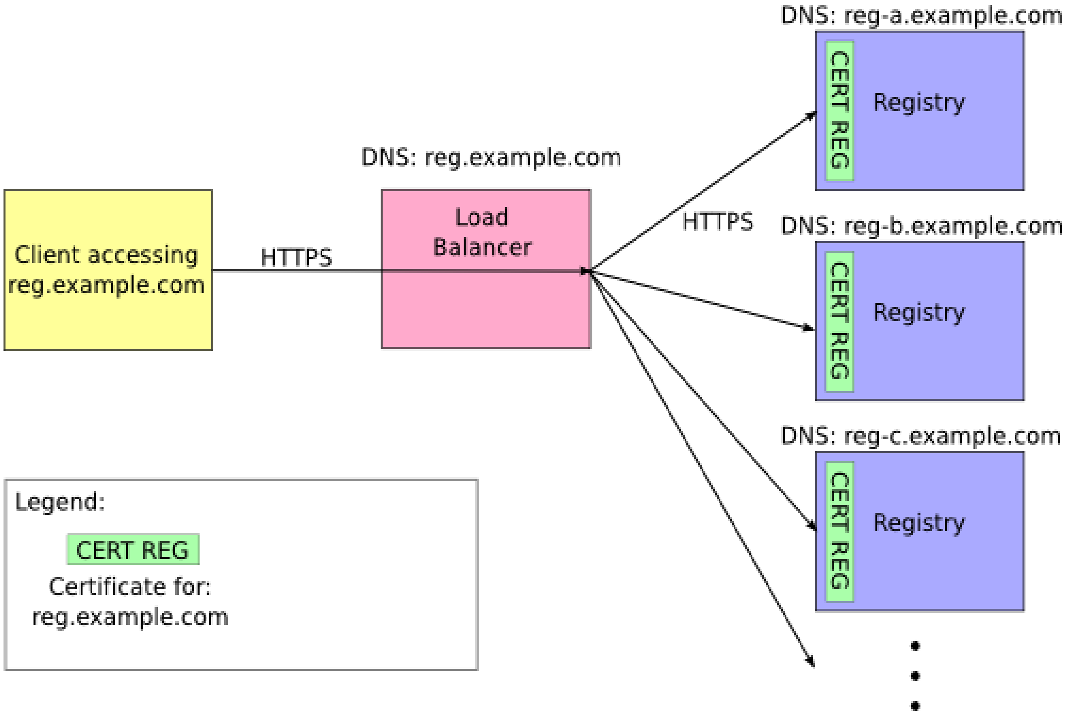
> **Note:** Latest configurations are identified by internal index sequencing. Time stamps of configurations as displayed in configuration management UI are not relevant as they may be unreliable in case of clock skew on a cluster node.

Cluster operation is affected by the interaction of connection security (HTTPS) and the load balancer. For security reasons, client access is done using the HTTPS protocol. This protocol requires that there is a valid and matching security certificate on the server side (possibly on the client side too if client authentication is required). There are generally two methods for achieving clustered operation via independent load balancer. If you deploy on an application server it may provide an integrated load balancer for you which may be easier to configure than an independent load balancer.

1. Secure connection can take a place between a client and the load balancer. The load balancer would be the end point for the secure connection which originated at the client. The load balancer will make an independent connection to some of the HPE SOA Registry Foundation nodes. This connection may be either in HTTP or HTTPS. The certificate which the client checks has to be placed on the load balancer. A connection between the load balancer and each HPE SOA Registry Foundation can be protected by HTTPS in which case the load balancer and the registries should know each others certificates.



2. Secure connection can be passed by the load balancer and terminated at the cluster node. This case requires that the certificates on all the nodes be the same to provide the illusion of a single service. However the common name inside the certificate should specify the DNS name of the balancer.



> **Note:** Load balancer is not part of HPE SOA Registry Foundation product. You can use almost any HTTP/HTTPS load balancer that supports the described configurations.

Most of the Client - HPE SOA Registry Foundation interactions require an authentication token to be passed along the way. This token is encrypted by the HPE SOA Registry Foundation certificate. Therefore each HPE SOA Registry Foundation behind the balancer has to have the same certificate.

WEB interfaces of HPE SOA Registry Foundation (Registry Console) need to know the absolute HTTP addresses of themselves. This address in the cluster is the address of the load balancer and the possible context under which it is deployed. This address can be changed during setup.

Cluster installation

Cluster installation requires the setup of a load balancer and multiple registries. These steps are recommended on the HPE SOA Registry Foundation side when an application server is used:

1. Install HPE SOA Registry Foundation.
   a. Fill-in the hostname and ports of the load balancer.
2. Port HPE SOA Registry Foundation via the Deploy option in the HPE SOA Registry Foundation Setup program (or directly in Installer program).
3. Deploy the generated WAR or EAR to all cluster nodes via the application server.

These steps are recommended on the HPE SOA Registry Foundation side where multiple standalone instances of HPE SOA Registry Foundation are used:

1. Install the first HPE SOA Registry Foundation.
   a. Fill-in the hostname and ports of the load balancer.
2. Setup SSL certificates as required in the first HPE SOA Registry Foundation.
3. Install other Registries.
   - Do not create new databases, just connect to the database of first HPE SOA Registry Foundation.
   - Copy `REGISTRY_HOME\conf\pstore.xml` from the first registry to each HPE SOA Registry Foundation. This assures that each HPE SOA Registry Foundation will have the same identity with respect to authentication tokens.
   - Copy the configuration files in the `REGISTRY_HOME\app\uddi\conf\` directory from the first HPE SOA Registry Foundation. This is required because some fields in the configuration files are coded by a key specified in application_core.xml. Failure to do so may result in error messages during startup and inconsistent configuration data in the database.
4. Run the first installed HPE SOA Registry Foundation first so that its configuration files are stored in database first. The next time you can run the Registries in any order (including the first one).

Setting Up Security

If using a cluster of standalone registries, they must share the same private key for validating authentication tokens.

Sharing Token Key

If HPE SOA Registry Foundation is installed as a cluster of standalone registries, you must ensure that all cluster nodes share the same private key for checking authentication token validity. (By a standalone registry, we mean HPE SOA Registry Foundation that is not deployed to an application server. You do not need to do this if HPE SOA Registry Foundation is deployed to an application server). To set this up, choose one of the cluster nodes and copy its private key to all other nodes in the cluster by entering this command at a command prompt:

```
PStoreTool copy -alias authTokenIdentity -keyPassword SSL_CERTIFICATE_PASSWORD -
config REGISTRY_HOME\conf\pstore.xml -config2 TARGET_REGISTRY_HOME\conf\pstore.xml
```

`SSL_CERTIFICATE_PASSWORD` is a ssl certificate password entered during the installation

`TARGET_REGISTRY_HOME` is the directory where a cluster node is installed.

WebLogic Specific Configuration for Use with Cluster

This section will guide you through an example setup of clustering with a WebLogic application server.

To deploy HPE SOA Registry Foundation to a WebLogic cluster follow these steps:

1. Install WebLogic, then configure it by adding machines to the cluster. In our case, the cluster is named *`cluster`* and is running on *`10.0.0.79`*. The nodes in the WebLogic cluster are named:
   - *`kila`* (10.0.0.79), running on *`kila.mycompany.com`*, with an http port of 7101 and https port of 7102
   - *`fido`* (10.0.0.134), running on *`fido.mycompany.com`*, with an http port of 7101 and https port of 7102
2. Generate the certificates of all cluster nodes: Let's create proper certificates for our two nodes. It will be done via the CertGen tool provided by WebLogic. Go to the directory `%WEB_LOGIC_HOME%\weblogic81\server\lib`. CertGen is located in `weblogic.jar`'s `utils` package. Invoke it with the command:

```
java -cp weblogic.jar utils.CertGen changeit kilacert kilakey export
kila.mycompany.com
```

The output resembles the following:

```
kilacert kilakey export kila.mycompany.com
...... Will generate certificate signed by CA from CertGenCA.der file
...... With Export Key Strength
...... Common Name will have Host name kila.mycompany.com
...... Issuer CA name is
CN=CertGenCAB,OU=FOR TESTING ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US
```

Use the password **changeit** for starting the UDDI node servers. The output file with the certificate is **kilacert**, and **kilakey** is the output file containing the private key. Generate certificates for all remaining nodes from their CertGen tools. (In our case, the other node is `fido.mycompany.com`.)

3. Once you have certificates from all nodes (in our case files `kilacert.der` and `fidocert.der`), import them to pstore.xml using the PstoreTool. Also include `CertGenCA.der` (from the directory `%WEB_LOGIC_HOME%\weblogic81\server\lib`). The `pstore.xml` file is now ready. For more info about WebLogic certificates and SSL settings, please see Configuring SSL in BEA's WebLogic product documentation.

4. Prepare a registry deployment package (`REGISTRY_HOME\conf\porting\weblogic\registry.war`) as described in “Deployment to an Application Server”.

   In our case, the http port is ***7101***, the https port is ***7102***, and the application server context is **wasp**.

5. Check that the paths for `log4j.appender.eventLog.File`, `log4j.appender.errorLog.File`, and `registry.war\conf\log4j.config` are valid on all cluster nodes.

6. Deploy `registry.war` into all WebLogic cluster nodes.

You must also prepare the package for the balancer which will only be deployed to the cluster manager server. To do so:

1. Create a balancer directory, in, for example, `REGISTRY_HOME`. This directory is referenced in this section as `PACKAGE_HOME`.

2. Create a subdirectory of `PACKAGE_HOME` named `WEB-INF`.

3. In this subdirectory, create the file `web.xml` containing the following text. Under `WebLogicCluster` specify the names and ports of your cluster nodes separated by a pipe (|). In our case, the file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN"
                                       "http://java.sun.com/dtd/web-app_2_
3.dtd">
<web-app>
   <servlet>
   <servlet-name>HttpClusterServlet</servlet-name>
   <servlet-class>weblogic.servlet.proxy.HttpClusterServlet</servlet-class>
   <init-param>
       <param-name>WebLogicCluster</param-name>
       <param-value>kila:7101|fido:7101</param-value>
   </init-param>
   </servlet>

   <servlet>
       <servlet-name>FileServlet</servlet-name>
       <servlet-class>weblogic.servlet.FileServlet</servlet-class>
   </servlet>

   <servlet-mapping>
       <servlet-name>FileServlet</servlet-name>
       <url-pattern>/uddi/webdata*</url-pattern>
   </servlet-mapping>

   <servlet-mapping>
       <servlet-name>HttpClusterServlet</servlet-name>
       <url-pattern>/</url-pattern>
   </servlet-mapping>
</web-app>
```

4. In the WEB-INF subdirectory, create the file weblogic.xml containing the following text, where /wasp is the context of HPE SOA Registry Foundation deployed to this application server. Your text must be customized for your own installation.

```
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web Application
8.1//EN"
"http://www.bea.com/servers/wls810/dtd/weblogic810-web-jar.dtd">
<weblogic-web-app>
   <context-root>/wasp</context-root>
</weblogic-web-app>
```

5. Create the directory %PACKAGE_HOME%\uddi\webdata.

6. Unjar `REGISTRY_HOME\app\uddi\web.jar` and copy the content of the webroot subdirectory from the jar to `%PACKAGE_HOME%\uddi\webdata`.

7. Package the content of `%PACKAGE_HOME%` into the file `balancer.war` using jar or some other compression utility.

8. Deploy balancer.war into the cluster manager server.

Authentication Configuration

In this section, we will show you how to change the HPE SOA Registry Foundation configuration to allow the following authentication options:

- "HTTP Basic" below
- "Netegrity SiteMinder" on page 167
- "SSL Client authentication" on page 169
- "J2EE Server Authentication" on page 172
- "Internal SSL Client Authentication Mapping in J2EE" on page 173
- "Disabling Normal Authentication" on page 174
- "Consoles Configuration" on page 175
- "Outgoing Connections Protected with SSL Client Authentication" on page 176

HTTP Basic

To allow HTTP Basic authentication:

> **Note:** In case Registry is deployed to Oracle WebLogic Server
>
> - Add the `<enforce-valid-basic-auth-credentials>false</enforce-valid-basic-auth-credentials>` element to `config.xml` in directory [domain]\config within the `<security-configuration>` element.
> - The enforce-valid-basic-auth-credentials flag effects the entire domain. Client requests that use HTTP BASIC authentication will be ignored by WebLogic Server authentication.

1. 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"/>`. This enables the HTTP Basic authentication for UDDI Publishing API v1, v2, v3.

b. Under `<processing name="UDDIv1v2v3InquiryProcessing">`, add `<use ref="tns:HttpBasicInterceptor"/>`. This enables the HTTP Basic authentication for all three versions of the UDDI Inquiry API.

c. Under `<processing name="wsdl2uddiProcessing">`, add `<use ref="tns:HttpBasicInterceptor"/>` . This enables the HTTP Basic authentication for versions 2 and 3 of the WSDL2UDDI API.

d. Add the attribute accepting-security-providers="HttpBasic" to other service-endpoints (except UDDI publishing and Inquiry endpoint) you wish to access via HTTP Basic authentication.

   A fragment of the `package.xml` is shown below in Example 5, “package.xml - HTTP Basic Enabled”

2. Shutdown HPE SOA Registry Foundation, delete the `REGISTRY_HOME/work` directory, and restart the registry.

Example: 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.MessageTooLargeFaultSerialize
r"
       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>
.....
```

Netegrity SiteMinder

To allow Netegrity SiteMinder authentication:

1. Modify `REGISTRY_HOME/app/uddi/services/Wasp-inf/package.xml` as follows:

   a. Under `<processing name="UDDIv1v2v3PublishingProcessing"/>`, add `<use ref="tns:SiteMinderInterceptor"/>`. This enables the SiteMinder authentication for all three versions of the UDDI Publishing API.

   b. Under `<processing name="UDDIv1v2v3InquiryProcessing">`, add `<use ref="tns:SiteMinderInterceptor"/>`. This enables the SiteMinder authentication for versions 1, 2, and 3 of the Inquiry API.

   c. Under `<processing name="wsdl2uddiProcessing">`, add `<use ref="tns:SiteMinderInterceptor"/>` . This enables the SiteMinder authentication for versions 2 and 3 of the WSDL2UDDI API.

d. Add the attribute accepting-security-providers="Siteminder" to other service-endpoints (except UDDI publishing and Inquiry endpoint) you wish to access via Netegrity SiteMinder authentication.

e. Under the elements `<securityProviderPreferences>` and `<interceptor name="SiteMinderInterceptor"`, fill in:

- `<loginNameHeader>` - login name header
- `<groupHeader>` - group header
- `<delimiter>` - group name delimiter.

> **Note:** You must set the same element values to both <securityProviderPreferences> and <interceptor name="SiteMinderInterceptor" elements.

A fragment of the `package.xml` is shown in Example 6 below.

2. Shutdown HPE SOA Registry Foundation, delete the REGISTRY_HOME/work directory, and restart the registry.

Example 6. package.xml - Netegrity SiteMinder Enabled

```
.....
   <!-- Netegrity SiteMinded security provider preferences for the server side
-->
      <securityProviderPreferences
xmlns="http://systinet.com/wasp/package/extension"
         name="Siteminder">
           <loginNameHeader>sm-userdn</loginNameHeader>
           <groupHeader>sm-role</groupHeader>
           <delimiter>^</delimiter>
      </securityProviderPreferences>

   <!-- Netegrity SiteMinded interceptor-->
   <interceptor name="SiteMinderInterceptor"
             implementation-
class="com.systinet.uddi.security.siteminder.SmInterceptor" >
           <config:loginNameHeader>sm-userdn</config:loginNameHeader>
           <config:groupHeader>sm-role</config:groupHeader>
           <config:delimiter>^</config:delimiter>
   </interceptor>
.....
```

SSL Client authentication

Standalone registry can be configured to perform authentication using client certificate obtained via 2-way SSL, where also the client has to authenticate itself to a server. Setup instructions differs for a standalone and a deployed registry. This section is focused on a standalone registry. See "J2EE Server Authentication" on page 172 for instructions about configuring SSL client authentication for a deployed registry.

To allow SSL client authentication for a standalone registry:

1. Make sure that the registry is not running.
2. Modify `REGISTRY_HOME/conf/serverconf.xml` as follows:
   - Under `<httpsPreferences name="https">`, change `<needsClientAuth>` to true. This will setup HTTPS transport to require client certificates.
   - Under `<securityPreferences name="main">`, add `<acceptingSecurityProvider>SSL</acceptingSecurityProvider>`. This will turn on mapping of client certificates to a user name.

   A fragment of changed `REGISTRY_HOME/conf/serverconf.xml` is shown in Example 7 below, “A fragment of serverconf.xml with 2-way SSL turned on”.
3. Trust the certificate of a certification authority that is used to issue client certificates. Run the `PStoreTool` tool from the `REGISTRY_HOME/bin` directory to import this certificate to a truststore that is used by registry.

   `PStoreTool add -certFile <client certificates authority certificate file> -config <path to pstore.xml>`
4. Configure a way how a client certificate is mapped to a user name. Registry comes with JAAS login module that extracts the user name out of a subject that is necessary part of a client certificate. The login module that performs this mapping is configured under the `CertsMapping` entry of the `REGISTRY_HOME/conf/jaas.conf` file. An example of `CertsMapping` entry is shown in Example 8 below, “CertsMapping JAAS configuration”.

   You can configure the following options:
   - `debug` - if set it to true, debug actions of the login module are printed to error stream. False by default.
   - `issuer` - issuer name, recommended to set. If set, mapped certificate must be issued by a

certification authority with this subject name.

- `pattern` - regular expression (as per java.util.regexp) that is used to get user name. The first capturing group of a specified pattern is used as a user name. When there is no capturing group and the pattern matches, the whole subject becomes a user name. Used regular expressions are case-insensitive. Examples are:

    - The default is `(?<!\\,\s?)EMAILADDRESS=(.+)@`. It matches a name listed in EMAILADDRESS. This regular expression ignores the case of EMAILADDRESS possibly contained in another part of subject.
    - `CN=([^,]+)` matches common name.
    - `.*` matches every subject. Since it has no capturing group, the whole subject DN is used.

  You can configure more than one login module to perform certificate mapping. This is usefull when you have to accept different issuers and/or provide a fallback to a failed certificate mapping of the first configured login module. An example of a `CertsMapping` entry that allows to map certificates issued by 2 issuers with a different way of mapping is shown in Example 9, “CertsMapping JAAS configuration with 2 possible issuers”.

5. Now the registry is configured for SSL client authentication. You may also change the applicability of SSL client authentication by changing configuration of SSL security provider. This configuration is in the `<securityProviderPreferences name="SSL">` element of the `REGISTRY_HOME/conf/serverconf.xml` file. An example is shown below in Example 7, “A fragment of serverconf.xml with 2-way SSL turned on”.

   Example 7. A fragment of serverconf.xml with 2-way turned on

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <config name="main">
       ...
      <securityPreferences name="main">
         <!-- Added acceptingSecurityProvider -->
       <acceptingSecurityProvider>SSL</acceptingSecurityProvider>
       <pstoreInitParams/>
       ...
      </securityPreferences>
   ...
   <httpsPreferences name="https">
       ...
       <!-- Client authentication required -->
       <needsClientAuth>true</needsClientAuth>
     ...
   </httpsPreferences>
   ...
   <!-- security provider preferences intended mainly for SSL client
   ```

```
authentication -->
<securityProviderPreferences name="SSL">
      <!-- What to do when SSL is not used to access the resource? Available
options:
      redirect
        - perform HTTP redirect to associated HTTPS URL (302 Moved Temporarily)
      fail
        - return a message that informs to use HTTPS URL (400 Bad Request)
      skip
        - do not perform certificate mapping at all
      perform
        - try to perform certificate mapping with no client certificates
      -->
      <whenNotSsl>skip</whenNotSsl>
      <!-- Can certificate mapping fail? If set to true and it fails, no
received subject will be constructed.
-->
      <certMappingMayFail>false</certMappingMayFail>
      <!-- Can a default account be created when no account for a mapped user
exists? -->
      <createDefaultAccount>false</createDefaultAccount>
  </securityProviderPreferences>
</config>
```

Example 8: CertsMapping JAAS configuration

```
CertsMapping{
  com.systinet.uddi.security.jaas.CertMappingLoginModule sufficient pattern="
(?<!\\,\s?)EMAILADDRESS=(.+)@"
  debug=false issuer="CN=Company CA, OU=mycomp";
};
```

Example 9: CertsMapping JAAS configuration with 2 possible issuers

```
CertsMapping{
  com.systinet.uddi.security.jaas.CertMappingLoginModule sufficient pattern="
(?<!\\,\s?)EMAILADDRESS=(.+)@"
  debug=false issuer="CN=Company CA, OU=mycomp";
  com.systinet.uddi.security.jaas.CertMappingLoginModule sufficient
pattern="CN=([^,]*)" issuer="CN=Company
  CA2, OU=mycomp" debug=false;
};
```

J2EE Server Authentication

The registry can be configured to let a J2EE application server perform authentication. Unlike "Netegrity SiteMinder" on page 167 and "HTTP Basic" on page 165, the authentication takes place for a whole registry application. To allow J2EE server authentication:

1. Create a deployment package using instructions provided in "Deployment to an Application Server" on page 120.
2. Modify `WEB-INF/web.xml` file of the resulted war file as follows:
   a. Change the value of context parameter use.request.user to true.

   b. Add a login-config element with a type of chosen J2EE authentication. Example 10, “A fragment of web.xml” below shows a login config that will turn on CLIENT-CERT authentication method, which is essentially used for SSL client authentication.

      You may also add security-constraint element to specify a set of resources where confidentially and/or integrity is required. Example 10, “A fragment of web.xml” below contains a security-constraint that requires confidential communication between client and server for all registry resources, which typically means to allow only HTTPS in the communication with registry.

   c. Configure a J2EE application server for an authentication method of your choice. For SSL client authentication, this typically means to setup HTTPS transport to require client certificates and to map client certificates to user name. Consult documentation of a target J2EE application server for details.
3. Go on with deployment of a modified war file.

   Example 10: A fragment of web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <display-name>Registry</display-name>
...
    <context-param>
        <param-name>use.request.user</param-name>
        <param-value>true</param-value>
    </context-param>
....
<!-- Added CLIENT-CERT authentication method -->
    <login-config>
      <auth-method>CLIENT-CERT</auth-method>
    </login-config>
<!-- Added security constraint that allow to access registry only via HTTPS -->
    <security-constraint>
       <display-name>HTTPS required to access registry</display-name>
       <web-resource-collection>
         <web-resource-name>Protected Area</web-resource-name>
         <url-pattern>/*</url-pattern>
         <http-method>DELETE</http-method>
         <http-method>GET</http-method>
         <http-method>POST</http-method>
         <http-method>PUT</http-method>
       </web-resource-collection>
       <user-data-constraint>
         <description>Require confidentiality</description>
         <transport-guarantee>CONFIDENTIAL</transport-guarantee>
     </user-data-constraint>
    </security-constraint>
</web-app>
```

Internal SSL Client Authentication Mapping in J2EE

While J2EE application authentication can be configured in many ways, some configurations can be cumbersome on some application servers. Internal SSL client authentication mapping can be easier to configure for simple deployments. This method has been tested on Tomcat 5.5 and JBOSS 4.0.5.

Internal client authentication mapping offers the same options for configuration as CertMapper described in "SSL Client authentication" on page 169. Installation steps:

1. Ensure that certificates are trusted by the J2EE server. Some servers have dedicated trust stores, while others use the `cacerts` java keystore file inside Java runtime. Add the certificate of the Certification Authority you are using to the server's trust store as a trusted certificate.

2. Set up your J2EE server SSL. You usually need to provide the Java trust store file with the server identity. Configure the server SSL to use the trust store by specifying file, alias and store password.

3. Set up your J2EE server to ask for or require Client Authentication.

4. Edit `web.xml` inside the deployed registry.

   - Change tag `servlet-class` to contain `com.systinet.transport.servlet.server.registry.RegistryServletTwoWaySSL`.

- Add the CLIENT-CERT authentification method (as seen below in Example 11, “A fragment of web.xml”).
- Add context parameters. Set the context parameter "twowayssl.use_user" to value "true".
- Set the context parameter "twowayssl.issuer" to the X.509 Issuer DN of certificates you want to allow.
- You can set the context parameter "twowayssl.mapping" to a regular expression for matching parts of Subject DN (by default, it is set to the name part of the email address in the email field).
- You can set the context parameter "twowayssl.debug" to "true" for run-time information about matching.

All context parameters that you set correspond to parameters in "SSL Client authentication" on page 169. For examples of these parameters, see Example 11, “A fragment of web.xml” below.

Example 10: A fragment of web.xml

```
<login-config>
   <auth-method>CLIENT-CERT</auth-method>
</login-config>

<context-param>
   <param-name>twowayssl.use_user</param-name>
   <param-value>true</param-value>
</context-param>
<context-param>
   <param-name>twowayssl.issuer</param-name>
   <param-value>C=CZ, ST=Czech, L=Prague, O=Example company, OU=Security Team,
CN=CA</param-value>
</context-param>
```

Disabling Normal Authentication

After you implement a custom authentication mechanism, such as a client SSL certificate, you may want to disable normal authentication. Disable normal authentication by removing permission for the `get_authToken` UDDI API from the `system#everyone` group. (The `get_authToken` API has this permission by default.)

To remove permission for the `get_authToken` UDDI API from the `system#everyone` group:

1. Log into the WEB UI using your administrative account and open the **Management** tab.
2. Open the **Permissions** page.

3. Select the **Group** radio button.
4. Edit the group `system#everyone` and remove the following permissions (Permission type / Api name / Actions):
   - org.systinet.uddi.security.permission.ApiUserPermission / org.systinet.uddi.client.v3.UDDI_Security_PortType / get_authToken,
   - org.systinet.uddi.security.permission.ApiUserPermission / org.systinet.uddi.client.v3.UDDI_Security_PortType / get_authToken,
   - org.systinet.uddi.security.permission.ApiUserPermission / org.systinet.uddi.client.v1.PublishSoap / get_authToken.

   Remember that you will not be able to log in to WEB user interfaces with the normal login dialog after you disable normal authentication.

Consoles Configuration

In this section, we will show you how to configure HTTP Basic or Netegrity Siteminder authentication for Registry Console. The configuration of consoles is very similar to the configuration of other endpoints.

> **Note: Referring to jar packages**
>
> The file path `REGISTRY_HOME/app/uddi/web.jar/WASP-INF/package.xml` means the `/WASP-INF/package.xml` inside the `jar` package `REGISTRY_HOME/app/uddi/web.jar`.

For the Registry Console, modify the file `REGISTRY_HOME/app/uddi/web.jar/WASP-INF/package.xml` with the following:

```
<service-endpoint path="/web" name="WebUIEndpoint1"

service-instance="tns:WebUI" type="raw" other-methods="get"

accepting-security-providers="HttpBasic"/>

<service-endpoint path="/web/*" name="WebUIEndpoint2"

service-instance="tns:WebUI" type="raw" other-methods="get"

accepting-security-providers="HttpBasic"/>
```

If you want to set Netegrity SiteMinder provider, use `accepting-security-providers="Siteminder"`

We just set authentication providers for both HTTP and HTTPS protocols. Now, we must specify which protocol consoles will be using for user authentication. The default registry configuration is to use

HTTP for browsing and searching. HTTPS is used for publishing. To avoid displaying the login dialog twice, (for the first time when accessing via HTTP then the second time when accessing via HTTPS), modify the configuration to use only one protocol.

For the Registry Console, modify url and secureUrl elements in the file `REGISTRY_HOME/app/uddi/conf/web.xml` to have the same value:

```
<url>https://servername:8443</url>
```

```
<secureUrl>https://servername:8443</secureUrl>
```

Outgoing Connections Protected with SSL Client Authentication

HPE SOA Registry Foundation can be the client in SSL Client Authentication. This allows the following scenarios:

- SOAP Client - This is commonly used in following scenarios
  - Approval process
  - Replications
  - Cluster

  Approval process, Replications, or Cluster functionality connects via SOAP endpoints. Deployment in those scenarios does not usually need SSL protection because all registries are located in a dedicated internal network, but HPE SOA Registry Foundation can be configured to use client SSL certificates in those scenarios. When registry on the other side is protected with Client SSL Authentication and plain HTTP connection is not allowed, your registry has to connect with an SSL Certificate. This can be achieved by configuring `destinationConfig` inside security.xml. See the documentation for sslTool in the Administration Guide, which describes the tool for SSL related tasks and `destinationConfig`. Destination config allows you to specify different certificates for different endpoints by either specifying the SOAP stub or the URL prefix.

- HTTPS protected resources
  - WSDL
  - XML
  - XSD
  - XSLT

Resources which are downloaded for processing by HPE SOA Registry Foundation can be behind HTTPS protected by Client SSL Authentication. HPE SOA Registry Foundation can be set up so that these connections use a specified certificate. The certificate has to be present as a key entry inside `pstore.xml`. This key entry is identified by its alias. The alias and password has to be specified in `REGISTRY_HOME/app/uddi/conf/security.xml` inside security which is contained in config as shown in example:

```
<sslConnectionAlias>myAliasName</sslConnectionAlias>

<sslConnectionPassword_coded>9vTJ9GKyjIURFY0qrWvADA==

</sslConnectionPassword_coded>
```

To get encoded password from clear-text password, use `REGISTRY_HOME/bin/sslTool(.bat or .sh)` with "encrypt" option.

Migration

Migration is used to migrate data from one database to another. You can migrate data during installation or during setup. Often users evaluate HPE SOA Registry Foundation using the preconfigured Hypersonic SQL database, and migrate data to another database after evaluation.

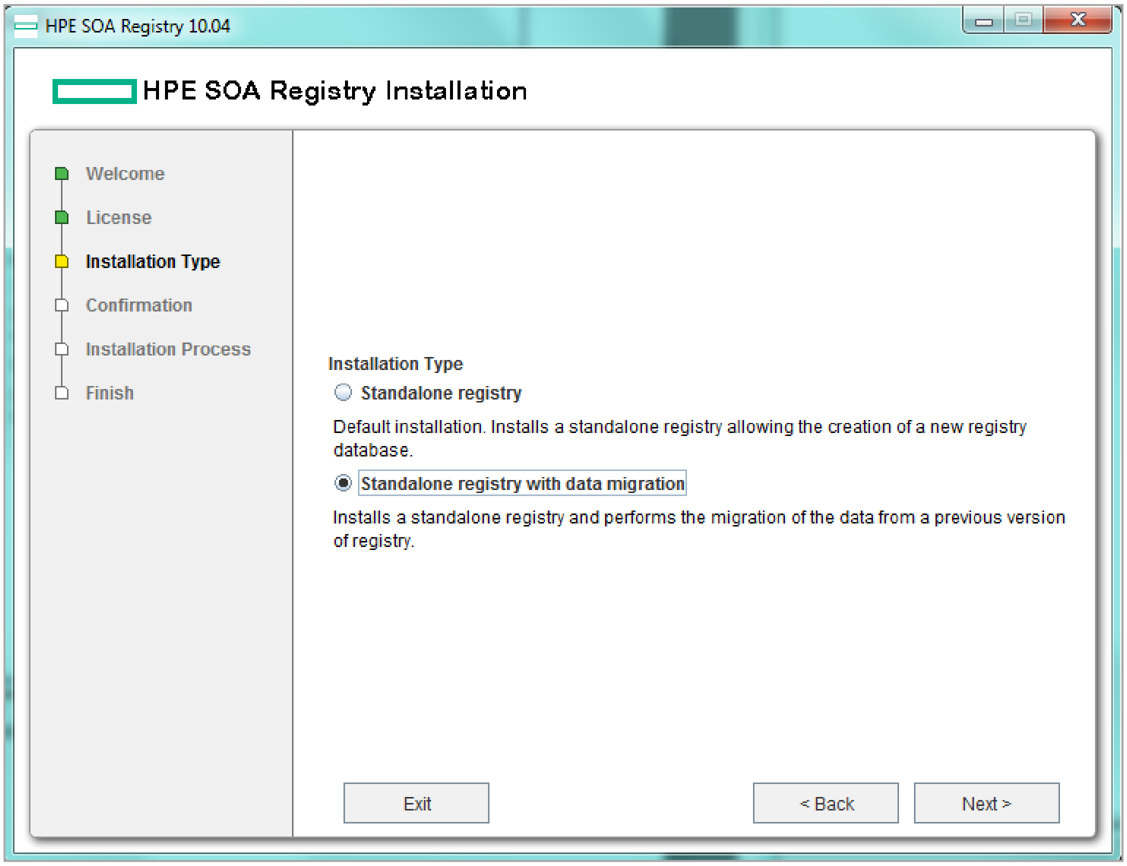
> **Note:** Demo data are not migrated. Internal UDDI data such as built-in T-Models are not migrated since they are available in any installation by default. The list of such skipped entities is inside `migration*.xml` in `app\uddi\conf` directory which you may view before migration if you use "Migration After Installation" on page 180.

Migration During Installation

To migrate data during installation:

1. Select **Standalone registry with data migration** as shown in the figure below.

   Standalone Installation with Migration



Installation Type

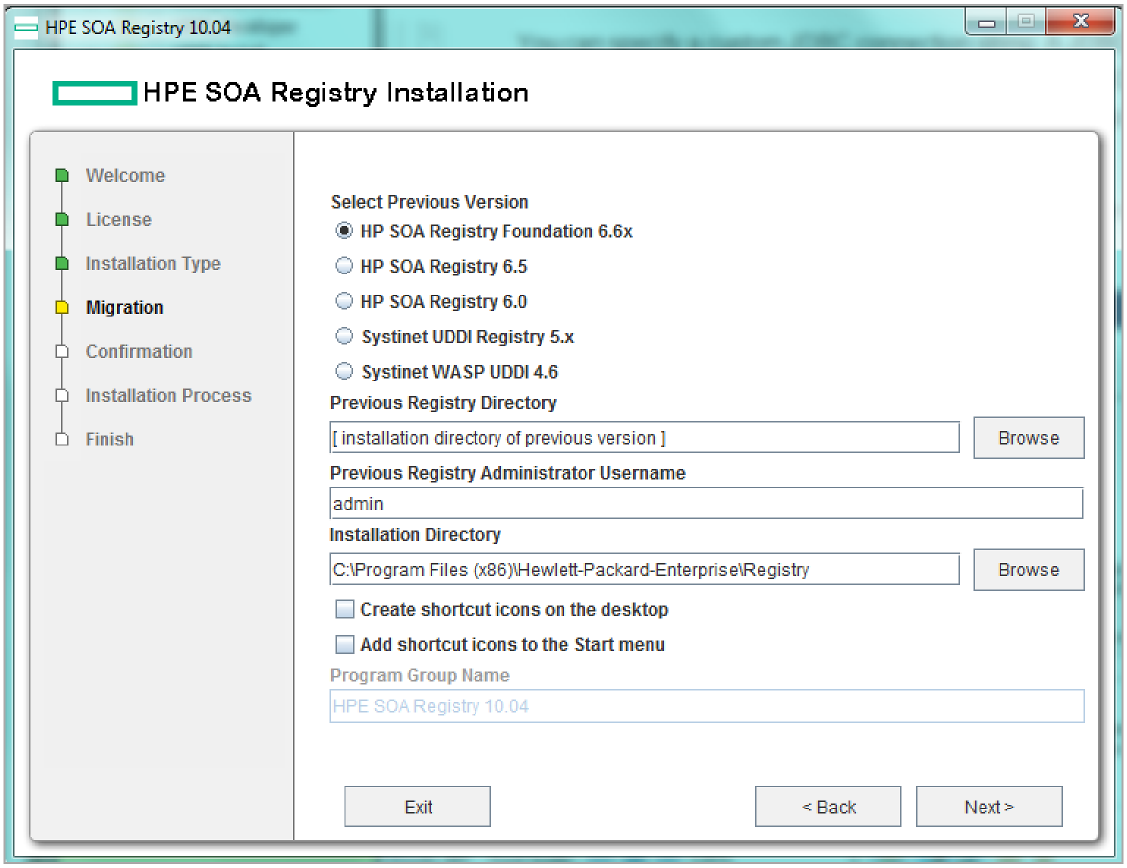
○ Standalone registry

Default installation. Installs a standalone registry allowing the creation of a new registry database.

● Standalone registry with data migration

Installs a standalone registry and performs the migration of the data from a previous version of registry.

2. Click **Next**. This returns the Migration panel shown below.



Select Previous Version

● HP SOA Registry Foundation 6.6x
○ HP SOA Registry 6.5
○ HP SOA Registry 6.0
○ Systinet UDDI Registry 5.x
○ Systinet WASP UDDI 4.6

Previous Registry Directory: [installation directory of previous version]

Previous Registry Administrator Username: admin

Installation Directory: C:\Program Files (x86)\Hewlett-Packard-Enterprise\Registry

□ Create shortcut icons on the desktop
□ Add shortcut icons to the Start menu

Program Group Name: HPE SOA Registry 10.04

3. Fill the following properties:

   - **Previous Registry Version** - HPE SOA Registry version from which you are migrating data
   - **Previous Registry Directory** - the directory containing the previous installation of HPE SOA Registry Foundation. The existing data will be migrated from it.
   - **Previous Registry Administrator Username** - name of the user having rights to retrieve data from the previous version Registry. By default, only administrator can migrate all data including private data.
   - **Installation directory** - select the directory where HPE SOA Registry Foundation will be installed.

4. Click **Next** and continue your Standalone installation as described in "Server Settings" on page 1. During the installation process, all data will be migrated from the specified previous HPE SOA Registry Foundation installation to the current installation.

Migration After Installation

> **Note:** Migration is additive. It does not delete entities that are already present in HPE SOA Registry Foundation and not present in migration source.

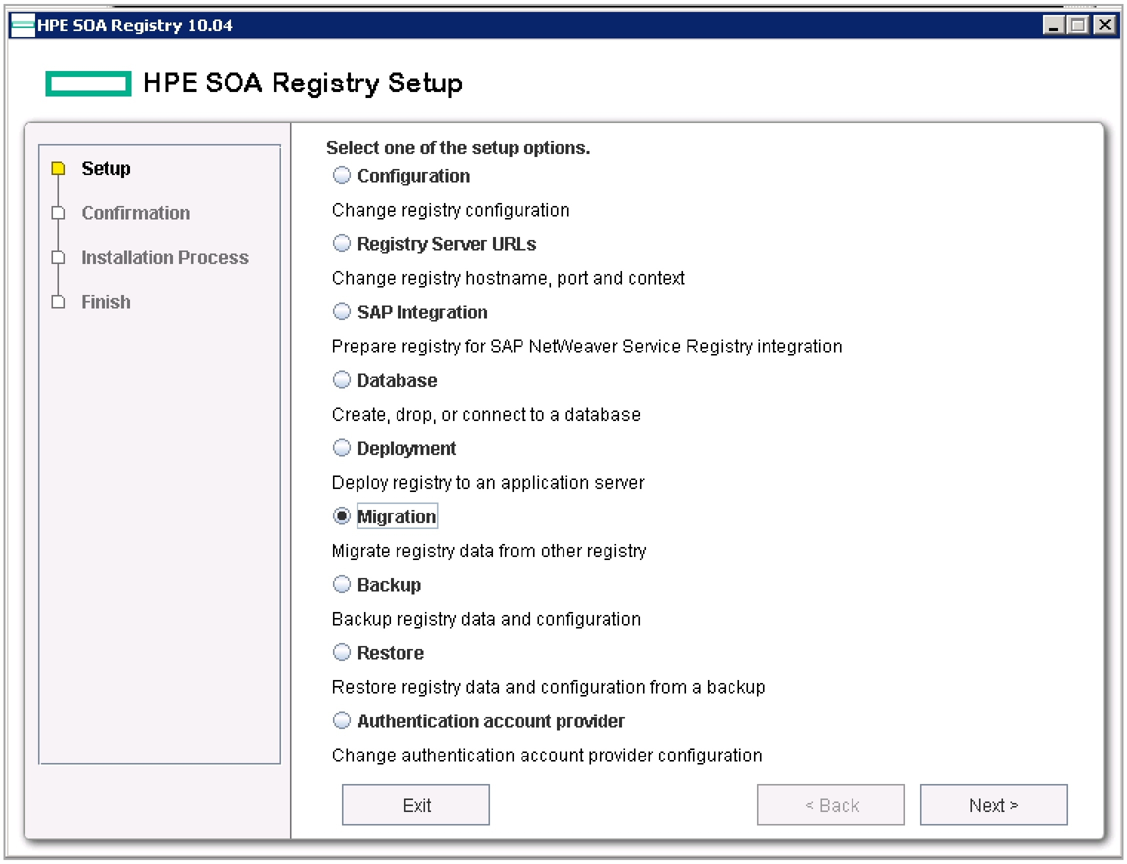
To migrate data after installation, use the Setup tool described in "Reconfiguring After Installation" on page 62. Briefly:

1. Launch the Setup tool by issuing the following command from the `bin` subdirectory of your installation:

Windows:	setup.bat
UNIX:	./setup.sh

   See command-line parameters in "Setup" in "Command Line Scripts" on page 59.

2. Select the Migration tool on first panel:



HPE SOA Registry 10.04

HPE SOA Registry Setup

Select one of the setup options.

○ Configuration

Change registry configuration

○ Registry Server URLs

Change registry hostname, port and context

○ SAP Integration

Prepare registry for SAP NetWeaver Service Registry integration

○ Database

Create, drop, or connect to a database

○ Deployment

Deploy registry to an application server

● Migration

Migrate registry data from other registry

○ Backup

Backup registry data and configuration

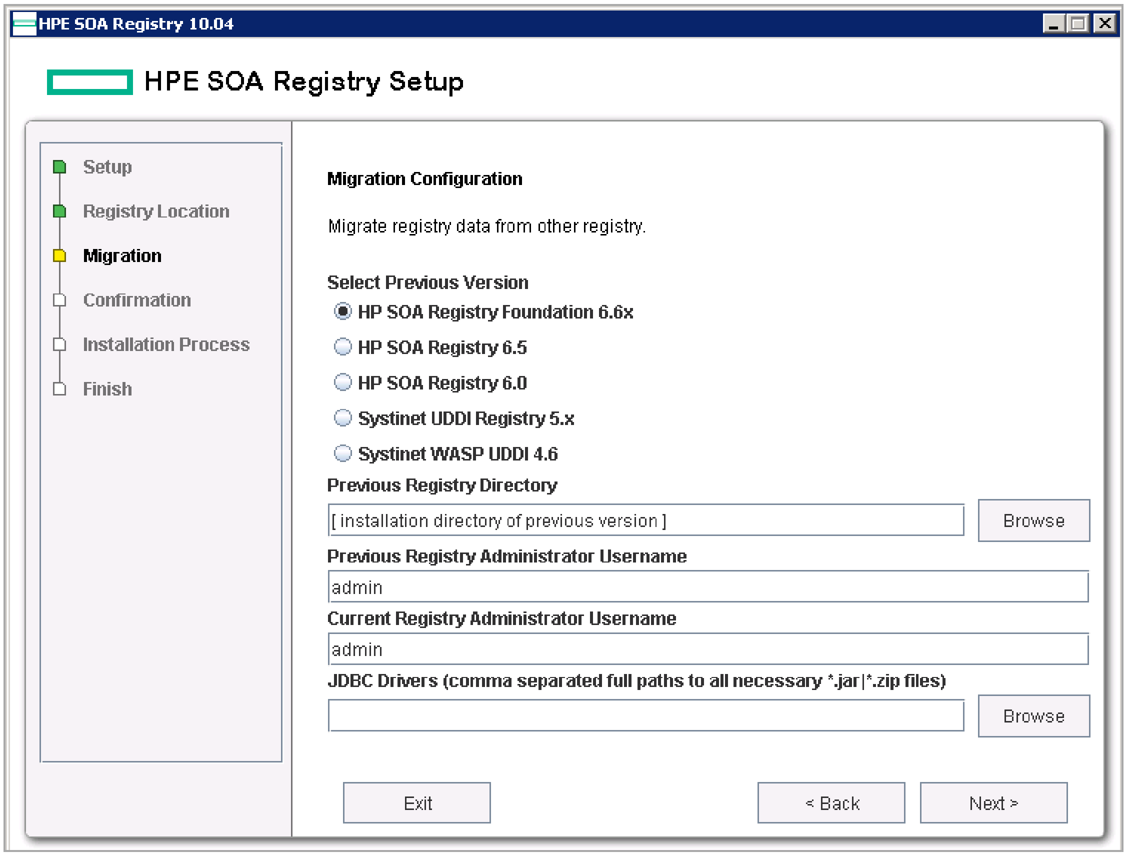
○ Restore

Restore registry data and configuration from a backup

○ Authentication account provider

Change authentication account provider configuration

3. Fill in the following properties:



Migration Configuration

Migrate registry data from other registry.

Select Previous Version

● HP SOA Registry Foundation 6.6x

○ HP SOA Registry 6.5

○ HP SOA Registry 6.0

○ Systinet UDDI Registry 5.x

○ Systinet WASP UDDI 4.6

Previous Registry Directory: [installation directory of previous version]

Previous Registry Administrator Username: admin

Current Registry Administrator Username: admin

JDBC Drivers (comma separated full paths to all necessary *.jar|*.zip files)

- **Previous Registry Version** - HPE SOA Registry version from which you are migrating data
- **Previous Registry Directory** - the directory in which the previous HPE SOA Registry Foundation is installed. The existing data will be migrated from it.
- **Previous Registry Administrator Username** - name of the user having rights to retrieve data from the previous version Registry.
- **Current Registry Administrator Username** - name of the user having rights to save UDDI structure keys. By default, only administrator can migrate all data including private data.
- **JDBC drivers** - Set path to the directory in which the .jar (.zip) of JDBC drivers is located.

> **Note:** Enter this path only if the previous HPE SOA Registry Foundation installation is configured with a different type of database than the current one.

Backup

Backup functionality allows you to save the HPE SOA Registry Foundation data and configuration to a filesystem directory. Later the backup data can serve for full restore of HPE SOA Registry Foundation data and configuration.

What is subject to backup?

- All registry data stored in the database.
- Configuration files.
- HPE SOA Registry Foundation libraries and JSP files.

> **Note:** The HPE SOA Registry Foundation server must be shut down before you start backup or restore operations.
>
> Restoration is additive. It does not delete entities that are already present in HPE SOA Registry Foundation and not present in the data source. If you need to restore and don't want to retain any current data, you must clean the database which can be done via: drop schema, create schema during setup. For details, see "Database Installation" on page 80.

Backup HPE SOA Registry Foundation

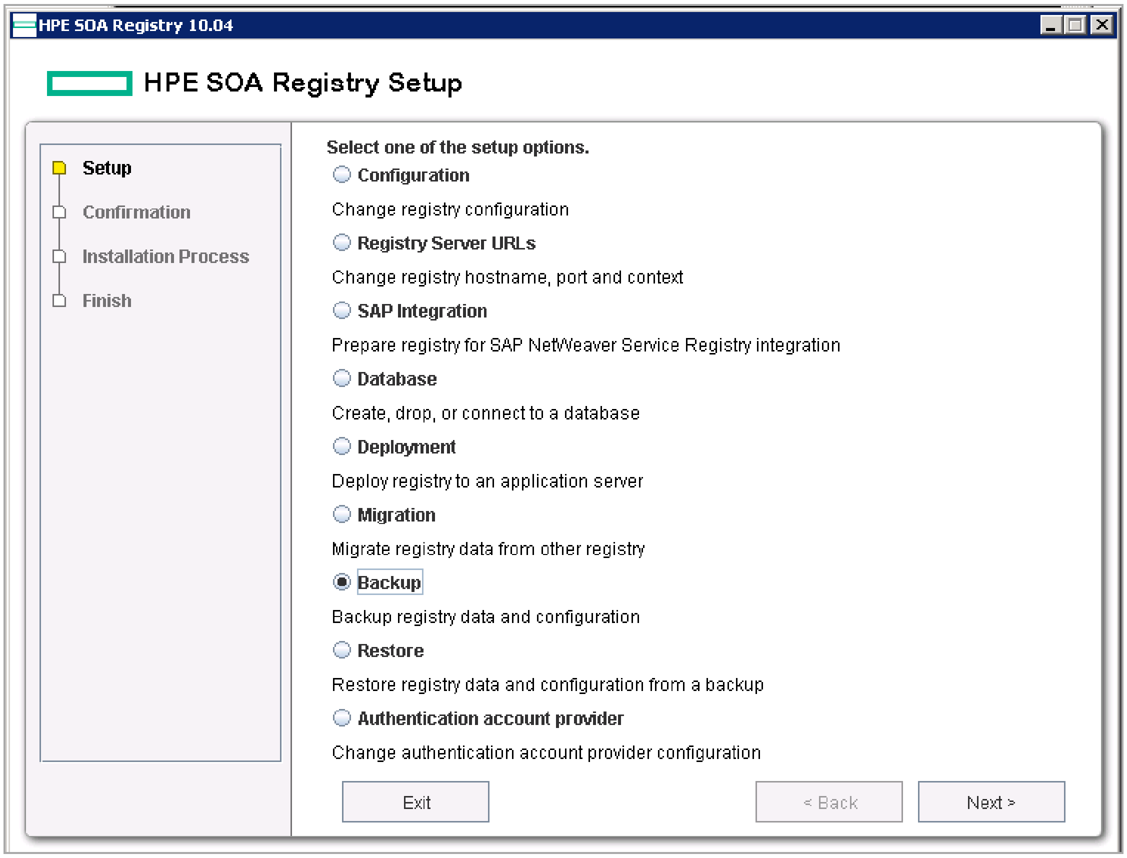
To back up HPE SOA Registry Foundation data:

1. Use the Setup tool and choose Backup. To run the Setup tool, execute the following script from the bin subdirectory of your installation:

Windows:	setup.bat
UNIX:	./setup.sh

For more information, see command-line parameters in "Setup" in "Command Line Scripts" on page 59..

Setup Tool - Select Backup



Select one of the setup options.

○ Configuration

Change registry configuration

○ Registry Server URLs

Change registry hostname, port and context

○ SAP Integration

Prepare registry for SAP NetWeaver Service Registry integration

○ Database

Create, drop, or connect to a database

○ Deployment

Deploy registry to an application server

○ Migration

Migrate registry data from other registry

● Backup

Backup registry data and configuration

○ Restore

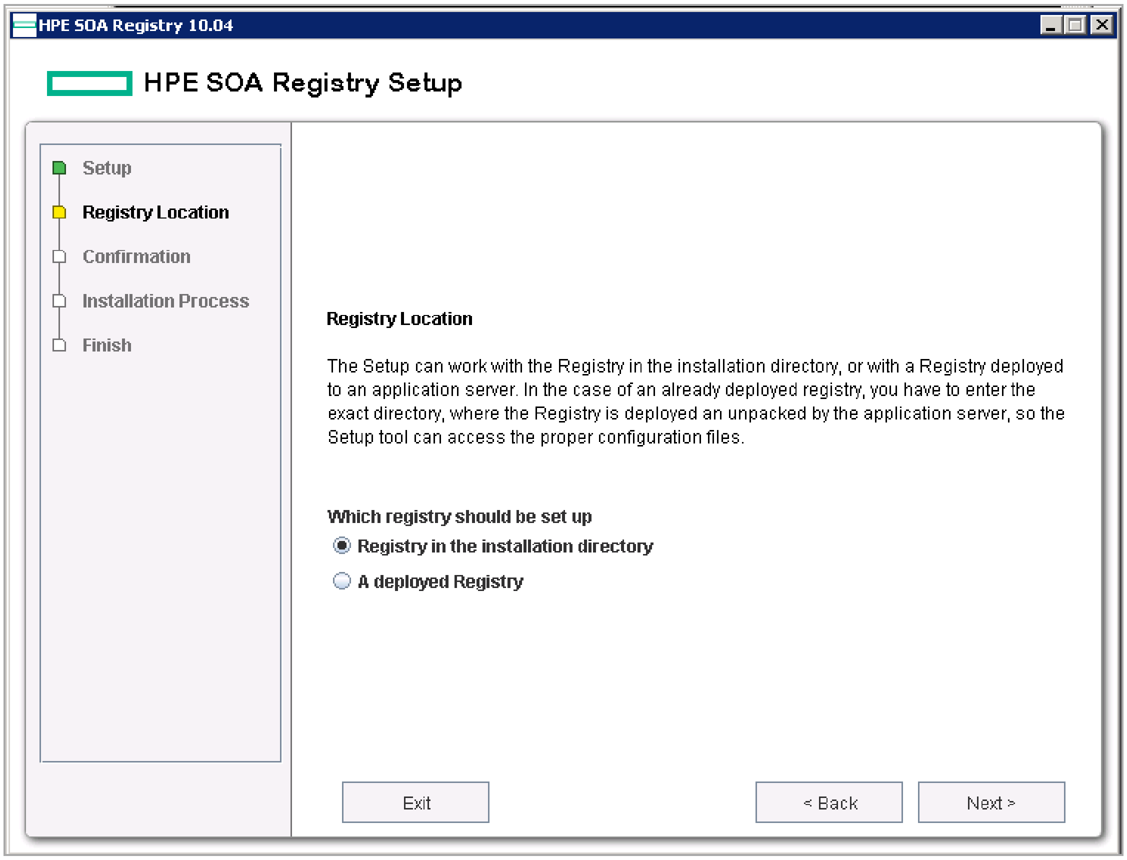
Restore registry data and configuration from a backup

○ Authentication account provider

Change authentication account provider configuration

2. Select whether you want to use HPE SOA Registry Foundation that has been deployed (second choice) or not (first choice).

Setup



Registry Location

The Setup can work with the Registry in the installation directory, or with a Registry deployed to an application server. In the case of an already deployed registry, you have to enter the exact directory, where the Registry is deployed an unpacked by the application server, so the Setup tool can access the proper configuration files.

Which registry should be set up

● Registry in the installation directory

○ A deployed Registry

3. Specify the location of the backup directory. You can check which items you wish to back up as shown in the “Setup Tool - Backup” figure below.

   Item description:

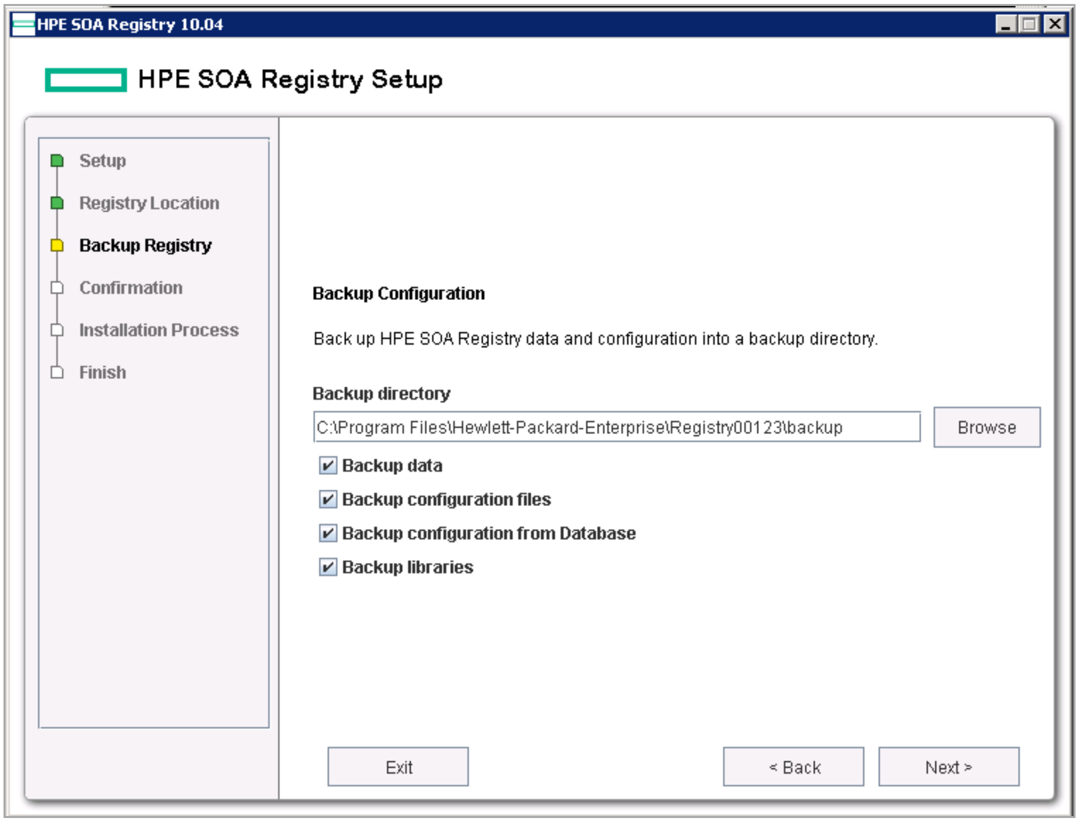
   a. Backup data makes a backup of UDDI data such as different kind of entities and taxonomies

   b. Backup configuration files makes a backup of configuration files from REGISTRY_HOME/app/uddi/conf and REGISTRY_HOME/work/uddi/bsc.jar/conf.

   c. Backup configuration from Database makes a backup of configuration files and their history as they are stored in database. See "Configuration in Database" on page 343.

   d. Backup libraries makes a backup of bsc.jar and web.jar from both app and work directories. These files and directories contain UI customizations and modifications.

Setup Tool - Backup



Backup Configuration

Back up HPE SOA Registry data and configuration into a backup directory.

Backup directory: C:\Program Files\Hewlett-Packard-Enterprise\Registry00123\backup

☑ Backup data
☑ Backup configuration files
☑ Backup configuration from Database
☑ Backup libraries

Restore HPE SOA Registry Foundation

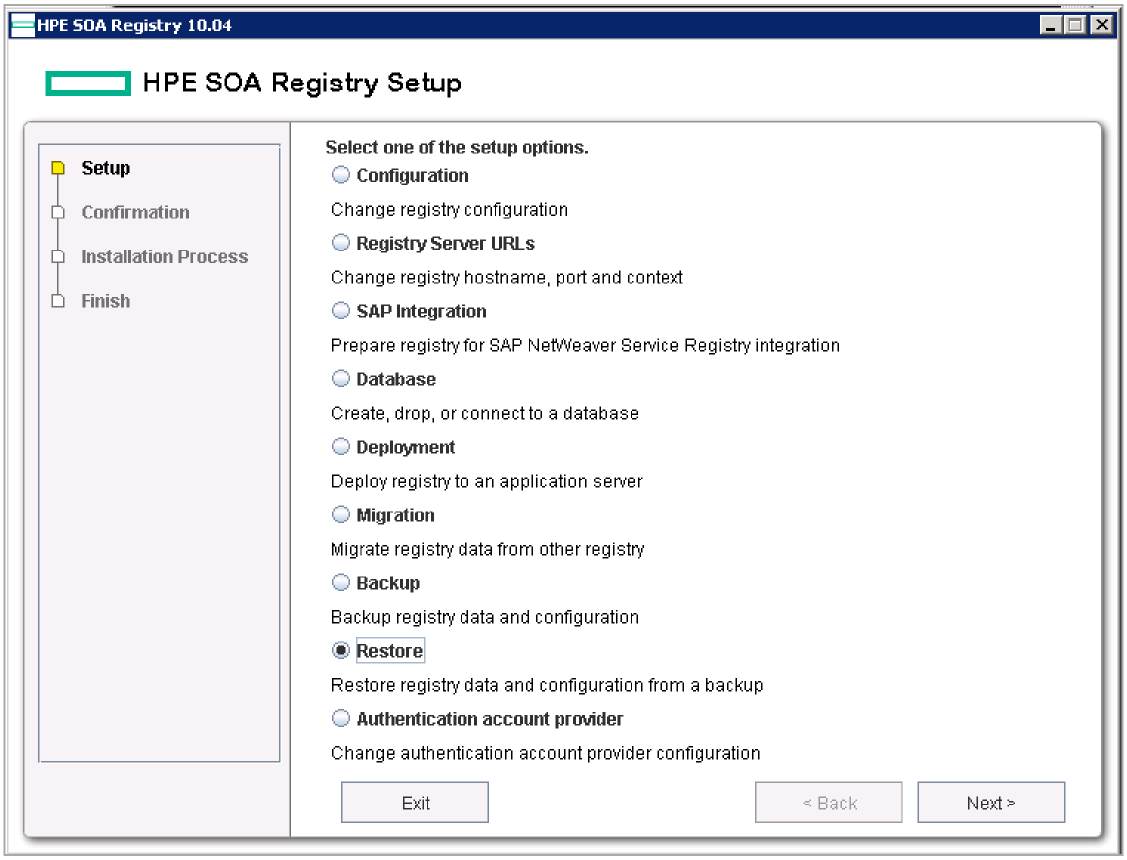
To restore registry data and configuration from a backup:

1. Use the Setup tool and choose Restore. To run the Setup tool, execute the following script from the bin subdirectory of your installation:

Windows:	setup.bat
UNIX:	./setup.sh

See command-line parameters in "Setup" in "Command Line Scripts" on page 59.

Setup Tool - Select Restore



Select one of the setup options.

