JBoss Enterprise SOA Platform 5

JUDDI Registry Guide

This is a guide to using the service registry for developers.

Edition 5.3.1

Last Updated: 2017-10-27

JBoss Enterprise SOA Platform 5 JUDDI Registry Guide

This is a guide to using the service registry for developers.

Edition 5.3.1

David Le Sage
Red Hat Engineering Content Services

Suzanne Dorfield
Red Hat Engineering Content Services

Legal Notice

Copyright © 2013 Red Hat, Inc.

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This document teaches developers about the service registry.

Table of Contents

PREFACE 2
CHAPTER 1. PREFACE 3
CHAPTER 2. INTRODUCING THE JUDDI REGISTRY 8
CHAPTER 3. SETTING UP YOUR ROOT SEED DATA 14
CHAPTER 4. SIMPLE PUBLISHING VIA THE API 21
CHAPTER 5. SUBSCRIPTIONS 27
CHAPTER 6. SUPPORT FOR M-BEANS 34
CHAPTER 7. USING THE JUDDI CLIENT 35
APPENDIX A. REVISION HISTORY 56

PREFACE

CHAPTER 1. PREFACE

1.1. BUSINESS INTEGRATION

In order to provide a dynamic and competitive business infrastructure, it is crucial to have a service-oriented architecture in place that enables your disparate applications and data sources to communicate with each other with minimum overhead.

The JBoss Enterprise SOA Platform is a framework capable of orchestrating business services without the need to constantly reprogram them to fit changes in business processes. By using its business rules and message transformation and routing capabilities, JBoss Enterprise SOA Platform enables you to manipulate business data from multiple sources.

Report a bug

1.2. WHAT IS A SERVICE-ORIENTED ARCHITECTURE?

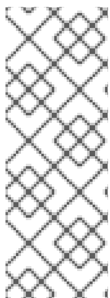
Introduction

A *Service Oriented Architecture* (SOA) is not a single program or technology. Think of it, rather, as a software design paradigm.

As you may already know, a *hardware bus* is a physical connector that ties together multiple systems and subsystems. If you use one, instead of having a large number of point-to-point connectors between pairs of systems, you can simply connect each system to the central bus. An *enterprise service bus* (ESB) does exactly the same thing in software.

The ESB sits in the architectural layer above a messaging system. This messaging system facilitates *asynchronous communications* between services through the ESB. In fact, when you are using an ESB, everything is, conceptually, either a *service* (which, in this context, is your application software) or a *message* being sent between services. The services are listed as connection addresses (known as *end-points references*.)

It is important to note that, in this context, a "service" is not necessarily always a web service. Other types of applications, using such transports as File Transfer Protocol and the Java Message Service, can also be "services."



NOTE

At this point, you may be wondering if an enterprise service bus is the same thing as a service-oriented architecture. The answer is, "Not exactly." An ESB does not form a service-oriented architecture of itself. Rather, it provides many of the tools than can be used to build one. In particular, it facilitates the *loose-coupling* and *asynchronous message passing* needed by a SOA. Always think of a SOA as being more than just software: it is a series of principles, patterns and best practices.

Report a bug

1.3. KEY POINTS OF A SERVICE-ORIENTED ARCHITECTURE

These are the key components of a service-oriented architecture:

1. the *messages* being exchanged
2. the *agents* that act as service requesters and providers
3. the *shared transport mechanisms* that allow the messages to flow back and forth.

Report a bug

1.4. WHAT IS THE JBOSS ENTERPRISE SOA PLATFORM?

The JBoss Enterprise SOA Platform is a framework for developing enterprise application integration (EAI) and service-oriented architecture (SOA) solutions. It is made up of an enterprise service bus (JBoss ESB) and some business process automation infrastructure. It allows you to build, deploy, integrate and orchestrate business services.

Report a bug

1.5. THE SERVICE-ORIENTED ARCHITECTURE PARADIGM

The service-oriented architecture (SOA) consists of three roles: requester, provider, and broker.

Service Provider

A service provider allows access to services, creates a description of a service and publishes it to the service broker.

Service Requester

A service requester is responsible for discovering a service by searching through the service descriptions given by the service broker. A requester is also responsible for binding to services provided by the service provider.

Service Broker

A service broker hosts a registry of service descriptions. It is responsible for linking a requester to a service provider.

Report a bug

1.6. CORE AND COMPONENTS

The JBoss Enterprise SOA Platform provides a comprehensive server for your data integration needs. On a basic level, it is capable of updating business rules and routing messages through an Enterprise Service Bus.

The heart of the JBoss Enterprise SOA Platform is the Enterprise Service Bus. JBoss (ESB) creates an environment for sending and receiving messages. It is able to apply “actions” to messages to transform them and route them between services.

There are a number of components that make up the JBoss Enterprise SOA Platform. Along with the ESB, there is a registry (jUDDI), transformation engine (Smooks), message queue (HornetQ) and BPEL engine (Riftsaw).

Report a bug

1.7. COMPONENTS OF THE JBOSS ENTERPRISE SOA PLATFORM

- A full Java EE 5-compliant application server (the JBoss Enterprise Application Platform)
- an enterprise service bus (JBoss ESB)
- a business process management system (jBPM)
- a business rules engine (JBoss Rules)
- support for the optional JBoss Enterprise Data Services (EDS) product.

Report a bug

1.8. JBOSS ENTERPRISE SOA PLATFORM FEATURES

The JBoss Enterprise Service Bus (ESB)

The ESB sends messages between services and transforms them so that they can be processed by different types of systems.

Business Process Execution Language (BPEL)

You can use web services to orchestrate business rules using this language. It is included with SOA for the simple execution of business process instructions.

Java Universal Description, Discovery and Integration (jUDDI)

This is the default service registry in SOA. It is where all the information pertaining to services on the ESB are stored.

Smooks

This transformation engine can be used in conjunction with SOA to process messages. It can also be used to split messages and send them to the correct destination.

JBoss Rules

This is the rules engine that is packaged with SOA. It can infer data from the messages it receives to determine which actions need to be performed.

Report a bug

1.9. FEATURES OF THE JBOSS ENTERPRISE SOA PLATFORM'S JBOSSESB COMPONENT

The JBoss Enterprise SOA Platform's JBossESB component supports:

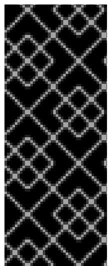
- Multiple transports and protocols
- A listener-action model (so that you can loosely-couple services together)

- Content-based routing (through the JBoss Rules engine, XPath, Regex and Smooks)
- Integration with the JBoss Business Process Manager (jBPM) in order to provide service orchestration functionality
- Integration with JBoss Rules in order to provide business rules development functionality.
- Integration with a BPEL engine.

Furthermore, the ESB allows you to integrate legacy systems in new deployments and have them communicate either synchronously or asynchronously.

In addition, the enterprise service bus provides an infrastructure and set of tools that can:

- Be configured to work with a wide variety of transport mechanisms (such as e-mail and JMS),
- Be used as a general-purpose object repository,
- Allow you to implement pluggable data transformation mechanisms,
- Support logging of interactions.



IMPORTANT

There are two trees within the source code: `org.jboss.internal.soa.esb` and `org.jboss.soa.esb`. Use the contents of the `org.jboss.internal.soa.esb` package sparingly because they are subject to change without notice. By contrast, everything within the `org.jboss.soa.esb` package is covered by Red Hat's deprecation policy.

Report a bug

1.10. TASK MANAGEMENT

JBoss SOA simplifies tasks by designating tasks to be performed universally across all systems it affects. This means that the user does not have to configure the task to run separately on each terminal. Users can connect systems easily by using web services.

Businesses can save time and money by using JBoss SOA to delegate their transactions once across their networks instead of multiple times for each machine. This also decreases the chance of errors ocurring.

Report a bug

1.11. INTEGRATION USE CASE

Acme Equity is a large financial service. The company possesses many databases and systems. Some are older, COBOL-based legacy systems and some are databases obtained through the acquisition of smaller companies in recent years. It is challenging and expensive to integrate these databases as business rules frequently change. The company wants to develop a new series of client-facing e-commerce websites, but these may not synchronise well with the existing systems as they currently stand.

The company wants an inexpensive solution but one that will adhere to the strict regulations and security requirements of the financial sector. What the company does not want to do is to have to write and maintain “glue code” to connect their legacy databases and systems.

The JBoss Enterprise SOA Platform was selected as a middleware layer to integrate these legacy systems with the new customer websites. It provides a bridge between front-end and back-end systems. Business rules implemented with the JBoss Enterprise SOA Platform can be updated quickly and easily.

As a result, older systems can now synchronise with newer ones due to the unifying methods of SOA. There are no bottlenecks, even with tens of thousands of transactions per month. Various integration types, such as XML, JMS and FTP, are used to move data between systems. Any one of a number of enterprise-standard messaging systems can be plugged into JBoss Enterprise SOA Platform providing further flexibility.

An additional benefit is that the system can now be scaled upwards easily as more servers and databases are added to the existing infrastructure.

Report a bug

1.12. UTILISING THE JBOSS ENTERPRISE SOA PLATFORM IN A BUSINESS ENVIRONMENT

Cost reduction can be achieved due to the implementation of services that can quickly communicate with each other with less chance of error messages occurring. Through enhanced productivity and sourcing options, ongoing costs can be reduced.

Information and business processes can be shared faster because of the increased connectivity. This is enhanced by web services, which can be used to connect clients easily.

Legacy systems can be used in conjunction with the web services to allow different systems to "speak" the same language. This reduces the amount of upgrades and custom code required to make systems synchronise.

Report a bug

CHAPTER 2. INTRODUCING THE JUDDI REGISTRY

2.1. INTENDED AUDIENCE

This book has been written for experienced Java developers wishing to learn how to work with the JBoss Enterprise Service Platform's service registry.

Report a bug

2.2. AIM OF THIS BOOK

Read this book in order to learn how to develop for and operate the jUDDI Service Registry. Note that this Guide does not show you how to initially configure and secure the product for use in a production environment. For that information, refer to the full *Installation and Configuration Guide*.

Report a bug

2.3. BACK UP YOUR DATA

WARNING

Red Hat recommends that you back up your system settings and data before undertaking any of the configuration tasks mentioned in this book.

Report a bug

2.4. JUDDI REGISTRY

The jUDDI (Java Universal Description, Discovery and Integration) Registry is a core component of the JBoss Enterprise SOA Platform. It is the product's default service registry and comes included as part of the product. In it are stored the addresses (end-point references) of all the services connected to the Enterprise Service Bus. It was implemented in JAXR and conforms to the UDDI specifications.