○ Configuration

Change registry configuration

○ Registry Server URLs

Change registry hostname, port and context

○ SAP Integration

Prepare registry for SAP NetWeaver Service Registry integration

○ Database

Create, drop, or connect to a database

○ Deployment

Deploy registry to an application server

○ Migration

Migrate registry data from other registry

○ Backup

Backup registry data and configuration

● Restore

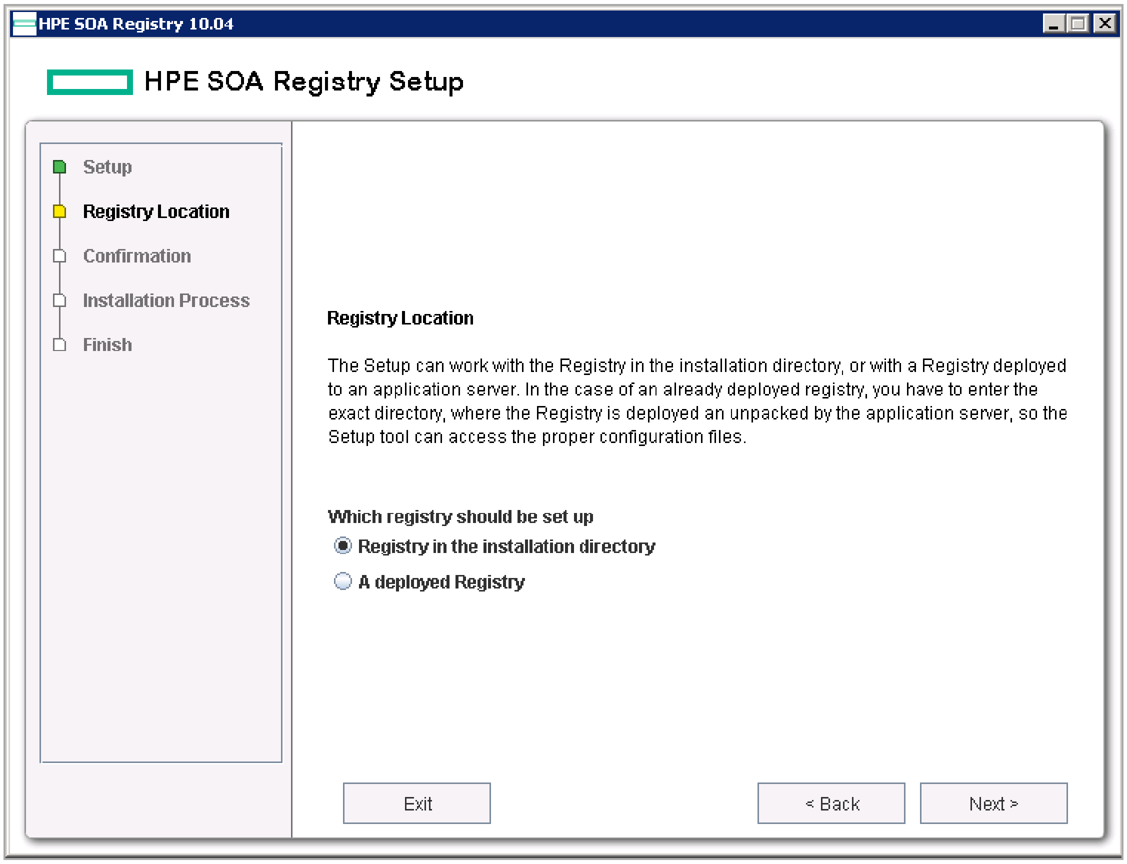
Restore registry data and configuration from a backup

○ Authentication account provider

Change authentication account provider configuration

2. Select whether you want to use HPE SOA Registry Foundation that has been deployed (second choice) or not (first choice).

Setup



Registry Location

The Setup can work with the Registry in the installation directory, or with a Registry deployed to an application server. In the case of an already deployed registry, you have to enter the exact directory, where the Registry is deployed an unpacked by the application server, so the Setup tool can access the proper configuration files.

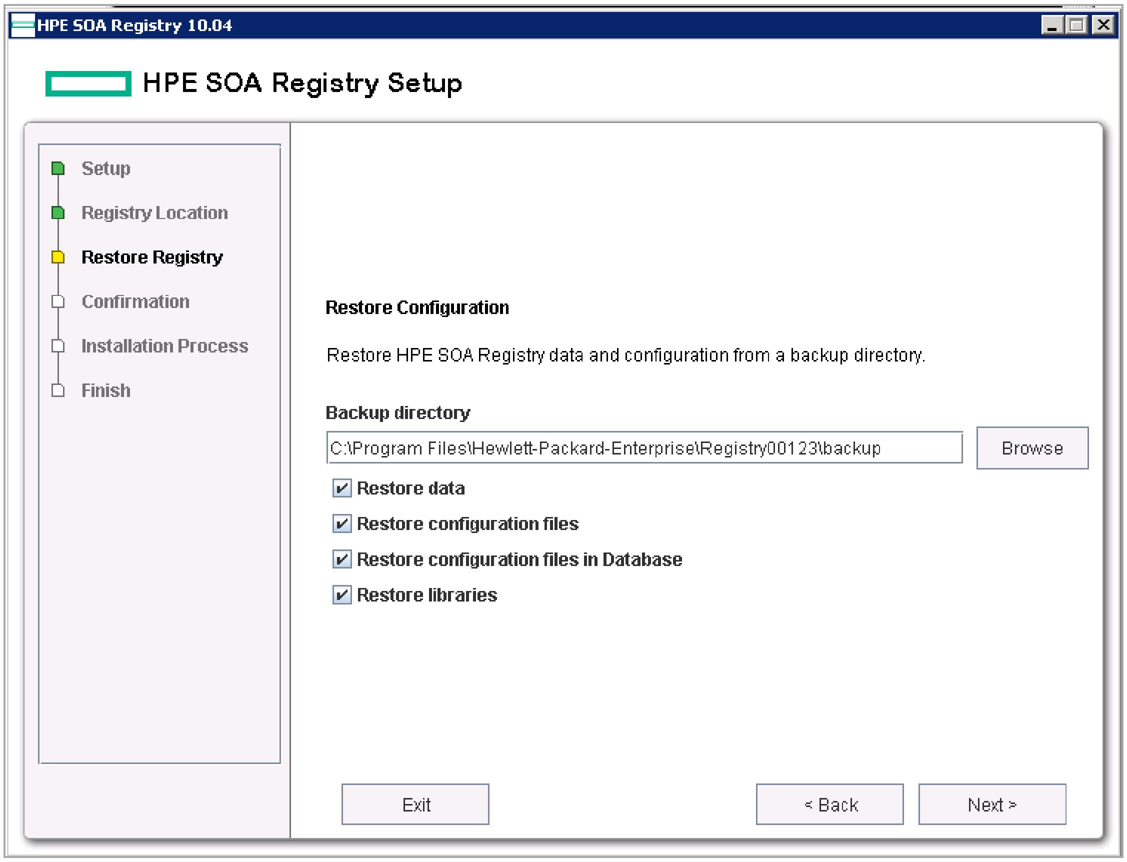
Which registry should be set up

● Registry in the installation directory

○ A deployed Registry

3. Specify the location of backup directory and check the items you wish to restore. The restore will work only for items that have been backed up previously.

   Setup Tool - Restore from Backup



Restore Configuration

Restore HPE SOA Registry data and configuration from a backup directory.

Backup directory: C:\Program Files\Hewlett-Packard-Enterprise\Registry00123\backup

☑ Restore data
☑ Restore configuration files
☑ Restore configuration files in Database
☑ Restore libraries

NT Service Support

The HPE SOA Registry Foundation server can be run as a service on Windows server 2008 and Windows server 2012 64-bits (CPU Xeon). Support for NT service installation is included by default on Windows servers, and cannot be installed on UNIX machines. The support is a set of executable files that enable you to install, start, stop, and uninstall HPE SOA Registry Foundation as an NT service.

The server log is by default written into the log file. The output to the NT log can be manually configured.

Installation

When the HPE SOA Registry Foundation installation is complete, the `REGISTRY_HOME\bin` directory contains these four batch files related to NT service support:

- InstallService.bat
- UnInstallService.bat
- StartService.bat
- StopService.bat

> **Note:** After installing HPE SOA Registry Foundation with NT Service support, the registry server is *not* installed as an NT service. It must be installed manually, as follows.

If you want to customize the NT service first (set-up the JVM max memory, add files to classpath, etc.), please read the Customizing section now.

Make sure that the JAVA_HOME environment variable points to your JDK and run the InstallService.bat command.

When the installation is finished, the name of the installed NT service is printed to the screen. The default name is HPE SOA Registry Foundation.

> **Note:** You may need extra permissions to install a new service into your OS. To determine whether you have these permissions, please consult your system administrator.

If the installation fails, read the Customizing section. If it does not contain the solution, contact HPE Support.

Starting and Stopping

Once the HPE SOA Registry Foundation server NT service is installed, start it as you would any NT service, by selecting **Control Panel**> **Administrative Tools** > **Services** > **start**.

As a shortcut, you can use the `StartService.bat` command in the `REGISTRY_HOME\bin` directory.

> **Note:** You may need extra permissions to start or stop an NT service in your OS. To determine whether you have these permissions, please consult your system administrator.

To stop the server, use either the system tools or the StopService.bat command.

> **Note:** For security reasons, you cannot use serverstop.bat or server.bat stop to stop a HPE SOA Registry Foundation server that is running as an NT service.

Logging

By default, the logs of the server are written into the `REGISTRY_HOME\log\registry_NTService.log` file. The default maximum size of the log file is 1MB. When the file is full, a backup is created and the content of the file is cleaned. By default, 3 backups are kept and older backups are deleted.

Logging Customization

HPE SOA Registry Foundation uses the Log4J library for logging. You can manually change its logging behavior. The configuration is stored in the file `REGISTRY_HOME\conf\log4j_NTservice.config`. You can change the log output, message priority and other settings in this file as follows:

Message Priority Settings

To change the message priority from INFO to ERROR, comment out the following lines in the `config` file:

```
log4j.category.com.systinet.wasp.events=INFO,R
log4j.additivity.com.systinet.wasp.events=false
```

```
log4j.category.com.systinet.wasp.events=INFO,ntlog
log4j.additivity.com.systinet.wasp.events=false
```

Log File Properties

To change the log file properties, change the Rolling File appender settings:

```
log4j.appender.R.File=log/registry_NTService.log
log4j.appender.R.MaxFileSize=1024KB
log4j.appender.R.MaxBackupIndex=3
```

Switching to NT Log

To switch logging from file to NT log, comment out the lines:

```
log4j.category.com.systinet.wasp.events=INFO,R
log4j.additivity.com.systinet.wasp.events=false
log4j.category.com.systinet.wasp.errors=ERROR,R
log4j.additivity.com.systinet.wasp.errors=false
```

and uncomment the lines:

```
#log4j.category.com.systinet.wasp.events=INFO,ntlog
#log4j.additivity.com.systinet.wasp.events=false
#log4j.category.com.systinet.wasp.errors=ERROR,ntlog
#log4j.additivity.com.systinet.wasp.errors=false
```

from this section:

```
# Assigning appenders to categories
# (using rolling file appender by default)
log4j.category.com.systinet.wasp.events=INFO,R
log4j.additivity.com.systinet.wasp.events=false
log4j.category.com.systinet.wasp.errors=ERROR,R
log4j.additivity.com.systinet.wasp.errors=false

# Uncomment next line if you want use NT Event Log
# for logging of error messages
#log4j.category.com.systinet.wasp.events=INFO,ntlog
#log4j.additivity.com.systinet.wasp.events=false
#log4j.category.com.systinet.wasp.errors=ERROR,ntlog
#log4j.additivity.com.systinet.wasp.errors=false
```

so that the section reads:

```
# Assigning appenders to categories
# (using rolling file appender by default)
#log4j.category.com.systinet.wasp.events=INFO,R
#log4j.additivity.com.systinet.wasp.events=false
#log4j.category.com.systinet.wasp.errors=ERROR,R
#log4j.additivity.com.systinet.wasp.errors=false

# Uncomment next line if you want use NT Event Log
# for logging of error messages
log4j.category.com.systinet.wasp.events=INFO,ntlog
log4j.additivity.com.systinet.wasp.events=false
log4j.category.com.systinet.wasp.errors=ERROR,ntlog
log4j.additivity.com.systinet.wasp.errors=false
```

> **Note:** We recommend that you log only errors to the NT log.

> **Note:** The `REGISTRY_HOME\lib\NTEventLogAppender.dll` file must be copied into the system PATH if you want to use the NT event log for logging.

> For security reasons, on 64-bit Windows, the folder "windows\system32" remains hidden from 32-bit programs and the system redirects all operations concerning this folder to "windows\syswow64". Using a 32-bit program to copy NTEventLogAppender.dll to "windows\system32" may cause problems with the NT log.

Using Other Log4J Appenders

Rolling File and NTLog are the two default appenders. You can choose any Log4J appender that suits your needs. To add custom classes to the HPE SOA Registry Foundation NT service classpath, please see the Customizing section.

You must restart the HPE SOA Registry Foundation NT service to put the changes into effect.

For more information about Log4J and its settings, visit Apache/Jakarta's Log4j Project website.

Customizing

You can manually set up the name "HPE SOA Registry Foundation NT Service" and the JVM parameters that are used to start HPE SOA Registry Foundation as an NT service. To customize logging, please visit the previous section, Logging.

All customizable files in the following instructions are located in the `REGISTRY_HOME\bin directory`.

> **Note:** All the following changes require re-installation of the HPE SOA Registry Foundation NT Service. Uninstall it first, make your modifications and reinstall the service.

NT Service Name Change

To change the service name:

1. Uninstall the existing service by running UnInstallService.bat.
2. Manually edit the files
   - `UnInstallService.bat`
   - `InstallService.bat`

   - `StartService.bat`
   - `StopService.bat`

3. Change the system variable NT_SERVICE_NAME, so the row with the variable resembles:

   `set NT_SERVICE_NAME=HPE SOA Registry Foundation`

4. Install your NT service with its new name by running `InstallService.bat`.
5. Start the new service by running StartService.bat.

JVM Startup Parameters

The parameters of the Java Virtual Machine are set up during the installation of the NT service. If you modify the parameters, you must reinstall the NT service to put the changes into effect. To modify the parameters of the NT service, open `InstallService.bat` in a text editor and do the following:

- To change the maximum size of available memory, change the value of the `JVM_MEM` variable, with a command like set **`JVM_MEM=-Xmx256m`**.
- To add custom files to the classpath, edit the `RegistryService.exe` parameters. These are in the line **`"-Djava.class.path=%REGISTRY_HOME%\lib\wasp.jar"`**.

HPE SOA Registry Foundation Deployed to Application Server

Systinet does not support installation of deployed HPE SOA Registry Foundation as an NT Service. For more information, please see the documentation of your application server provider. However, any Java application can be installed as an NT Service with Systinet's NT service solution. Contact http://www.hp.com/go/hpsoftwaresupport if you need to run a deployed HPE SOA Registry Foundation server as an NT service.

Uninstallation

To uninstall the HPE SOA Registry Foundation server NT service, run `UnInstallService.bat` from the `REGISTRY_HOME\bin` directory. The uninstaller first tries to stop the NT service. It then removes the NT service from your OS.

Running in Linux

This section has the following topics:

"Using the syslog Daemon with HPE SOA Registry Foundation" below

"Running HPE SOA Registry Foundation as a UNIX Daemon" on the next page

Using the `syslog` Daemon with HPE SOA Registry Foundation

The log4j system used in HPE SOA Registry Foundation can be configured to send log messages to the `syslog` daemon. In order to utilize this feature, your system must be configured as follows:

1. Change `log4j` in `REGISTRY_HOME/conf/log4j.config`. First add a syslog appender, as shown in Example 12, “log4j.config--syslog Appender” below. Note the following properties in particular:

   - `syslogHost` - Set to host name of the computer where `syslog` is running.
   - `Facility` - HPE SOA Registry Foundation log message facility recognized by `syslog`.

   Example 12. log4j.config--syslog Appender

   ```
   # Appender to syslog
   log4j.appender.syslog=org.apache.log4j.net.syslogAppender
   log4j.appender.syslog.syslogHost=localhost
   log4j.appender.syslog.Facility=local6
   log4j.appender.syslog.layout=org.apache.log4j.PatternLayout
   log4j.appender.syslog.layout.ConversionPattern=%p: %c{2} - %m%n
   ```

   Then add `syslog` to the value of the `property` `log4j.category.com.systinet.wasp.events` under `# event monitoring`, as follows:

   Example 13. log4j.config--Event Monitoring

   ```
   # event monitoring
   log4j.category.com.systinet.wasp.events=INFO,eventLog,syslog
   ```

2. Set the `syslogd` configuration to recognize log messages from HPE SOA Registry Foundation. Implicitly, HPE SOA Registry Foundation sends log messages to `syslog` under the facility `local6`. Therefore, modify the `/etc/syslog.conf` file by adding the following line of text:

```
local6.* /var/log/registry.log
```

HPE SOA Registry Foundation will now log messages of all priorities into the file `/var/log/registry.log`. You should create this file now with appropriate permissions (otherwise `syslogd` will create it for you automatically with default permissions, which may not be suitable for you).

3. Your `syslog` daemon must be started with remote logging enabled (the `-r` command line option). To make sure that:

   - `syslogd` is running, use the pgrep syslogd command.
   - remote logging is enabled, use the netstat -l command (`syslog`'s udp port is 514).

   > **Note:** The `local6` facility is not mandatory in any way. You may use other `localX` facilities instead.

Running HPE SOA Registry Foundation as a UNIX Daemon

HPE SOA Registry Foundation can be forced to start as a system daemon using the script REGISTRY_HOME/etc/bin/registry.sh. This script can be renamed registry as per UNIX conventions. The directions for using this script follow.

1. Tailor the service script as needed. The meaning of variables is shown in the following table:

   Variables in the HPE SOA Registry Service Script

Name of variable in `registry` service script	Description	Default value
REGISTRY_HOME	Home directory of HPE SOA Registry Foundation	HPE SOA Registry Foundation Installation directory.
JAVA_HOME	Home directory of Java	None. This variable must be set manually.
REGISTRY_USER	User under whom the HPE SOA Registry server should run.	Determined during runtime according to the user who owns the `REGISTRY_HOME`

	If this is set to root, it will be changed to "nobody".	directory. If the user is root, this value reverts to "nobody".
TIMEOUT	Number of seconds the system waits for HPE SOA Registry to successfully start up.	60 seconds.

2. Rename the script `registry` (without the `.sh` extension) and save it in the `/etc/init.d/` directory.
3. (optional) To start HPE SOA Registry Foundation automatically in the appropriate run-level, create `SXXregistry` and `KXXregistry` symbolic links in the appropriate `/etc/rcX.d/` directory.

Now you may start and stop HPE SOA Registry Foundation using the installed script. You can invoke this script directly or by using specific OS tools. For example, on RedHat, by using the redhat-config-services command.

The parameters of the script are shown in the following table:

Parameters of `init.d` Scripts

Parameter	Function
`start`	Starts HPE SOA Registry Foundation
`stop`	Stops HPE SOA Registry Foundation
`restart`	Restarts HPE SOA Registry Foundation
`condrestart`	Restarts HPE SOA Registry Foundation only if it is already running
`status`	Displays whether HPE SOA Registry Foundation is running or not

The provided startup script may be run by the root user. The script uses the `su` command to run as `REGISTRY_USER`.

Uninstallation

This section describes how to uninstall on:

"Windows" on the next page

"Linux" on the next page

Windows

1. If you installed HPE SOA Registry Foundation as NT service, uninstall it by executing script **`REGISTRY_HOME\bin\UninstallService.bat`**. See more information on "NT Service Support" on page 189.
2. Remove Icons and Start menu items on Windows platform.
3. Drop database manually via the Setup tool. Setup should automatically detect the current configuration of the database. See "Reconfiguring After Installation" on page 62.
4. Delete installation directory of HPE SOA Registry Foundation.

Linux

1. If you installed HPE SOA Registry Foundation as Linux daemon, remove the `registry` files from `/etc/init.d`. Remove also links `KXXregistry` and `SXXregistry` from appropriate directory /etc/rcX.d. Unregister the daemon by system tools.
2. Drop database manually via the Setup tool. Setup should automatically detect the current configuration of the database. See "Reconfiguring After Installation" on page 62.
3. Delete installation directory of HPE SOA Registry Foundation.

Chapter 3: User's Guide

The HPE SOA Registry Foundation 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 HPE SOA Registry Foundation**

  This section is a brief intoduction to HPE SOA Registry Foundation including basic concepts of UDDI specifications.

- **Registry Consoles**

  This section presents Registry Consoles.

- **Demo Data Description**

  The HPE SOA Registry Foundation's Demo Data chapter describes the business domain and UDDI data structures used in the HPE SOA Registry Foundation Demo Suite and both registry consoles.

- **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**

    HPE SOA Registry Foundation'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 HPE SOA Registry Foundation.

  - **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.

Introduction to HPE SOA Registry Foundation

HPE SOA Registry Foundation 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:

- "UDDI's Role in the Web Services World - UDDI Benefits" below
- " Typical Application of a UDDI Registry" on the next page
- "Basic Concepts of the UDDI Specification" on the next page
- " Subscriptions in HPE SOA Registry Foundation" on page 207

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.

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.

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.

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 tModels.

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.
- Names and Descriptions. The business entity can have a set of names and descriptions, in a variety of languages if necessary.
- 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.

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.

HPE SOA Registry Foundation allows definition of enterprise taxonomies. Users can also download and upload any taxonomy as an XML file. HPE SOA Registry Foundation offers tools for 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 HPE SOA Registry Foundation 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.

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.

HPE SOA Registry Foundation 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 HPE SOA Registry Foundation, 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.

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.

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.

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 .

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".

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.

Subscriptions in HPE SOA Registry Foundation

Subscriptions are used to alert interested users in changes made to structures in HPE SOA Registry Foundation. The HPE SOA Registry Foundation 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`

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, see "Registry Configuration" on page 334 in the Administrator's Guide.
- `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)

Subscription Notification

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

- 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.)

- 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`. For details, see "Writing a Subscription Notification Service" on page 502 in the Developer's Guide.
- `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. For details, see "Subscription" on page 582 in Demos.

XSLT Over Notification

To improve the readability of notifications sent to users via email, HPE SOA Registry Foundation 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`".

Suppressing Empty Notifications

Another HPE SOA Registry Foundation 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`".

Related Links

- To manage subscriptions via the Registry Console, see "Registry Console Reference" on page 243.
- To use and manage subscriptions, see the Subscription API chapter of the UDDI v3 Specification.

Registry Console

HPE SOA Registry Foundation web 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 HPE SOA Registry Foundation is installed. The default port is 8080. See “Registry Console Overview”

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

Demo Data

Demo data is pre-installed with HPE SOA Registry Foundation. There is one demo data set:

- To demonstrate Registry Console and Demo Suite

Demo data for Registry Console and demos

Demo data describes a multinational company with offices in several locations and HPE SOA Registry Foundation 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 4, “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	Yes	stored in the HR department, projected to IT department; takes single

list		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 HPE SOA Registry Foundation 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

Advanced Topics

Data Access Control: Principles

This chapter describes the entity access control mechanism, which defines permissions for users and groups to access structures in HPE SOA Registry Foundation.

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 HPE SOA Registry Foundation used to control access to APIs and their operations. API permissions are relations between the user or group and operation only. For details, see "Permissions: Principles" on page 349 in the Administrator's Guide.

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.

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.

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.

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.

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 HPE SOA Registry Foundation users.

- `system#everyone`
  Holds all users of HPE SOA Registry Foundation (both users who have and who do not have a HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation via a local intranet. (This group is reserved for a future release. There is no implementation behind it as of HPE SOA Registry Foundation 6.65).

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
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

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 14. 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>
```

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.

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, see the UDDI v3 Specification.

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`.

> **Note:** 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 “Publish Custody Transfer” in "Publishing" in .

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 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:

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 “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.

Range Queries

HPE SOA Registry Foundation'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 “Find Business by Categories” in "Searching" on page 256 for instructions on how to search for UDDI data structures using range queries with the Registry Console.

> **Note:** The HPE SOA Registry Foundation 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.

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 "Advanced Inquiry - Range Queries" on page 571 in Demos.

Taxonomy: Principles, Creation and Validation

The UDDI Version 3 Specification 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.

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 HPE SOA Registry Foundation 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.

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.

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

HPE SOA Registry Foundation 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, HPE SOA Registry Foundation 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.

HPE SOA Registry Foundation Requirements

HPE SOA Registry Foundation conforms to the technical note "Providing A Value Set For Use" in UDDI Version 3.

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.

HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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 `org.systinet.uddi.client.valueset.validation.v3.UDDI_ValueSetValidation_PortType` and is accessible by the HPE SOA Registry Foundation 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

HPE SOA Registry Foundation 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.

Types of keyValues

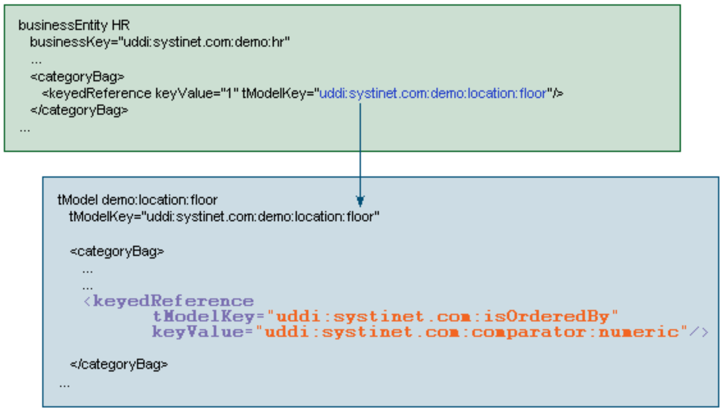
The creator of the taxonomy must specify types of keyValues by assigning the appropriate comparator reference (comparator tModel) of the `systinet-com:isOrderedBy` taxonomy to the categorization taxonomy you want to use to categorize a UDDI entity. The following types of key values types are supported:

- `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. For more information, see “Custom Ordinal Types” below.

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"/>
```

Example of Numeric Categorization



```
businessEntity HR
  businessKey="uddi:systinet.com:demo:hr"
  ...
  <categoryBag>
    <keyedReference keyValue="1" tModelKey="uddi:systinet.com:demo:location:floor"/>
  </categoryBag>
...
```

```
tModel demo:location:floor
  tModelKey="uddi:systinet.com:demo:location:floor"

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

  </categoryBag>
...
```

The above example shows how the demo:location:floor taxonomy from Demo data can be assigned numeric key values.

> **Note:** If you change type of keyValues of the taxonomy and there are entities in the HPE SOA Registry Foundation that were already categorized with the taxonomy, the HPE SOA Registry Foundation 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 "Accessing Registry Management" on page 297 in the Administrator's Guide.

- To learn how to make this assignment using the Registry Console , see “Adding a Category” in "Publishing" on page 265 in the User's Guide.
- For instructions on how to search UDDI data structures using range queries with Registry Console, see "Searching" on page 256 in the User's Guide.

Custom Ordinal Types

You can define your custom ordinal types. To demonstrate possible extensions, HPE SOA Registry Foundation contains two demo comparators:

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

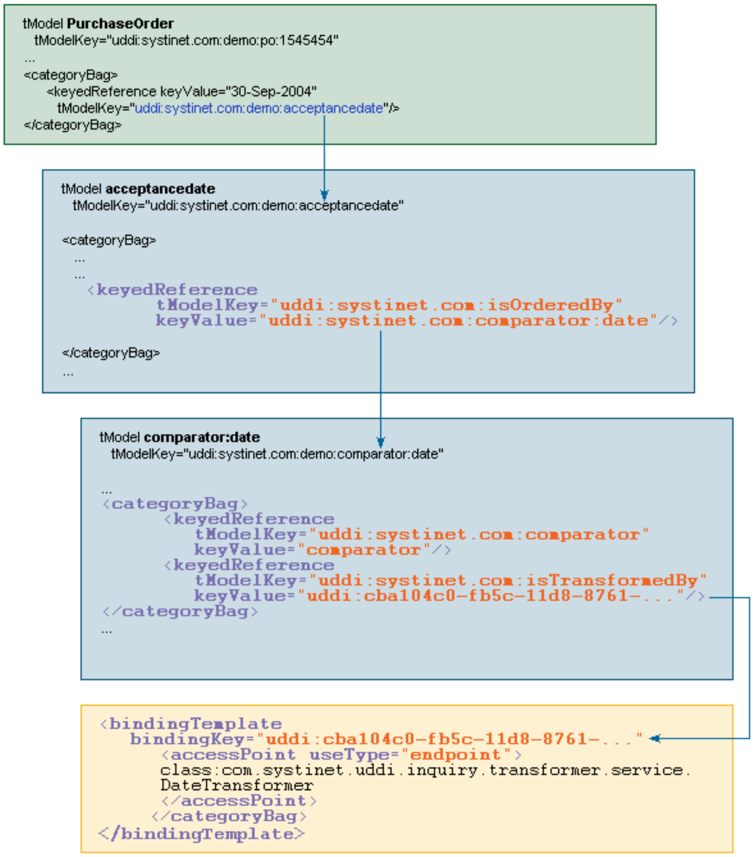
Let us assume that you want to create a taxonomy with date values in keyValues. You must mark the taxonomy tModel (that is, add the following `keyedReference into its categoryBag) by <keyedReference tModelKey="uddi:systinet.com:isOrderedBy" keyValue="uddi:systinet.com:comparator:date"/>`. It is quite 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 HPE SOA Registry Foundation 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.

   > **Note:** If you change implementation of the of the transformer service of the taxonomy and there are entities in the HPE SOA Registry Foundation that were already categorized with the taxonomy, the HPE SOA Registry Foundation 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 "Accessing Registry Management" on page 297 in the Administrator's Guide.

The following figure, “Example of Custom Categorization (date)” shows the tModel references for date categorization ordering. It describes a purchase order document that has been mapped to HPE SOA Registry Foundation 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.

Example of Custom Categorization (date)



```
tModel PurchaseOrder
  tModelKey="uddi:systinet.com:demo:po:1545454"
...
<categoryBag>
    <keyedReference keyValue="30-Sep-2004"
      tModelKey="uddi:systinet.com:demo:acceptancedate"/>
</categoryBag>
```

```
tModel acceptancedate
  tModelKey="uddi:systinet.com:demo:acceptancedate"

<categoryBag>
  ...
  ...
   <keyedReference
          tModelKey="uddi:systinet.com:isOrderedBy"
          keyValue="uddi:systinet.com:comparator:date"/>

</categoryBag>
...
```

```
tModel comparator:date
  tModelKey="uddi:systinet.com:demo:comparator:date"

...
<categoryBag>
      <keyedReference
         tModelKey="uddi:systinet.com:comparator"
         keyValue="comparator"/>
      <keyedReference
         tModelKey="uddi:systinet.com:isTransformedBy"
         keyValue="uddi:cba104c0-fb5c-11d8-8761-..."/>
</categoryBag>
...
```

```
<bindingTemplate
   bindingKey="uddi:cba104c0-fb5c-11d8-8761-..."
      <accessPoint useType="endpoint">
      class:com.systinet.uddi.inquiry.transformer.service.
      DateTransformer
      </accessPoint>
     </categoryBag>
</bindingTemplate>
```

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.

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 "Taxonomy" on page 395 in the Developer's Guide.

> **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.

> **Note:** 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.

Predeployed Taxonomies

HPE SOA Registry Foundation 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
Checked	yes, Internal Validation Service

- `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	categorization
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: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

- `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:system`
v2 UUID key	`uuid:e148d85e-cc08-32f6-8f00-db85e258e511`
Categorization	categorization
Compatibility	bindingTemplate
Checked	yes

`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	yes

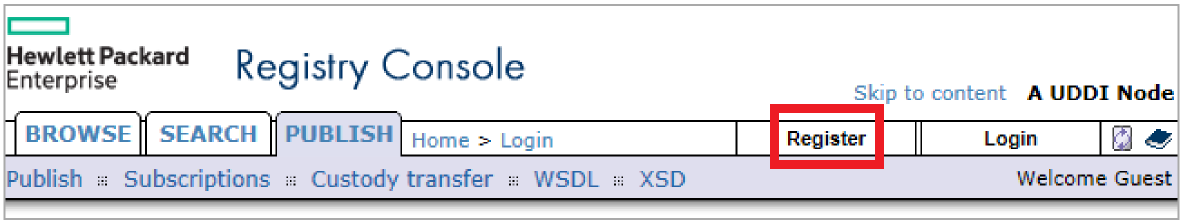
Registry Console Reference

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

Register/Create Account

Register

Before you can publish data to the registry, you must have a HPE SOA Registry Foundation account. You can create an account via the web interface.

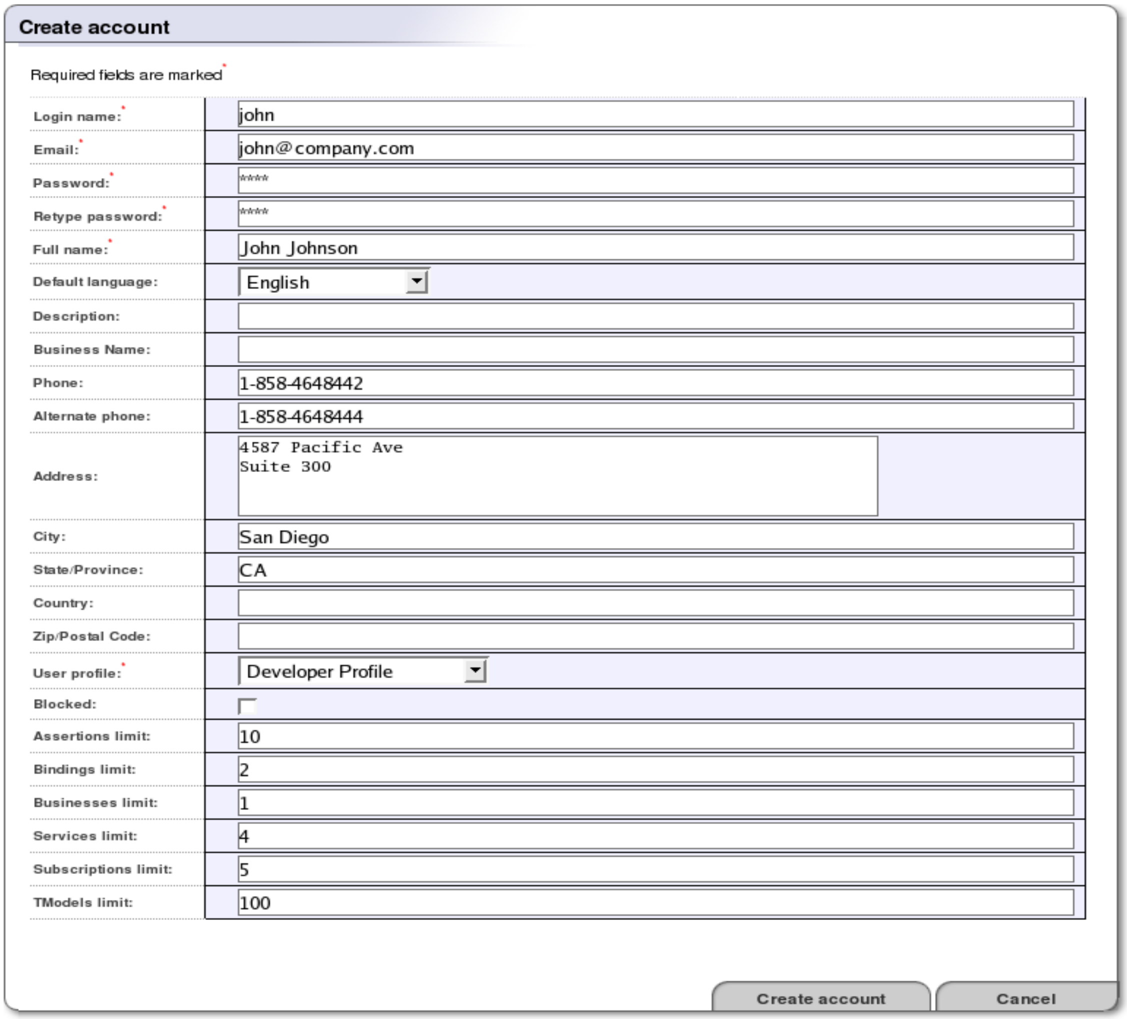


Follow these steps to register a user account:

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.

   Create Account Page



Create account

Required fields are marked*

Login name:*	john
Email:*	john@company.com
Password:*	****
Retype password:*	****
Full name:*	John Johnson
Default language:	English
Description:	
Business Name:	
Phone:	1-858-4648442
Alternate phone:	1-858-4648444
Address:	4587 Pacific Ave Suite 300
City:	San Diego
State/Province:	CA
Country:	
Zip/Postal Code:	
User profile:*	Developer Profile
Blocked:	□
Assertions limit:	10
Bindings limit:	2
Businesses limit:	1
Services limit:	4
Subscriptions limit:	5
TModels limit:	100

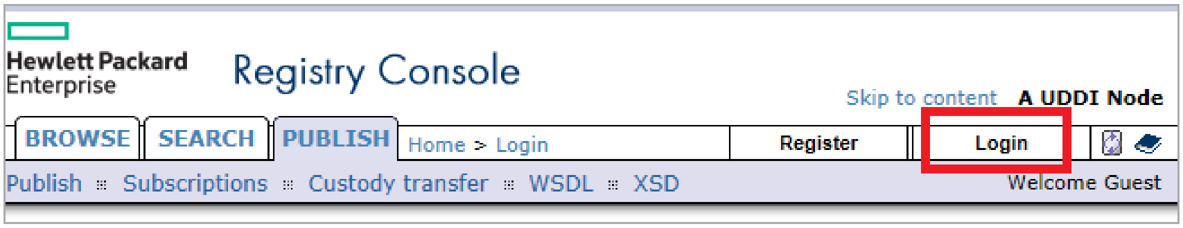
3. Click the **Create account** button.

The new account is now enabled.

> **Note:** HPE SOA Registry Foundation 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.



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.

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 HPE SOA Registry Foundation when you know a key of UDDI data types (business, service, binding, and tModel)

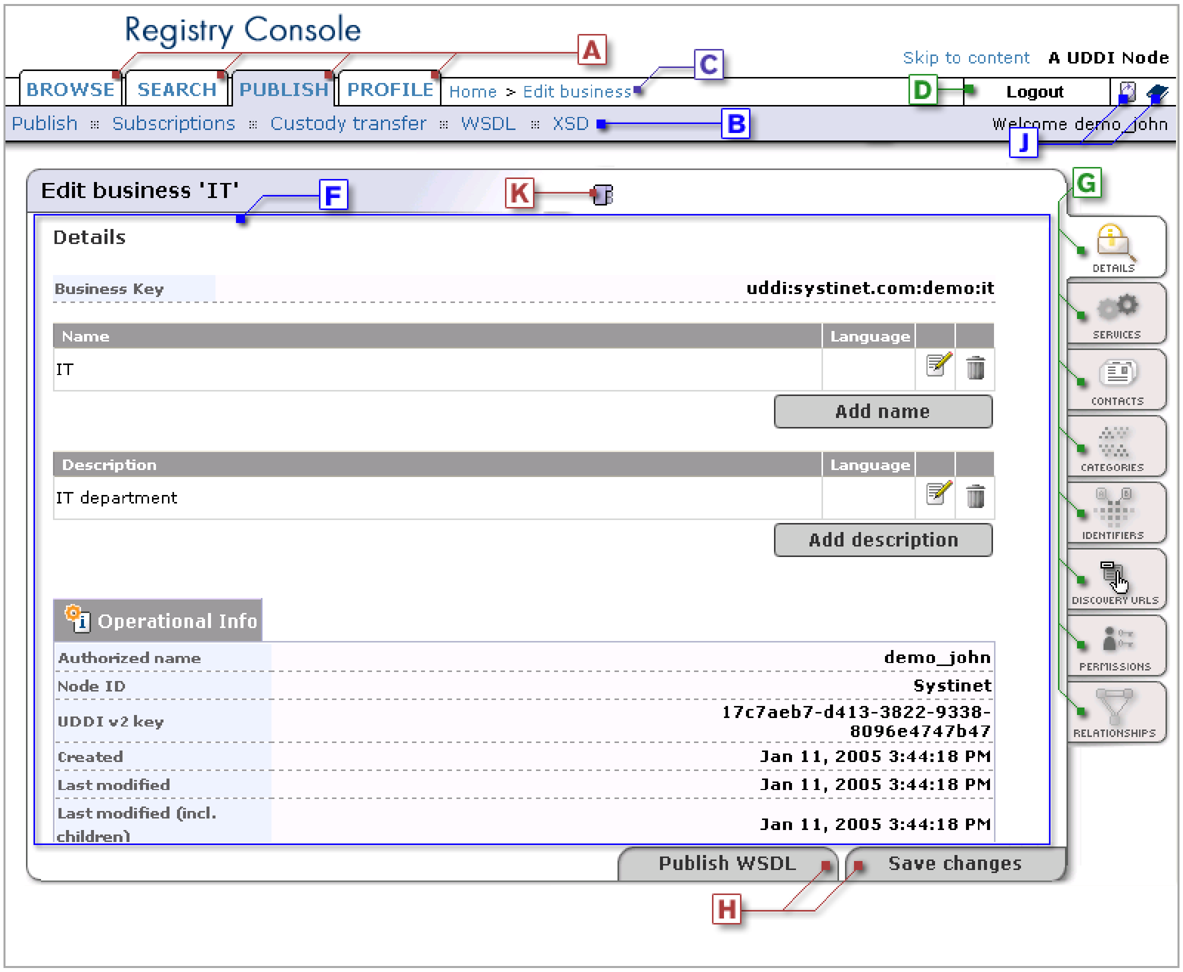
Publish — This tab allows you to publish UDDI structures (businessEntities, businessServices, bindingTemplates, and tModels). On this tab, you can also assert relationships between business entities, subscribe interest in receiving information about changes made to a registry, transfer ownership of selected 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 HPE SOA Registry Foundation administrator to perform management tasks. See Administrators Guide for more information.

B: Menu Bar Sub menu options are located here.

Registry Console Overview



Edit business 'IT'

Details

Business Key: uddi:systinet.com:demo:it

Name	Language
IT	

Description	Language
IT department	

Operational Info

Authorized name	demo_john
Node ID	Systinet
UDDI v2 key	17c7aeb7-d413-3822-9338-8096e4747b47
Created	Jan 11, 2005 3:44:18 PM
Last modified	Jan 11, 2005 3:44:18 PM
Last modified (incl. children)	Jan 11, 2005 3:44:18 PM

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

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.

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.

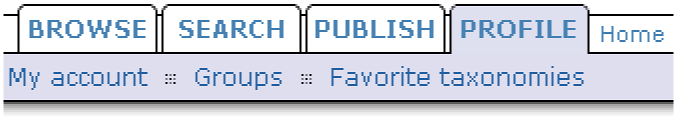
K: Action Icons Icons from this area allow you to switch on/off display tabs and open the current page in the printer friendly mode.

For more information, see the Registry Console Overview figure above.

User Profile

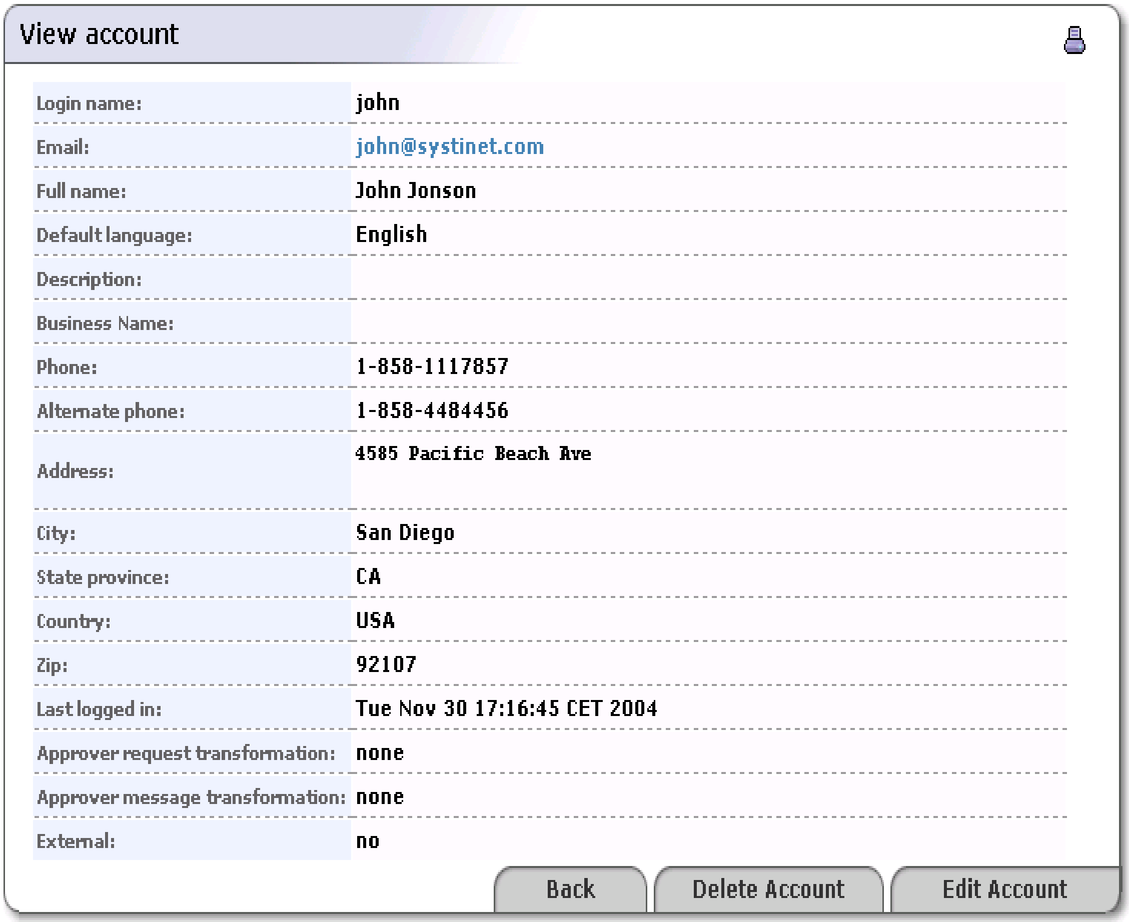
You can manage your user account, user groups, and favorite taxonomies under the Profile menu tab.

Profile Menu Tab



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

View Account



View account

Login name:	john
Email:	john@systinet.com
Full name:	John Jonson
Default language:	English
Description:	
Business Name:	
Phone:	1-858-1117857
Alternate phone:	1-858-4484456
Address:	4585 Pacific Beach Ave
City:	San Diego
State province:	CA
Country:	USA
Zip:	92107
Last logged in:	Tue Nov 30 17:16:45 CET 2004
Approver request transformation:	none
Approver message transformation:	none
External:	no

Field descriptions (self-explanatory fields are omitted):

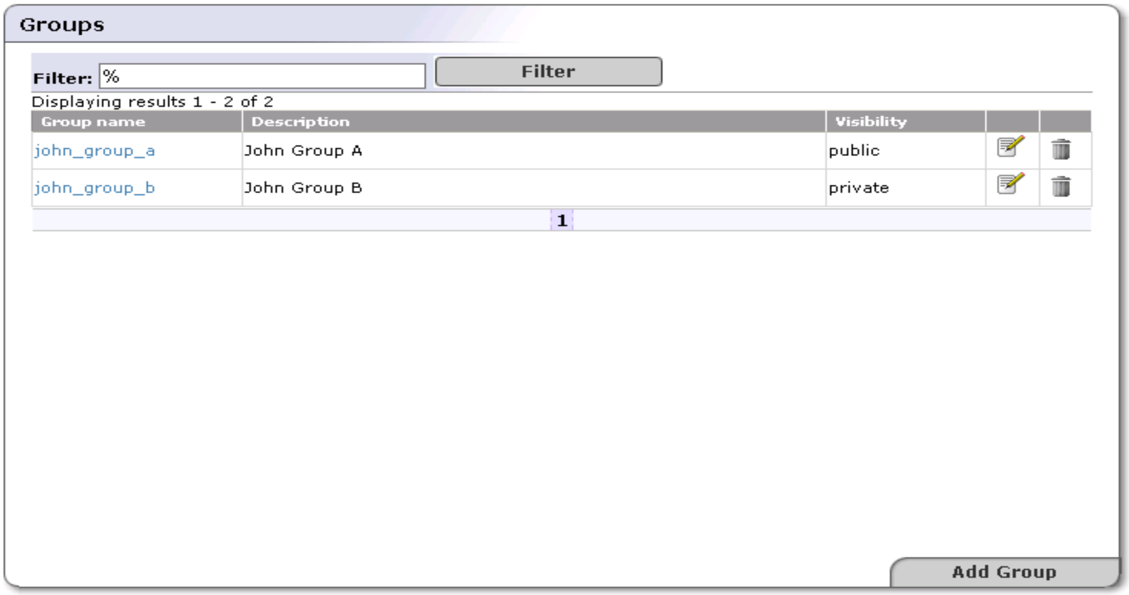
Default Language Code — Set the default language code. Used when publishing, it is the language code associated with a particular field when the language is not specified.

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

View User Groups



Displaying results 1 - 2 of 2

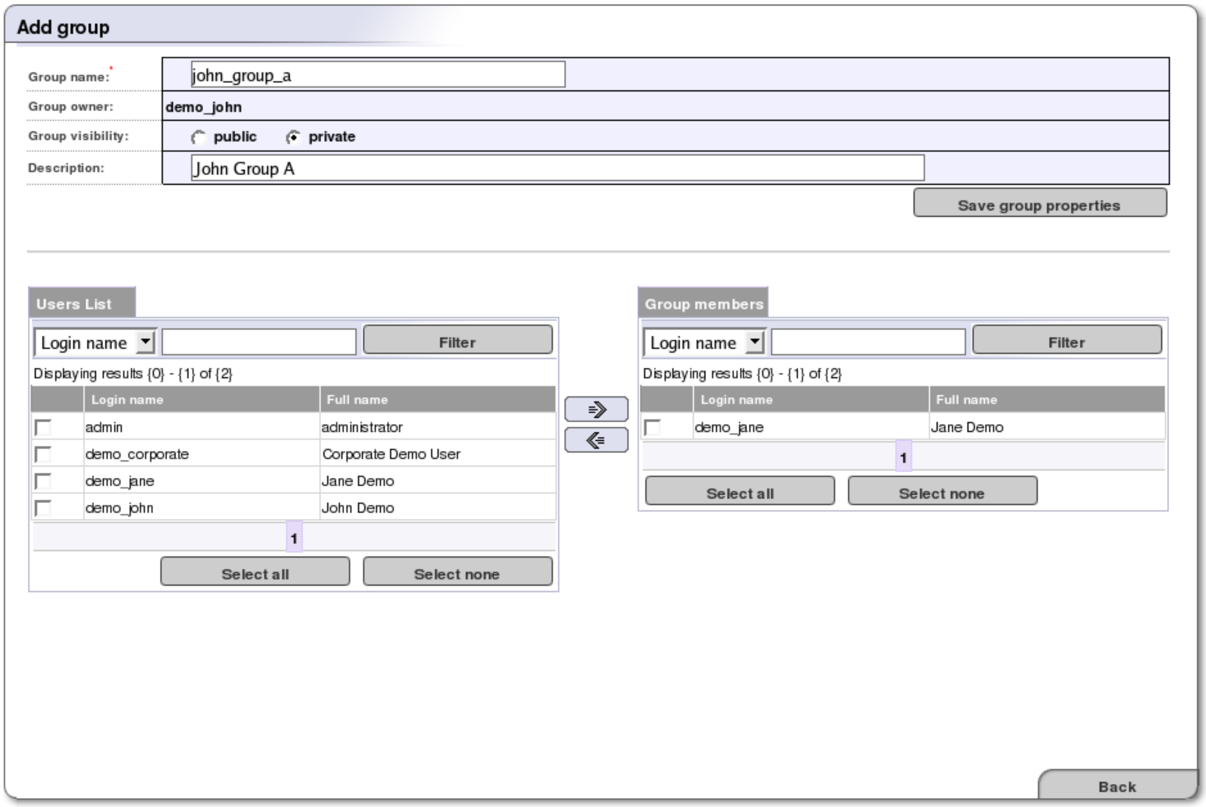
Group name	Description	Visibility		
john_group_a	John Group A	public		
john_group_b	John Group B	private		

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 the “View User Groups” screen above.
2. Click the **Add Group** button.

Edit Group Membership



Add group

Group name:*	john_group_a
Group owner:	demo_john
Group visibility:	○ public ● private
Description:	John Group A

Users List

Displaying results {0} - {1} of {2}

	Login name	Full name
□	admin	administrator
□	demo_corporate	Corporate Demo User
□	demo_jane	Jane Demo
□	demo_john	John Demo

Group members

Displaying results {0} - {1} of {2}

	Login name	Full name
□	demo_jane	Jane Demo

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 G**roup members** table.
7. Once users are added, click **Save Group** to update HPE SOA Registry Foundation.

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 the “View User Groups” screen above.

3. Click on the **Edit** button.
4. Use arrow buttons to add and remove users as shown in the “Edit Group Membership” screen above.

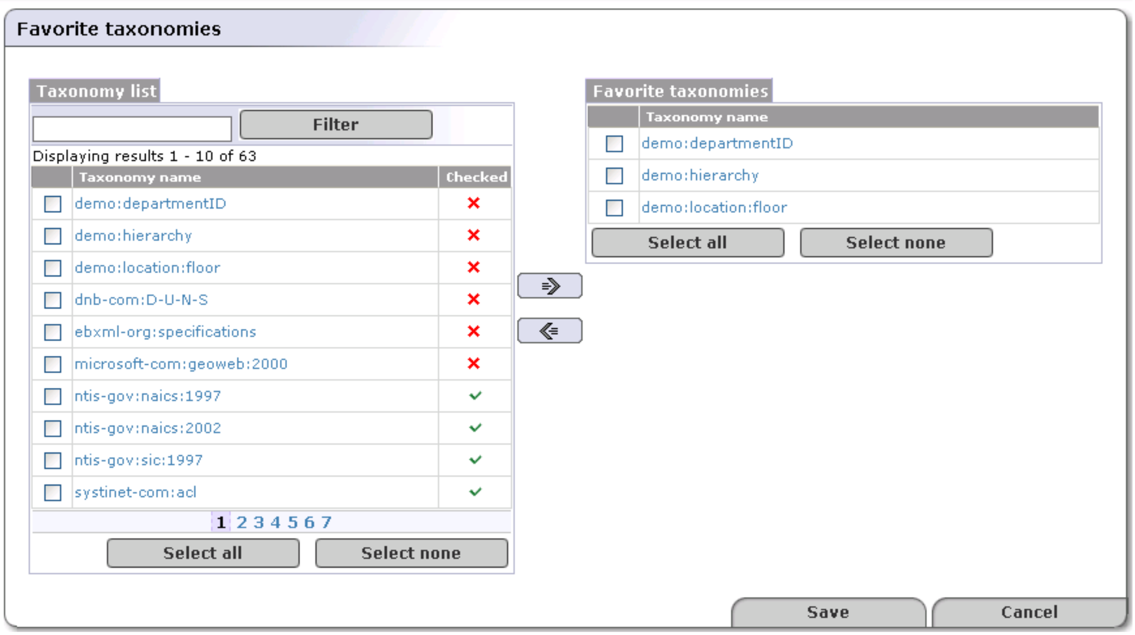
Favorite Taxonomies

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 the **Profile** menu tab. Click the **Favorite taxonomies** link. This returns the list of your favorite taxonomies shown in the “Manage Favorite Taxonomies” screen below.
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.

Manage Favorite Taxonomies



Displaying results 1 - 10 of 63

	Taxonomy name	Checked
□	demo:departmentID	✗
□	demo:hierarchy	✗
□	demo:location:floor	✗
□	dnb-com:D-U-N-S	✗
□	ebxml-org:specifications	✗
□	microsoft-com:geoweb:2000	✗
□	ntis-gov:naics:1997	✓
□	ntis-gov:naics:2002	✓
□	ntis-gov:sic:1997	✓
□	systinet-com:acl	✓

1 2 3 4 5 6 7

Favorite taxonomies

	Taxonomy name
□	demo:departmentID
□	demo:hierarchy
□	demo:location:floor

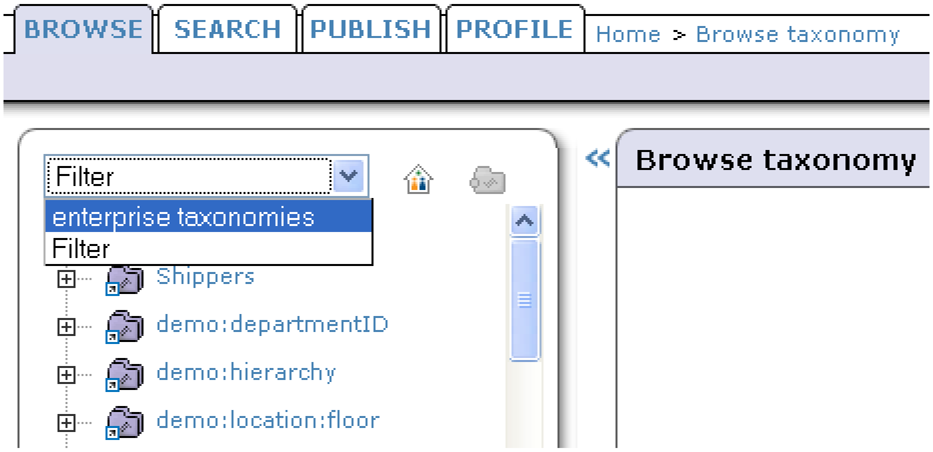
Browsing

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 HPE SOA Registry Foundation. This demonstration set is designed to help familiarize you with the registry

To browse taxonomies and UDDI entities:

- Click on the **Taxonomies** link under the **Browse** main menu tab.
  The following page will appear.

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 “Favorite Taxonomies” in "User Profile" on page 248. The list of enterprise taxonomies is defined by an administrator. For more information, see "Taxonomy Management" on page 310 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 of the taxonomy categories. Those with sub-categories can be expanded and collapsed.

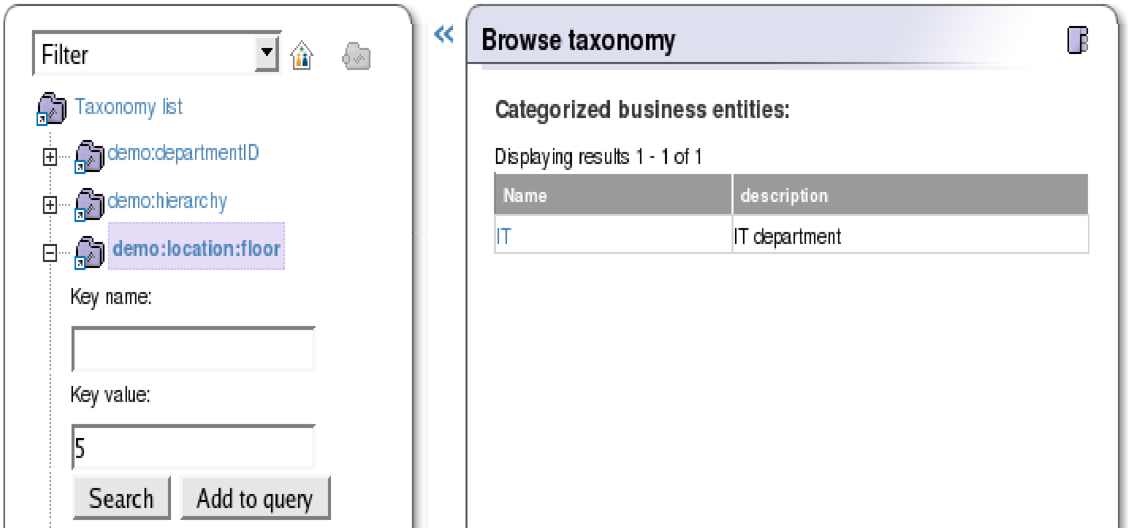
When you browse internally-checked taxonomies you can see their value set to see UDDI entities categorized by these key values. For unchecked or externally checked taxonomies, you can search UDDI entities by key values. We will show you how to browse an unchecked taxonomy from the demo data.

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

1. In the **Browse MenuTab**, select 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 the key value as 5, then click **Search**.

   You will get a list of UDDI entities categorized by this taxonomy with matching key value (IT in this case) as shown in the following Browse Demo figure.