Report a bug

2.5. JUDDI AND THE JBOSS ENTERPRISE SOA PLATFORM

jUDDI and the JBoss Enterprise SOA Platform

The JBoss Enterprise SOA Platform product includes a pre-configured installation of a jUDDI registry. You can use a specific API to access this registry through your custom client. However, any custom client that you build will not covered by your SOA Platform support agreement. You can access the full set of jUDDI examples, documentation and APIs from: http://juddi.apache.org/.

Report a bug

2.6. OTHER SUPPORTED SERVICE REGISTRIES

The JBoss Enterprise SOA Platform also supports these other UDDI registries:

- SOA Software SMS
- HP Systinet

Report a bug

2.7. SERVICE REGISTRY

A service registry is a central database that stores information about services, notably their end-point references. The default service registry for the JBoss Enterprise SOA Platform is jUDDI (Java Universal Description, Discovery and Integration). Most service registries are designed to adhere to the Universal Description, Discovery and Integration (UDDI) specifications.

From a business analyst’s perspective, the registry is similar to an Internet search engine, albeit one designed to find web services instead of web pages. From a developer's perspective, the registry is used to discover and publish services that match various criteria.

In many ways, the Registry Service can be considered to be the "heart" of the JBoss Enterprise SOA Platform. Services can "self-publish" their end-point references to the Registry when they are activated and then remove them when they are taken out of service. Consumers can consult the registry in order to determine which end-point reference is needed for the current service task.

Report a bug

2.8. SERVICE PROVIDER

A service provider gives access to services, creates descriptions of them and publishes them to the service broker.

Report a bug

2.9. SERVICE BROKER

A service broker hosts the registry of service descriptions. It is responsible for linking a service requester to a service provider.

Report a bug

2.10. SERVICE REQUESTER

A service requester is responsible for discovering a service. It does so by searching through the service descriptions given to it by the service broker. A requester is also responsible for binding together services obtained from the service provider.

Report a bug

2.11. WEB SERVICE

A web service is a way of making two applications communicate over the web. A web service consists of a set of tools to achieve this aim. There are two types of web service: REST-compliant ones, (the purpose of which is to manipulate XML representations of web resources) and arbitrary Web services (through which the service can expose any operation).

Report a bug

2.12. WEB SERVICE END-POINT

A web service end-point is software that implements a web service. They are used to implement message-based communication between web services in a service-oriented architectural environment.

The external applications to which these registry entries point can include .NET programs, other external Java-based application servers and LAMP software bundles.

Report a bug

2.13. WEB SERVICES DESCRIPTION LANGUAGE (WSDL)

The Web Services Description Language (WSDL) is an XML-based language that is used to define Web service interfaces. An application that consumes a Web service parses the service's WSDL document to discover the:

- location of the service
- the operations that the service supports
- the protocol bindings the service supports (SOAP, HTTP, etc)
- access procedure

For each operation, the WSDL describes the interface format to which the client must adhere.

Report a bug

2.14. UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION (UDDI) REGISTRY

The Universal Description, Discovery and Integration Registry (UDDI) is a directory for web services. Use it to locate services by running queries through it at design- or run-time. Within an UDDI Registry, information is categorized in Pages. UDDI creates a standard interoperable platform that enables companies and applications to quickly, easily, and dynamically find and use Web services over the Internet. UDDI also allows operational registries to be maintained for different purposes in different contexts.

The UDDI also allows providers to publish descriptions of their services. The typical UDDI Registry will contain a uniform resource locator (URL) that points to both the WSDL document for the web services and the contact information for the service provider.

A business publishes services to the UDDI registry. A client looks up the service in the registry and receives service binding information. The client then uses the binding information to invoke the service. The UDDI APIs are SOAP-based for interoperability reasons.

Report a bug

2.15. UDDI APPLICATION PROGRAMMING INTERFACES

The UDDI v3 specification defines nine APIs:

UDDI_Security_PortType

This defines the API to obtain a security token. With a valid security token a publisher can publish to the registry. A security token can be used for the entire session.

UDDI_Publication_PortType

This defines the API to publish business and service information to the UDDI registry.

UDDI_Inquiry_PortType

This defines the API to query the UDDI registry. This API does not normally require a security token.

UDDI_CustodyTransfer_PortType

This API can be used to transfer the custody of a business from one UDDI node to another.

UDDI_Subscription_PortType

This defines the API to register for updates on a particular business of service.

UDDI_SubscriptionListener_PortType

This defines the API a client must implement to receive subscription notifications from a UDDI node.

UDDI_Replication_PortType

This defines the API to replicate registry data between UDDI nodes.

UDDI_ValueSetValidation_PortType

This is used by nodes to allow external providers of value set the validation. Web services to assess whether keyedReferences or keyedReferenceGroups are valid.

UDDI_ValueSetCaching_PortType

UDDI nodes may perform validation of publisher references themselves using the cached values obtained from such a Web service.

Report a bug

2.16. UDDI PAGE TYPES

Green Pages

Green Pages provide information that enables you to bind a client to the service being provided.

Yellow Pages

Yellow Pages are used to categorize businesses based upon the industries to which they belong.

White Pages

White Pages contain general information, such as the name, address and other contact details for the company providing the service.

Report a bug

2.17. THE SERVICE REGISTRY AND THE JBOSS ENTERPRISE SOA PLATFORM

The Service Registry is a key part of the JBoss Enterprise SOA Platform. When you deploy services to SOA Platform's ESB, their end-point references are stored in it.

Report a bug

2.18. JUDDI AND THE ESB

The JBoss Enterprise Service Bus directs all interaction with the Registry through the registry interface, the default version of which uses Apache Scout.

Report a bug

2.19. HOW THE REGISTRY WORKS

1. The JBoss Enterprise Service Bus funnels all interaction with the Registry through the registry interface.
2. It then calls a JAXR implementation of this interface.
3. The JAXR API needs to utilize a JAXR implementation. (By default, this is Apache Scout.)
4. Apache Scout, in turn, calls the Registry.

Report a bug

2.20. APACHE SCOUT

Apache Scout is an open source implementation of JAXR, created by the Apache Project.

There are currently four implementations of the **`org.jboss.soa.esb.scout.proxy.transportClass`** class, one each for SOAP, SAAJ, RMI and Embedded Java (Local).

Report a bug

2.21. JAVA API FOR XML REGISTRIES (JAXR)

Java API for XML Registries (JAXR) is an API that provides a standard way to develop for service registries.

Report a bug

2.22. REGISTRY INTERFACE

The Registry Interface is the means by which the ESB communicates with the Service Registry.

Report a bug

2.23. VARIABLE NAME: SOA_ROOT DIRECTORY

SOA Root (often written as SOA_ROOT) is the term given to the directory that contains the application server files. In the standard version of the JBoss Enterprise SOA Platform package, SOA root is the **`jboss-soa-p-5`** directory. In the Standalone edition, though, it is the **`jboss-soa-p-standalone-5`** directory.

Throughout the documentation, this directory is frequently referred to as **SOA_ROOT**. Substitute either **`jboss-soa-p-5`** or **`jboss-soa-p-standalone-5`** as appropriate whenever you see this name.

Report a bug

2.24. VARIABLE NAME: PROFILE

PROFILE can be any one of the server profiles that come with the JBoss Enterprise SOA Platform product: default, production, all, minimal, standard or web. Substitute one of these that you are using whenever you see "PROFILE" in a file path in this documentation.

Report a bug

CHAPTER 3. SETTING UP YOUR ROOT SEED DATA

3.1. SERVICE REGISTRY PUBLISHERS

Publisher is an identity management system for the Service Registry. Two publishers are installed by default:

root

The root publisher is the owner of the root partition and UDDI services (such as **inquiry** and **publication**). Each registry need to have a *root publisher*. There can only be one root publisher per node. The jUDDI Registry ships some default seed data for the root account, found in the **juddi-core-3.x.jar** file's **juddi_install_data/** directory.

uddi

uddi owns all of the other seed data (such as the pre-defined **tModels**.

Report a bug

3.2. SEED DATA

Seed data is some initial data you feed into the system. It functions as a template.

Report a bug

3.3. SERVICE REGISTRY SEED DATA FILES

For each publisher there are four seed data files. These are read the first time you start the jUDDI:

- <publisher>_Publisher.xml
- <publisher>_tModelKeyGen.xml
- <publisher>_BusinessEntity.xml
- <publisher>_tModels.xml

Report a bug

3.4. ROOT_PUBLISHER.XML

The contents of the **root_Publisher.xml** data seed file looks like this:

```
<publisher xmlns="urn:juddi-apache-org:api_v3" authorizedName="root">
    <publisherName>root publisher</publisherName>
    <isAdmin>true</isAdmin>
</publisher>
```

Report a bug

3.5. KEY GENERATOR SCHEMA

As its name implies, a key generator schema generates keys. Using it allows one to avoid the risk of creating keys that are identical to those generated by other publishers. Each publisher should always have its own key generator schema. The root publisher can only own one KeyGenerator while but any other publisher can own more than one.

Report a bug

3.6. TMODEL

A tModel ("technical model") is an UDDI data structure that represents one of the services registered in the registry. They are are a catch-all structure that can do anything from categorising one of the main entities, describing the technical details of a binding (such as a protocol or a transports), to registering a key partition.

Report a bug

3.7. ROOT_TMODELKEYGEN.XML

The **tModel Key Generator** is defined in the **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/juddi_custom_install_data/root_tModelKeyGen.xml** file:

```
<tModel tModelKey="uddi:juddi.apache.org:keygenerator" xmlns="urn:uddi-org:api_v3">
    <name>uddi-org:keyGenerator</name>
    <description>Root domain key generator</description>
    <overviewDoc>
        <overviewURL useType="text">
        http://uddi.org/pubs/uddi_v3.htm#keyGen
        </overviewURL>
    </overviewDoc>
    <categoryBag>
        <keyedReference tModelKey="uddi:uddi.org:categorization:types"
            keyName="uddi-org:types:keyGenerator"
            keyValue="keyGenerator" />
    </categoryBag>
</tModel>
```

This means that the keys used by the **root publisher** need to be in the following form: **uddi:juddi.apache.org:<text-of-choice>**.