   Browse Demo



Key value: 5

Categorized business entities:

Displaying results 1 - 1 of 1

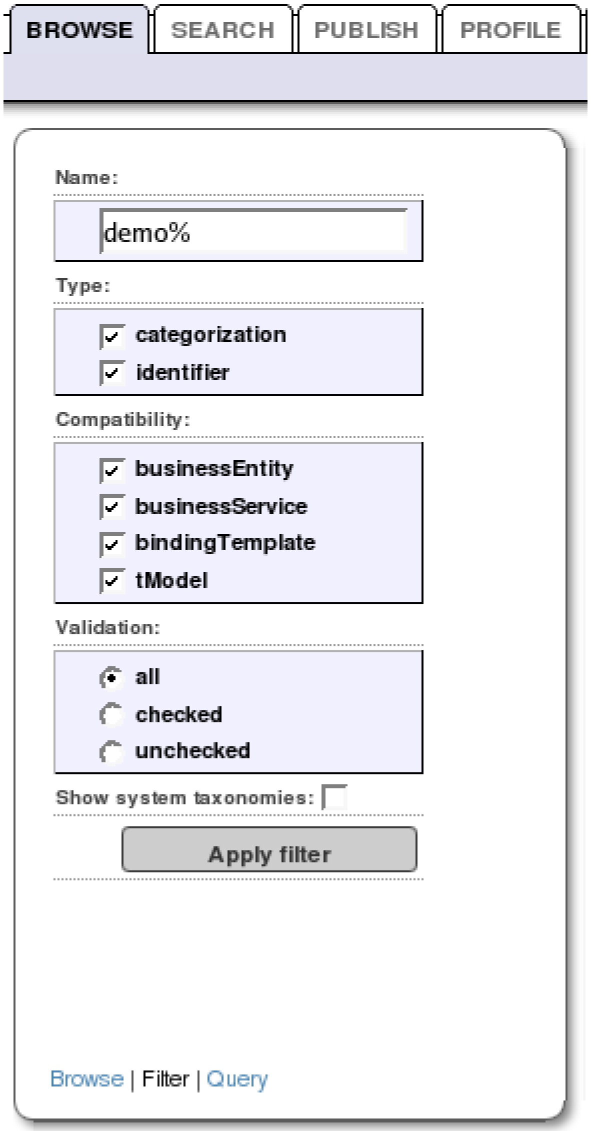
Name	description
IT	IT department

   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 following Taxonomy Filter page will appear.

Taxonomy Filter



Name: demo%

Type:

☑ categorization
☑ identifier

Compatibility:

☑ businessEntity
☑ businessService
☑ bindingTemplate
☑ tModel

Validation:

● all
○ checked
○ unchecked

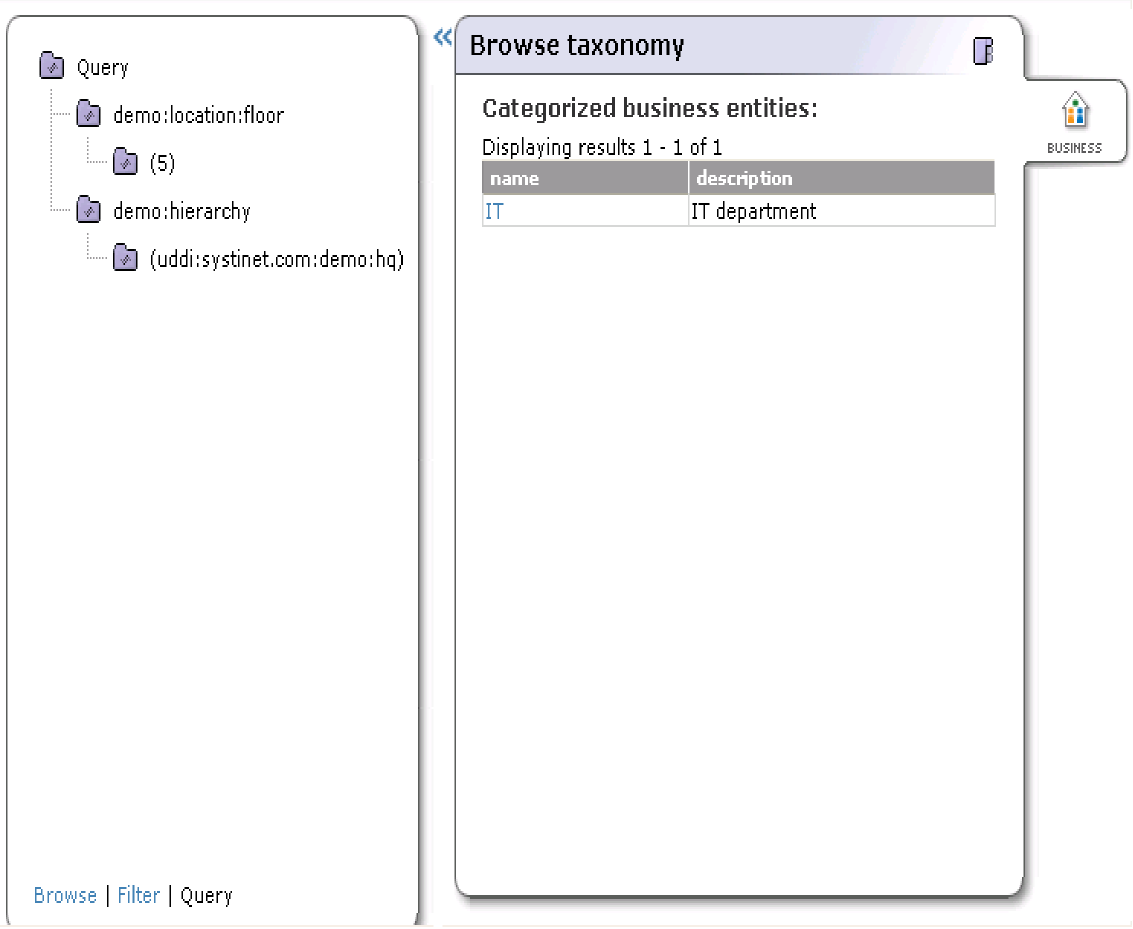
Show system taxonomies: □

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 the “Browse Demo" figure above.. Then, you can expand another taxonomy and specify a new criterion. The following Query page 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).

Query



Categorized business entities:

Displaying results 1 - 1 of 1

name	description
IT	IT department

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.

Searching

HPE SOA Registry Foundation 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 HPE SOA Registry Foundation when you know the key of the UDDI entity you want to retrieve.

Find Resources

You can search for resources:

- Find WSDL
- Find XSD

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

> **Note:** HPE SOA Registry Foundation 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 “Find Business by Categories” below for instructions on how to use the range queries functionality.

Find Business

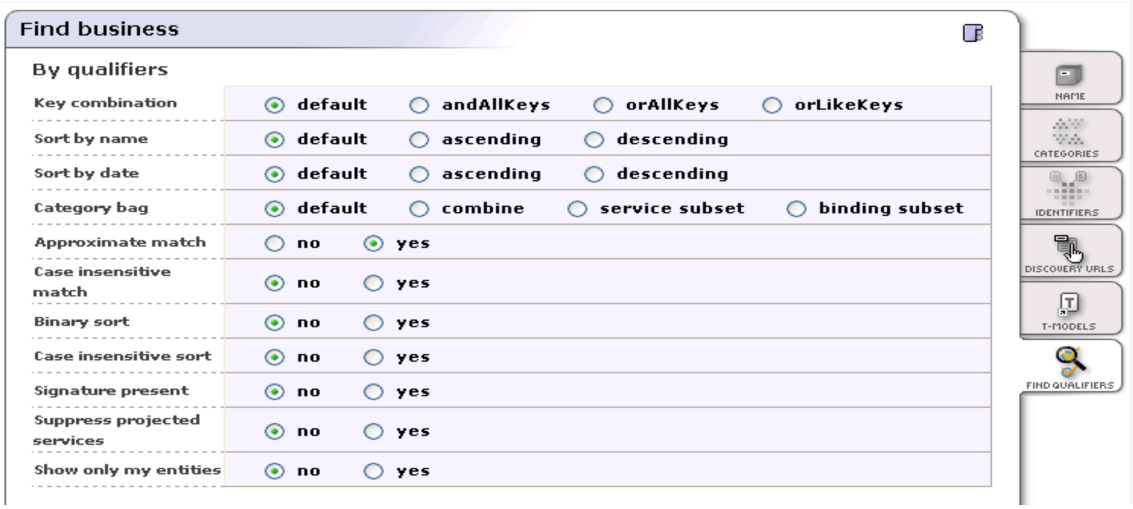
In this section, we cover locating business entities using a number of different methods. You can locate business entities by:

- Name
- Categories
- Identifiers

- Discovery URL
- tModel

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

Find Qualifiers



Find business

By qualifiers

Key combination	● default	○ andAllKeys	○ orAllKeys	○ orLikeKeys
Sort by name	● default	○ ascending	○ descending	
Sort by date	● default	○ ascending	○ descending	
Category bag	● default	○ combine	○ service subset	○ binding subset
Approximate match	○ no	● yes		
Case insensitive match	● no	○ yes		
Binary sort	● no	○ yes		
Case insensitive sort	● no	○ yes		
Signature present	● no	○ yes		
Suppress projected services	● no	○ yes		
Show only my entities	● no	○ yes		

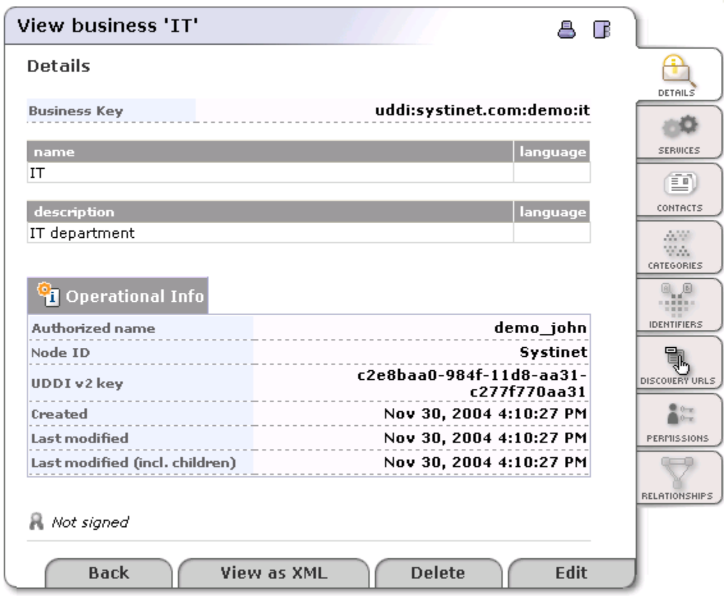
Find Business by Name

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 following View Business Detail page.

View Business Detail



View business 'IT'

Details

Business Key	uddi:systinet.com:demo:it

name	language
IT	

description	language
IT department	

Operational Info

Authorized name	demo_john
Node ID	Systinet
UDDI v2 key	c2e8baa0-984f-11d8-aa31-c277f770aa31
Created	Nov 30, 2004 4:10:27 PM
Last modified	Nov 30, 2004 4:10:27 PM
Last modified (incl. children)	Nov 30, 2004 4:10:27 PM

Not signed

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.

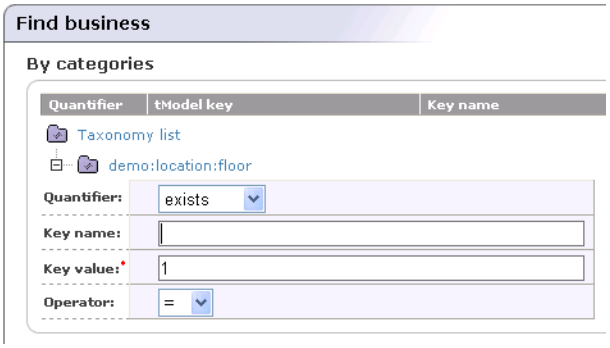
   You can switch the **Show** drop down list from **favorite taxonomies** to see all **taxonomies.** To manage **favorite taxonomies** see "User Profile" on page 248.
3. Click on the desired taxonomy.

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

   Select demo:location:floor from our demo data.
4. 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.

Find Business by Category



Find business

By categories

Quantifier	tModel key	Key name

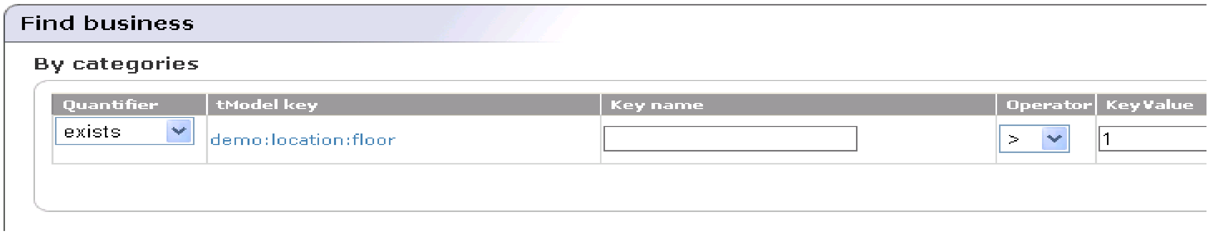
Quantifier:	exists
Key name:	
Key value:*	1
Operator:	=

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 following page is returned.

Find Business by Range Category



Find business

By categories

Quantifier	tModel key	Key name	Operator	Key Value
exists	demo:location:floor		>	1

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:

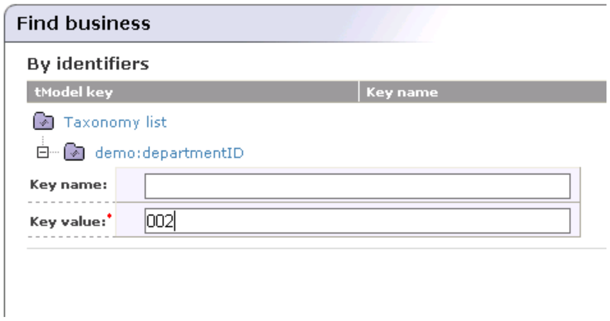
1. Under the main **Search** tab, click the **Businesses** link.
2. Click the **Identifiers** tab. Then click the **Add identifier** button. This returns a list of available taxonomies.
3. Click on the desired taxonomy.

   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**.

   Find Business by Identifier

   Key name:

   Key value:* 002
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:

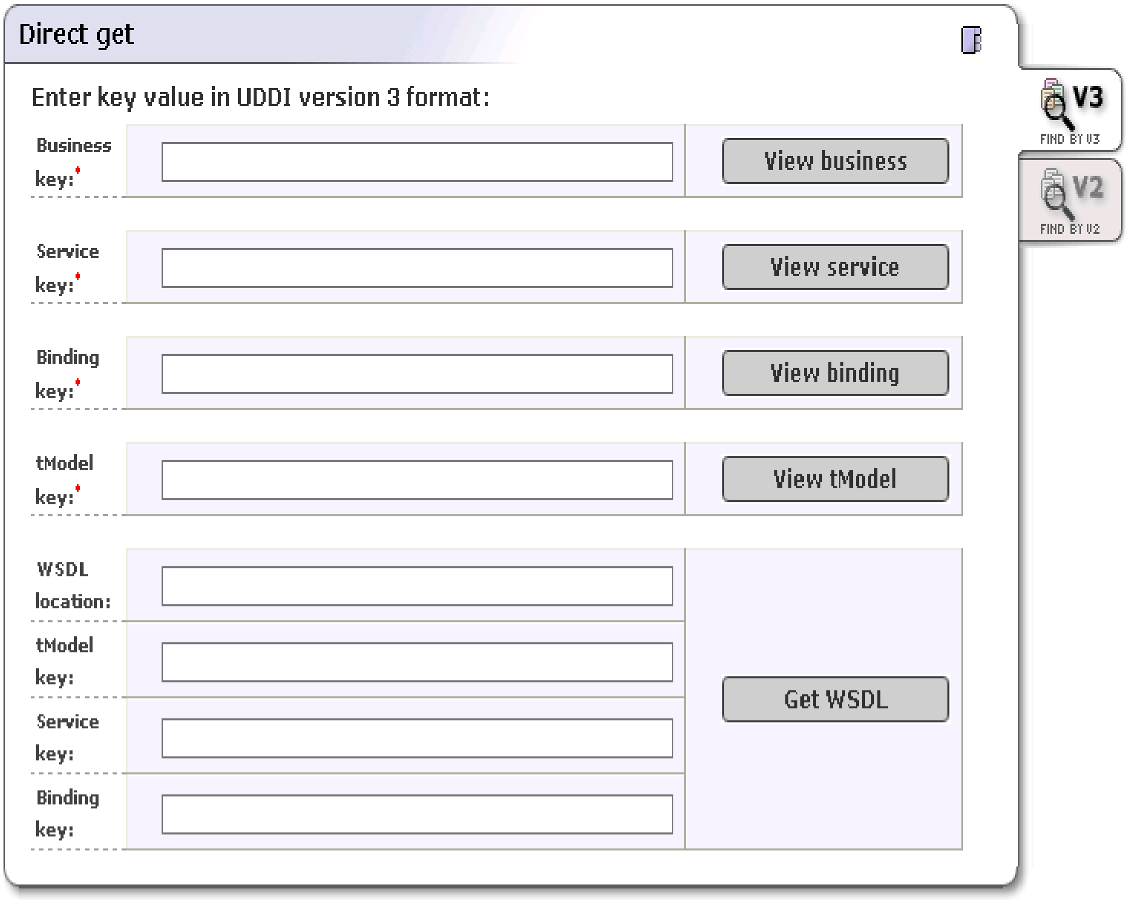
- Name
- Category
- Identifiers

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 HPE SOA Registry Foundation when you know the key of the UDDI structure you want to retrieve. HPE SOA Registry Foundation 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.

Direct Get



Enter key value in UDDI version 3 format:

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:
```

`uddinodebusinessKey&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
```

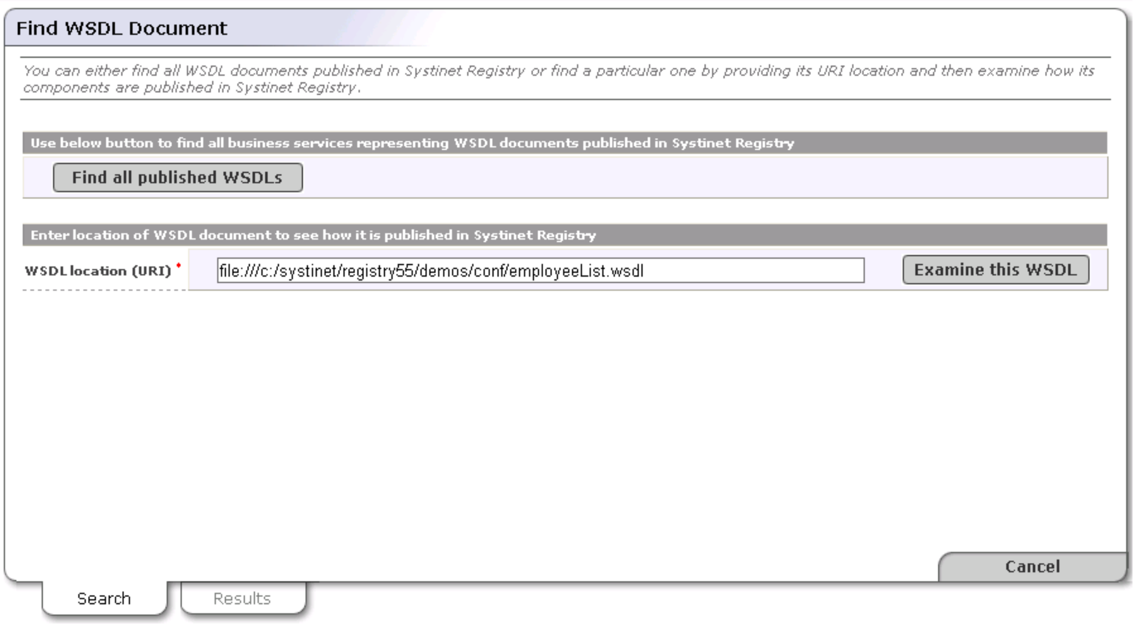
Find WSDL

You can find all WSDL documents published in HPE SOA Registry Foundation. When you supply the WSDL location URI, you can review how artifacts of the WSDL document are published in HPE SOA Registry Foundation. 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 HPE SOA Registry Foundation:

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

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

   Find WSDL



Find WSDL Document

You can either find all WSDL documents published in Systinet Registry or find a particular one by providing its URI location and then examine how its components are published in Systinet Registry.

Use below button to find all business services representing WSDL documents published in Systinet Registry

Enter location of WSDL document to see how it is published in Systinet Registry

WSDL location (URI) *: file:///c:/systinet/registry55/demos/conf/employeeList.wsdl

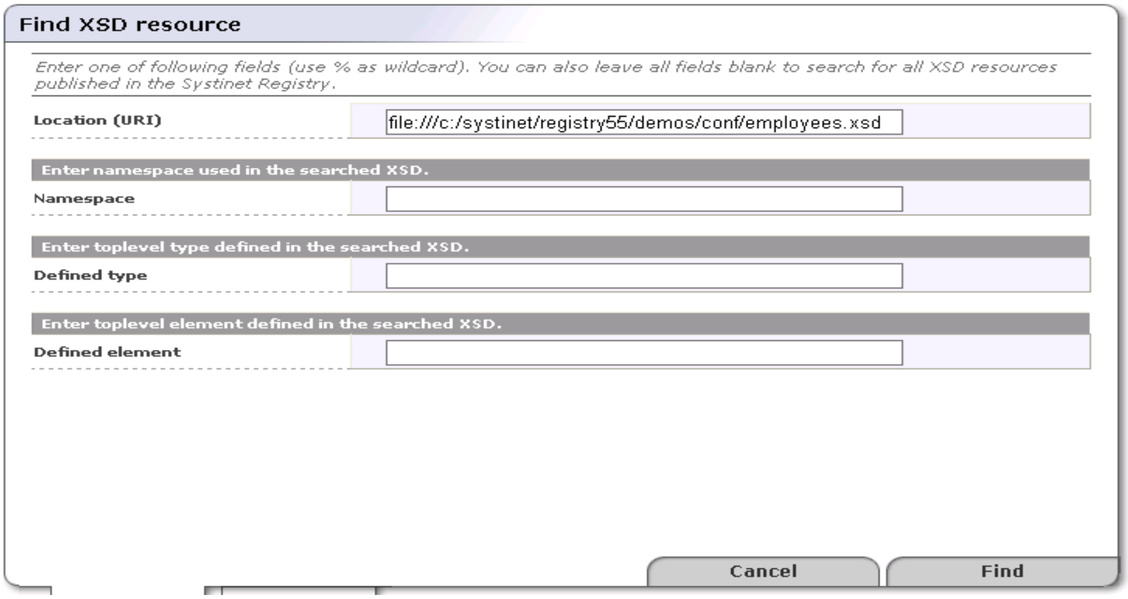
Find XSD

You can search for an XML Schema in HPE SOA Registry Foundation 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 following Find XSD page 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**.

   Find XSD



Find XSD resource

Enter one of following fields (use % as wildcard). You can also leave all fields blank to search for all XSD resources published in the Systinet Registry.

Location (URI): file:///c:/systinet/registry55/demos/conf/employees.xsd

Enter namespace used in the searched XSD.

Namespace:

Enter toplevel type defined in the searched XSD.

Defined type:

Enter toplevel element defined in the searched XSD.

Defined element:

Publishing

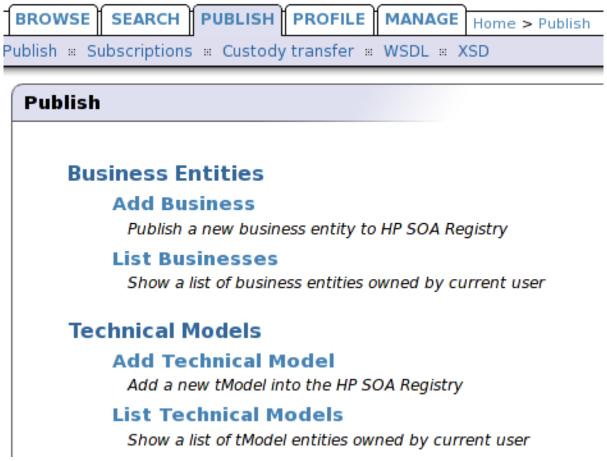
Publishing in HPE SOA Registry Foundation has several components:

- Publish UDDI core structures:
  - "Publishing a Business" on the next page
  - "Publishing a Service" on page 272

  - "Publishing a Binding Template" on page 272
  - "Publishing a tModel" on page 273
  - "Publishing Assertions" on page 275 - Asserting relationships between business entities.
- "Publishing Subscriptions" on page 277 - Subscribing interest in receiving alerts regarding changes made to a registry.
- "Publish Custody Transfer" on page 282- Transferring ownership of selected UDDI structures.
- Publish Resources
  - "Publishing WSDL Documents" on page 283 - Publishing Web Services Description Language documents (WSDL) to HPE SOA Registry Foundation.
  - "Publish XSD" on page 287- Publishing XML Schema Definition (XSD) Documents.

> **Note:** You must be logged into HPE SOA Registry Foundation to publish to it. There is a limitation of how many UDDI structures a user can store. See "Account Limits" in "Registry Management" on page 297 in the Administrator's Guide.

Publish Page



Business Entities

Add Business

Publish a new business entity to HP SOA Registry

List Businesses

Show a list of business entities owned by current user

Technical Models

Add Technical Model

Add a new tModel into the HP SOA Registry

List Technical Models

Show a list of tModel entities owned by current user

Publishing a Business

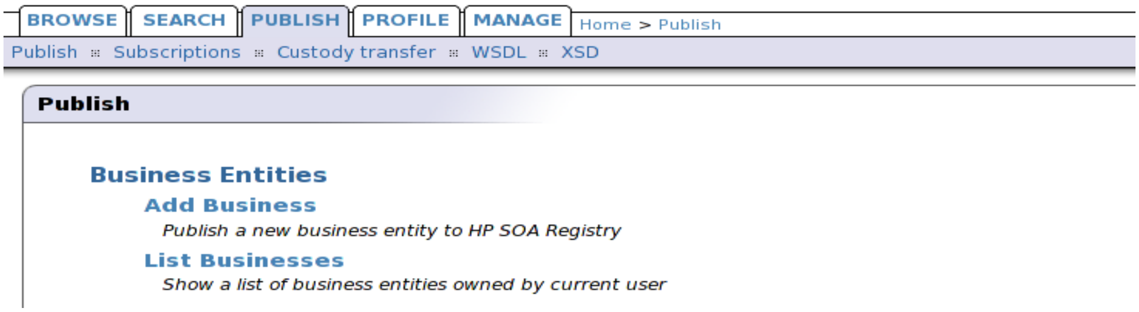
This section explains how to publish a businessEntity and edit businessEntity-related structures:

- 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**.

   Add Business

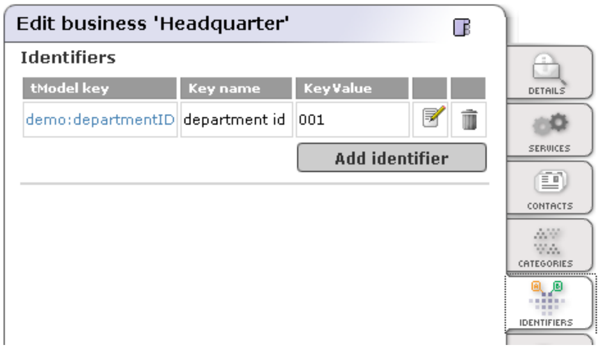
   

2. Enter the business name and a description, then click **Add Business**.
3. The business will appear in the left tree panel under the **Business entities node**.

To edit a business entity:

1. Select the **Publish** menu tab.
2. Click the **Publish** link.
3. Click the **List Businesses** link and click on edit icon next the name of business you wish to edit.

   Edit Business



Edit business 'Headquarter'

Identifiers

tModel key	Key name	KeyValue		
demo:departmentID	department id	001		

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.

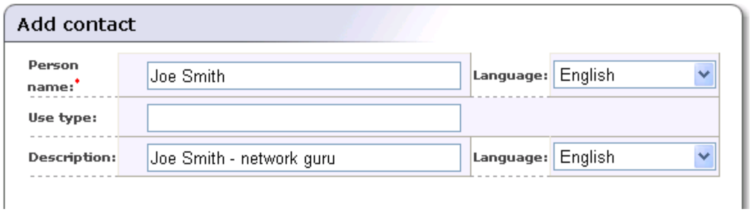
It is recommended that you use the description field to give a brief explanation of how the contact should be used.

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 following Add contact page where you can specify the contact's name and use type:

   Add Contact

   Add contact

Person name:*	Joe Smith	Language:	English
Use type:			
Description:	Joe Smith - network guru	Language:	English

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.

> **Note:** 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.

HPE SOA Registry Foundation 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:

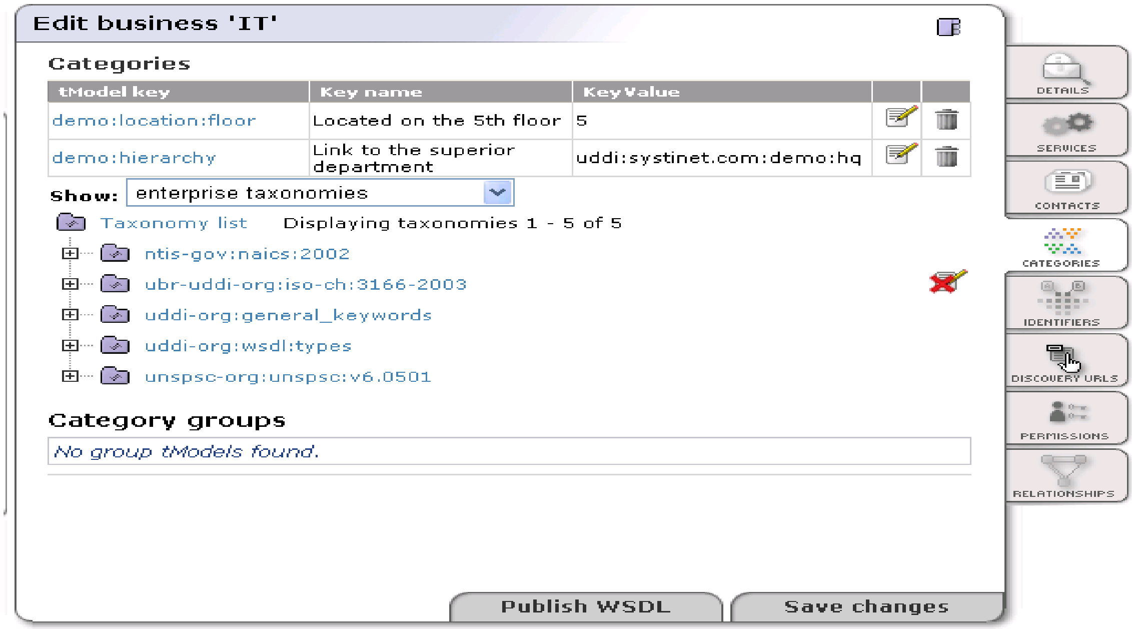
   a. Select Albania and click **Add category**.

   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**.

Add Category



Edit business 'IT'

Categories

tModel key	Key name	Key Value
demo:location:floor	Located on the 5th floor	5
demo:hierarchy	Link to the superior department	uddi:systinet.com:demo:hq

Show: enterprise taxonomies

Taxonomy list Displaying taxonomies 1 - 5 of 5

- ntis-gov:naics:2002
- ubr-uddi-org:iso-ch:3166-2003
- uddi-org:general_keywords
- uddi-org:wsdl:types
- unspsc-org:unspsc:v6.0501

Category groups

No group tModels found.

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.

   > **Note:** 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.

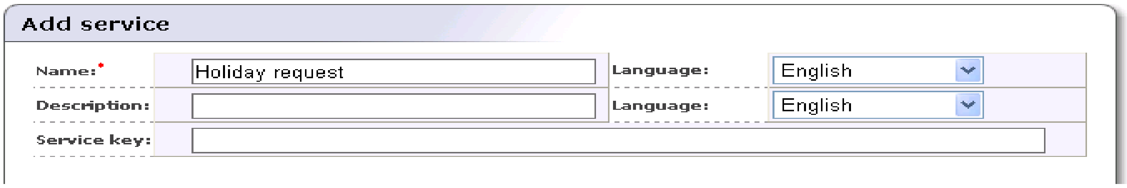
Publishing a Service

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.

   Add Service

   Add service

Name:*	Holiday request	Language:	English
Description:		Language:	English
Service key:			

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`.

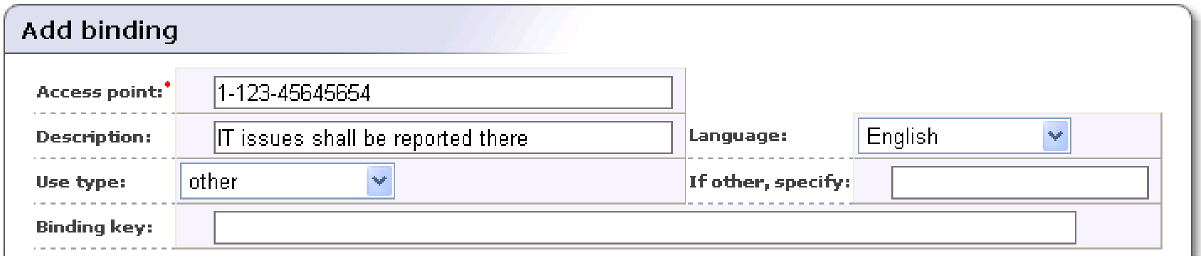
> **Note:** Alternatively, reference to another bindingTemplate can be stored in a hostingRedirector structure instead of in an accessPoint. However the hostingRedirector structure (not the hostingRedirector value of useType) is a relic of UDDI v2 and is deprecated in UDDI v3.

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.

   Add Binding



Add binding

Access point:*	1-123-45645654		
Description:	IT issues shall be reported there	Language:	English
Use type:	other	If other, specify:	
Binding key:			

Publishing a tModel

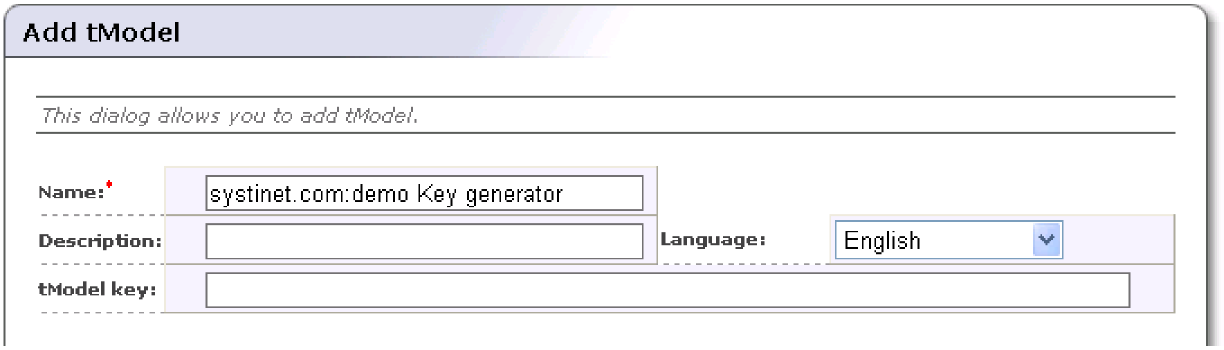
The tModel is a structure that takes the form of keyed metadata (data about data). In a general sense, the purpose of a tModel within HPE SOA Registry Foundation is to provide a reference system based on abstraction. Among the roles that a tModel plays in UDDI is the ability to provide and to describe compliance with a specification or concept, to a taxonomy, for example.

To publish a tModel:

1. Select the **Publish** menu 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.

   Add tModel

   Add tModel

   This dialog allows you to add tModel.

Name:*	systinet.com:demo Key generator		
Description:		Language:	English
tModel key:			

3. Enter the 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 HPE SOA Registry Foundation Administrator can change this behavior that tModels will be only marked as deleted. See "Node" on page 341 in the Administrator's Guide.

Adding a Category

In this section you will see how to assign demo:location:floor taxonomy to the numeric ordering as shown in the “Example of Numeric Categorization” in "Types of key Values" in "Taxonomy: Principles, Creation and Validation" on page 224.

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 HPE SOA Registry Foundation 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:

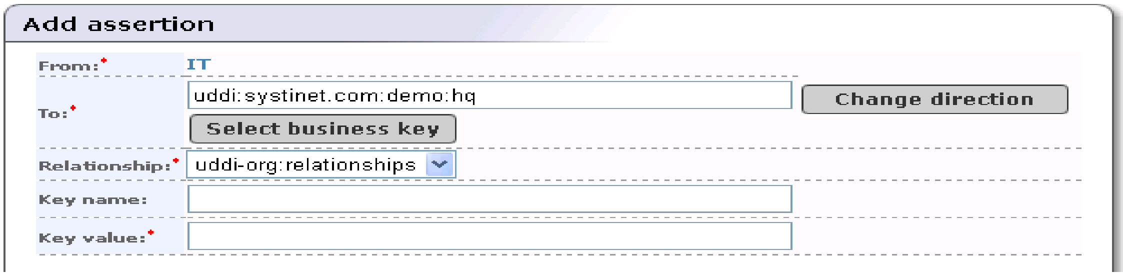
- The identity of the business from which the assertion is being made
- The identity of the business to which it is making a claim. HPE SOA Registry Foundation specifies these business identities through their UUID keys.
- A reference explaining the nature of the relationship. References about the nature 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.

   Add Assertion



Add assertion

From:* IT

To:* uddi:systinet.com:demo:hq

Relationship:* uddi-org:relationships

Key name:

Key value:*

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.

> **Note:** 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 HPE SOA Registry Foundation. 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.

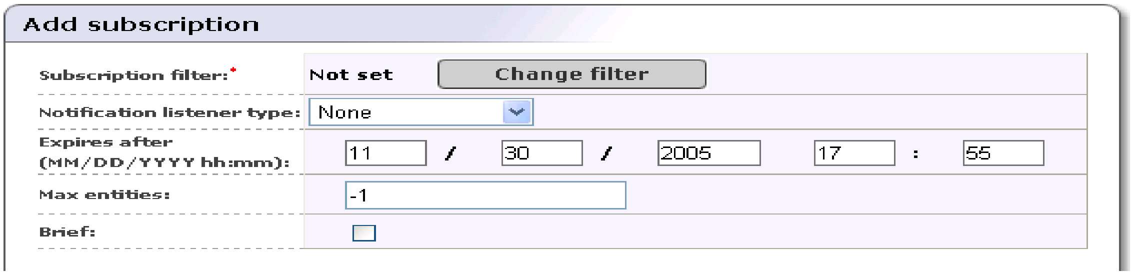
Subscriptions enable:

- 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

Add Subscription



Add subscription

Subscription filter:*	Not set	Change filter
Notification listener type:	None	
Expires after (MM/DD/YYYY hh:mm):	11 / 30 / 2005 17 : 55	
Max entities:	-1	
Brief:	□	

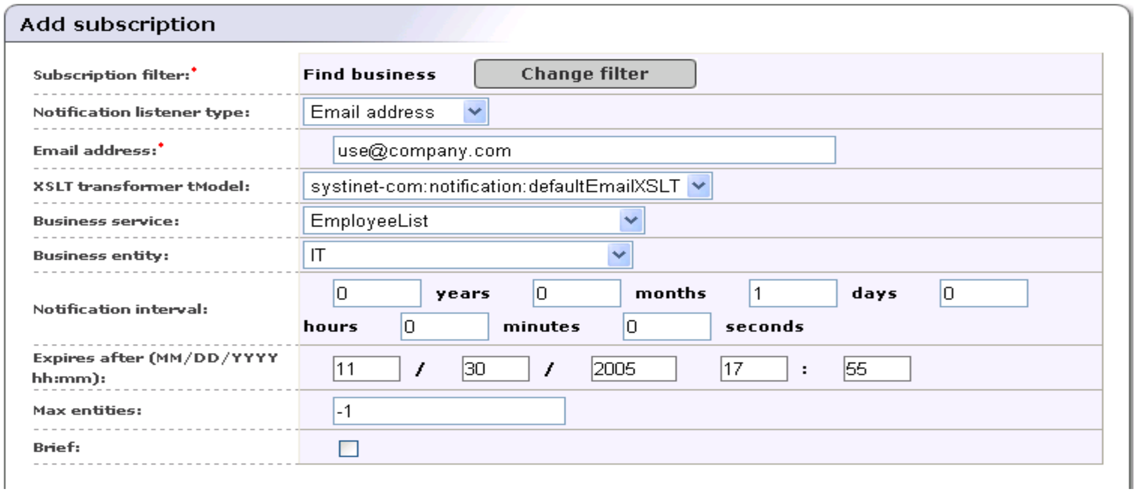
Adding Subscriptions

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 the “Add Subscription” figure above.
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 the “Add Subscription” figure above.
9. Fill in the other subscription fields if needed. These are described below.

Notification Listener Types

Add Subscription - Email Notification Listener Type

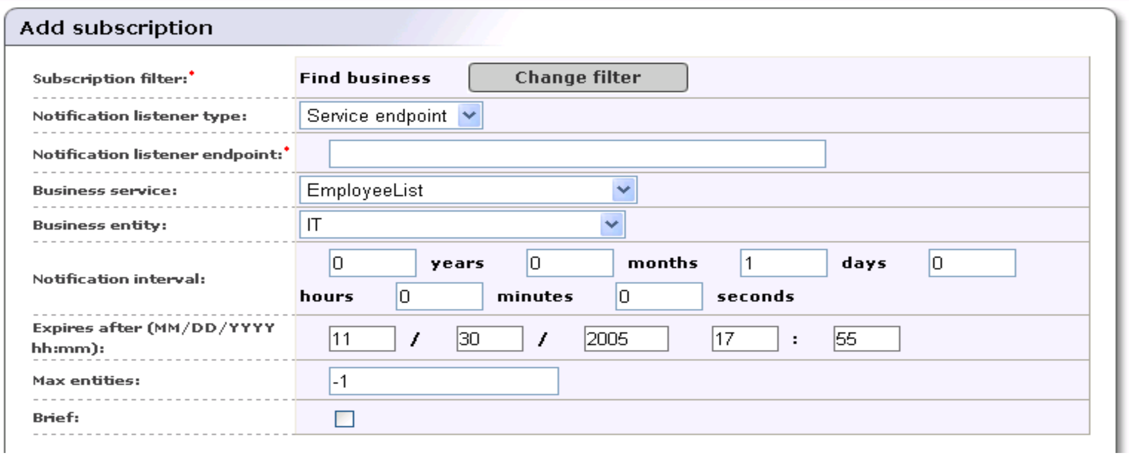


Add subscription

Subscription filter:*	Find business
Notification listener type:	Email address
Email address:*	use@company.com
XSLT transformer tModel:	systinet-com:notification:defaultEmailXSLT
Business service:	EmployeeList
Business entity:	IT
Notification interval:	0 years 0 months 1 days 0 hours 0 minutes 0 seconds
Expires after (MM/DD/YYYY hh:mm):	11 / 30 / 2005 17 : 55
Max entities:	-1
Brief:	□

- **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.

Add Subscription - Service Endpoint Listener Type

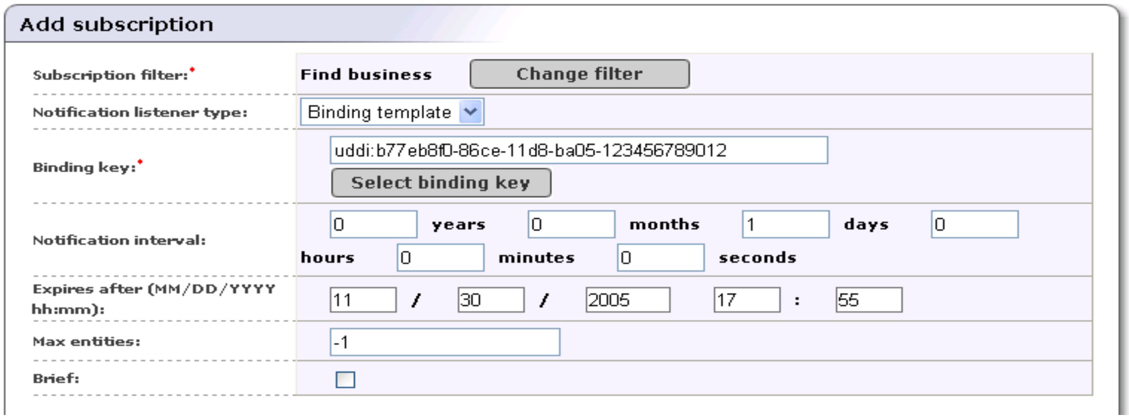


Add subscription

Subscription filter:*	Find business [Change filter]
Notification listener type:	Service endpoint
Notification listener endpoint:*	
Business service:	EmployeeList
Business entity:	IT
Notification interval:	0 years 0 months 1 days 0 hours 0 minutes 0 seconds
Expires after (MM/DD/YYYY hh:mm):	11 / 30 / 2005 17 : 55
Max entities:	-1
Brief:	□

- **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.

Add Subscription - Binding Template Listener Type



Subscription filter:*	Find business
Notification listener type:	Binding template
Binding key:*	uddi:b77eb8f0-86ce-11d8-ba05-123456789012
Notification interval:	0 years 0 months 1 days 0 hours 0 minutes 0 seconds
Expires after (MM/DD/YYYY hh:mm):	11 / 30 / 2005 17 : 55
Max entities:	-1
Brief:	□

- **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.

> **Note:** 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

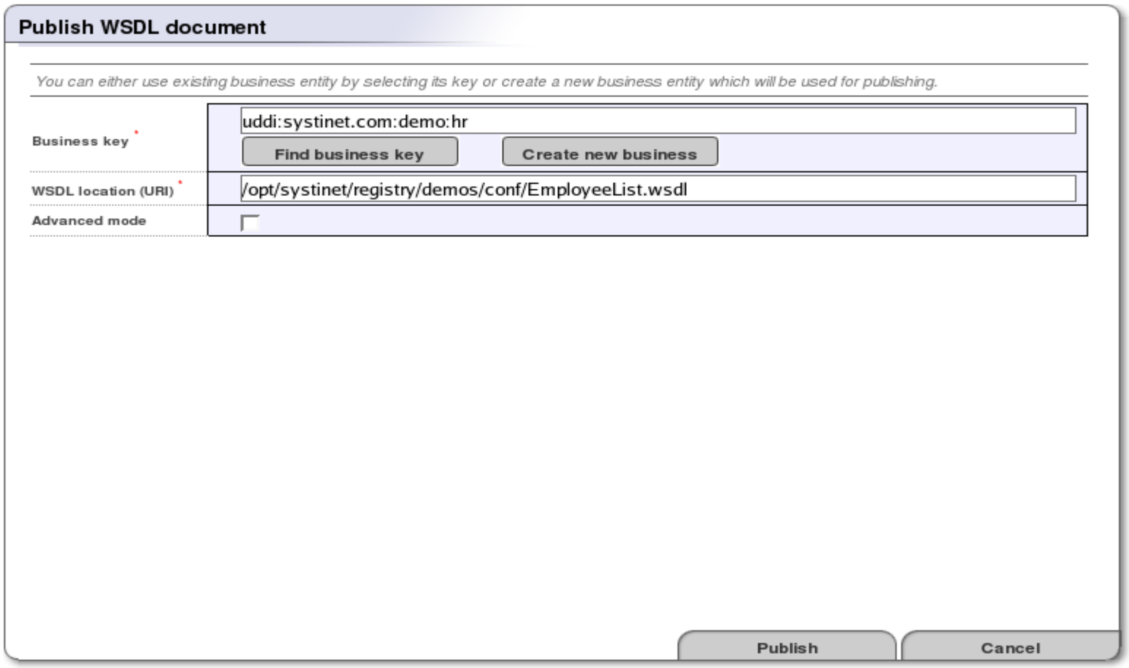
HPE SOA Registry Foundation WSDL to UDDI (WSDL2UDDI) mapping is compliant with OASIS's technical note Using WSDL in a UDDI registry Version 2.0 [http://www.oasis-open.org/committees/uddispec/doc/tn/uddi-spec-tc-tn-wsdl-v200-20031104.htm]. It enables the automatic publishing of WSDL documents to UDDI, 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 following page will appear.

   Publish WSDL



Publish WSDL document

You can either use existing business entity by selecting its key or create a new business entity which will be used for publishing.

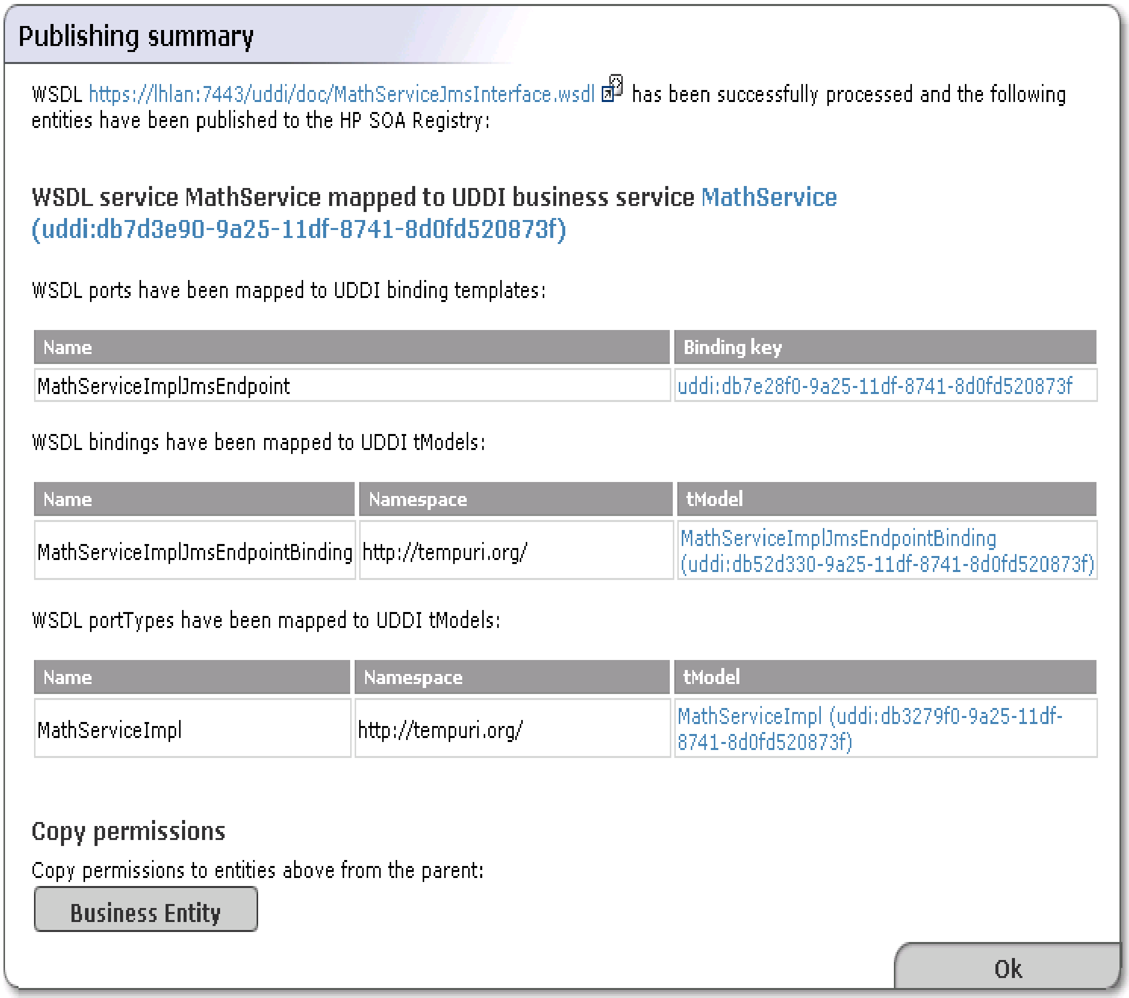
Business key *	uddi:systinet.com:demo:hr
WSDL location (URI) *	/opt/systinet/registry/demos/conf/EmployeeList.wsdl
Advanced mode	□

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 HPE SOA Registry Foundation demos from `REGISTRY_HOME/demos/conf/employeeList.wsdl`.
5. Leave the **Advanced mode** check box unchecked, then click **Publish** button.

> **Note:** Publishing a WSDL document by using basic authentication, HPE SOA Registry Foundation must be started with the below parameters from startup command-line:
>
> ```
> -Dsystinet.wsdlpublishing_http.username=admin
> ```
>
> ```
> -Dsystinet.wsdlpublishing_http.password=changeit
> ```

The WSDL document will be published to HPE SOA Registry Foundation. You can review how WSDL artifacts of the document have been mapped to the HPE SOA Registry Foundation as shown in the following figure.

Publish WSDL Summary



Publishing summary

WSDL https://lhlan:7443/uddi/doc/MathServiceJmsInterface.wsdl has been successfully processed and the following entities have been published to the HP SOA Registry:

WSDL service MathService mapped to UDDI business service MathService (uddi:db7d3e90-9a25-11df-8741-8d0fd520873f)

WSDL ports have been mapped to UDDI binding templates:

Name	Binding key
MathServiceImplJmsEndpoint	uddi:db7e28f0-9a25-11df-8741-8d0fd520873f

WSDL bindings have been mapped to UDDI tModels:

Name	Namespace	tModel
MathServiceImplJmsEndpointBinding	http://tempuri.org/	MathServiceImplJmsEndpointBinding (uddi:db52d330-9a25-11df-8741-8d0fd520873f)

WSDL portTypes have been mapped to UDDI tModels:

Name	Namespace	tModel
MathServiceImpl	http://tempuri.org/	MathServiceImpl (uddi:db3279f0-9a25-11df-8741-8d0fd520873f)

Copy permissions

Copy permissions to entities above from the parent:

The **Business Entity** button in the summary screen enables you to copy permissions from the business entity for which the WSDL was published to all other entities involved in the publishing operation. If the entities already contain permissions, the permission lists are merged.

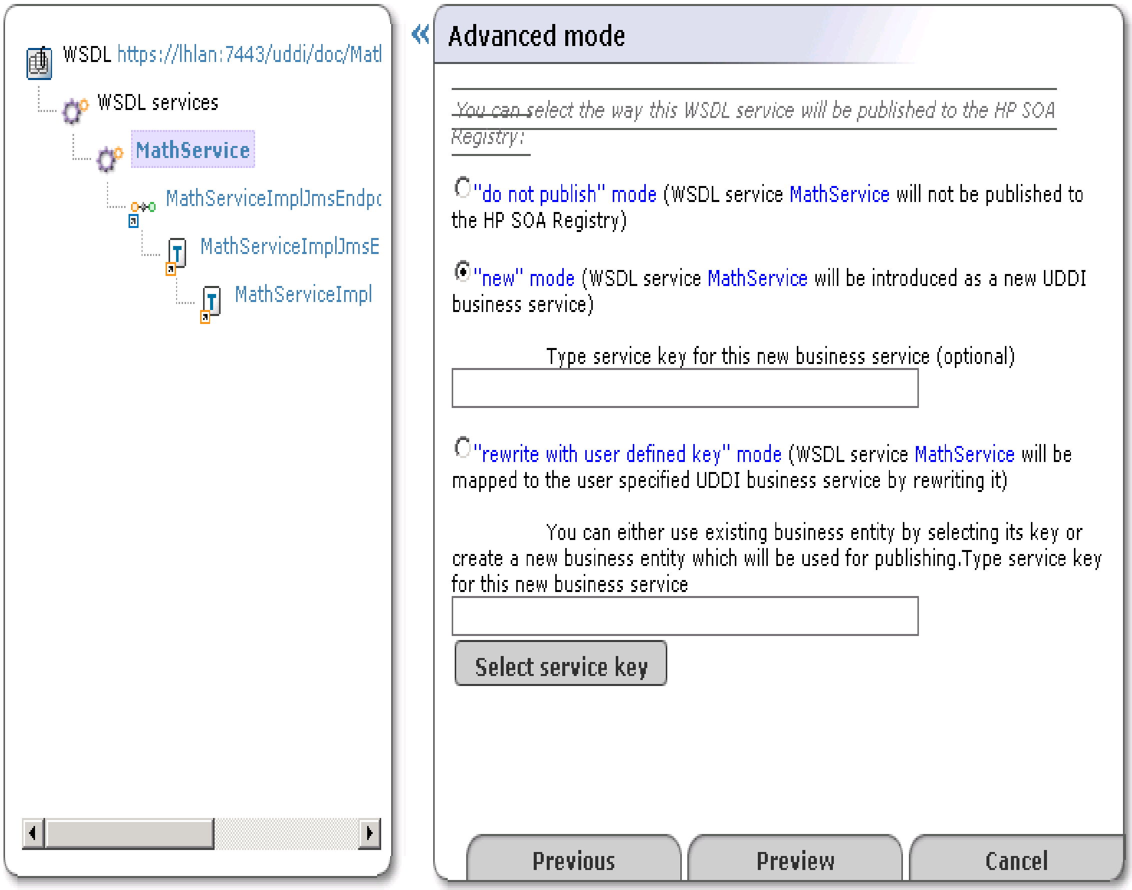
When a business service is reused during the publishing step, it may also contain permissions to distribute to associated binding templates and tModels.

The Business Service button, shown only when a business service is reused, enables you to copy permissions from a business service to associated binding templates and tModels involved in the publishing operation. If the entities already contain permissions, the permission lists are merged.

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 following Advanced Mode Publish page will appear.

Publish WSDL (Advanced Mode)



You can select the way this WSDL service will be published to the HP SOA Registry:

○ "do not publish" mode (WSDL service MathService will not be published to the HP SOA Registry)

● "new" mode (WSDL service MathService will be introduced as a new UDDI business service)

Type service key for this new business service (optional)

○ "rewrite with user defined key" mode (WSDL service MathService will be mapped to the user specified UDDI business service by rewriting it)

You can either use existing business entity by selecting its key or create a new business entity which will be used for publishing.Type service key for this new business service

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 HPE SOA Registry Foundation. 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 the above figure 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.

HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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 XSD

HPE SOA Registry Foundation 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.

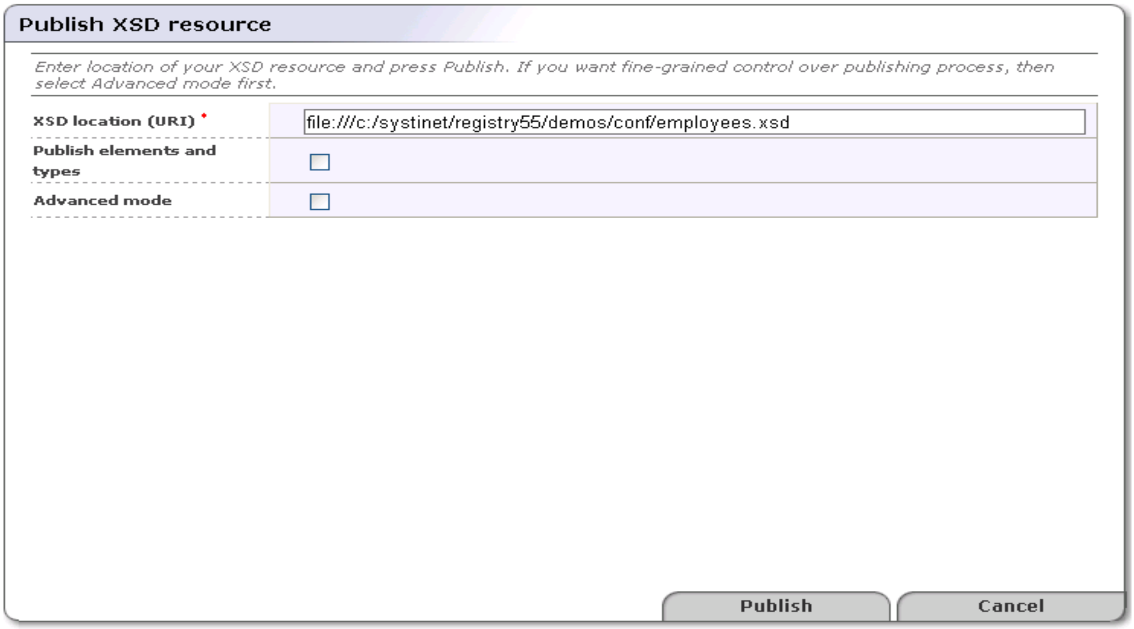
If you want to unpublish an XML schema document, use the Find XSD button and click the Unpublish button in the search result page.

Publishing an XML Schema

To publish an XML Schema document:

1. Click on the **XSD** link under the **Publish** main menu tab.
2. The following page will appear.

   Publish XSD



Publish XSD resource

Enter location of your XSD resource and press Publish. If you want fine-grained control over publishing process, then select Advanced mode first.