Report a bug

3.8. CREATE A KEY GENERATOR SCHEMA FOR A PUBLISHER

Procedure 3.1. Task

1. Each publisher need to define its own KenGenerator tModel. The tModel Key Generator is defined in the root_tModelKeyGen.xml file:

```
<tModel tModelKey="uddi:juddi.apache.org:keygenerator"
xmlns="urn:uddi-org:api_v3">
    <name>uddi-org:keyGenerator</name>
    <description>Root domain key generator</description>
    <overviewDoc>
        <overviewURL useType="text">
        http://uddi.org/pubs/uddi_v3.htm#keyGen
        </overviewURL>
    </overviewDoc>
    <categoryBag>
        <keyedReference
tModelKey="uddi:uddi.org:categorization:types"
            keyName="uddi-org:types:keyGenerator"
            keyValue="keyGenerator" />
    </categoryBag>
</tModel>
```

2. The keys used by the root publisher need to be in the following form: **`uddi:juddi.apache.org:<text-of-choice>`**. If you use another format, you will see an illegal key error.

WARNING

Do not share KeyGenerators.

Report a bug

3.9. VIEW A PUBLISHER VIA MORE THAN JUST ONE KEYGENERATOR TMODEL

Procedure 3.2. Task

- Use the **`<publisher>_tModels.xml`** file.

Report a bug

3.10. CONFIGURE BUSINESS AND SERVICE DATA USING A PUBLISHER

Procedure 3.3. Task

1. Open the **<publisher>_BusinessEntity.xml** file in a text editor: **vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/juddi_custom_install_data/root_BusinessEntity.xml**

2. Edit the file to specify your services. Here is an example:

```
<businessEntity xmlns="urn:uddi-org:api_v3"
    xmlns:xml="http://www.w3.org/XML/1998/namespace"
    businessKey="uddi:juddi.apache.org:businesses-asf">
    <!-- Change the name field to represent the name of your
registry -->
    <name xml:lang="en">An Apache jUDDI Node</name>
        <!-- Change the description field to provided
        a brief description of your registry -->
        <description xml:lang="en">
            This is a UDDI v3 registry node as implemented by Apache
jUDDI.
        </description>
        <discoveryURLs>
        <!-- This discovery URL should point to the home
installation URL of jUDDI -->
        <discoveryURL useType="home">
            http://${juddi.server.baseurl}/juddiv3
        </discoveryURL>
    </discoveryURLs>
    <categoryBag>
        <keyedReference
tModelKey="uddi:uddi.org:categorization:nodes" keyValue="node" />
    </categoryBag>

    <businessServices>
    <!-- As mentioned above, you may want to provide user-defined
keys for
    these (and the services/bindingTemplates below.  Services that
you
    don't intend to support should be removed entirely -->
        <businessService serviceKey="uddi:juddi.apache.org:services-
inquiry"
            businessKey="uddi:juddi.apache.org:businesses-asf">
            <name xml:lang="en">UDDI Inquiry Service</name>
            <description xml:lang="en">Web Service supporting UDDI
Inquiry API</description>
            <bindingTemplates>
                <bindingTemplate
bindingKey="uddi:juddi.apache.org:servicebindings-inquiry-ws"
                    serviceKey="uddi:juddi.apache.org:services-
inquiry">
                    <description>UDDI Inquiry API V3</description>
                    <!-- This should be changed to the WSDL URL of
the inquiry API.
                    An access point inside a bindingTemplate will
be found for every service
                    in this file.  They all must point to their
API's WSDL URL -->
                    <accessPoint useType="wsdlDeployment">
```

```
http://${juddi.server.baseurl}/juddiv3/services/inquiry?wsdl
                    </accessPoint>
                    <tModelInstanceDetails>
                        <tModelInstanceInfo
tModelKey="uddi:uddi.org:v3_inquiry">
                            <instanceDetails>
                                <instanceParms>
                                <![CDATA[
                                <?xml version="1.0" encoding="utf-
8" ?>
                                <UDDIinstanceParmsContainer
                                    xmlns="urn:uddi-
org:policy_v3_instanceParms">
                                    <defaultSortOrder>

uddi:uddi.org:sortorder:binarysort
                                    </defaultSortOrder>
                                </UDDIinstanceParmsContainer>
                                ]]>
                                </instanceParms>
                            </instanceDetails>
                        </tModelInstanceInfo>
                    </tModelInstanceDetails>
                    <categoryBag>
                        <keyedReference keyName="uddi-
org:types:wsdl" keyValue="wsdlDeployment"

tModelKey="uddi:uddi.org:categorization:types"/>
                    </categoryBag>
                </bindingTemplate>
            </bindingTemplates>
        </businessService>
        <businessService serviceKey="uddi:juddi.apache.org:services-
publish"
            businessKey="uddi:juddi.apache.org:businesses-asf">
            <name xml:lang="en">UDDI Publish Service</name>
            ...........
        </businessService>
    </businessServices>
</businessEntity>
```

3. Save the file and exit.

WARNING

The seeding process only commences if there are no `publishers` in the database, unless you set juddi.seed.always to `true`. (It will then re-apply all files with the exception of the root data files.)

This can lead to data loss, as any information added to entities that are re-seeded will be erased. This is because that data is not merged.

Report a bug

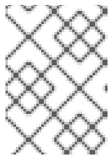
3.11. CONFIGURE TOKENS IN THE ROOT_BUSINESSENTITY.XML FILE

Procedure 3.4. Task

1. To set the value of the tokens in the root_**`BusinessEntity.xml`** file (**`${juddi.server.baseurl}`**), modify the **`SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml`** file. Open the **`<publisher>_BusinessEntity.xml`** file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/juddi_custom_install_data/root_BusinessEntity.xml`**

2. Here you can set keys and security tokens. Modify these sections to suit your needs.

3. Save the file and exit.



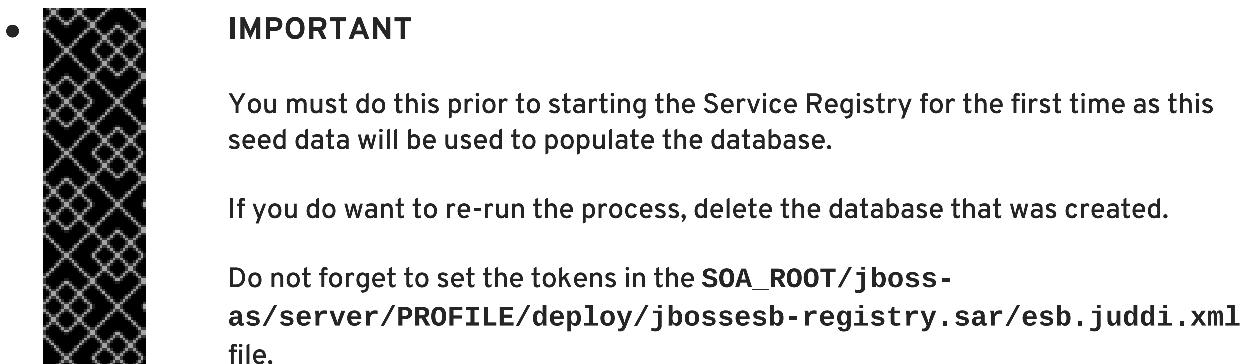
NOTE

The value substitution takes place at run-time, so different nodes can substitute their own values if need be.

Report a bug

3.12. CUSTOMISE THE SEED DATA

Procedure 3.5. Task

- 

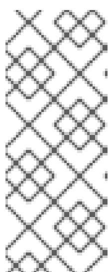
  IMPORTANT

  You must do this prior to starting the Service Registry for the first time as this seed data will be used to populate the database.

  If you do want to re-run the process, delete the database that was created.

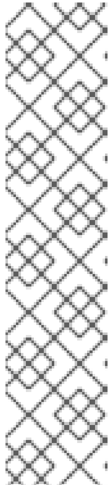
  Do not forget to set the tokens in the **`SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml`** file.

  Add your custom seed data to **`juddiv3.war/WEB-INF/classes/juddi_custom_install_data/`** directory.



NOTE

To use the Sales Seed Data rename the directory found in the **`SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar//WEB-INF/classes/`** by running this command: **`mv RENAME4Sales_juddi_custom_install_data juddi_custom_install_data`**



NOTE

If your root publisher is not called **`root`** you will need to set the **`SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml`** file's juddi.root.publisher property to something other than this:

```
juddi.root.publisher=root
```

Properties are set in this way:

```
<entry key="juddi.root.publisher">root</entry>
```

Report a bug

CHAPTER 4. SIMPLE PUBLISHING VIA THE API

4.1. UDDI DATA MODEL

UDDI organises data into the following hierarchy:

Business entities

Business entities are at the top of the pyramid. They contain business services.

Business services

Business services contain end-point references.

To model your data in a form consistent with this hierarchy, you must first create a business entity before you can publish any services and you must create a business service before you can publish any binding templates.

Report a bug

4.2. BUSINESS ENTITY

Business Entities are the organisational units responsible for the services. They contain information (consisting of a description and contact information) about the parties who are publishing services. How you choose to use the business entities depends on your situation. If you have a small company, you may only need one entity. If you are a larger company with multiple departments, you may want to have one entity per department.

Another possibility is to have one overarching entity with multiple child entities representing the departments. To do so, you create relationships between the entities by using publisher assertions.

Report a bug

4.3. BUSINESS SERVICE

Business services represent units of functionality that are consumed by clients. In UDDI, a service mainly consists of descriptive information such as a name, a description and categories. The technical details about the service are actually contained in its binding templates instead.

Report a bug

4.4. END-POINT REFERENCE

An end-point reference (EPR) contains the address information and technical specifications for a service. Indeed, all ESB-aware services are identified using end-point references. It is through these references that services are contacted. The are stored in the registry. Services add their end-point references to the registry when they are launched and should automatically remove them when they terminate. A service may have multiple end-point references. End-point references are also known as binding templates.

End-point references can contain links to the tModels designating the interface specifications for a particular service.

Report a bug

4.5. UDDI AND THE JUDDI API

In order to use the UDDI API, you must understand how it is structured. It is divided into several `sets`. Each `set` represents a specific area of functionality:

- `Inquiry` handles queries to the Registry for details about entities.
- `Publication` handles the registration of entities.
- `Security` is an open-ended specification that handles authentication.
- `Custody and Ownership Transfer`handles the transference of ownership of entities.
- `Subscription` allows clients to retrieve information about entities using a subscription format.
- `Subscription Listener` is a client API for accepting subscription results.
- `Value Set (Validation and Caching)` validates keyed reference values. (This set is not yet available for the jUDDI.)
- `Replication` allows you to federate data between registry nodes. (This set is not yet available for the jUDDI.)