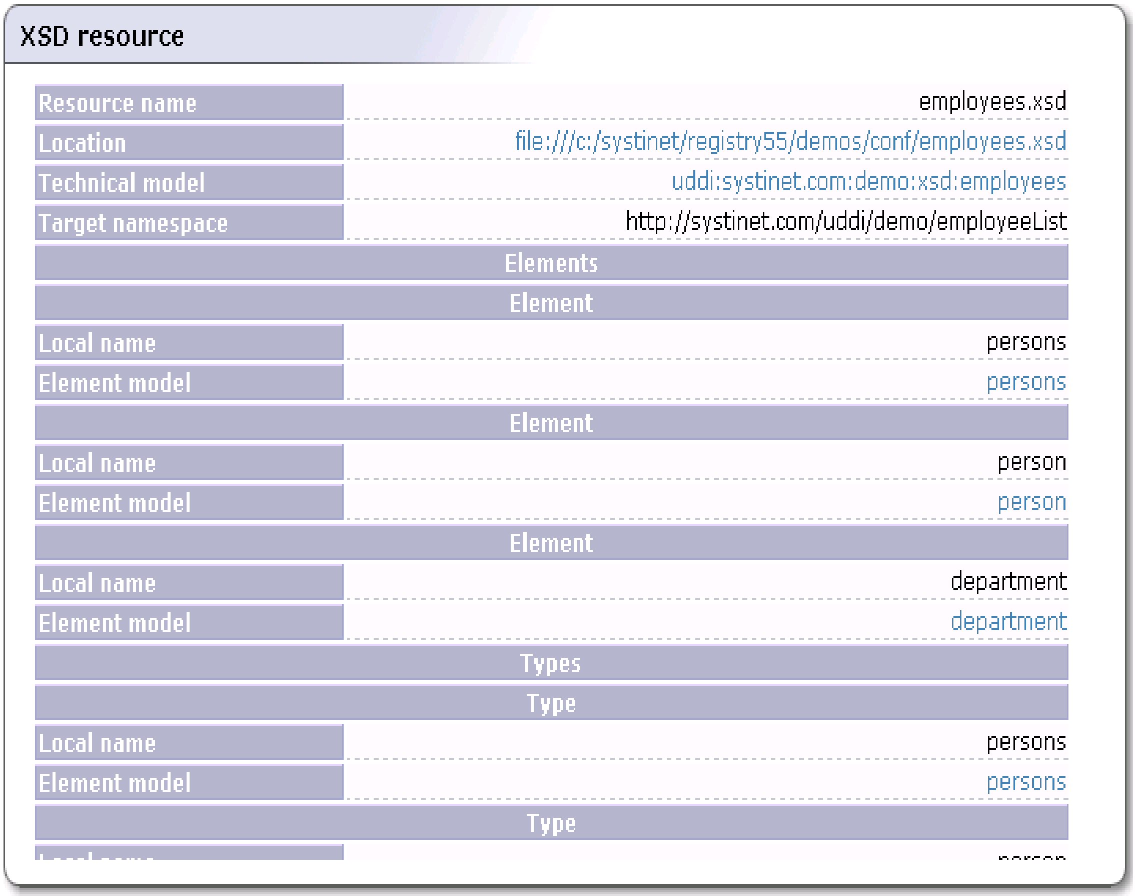
XSD location (URI) *	file:///c:/systinet/registry55/demos/conf/employees.xsd
Publish elements and types	□
Advanced mode	□

3. Enter an **XML Schema location**. To demonstrate, use the file `REGISTRY_HOME/demos/conf/employees.xsd` from the HPE SOA Registry Foundation 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 as shown in the following figure.

   Publish XSD Summary



XSD resource

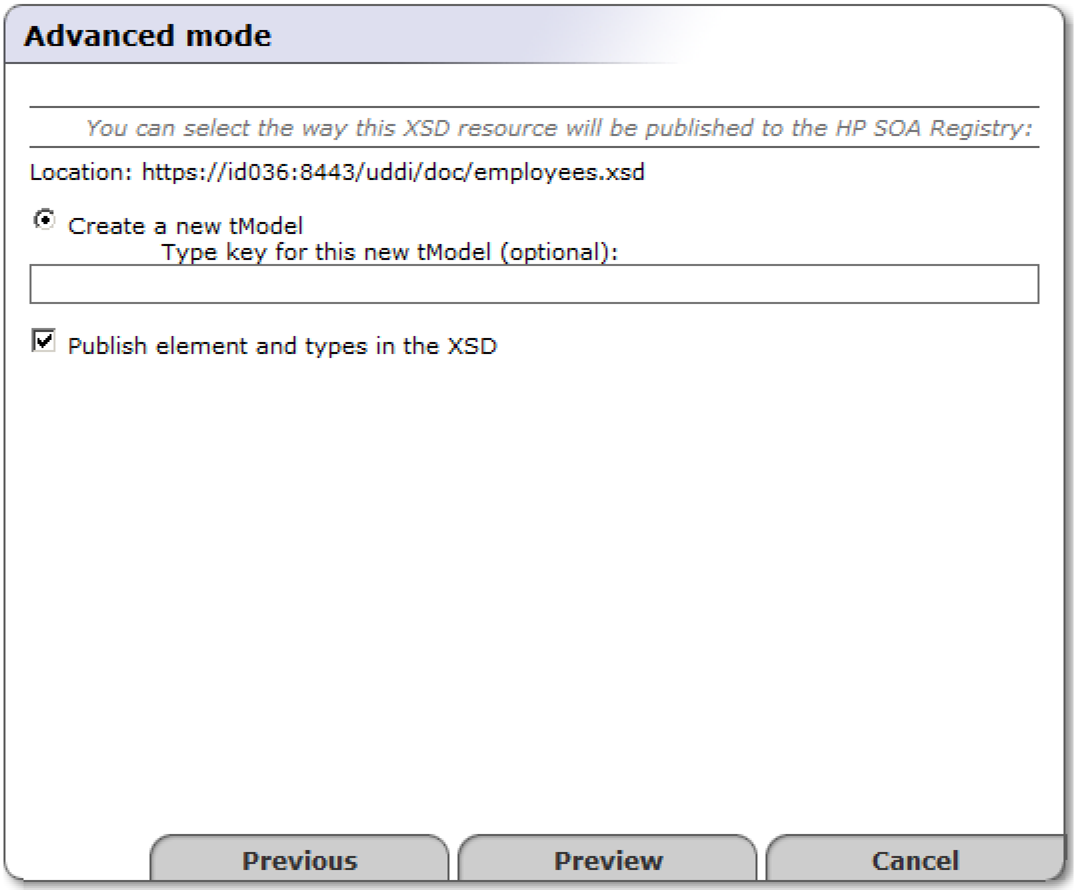
Resource name	employees.xsd
Location	file:///c:/systinet/registry55/demos/conf/employees.xsd
Technical model	uddi:systinet.com:demo:xsd:employees
Target namespace	http://systinet.com/uddi/demo/employeeList
Elements	
Element	
Local name	persons
Element model	persons
Element	
Local name	person
Element model	person
Element	
Local name	department
Element model	department
Types	
Type	
Local name	persons
Element model	persons
Type	
Local name	person

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 following Advanced Mode Publish page.

   Publish XSD - Advanced



Advanced mode

You can select the way this XSD resource will be published to the HP SOA Registry:

Location: https://id036:8443/uddi/doc/employees.xsd

● Create a new tModel
Type key for this new tModel (optional):

☑ Publish element and types in the XSD

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 HPE SOA Registry Foundation. The explanatory instructions in the right panel describe the mapping options.

4. Click **Preview** to see how the XML Schema document will be mapped to HPE SOA Registry Foundation. 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 HPE SOA Registry Foundation. To unpublish an XML Schema document, you must search for the XML Schema document first.

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 HPE SOA Registry Foundation Signer tool simplifies signature manipulation. You can find this tool's script in the bin directory of your HPE SOA Registry Foundation 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 "System Requirements" on page 37 in the Installation Guide.

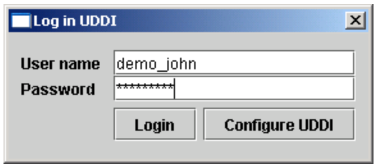
Starting the Signer

1. To start the Signer tool, first ensure that HPE SOA Registry Foundation is running, then execute the following script from the bin subdirectory of your HPE SOA Registry Foundation 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.

Login Dialog



Log in UDDI

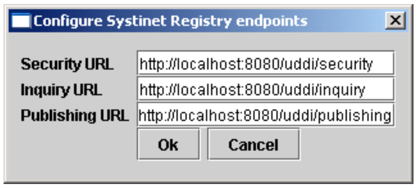
User name: demo_john

Password: *********

3. On the returned screen, set the endpoints of the Security, Inquiry, and Publishing Web services.

For help, ask the administrator of your registry.

Configure Dialog



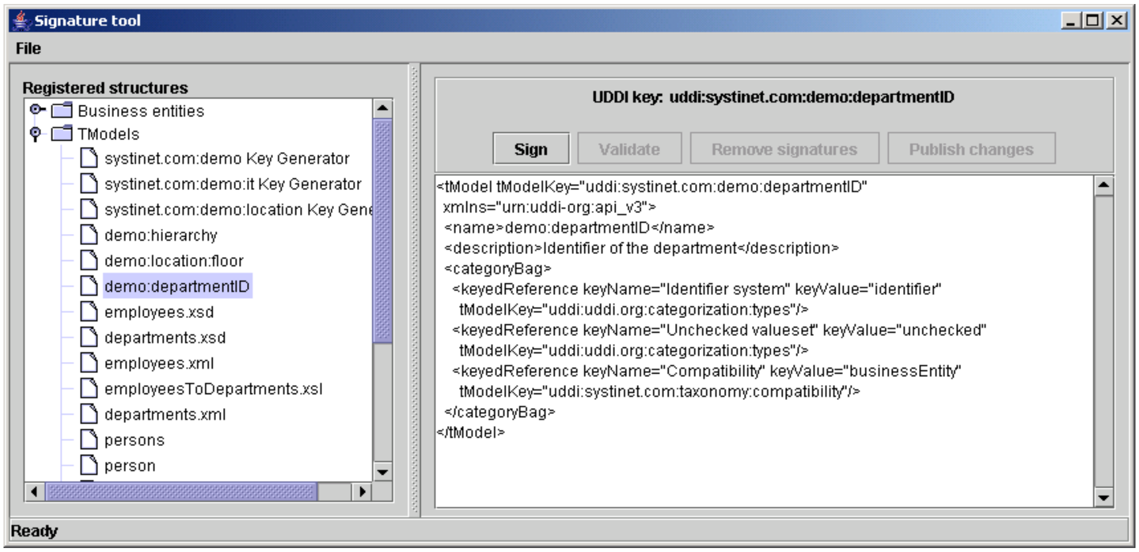
Security URL: http://localhost:8080/uddi/security
Inquiry URL: http://localhost:8080/uddi/inquiry
Publishing URL: http://localhost:8080/uddi/publishing

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 HPE SOA Registry Foundation for your registered information (businessEntities and tModels that you have published).

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.

Signature Tool - Main Screen



UDDI key: uddi:systinet.com:demo:departmentID

```
<tModel tModelKey="uddi:systinet.com:demo:departmentID"
 xmlns="urn:uddi-org:api_v3">
 <name>demo:departmentID</name>
 <description>Identifier of the department</description>
 <categoryBag>
  <keyedReference keyName="Identifier system" keyValue="identifier"
   tModelKey="uddi:uddi.org:categorization:types"/>
  <keyedReference keyName="Unchecked valueset" keyValue="unchecked"
   tModelKey="uddi:uddi.org:categorization:types"/>
  <keyedReference keyName="Compatibility" keyValue="businessEntity"
   tModelKey="uddi:systinet.com:taxonomy:compatibility"/>
 </categoryBag>
</tModel>
```

The status bar at the bottom of the application informs the user of current action progress and results.

Sign

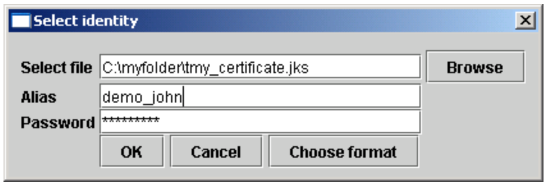
To sign a UDDI structure, you must set up the Java keystore. Use JDK tool keytool to generate the keystore. Please, see your JDK documentation for more information how to use keytool. The Signer tool has been tested with keystores in JKS and PKCS12 formats.

> **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 **Selectidentity**, 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.

   > **Note:** If you enter the wrong value for the alias or the password, the tool will not be able to open the identity.



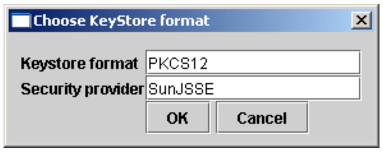
Select file C:\myfolder\tmy_certificate.jks

Alias demo_john

Password **********

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.

   KekyStore Format Dialog

   Keystore format PKCS12

   Security provider SunJSSE

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.

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.

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.

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.

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.

Chapter 4: Administrator's Guide

The HPE SOA Registry Foundation Administrator's Guide contains information necessary for the management of HPE SOA Registry Foundation. It is aimed at the user responsible for configuring the registry and managing permissions, and replication. This guide is divided into the following sections:

"Registry Management" on the next page . Registry management includes also management of user accounts and permissions and taxonomy management.

"Registry Configuration" on page 334 . How to configure the Registry Console.

"Registry Console Configuration" on page 345. This section covers setting the URLs, directories, contexts, timeouts and limits associated with the HPE SOA Registry Foundation interface.

"Permissions: Principles" on page 349. This section discusses the mechanism HPE SOA Registry Foundation provides for the management of users' rights; permissions allow the administrator to manage or make available different parts of the registry to different users.

"PStore Tool" on page 361. Describes a tool for management of protected stores for certificates and security identities.

> **Note:** Make sure HPE SOA Registry Foundation 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`.

Hostname and port are defined when HPE SOA Registry Foundation 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.
>
> Be very careful when editing the `Operational business entity`, or deleting of the taxonomy uddi-org:types. Modification of these structures can lead to registry instability.

Registry Management

Accessing Registry Management

Registry Management is a set of tasks that the administrator can address through the Registry Console. These tasks are listed in the “Registry Management” figure below.

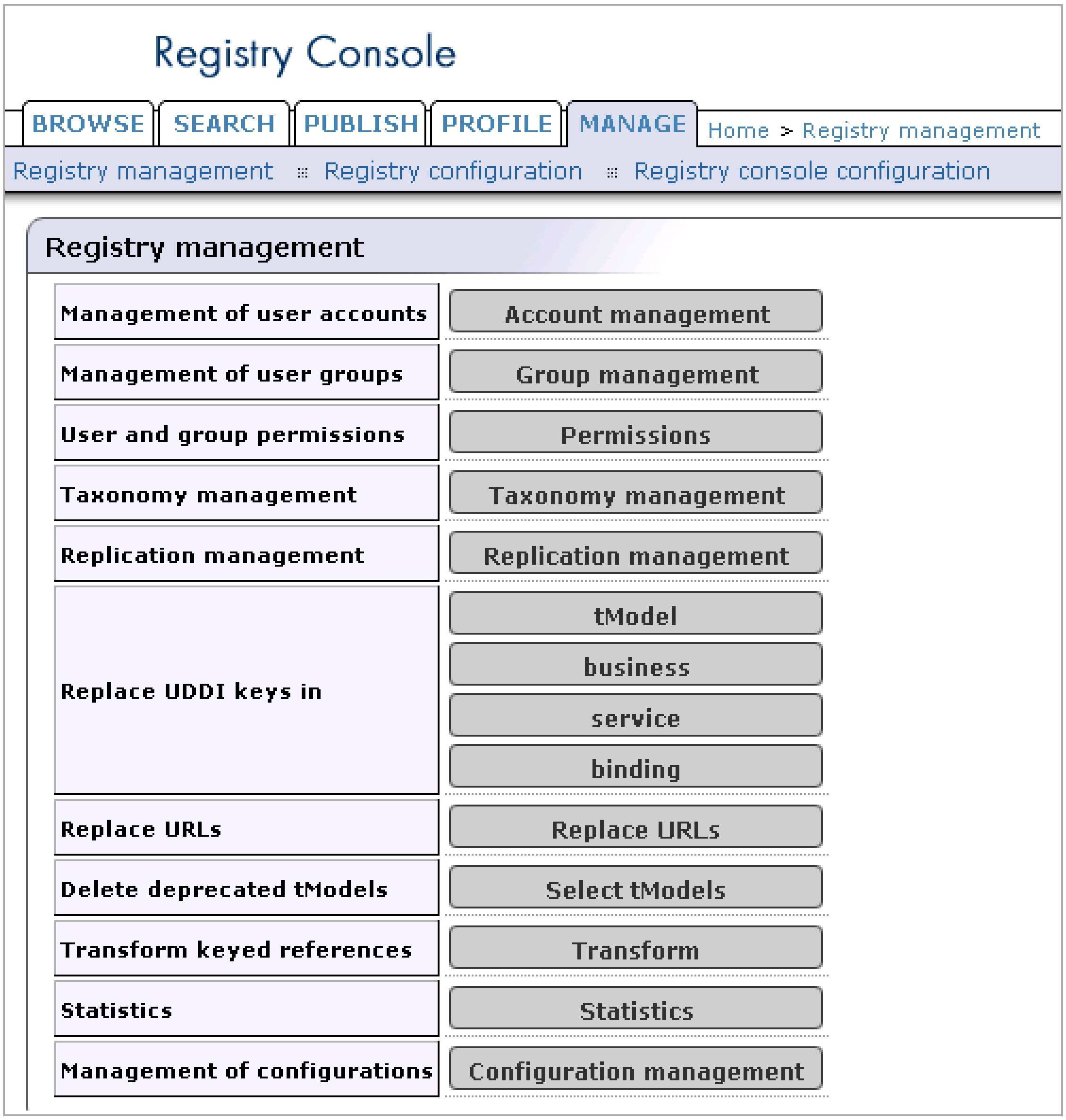
To access the Registry Management console:

1. Log on as administrator or as a user with privilege to display **Manage** tab as described in the Rules to Display the Manage Tab note below.
2. Click the **Manage** main menu tab.
3. Select the **Registry management** link under **Manage** tab. This returns the “Registry Management” screen shown below.

> **Note: 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,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.

Registry Management



Registry management

Management of user accounts	Account management
Management of user groups	Group management
User and group permissions	Permissions
Taxonomy management	Taxonomy management
Replication management	Replication management
Replace UDDI keys in	tModel business service binding
Replace URLs	Replace URLs
Delete deprecated tModels	Select tModels
Transform keyed references	Transform
Statistics	Statistics
Management of configurations	Configuration 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 - Upload, download and removing 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, see "Replication Management" on page 316.)
- 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 "Node" on page 341for instructions on 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, "Taxonomy: Principles, Creation and Validation" on page 224 in the User's Guide.
- 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.

Account Management

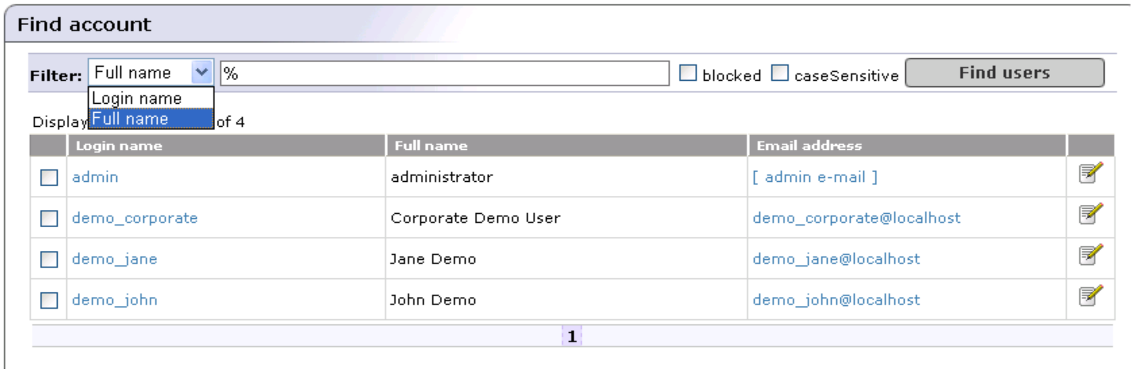
The HPE SOA Registry Foundation administrator manages user accounts using the Registry Console. Use this console whenever you want to disable an account, change limits for a particular account, or take care of general housekeeping.

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 “Find Account”.You can search accounts using the Find users button.

   Find Account



Find account

Filter: Full name | Login name / Full name | % | □ blocked □ caseSensitive | Find users

Display of 4

	Login name	Full name	Email address	
□	admin	administrator	[admin e-mail]	
□	demo_corporate	Corporate Demo User	demo_corporate@localhost	
□	demo_jane	Jane Demo	demo_jane@localhost	
□	demo_john	John Demo	demo_john@localhost	

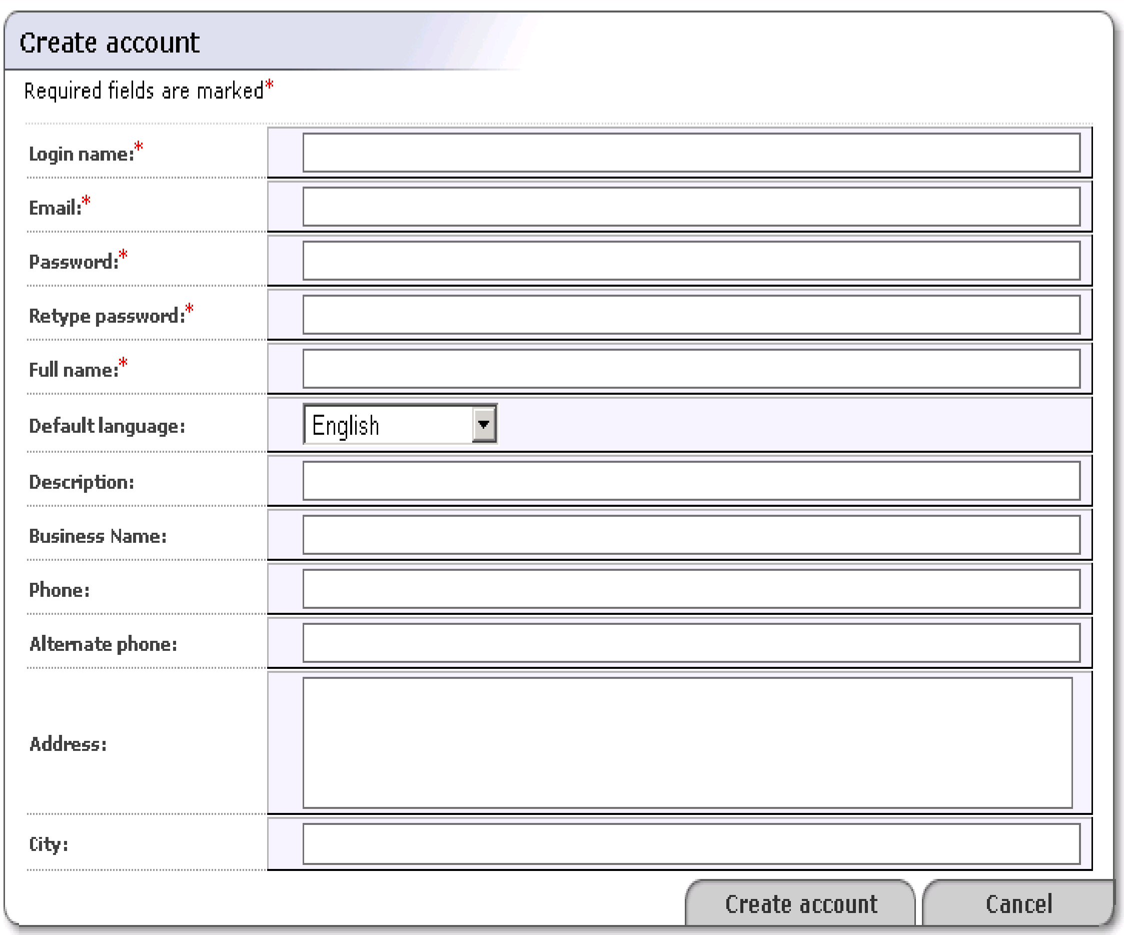
1

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 “Create Account”.

   Create Account



Create account

Required fields are marked*

Login name:*

Email:*

Password:*

Retype password:*

Full name:*

Default language: English

Description:

Business Name:

Phone:

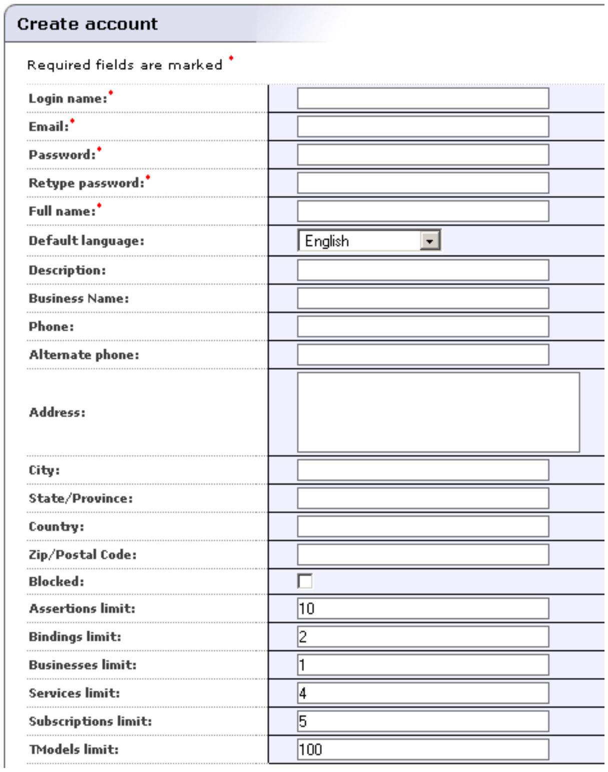
Alternate phone:

Address:

City:

2. Provide the information shown in . Fields marked with a red asterisk (*) are required.

   New Account - All Fields



Create account

Required fields are marked *

Login name:*	
Email:*	
Password:*	
Retype password:*	
Full name:*	
Default language:	English
Description:	
Business Name:	
Phone:	
Alternate phone:	
Address:	
City:	
State/Province:	
Country:	
Zip/Postal Code:	
Blocked:	□
Assertions limit:	10
Bindings limit:	2
Businesses limit:	1
Services limit:	4
Subscriptions limit:	5
TModels limit:	100

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** - deprecated and unused now.

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 "Permissions: Principles" on page 349. By default, only the administrator has these permissions, and therefore the administrator has an unlimited account.

Edit Account

To edit an account:

1. On the **Find account** page shown in the “Find Account” figure, 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 the “New Account - All Fields” figure.

   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.

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 HPE SOA Registry Foundation, 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.

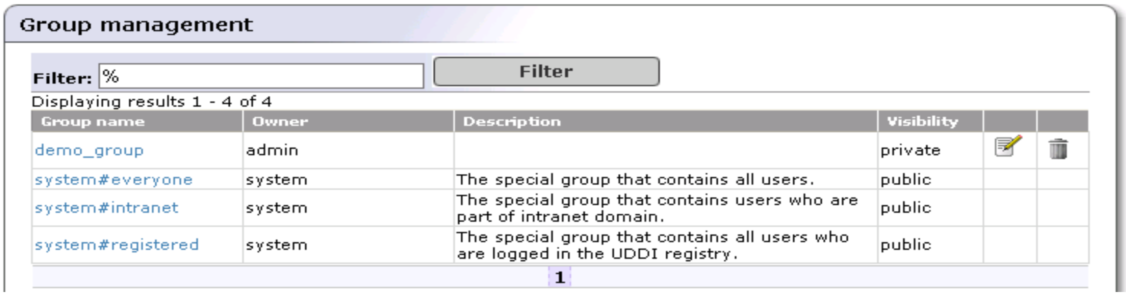
Group Management

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

View User Groups



Group management

Filter: %

Displaying results 1 - 4 of 4

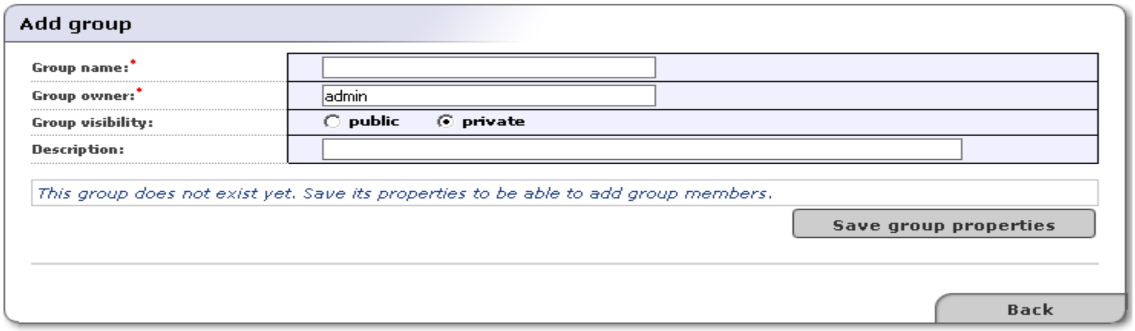
Group name	Owner	Description	Visibility		
demo_group	admin		private	✎	🗑
system#everyone	system	The special group that contains all users.	public		
system#intranet	system	The special group that contains users who are part of intranet domain.	public		
system#registered	system	The special group that contains all users who are logged in the UDDI registry.	public		

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 the “View User Groups” figure above.
3. Click the **Add Group** button. This returns a blank Add group page much like the one shown in the “Add Group Page” figure below.

   Add Group Page



Group name:*	
Group owner:*	admin
Group visibility:	○ public ● private
Description:	

This group does not exist yet. Save its properties to be able to add group members.

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.

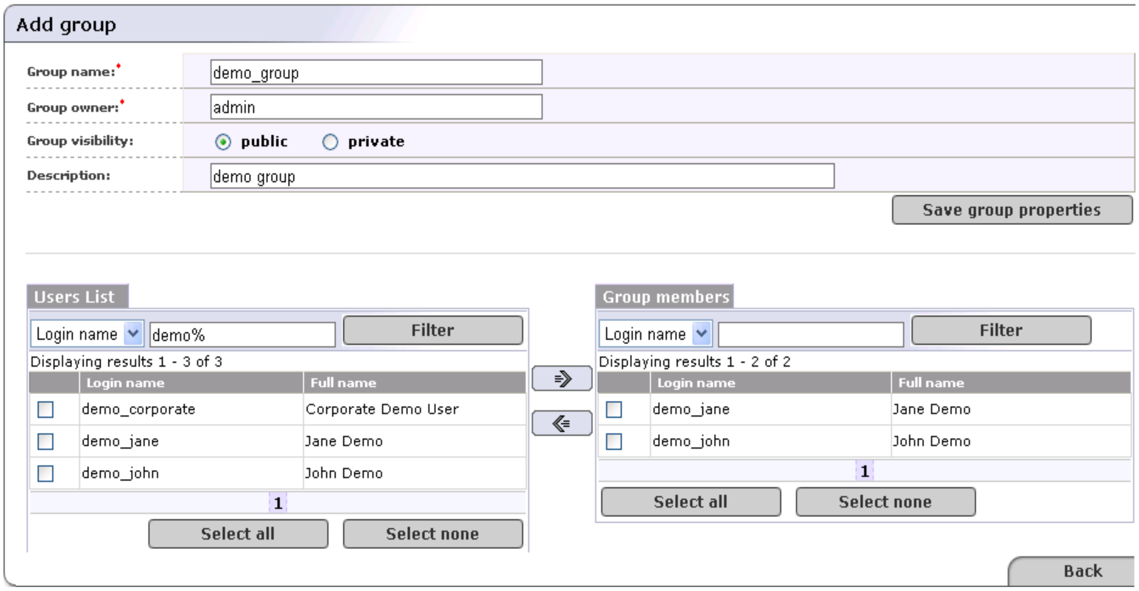
5. Use the radio buttons labeled **public** and **private** to set group visibility.

   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 the “View User Groups” figure.

   Edit Group Membership



Group name:*	demo_group
Group owner:*	admin
Group visibility:	● public ○ private
Description:	demo group

Users List: Login name demo%
Displaying results 1 - 3 of 3

	Login name	Full name
□	demo_corporate	Corporate Demo User
□	demo_jane	Jane Demo
□	demo_john	John Demo

1

Group members: Login name
Displaying results 1 - 2 of 2

	Login name	Full name
□	demo_jane	Jane Demo
□	demo_john	John Demo

1

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. 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.

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 the “View User Groups” figure above.
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 the “Edit Group Membership” figure above.

Permissions

This chapter describes how you can set permissions using the Registry Console. Before you start to work with permissions, we recommend reading "Permissions: Principles" on page 349 to become familiar with permissions principles.

HPE SOA Registry Foundation 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.

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 "Permissions Definitions" on page 349.
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).

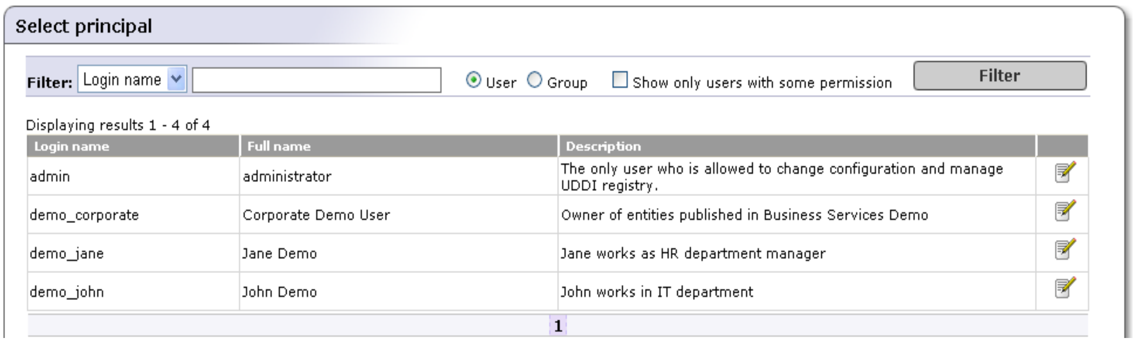
   > **Note:** Use the drop down list in this section, labeled Filter: to sort users by ***Login name*** or ***Full name***.
   >
   > Use the text box to further filter users by name. You can use % as wildcard in this field.

   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 the following “Select Principal” page.

   Select Principal



Select principal

Filter: Login name ▾ ● User ○ Group □ Show only users with some permission [Filter]

Displaying results 1 - 4 of 4

Login name	Full name	Description	
admin	administrator	The only user who is allowed to change configuration and manage UDDI registry.	✎
demo_corporate	Corporate Demo User	Owner of entities published in Business Services Demo	✎
demo_jane	Jane Demo	Jane works as HR department manager	✎
demo_john	John Demo	John works in IT department	✎

1

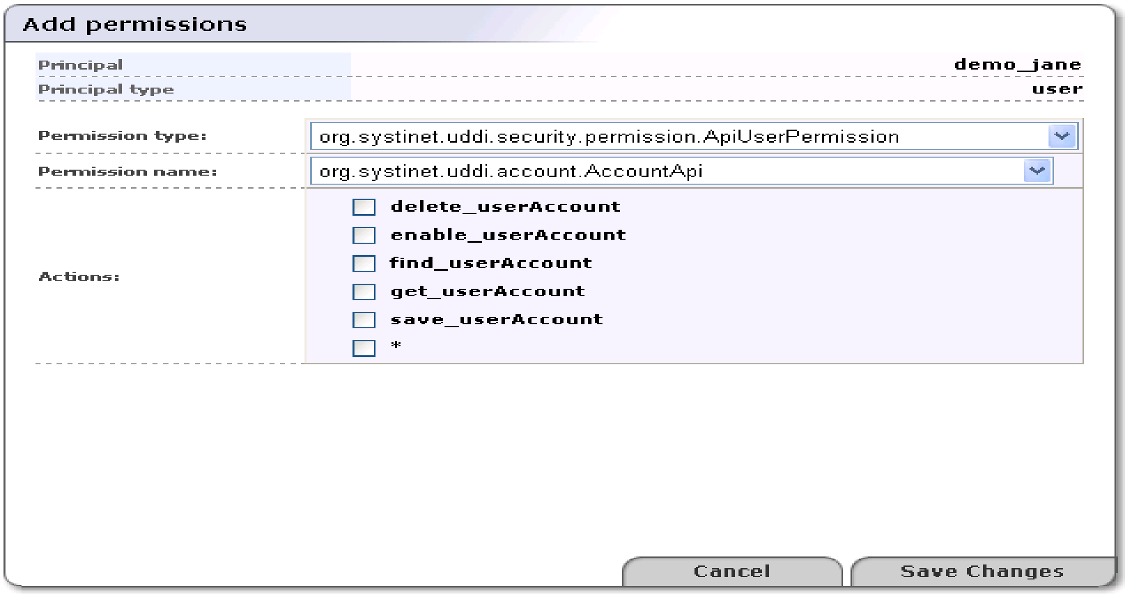
4. Click the **Edit** icon (✎) associated with the user or group whose permissions you wish to set.

Add Permission

To add permissions:

1. Access permission management as described above in “Accessing Permission Management”.
2. On the principal list page shown in the “Select Principal” figure above, 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 the following figure will appear.

   Add Permission



Add permissions

Principal	demo_jane
Principal type	user
Permission type:	org.systinet.uddi.security.permission.ApiUserPermission
Permission name:	org.systinet.uddi.account.AccountApi
Actions:	□ delete_userAccount □ enable_userAccount □ find_userAccount □ get_userAccount □ save_userAccount □ *

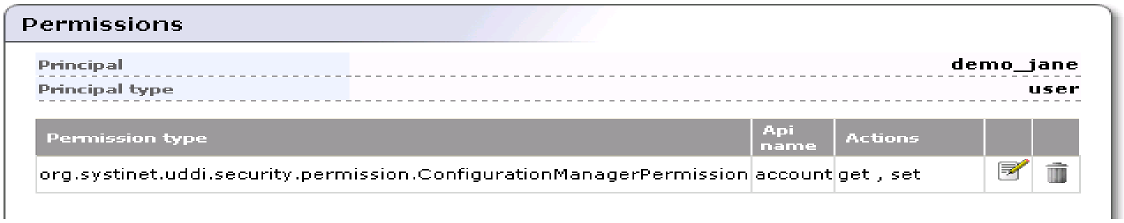
4. To add permissions:
   a. Select the type of permission from the drop down list labeled **Permission type**.
   b. From the drop down list labeled **Permission name**, select the name of the permission to add.
   c. Check the box(es) next to the actions associated with the permission name in order 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.

Editing and Deleting Permissions

To edit a permission:

1. On the principal list page shown in the “Select Principal” figure above, 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 the following figure will appear.

   Permissions List

   Permissions

Principal	demo_jane
Principal type	user

Permission type	Api name	Actions		
org.systinet.uddi.security.permission.ConfigurationManagerPermission	account	get , set	✎	🗑

3. Click the **Edit** or **Delete** icon (🗑) associated with the permission you want to address.

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 names** and all **actions** using the asterisk (*).

Taxonomy Management

This chapter describes how administrators can build and maintain taxonomies using the Registry Console. Before you start to manage taxonomies, it is recommended that you read "Taxonomy: Principles, Creation and Validation" on page 224 in the User's Guide, to become familiar with taxonomy principles.

The following tasks are described in this chapter:

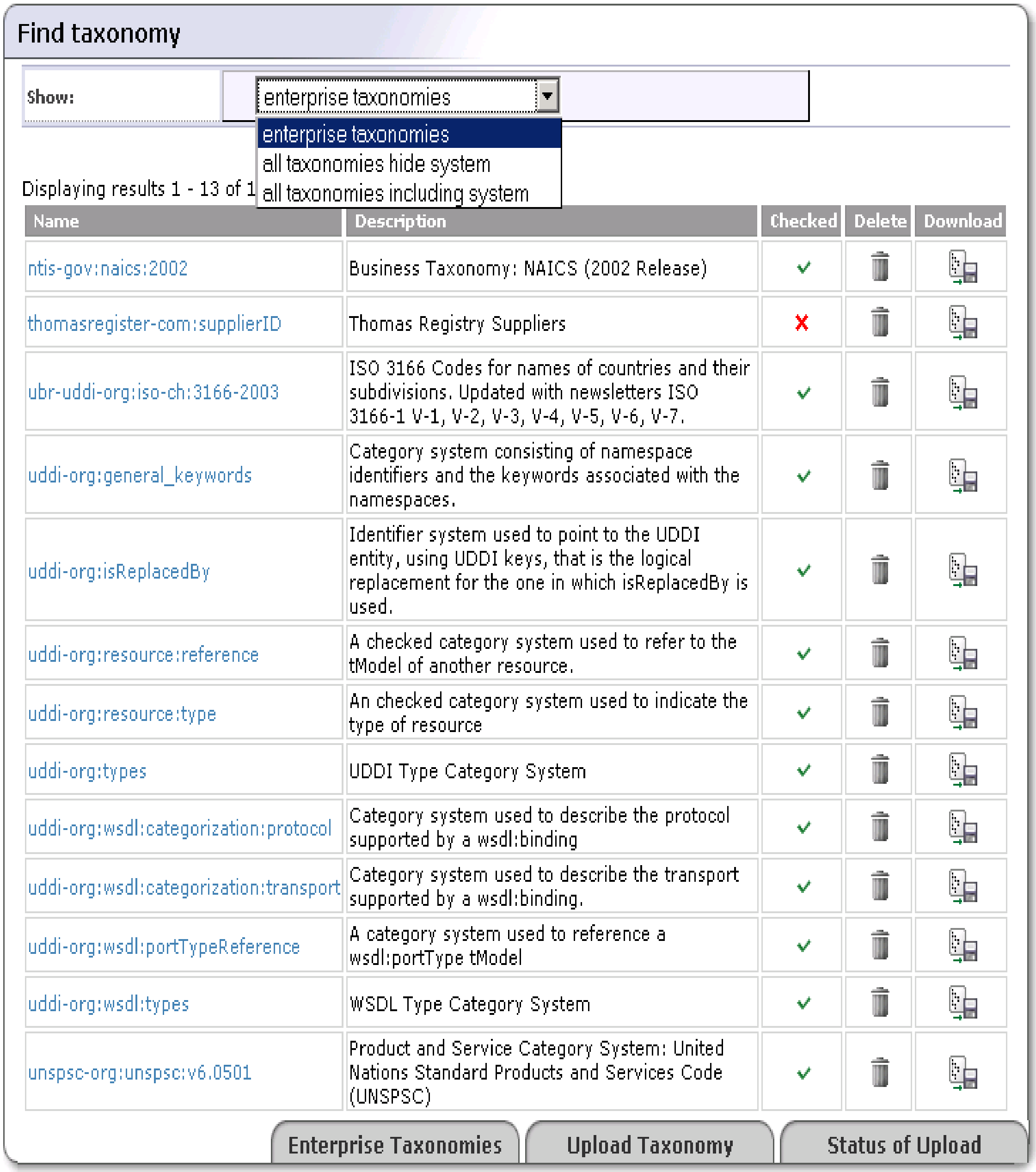
- Finding taxonomies - How to locate taxonomies in HPE SOA Registry Foundation.
- Uploading a taxonomy
- Downloading a taxonomy

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 the following figure.

Find Taxonomy (Enterprise Taxonomies)



Find taxonomy

Show: enterprise taxonomies

Displaying results 1 - 13 of 1

Name	Description	Checked	Delete	Download
ntis-gov:naics:2002	Business Taxonomy: NAICS (2002 Release)	✓	🗑	📥
thomasregister-com:supplierID	Thomas Registry Suppliers	✗	🗑	📥
ubr-uddi-org:iso-ch:3166-2003	ISO 3166 Codes for names of countries and their subdivisions. Updated with newsletters ISO 3166-1 V-1, V-2, V-3, V-4, V-5, V-6, V-7.	✓	🗑	📥
uddi-org:general_keywords	Category system consisting of namespace identifiers and the keywords associated with the namespaces.	✓	🗑	📥
uddi-org:isReplacedBy	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.	✓	🗑	📥
uddi-org:resource:reference	A checked category system used to refer to the tModel of another resource.	✓	🗑	📥
uddi-org:resource:type	An checked category system used to indicate the type of resource	✓	🗑	📥
uddi-org:types	UDDI Type Category System	✓	🗑	📥
uddi-org:wsdl:categorization:protocol	Category system used to describe the protocol supported by a wsdl:binding	✓	🗑	📥
uddi-org:wsdl:categorization:transport	Category system used to describe the transport supported by a wsdl:binding.	✓	🗑	📥
uddi-org:wsdl:portTypeReference	A category system used to reference a wsdl:portType tModel	✓	🗑	📥
uddi-org:wsdl:types	WSDL Type Category System	✓	🗑	📥
unspsc-org:unspsc:v6.0501	Product and Service Category System: United Nations Standard Products and Services Code (UNSPSC)	✓	🗑	📥

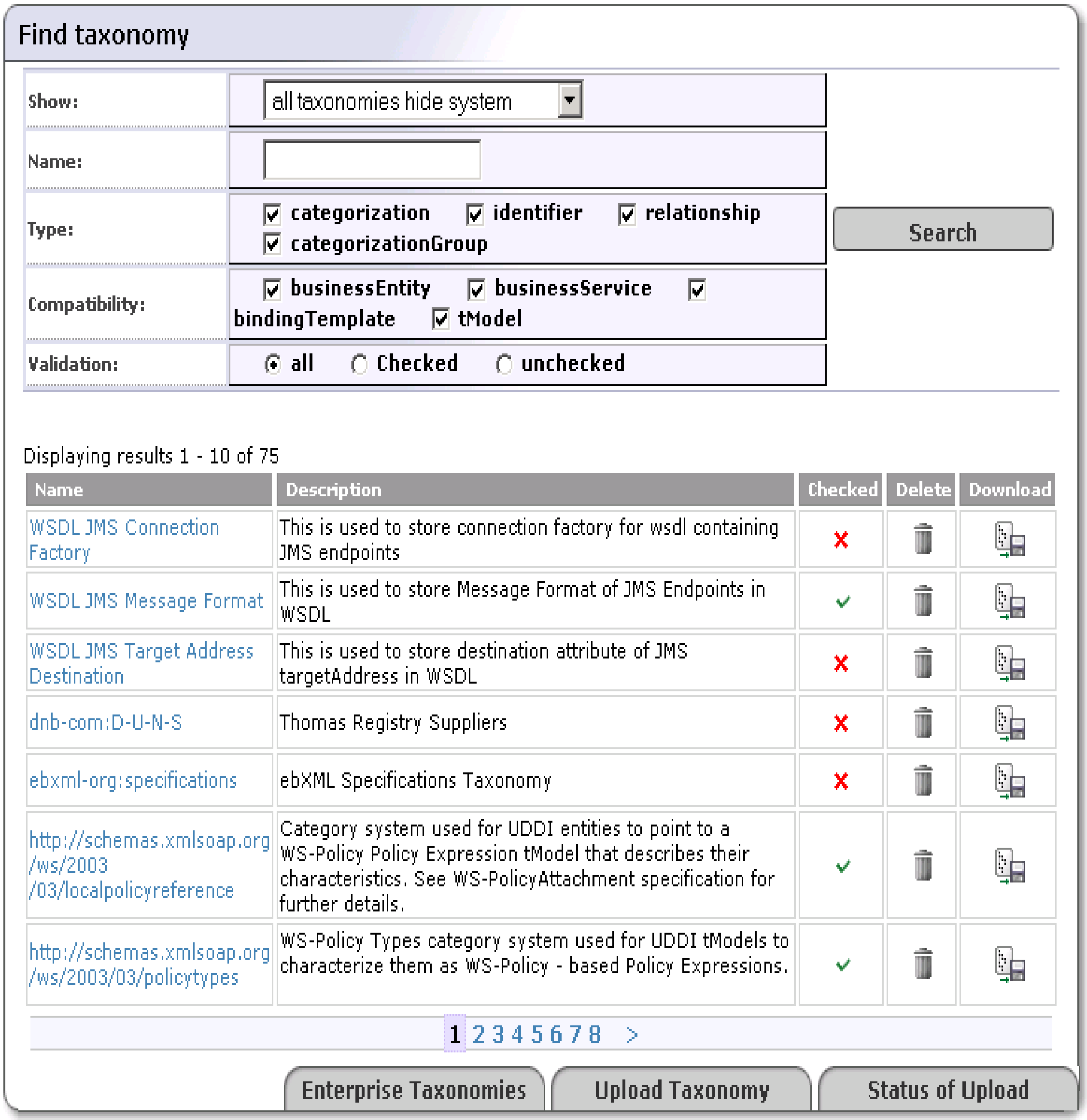
Use the above page, “Find Taxonomy (Enterprise Taxonomies),” to search enterprise taxonomies. You can classify taxonomies according to the following overlapping groups:

- **Enterprise taxonomies** - The HPE SOA Registry Foundation administrator can define which taxonomies will be present in the enterprise taxonomies list. The **Enterprise taxonomies** button located in the bottom part of the “Find Taxonomy (Enterprise Taxonomies)” figure 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 “Favorite Taxonomies” in "User Profile" on page 248 in the User's Guide, 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 following "Find Taxonomy" page will appear. You can search taxonomies using the following criteria: taxonomy name, type, compatibility, and validation.

Find Taxonomy



Find taxonomy

Show:	all taxonomies hide system ▾
Name:	
Type:	☑ categorization ☑ identifier ☑ relationship ☑ categorizationGroup
Compatibility:	☑ businessEntity ☑ businessService ☑ bindingTemplate ☑ tModel
Validation:	● all ○ Checked ○ unchecked

Displaying results 1 - 10 of 75

Name	Description	Checked	Delete	Download
WSDL JMS Connection Factory	This is used to store connection factory for wsdl containing JMS endpoints	✗	🗑	⭳
WSDL JMS Message Format	This is used to store Message Format of JMS Endpoints in WSDL	✓	🗑	⭳
WSDL JMS Target Address Destination	This is used to store destination attribute of JMS targetAddress in WSDL	✗	🗑	⭳
dnb-com:D-U-N-S	Thomas Registry Suppliers	✗	🗑	⭳
ebxml-org:specifications	ebXML Specifications Taxonomy	✗	🗑	⭳
http://schemas.xmlsoap.org/ws/2003/03/localpolicyreference	Category system used for UDDI entities to point to a WS-Policy Policy Expression tModel that describes their characteristics. See WS-PolicyAttachment specification for further details.	✓	🗑	⭳
http://schemas.xmlsoap.org/ws/2003/03/policytypes	WS-Policy Types category system used for UDDI tModels to characterize them as WS-Policy - based Policy Expressions.	✓	🗑	⭳

Finding Taxonomies

To locate a taxonomy in HPE SOA Registry Foundation:

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:

   a. Favorite taxonomies

   b. Enterprise taxonomies

   c. All taxonomies hide system

   d. All taxonomies including system

   This returns a list of taxonomies similar to that shown in the “Find Taxonomy (Enterprise Taxonomies)” figure above.

4. On the returned Find taxonomy page, you can further filter the results by

   a. name

   b. type - Types are discussed in “Taxonomy Types” in "Taxonomy: Principles, Creation and Validation" on page 224 in the User's Guide

   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.

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. A list of taxonomies like the one shown in the “Find Taxonomy” figure above will appear.
4. Click the **Upload taxonomy** button.
5. Choose a taxonomy file using the **Browse** button.
6. Click the **Upload taxonomy** button.

   > **Note:** The format of data on this page is described in the “Persistence Format” in "Taxonomy" on page 395 in the Developer's Guide.

> To upload multiple taxonomies at once you should add them into one ZIP archive and upload this archive.

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.

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.

> **Note:** Because this procedure is irreversible you will be asked to confirm your deletion.

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 entities from all subscriptions. These changes are then saved.