The `sets` you will use most frequently are `Inquiry`, `Publication` and `Security`. They provide the standard functions you need for interacting with the Registry.

The jUDDI server implements each of these `sets` as a JAX-WS-compliant web service. (Each method defined in a set is simply a method in the corresponding web service.)

The jUDDI provides a client module that uses a transport class to define how each call is to be made. (The default transport uses JAX-WS but there are several alternative ways to make calls to the API.)

Finally, the jUDDI also defines its own API set. This API set contains methods that deal with handling Publishers and other maintenance functions, mostly related to the jUDDI's subscription model. This API set is specific to jUDDI and does not conform to the UDDI specification.

Report a bug

4.6. JUDDI ADDITIONS TO THE UDDI MODEL

Introduction

The jUDDI Registry's implementation provides the data model with some additional structure beyond that described in the specification. Prime amongst this is the concept of the publisher.

The UDDI specification mentions the ownership of the entities that are published within the registry, but makes no mention of how this ownership should be implemented. It is left up to the individual implementation.

Publisher is an identity management system that provides this functionality for the jUDDI Registry.

Before assets can be added to the jUDDI, the Registry authenticates the publisher by allowing it to retrieve an authorisation token. This token is attached to subsequent **publish** calls to assign ownership to the published entities.

You can save a publisher to the jUDDI Registry by using the latter's custom API.

The Publisher is an out-of-the-box implementation of an identity management system. jUDDI also allows for integration into your own identity management system, circumventing the Publisher entirely if desired. In this guide, we will be using the simple out-of-the-box Publisher solution.

Report a bug

4.7. TUTORIAL: USE THE SERVICE REGISTRY FOR THE FIRST TIME

This tutorial steps you through the a simple example. In this example, after retrieving an authentication token, a Publisher, a BusinessEntity and a BusinessService are registered in the jUDDI.

Procedure 4.1. Task

1. Instantiate the **SimplePublish** class. Here is the constructor:

```
      public SimplePublish()
{
    try
    {
        String clazz =
UDDIClientContainer.getUDDIClerkManager(null).

getClientConfig().getUDDINode("default").getProxyTransport();
        Class<?> transportClass = ClassUtil.forName(clazz,
Transport.class);
        if (transportClass!=null)
        {
            Transport transport = (Transport) transportClass.

getConstructor(String.class).newInstance("default");

            security = transport.getUDDISecurityService();
            juddiApi = transport.getJUDDIApiService();
            publish = transport.getUDDIPublishService();
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

This constructor uses the jUDDI client API to retrieve the transport from the default node. In this example, it is simply retrieving the default client transport class which is designed to make UDDI calls out using JAX-WS web services.

2. Obtain the three API sets needed for the example:

   - the **Security** API set, so you can obtain authorisation tokens,
   - the proprietary **jUDDI** API set so you can save a publisher,
   - the **Publication** API set so that you can register entities in the jUDDI.

   Here are the first few lines of the **publish** method:

```
 // Setting up the values to get an authentication token for the
'root' user
// ('root' user has admin privileges and can save other publishers).
GetAuthToken getAuthTokenRoot = new GetAuthToken();
getAuthTokenRoot.setUserID("root");
getAuthTokenRoot.setCred("");

// Making API call that retrieves the authentication token for the
'root' user.
AuthToken rootAuthToken = security.getAuthToken(getAuthTokenRoot);
System.out.println ("root AUTHTOKEN = " +
rootAuthToken.getAuthInfo());
```

   This code simply retrieves the authorisation token for the *root* user.

3. Add a publisher:

```
// Creating a new publisher that we will use to publish our entities
to.
Publisher p = new Publisher();
p.setAuthorizedName("my-publisher");
p.setPublisherName("My Publisher");

// Adding the publisher to the "save" structure, using the 'root'
user authentication
// info and saving away.
SavePublisher sp = new SavePublisher();
sp.getPublisher().add(p);
sp.setAuthInfo(rootAuthToken.getAuthInfo());
juddiApi.savePublisher(sp);
```

   The code above uses the jUDDI API to save a publisher with authorised name, my-publisher. Note that the authorisation token for the root user is utilised.

4. Obtain the authorisation token for this new publisher:

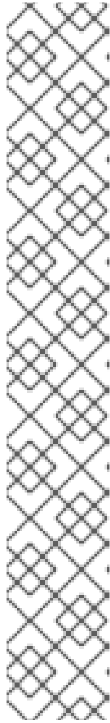
```
 // Our publisher is now saved, so now we want to retrieve its
authentication token
GetAuthToken getAuthTokenMyPub = new GetAuthToken();
getAuthTokenMyPub.setUserID("my-publisher");
getAuthTokenMyPub.setCred("");
```

```
AuthToken myPubAuthToken = security.getAuthToken(getAuthTokenMyPub);
System.out.println ("myPub AUTHTOKEN = " +
myPubAuthToken.getAuthInfo());
```

Note that no credentials have been set on either of the authorisation calls. This is because, by using the default authenticator, you do not have to provide any.

5. Publish:

```
// Creating the parent business entity that will contain our service.
BusinessEntity myBusEntity = new BusinessEntity();
Name myBusName = new Name();
myBusName.setValue("My Business");
myBusEntity.getName().add(myBusName);

// Adding the business entity to the "save" structure, using our
publisher's
// authentication info and saving away.
SaveBusiness sb = new SaveBusiness();
sb.getBusinessEntity().add(myBusEntity);
sb.setAuthInfo(myPubAuthToken.getAuthInfo());
BusinessDetail bd = publish.saveBusiness(sb);
String myBusKey = bd.getBusinessEntity().get(0).getBusinessKey();
System.out.println("myBusiness key:  " + myBusKey);

// Creating a service to save.  Only adding the minimum data: the
parent
// business key retrieved from saving the business above and a single
name.
BusinessService myService = new BusinessService();
myService.setBusinessKey(myBusKey);
Name myServName = new Name();
myServName.setValue("My Service");
myService.getName().add(myServName);
// Add binding templates, etc...