> **Note:** 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
```

> **Note:** Each registry must have a unique Operator Name. This value is set in the SMTP Configuration step during Registry installation. The Operator Names for the master registry and for any slave registries must not be the same.

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 HPE SOA Registry Foundation 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 “Publishing Subscriptions” in "Publishing" on page 265 in the User's Guide.

Slave Registry Setup

> **Note:** Only the administrator of the slave registry should do this.

There are two parts to the slave registry configuration:

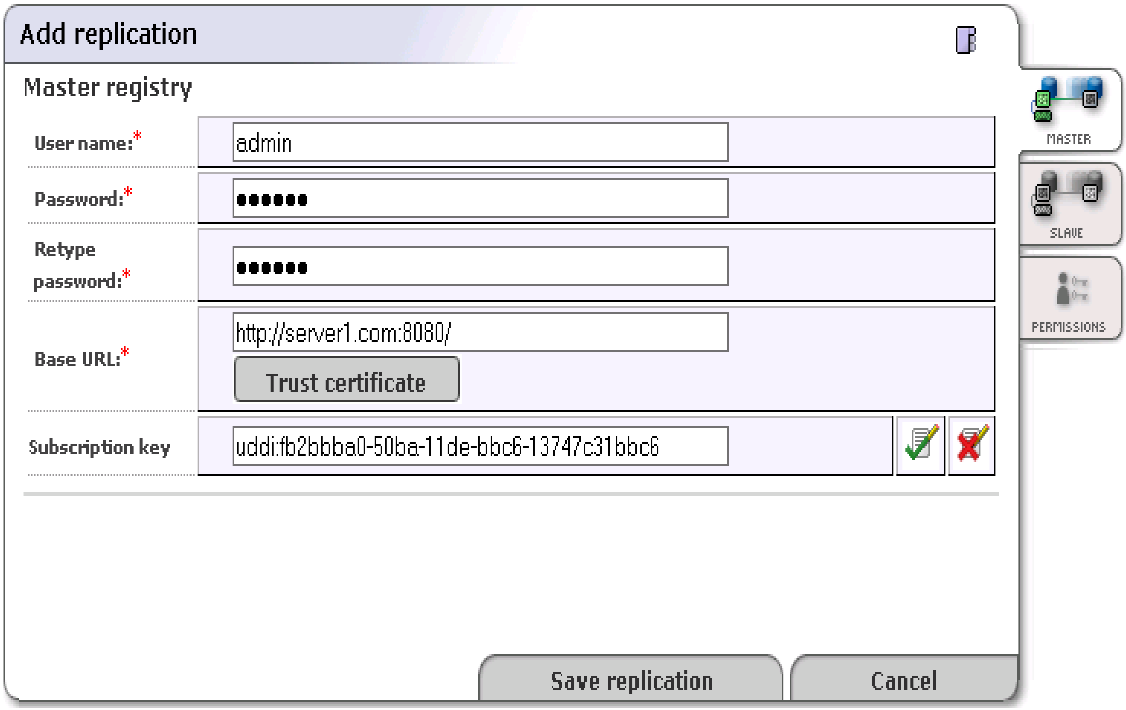
- Master registry information including the location of master registry endpoints for inquiry, subscription and security APIs, and the username/password pair on the master registry needed to obtain notifications;
- Slave registry information including the username/password pair on the slave registry 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 the “Add Replication Master” figure below.

6. Fill in the form under the **Slave** tab as described in the “Add Replication Slave” figure below..
7. Specify permissions for replicated data under the **Permissions** tab as shown in the “Add Replication Permissions” figure below.
8. Click **Save replication**.

Add Replication Master



Add replication

Master registry

User name:*	admin
Password:*	●●●●●●
Retype password:*	●●●●●●
Base URL:*	http://server1.com:8080/
Subscription key	uddi:fb2bbba0-50ba-11de-bbc6-13747c31bbc6

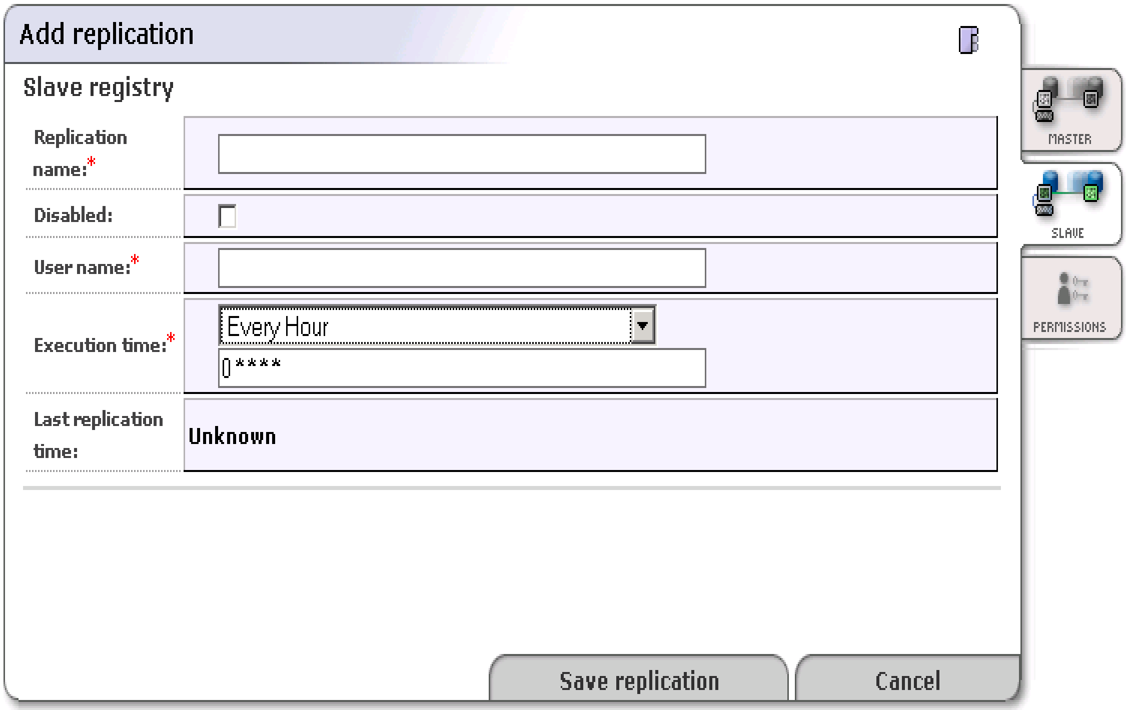
- **User name** - Name of the user who created the replication subscription on the master registry
- **Password** - Password of the user who created this subscription. This password is encrypted in the configuration file.
- **URLs of Master Registry** - All URLs (Inquiry URL, Subscription URL and Security 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.



Add replication

Slave registry

Replication name:*	
Disabled:	□
User name:*	
Execution time:*	Every Hour 0 * * * *
Last replication time:	Unknown

- ○ **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.

> **Note:** 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 "Generating Keys" on page 219 in the User's Guide. By default, the only user who can do

> this is the admin.

- **Execution time** - Specify the period between replications by selecting an option from the drop-down box or select "Other" and input a period pattern. The pattern follows the UNIX Crontab format. It consists of five fields that describe the time period in the following order:

  - Minute of the hour (0-59)
  - Hour of the day (0-24)
  - Day of the month (1-31)
  - Month of the year (Jan-Dec or 1-12)
  - Day of the week (Sun-Sat or 0-6, 7 is same as 0)

  Instead of a number, the patterns can contain a star character which means any value. They can also contain a star, a slash, and a number N which means that only N-th occurrence of such a time would be active. A list of comma separated numbers, months or days (without spaces) is also allowed. Examples:

  - 15 * * * *

    every hour on 15th minute
  - */10 * * * *

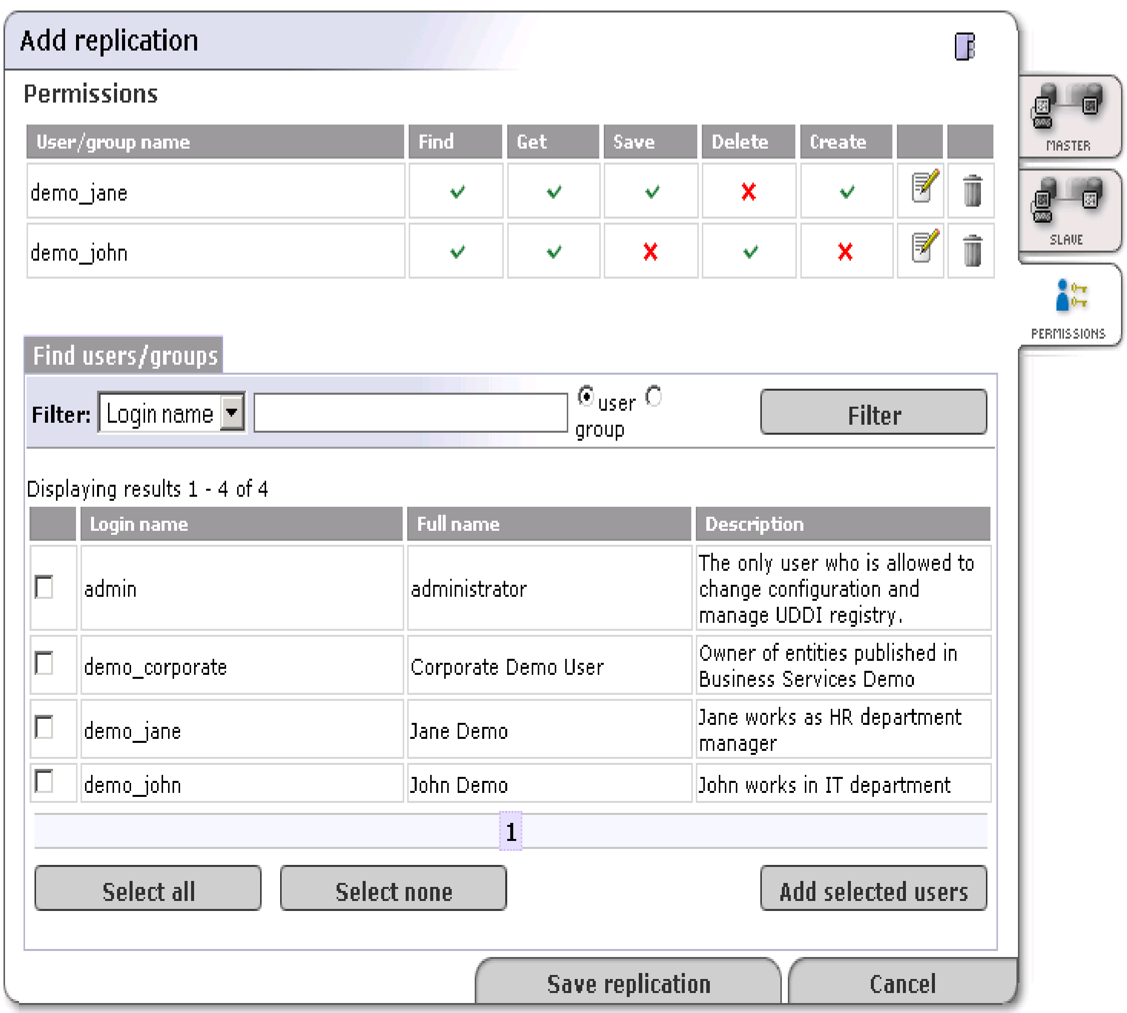
    every 10 minutes
  - 0 0 * * Sat-Sun

    every weekend at midnight
  - 0 18 1 *

    6:00 PM on the 1st day of any month
  - 0 */6 * * Fri

    every sixth hour on Friday
- **Last replication time** - The date and time when the last replication occurred.

Add Replication Permissions



Add replication

Permissions

User/group name	Find	Get	Save	Delete	Create		
demo_jane	✓	✓	✓	✗	✓		
demo_john	✓	✓	✗	✓	✗		

Find users/groups

Filter: Login name ● user ○ group

Displaying results 1 - 4 of 4

	Login name	Full name	Description
□	admin	administrator	The only user who is allowed to change configuration and manage UDDI registry.
□	demo_corporate	Corporate Demo User	Owner of entities published in Business Services Demo
□	demo_jane	Jane Demo	Jane works as HR department manager
□	demo_john	John Demo	John works in IT department

In the page shown in the above figure, 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.

> **Note:** Use the button **Replicate now** on the **replication** page to test the replication settings.

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**.

   > **Note:** The replace key operation can break digital signatures on changing entity as well as on other entities which reference to the changing entity.

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

Replacing businessEntity keys

When you replace a businessEntity key, the key will be updated in the following data structures:

- businessEntity
- services
- keyedReferences

Replacing businessService keys

When you replace a businessService key, the key will be updated in the following data structures:

- businessService
- bindingTemplates
- keyedReferences

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

Registry Statistics

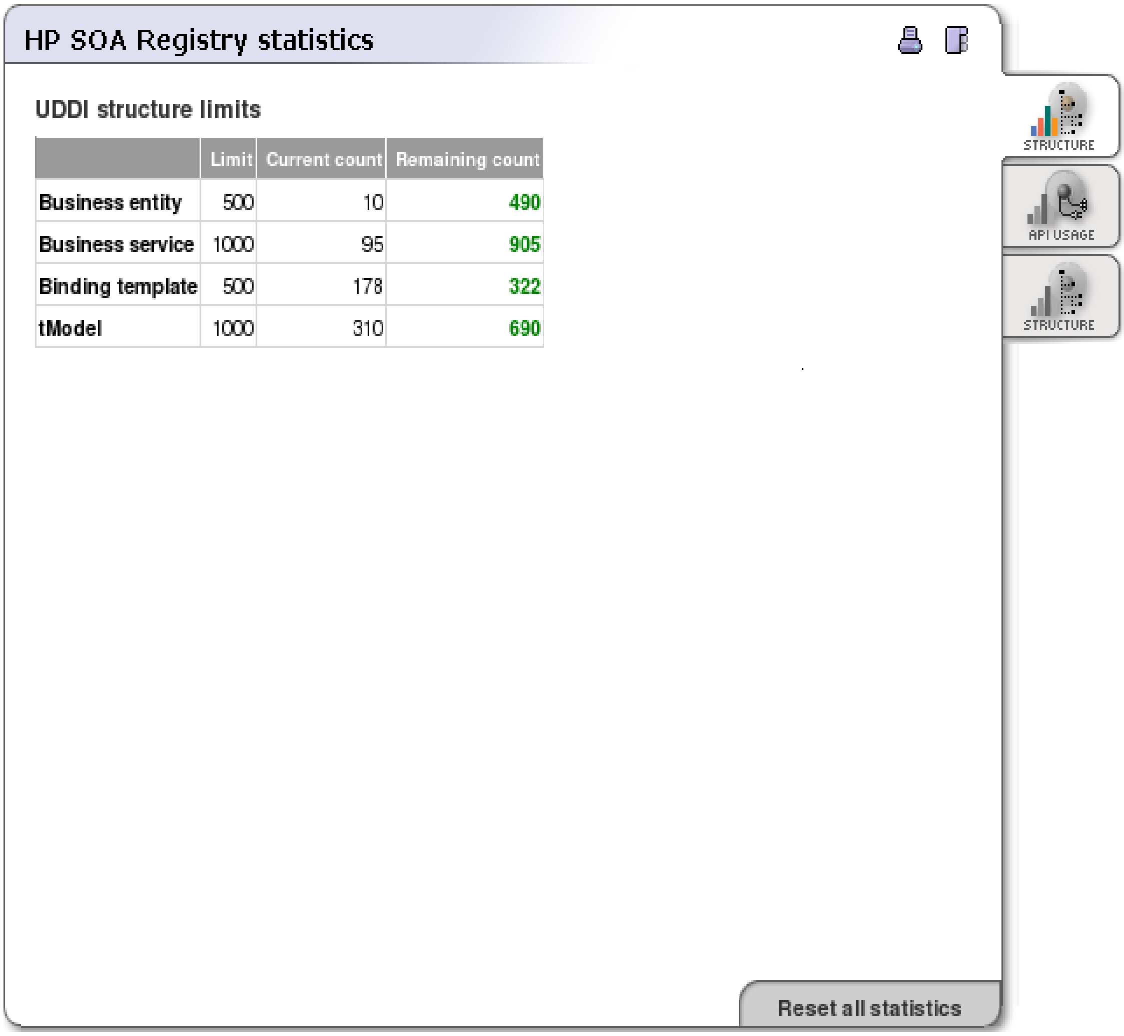
Registry statistics include statistics on::

- 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 to the figure below will appear, summarizing publishing limits imposed by the product license, current counts and the number remaining.

   Statistics - Publication Limits



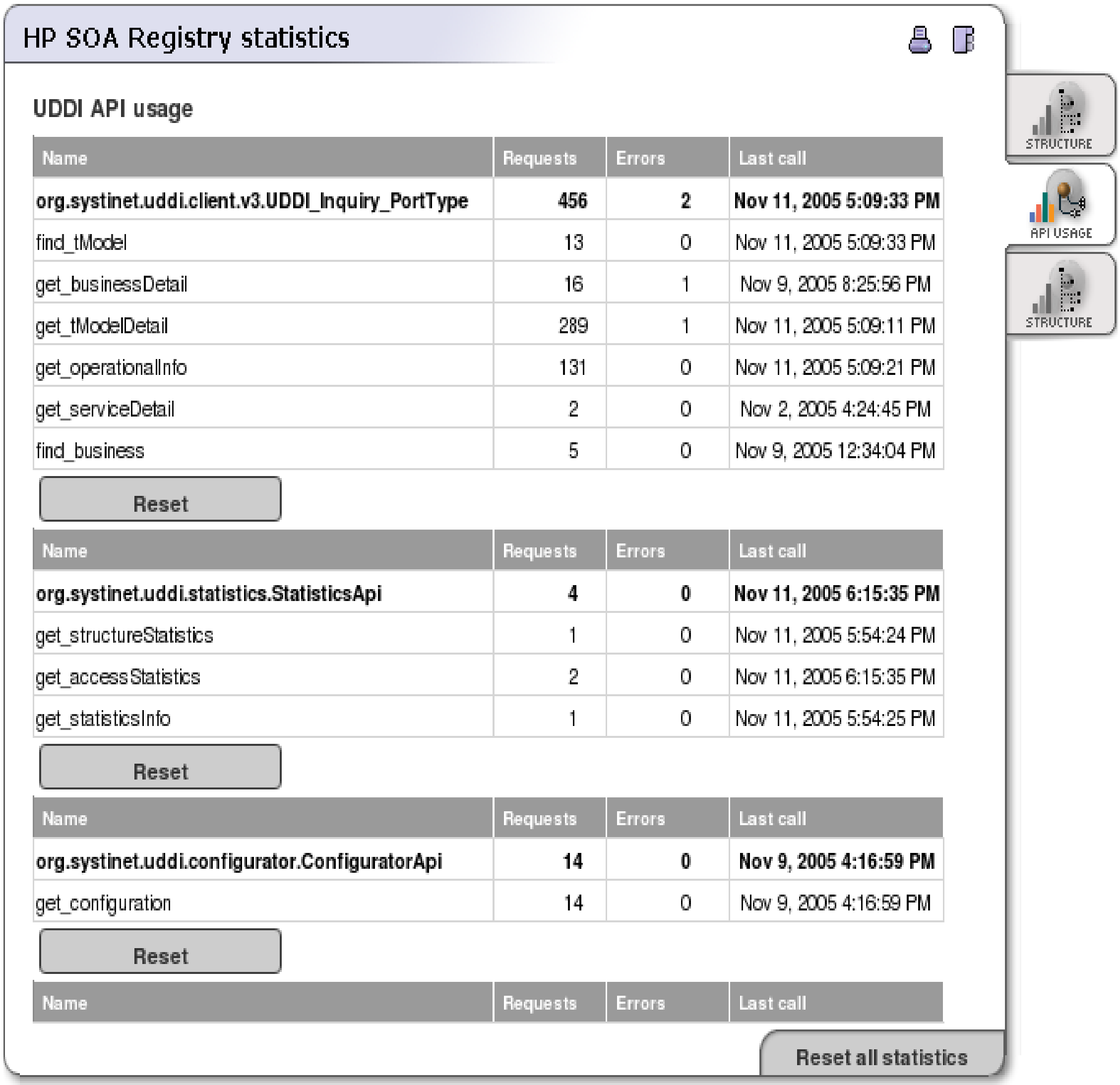
HP SOA Registry statistics

UDDI structure limits

	Limit	Current count	Remaining count
Business entity	500	10	490
Business service	1000	95	905
Binding template	500	178	322
tModel	1000	310	690

5. Click the **API Usage** tab and you will see a page similar to the following figure 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**.

   Statistics - API usage



HP SOA Registry statistics

UDDI 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

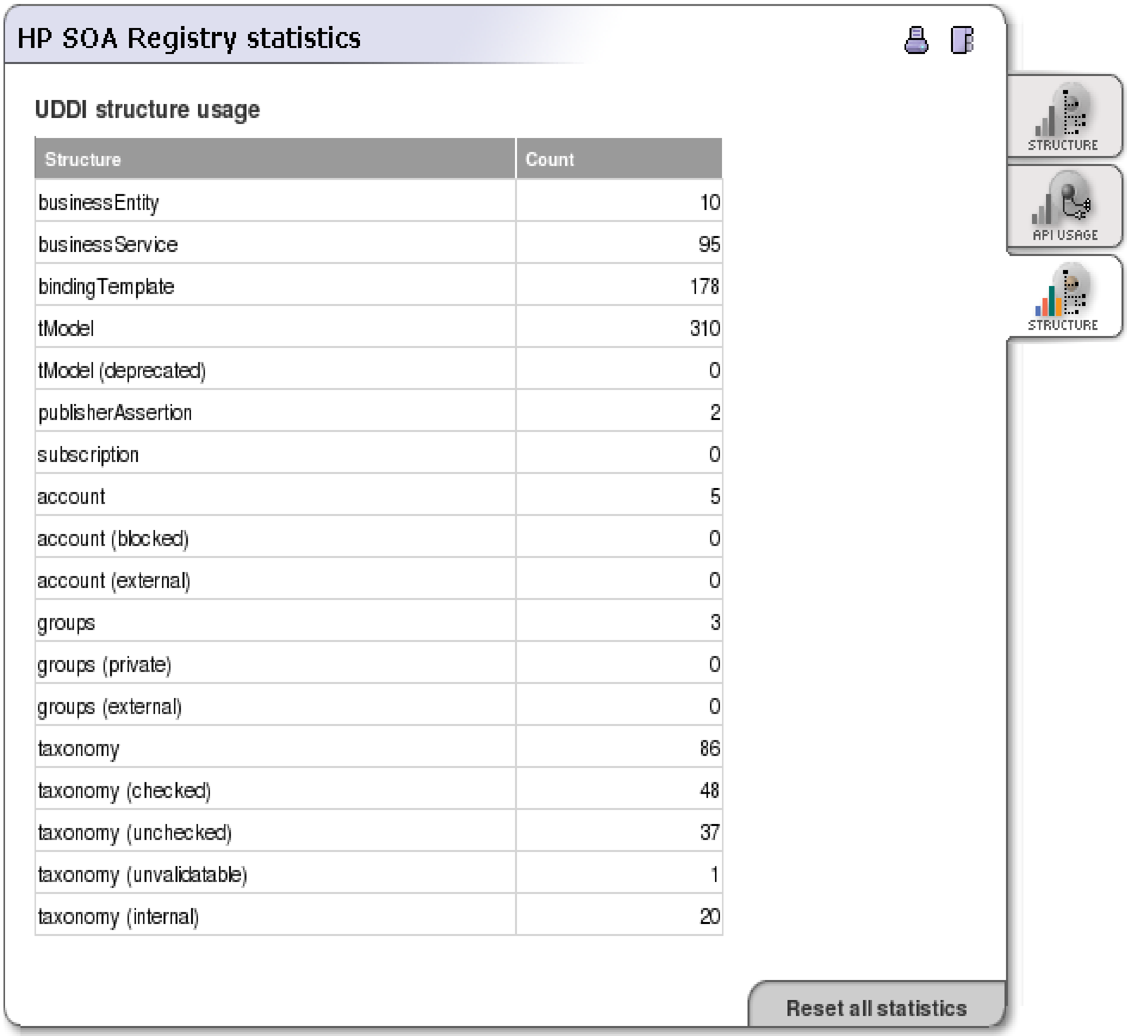
Name	Requests	Errors	Last call
org.systinet.uddi.statistics.StatisticsApi	**4**	**0**	**Nov 11, 2005 6:15:35 PM**
get_structureStatistics	1	0	Nov 11, 2005 5:54:24 PM
get_accessStatistics	2	0	Nov 11, 2005 6:15:35 PM
get_statisticsInfo	1	0	Nov 11, 2005 5:54:25 PM

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 to the figure below appears. On that page you can see number of UDDI entities stored in HPE SOA Registry Foundation.

   Statistics - Structure



UDDI structure usage

Structure	Count
businessEntity	10
businessService	95
bindingTemplate	178
tModel	310
tModel (deprecated)	0
publisherAssertion	2
subscription	0
account	5
account (blocked)	0
account (external)	0
groups	3
groups (private)	0
groups (external)	0
taxonomy	86
taxonomy (checked)	48
taxonomy (unchecked)	37
taxonomy (unvalidatable)	1
taxonomy (internal)	20

Management of Configuration - User Interface

Configuration Management User Interface is available on the Registry Console, "Manage" tab, "Registry management" sub-tab (default), Configuration management button.

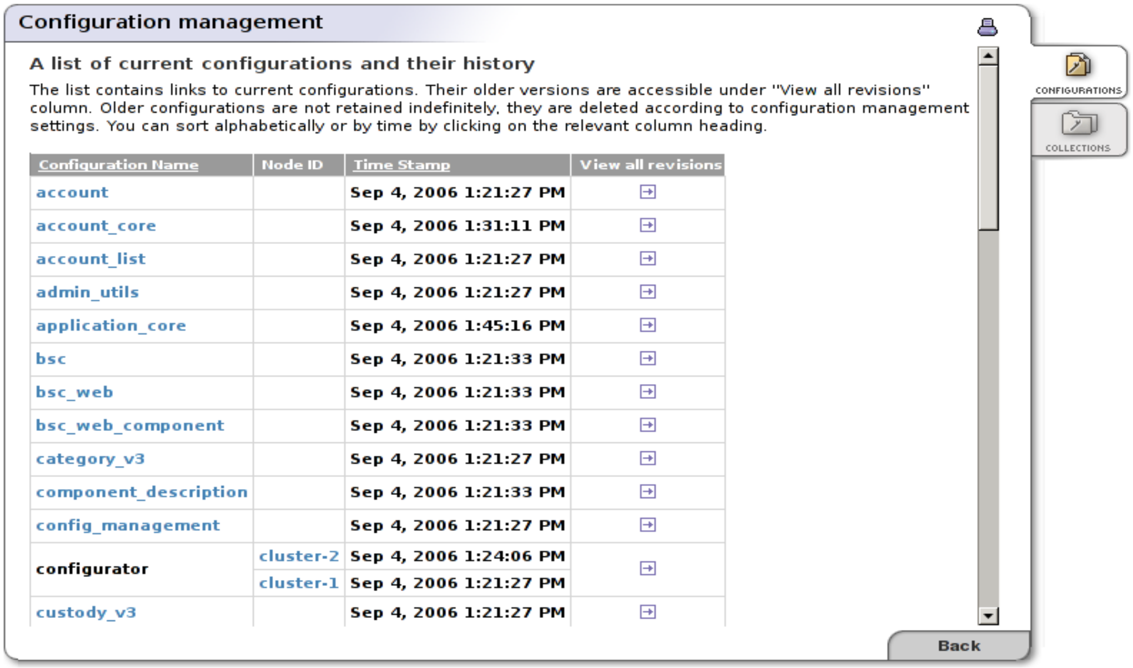
This management tool has two main parts designed for the following tasks:

1. Inspection of current configurations and their history.

2. Saving configuration states into collections to compare or restore them later.

Current Configurations and Their History

View of current configurations



Configuration management

A list of current configurations and their history

The list contains links to current configurations. Their older versions are accessible under "View all revisions" column. Older configurations are not retained indefinitely, they are deleted according to configuration management settings. You can sort alphabetically or by time by clicking on the relevant column heading.

Configuration Name	Node ID	Time Stamp	View all revisions
account		Sep 4, 2006 1:21:27 PM	⇥
account_core		Sep 4, 2006 1:31:11 PM	⇥
account_list		Sep 4, 2006 1:21:27 PM	⇥
admin_utils		Sep 4, 2006 1:21:27 PM	⇥
application_core		Sep 4, 2006 1:45:16 PM	⇥
bsc		Sep 4, 2006 1:21:33 PM	⇥
bsc_web		Sep 4, 2006 1:21:33 PM	⇥
bsc_web_component		Sep 4, 2006 1:21:33 PM	⇥
category_v3		Sep 4, 2006 1:21:27 PM	⇥
component_description		Sep 4, 2006 1:21:33 PM	⇥
config_management		Sep 4, 2006 1:21:27 PM	⇥
configurator	cluster-2 cluster-1	Sep 4, 2006 1:24:06 PM Sep 4, 2006 1:21:27 PM	⇥
custody_v3		Sep 4, 2006 1:21:27 PM	⇥

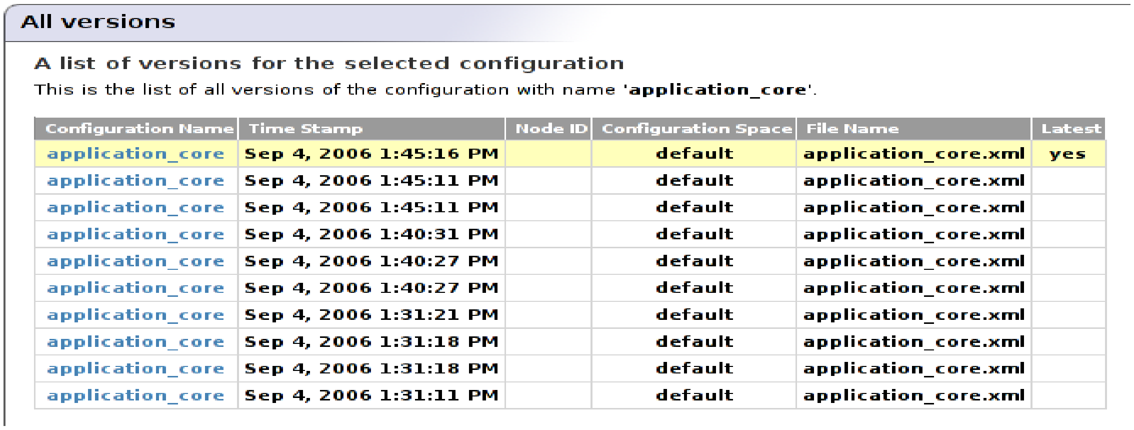
This view shows current configurations. You can either sort it alphabetically or by time by clicking on the relevant column heading. Configurations that are local to a cluster node are displayed for all nodes. You can switch to the named collections view with the left tab.

Two actions are available:

1. View the current configuration by clicking on the configuration name or in the case of cluster-local configurations on its Node ID.
2. View all versions of some configuration by clicking on the icon in the last column

When the list is sorted by time the configurations with the same name but different Node IDs are not grouped together.

View of all versions



All versions

A list of versions for the selected configuration

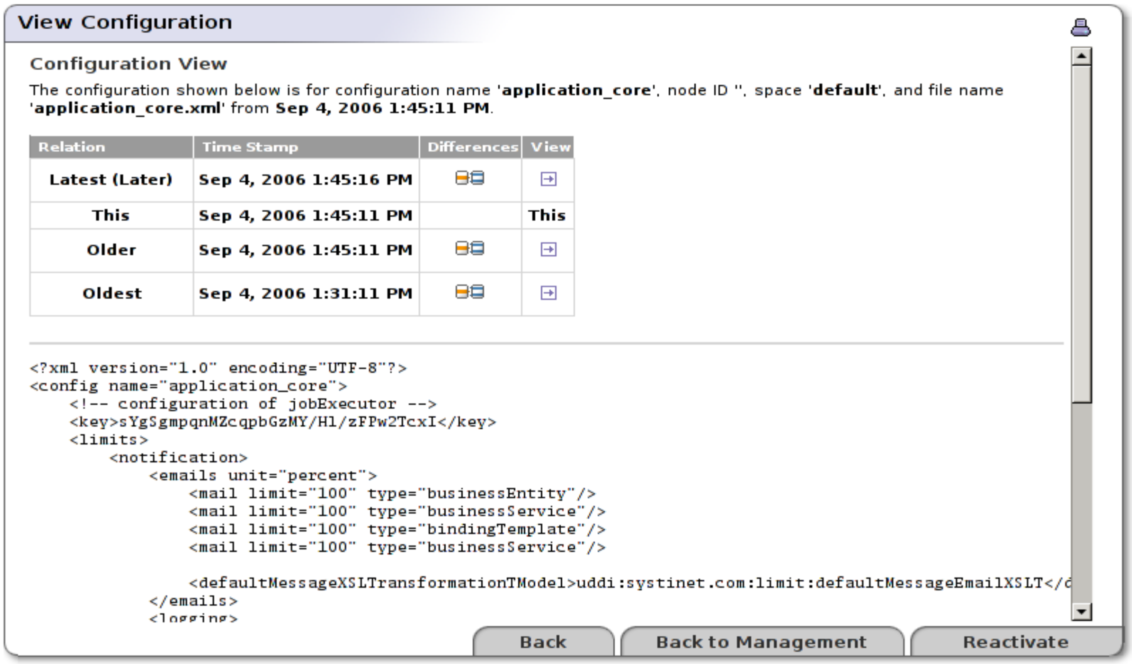
This is the list of all versions of the configuration with name '**application_core**'.

Configuration Name	Time Stamp	Node ID	Configuration Space	File Name	Latest
application_core	Sep 4, 2006 1:45:16 PM		default	application_core.xml	yes
application_core	Sep 4, 2006 1:45:11 PM		default	application_core.xml	
application_core	Sep 4, 2006 1:45:11 PM		default	application_core.xml	
application_core	Sep 4, 2006 1:40:31 PM		default	application_core.xml	
application_core	Sep 4, 2006 1:40:27 PM		default	application_core.xml	
application_core	Sep 4, 2006 1:40:27 PM		default	application_core.xml	
application_core	Sep 4, 2006 1:31:21 PM		default	application_core.xml	
application_core	Sep 4, 2006 1:31:18 PM		default	application_core.xml	
application_core	Sep 4, 2006 1:31:18 PM		default	application_core.xml	
application_core	Sep 4, 2006 1:31:11 PM		default	application_core.xml	

This view shows all versions of a specified configuration. If such a configuration is local, multiple entries may be marked as latest, one for each node. Latest nodes are also highlighted. The Length of this list is limited by rules for retaining older configurations (see Configuration in database section).

Clicking on a configuration name will show the configuration which is described in the row.

Configuration view



View Configuration

Configuration View

The configuration shown below is for configuration name '**application_core**', node ID '', space '**default**', and file name '**application_core.xml**' from **Sep 4, 2006 1:45:11 PM**.

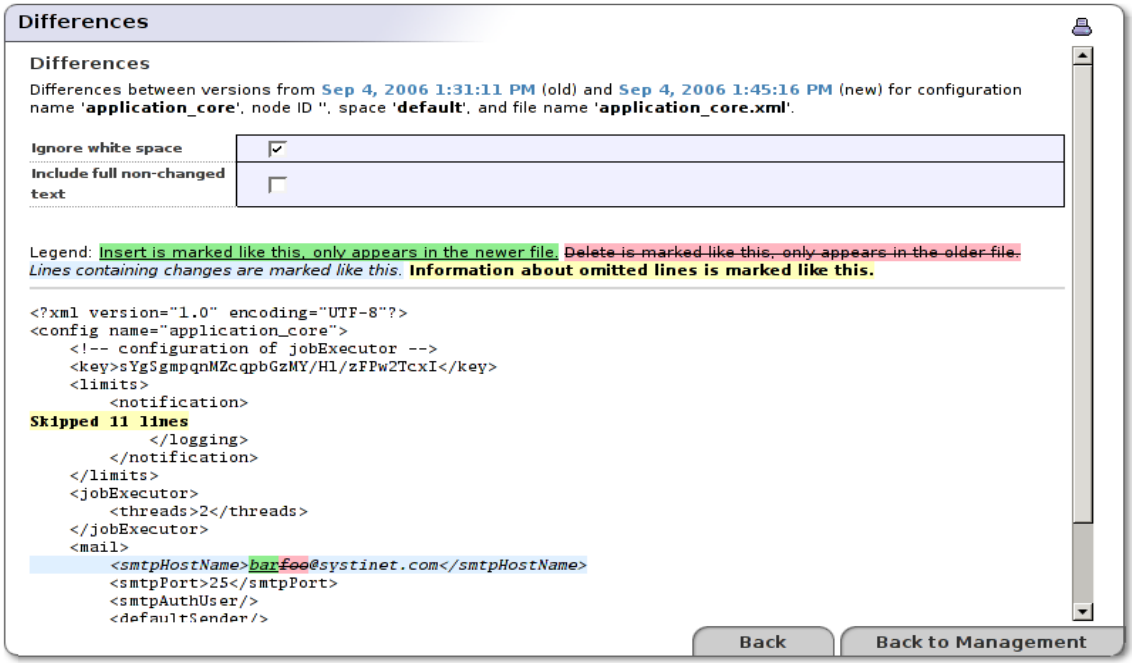
Relation	Time Stamp	Differences	View
Latest (Later)	Sep 4, 2006 1:45:16 PM	⧉	→
This	Sep 4, 2006 1:45:11 PM		This
Older	Sep 4, 2006 1:45:11 PM	⧉	→
Oldest	Sep 4, 2006 1:31:11 PM	⧉	→

```
<?xml version="1.0" encoding="UTF-8"?>
<config name="application_core">
    <!-- configuration of jobExecutor -->
    <key>sYgSgmpqnMZcqpbGzMY/H1/zFPw2TcxI</key>
    <limits>
        <notification>
            <emails unit="percent">
                <mail limit="100" type="businessEntity"/>
                <mail limit="100" type="businessService"/>
                <mail limit="100" type="bindingTemplate"/>
                <mail limit="100" type="businessService"/>

                <defaultMessageXSLTransformationTModel>uddi:systinet.com:limit:defaultMessageEmailXSLT</d
            </emails>
            <logging>
```

This view shows specified configuration information including the content. There are also links to related versions of the configurations (such as latest, later, older, or oldest). You can see these related configurations by clicking on the view icon or compare differences between the displayed version and a selected version by clicking the differences icon in the selected row.

If the displayed configuration is not the latest, a **Reactivate** button appears in the window. Its function is to make the displayed configuration active (after confirmation). It does so by adding it as a new configuration entry with the latest time stamp.

Differences



Differences

Differences between versions from Sep 4, 2006 1:31:11 PM (old) and Sep 4, 2006 1:45:16 PM (new) for configuration name '**application_core**', node ID '', space '**default**', and file name '**application_core.xml**'.

Ignore white space	☑
Include full non-changed text	☐

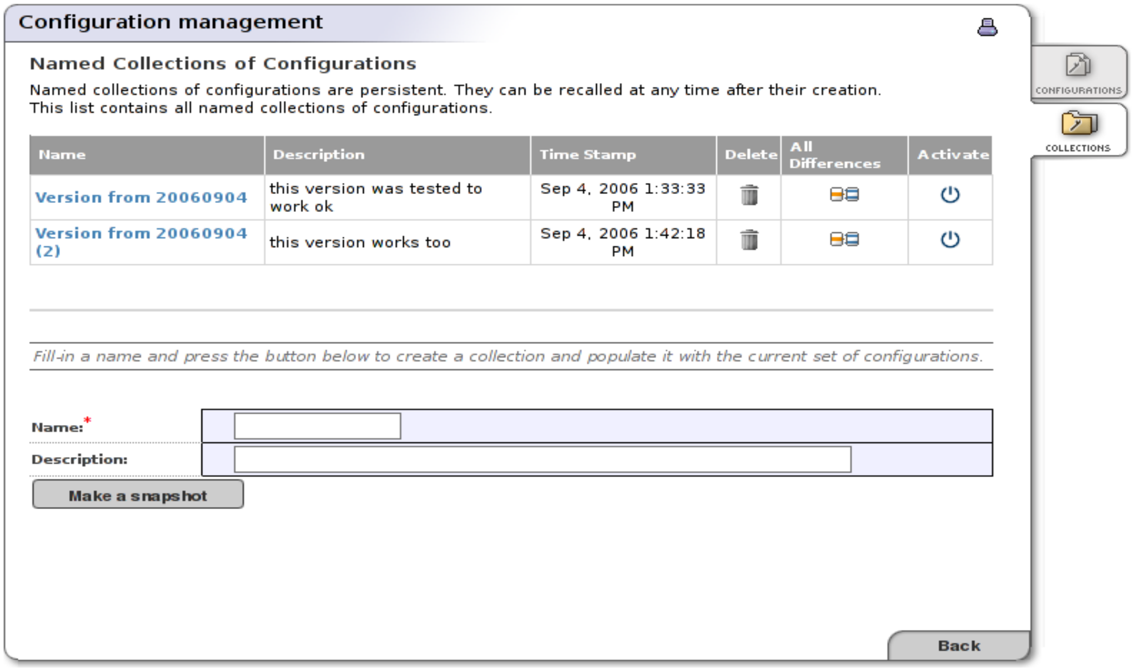
Legend: Insert is marked like this, only appears in the newer file. ~~Delete is marked like this, only appears in the older file.~~ *Lines containing changes are marked like this.* **Information about omitted lines is marked like this.**

```
<?xml version="1.0" encoding="UTF-8"?>
<config name="application_core">
    <!-- configuration of jobExecutor -->
    <key>sYgSgmpqnMZcqpbGzMY/H1/zFPw2TcxI</key>
    <limits>
        <notification>
Skipped 11 lines
            </logging>
        </notification>
    </limits>
    <jobExecutor>
        <threads>2</threads>
    </jobExecutor>
    <mail>
        <smtpHostName>barfoo@systinet.com</smtpHostName>
        <smtpPort>25</smtpPort>
        <smtpAuthUser/>
        <defaultSender/>
```

This view can be invoked from the configuration view. It shows a comparison between two versions of the configuration. You can alter the options for differences comparison: whether it is case sensitive and whether the full text is shown or omitted.

Named Collections of Configuration

List of named collections



Configuration management

Named Collections of Configurations

Named collections of configurations are persistent. They can be recalled at any time after their creation.
This list contains all named collections of configurations.

Name	Description	Time Stamp	Delete	All Differences	Activate
Version from 20060904	this version was tested to work ok	Sep 4, 2006 1:33:33 PM			
Version from 20060904 (2)	this version works too	Sep 4, 2006 1:42:18 PM			

Fill-in a name and press the button below to create a collection and populate it with the current set of configurations.

Name:*	
Description:	

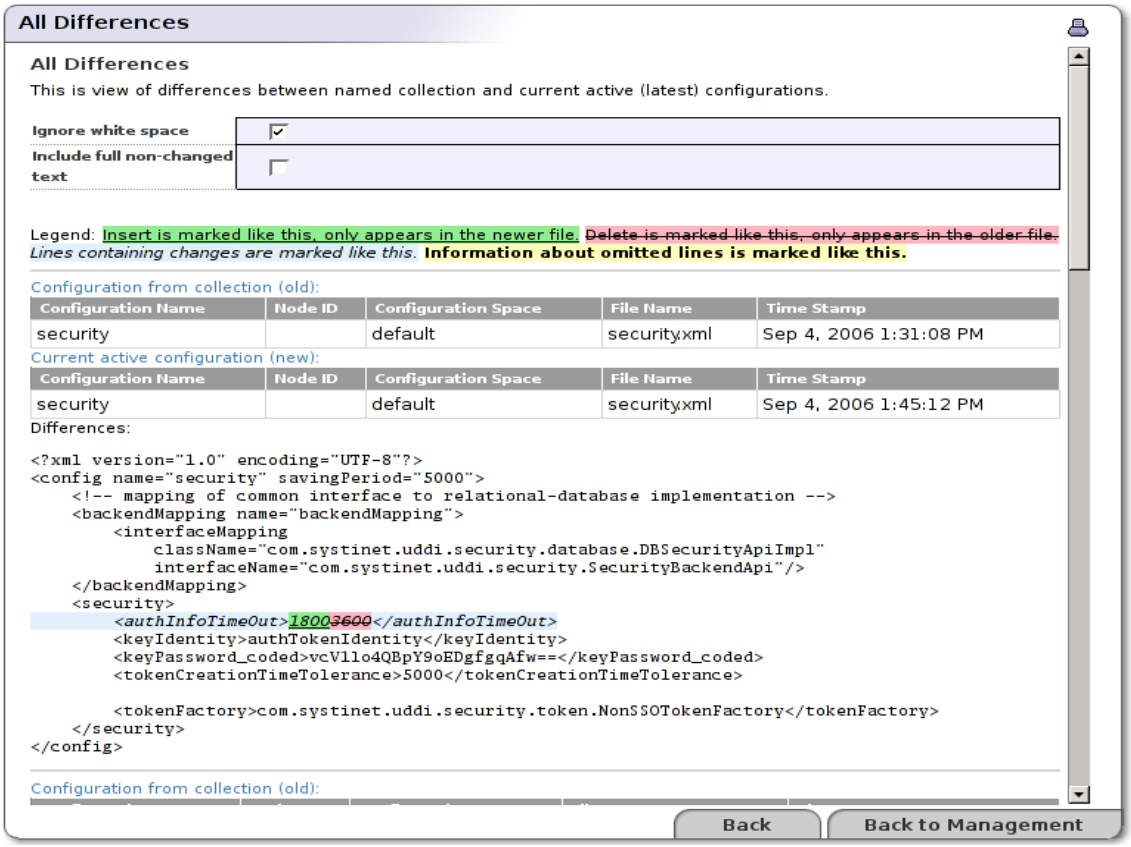
This view shows named collections of configurations which are stored in the database. It also allows you to capture the current state of configurations into such a collection so that you can later compare or restore them.

Creating new collections is easy. Just fill in the name and optionally the description of the collection and press the **Make a snapshot** button.

Once some collections are created, you can view their contents (by clicking the name) and compare them to the current set of configurations (by clicking the differences icon).

Activation of a collection means that all configurations that the collection holds will be added as new current configurations. Activation can be done on the collection as a whole (by clicking the icon in this view) or selectively on specified configurations (by button in the collection configuration view). Before activation proceeds a confirmation is required.

All Differences



All Differences

All Differences

This is view of differences between named collection and current active (latest) configurations.

Ignore white space	☑
Include full non-changed text	□

Legend: Insert is marked like this, only appears in the newer file. ~~Delete is marked like this, only appears in the older file.~~
Lines containing changes are marked like this. **Information about omitted lines is marked like this.**

Configuration from collection (old):

Configuration Name	Node ID	Configuration Space	File Name	Time Stamp
security		default	security.xml	Sep 4, 2006 1:31:08 PM

Current active configuration (new):

Configuration Name	Node ID	Configuration Space	File Name	Time Stamp
security		default	security.xml	Sep 4, 2006 1:45:12 PM

Differences:

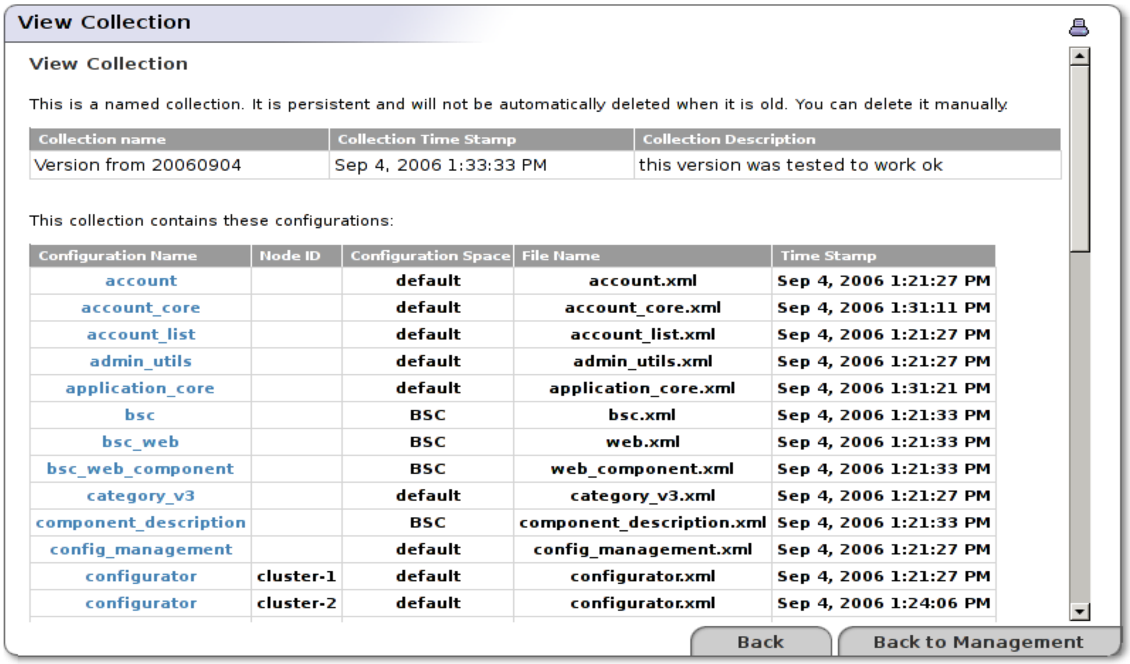
```
<?xml version="1.0" encoding="UTF-8"?>
<config name="security" savingPeriod="5000">
    <!-- mapping of common interface to relational-database implementation -->
    <backendMapping name="backendMapping">
        <interfaceMapping
            className="com.systinet.uddi.security.database.DBSecurityApiImpl"
            interfaceName="com.systinet.uddi.security.SecurityBackendApi"/>
    </backendMapping>
    <security>
        <authInfoTimeOut>1800~~3600~~</authInfoTimeOut>
        <keyIdentity>authTokenIdentity</keyIdentity>
        <keyPassword_coded>vcVllo4QBpY9oEDgfgqAfw==</keyPassword_coded>
        <tokenCreationTimeTolerance>5000</tokenCreationTimeTolerance>

        <tokenFactory>com.systinet.uddi.security.token.NonSSOTokenFactory</tokenFactory>
    </security>
</config>
```

Configuration from collection (old):

This is what a comparison between a collection and the current set of configurations looks like. It shows the differences of matching pairs of all configurations. Matching configurations where no differences appear are listed below. Non-matching configurations (when the configuration appears in the collection only or in the current set only) are also listed below.

You can alter the options for differences comparison: whether it is case sensitive and whether the full text is shown or omitted. It is not recommended to show full text in all differences because the resulting page might get very long.

View collection



View Collection

This is a named collection. It is persistent and will not be automatically deleted when it is old. You can delete it manually.

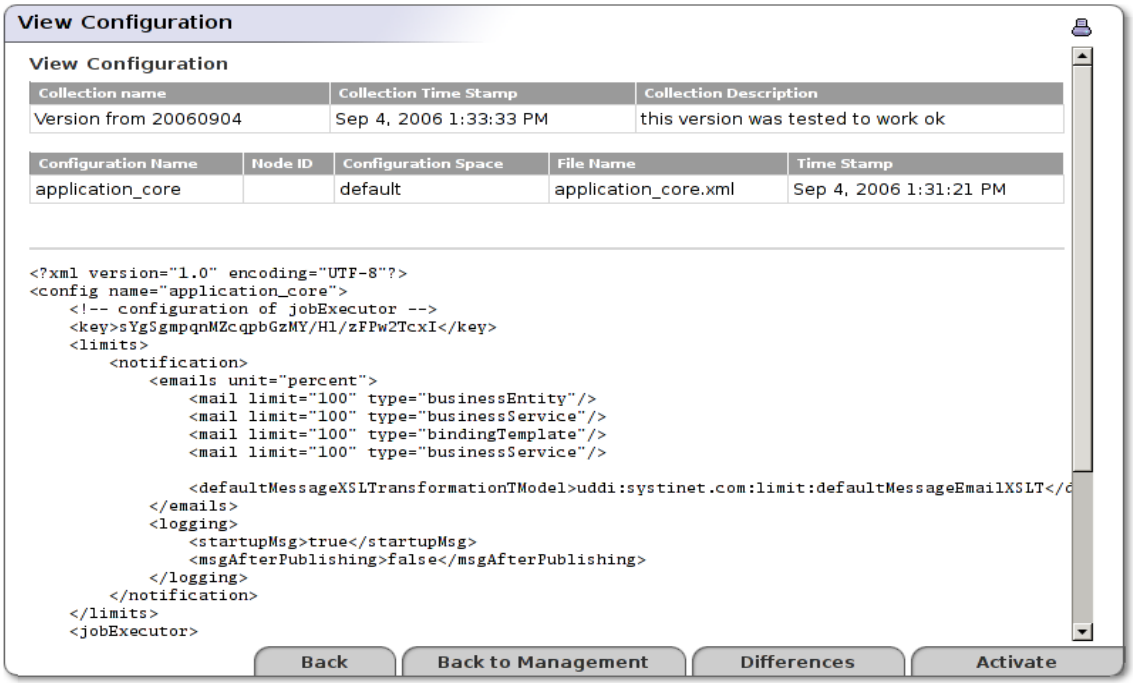
Collection name	Collection Time Stamp	Collection Description
Version from 20060904	Sep 4, 2006 1:33:33 PM	this version was tested to work ok

This collection contains these configurations:

Configuration Name	Node ID	Configuration Space	File Name	Time Stamp
account		default	account.xml	Sep 4, 2006 1:21:27 PM
account_core		default	account_core.xml	Sep 4, 2006 1:31:11 PM
account_list		default	account_list.xml	Sep 4, 2006 1:21:27 PM
admin_utils		default	admin_utils.xml	Sep 4, 2006 1:21:27 PM
application_core		default	application_core.xml	Sep 4, 2006 1:31:21 PM
bsc		BSC	bsc.xml	Sep 4, 2006 1:21:33 PM
bsc_web		BSC	web.xml	Sep 4, 2006 1:21:33 PM
bsc_web_component		BSC	web_component.xml	Sep 4, 2006 1:21:33 PM
category_v3		default	category_v3.xml	Sep 4, 2006 1:21:27 PM
component_description		BSC	component_description.xml	Sep 4, 2006 1:21:33 PM
config_management		default	config_management.xml	Sep 4, 2006 1:21:27 PM
configurator	cluster-1	default	configurator.xml	Sep 4, 2006 1:21:27 PM
configurator	cluster-2	default	configurator.xml	Sep 4, 2006 1:24:06 PM

Collection content usually looks like this. When you click on the configuration name its view with actions is displayed.

View configuration



View Configuration

Collection name	Collection Time Stamp	Collection Description
Version from 20060904	Sep 4, 2006 1:33:33 PM	this version was tested to work ok

Configuration Name	Node ID	Configuration Space	File Name	Time Stamp
application_core		default	application_core.xml	Sep 4, 2006 1:31:21 PM

```
<?xml version="1.0" encoding="UTF-8"?>
<config name="application_core">
    <!-- configuration of jobExecutor -->
    <key>sYgSgmpqnMZcqpbGzMY/H1/zFPw2TcxI</key>
    <limits>
        <notification>
            <emails unit="percent">
                <mail limit="100" type="businessEntity"/>
                <mail limit="100" type="businessService"/>
                <mail limit="100" type="bindingTemplate"/>
                <mail limit="100" type="businessService"/>

                <defaultMessageXSLTransformationTModel>uddi:systinet.com:limit:defaultMessageEmailXSLT</d
            </emails>
            <logging>
                <startupMsg>true</startupMsg>
                <msgAfterPublishing>false</msgAfterPublishing>
            </logging>
        </notification>
    </limits>
    <jobExecutor>
```

This view shows a configuration stored inside a collection. You can see the comparison between this configuration and the current configuration by clicking on button **Differences**. You can also make this configuration the current with the **Activate** button (after confirmation).

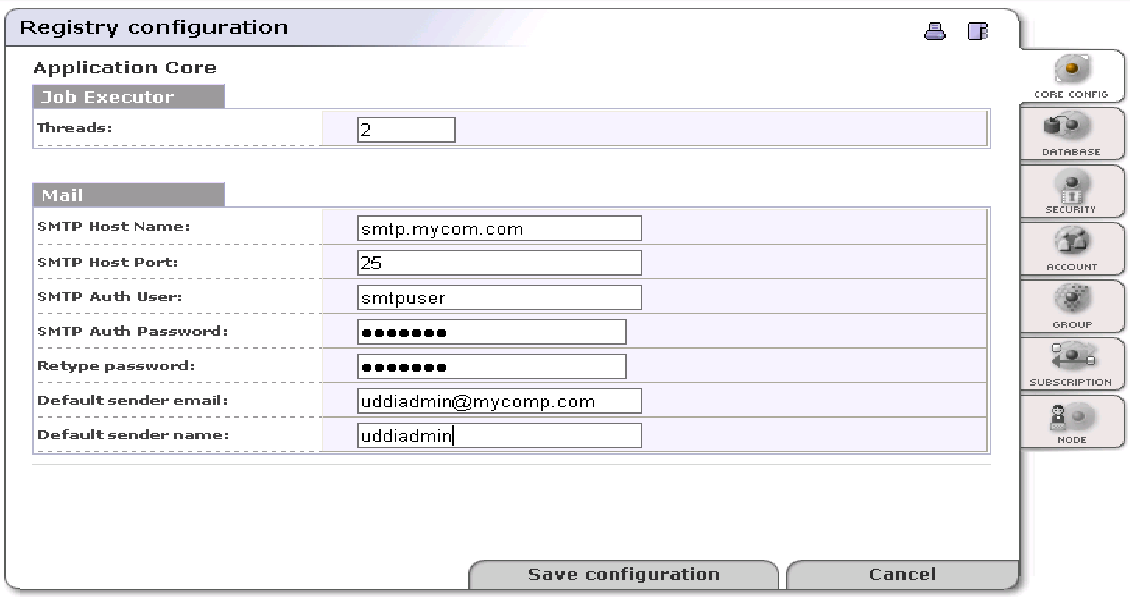
Registry Configuration

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 the "Rules to Display the Manage Tab" note in "Accessing Registry Management" on page 297.
2. Click the **Manage** main menu tab.
3. Select the **Registry configuration** link under **Manage** tab. This returns the Registry configuration panel shown in the following figure.

Registry Configuration



Registry configuration

Application Core

Job Executor

Threads:	2

Mail

SMTP Host Name:	smtp.mycom.com
SMTP Host Port:	25
SMTP Auth User:	smtpuser
SMTP Auth Password:	●●●●●●●
Retype password:	●●●●●●●
Default sender email:	uddiadmin@mycomp.com
Default sender name:	uddiadmin

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.

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.

Retries in case of DB deadlock

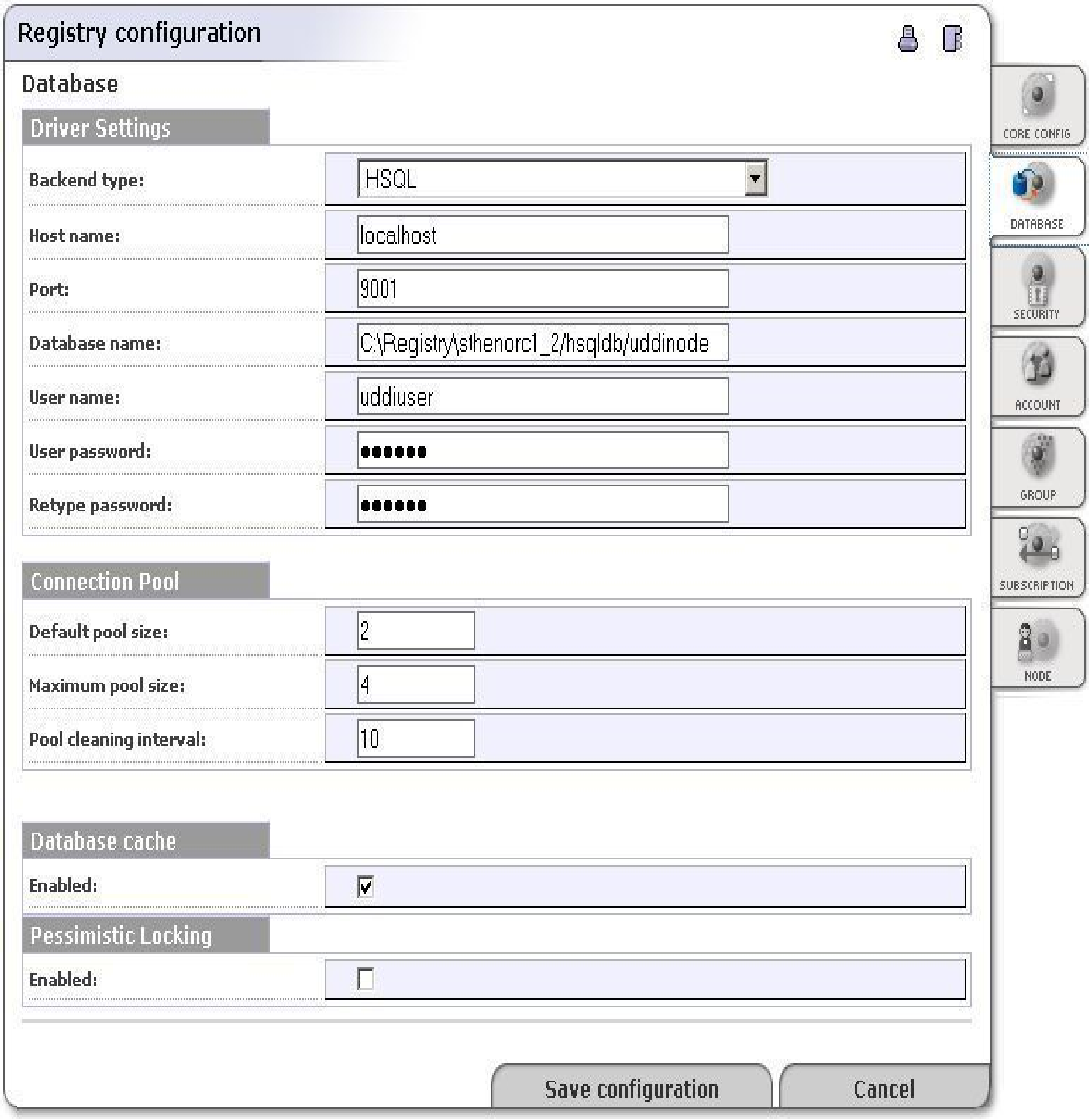
The maximum number of retries in case of DB deadlocks. This option doesn't appear in the Registry Configuration, you can edit the option manually in the configuration file `REGISTRY_HOME\app\uddi\conf\application_core.xml`. After editing, add the attribute updateDB=”true” to the top level config tag to update the database from the edited configuration file. See more at "Configuration in Database" on page 343.

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, see the “Default Ports for Supported Database Servers” table below.

Database installation, that is, creating the database schema and loading basic data, is described in "Database Installation" on page 80.

Registry Configuration - Database



Registry configuration

Database

Driver Settings

Backend type:	HSQL
Host name:	localhost
Port:	9001
Database name:	C:\Registry\sthenorc1_2/hsqldb/uddinode
User name:	uddiuser
User password:	••••••
Retype password:	••••••

Connection Pool

Default pool size:	2
Maximum pool size:	4
Pool cleaning interval:	10

Database cache

Enabled:	☑

Pessimistic Locking

Enabled:	□

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 5, “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.

Pessimistic Locking: This option is not recommended unless there are excessive deadlock recovery messages in the log. It affects performance.

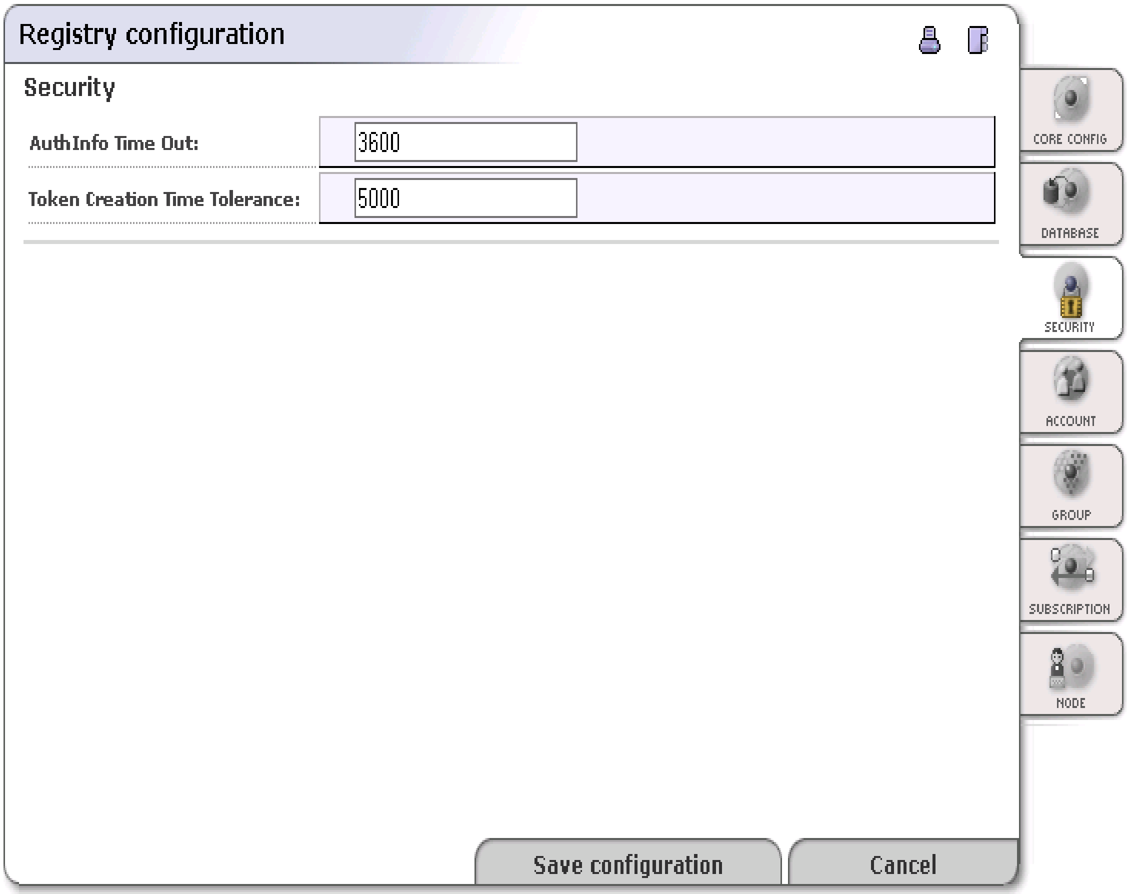
Default Ports for Supported Database Servers

Database	Default Port
Oracle 11g or 12c	1521
MS SQL 2005 or 2008	1433
Sybase ASE 12.5	5000
PostgreSQL	5432
hsqldb 1.7.3	-

Security

On the **Security** tab, you can configure your digital signature token and key properties.

Registry Configuration - Security



Security

AuthInfo Time Out: 3600

Token Creation Time Tolerance: 5000

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.

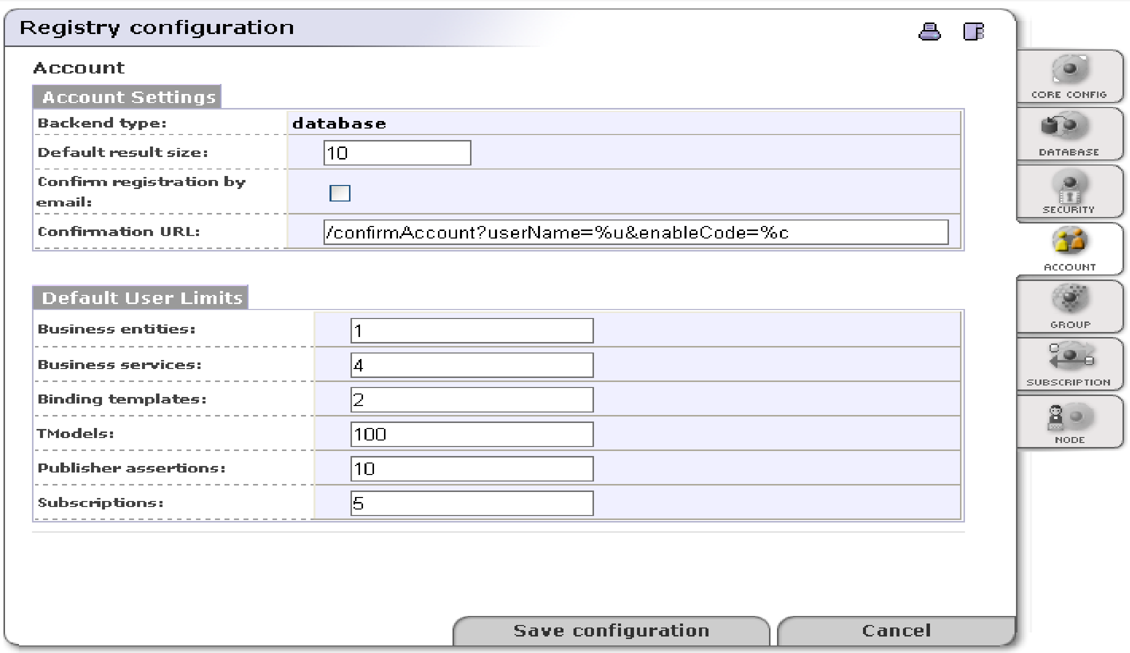
Token Creation Time Tolerance: Tolerance interval of token validity, expressed in milliseconds.

Token Signature: Whether authorization token is signed. We recommend you toggle this switch on.

Account

On this tab, you can specify accounts properties applicable for all HPE SOA Registry Foundation user accounts.

Registry Configuration - Account



Registry configuration

Account

Account Settings

Backend type:	database
Default result size:	10
Confirm registration by email:	□
Confirmation URL:	/confirmAccount?userName=%u&enableCode=%c

Default User Limits

Business entities:	1
Business services:	4
Binding templates:	2
TModels:	100
Publisher assertions:	10
Subscriptions:	5

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.

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.

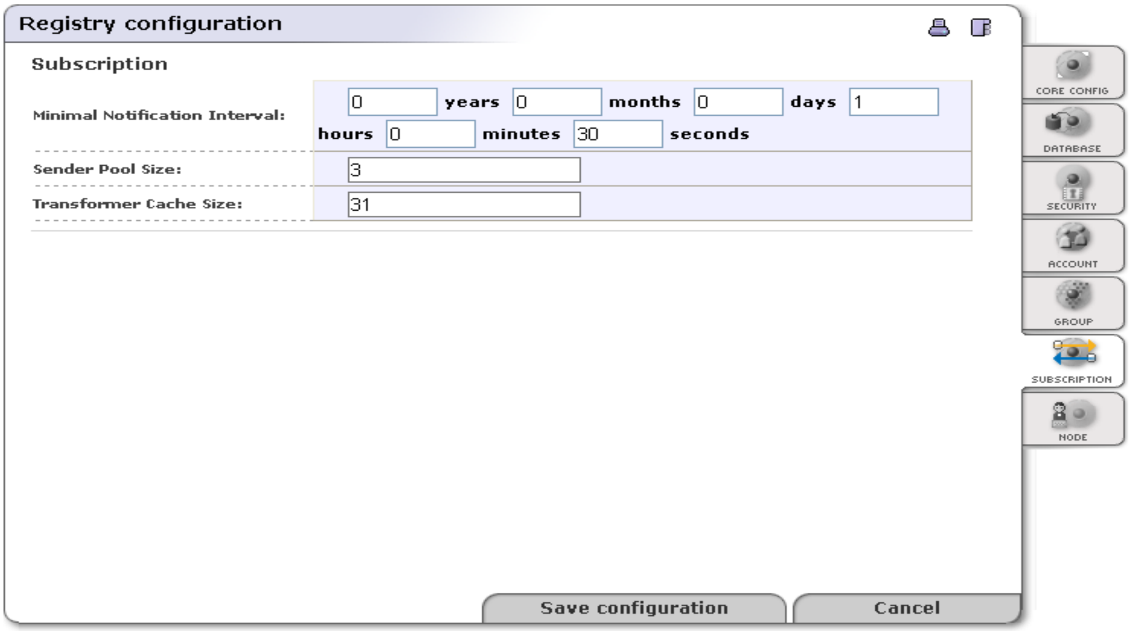
Default result size

Number of items returned in search results when querying groups; the default value for this field is 10.

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.

Registry Configuration - Subscriptions



Registry configuration

Subscription

Minimal Notification Interval: 0 years 0 months 0 days 1 hours 0 minutes 30 seconds

Sender Pool Size: 3

Transformer Cache Size: 31

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

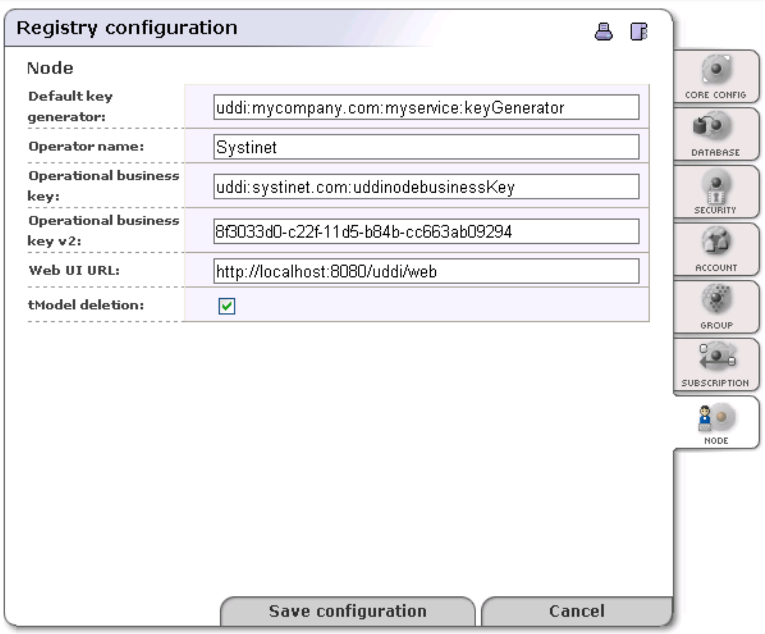
Transformer Cache Size

Number of cached XSLT transformations

Node

On the **Node** tab, you can configure UDDI node properties.

Registry Configuration - Node



Registry configuration

Node

Default key generator:	uddi:mycompany.com:myservice:keyGenerator
Operator name:	Systinet
Operational business key:	uddi:systinet.com:uddinodebusinessKey
Operational business key v2:	8f3033d0-c22f-11d5-b84b-cc663ab09294
Web UI URL:	http://localhost:8080/uddi/web
tModel deletion:	☑

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, see "Publisher-Assigned Keys" on page 218 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.)

Configuration in Database

HPE SOA Registry Foundation uses many configuration files. They are stored in `REGISTRY_HOME\app\uddi\conf` and `REGISTRY_HOME\work\uddi\bsc.jar\conf` directories. Some of them may be changed during setup or with web interfaces.

Each such configuration file is an XML file containing tag config with some information about how the configuration file is used.

These attributes are generally recognized:

Attributes of config tag

Attribute	Meaning	Optional	Default value
name	configuration name	no	
local	true when the file is local to the cluster node	yes	false
updateDB	when true the file is stored in the database on HPE SOA Registry Foundation start	yes	false
history	when false configuration history is not logged	yes	true
savingPeriod	delay before changes in memory are written to file in milliseconds	yes	2000

The most important attribute is name which is the identifier by which HPE SOA Registry Foundation tries to find the configuration. Some configuration files have attribute local set to true. That means that the configuration is only used by this HPE SOA Registry Foundation and other Registries in the cluster will not share it. Other nodes will have their own independent versions. These cluster nodes are distinguished by the Node ID which is specified inside `nodeid.xml`. If its value is empty, a unique ID will be generated at HPE SOA Registry Foundation startup.

The configuration files are always present in the directories, however their copy is in the database. If a configuration file is present in both database and file-system, the one in the database has priority. After the initial startup of HPE SOA Registry Foundation all configurations are put into the database. When HPE SOA Registry Foundation needs to change some configuration settings it does so in the both the database and file-system.

If a user or another program like HPE SOA Registry Foundation setup wants to edit the configuration file the priority of the configuration in database has to be overridden. This can be done in two ways:

1. By setting attribute `updateDB` to true in the top-level tag config in all configuration files where modifications have been done. Once HPE SOA Registry Foundation starts, the attribute will be automatically removed.
2. By setting attribute updateDBAll (see in table below) to true in tag dbconfig in `database.xml` Once HPE SOA Registry Foundation starts the attribute will be automatically cleared. There can be also time stamp in this attribute in format like 20070321133058 where digits denote year, month, day of month, hour, minute, and second in GMT time zone. Such time stamp is compared to time stamp in database. When config files have more recent time, they will be put in database on HPE SOA Registry Foundation start. When stamp in database is more recent, database version will be used. In both cases the attribute will be cleared.

   Time stamp in updateDBAll is used by setup. Each time setup task is run it updates time stamp except for task that do not modify configuration files like drop database and backup. Purpose of the time stamp is to prevent overwritting current configuration with old one while redeploying same EAR/WAR file to application server.

   When HPE SOA Registry Foundation operates in cluster mode the other means than the time stamp is used for synchronization. Clocks on cluster nodes are assumed to be not enough precise for that, but enough precise for redeployment and configuration changes. There is only one time stamp in database.xml, individual configuration files allow only true/false values in updateDB attribute.

The other important configuration setting for configurations is inside the `database.xml` file, in the dbconfig tag. The tag has following attributes:

Attributes of dbconfig tag

Attribute	Meaning	Optional	Default value
configRetainCount	number of latest configuration versions that are not deleted	yes	10
configRetainMinutes	age of configuration version before it can be deleted	yes	10
eventRetainMinutes	age of event information before it can be deleted	yes	5
updateDBAll	when true all configuration files will be stored in the database at HPE SOA Registry Foundation startup, can be also a time stamp	yes	false

> **Note:** HPE SOA Registry Foundation setup automatically sets the updateDBAll attribute when its

> operation has been successful so that all changed configurations will be stored in the database at HPE SOA Registry Foundation startup. This is usually desirable behavior.

> **Note:** When HPE SOA Registry Foundation encounters an identical configuration in the database to the one that is being stored (e.g. when set updateDB or updateDBAll is encountered) then the store operation is ignored. This may be surprising as there would be no entry in the log of configurations, however the resulting state of the configurations is correct.

The database not only holds the current set of configurations but also their history in a log. You can monitor configuration changes, what the previous content was, or let HPE SOA Registry Foundation show you differences between versions. This configuration history log is purged every few minutes. Old configurations are not retained indefinitely. There are rules on how many older versions are left there and the age of a configuration before it can be deleted. The purpose of these rules is to avoid running out of space in the database and yet still have information about recent changes. Rules can be configured inside tag dbconfig in `database.xml`. Their defaults are in the table above. Default settings specify that there must be at least `configRetainCount` new versions of the configuration before it can be deleted automatically. Also, the configuration has to be older than configRetainMinutes before it can be deleted automatically. This allows the correction of most non-fatal configuration errors after an invalid change or to track which configuration change might have caused unexpected behavior.

To allow easy comparison of current and older configurations or try-then-rollback scenarios, the current set of configurations can be stored into a named collection of configurations. These collections are not deleted automatically. They allow you to store a configuration that works correctly and compare it with the current version if something breaks later. You can then activate the old one if needed or change the incorrect setting manually.

> **Note:** Backup tool in setup can store both file and database configurations. You can select which you want to backup.

Configurations in the database can be managed with the "Configuration Management" component of the Registry Console. You can find it under the Manage tab, then Registry management sub-tab (default), then Configuration Management button.

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.

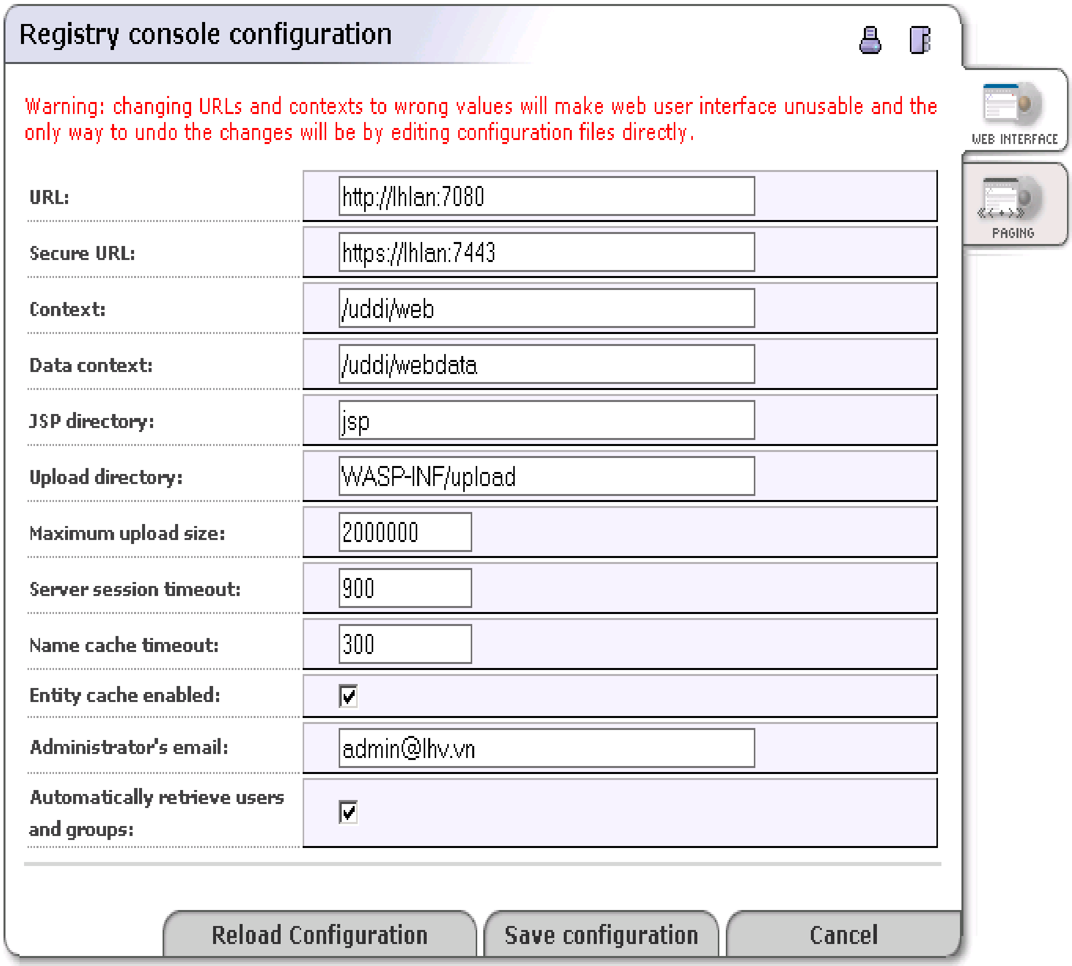
To access the Registry Console configuration:

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, “Registry Console Configuration - Web Interface Tab”. The Registry Console Configuration screen has two tabs:
   - On the **Web Interface** tab, you can set various parameters associated with HPE SOA Registry Foundation'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.

Web Interface Configuration

Registry Console Configuration - Web Interface Tab



Registry console configuration

Warning: changing URLs and contexts to wrong values will make web user interface unusable and the only way to undo the changes will be by editing configuration files directly.

URL:	http://lhlan:7080
Secure URL:	https://lhlan:7443
Context:	/uddi/web
Data context:	/uddi/webdata
JSP directory:	jsp
Upload directory:	WASP-INF/upload
Maximum upload size:	2000000
Server session timeout:	900
Name cache timeout:	300
Entity cache enabled:	☑
Administrator's email:	admin@lhv.vn
Automatically retrieve users and groups:	☑

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

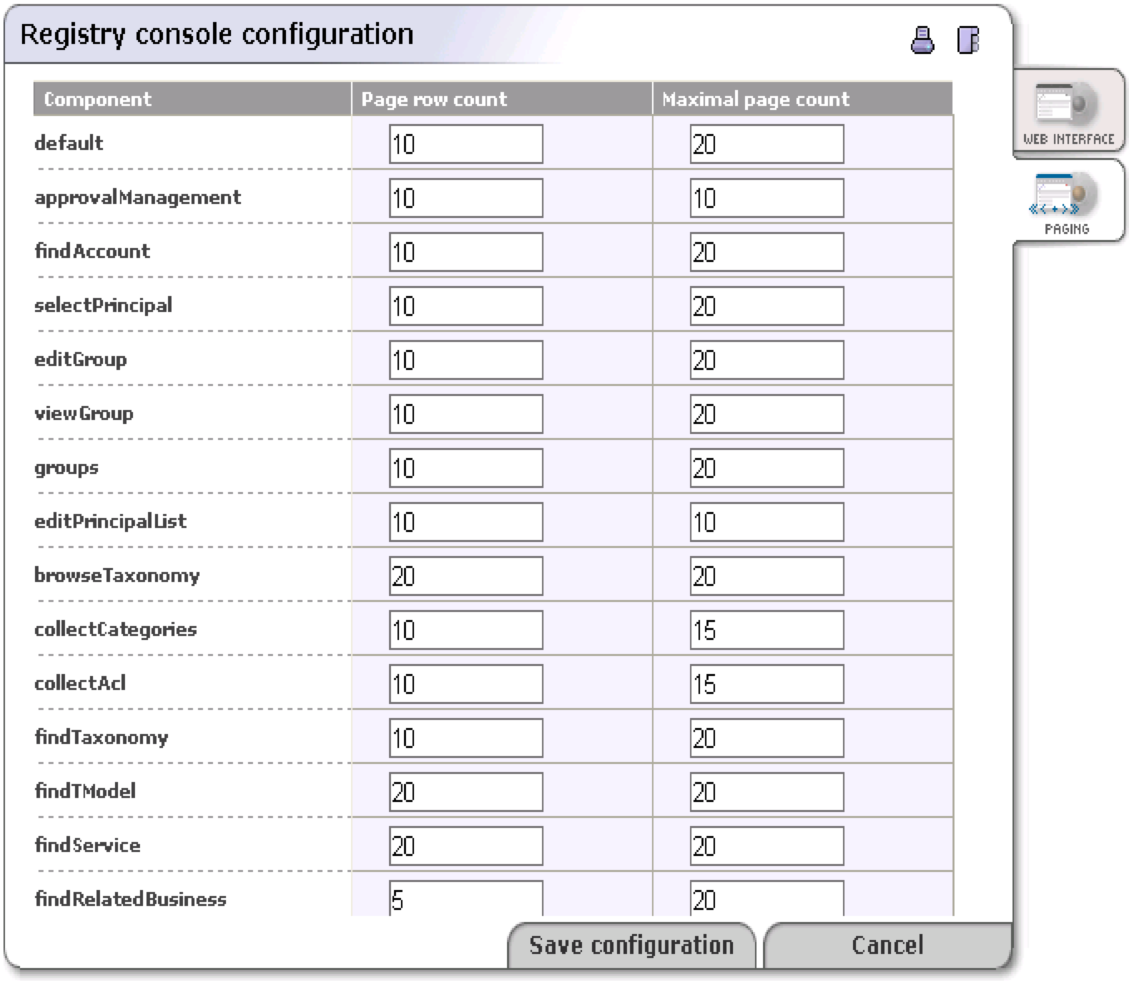
(measured in seconds).

- **Entity cache enabled** - If you check this check box, entities will be cached.

Click **Save configuration** when finished.

Paging Configuration

Registry Console Configuration - Paging Tab



Registry console configuration

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.

Permissions: Principles

Permissions in HPE SOA Registry Foundation were developed so that administrators might exercise control over users. Permissions:

- Provide a simple mechanism for the management of users' rights in HPE SOA Registry Foundation.
- Allow the administrator to manage or make available different parts of the registry to different users.
- Help HPE SOA Registry Foundation 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 HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation. 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.

Permissions Definitions

There are two basic kinds of permission:

- The first, consisting of "ApiUserPermission" on the next pageand "ApiManagerPermission" on the next page, is used to restrict access for some users on some interfaces.

- The second, "" below, is used to restrict the ability to change configurations in HPE SOA Registry Foundation.

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.

HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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 " Data Access Control: Principles" on page 212 in the User's Guide.

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.

> **Note:** 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`.

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 HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation is nearly the same as user role mechanism and it is used instead of user roles.

HPE SOA Registry Foundation contains the following built-in groups that represent basic roles. Each role has appropriate permissions already defined. So, administrator can set simply permissions by adding users into these groups. For more information, see "Group Management" on page 305.

`accountManagerGroup`

Members of the group `accountManagerGroup` are able to manage accounts. For example, they can create new accounts, edit and delete existing ones.

`administrationUtilsManagerGroup`

Members of the group `administrationUtilsManagerGroup` are able to use administration utilities. For example, they can delete tModels permanently, replace keys, replace URLs.

`bscConfiguratorGroup`

Members of the group `bscConfiguratorGroup` are able to configure settings for Business Service Console.

`configuratorGroup`

Members of the group `configuratorGroup` are able to configure setting for HPE SOA Registry Foundation. This means that they can set consoles, database, mail settings and so on.

`groupManagerGroup`

Members of the group `groupManagerGroup` are able to manage groups. For example, they can create new groups, edit or delete existing ones.

`permissionManagerGroup`

Members of the group `permissionManagerGroup` are able to manage permissions. For example, they can add permission to some principal or remove permission from some principal.

`replicationManagerGroup`

Members of the group `replicationManagerGroup` are able to manage replication. For example, they can create new replication or manage the existing one.

`statisticsManagerGroup`

Members of the group `statisticsManagerGroup` are able to view or reset statistics.

`taxonomyManagerGroup`

Members of the group `taxonomyManagerGroup` are able to manage taxonomies. For example, they can upload or delete taxonomy.

`webConfiguratorGroup`

Members of the group `webConfiguratorGroup` are able to configure Registry Console.

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.

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.

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.)

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.

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.

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.

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.

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.

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.

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.)

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	Default in system#everyone group. When removed, only other authentication methods will work.

discard_ authToken	Default in system#everyone group.
get_ registeredInfo	Not used.
validate_ categorization	Not used.

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	Default in system#everyone group. When removed, only other authentication methods will work.
discard_authToken	Default in system#everyone group.
get_registeredInfo	Not used.

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.

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.)

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.)

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.

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.)

PStore Tool

The PStoreTool provides HPE SOA Registry Foundation Protected Store management. It provides functionality to:

- Import and export trusted certificates locally to or from a file.
- Create new security identities in the HPE SOA Registry Foundation configuration file.
- Copy identities between protected stores.

> **Note:** Use "SSL Tool" on page 368 to import and export a key entry to or from HPE SOA Registry Foundation protected store.

> **Note:** Remote protected store management via SOAP is not supported with HPE SOA Registry Foundation.

The general usage is:

```
PStoreTool [command [options]]
```

You can perform operations from the command line or start up a GUI interface.

Commands

The PStore tool has the following commands (see also "Options" on the next page):

- **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 HPE SOA Registry Foundation. 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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation. This command also adds the mapping between the security identity alias and its X.509 certificate to the user store part of the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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.

Options

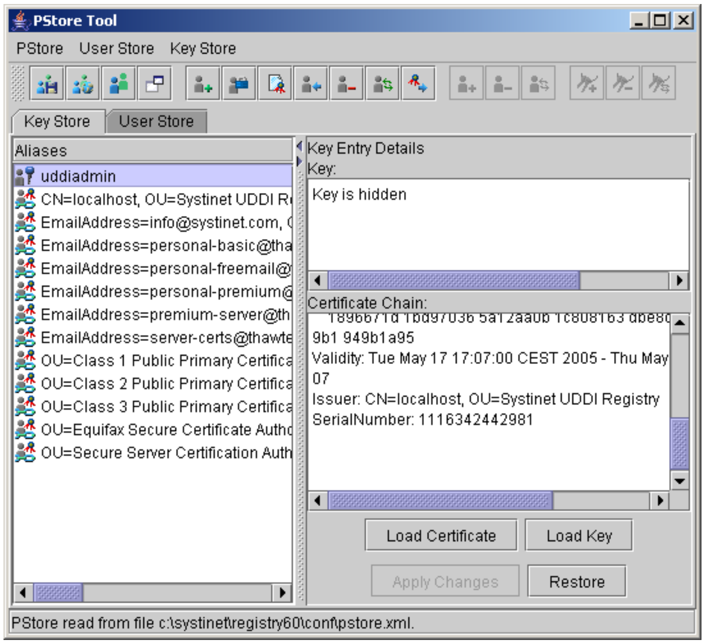
The PStore tool has the following options:

- `-alias alias` - This option must be used with a command that refers to an alias.
- `-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 HPE SOA Registry Foundation 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.

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.

PStore Tool



uddiadmin
CN=localhost, OU=Systinet UDDI R
EmailAddress=info@systinet.com,
EmailAddress=personal-basic@tha
EmailAddress=personal-freemail@
EmailAddress=personal-premium@
EmailAddress=premium-server@th
EmailAddress=server-certs@thawte
OU=Class 1 Public Primary Certifica
OU=Class 2 Public Primary Certifica
OU=Class 3 Public Primary Certifica
OU=Equifax Secure Certificate Autho
OU=Secure Server Certification Auth

Key is hidden

```
1896671d 1bd97036 5a12aa0b 1c808163 dbe8c
9b1 949b1a95
Validity: Tue May 17 17:07:00 CEST 2005 - Thu May
07
Issuer: CN=localhost, OU=Systinet UDDI Registry
SerialNumber: 1116342442981
```

PStore read from file c:\systinet\registry60\conf\pstore.xml.

Running the GUI PStore Tool

To run the graphical version of this tool, use gui as parameter with the `PStoreTool` command.

```
PStoreTool gui
```

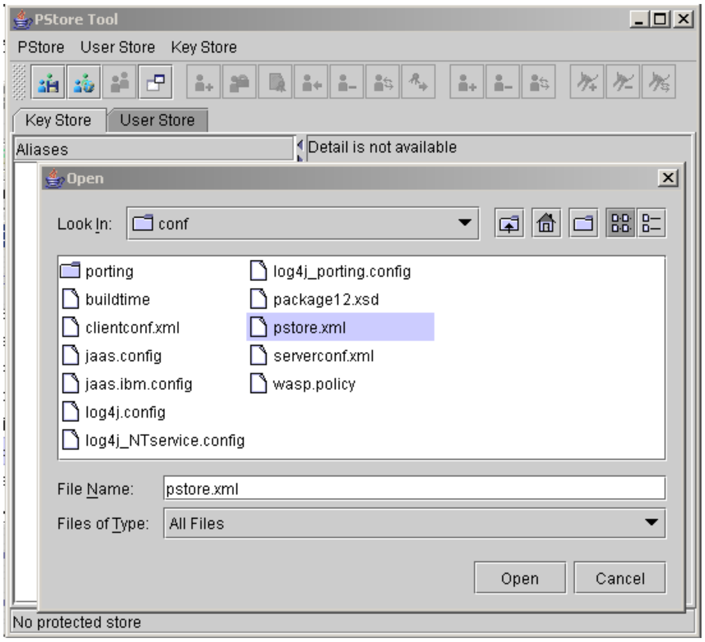
Opening and Closing the Protected Store

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 the following figure, “Open Protected Store from a File”.

Open Protected Store from a File



Detail is not available

File Name: pstore.xml

Files of Type: All Files

No protected store

Closing Protected Store

To close the protected store, select **Close** from the **PStore** menu.

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.

Copy Data Between Protected Stores

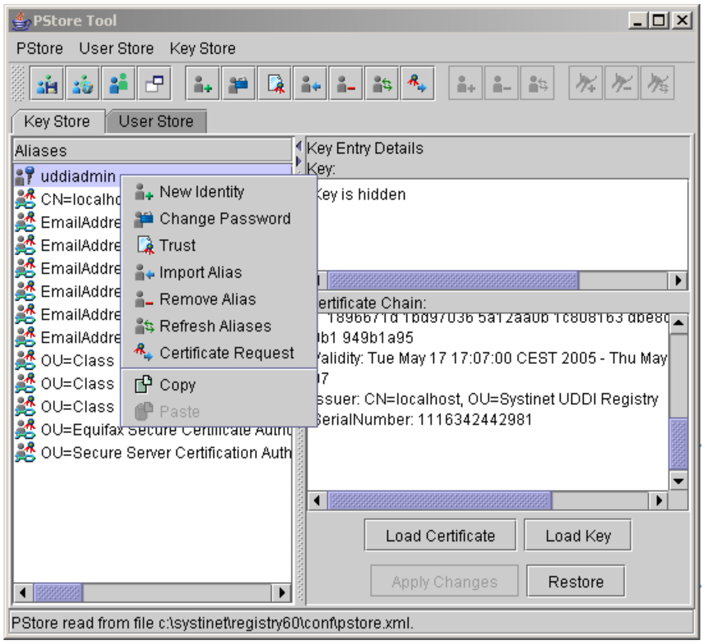
With the PStore Tool, you can manipulate more than one protected store at the same time. You can simply copy identities, certificates, users, and user properties from one protected store to another using the Copy and Paste actions located in context menus of the Aliases, Users, and Properties panels.

> **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.

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.

Key Store Tab



```
Key is hidden
```

```
1896671d 1bd97036 5a12aa0b 1c808163 dbe8c
b1 949b1a95
'alidity: Tue May 17 17:07:00 CEST 2005 - Thu May
7
ssuer: CN=localhost, OU=Systinet UDDI Registry
erialNumber: 1116342442981
```

```
PStore read from file c:\systinet\registry60\conf\pstore.xml.
```

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.

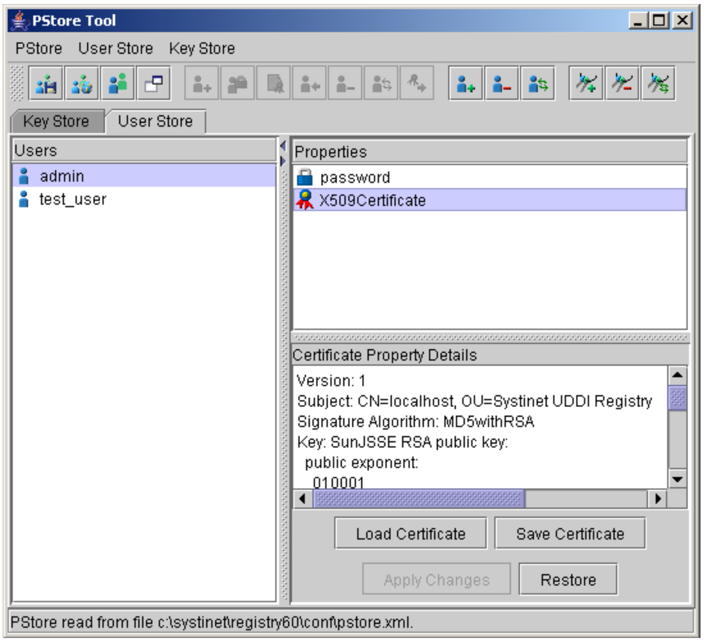
Alias Details Panel

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**.

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.

User Store Tab



```
Version: 1
Subject: CN=localhost, OU=Systinet UDDI Registry
Signature Algorithm: MD5withRSA
Key: SunJSSE RSA public key:
  public exponent:
    010001
```

PStore read from file c:\systinet\registry60\conf\pstore.xml.

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**.

SSL Tool

The sslTool helps with setup of SSL on the client side of HPE SOA Registry Foundation. The general usage is:

```
sslTool [command [options]]
```

The SSL tool has the following commands:

- **serverInfo** - Prints out security requirements of an SSL server and saves a server certificate to a file.
- **encrypt** - Prints out the encrypted form of a password supplied as plain text. Encrypted passwords are used in the configuration files of HPE SOA Registry Foundation.
- **pstoreEI** - Exports and imports a java keystore to or from the HPE SOA Registry Foundation Protected Store. Both `PKCS12` and `JKS` keystores are supported. The type of a supplied keystore is automatically detected during import.

SSL Tool Examples

To print out security requirements of an SSL server:

```
sslTool serverInfo --url https://localhost:8443
```

To print out security requirements of an SSL server and save server certificates:

```
sslTool serverInfo --url https://localhost:8443 --certFile /tmp/sever.cer
```

To print out an encrypted password for use in HPE SOA Registry Foundation configuration files:

```
sslTool encrypt --password changeit
```

To import a key entry from a java keystore to HPE SOA Registry Foundation client Protected Store:

```
sslTool pstoreEI -i --keystore /tmp/java.keystore
                      --storepass changeit --alias mykey --keypass changeit
                    --pstore ../conf/clientconf.xml
                      --pstoreAlias registryclient --pstoreKeypass changeit2
```

To export a key entry from HPE SOA Registry Foundation Protected Store to a java keystore:

```
sslTool pstoreEI -e --keystore /tmp/java.keystore2
                      --storepass changeit --alias mykey --keypass changeit
                    --pstore ../conf/clientconf.xml
                      --pstoreAlias registryclient --pstoreKeypass changeit2
```

Associating an SSL client identity with a registry client

Instructions on how to associate an SSL client identity with a registry client are explained in "Example Client" on page 480 in the Developer's Guide. In this case, a key entry must be imported to registry's client protected store, which is the `conf/clientconf.xml` file of the registry installation directory and a few system properties must be added to a script that runs the client application.

There are also cases where a registry acts as a client to another registry. These include:

- Communication between nodes in a clustered HPE SOA Registry Foundation.

Associating an SSL client identity with a HPE SOA Registry Foundation server can be done in the `app/uddi/conf/security.xml` file of a registry installation directory (or deployed package for a deployed registry) by adding the destinationConfig elements. A fragment of the `security.xml` with example `destinationConfig` elements is shown in the following example, “Association of client identities with a registry server”.

Association of client identities with a registry server

```
<?xml version="1.0" encoding="UTF-8"?>
<config name="security" savingPeriod="5000">
     ...
     <security>
     ...
    </security>
    <!-- For communication with other nodes in the cluster -->
    <destinationConfig>
      <alias>clusterClient</alias>
      <password_coded>gNFDFWMNdkU=</password_coded>
      <destination
proxyName="com.systinet.uddi.configurator.cluster.ConfiguratorManagerStub"/>
      <destination
proxyName="com.systinet.uddi.configurator.cluster.ConfiguratorListenerStub"/>
    </destinationConfig>
    <!-- For communication via registry client to services accessible
      at URLs that start with https://pc1.mycom.com or https://pc2.mycom.com -->
    <destinationConfig>
      <alias>otherClient</alias>
      <password_coded>Vr+i+UzC2WLJXWg0ih6J+Q==</password_coded>
      <destination url="https://pc1.mycom.com/*"/>
      <destination url="https://pc2.mycom.com/*"/>
    </destinationConfig>
</destinationConfig>

</config>
```

There can be more `destinationConfig` elements. A `destinationConfig` element is used to associate a particular SSL client identity with a set of destinations. It contains:

- alias in the server protected store. A key entry with the same name as the alias must exist in a server's Protected Store. This key entry represents security material used to establish SSL with a destination server. The HPE SOA Registry Foundation server Protected Store is in the `conf/pstore.xml` file of a registry deployment package. Use this file when importing a key entry from a java keystore, as shown in "SSL Tool Examples" on page 368.
- password_coded element, which contains the encrypted password that is used to access a private key stored under the alias supplied. See "SSL Tool Examples" on page 368 for an example that prints out the encrypted form of a password supplied in plain text.
- One or more destination elements each specify a rule. The rule can contain `url` or `proxyName` attributes. The rule matches when a client use a proxy class specified by the proxyName attribute or connects to a URL that is specified by the `url` attribute. The value of the url can end with a wildcard * to specify a match of all URLs that start with the string specified before the wildcard. The whole `destinationConfig` element matches if at least one rule matches.

The first matching `destinationConfig` is used.

Chapter 5: Developer's Guide

The Developer's Guide is divided into the following main parts:

- Mapping of Resources covers registering various XML resources in HPE SOA Registry Foundation including WSDL definitions, schemas, and transformations.
- Client-Side Development describes the basic principles of using HPE SOA Registry Foundation 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 HPE SOA Registry Foundation web framework is also described in this section.
- UDDI From Developer Tools discusses how to access UDDI Microsoft Visual Studio .NET.
- How to debug describes logging and using the SOAPSpy tool.

Mapping of Resources

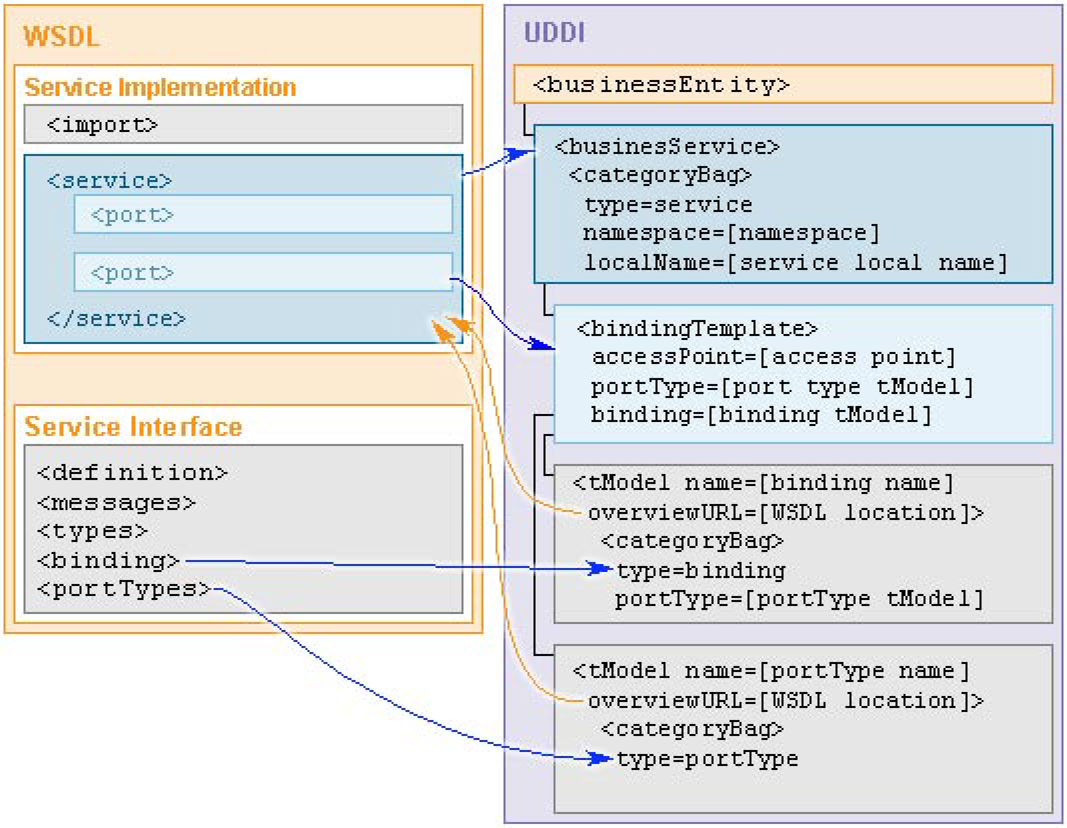
HPE SOA Registry Foundation provides you with functionality to register the following resources:

- "WSDL" below definition
- "XML" on page 375 Schema (XSD)
- "XSD" on page 375
- "XSLT" on page 377

WSDL

This describes how to publish a WSDL file to HPE SOA Registry Foundation. The implementation reflects the OASIS UDDI technical note Using WSDL in a UDDI Registry, Version 2.0. As shown in the following figure, the technical note suggests a mapping between WSDL and UDDI.

WSDL TO UDDI



WSDL PortTypes

As shown in the following table, “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.

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

WSDL Bindings

In similar fashion, as summarized in the following table, “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.

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

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.

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

Use Cases

HPE SOA Registry Foundation 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

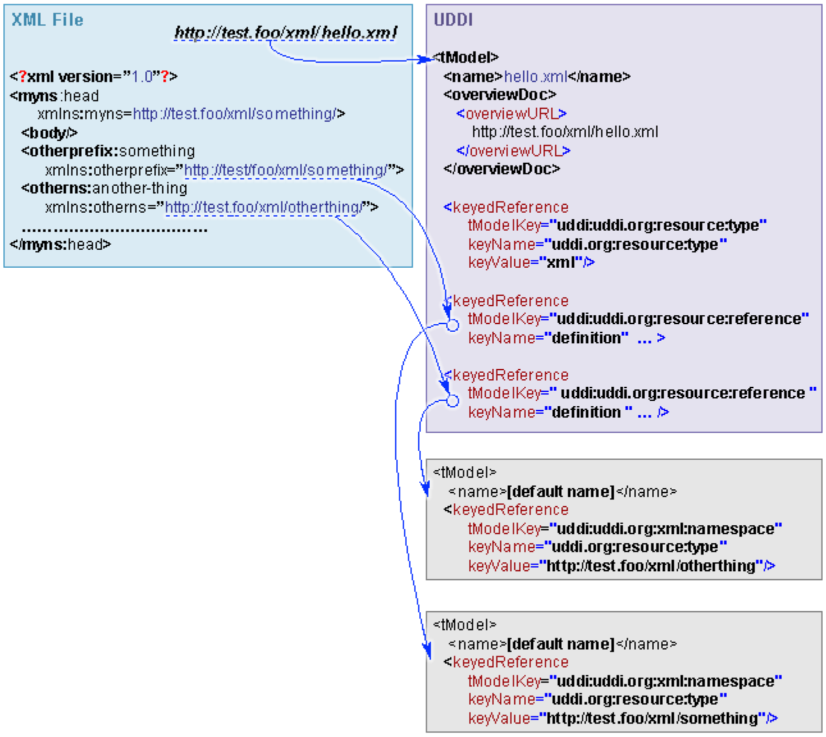
For more information, also see:

- User's Guide, “Publishing WSDL Documents” in "Publishing" on page 265
- User's Guide, “Find WSDL” in "Searching" on page 256
- Developer's Guide, "WSDL Publishing" on page 422

XML

As shown in the following figure, 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.

XML TO UDDI



```
<?xml version="1.0"?>
<myns:head
     xmlns:myns=http://test.foo/xml/something/>
  <body/>
  <otherprefix:something
      xmlns:otherprefix="http://test/foo/xml/something/">
  <otherns:another-thing
      xmlns:otherns="http://test.foo/xml/otherthing/">
  ....................................
</myns:head>
```

```
<tModel>
  <name>hello.xml</name>
  <overviewDoc>
    <overviewURL>
      http://test.foo/xml/hello.xml
    </overviewURL>
  </overviewDoc>

  <keyedReference
      tModelKey="uddi:uddi.org:resource:type"
      keyName="uddi.org:resource:type"
      keyValue="xml"/>

  <keyedReference
      tModelKey="uddi:uddi.org:resource:reference"
      keyName="definition"  ... >

  <keyedReference
      tModelKey=" uddi:uddi.org:resource:reference "
      keyName="definition " ... />
```

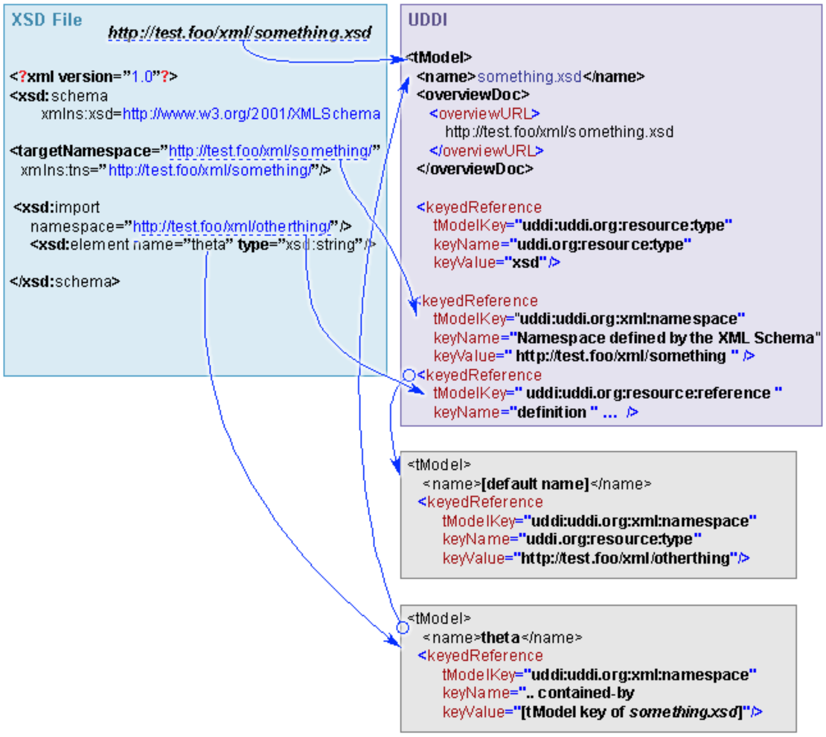
```
<tModel>
   <name>[default name]</name>
  <keyedReference
      tModelKey="uddi:uddi.org:xml:namespace"
      keyName="uddi.org:resource:type"
      keyValue="http://test.foo/xml/otherthing"/>
```

```
<tModel>
   <name>[default name]</name>
  <keyedReference
      tModelKey="uddi:uddi.org:xml:namespace"
      keyName="uddi.org:resource:type"
      keyValue="http://test.foo/xml/something"/>
```

XSD

As shown in the following figure, 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.

XSD to UDDI



```
<?xml version="1.0"?>
<xsd:schema
    xmlns:xsd=http://www.w3.org/2001/XMLSchema

<targetNamespace="http://test.foo/xml/something/"
  xmlns:tns="http://test.foo/xml/something/"/>

 <xsd:import
    namespace="http://test.foo/xml/otherthing/"/>
    <xsd:element name="theta" type="xsd:string"/>

</xsd:schema>
```

```
<tModel>
  <name>something.xsd</name>
  <overviewDoc>
    <overviewURL>
      http://test.foo/xml/something.xsd
    </overviewURL>
  </overviewDoc>

  <keyedReference
    tModelKey="uddi:uddi.org:resource:type"
    keyName="uddi.org:resource:type"
    keyValue="xsd" />

<keyedReference
    tModelKey="uddi:uddi.org:xml:namespace"
    keyName="Namespace defined by the XML Schema"
    keyValue=" http://test.foo/xml/something " />
<keyedReference
    tModelKey=" uddi:uddi.org:resource:reference "
    keyName="definition " ... />
```

```
<tModel>
  <name>[default name]</name>
 <keyedReference
    tModelKey="uddi:uddi.org:xml:namespace"
    keyName="uddi.org:resource:type"
    keyValue="http://test.foo/xml/otherthing"/>
```

```
<tModel>
  <name>theta</name>
 <keyedReference
    tModelKey="uddi:uddi.org:xml:namespace"
    keyName=".. contained-by
    keyValue="[tModel key of something.xsd]"/>
```

Use Cases

HPE SOA Registry Foundation 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, “Find XSD” in in "Searching" on page 256
- User's Guide, “Publish XSD” in "Publishing" on page 265

- Developer's Guide, "XSD Publishing" on page 435

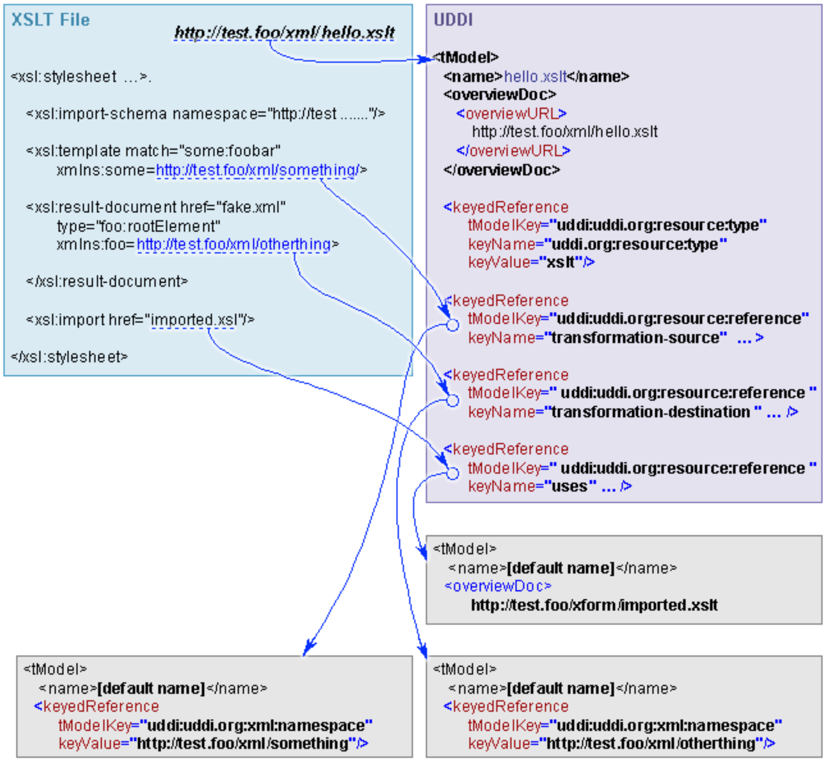
XSLT

As shown in the following figure, “XSLT TO UDDI”, 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.

XSLT TO UDDI



```
<xsl:stylesheet ...>.

   <xsl:import-schema namespace="http://test ......"/>

   <xsl:template match="some:foobar"
         xmlns:some=http://test.foo/xml/something/>

   <xsl:result-document href="fake.xml"
         type="foo:rootElement"
         xmlns:foo=http://test.foo/xml/otherthing>

   </xsl:result-document>

   <xsl:import href="imported.xsl"/>

</xsl:stylesheet>
```

```
<tModel>
  <name>hello.xslt</name>
  <overviewDoc>
    <overviewURL>
       http://test.foo/xml/hello.xslt
    </overviewURL>
  </overviewDoc>

  <keyedReference
     tModelKey="uddi:uddi.org:resource:type"
     keyName="uddi.org:resource:type"
     keyValue="xslt"/>

  <keyedReference
     tModelKey="uddi:uddi.org:resource:reference"
     keyName="transformation-source"   ... >

  <keyedReference
     tModelKey=" uddi:uddi.org:resource:reference "
     keyName="transformation-destination " ... />

  <keyedReference
     tModelKey=" uddi:uddi.org:resource:reference "
     keyName="uses" ... />
```

```
<tModel>
  <name>[default name]</name>
 <overviewDoc>
    http://test.foo/xform/imported.xslt
```

```
<tModel>
  <name>[default name]</name>
 <keyedReference
    tModelKey="uddi:uddi.org:xml:namespace"
    keyValue="http://test.foo/xml/something"/>
```

```
<tModel>
  <name>[default name]</name>
 <keyedReference
    tModelKey="uddi:uddi.org:xml:namespace"
    keyValue="http://test.foo/xml/otherthing"/>
```

Client-Side Development

Client-Side Development includes the following sections:

- "UDDI APIs" below - Describes the principles of how to use HPE SOA Registry Foundation 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" on page 393 - Advanced APIs cover the following APIs: Validation API, Taxonomy API, Category APIs, Administration Utilities API, Replication API, Statistics API, Inquiry UI API, Subscription Ext Api, and Publishing API for resources:
    - WSDL Publishing
    - XSD Publishing
- "Security APIs " on page 453 - Security APIs cover the following APIs: Account API, Group API, Permission API.
- "Registry Client" on page 471 - This section describes how to prepare your own client distribution. A client created this way allows you to access the HPE SOA Registry Foundation API through a SOAP interface.
- "Client Authentication" on page 480 - describes how to create a client that autheticates thru HTTP Basic.

UDDI APIs

UDDI (Universal Description Discovery and Integration) is a 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.

The UDDI API set is described in the following topics:

- "Principles To Use UDDI API" below
- UDDI Version 1
- "UDDI Version 2" on page 385
- "UDDI Version 3" on page 385
- "UDDI Version 3 Extension" on page 387

Principles To Use UDDI API

This section will show you how to use the HPE SOA Registry Foundation API. Examples are based on the UDDI version 3 Specification.

To use Inquiry APIs you can follow these steps. The complete code fragment is shown in "Example: FindBinding v3" on page 381.

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 HPE SOA Registry Foundation is running.

To use the publishing API, follow these steps. The complete code fragment is shown in "Example: SaveService v3" on page 382.

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: FindBinding v3

```
// //(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
// 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: SaveService v3

```
// //(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
// 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");
     }
}
```

UDDI Version 1

The UDDI version 1 Specification has provided a foundation for next versions.

Inquire

- WSDL: `REGISTRY_HOME/doc/wsdl/inquire_v1.wsdl`
- API endpoint: `http://<host name>:<port>/uddi/inquiry`
- Java API: org.systinet.uddi.client.v1.InquireSoap
- Demos: Inquiry demos v1

Publish

- WSDL: `REGISTRY_HOME/doc/wsdl/publish_v1.wsdl`
- API endpoint: `http://<host name>:<port>/uddi/publishing`
- Java API: org.systinet.uddi.client.v1.PublishSoap
- Demos: Publishing demos v1

UDDI Version 2

The UDDI version 2 Specification has introduced many improvements of existing concepts and new features like service projections.

Inquiry

- Specification: Inquiry API functions
- WSDL: `REGISTRY_HOME/doc/wsdl/inquire_v2.wsdl`
- API endpoint: `http://<host name>:<port>/uddi/inquiry`
- Java API: org.systinet.uddi.client.v2.Inquire
- Demos: Inquiry demos v2

Publish

- Specification: Publishing API Function
- WSDL: `REGISTRY_HOME/doc/wsdl/publish_v2.wsdl`
- API endpoint: `http://<host name>:<port>/uddi/publishing`
- Java API: org.systinet.uddi.client.v2.Publish
- Demos: Publishing demos v2

UDDI Version 3

The UDDI version 3 Specification 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" in the UDDI version 3 Specification
- API endpoint: `http://<host name>:<port>/uddi/inquiry`
- Java API: `org.systinet.uddi.client.v3.UDDI_Inquiry_PortType`
- Demos: Inquiry demos v3

Publication

- Specification: "Publication API set" in the UDDI version 3 Specification
- API endpoint: `http://<host name>:<port>/uddi/publishing`
- Java API: org.systinet.uddi.client.v3.UDDI_Publication_PortType
- Demos: Publishing demos v3

Security

- Specification: "Security API set" in the UDDI version 3 Specification
- API endpoint: `http://<host name>:<port>/uddi/security`
- Java API: org.systinet.uddi.client.v3.UDDI_Security_PortType

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
- API endpoint: `http://<host name>:<port>/uddi/custody`
- Java API: org.systinet.uddi.client.custody.v3.UDDI_CustodyTransfer_PortType
- 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.

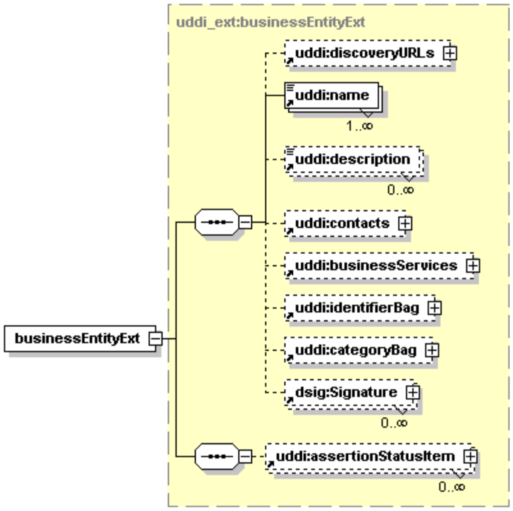
- Specification: Subscription API Set
- API endpoint: `http://<host name>:<port>/uddi/custody`
- Java API: org.systinet.uddi.client.subscription.v3.UDDI_Subscription_PortType
- Demos: Subscription Demos

UDDI Version 3 Extension

UDDI Version 3 Extensions are HPE extensions of the UDDI Version 3 Specification. 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

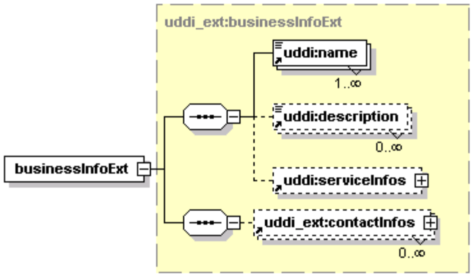


Attributes

Name	Required
`businessKey`	Optional

This structure is used by the Registry Console for performance enhancements. The structure is an extension of businessEntity, the added element is uddi:assertionStatusItem that points to the related businessEntity,

businessInfoExt

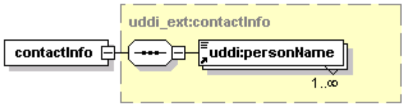


Attributes

Name	Required
`businessKey`	Optional

This structure is an extension of the `businessInfo` structure; the added element is `uddi_ext:contactInfos`.

contactInfo

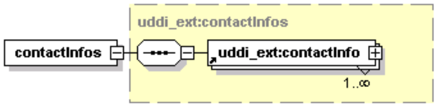


Attributes

Name	Required
`useType`	Optional

This structure represents a person name for the businessInfoExt.

contactInfos

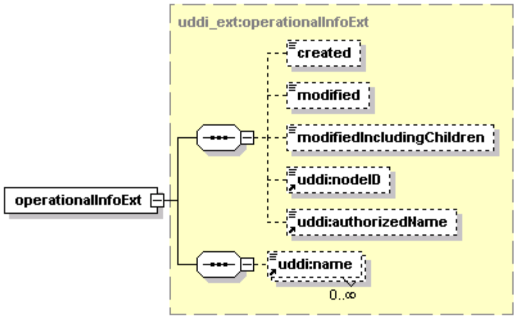


Attributes

Name	Required
`useType`	Optional

This structure holds a list of contactInfos.

operationalInfoExt

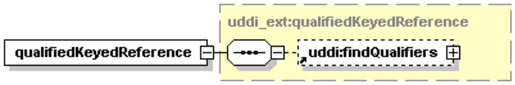


Attributes

Name	Required
`entityKey`	Required
`entityKeyV2`	Optional

This structure is an extension of the operationalInfo structure, the added element is uddi:name. The entityKeyV2 holds UDDI v2 key values.

qualifiedKeyedReference

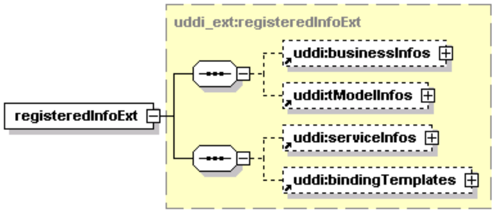


Attributes

Name	Required
`tModelKey`	Required
`keyName`	Optional
`keyValue`	Required

This structure holds findQualifiers that are used in Range Queries.

registeredInfoExt

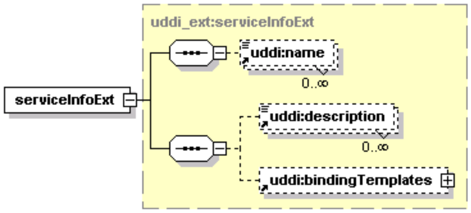


Attributes

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



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 permits vendors to define new find qualifiers. Table “Summary of Additional Find Qualifiers in HPE SOA Registry Foundation ” summarizes the additional find qualifiers in HPE SOA Registry Foundation and the find_xx operations that support them. See “Inquiry” in "UDDI Version 3" on page 385 for more information on inquiry API operations.

Each short name in the following table, links to a subsection that follows. Note that the tModel key is the short name prefixed with `uddi:systinet.com:findQualifier:`.

Summary of Additional Find Qualifiers in HPE SOA Registry Foundation

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 behavior required by the specification);
- really deletes the tModel, provided there are no dependencies on it;

See "Node" on page 341 in the Administrator's Guide for more information.

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.

Advanced APIs

Advanced APIs cover the following APIs:

- **"Validation" on the next page** - 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. Every checked taxonomy requires a Web service that implements this API.
- **"Taxonomy" on page 395** - 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.
- **"Category" on page 406** - The Systinet Category API complements the Systinet Taxonomy API. It is used to query and to manipulate Internal taxonomies in HPE SOA Registry Foundation. 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.
- **"Administration Utilities" on page 412** - The Systinet Administration Utilities API provides an interface to perform several low level administrative tasks in HPE SOA Registry Foundation.
- **"Replication" on page 417** - The Replication API is used to launch replications in HPE SOA Registry Foundation.
- **"Statistics" on page 418**- The Systinet Statistics API provides useful information about HPE SOA Registry Foundation usage.

- **"WSDL Publishing" on page 422** - HPE SOA Registry Foundation WSDL-to-UDDI mapping is compliant with OASIS's Technical Note, Using WSDL in a UDDI registry Version 2.0. 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.
- **"XSD Publishing" on page 435 - XSD2UDDI**. These API sets allow you to manipulate with resources in HPE SOA Registry Foundation. XML Schemas are supported.
- **"Inquiry UI" on page 446** - 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.

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. 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.

HPE SOA Registry Foundation is built according to the UDDI technical note Providing A Value Set For Use In UDDI Version 3. 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 HPE SOA Registry Foundation'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 .

SOAP

Specification: `REGISTRY_HOME/doc/wsdl/uddi_vs_v3.wsdl`

Java

- Java API: org.systinet.uddi.client.valueset.validation.v3.UDDI_ValueSetValidation_PortType
- Demos: Validation demos

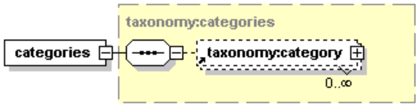
Taxonomy

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.

Data Structures

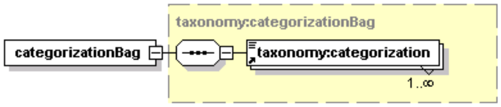
The following structures are used by the Systinet Taxonomy API:

Categories



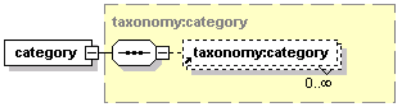
This structure is a container for zero or more `category` structures. If the taxonomy is internal, then categories are used to hold possible values of its keyedReferences.

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

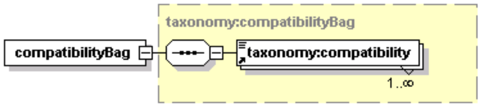


This structure corresponds to the keyedReference. It defines the keyedReference of the taxonomy in which it is used. The `keyValue` must be unique. The disabled attribute is used to mark the category as either helper or deprecated, so it cannot be used as a valid option in keyedReferences. The keyName attribute serves as a label for this category.

Attributes

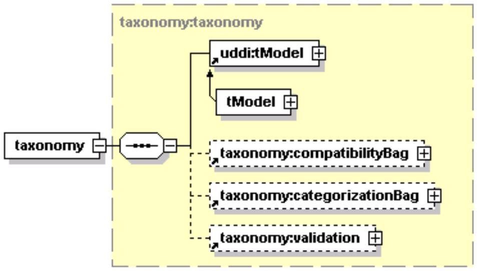
Name	Required
`keyName`	Yes
`keyValue`	Yes
`disabled`	No

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



Attributes

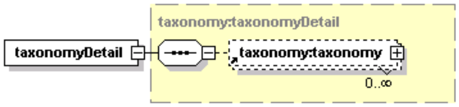
Name	Required
check	No

`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

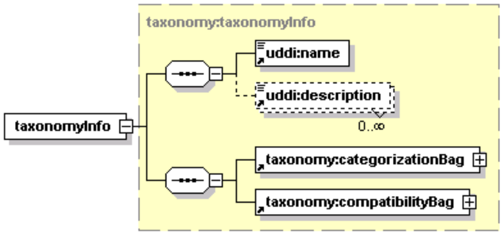


Attributes

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



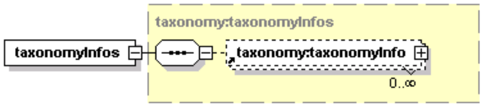
Attributes

Name	Required
`check`	Yes
`unvalidatable`	No

The taxonomyInfo is an extension of the tModelInfo structure.

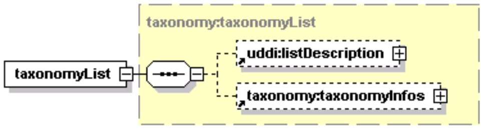
- The `check` attribute indicates whether or not the taxonomy is checked.
- The `unvalidatable` attribute is used to mark the checked taxonomy as unvalidatable. Unvalidatable taxonomies cannot be used in keyedReferences.

taxonomyInfos



This structure is a container for zero or more `taxonomyInfo` structures.

taxonomyList

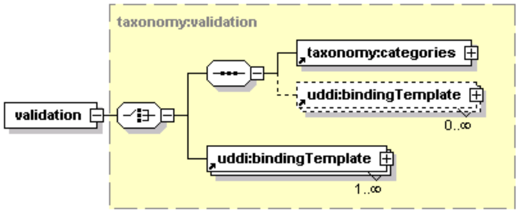


This structure serves as a container for optional `listDescription` and optional `taxonomyInfos` structures. The `truncated` attribute indicates whether the list of taxonomies is truncated.

Attributes

Name	Required
`truncated`	Yes

validation

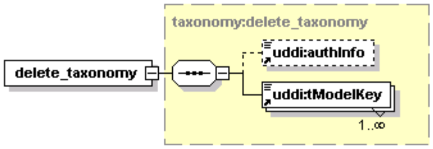


This structure is used to hold information for validating a checked taxonomy. The `categories` structure defines the list of available values for keyedReferences checked by the Internal validation service. `Binding templates` contains the valueset validation Web service endpoint.

Operations

delete_taxonomy

The delete_taxonomy API call is used to delete one or more taxonomies from HPE SOA Registry Foundation. 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.

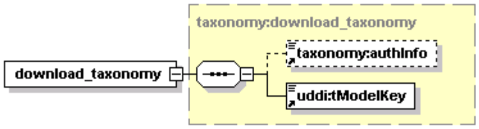
Upon successful completion, a disposition report is returned with a single success indicator.

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 HPE SOA Registry Foundation. 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

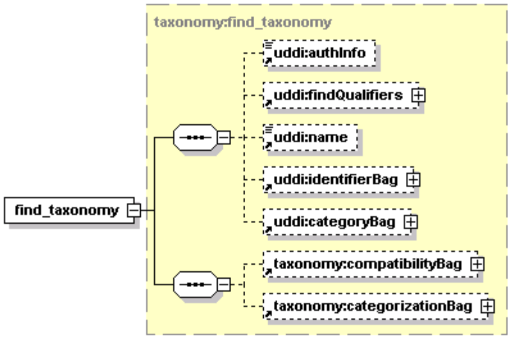
This API call returns a ResponseMessageAttachment with the selected taxonomy upon success.

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.



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.

> **Note:** 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`

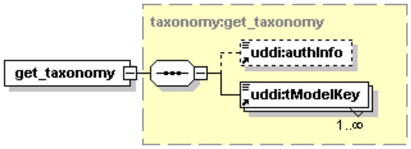
This API call returns the TaxonomyList upon success.

`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.



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.

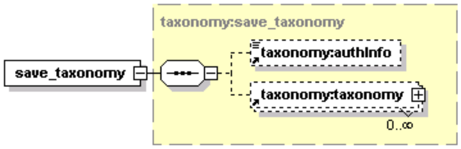
> **Note:** If the tModel of a checked taxonomy does not contain the validatedBy keyedReference, the taxonomy's `unvalidatable` attribute will be set to true and the validation structure will be missing.

`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 HPE SOA Registry Foundation.



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`

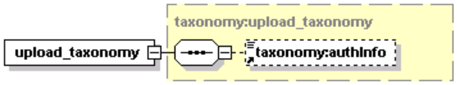
This API call returns the TaxonomyDetail on success.

`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 HPE SOA Registry Foundation. This call is stream oriented and is useful for publishing very large taxonomies.



`Permissions`

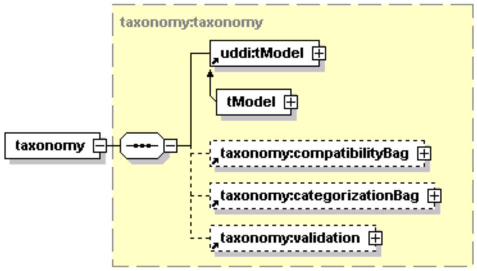
This API call requires the API manager permission named `org.systinet.uddi.client.taxonomy.v3.TaxonomyApi` and the action upload_taxonomy.

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 `REGISTRY_HOME/doc/wsdl/taxonomy.wsdl` XML Schema Definition file. For more information, see the data structure of “taxonomy”.



WSDL

You can find the WSDL specification in the file REGISTRY_HOME/doc/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.

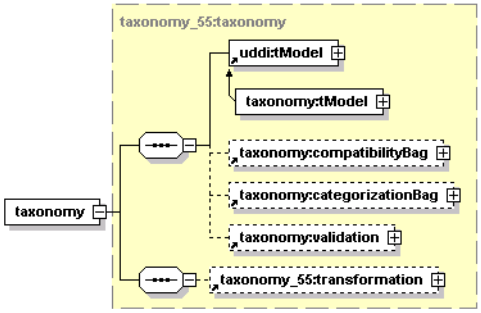
Taxonomy 5.5 Extension

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`

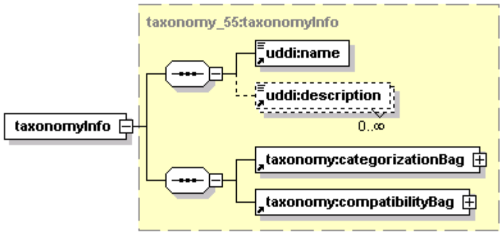

taxonomy_55:taxonomy
uddi:tModel
taxonomy:tModel
taxonomy:compatibilityBag
taxonomy:categorizationBag
taxonomy:validation
taxonomy
taxonomy_55:transformation

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`

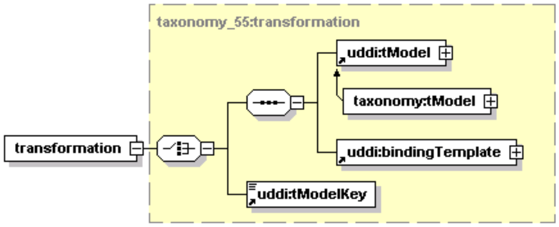


Attributes

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, see “Custom Ordinal Types” in "Taxonomy: Principles, Creation and Validation" on page 224 in the User's Guide.

- `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`.

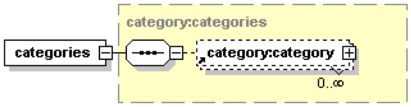
Category

The Systinet Category API complements the Systinet Taxonomy API. It is used to query and to manipulate Internal taxonomies in HPE SOA Registry Foundation. 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

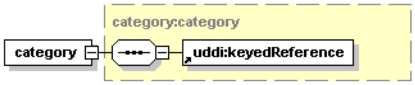
The following structures are used by the Systinet Category API:

Categories



This structure is a container for zero or more `category` elements.

category



Attributes

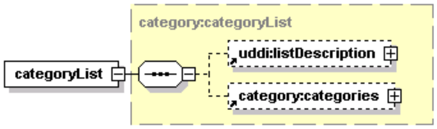
Name	Required
`disabled`	No
`leaf`	No

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



Attributes

Name	Required
`truncated`	No

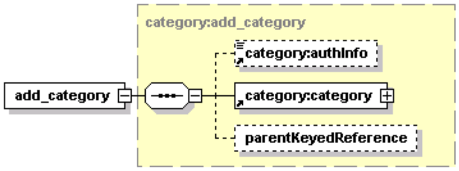
This structure serves as a container for optional `listDescription` and `categories` structures. The `truncated` attribute indicates whether a returned list of categories is truncated.

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.

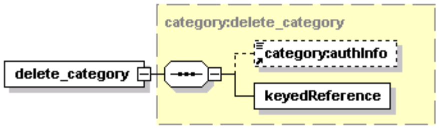
Permissions

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 HPE SOA Registry Foundation.

Syntax



Arguments

- `category:authInfo` - This optional argument is an element that contains an authentication token.
- `keyedReference` - Category to be deleted.

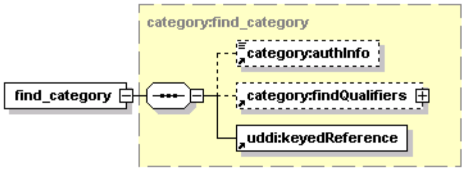
Permissions

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 HPE SOA Registry Foundation 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.

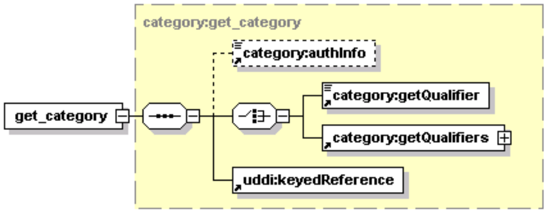
Returns

This API call returns a CategoryList upon success.

get_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.

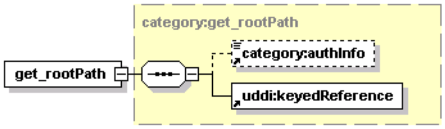
Returns

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

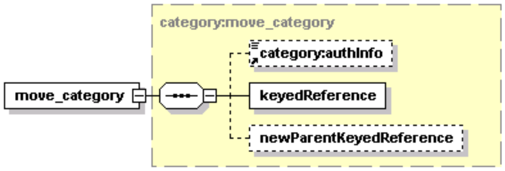
Returns

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.

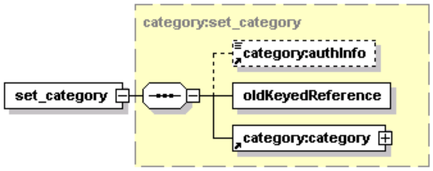
Permissions

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 HPE SOA Registry Foundation.

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 `REGISTRY_HOME/doc/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.

Administration Utilities

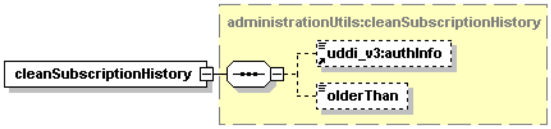
The Systinet Administration Utilities API provides an interface to perform several low level administration tasks in HPE SOA Registry Foundation.

Operations

cleanSubscriptionHistory

This utility removes subscription histories from HPE SOA Registry Foundation. 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.

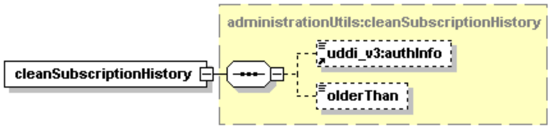
Permissions

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. HPE SOA Registry Foundation 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



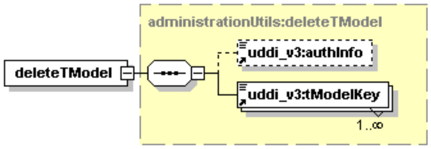
Permissions

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 HPE SOA Registry Foundation. 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.

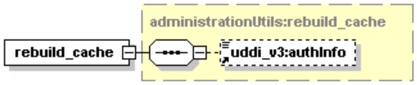
Permissions

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

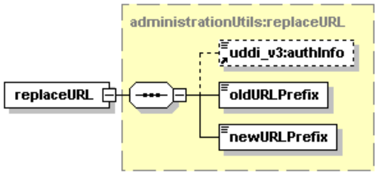
This API call requires API manager permissions for `org.systinet.uddi.admin.AdministrationUtilsApi` and for the rebuild_cache action.

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

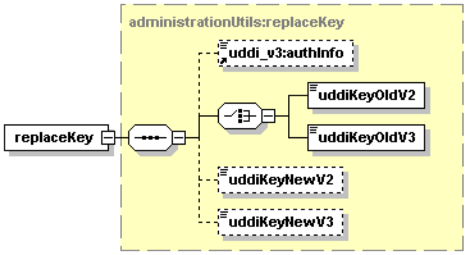
Permissions

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 HPE SOA Registry Foundation. 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.

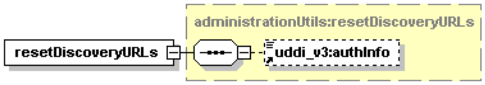
Permissions

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 HPE SOA Registry Foundation to its default value.

Syntax



Arguments

`uddi_v3:authInfo` - This optional argument is an element that contains an authentication token.

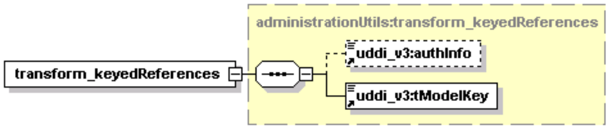
Permissions

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, “Taxonomy: Principles, Creation and Validation”.

Syntax


administrationUtils:transform_keyedReferences
transform_keyedReferences
uddi_v3:authInfo
uddi_v3:tModelKey

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 `REGISTRY_HOME/doc/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 org.systinet.uddi.admin.AdministrationUtilsApi for more information.

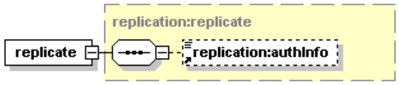
Replication

The Replication API is used to launch replications in HPE SOA Registry Foundation.

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 `REGISTRY_HOME/doc/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`.

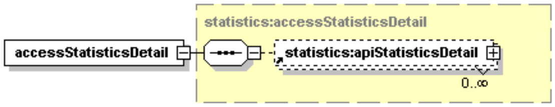
Statistics

The Systinet Statistics API provides useful information about HPE SOA Registry Foundation usage.

Data Structures

The following structures are used by the Systinet Statistics API:

accessStatisticsDetail

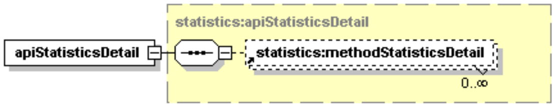

statistics:accessStatisticsDetail
accessStatisticsDetail
statistics:apiStatisticsDetail
0..∞

Attributes

Name	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


statistics:apiStatisticsDetail
apiStatisticsDetail
statistics:methodStatisticsDetail
0..∞

Attributes

Name	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 HPE SOA Registry Foundation 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

Attributes

Name	Required
methodName	Yes
requestCount	Yes
exceptionCount	Yes
lastCall	Yes

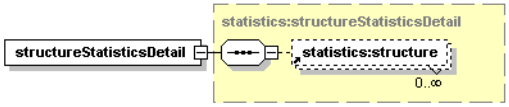
This element contains information about usage of the method specified in the attribute `methodName`.

The `requestCount` attribute holds a number indicating how many times this method has been called since its last reset or since HPE SOA Registry Foundation 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


statistics:structureStatisticsDetail
structureStatisticsDetail
statistics:structure
0..∞

This structure serves as a container for the `structure` element.

Structure

Attributes

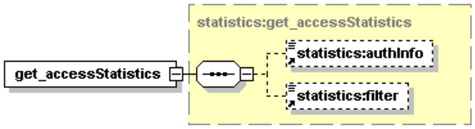
Name	Required
`name`	Yes
`count`	Yes

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 HPE SOA Registry Foundation. 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`

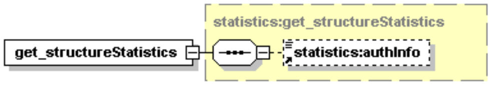
Upon successful completion, an accessStatisticsDetail structure is returned.

`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 HPE SOA Registry Foundation.



`Arguments`

`statistics:authInfo` - This optional argument is an element that contains an authentication token.

`Returns`

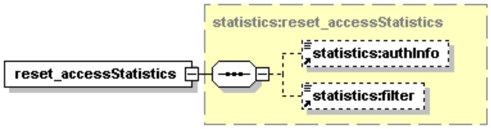
Upon successful completion, an structureStatisticsDetail structure is returned.

`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 HPE SOA Registry Foundation. 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 `REGISTRY_HOME/doc/wsdl/statistics.wsdl` .

API Endpoint

You can find the Statistics API endpoint at `http://<host name>:<port>/uddi/statistics`.

Java

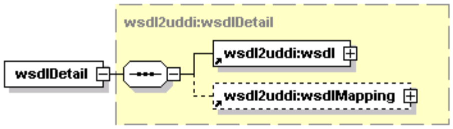
Systinet Java API is generated directly from WSDL. You are encouraged to browse org.systinet.uddi.statistics.StatisticsApi.

WSDL Publishing

HPE SOA Registry Foundation WSDL-to-UDDI mapping is compliant with OASIS's Technical Note, Using WSDL in a UDDI registry Version 2.0. 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

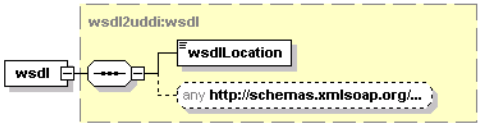


wsdlDetail completes information about the WSDL to be mapped.

`Arguments`

- `wsdl2uddi:wsdl` - Contains URI or physical location of mapped WSDL.
- `wsdl2uddi:wsdlMapping` - Describes wsdl:types to be mapped.

wsdl

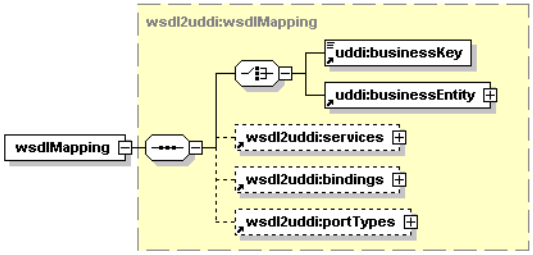


WSDL contains information about location of a mapped WSDL.

`Arguments`

- `wsdlLocation` - The URI or physical location of a mapped WSDL.
- `any` - Used to make extensible documents (see XML schema). It is generally used as the DOM pattern of a mapped WSDL.

wsdlMapping



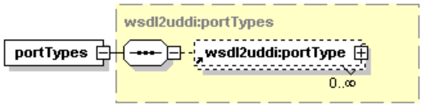
WsdlMapping describes the `wsdl:types` to be mapped. It is used to alter the default behavior of mapping the specified WSDL. In contained structures, it is possible to describe each mapped `wsdl:type` correctly. This is to ensure exact mapping and prevent duplication of data in the registry.

`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.

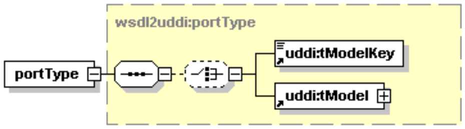
> **Note:** Note that uddi:businessKey and uddi:businessEntity are mutually exclusive.

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 publishingMethod selected by the user) with data from `wsdl:portType`.

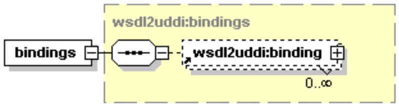
> **Note:** Note that uddi:tModelKey and uddi:tModel are mutually exclusive.

Attributes

Name	Required
`name`	optional
`namespace`	optional
`publishingMethod`	optional

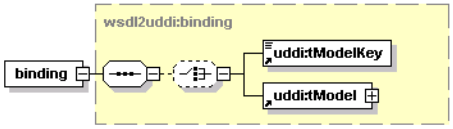
These attributes describe the `wsdl:portType` of the appropriate WSDL. Name and namespace represent the `wsdl:portType` QName. `publishingMethod` represents an enumeration of available mapping use cases. It can be set to `rewrite`, `create`, `reuse`, or `ignore`. The default publishingMethod is `reuse`.

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 publishingMethod selected by the user) with data from `wsdl:binding`.

> **Note:** Note that `uddi:tModelKey` and `uddi:tModel` are mutually exclusive.

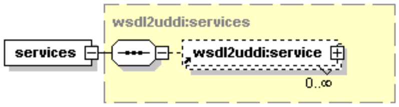
Attributes

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.

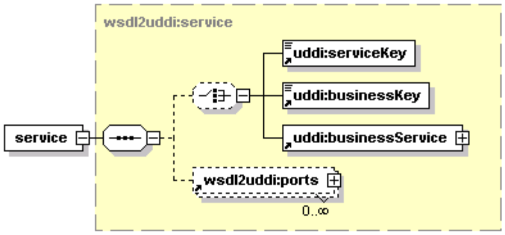
`publishingMethod` represents an enumeration of the available mapping use cases. It can be set to `rewrite`, `create`, `reuse`, or `ignore`. The default publishingMethod is `reuse`.

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.

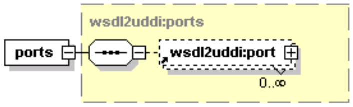
Attributes

Name	Required
`name`	optional
`namespace`	optional
`publishingMethod`	optional

These attributes describe the `wsdl:service` from an appropriate WSDL. Name and namespace represents the `wsdl:service` QName.

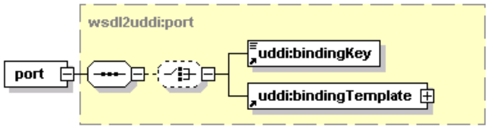
`publishingMethod` represents an enumeration of available mapping use cases. It can be set to `rewrite`, `create`, `reuse`, or `ignore`. The default publishingMethod is `reuse`.

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`.

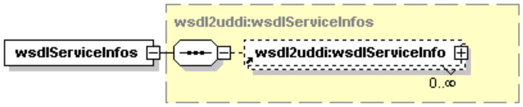
> **Note:** Note that uddi:bindingKey and uddi:bindingTemplate are mutually exclusive.

Attributes

Name	Required
`name`	optional
`publishingMethod`	optional

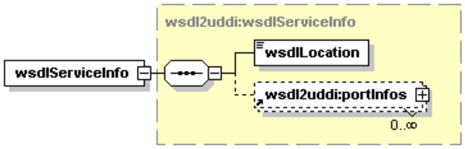
These attributes describe the `wsdl:port` from an appropriate WSDL.Name represents the `wsdl:port` name. `publishingMethod` represents an enumeration of available mapping use cases. It can be set to `rewrite`, `create`, or `reuse`. The default publishingMethod is `reuse`.

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`.

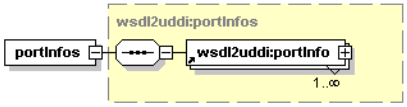
Attributes

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.

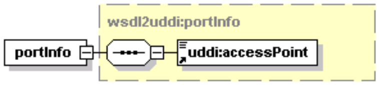
The `serviceKey` represents the `uddi:businessService` on which the `wsdl:service` is mapped.

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.

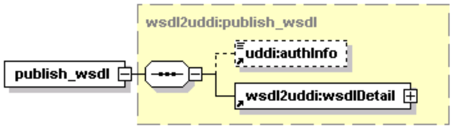
Attributes

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, see UDDI v3 - publishing API).

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.

  > **Note:** 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`

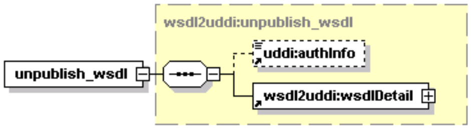
- `uddi:authInfo` - This required argument is the string representation of the `uddi:authToken`.
- `wsdl2uddi:wsdlDetail` - Completes WSDL location and user-defined WSDL mapping rules. For more information, see `wsdl2uddi:wsdlDetaill`.

  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, see `wsdl2uddi:publishingMethod`.

`Returns`

`wsdl2uddi:wsdlDetail` - Contains detailed information about how the individual `wsdl:types` are published. For more information, 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, see UDDI v3 - publishing API.

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.)

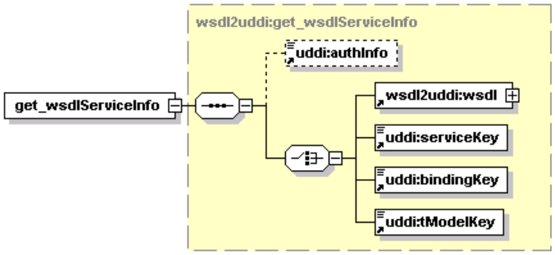
`Arguments`

- `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, 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, see UDDI V3 - UDDI-inquiry API.

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.

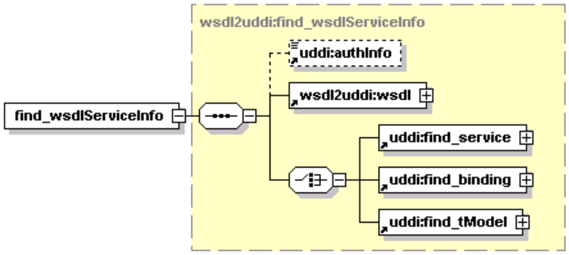
  Note that uddi:bindingTemplates in v2 do not contain uddi:categoryBag. Even though the found uddi:bindingTemplate must contain uddi:tModels compliant with "WSDL Type Category System" in its uddi:tModelInstanceDetails.
- uddi:tModelKey - the uddi:tModelKey of the uddi:tModel existing on the target registry. 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, see UDDI V3-inquiry API).

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`

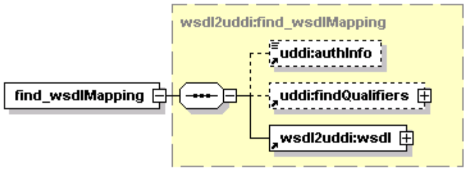
- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.
- `wsdl2uddi:wsdl` - required argument used to discover wsdl2uddi:wsdlServiceInfos. This argument ensures that only the `uddi:businessService` corresponding exactly to the `wsdl:service` from that WSDL will be returned. For more information, see `wsdl2uddi:wsdl`.
- `uddi:find_service` - Argument used for a more detailed description of search criteria. For more information, see `uddi:find_service`. 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`. 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`. 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, 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 UDDI Version 3 - Find Qualifiers
- wsdl2uddi:wsdl

Returns

This operation returns wsdl2uddi:wsdlMapping.

WSDL

REGISTRY_HOME/doc/wsdl/wsdl2uddi_v2.wsdl

REGISTRY_HOME/doc/wsdl/wsdl2uddi_v3.wsdl

API Endpoint

You can find the WSDL2UDDI API endpoint at **http://<host name>:<port>/uddi/wsdl2uddi**.

Java

org.systinet.uddi.client.wsdl2uddi.v3.Wsdl2uddiApi

"WSDL2UDDI v2" on page 623 demos

"WSDL2UDDI v3" on page 630 demos

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.

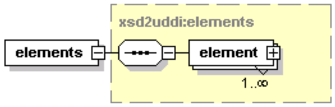
The mapping of XML Schema Document information to UDDI covers:

- 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

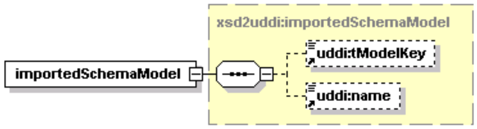


This structure represents elements declared by the published XML Schema Document.

`Arguments`

`element` - This argument represents an element declared by the published XML Schema Document.

importedSchemaModel

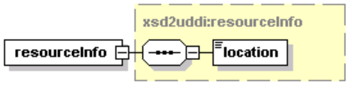


This structure contains the basics of the imported XML Schema tModel.

Arguments

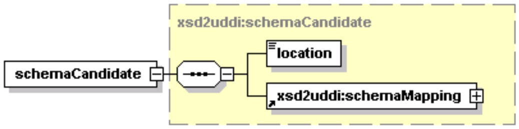
- uddi:tModelKey - The key of the tModel of the schema of the imported XML namespace.
- uddi:name - The name of that schema's tModel.

resourceInfo



This structure describes the location of the XML Schema Document.

schemaCandidate

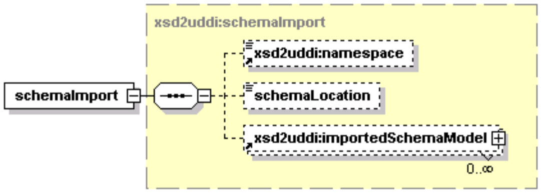


This structure holds possible mappings of how the XML Schema Document can be published.

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.

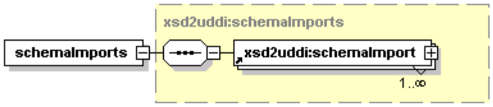
Arguments

- 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.

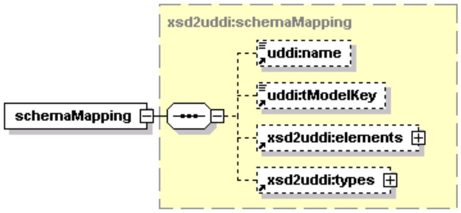
- `xsd2uddi:importedSchemaModel` - The tModel information of the candidates for this import.

schemaImports



This structure describes a list of `xs:imports` in the schema.

schemaMapping

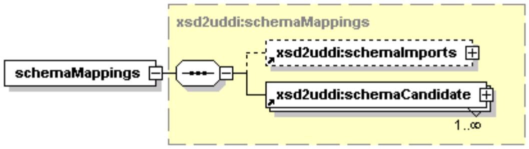


This structure describes a mapping of the XSD contents to an individual XSD tModel and its contents.

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

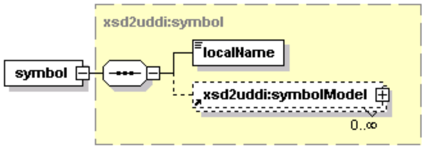


This structure describes a mapping from the contents of a XML Schema Document to UDDI entities. There are two parts. The first part describes possible matches for `xs:imports` specified by the XML Schema Document; the second, individual candidates that may match the XML Schema Document contents. The 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

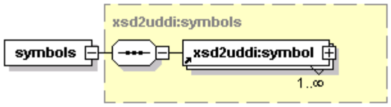


This structure holds mapping of an individual symbol (XSD element and type) to the registry.

`Arguments`

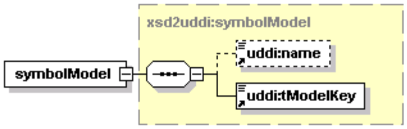
- `localName` - Local name of the mapped symbol.
- `xsd2uddi:symbolModel` - The basics of the tModel that represents the symbol.

symbols



A common structure for mapping types and elements.

symbolModel

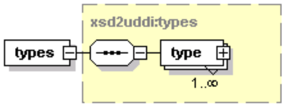


Basic information about a tModel that represents an element or a type declared by the XML Schema Document

`Arguments`

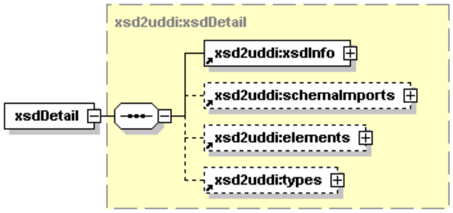
- `uddi:name` - Name of the symbol's tModel. This argument is optional when publishing a XML Schema Document; it is always filled in API responses.
- `uddi:tModelKey` - tModelKey of the symbol's model.

types



Mapping of types declared by the XML Schema Document being mapped

xsdDetail



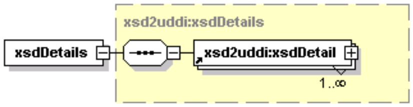
The structure provides detailed information about a specific XML Schema Document, its contents and its references.

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

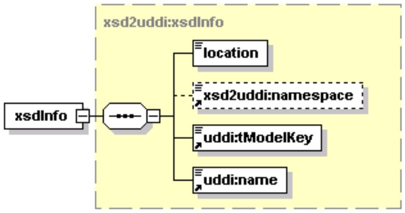
`xsd2uddi:types` - List of types in the schema

xsdDetails



Details of the XSD

xsdInfo

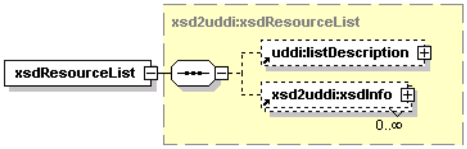


This structure holds general information about the XML Schema Document.

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



Attributes

Name	Required
`truncated`	optional

This structure holds a list of XSDs, returned from a find_xsd call.

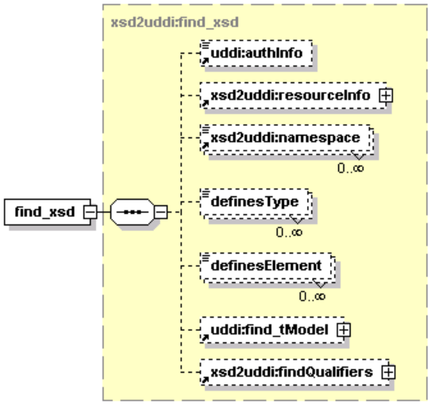
`Arguments`

- `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.

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 Version 3 - uddi:find_tModel. These criteria are combined with the other

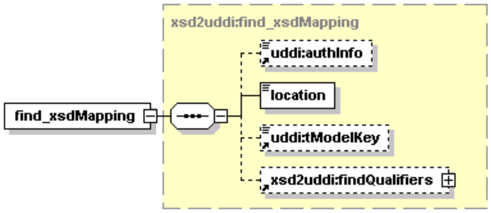
criteria specified by the find_xsd structure. In the case of a conflict, the criteria in find_xsd take precedence.

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.

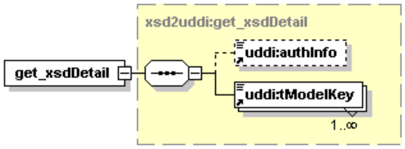
Returns

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.

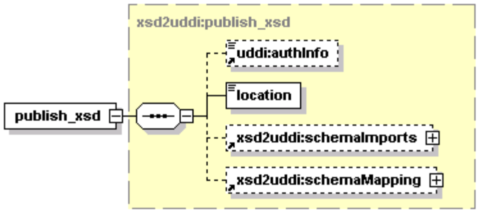
Returns

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



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 `publishingMethod`, `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`

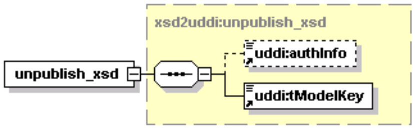
- `uddi:authInfo` - (Optional) - authentication
- `location` - XSD identification (location).
- `xsd2uddi:schemaImports` - Mapping for referenced (imported) XML Schema Documents
- `xsd2uddi:schemaMapping` - (Optional) customized mapping for the schema contents and references
- `importPolicy` - attribute 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

`REGISTRY_HOME/doc/wsdl/xsd2uddi_v3.wsdl`

API Endpoint

You can find the XSD2UDDI API endpoint at `http://<host name>:<port>/uddi/xsd2uddi`.

Java

org.systinet.uddi.client.xsd2uddi.v3.Xsd2uddiApi

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.

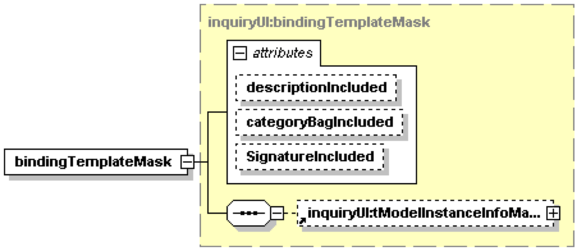
Data Structures

The following structures are used by the Systinet Inquiry UI API:

- "bindingTemplateMask" on the next page
- "businessEntityMask" on the next page
- "businessServiceMask" on page 449
- "contactMask" on page 449
- "entityDetail" on page 450

- "entityDetail" on page 450
- "tModelInstanceInfoMask" on page 451
- "tModelMask" on page 451

bindingTemplateMask

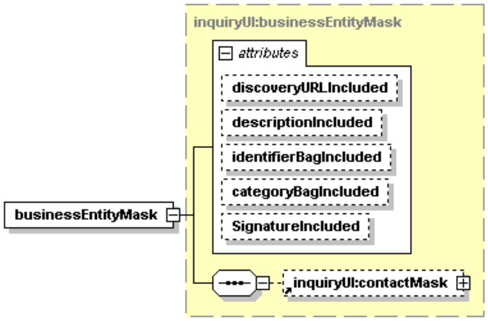


Attributes

Name	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



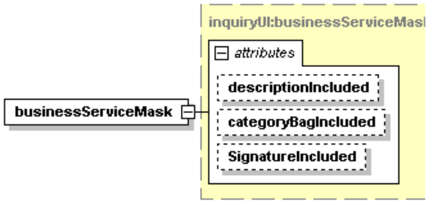
Attributes

Name	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

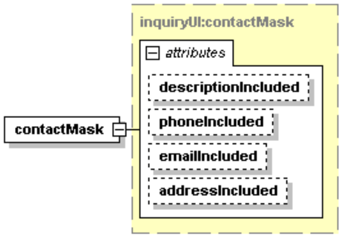


Attributes

Name	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

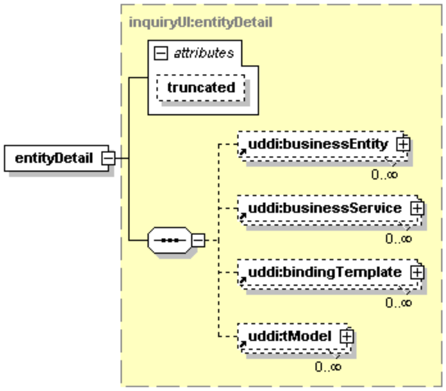


The `contactMask` structure specifies the submask of the business entity mask of the outputFormat. Optional attributes define which elements will be returned in the entityDetail.

Attributes

Name	Required
`descriptionIncluded`	No
`phoneIncluded`	No
`emailIncluded`	No
`addressIncluded`	No

entityDetail

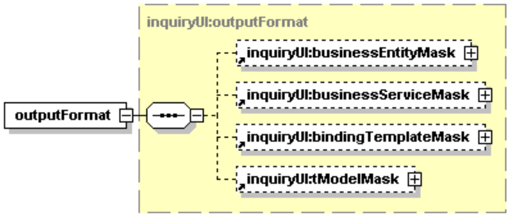


The `entityDetail` structure is returned by the get_entityDetail operation. The attribute truncated indicates a truncated result list.

Attributes

Name	Required
`uddi:truncated`	No

outputFormat

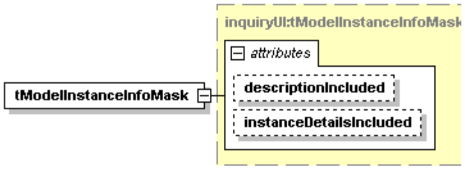

inquiryUI:outputFormat
outputFormat
inquiryUI:businessEntityMask
inquiryUI:businessServiceMask
inquiryUI:bindingTemplateMask
inquiryUI:tModelMask

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

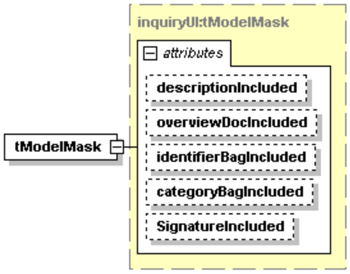


The `tModelInstanceInfoMask` structure specifies the mask of the tModel instance info of the outputFormat. Optional attributes define which elements will be returned in the entityDetail.

Attributes

Name	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.

Attributes

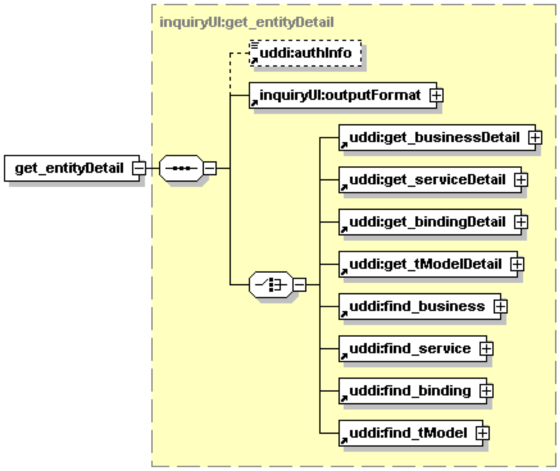
Name	Required
`descriptionIncluded`	No

`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 `REGISTRY_HOME/doc/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.

Security APIs

Security APIs cover the following APIs:

- "Accounts" below- Systinet Account API is used to query and manage user accounts in HPE SOA Registry Foundation.
- "Group" on page 460 - Systinet Group API is used to query and manage user groups in HPE SOA Registry Foundation.
- "Permission" on page 467- Systinet Permission API is used to query and manage permissions in HPE SOA Registry Foundation.

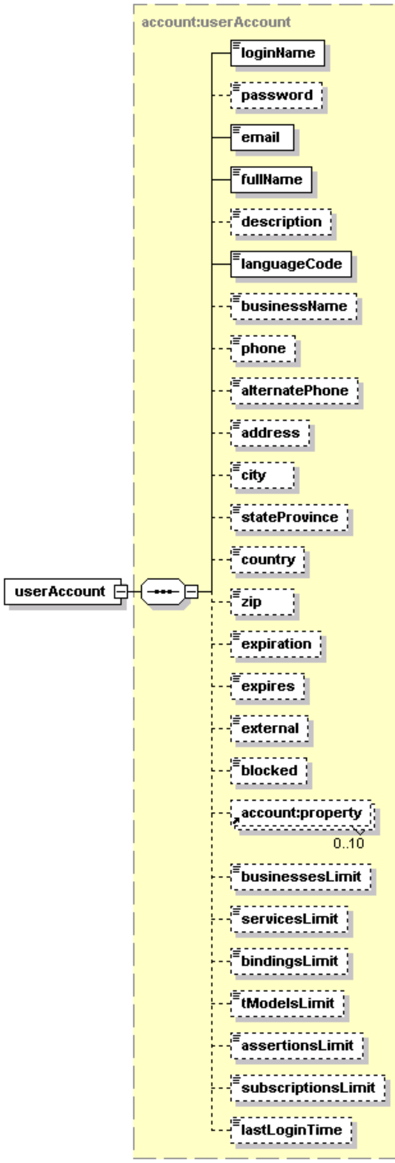
Accounts

Systinet Account API is used to query and manage user accounts in HPE SOA Registry Foundation.

Data Structures

The following structures are used by the Systinet Account API:

userAccount



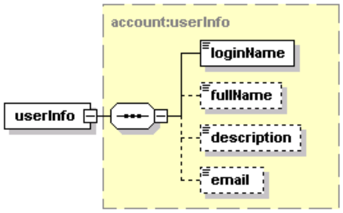
The `userAccount` element is container that holds the attributes of a user account in the HPE SOA Registry Foundation. The required elements are:

- `loginName`
- `email`
- `fullName`
- `languageCode`

All other elements are optional.

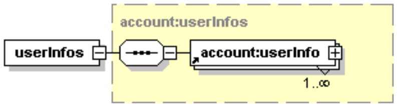
Element	Description
loginName	contains the login name of the user account
password	contains the password used to authorize the user
email	holds the user's email address
fullName	holds the user's full name
description	use for describing the user or the user's role
languageCode	the language the user speaks
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



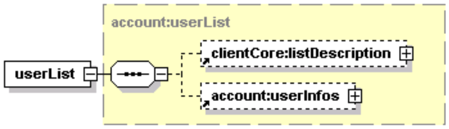
This element serves as a container for short information about single userAccount. It contains the required element loginName, and the optional elements `fullName`, `description`, and `email`.

userInfos



This element holds one or more `userInfo` elements.

userList



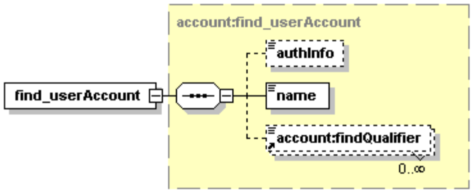
This element contains optional `listDescription` and `userInfos` elements.

Operations

find_userAccount

The find_userAccount API call is used to find user accounts in HPE SOA Registry Foundation 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.

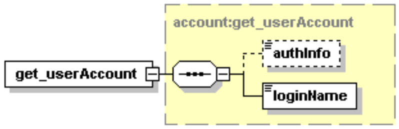
Permissions

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.

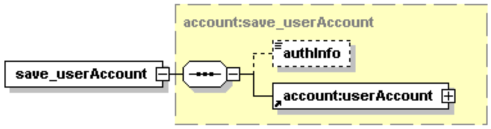
Permissions

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 HPE SOA Registry Foundation. Whether public registration is allowed or not depends on the HPE SOA Registry Foundation 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

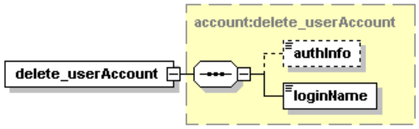
This API call returns userAccount upon success.

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

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.

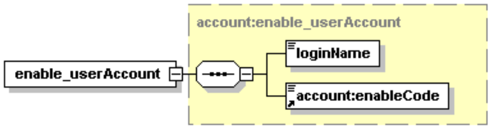
Permissions

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 HPE SOA Registry Foundation.

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 `REGISTRY_HOME/doc/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.

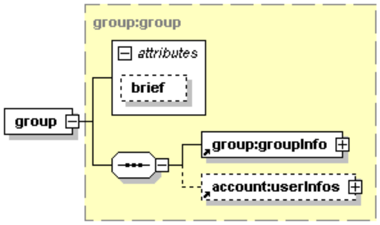
Group

Systinet Group API is used to query and manage user groups in HPE SOA Registry Foundation.

Data Structures

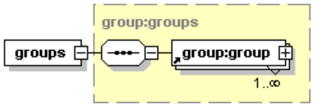
The following structures are used by the Systinet Group API:

group



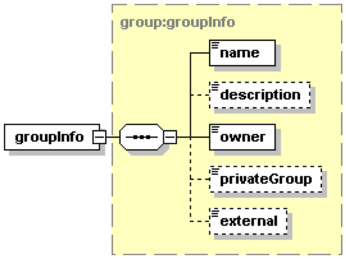
This element serves as a container for `groupInfo` and `userInfos` structures.

groups



This element serves as a container for one or more `group` structures.

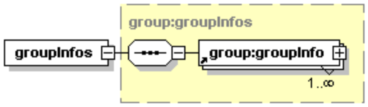
groupInfo



This element contains information about one user group:

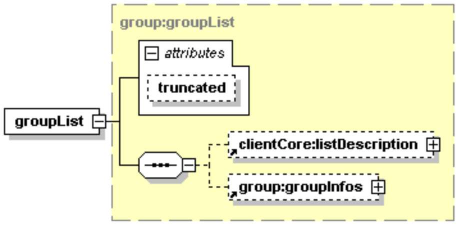
- The required `name` element holds the name of the group.
- The optional `description` element is used to describe group and its usage.
- The `owner` element contains the loginName of the user who created this group.
- The `privateGroup` element indicates whether the group is public or private.
- The `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



Attributes

Name	Required
`truncated`	No

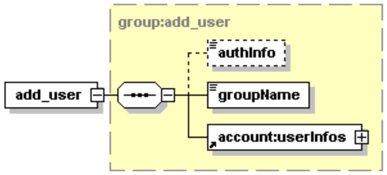
This structure server as a container for optional `listDescription` and optional `groupInfos` structures. The truncated attribute indicates whether the list of `groupInfos` is truncated.

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.

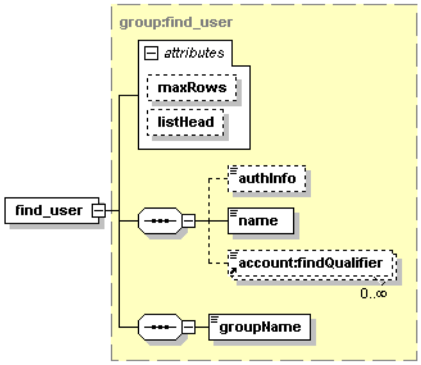
Permissions

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.

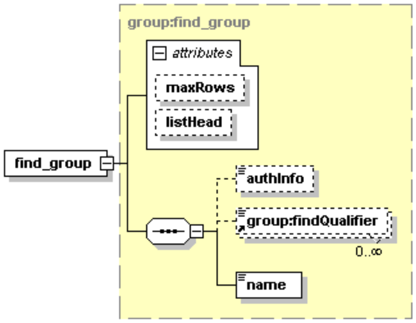
Returns

Upon successful completion, the UserList structure is returned.

find_group

The find_group API call is used to search groups in HPE SOA Registry Foundation.

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 findMyGroups 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.

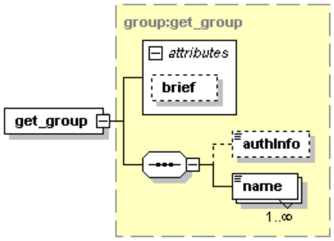
Permissions

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 HPE SOA Registry Foundation.

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.

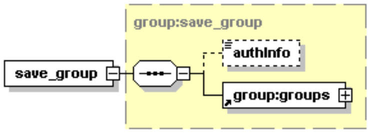
Permissions

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 HPE SOA Registry Foundation.

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.

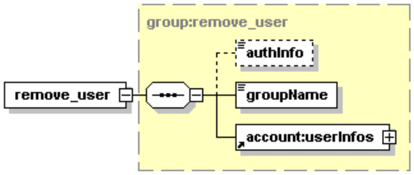
Permissions

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

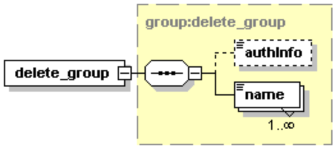
Permissions

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 HPE SOA Registry Foundation.

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.

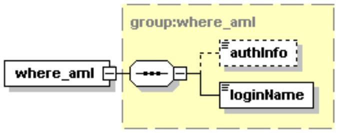
Permissions

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 `REGISTRY_HOME/doc/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.

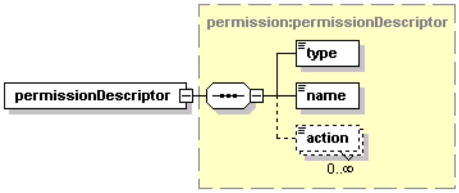
Permission

The Systinet Permission API is used to query and manage permissions in HPE SOA Registry Foundation.

Data Structures

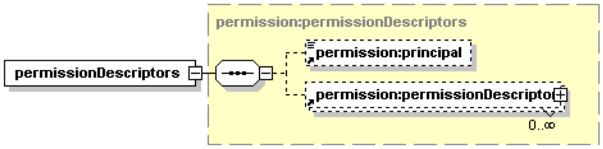
The following structures are used by the Systinet Permission API:

permissionDescriptor



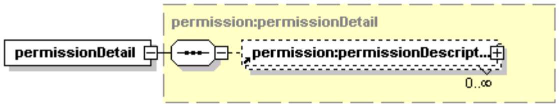
This structure serves as a container for one `permission` and its actions. The `type` element contains the type of the permission. The `name` element contains the permission's name. Optional `action` elements are used to provide finer granularity to the permission and contain individual actions of this permission.

permissionDescriptors


permissionDescriptors
permission:permissionDescriptors
permission:principal
permission:permissionDescripto
0..∞

This structure holds an optional `principal` element and zero or more `permissionDescriptor` structures.

permissionDetail

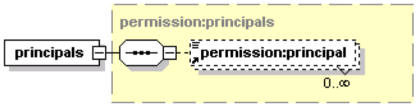


This structure is a container for zero or more `permissionDescriptors` structures.

principal

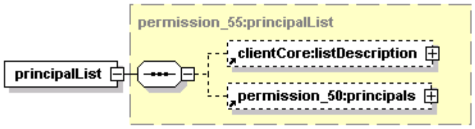
This element contains the optional attribute `principalType`, which may be assigned to a user or group. The element's text contains the loginName of the user, or the group name, depending on the principalType value.

principals



This structure serves as a container for zero or more `principal` elements.

principalList



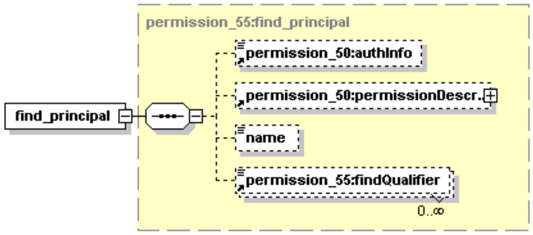
This structure serves as a list principals returned from the operation find_principal.

Operations

find_principal

This operation is used to find principals, it replaces the deprecared operation who_hasPermission .

Syntax


permission_55:find_principal
find_principal
permission_50:authInfo
permission_50:permissionDescr.
name
permission_55:findQualifier
0..∞

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.

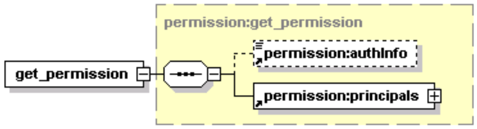
Permissions

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 HPE SOA Registry Foundation, 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.

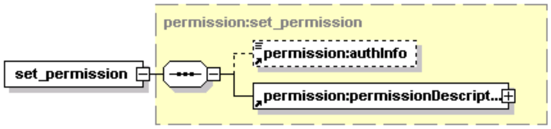
Permissions

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 HPE SOA Registry Foundation. 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

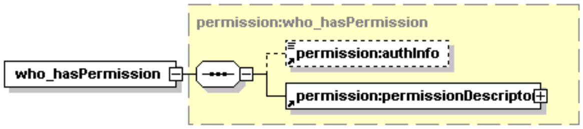
This API call requires API manager permission for `org.systinet.uddi.permission.PermissionApi` and the action set_permission.

who_hasPermission

> **Note:** 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 `REGISTRY_HOME/doc/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.

Registry Client

The ;following topics in this section describes how to prepare your own client distribution. A client created this way allows you to access the HPE SOA Registry Foundation API through a SOAP interface.

- "Client Package" on the next page
- "JARs on the Client Classpath" on page 473

Client Package

> **Note:** `CLIENT_HOME` refers to the directory in which the HPE SOA Registry Foundation Client distribution will be created.
>
> `REGISTRY_HOME` refers to the directory in which HPE SOA Registry Foundation is installed.

To create a client application distribution follow these steps:

1. Make sure HPE SOA Registry Foundation 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
   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
xsd2uddi_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 HPE SOA Registry Foundation, you must import the certificate file into `clientconf.xml` using the `PStoreTool`. This file contains the certificate of the HPE SOA Registry Foundation installation by default.

   > **Note:** 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.

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.

HPE SOA Registry Foundation Runtime

To enable the HPE SOA Registry Foundation 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, see “UDDI Version 1” in "UDDI APIs" on page 378.

```
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, see “UDDI Version 2” in "UDDI APIs" on page 378.

```
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 “UDDI Version 3” in "UDDI APIs" on page 378.

```
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, see "Accounts" on page 453.

```
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, see "Administration Utilities" on page 412.

```
admin_utils_client.jar
uddiclient_api_v3.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, see "Category" on page 406

```
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, see "Group" on page 460.

```
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, see "Permission" on page 467.

```
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, see "Replication" on page 417.

```
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, see "Statistics" on page 418.

```
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, see "Taxonomy" on page 395.

```
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, see “Custody” in "UDDI Version 3" in "UDDI APIs" on page 378.

```
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, see “Subscription” in "UDDI Version 3" in "UDDI APIs" on page 378.

```
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, see “Subscription” in "UDDI Version 3" in "UDDI APIs" on page 378.

```
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, see "Validation" on page 394.

```
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, see "Validation" on page 394.

```
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, see "Validation" on page 394.

```
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, see"WSDL Publishing" on page 422.

```
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, see"WSDL Publishing" on page 422.

```
wsdl2uddi_client_v3.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

Resources publishing (XSD) Client

To enable the client package, add these .jar files to the classpath.

```
uddiclient_api_v3.jar
uddiclient_core.jar
xsd2uddi_client_v3.jar
```

Classpath Examples

To run your HPE SOA Registry Foundation 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
```

Client Authentication

By default, all exposed registry APIs use the UDDI authentication scheme, where an authentication token is passed with every call to identify a remote user. This is shown in registry demos such as “Publishing v3” in "UDDI v3" on page 557. The UDDI authentication scheme can be replaced.

In this section, we will show you an example client that publishes a new business entity using HTTP-Basic or SSL client authentication.

Example Client

For simplicity, the example client uses a SOAP stack provided with HPE SOA Registry Foundation. You can use a SOAP stack of your choice to communicate with the registry.

ExampleClient.java

```
// //(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
// Use is subject to license terms.

import org.systinet.uddi.client.v3.UDDIPublishStub;
import org.systinet.uddi.client.v3.UDDI_Publication_PortType;
import org.systinet.uddi.client.v3.struct.*;

public class ExampleClient {
     public static void main(String[] args) {
          String registryBaseUrl = System.getProperty
("registry.base.url","http://localhost:8080");
          String urlPublishing = registryBaseUrl+ "/uddi/publishing";
          System.out.print("Using publishing URL "+urlPublishing + " .");

          try {
              UDDI_Publication_PortType publish = UDDIPublishStub.getInstance
(urlPublishing);
              System.out.println(publish.save_business(new Save_business
                  (new BusinessEntityArrayList(new BusinessEntity(new NameArrayList
                       (new Name("Created by Client Authentication Example")))))));
              System.out.println(" done");
          } catch (Exception e) {
              e.printStackTrace();
          }
     }
}
```

The client is created as follows:

1. Create the directory `CLIENT_HOME`.
2. Create a client class in the `CLIENT_HOME` directory. The example client is shown in the above example, “ExampleClient.java ”. It has no security calls or structures internally. Client-side security will be configured later using properties supplied to the java command that runs the client.
3. Create the `lib` subdirectory of CLIENT_HOME. Copy the jar files required for compilation and client execution to this directory. All the jars are in the HPE SOA Registry Foundation installation directory. They are:
    - `lib/activation.jar`
    - `lib/builtin_serialization.jar`
    - `lib/core_services_client.jar`
    - `lib/jaxm.jar`
    - `lib/jaxrpc.jar`
    - `lib/jetty.jar`
    - `lib/log4j.jar`
    - `lib/saaj.jar`
    - `lib/security-ng.jar`
    - `lib/security2-ng.jar`
    - `lib/security_providers_client.jar`
    - `lib/wasp.jar`
    - `lib/wsdl_api.jar`
    - `lib/xalan.jar`
    - `lib/xercesImpl.jar`
    - `lib/xml-apis.jar`
    - `dist/uddiclient_core.jar`
    - `dist/uddiclient_api_ v3.jar`
4. 4.Create the `conf` subdirectory of CLIENT_HOME. Copy configuration files required to run the client to this directory. These files are are also in the HPE SOA Registry Foundation installation

directory:

- `conf/clientconf.xml`
- `conf/package12.xml`
- `conf/package13.xml`
- `conf/jaas.config`

5. Compile the example client class using a CLASSPATH that includes all jar files in the `lib` subdirectory of `CLIENT_HOME`

Before running the client, configure registry for a particular authentication scheme, as explained in "HTTP Basic" on page 165 or "SSL Client authentication" on page 169. If you want to configure a deployed registry for SSL client authentication, follow instructions given in "J2EE Server Authentication" on page 172.

To run the client:

1. Use a classpath that includes all jar files from the `CLIENT_HOME/lib` directory, and the directory containing the compiled example class.
2. Add the following property definitions to the `java` command line:
   - `-Dwasp.location=CLIENT_HOME`
   - `-Djava.security.auth.login.config=CLIENT_HOME/conf/jaas.config`
3. To run the client with HTTP Basic authentication also add the following:
   - `-Dwasp.username=USERNAME`
   - `-Dwasp.password=PASSWORD`
   - `-Dwasp.securityMechanism=HttpBasic`
   - `-Dregistry.base.url=http://HOST:PORT/CONTEXT`

   Use the credentials of a registered user instead of `USERNAME` and `PASSWORD`. To register a new user, start with the main page of registry console. See "Registry Console" on page 210 for details. You may also use the demo user `demo_john` with password `demo_john` if you imported demo data during installation.

   The base URL of registry is specified using the `registry.base.url` property as shown in Example 18, “ExampleClient.java ”. Replace `HOST`,`PORT` and `CONTEXT` to match your registry deployment; for example `http://pc1.mycomp.com:8080`.
4. 4.To run the client with SSL client authentication add the following:

- `-Dwasp.username=USERNAME`
- `-Dwasp.password=PASSWORD`
- `-Dwasp.securityMechanism=SSL`
- `-Dregistry.base.url=https://HOST:PORT/CONTEXT`

Unlike HTTP Basic authentication, `USERNAME` and `PASSWORD` are used to obtain the client identity from a local protected store. You have to import the client identity using instructions provided in “SSL Tool”. The protected store of the example client is in the file `CLIENT_HOME/conf/clientconf.xml`. You also have to import a server certificate (or the certificate of a certification authority that issued the server certificate) in the same protected store using instructions provided in “PStore Tool”.

Use an alias in the protected store instead of `USERNAME`. `PASSWORD` stands for the password that is used to protect the private key stored under that alias.

The base URL of registry is specified using the `registry.base.url` System property as shown in the “ExampleClient.java ” example. Replace `HOST`,`PORT` and `CONTEXT` to match your registry deployment; for example `https://pc1.mycomp.com:8443`.

Server-Side Development

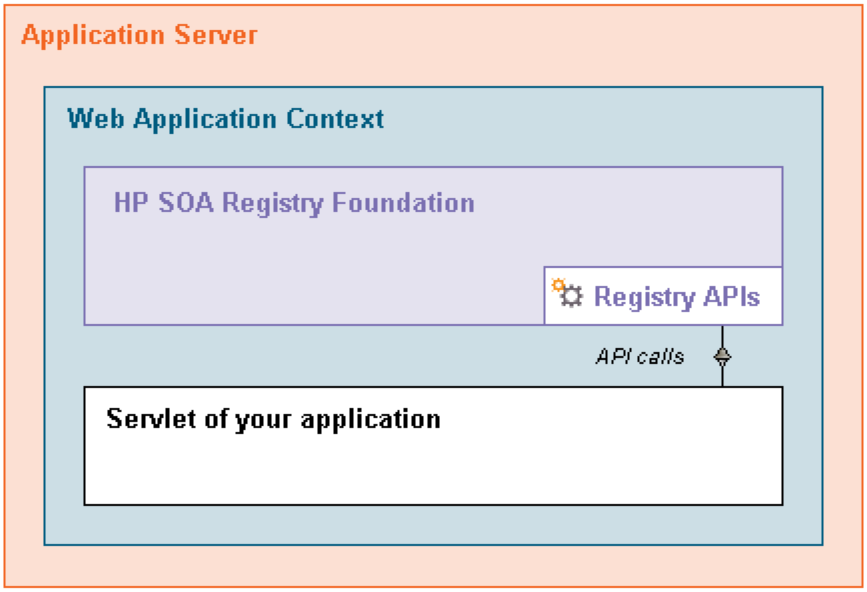
This chapter focuses on the server-side development of HPE SOA Registry Foundation extensions. Possible ways of accessing HPE SOA Registry Foundation are discussed including examples.

- Accessing backend APIs via servlet deployed on an application server.
- Custom HPE SOA Registry Foundation Modules - how to create and deploy custom HPE SOA Registry Foundation modules.
- Interceptors can monitor or modify the requests and responses of HPE SOA Registry Foundation. Interceptors are at the lowest level of HPE SOA Registry Foundation API call processing.
- Writing custom Validation services - HPE SOA Registry Foundation provides several ways to define and use validation services for taxonomies or identifier systems including remotely and locally deployed validation services and an internal validation service. For details, see "Taxonomy: Principles, Creation and Validation" on page 224 in the User's Guide. This chapter focuses on 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.

Accessing Backend APIs

This section will show you how to integrate HPE SOA Registry Foundation 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 HPE SOA Registry Foundation APIs as shown in the following figure.

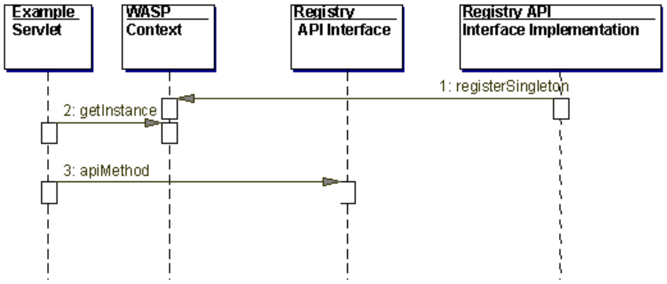
Accessing Backend Registry APIs - Architecture View



The sequence of steps that precedes access to the HPE SOA Registry Foundation API is shown in the figure below, “Accessing Backend Registry APIs - Sequence Diagram”.

1. HPE SOA Registry Foundation'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 HPE SOA Registry Foundation.

Accessing Backend Registry APIs - Sequence Diagram



> **Note:** We assume HPE SOA Registry Foundation is deployed to Tomcat. TOMCAT_HOME refers to the directory in which the application server is installed. The step-by-step procedure has been tested on Tomcat 5.0.28.

Follow these steps to create and deploy the example servlet:

1. Create the example servlet class shown in the example below, “ 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;
   %TOMCAT_HOME%\common\lib\servet-api.jar ExampleServlet.java
   ```

2. Copy `ExampleServlet.class` to the directory `TOMCAT_HOME/webapps/wasp/Web-inf/classes/com/systinet/example/servlet`.

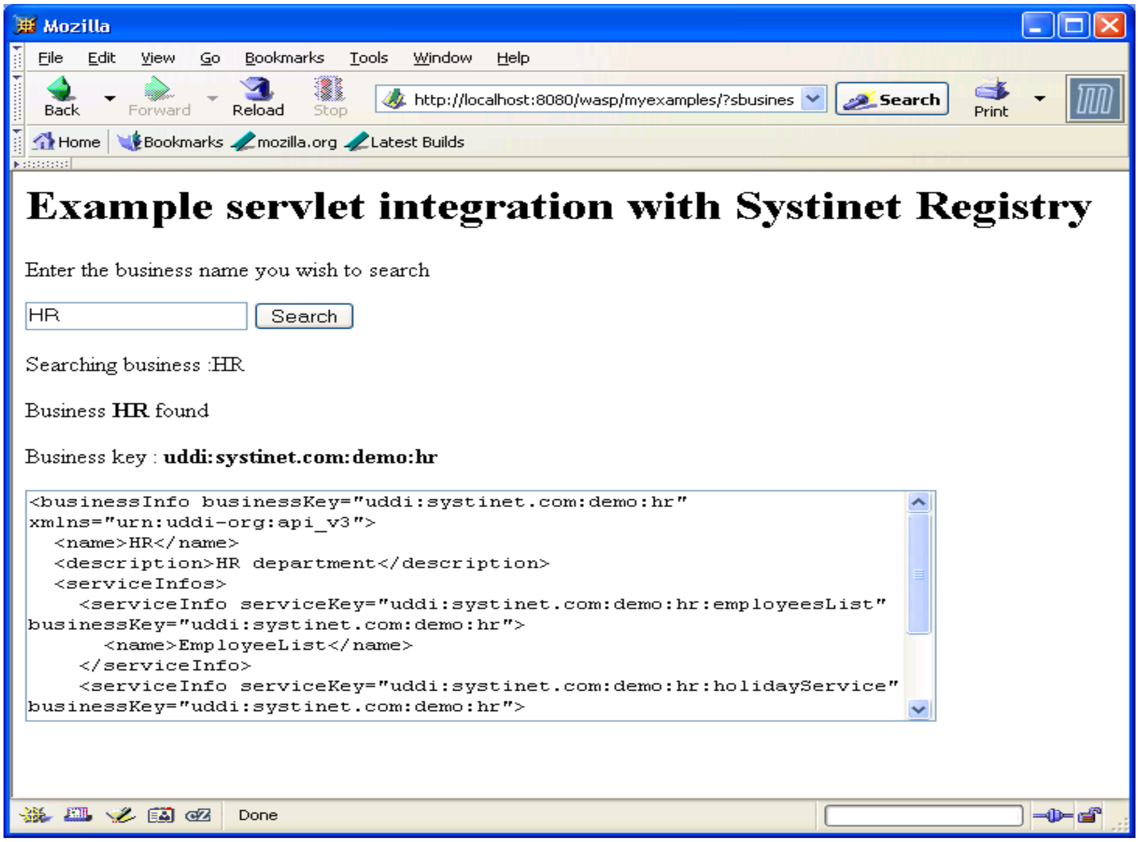
3. Add the example servlet to `TOMCAT_HOME/webapps/wasp/Web-inf/web.xml` as shown in the example below, “Example Servlet's web.xml ”.

4. Restart the Tomcat application server.

The example servlet will be available at `http://localhost:8080/wasp/myexamples`.

You can test it as shown in the following figure.

Example Servlet Output


Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost:8080/wasp/myexamples/?sbusines
Search
Print
Home Bookmarks mozilla.org Latest Builds
Done

Example servlet integration with Systinet Registry

Enter the business name you wish to search

HR [Search]

Searching business :HR

Business **HR** found

Business key : **uddi:systinet.com:demo:hr**

```
<businessInfo businessKey="uddi:systinet.com:demo:hr"
xmlns="urn:uddi-org:api_v3">
  <name>HR</name>
  <description>HR department</description>
  <serviceInfos>
    <serviceInfo serviceKey="uddi:systinet.com:demo:hr:employeesList"
businessKey="uddi:systinet.com:demo:hr">
      <name>EmployeeList</name>
    </serviceInfo>
    <serviceInfo serviceKey="uddi:systinet.com:demo:hr:holidayService"
businessKey="uddi:systinet.com:demo:hr">
```

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 HPE SOA
Registry</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) {
       }
```

```
    }
}
```

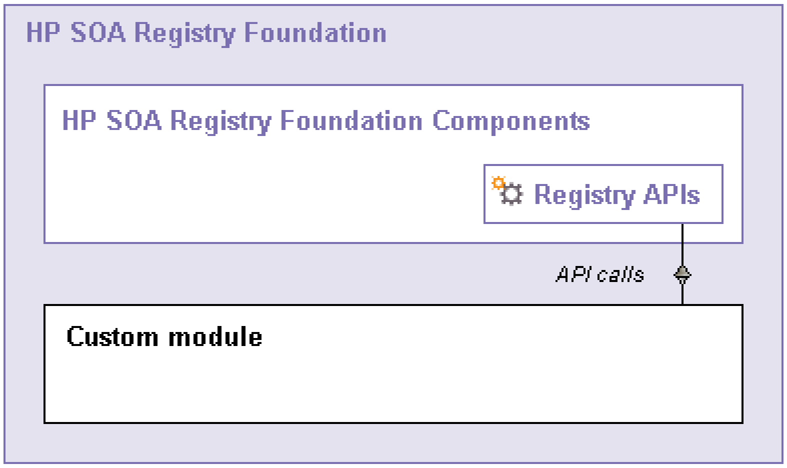
Example Servlet's web.xml

```
          <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>
```

Custom Registry Modules

In this section, we will show you how to extend HPE SOA Registry Foundation functionality with your custom modules. Custom modules can be added to HPE SOA Registry Foundation as shown in the following figure, “Custom Registry Module - Architecture View”.

Custom Registry Module - Architecture View


HP SOA Registry Foundation
HP SOA Registry Foundation Components
Registry APIs
API calls
Custom module

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 HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation shutdown.

Accessing Registry APIs

To access the HPE SOA Registry Foundation 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 the following table, “Mapping API Interface and Implemenation Classes”.

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_	com.systinet.uddi.publishing.v3.PublishingApiImpl

Publication_PortType	
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

Custom Module Sample

This section includes step-by-step instructions how to create a registry module that counts the number of restarts of HPE SOA Registry Foundation and saves the result to a configuration file.

Follow these steps:

1. Create Java file `ExampleModule.java` as shown in example, “ 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, see example, “Example configuration file for custom module”.
5. Shutdown HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation. See also the configuration file of the module where a new element counter is created.

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 configuration file for custom module

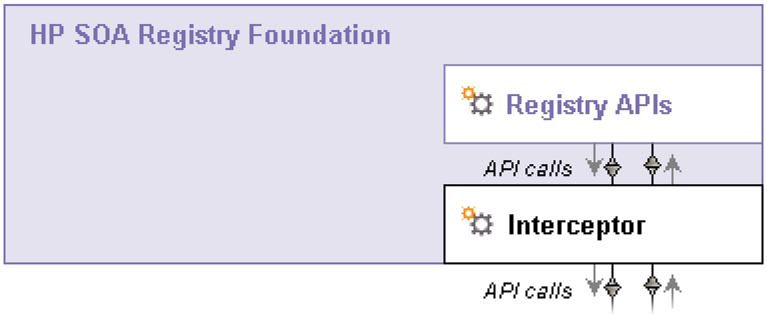
```
<?xml version="1.0" encoding="UTF-8"?>
<config name="myconf">
     <module loader="com.systinet.example.module.ExampleModule" name="MyModule">
     </module>
</config>
```

Interceptors

Interceptors can monitor or modify the requests and responses of HPE SOA Registry Foundation as shown in the figure below, “Registry Interceptors”. They are at the lowest level of HPE SOA Registry Foundation API call processing, and can be used for:

- Logging requests. See "Logging Interceptor Sample" on the next page.
- Computing message statistics. See "Request Counter Interceptor Sample" on page 496.
- Changing request arguments (adding default values)
- Prohibiting some API calls

Registry Interceptors



There are three types of HPE SOA Registry Foundation 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 HPE SOA Registry Foundation API see “Accessing Registry APIs” in " Custom Registry Modules" on page 488 for more information.

Creating and Deploying Interceptors

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 "Interceptor Configuration" on page 495.
4. Shutdown HPE SOA Registry Foundation, delete the `REGISTRY_HOME/work` directory, and restart the registry.

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, “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, see example, "Logging Interceptor Configuration File".
5. Shutdown HPE SOA Registry Foundation, delete the `REGISTRY_HOME/work` directory, and restart the registry.

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;
     }
}
```

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 see example, “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.

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 "Logging Interceptor Sample" on page 493.

Interceptor implementation is shown in example, “Request Counter Interceptor Class”. The configuration file is shown in example, “Request Counter Interceptor Configuration File”.

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
     }
}
```

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>
```

Writing a Custom Validation Service

HPE SOA Registry Foundation provides several ways to define and use validation services for taxonomies or identifier systems. For details about HPE SOA Registry Foundation taxonomies, see "Taxonomy: Principles, Creation and Validation" on page 224 in the User's Guide. This chapter focuses on custom validation services that you can deploy:

- Locally on HPE SOA Registry Foundation - 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.

Deploying Validation Service

Once the validation service is implemented, you can deploy the validation service locally on HPE SOA Registry Foundation. To deploy the validation service on HPE SOA Registry Foundation

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 HPE SOA Registry Foundation, delete the `REGISTRY/work` directory, and restart HPE SOA Registry Foundation.

For more information, see "Validation" in "Advanced Demos" on page 570. For details about the configuration of Validation Services, see "Taxonomy Management" on page 310 in the Administrator's Guide.

To deploy an external validation service, you must create a deployment package.

External Validation Service

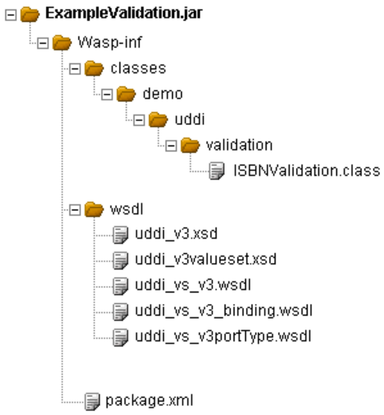
This section shows you how to implement and package an external validation service that will be deployed to Systinet Server for Java 5.5. We show you how to package and deploy the ISBN validation service from the validation demo described in Validation" in "Advanced Demos" on page 570. It is assumed you have already built the Validation demo.

> **Note:** We also assume HPE SOA Registry Foundation 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:



```
ExampleValidation.jar
  Wasp-inf
    classes
      demo
        uddi
          validation
            ISBNValidation.class
    wsdl
      uddi_v3.xsd
      uddi_v3valueset.xsd
      uddi_vs_v3.wsdl
      uddi_vs_v3_binding.wsdl
      uddi_vs_v3portType.wsdl
    package.xml
```

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 in example, “ package.xml ”, to the package.

2. Deploy the validation package with required HPE SOA Registry Foundation 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`**

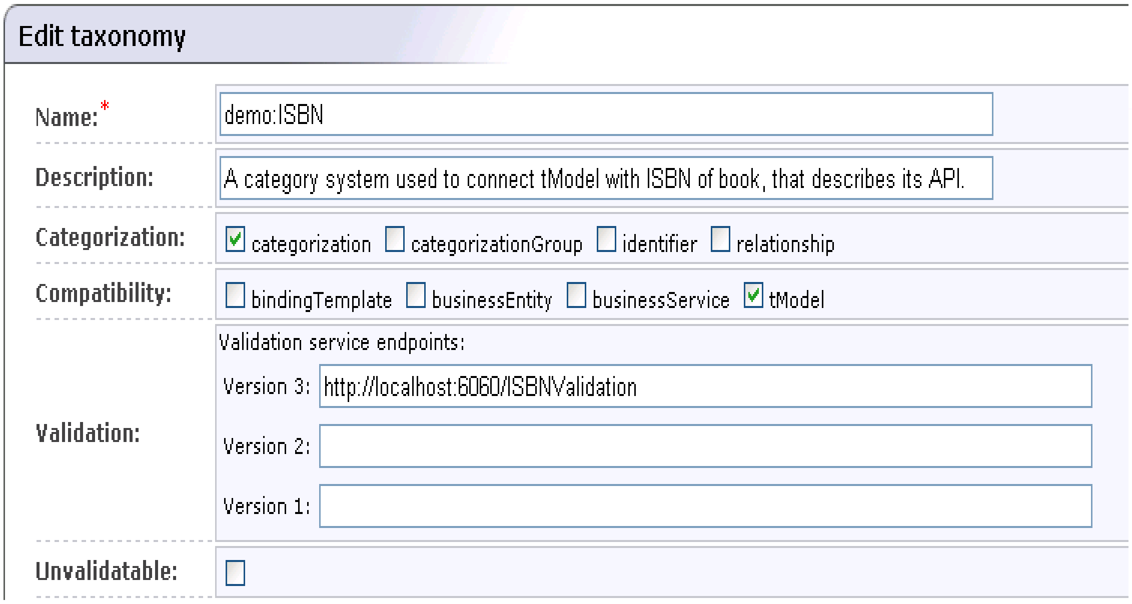
   c. **`copy ExampleValidation.jar %WASP_HOME%\app\system\uddi`**

3. Shut down 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, see “Uploading Taxonomies” in "Taxonomy Management" on page 310 in the User's Guide.

   Modify the validation service endpoint as shown in the following figure, “Validation for Checked Taxonomy” by using the Taxonomy Editor in the HPE Systinet Workbench.

   Validation for Checked Taxonomy



Edit taxonomy

Name:*	demo:ISBN
Description:	A category system used to connect tModel with ISBN of book, that describes its API.
Categorization:	☑ categorization ☐ categorizationGroup ☐ identifier ☐ relationship
Compatibility:	☐ bindingTemplate ☐ businessEntity ☐ businessService ☑ tModel
Validation:	Validation service endpoints: Version 3: http://localhost:6060/ISBNValidation Version 2: Version 1:
Unvalidatable:	☐

You can run and test the validation service using Validation demo described in "Validation" in "Advanced Demos" on page 570.

Sample File

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"
     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"
             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>
```

Writing a Subscription Notification Service

This section will show you how to implement a subscription notification service. When you create a HPE SOA Registry Foundation subscription you can specify a notification listener service endpoint as described in " Subscriptions in HPE SOA Registry Foundation" on page 207 in the User's Guide.

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 Product Documentation 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 HPE SOA Registry Foundation client packages into Systinet Server for Java.
3. Create a subscription using the Registry Console.

> **Note:** We assume HPE SOA Registry Foundation 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/`.

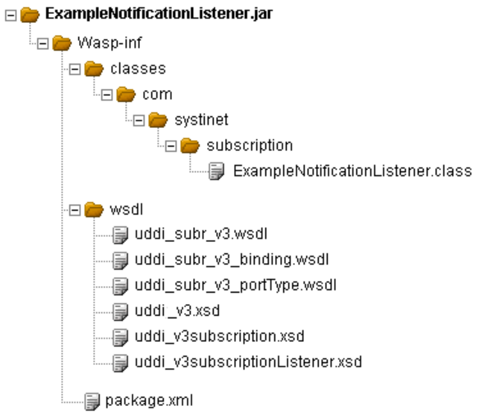
Now we will describe the process in detail:

1. Create the subscription notification service class shown in example, " 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:



```
ExampleNotificationListener.jar
  Wasp-inf
    classes
      com
        systinet
          subscription
            ExampleNotificationListener.class
    wsdl
      uddi_subr_v3.wsdl
      uddi_subr_v3_binding.wsdl
      uddi_subr_v3_portType.wsdl
      uddi_v3.xsd
      uddi_v3subscription.xsd
      uddi_v3subscriptionListener.xsd
    package.xml
```

   b. Copy the wsdl and schema files from `REGISTRY_HOME/doc/wsdl` to the package.

   c. Copy the `package.xml file` shown in example, “ package.xml ”, to the package.

4. Deploy the service notification package with required HPE SOA Registry Foundation client packages into the 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`**

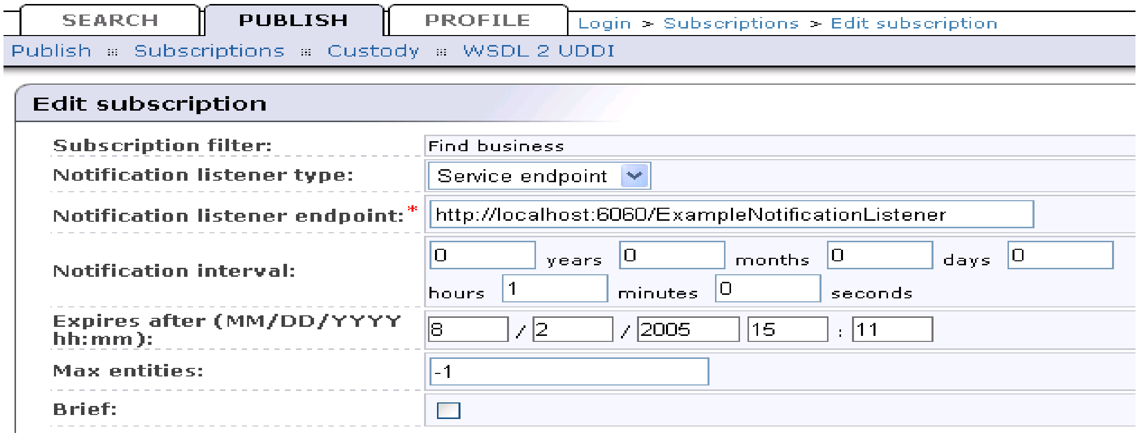
5. Shutdown the Systinet Server for Java, delete the WASP_HOME/work directory, and restart the Systinet Server for Java

6. Create a subscription using the Registry Console.

   See “Publishing Subscriptions” in "Publishing" on page 265 in the User's Guide 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 the following figure, “Create Subscription”

Create Subscription


SEARCH
PUBLISH
PROFILE
Login > Subscriptions > Edit subscription
Publish :: Subscriptions :: Custody :: WSDL 2 UDDI

Edit subscription

Subscription filter: Find business

Notification listener type: Service endpoint

Notification listener endpoint:* http://localhost:6060/ExampleNotificationListener

Notification interval: 0 years 0 months 0 days 0 hours 1 minutes 0 seconds

Expires after (MM/DD/YYYY hh:mm): 8 / 2 / 2005 15 : 11

Max entities: -1

Brief: □

Sample Files

Example: 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: 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="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>
```

Systinet Web Framework

This section describes HPE SOA Registry Foundation from the developer's point of view. It describes the HPE SOA Registry Foundation Framework architecture and configuration.

- “Architecture Description”
- “Directory Structure”
- “Framework Configuration”
- “syswf JSP tag library”
- “Typical Customization Tasks”

Architecture Description

The framework uses the Jasper engine, a part of the Tomcat server. It is able to run on Jasper1 from Tomcat version 4.1 (Servlet API 2.3/JSP spec 1.2) or Jasper2 from Tomcat version 5 (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.

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)`

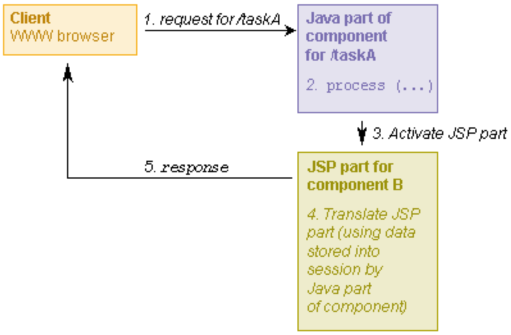
The `process()` method is called just before the translation of the component's JSP part is started, so it should take care of data preparation and it should also handle the actions requested by the user (react to pressed buttons, etc.).

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, “Request Diagram” demonstrates how requests for the page are handled by the Web Framework:

Request Diagram


Client
WWW browser
1. request for /taskA
Java part of component for /taskA
2. process (...)
3. Activate JSP part
5. response
JSP part for component B
4. Translate JSP part (using data stored into session by Java part of component)

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

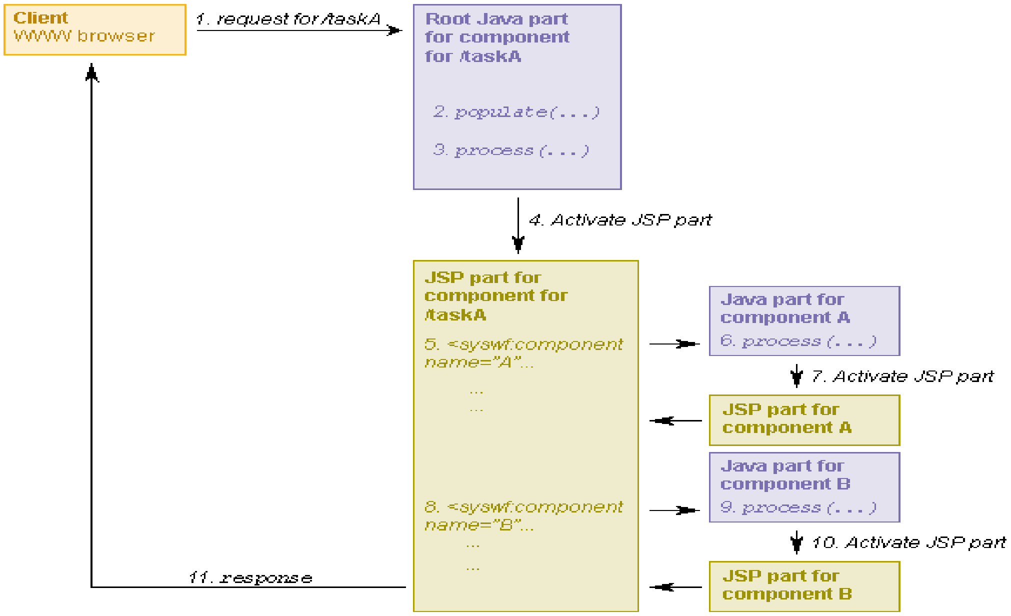
browser.

> **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, “Nesting Components Diagram” 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:

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

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.

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: 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 above example, “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: 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`.

Directory Structure

JSP pages for the HPE SOA Registry Foundation 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

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

Content of Page Directories

Directory	Description
`account`	All pages related to account management
`admin`	Administration tools for tModel deletion and key replacement
`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
`xsd2uddi`	Inquiry and publishing pages for mapping of XML schemas to UDDI

Framework Configuration

All needed configuration settings are stored in the file `REGISTRY_HOME/app/uddi/conf/web.xml`

Component

Specifies configuration of page components.

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.

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

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.

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

Configuration Elements

Attribute	Description
url	First part of the URL used to access HPE SOA Registry Foundation without encryption (plain HTTP); this part should contain the http protocol prefix, hostname, and port.
secureUrl	First part of the URL used to access HPE SOA Registry Foundation using encryption. This part should contain https protocol prefix, hostname and port.
context	Context part of the URL, used to access HPE SOA Registry Foundation tasks; the default value is uddi/web for standalone registries and wasp/uddi/web for registries deployed to an application server.
dataContext	Context part of the URL, used to access HPE SOA Registry Foundation'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 deployed 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

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 in which Tomcat is running.
`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.

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.

`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

`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: 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.

`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.

`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

`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.

`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

`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.

`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

`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.

`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

`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.

`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

`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.

`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

`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.

`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.

`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: Component Parameters

```
<syswf:component name="navigationPath" prefix="path"/>
```

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`.

UDDI From Developer Tools

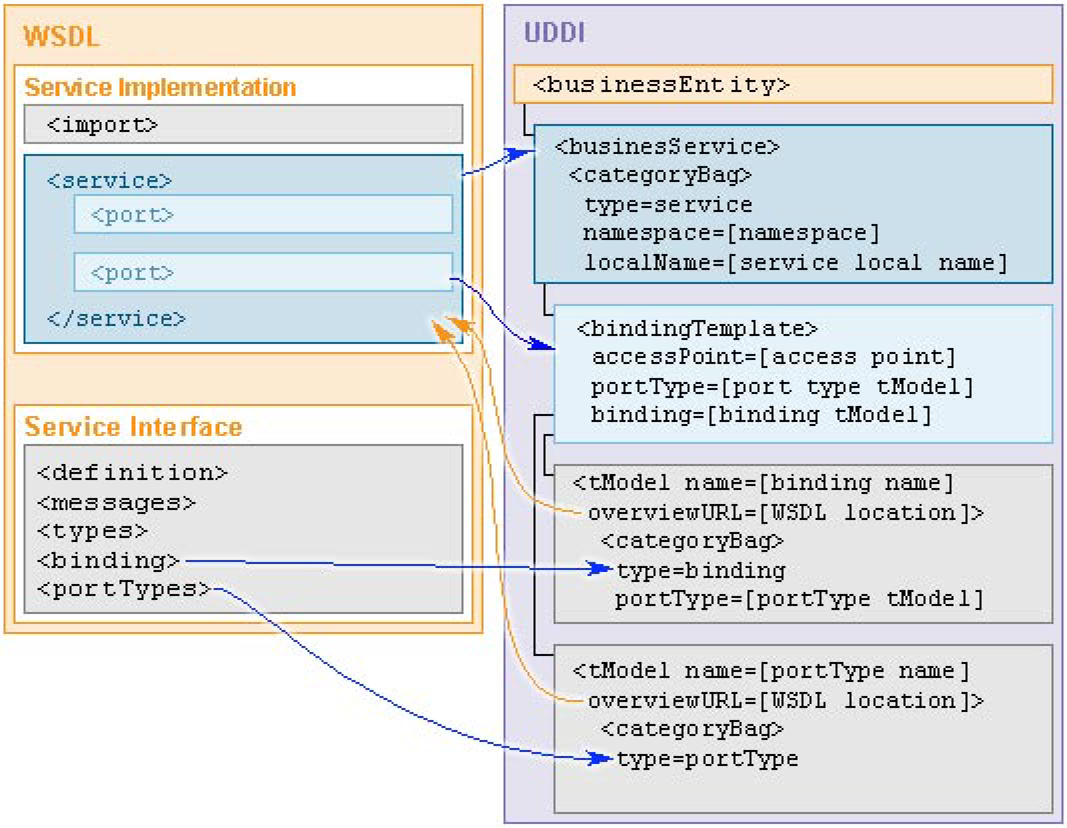
In this section, we will show you how to access UDDI from the following tools:

- 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 “WSDL Mapping to UDDI” shows how a WSDL is mapped to UDDI. For more information, see the OASIS Technical Note "Using WSDL in a UDDI Registry" in https://www.oasis-open.org/.

WSDL Mapping to UDDI



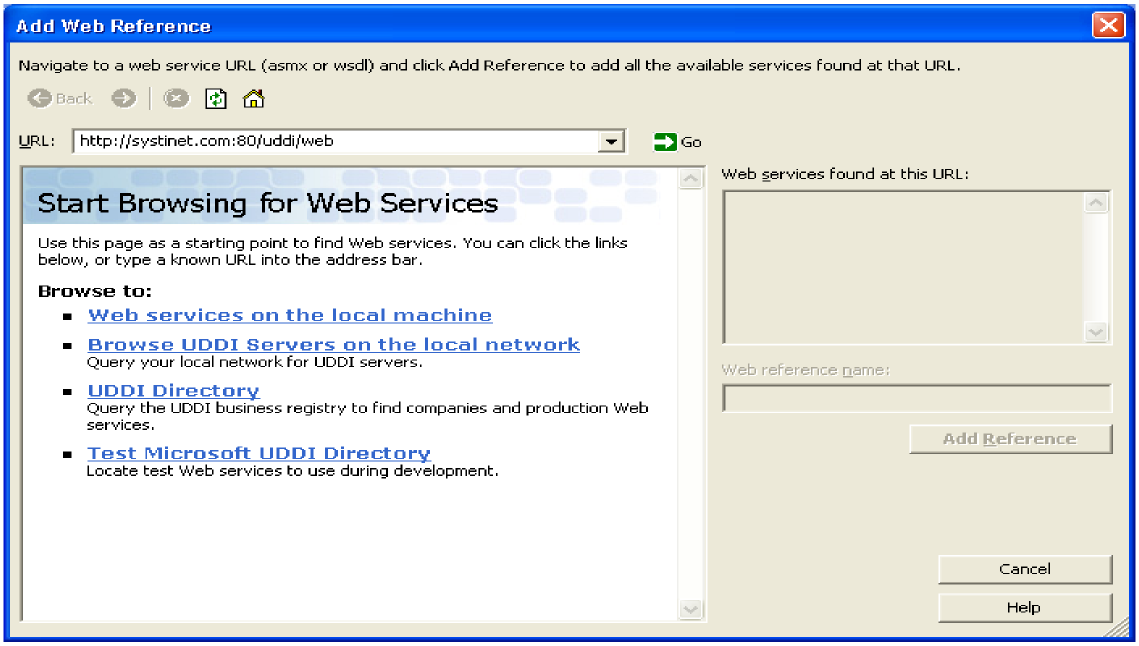
UDDI From MS Visual Studio

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, “Add Web Reference Default” appears. Enter the URI of a UDDI registry or the URI of a WSDL document representing the Web service.

Add Web Reference Default



Navigate to a web service URL (asmx or wsdl) and click Add Reference to add all the available services found at that URL.

URL: http://systinet.com:80/uddi/web

Start Browsing for Web Services

Use this page as a starting point to find Web services. You can click the links below, or type a known URL into the address bar.

Browse to:

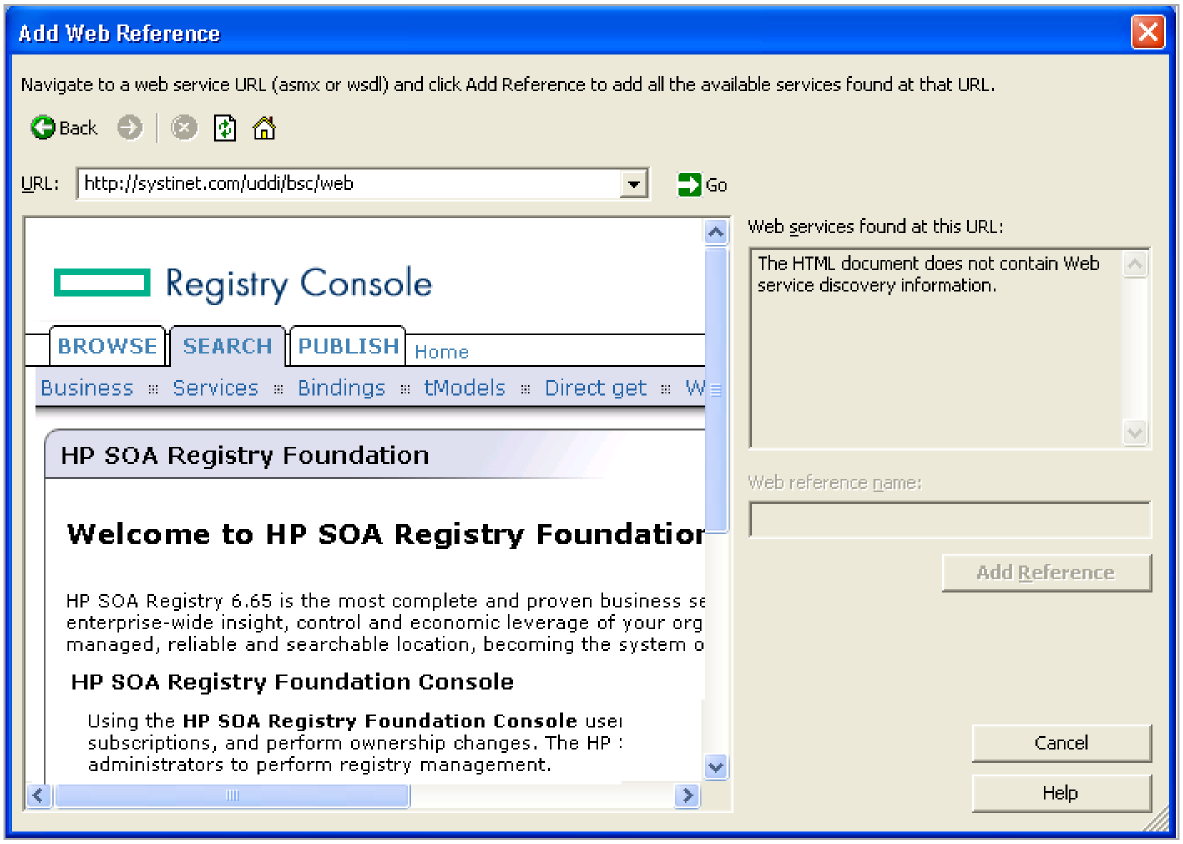
- Web services on the local machine
- Browse UDDI Servers on the local network
  Query your local network for UDDI servers.
- UDDI Directory
  Query the UDDI business registry to find companies and production Web services.
- Test Microsoft UDDI Directory
  Locate test Web services to use during development.

Web services found at this URL:

Web reference name:

Figure “Searching HPE SOA Registry Foundation via Web Reference Wizard” shows how to browse/search HPE SOA Registry Foundation via the **Add Web Reference Wizard**.

Searching HPE SOA Registry Foundation via Web Reference Wizard



Navigate to a web service URL (asmx or wsdl) and click Add Reference to add all the available services found at that URL.

URL: http://systinet.com/uddi/bsc/web

HP SOA Registry Foundation

Welcome to HP SOA Registry Foundatior

HP SOA Registry 6.65 is the most complete and proven business se
enterprise-wide insight, control and economic leverage of your org
managed, reliable and searchable location, becoming the system o

HP SOA Registry Foundation Console

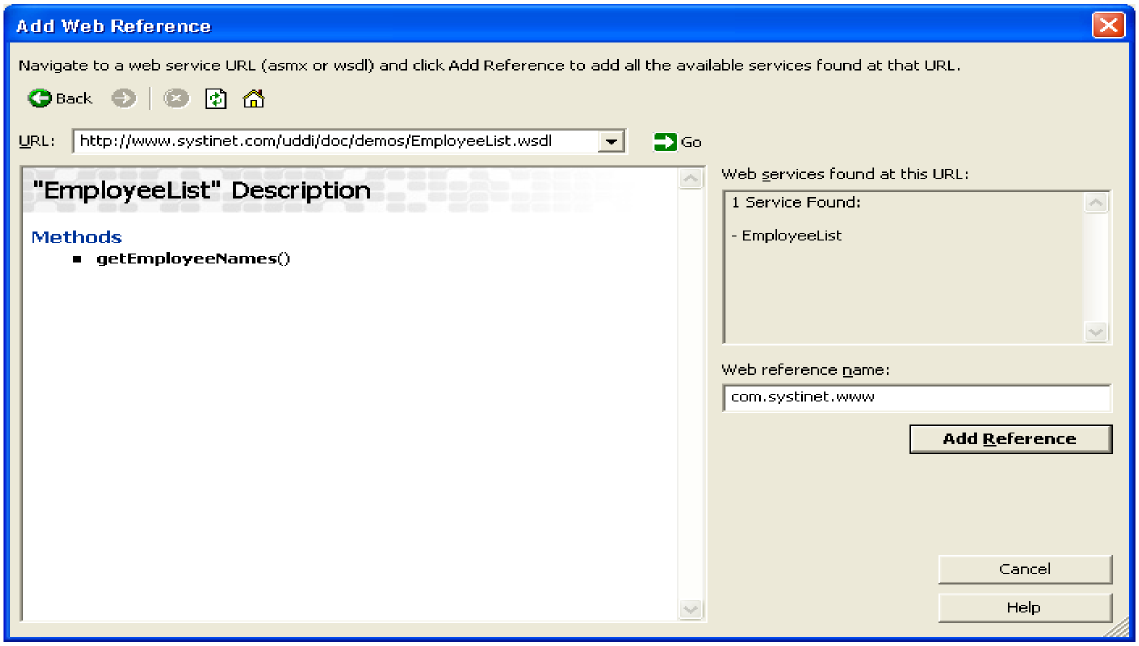
Using the **HP SOA Registry Foundation Console** usei
subscriptions, and perform ownership changes. The HP :
administrators to perform registry management.

Web services found at this URL:

The HTML document does not contain Web service discovery information.

Web reference name:

Add Web Reference - Found Web service



Navigate to a web service URL (asmx or wsdl) and click Add Reference to add all the available services found at that URL.

URL: http://www.systinet.com/uddi/doc/demos/EmployeeList.wsdl

"EmployeeList" Description

Methods

- getEmployeeNames()

Web services found at this URL:

1 Service Found:

- EmployeeList

Web reference name: com.systinet.www

If you find a WSDL file, the wizard shown in the above figure, “Add Web Reference - Found Web service” parses the WSDL file displaying Web service method. Then, you can click **Add Reference** button to add the reference to your project.

How to Debug

The following sections describe debugging:

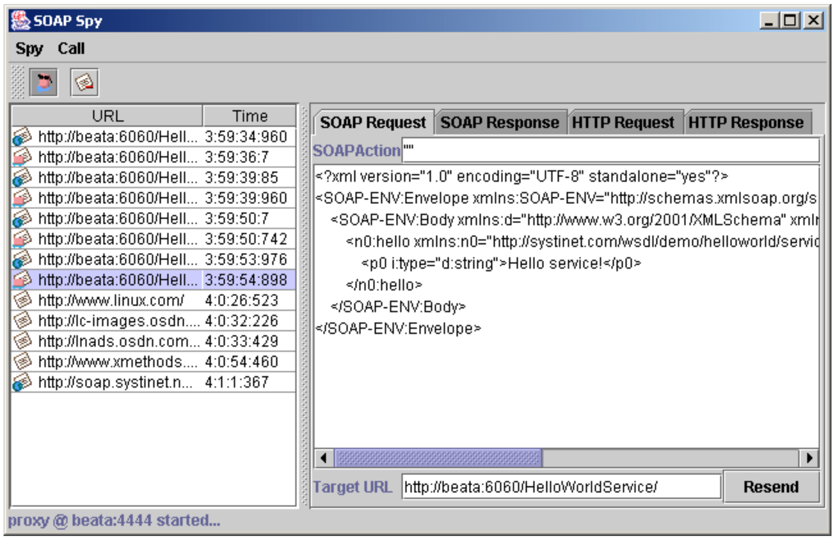
"SOAPSpy Tool" below

"Logging" on page 530

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 “SOAPSpy Tool” shows the SOAPSpy dialog box.

SOAPSpy Tool



URL	Time
http://beata:6060/Hell...	3:59:34:960
http://beata:6060/Hell...	3:59:36:7
http://beata:6060/Hell...	3:59:39:85
http://beata:6060/Hell...	3:59:39:960
http://beata:6060/Hell...	3:59:50:7
http://beata:6060/Hell...	3:59:50:742
http://beata:6060/Hell...	3:59:53:976
http://beata:6060/Hell...	3:59:54:898
http://www.linux.com/	4:0:26:523
http://lc-images.osdn....	4:0:32:226
http://lnads.osdn.com...	4:0:33:429
http://www.xmethods....	4:0:54:460
http://soap.systinet.n...	4:1:1:367

SOAPAction ""

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/s
  <SOAP-ENV:Body xmlns:d="http://www.w3.org/2001/XMLSchema" xmlı
    <n0:hello xmlns:n0="http://systinet.com/wsdl/demo/helloworld/servic
      <p0 i:type="d:string">Hello service!</p0>
    </n0:hello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Target URL http://beata:6060/HelloWorldService/

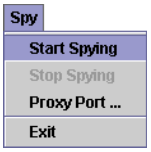
proxy @ beata:4444 started...

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.

Running SOAPSpy

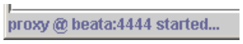
This tool is placed in the `bin` subdirectory of your HPE SOA Registry Foundation server distribution. To start SOAPSpy, enter the command **`SoapSpy.bat`** on Windows platforms, or **`./SoapSpy.sh`** on UNIX machines.

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 “Start Spying”.

Status Line



proxy @ beata:4444 started...

The lower part of the window contains a status bar, shown in figure “Status Line”, 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**

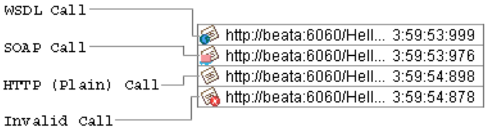
Using SOAPSpy

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 “Call Types”. 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.

Call Types

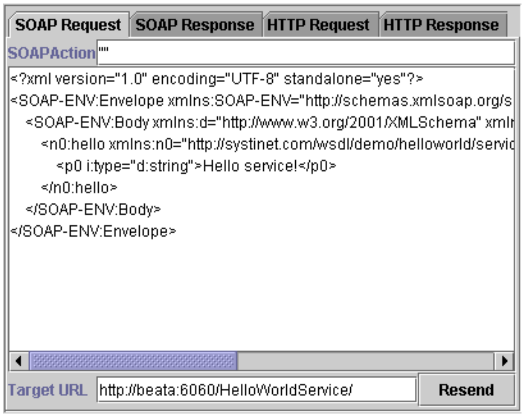


```
http://beata:6060/Hell... 3:59:53:999
http://beata:6060/Hell... 3:59:53:976
http://beata:6060/Hell... 3:59:54:898
http://beata:6060/Hell... 3:59:54:878
```

SOAP Request Tab

The SOAP Request tab, shown in figure “Request Tab”, 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.

Request Tab



SOAPAction ""

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/s
  <SOAP-ENV:Body xmlns:d="http://www.w3.org/2001/XMLSchema" xmlr
    <n0:hello xmlns:n0="http://systinet.com/wsdl/demo/helloworld/servic
      <p0 i:type="d:string">Hello service!</p0>
    </n0:hello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Target URL http://beata:6060/HelloWorldService/

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.

For example:

```
java -Dhttp.proxyHost=%CLIENT_COMPUTER_NAME% -Dhttp.proxyPort=4444 org.my.FooClient
```

> **Note:** 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.

Logging

HPE SOA Registry Foundation wraps the Log4j 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);
         }
   ...
   ```

Chapter 6: Demos

The HPE SOA Registry Foundation demos suite is used to teach the capabilities of the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation deployed to an application server, make sure you have properly imported the SSL certificate of the application server to the HPE SOA Registry Foundation configuration. For more information see "Deployment to an Application Server" on page 120 in the Installation Guide. You may also need to modify the HPE SOA Registry Foundation 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 HPE SOA Registry is running

The demos are divided into the following categories:

"Basic Demos" below - 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" on page 570- The Advanced demos discuss custody, subscriptions, validation, and taxonomies.

"Security Demos" on page 601 - In the Security demos, we cover accounts, groups, permissions, and access control lists (ACLs).

"Resources Demos" on page 623 - In the resources demos, we cover publishing of WSDL and XSD.

Basic Demos

Basic Demos section includes the following demos:

- "UDDI v1" on the next page
- "UDDI v2" on page 544
- "UDDI v3" on page 557

UDDI v1

- "Inquiry v1" below
- "Publishing v1" on page 537

Inquiry v1

The HPE SOA Registry Foundation basic inquiry demo set is used to demonstrate the HPE SOA Registry Foundation 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.

You will learn how to use the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation basic inquiry demo set contains following demos to assist you in learning the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation and set the REGISTRY_HOME environment variable to its installation location.

To run the HPE SOA Registry Foundation'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 HPE SOA Registry Foundation 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

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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation Basic Inquiry demo set. Our example continues with the FindTModel demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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 FindTModel
UNIX:	./run.sh FindTModel

The output of this demo will resemble the following:

```
Running FindTModel demo...
***************************************************************************
***    HPE SOA 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.

Publishing v1

The HPE SOA Registry Foundation basic publishing demo set demonstrates the HPE SOA Registry Foundation application programming interface's capabilities and teaches how to use this API to perform basic publishing calls to a UDDI registry.

The HPE SOA Registry Foundation basic publishing demos cover the publication aspect of the UDDI Version 1 Specification. You will learn, how to use the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation basic publishing demo set contains the following demos to assist you in learning the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to its installation location.

To run the HPE SOA Registry Foundation'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 HPE SOA Registry Foundation 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

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, authInfo);a
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(authInfo));
      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 HPE SOA Registry Foundation 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.getBusinessEntityArrayList();
      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 HPE SOA Registry Foundation Basic Publishing demo set. Let us continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
**************************************************************************
          HPE  SOA 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.

UDDI v2

- "Inquiry v2" below
- "Publishing v2" on page 549

Inquiry v2

The HPE SOA Registry Foundation basic inquiry demo set is used to demonstrate the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation basic inquiry demos cover inquiry aspects of the UDDI Version 2.0.4 Specification. You will learn how to use the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation basic inquiry demo set contains following demos to assist you in learning the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation registry and set the REGISTRY_HOME environment variable to its installation location.

To run HPE SOA Registry Foundation'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 HPE SOA Registry Foundation 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

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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation Basic Inquiry demo set. Our example continues with the FindTModel demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
**************************************************************************
*** HPE SOA 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.

Publishing v2

The HPE SOA Registry Foundation basic publishing demo set demonstrates the HPE SOA Registry Foundation application programming interface's capabilities and teaches how to use this API to perform basic publishing calls to a UDDI registry.

The HPE SOA Registry Foundation basic publishing demos cover the publication aspect of the UDDI Version 2 Specification. You will learn how to use the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation basic publishing demo set contains the following demos. They will assist you in learning the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation and set the REGISTRY_HOME environment variable to its installation location.

To run the HPE SOA Registry Foundation'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 HPE SOA Registry Foundation 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

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, authInfo);
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(authInfo));
      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 HPE SOA Registry Foundation 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.getBusinessEntityArrayList();
      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 HPE SOA Registry Foundation Basic Publishing demo set. Let us continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
**************************************************************************
***    HPE SOA 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.

UDDI v3

- "Inquiry v3" below
- "Publishing v3" on page 563

Inquiry v3

The HPE SOA Registry Foundation basic inquiry demo set is used to demonstrate the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation basic inquiry demos cover the inquiry aspect of the UDDI Version 3.0.1 Specification. You will learn how to use the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation basic inquiry demo set contains following demos. They will assist you in learning the HPE SOA Registry Foundation 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.

Prerequisites and Preparatory Steps: Code

We expect, that you have already installed the HPE SOA Registry Foundation and set the REGISTRY_HOME environment variable to its installation location.

To run HPE SOA Registry Foundation'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 HPE SOA Registry Foundation 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

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\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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation Basic Inquiry demo set. Our example continues with the FindTModel demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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 FindTModel
UNIX:	./run.sh FindTModel

The output of this demo will resemble the following:

```
*******************************************************************
***    HPE SOA 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.

Publishing v3

The HPE SOA Registry Foundation basic publishing demo set demonstrates the HPE SOA Registry Foundation application programming interface's capabilities and teaches how to use this API to perform basic publishing calls to a UDDI registry.

The HPE SOA Registry Foundation basic publishing demos cover the publication aspect of the UDDI Version 3 Specification. You will learn, how to use the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation basic publishing demo set contains the following demos. They will assist you in learning the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation and set the REGISTRY_HOME environment variable to its installation location.

To run the HPE SOA Registry Foundation'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 HPE SOA Registry Foundation 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

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, authInfo);
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 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 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 HPE SOA Registry Foundation 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.getBusinessEntityArrayList();
      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 HPE SOA Registry Foundation Basic Publishing demo set. Let's continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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:

```
**************************************************************************
***    HPE SOA 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.

Advanced Demos

Advanced demos section includes the following demos:

- **"Advanced Inquiry - Range Queries" on the next page** - The HPE SOA Registry Foundation Range queries demos set demonstrates, how to use HPE SOA Registry Foundation inquiry enhancement - Range Queries. HPE SOA Registry Foundation range queries functionality allows you to search UDDI entities with the ability to use comparative operators (>, <) for matching keyValues in keyedReferences.
- **"Custody" on page 576** - The HPE SOA Registry Foundation 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" on page 582** - The HPE SOA Registry Foundation advanced subscription demos cover the subscription aspects of the UDDI Version 3 Specification. They teach how to use the HPE SOA Registry Foundation client API to create new subscriptions, get lists of subscriptions, get subscription results, and delete subscriptions.
- **"Validation" on page 590** - 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" on page 595** - The Taxonomy API is used to manage and query taxonomies in the HPE SOA Registry Foundation. 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.

Advanced Inquiry - Range Queries

The HPE SOA Registry Foundation Range queries demos set demonstrates, how to use HPE SOA Registry Foundation inquiry enhancement - Range Queries. HPE SOA Registry Foundation 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`

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your registry must be running. To start the HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation 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

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

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
(qualifiedKeyedReference)));
     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;
}
```

Building and Running Demos

This section shows, how to build and run the HPE SOA Registry Foundation Advanced Inquiry demo set. Let us continue with our FindBusiness demo.

1. Be sure that the demo are properly configured and the HPE SOA Registry Foundation 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:

```
**********************************************************************
***     HPE SOA 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://systinet.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:systinet.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"
serviceKey="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"
businessKey="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"
serviceKey="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"
keyName="uddi.org:xml:namespace"
keyValue="http://systinet.com/wsdl/demo/uddi/services/"/>
              <keyedReference tModelKey="uddi:uddi.org:wsdl:types"
keyName="uddi.org:wsdl:types"
keyValue="port"/>
              <keyedReference tModelKey="uddi:uddi.org:xml:localName"
keyName="uddi.org:xml:localName"
keyValue="EmployeeList"/>
              <keyedReference
tModelKey="uddi:systinet.com:taxonomy:endpoint:availability"
keyName="Available"
```

```
keyValue="Available"/>
              <keyedReference
tModelKey="uddi:systinet.com:taxonomy:endpoint:status" keyName="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>
********************************************************
```

Custody

The HPE SOA Registry Foundation demo is used to demonstrate the registry's application programming interface's capabilities and to demonstrate how to use this API.

The HPE SOA Registry Foundation Custody demo covers the custody transfer aspects of the UDDI Version 3.01 Specification.. 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.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your registry must be running. To start the HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation 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

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

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_PASSWORD));
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");
}
```

Building and Running Demos

This section shows how to build and run the HPE SOA Registry Foundation Custody demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
**************************************************************************
*** HPE SOA 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.

Subscription

The HPE SOA Registry Foundation advanced subscription demo set demonstrates the HPE SOA Registry Foundation application programming interface's capabilities and shows how to use the Subscription API to perform subscription calls to the registry.

The HPE SOA Registry Foundation advanced subscription demos cover the subscription aspects of the UDDI Version 3 Specification. They teach how to use the HPE SOA Registry Foundation client API to create new subscriptions, get lists of subscriptions, get subscription results, and delete subscriptions.

The HPE SOA Registry Foundation basic publishing demo set contains the following demos to assist you in learning the HPE SOA Registry Foundation 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.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your registry must be running. To start the HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation 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

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

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;
}
```

Building and Running Demos

This section shows how to build and run the HPE SOA Registry Foundation Advanced Subscription demo set. Let us continue with our GetSubscriptionResults demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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...
************************************************************************
*** HPE SOA 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 HPE SOA Registry Foundation Basic Publishing demos to save business entity named Marketing.

```
Running SaveBusiness demo...
**************************************************************************
***  HPE SOA 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...
**************************************************************************
***HPE SOA 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.

```
**************************************************************************
***HPE SOA 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
```

Validation

The HPE SOA Registry Foundation 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 HPE SOA Registry Foundation.

ValidationDemo Demonstrates how to save a tModel with the keyedReference, that uses demo:ISBN categorization checked by ISBNValidation.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your registry must be running. To start the HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation 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

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

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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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 " +
keyedReference.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 HPE SOA Registry Foundation was configured to deny such usage. The optimal code would check only categoryBag in tModels.

Building and Running Demos

This section shows, how to build, deploy and run the HPE SOA Registry Foundation Advanced Validation demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation, delete the REGISTRY_HOME/work directory, and restart the HPE SOA Registry Foundation.

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.

Taxonomy

The HPE SOA Registry Foundation Taxonomy demos demonstrates the HPE SOA Registry Foundation's Taxonomy capabilities and show how to use this API.

The Taxonomy is used to manage and query taxonomies in the HPE SOA Registry Foundation. 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 HPE SOA Registry Foundation contains the following demos to assist you in learning the HPE SOA Registry Foundation 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 criteria.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your registry must be running. To start the HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation 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

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

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 HPE SOA Registry Foundation, 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++;
     }
}
```

Building and Running Demos

This section shows, how to build and run the HPE SOA Registry Foundation Advanced Taxonomy demo set. Let's continue with our SaveTaxonomy demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
**************************************************************************
*** HPE SOA 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.

Security Demos

Security Demos section includes the following demos:

- **"Account" below**- You will learn how to register new accounts (or update existing accounts), enable, get, find, and delete accounts.
- **"Group" on page 606** - You will learn how to create or update, get, find and delete groups.
- **"Permission" on page 612** - You will learn how to set and search permissions.
- **"ACL" on page 616** - 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.

Account

The HPE SOA Registry Foundation Account Demos are used to demonstrate the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation security account demo set contains the following demos to assist you in learning the HPE SOA Registry Foundation 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.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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

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

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 HPE SOA Registry Foundation 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 HPE SOA Registry Foundation API call `save_userAccount` is performed in the method `saveUserAccount`.

```
public static UserAccount saveUserAccount(Save_userAccount save) throws
SOAPException, AccountException {
     AccountApi accountApi = getAccountStub();
     System.out.print("Save in progress ...");
     UserAccount userAccount = accountApi.save_userAccount(save);
     System.out.println(" done");
     return userAccount;
}
```

Building and Running Demos

This section shows how to build and run the HPE SOA Registry Foundation Account demos.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
**************************************************************************
***   HPE SOA 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.

Group

The HPE SOA Registry Foundation Group demos are used to demonstrate the HPE SOA Registry Foundation 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 HPE SOA Registry Foundation security group demo set contains the following demos to assist you in learning the HPE SOA Registry Foundation 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.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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 `Group` demo are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\security\group\env.properties
UNIX:	$REGISTRY_HOME/demos/security/group/env.properties

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

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 HPE SOA Registry Foundation 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++;
     }
}
```

Building and Running Demos

This section shows how to build and run the HPE SOA Registry Foundation Group demos.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
**************************************************************************
***    HPE SOA 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.

Permission

The HPE SOA Registry Foundation Permission Demos are used to demonstrate the HPE SOA Registry Foundation application programming interface's capabilities and to demonstrate how to use this API.

You will learn how to set and search permissions.

The HPE SOA Registry Foundation security permission demo set contains the following demos to assist you in learning the HPE SOA Registry Foundation 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.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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

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

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.getProperty(USER_
JOHN_NAME));
String type = UserInput.readString("Enter permission type",

"org.systinet.uddi.security.permission.ApiManagerPermission");
String name = UserInput.readString("Enter permission name",
"org.systinet.uddi.client.taxonomy.v3.TaxonomyApi");
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(principal)));
      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 HPE SOA Registry Foundation 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");
}
```

Building and Running Demos

This section shows how to build and run the HPE SOA Registry Foundation Permission demos.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
**************************************************************************
***  HPE SOA 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.

ACL

The HPE SOA Registry Foundation ACL Demos demonstrate the HPE SOA Registry Foundation ACL application programming interface's capabilities and how to use this API.

The SOA Registry Foundation 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 HPE SOA Registry Foundation Security ACL demo set contains the following demos to assist you in learning the HPE SOA Registry Foundation 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.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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

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

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.getProperty(USER_JOHN_
NAME));
String passwordGood = UserInput.readString("Enter good user's password",
                                         DemoProperties.getProperty(USER_JOHN_
PASSWORD));
String loginBad = UserInput.readString("Enter bad user's login",
                                             DemoProperties.getProperty(USER_JANE_
NAME));
String passwordBad = UserInput.readString("Enter bad user's password",
                                         DemoProperties.getProperty(USER_JANE_
PASSWORD));
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfoOwner = getAuthInfo(owner, password, security);
Save_business saveBusiness = createSaveBusiness(name, description, loginGood,
loginBad, authInfoOwner);
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;
}
```

Building and Running Demos

This section shows how to build and run the HPE SOA Registry Foundation ACL demos.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
***************************************************************************
***    HPE  SOA 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.

Resources Demos

The Resources Demos section includes the following demos:

- "WSDL2UDDI v2" below- Teaches how to publish, unpublish and find a WSDL document in UDDI version 2.
- "WSDL2UDDI v3" on page 630- Teaches how to publish, unpublish and find a WSDL document in UDDI version 3.
- "XSD2UDDI" on page 636- Teaches how to publish, unpublish and find an XML Schema.

WSDL2UDDI v2

The HPE SOA Registry Foundation WSDL2UDDI demo set is used to demonstrate the HPE SOA Registry Foundation WSDL2UDDI application programming interface's capabilities and to demonstrate how to use this API. The HPE SOA Registry Foundation WSDL2UDDI demos cover the UDDI Version 2.0.4 Specification. You will learn how to query and publish a WSDL to a UDDI registry over a SOAP interface. The HPE SOA Registry Foundation 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.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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

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://localhost:8080/uddi/wsdl2uddi`	the wsdl2uddi 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 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(authInfo));
      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 HPE SOA Registry Foundation 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());
}
```

Building and Running Demos

This section shows, how to build and run the HPE SOA Registry Foundation Basic Publishing demo set. Let's continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation is up and running.
2. Change your working directory to

Windows:	%REGISTRY_HOME%\demos\security\acl\src\demo\uddi\acl\Find.java
UNIX:	$REGISTRY_HOME/demos/security/acl/src/demo/uddi/acl/Find.java

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...
**********************************************************************
***  HPE  SOA 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</wsdlLocat
ion>
     </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.

WSDL2UDDI v3

The HPE SOA Registry Foundation WSDL2UDDI demo set is used to demonstrate the HPE SOA Registry Foundation WSDL2UDDI application programming interface's capabilities and to show how to use this API. The HPE SOA Registry Foundation WSDL2UDDI demos cover the UDDI Version 3.01 Specification. You will learn how to query and publish a WSDL to a UDDI registry over a SOAP interface.

The HPE SOA Registry Foundation 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.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your HPE SOA Registry Foundation 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 HPE SOA Registry Foundation 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

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://localhost:8080/uddi/wsdl2uddi`	the wsdl2uddi 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 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/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 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 HPE SOA Registry Foundation 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());
}
```

Building and Running Demos

This section shows, how to build and run the HPE SOA Registry Foundation Basic Publishing demo set. Let's continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
**********************************************************************
***HPE  SOA 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</wsdlLocat
ion>
     </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://systinet.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://systinet.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.

XSD2UDDI

The HPE SOA Registry Foundation XSD2UDDI demo set demonstrates the HPE SOA Registry Foundation 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

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the HPE SOA Registry Foundation and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the HPE SOA Registry Foundation's demos, your registry must be running. To start the HPE SOA Registry Foundation, 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 HPE SOA Registry Foundation 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

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

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 importMethod, 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
(publishingMethod));
      publish.setImportPolicy(ImportPublishPolicy.getImportPublishPolicy
(importMethod));
      publish.setImportPublishingMethod
(ImportPublishingMethod.getImportPublishingMethod(importPolicy));

      publish.setContentPolicy(ContentPublishPolicy.getContentPublishPolicy
(contentPolicy));
      publish.setContentPublishingMethod
(ContentPublishingMethod.getContentPublishingMethod(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;
}
```

Building and Running Demos

This section shows, how to build and run the HPE SOA Registry Foundation XSD2UDDI demo set. Let us continue with our PublishXsd demo.

1. Be sure that the demos are properly configured and the HPE SOA Registry Foundation 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...
   **********************************************************************
   *** HPE SOA 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
```

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Product Documentation (SOA Registry Foundation 10.04)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to clouddocs@hpe.com.

We appreciate your feedback!