// Adding the service to the "save" structure, using our publisher's
// authentication info and saving away.
SaveService ss = new SaveService();
ss.getBusinessService().add(myService);
ss.setAuthInfo(myPubAuthToken.getAuthInfo());
ServiceDetail sd = publish.saveService(ss);
String myServKey = sd.getBusinessService().get(0).getServiceKey();
System.out.println("myService key:  " + myServKey);
```



NOTE

There are some important things to note with regard to the use of entity keys. Version three of the specification allows for publishers to create their own keys but also instructs implementers to have default methods.

Red Hat has chosen the default implementation approach by leaving each entity's key field blank in the `save` call. The jUDDI's default key generator simply takes the node's partition and appends a GUID to it.

In a default installation, it will look something like this: `uddi:juddi.apache.org:<GUID>`

(You can, of course, customise all of this if you wish.)

The second important point is that when the BusinessService is saved, its parent business key (retrieved from previous call saving the business) is set explicitly. This is a necessary step when the service is saved in an independent call like this because, otherwise, you would encounter an error as the jUDDI will not know where to find the parent entity.

Result

In this example you have created and saved a BusinessEntity and then created and saved a BusinessService. You have added the bare minimum of data to each entity (and, in fact, have not added any BindingTemplates to the service).

In a real life scenario, you will obviously want to fill out each structure with greater information, particularly with regard to services. However, this will prove a useful starting point.

Report a bug

CHAPTER 5. SUBSCRIPTIONS

5.1. SUBSCRIPTION API

The **subscription** API allows you to cross-register services on different service registries and keep them up-to-date by sending out notifications as the registry information in the parent registry changes. This is helpful if you have a large or complex organisation and decide to run different service registries for different departments, with some services being shared between them.

Report a bug

5.2. SUBSCRIPTION TYPES

There are two type of subscriptions. Each has a different notification system:

asynchronous

With this, you can save a subscription, and receive updates on a set schedule. It requires a listener service to be installed on the node to which the notifications will be sent.

synchronous

With this, you can save a subscription and invoke **get_Subscription** to obtain a synchronous reply.

Report a bug

5.3. TUTORIAL: SUBSCRIPTIONS

Prerequisites

- In this example, you will configure nodes for **sales** and **marketing**. To do so, you will need to deploy the Service Registry to two different services first.

Procedure 5.1. Task

1. **Configuring Node One: Sales**
   Create **juddi_custom_install_data** by running these commands:

   cd juddiv3/WEB-INF/classes

   mv RENAME4SALES_juddi_custom_install_data juddi_custom_install_data

2. Open the **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml** file in a text editor and set the following property values (where "sales" is the DNS name of your server):

   ```
   juddi.server.name=sales
   juddi.server.port=8080
   ```

3. Save the file and exit.

4. Start the server: **SOA_ROOT/jboss-as/bin/./run.sh**

5. Open a web browser and go to this address: http://sales:8080/juddiv3. You should see a message that provides you with information about the node.

6. **Configuring Node Two: Marketing**
   Create **juddi_custom_install_data** by running these commands:

   cd juddiv3/WEB-INF/classes

   mv RENAME4MARKETING_juddi_custom_install_data juddi_custom_install_data

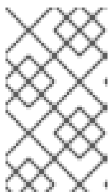
7. Open the **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml** file in a text editor and set the following property values (where "marketing" is the DNS name of your server)

```
juddi.server.name=marketing
juddi.server.port=8080
```

8. Save the file and exit.

9. Start the server: **SOA_ROOT/jboss-as/bin/./run.sh**

10. Open a web browser and go to this address: http://sales:8080/juddiv3. Again you should see a message that provides you with information about the node.



NOTE

Note that the root partition was kept the same, because both the sales and marketing departments are located in the same company, but the Node Id and Name changed to reflect that each node belongs to a different department.

11. Next, replace the sales server's **uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml** file with **uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml.sales**.

12. Open the **uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml** file in a text editor and add the following properties:

```
<name>default</name>
<properties>
    <property name="serverName" value="sales"/>
    <property name="serverPort" value="8080"/>
    <property name="rmiPort" value="1099"/>
</properties>
```

13. Save the file and exit.

14. Launch a web browser and go to this address: http://sales:8080/pluto.

15. Log in with both a user name and password of **sales** and observe what is displayed onscreen.

16. Before logging into the marketing portal, replace marketing's **uddi-portlet.war/WEB-INF/classes/META-INF/uddi.xml** file with **udd-portlet.war/WEB-INF/classes/META-INF/uddi.xml.marketing**.

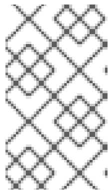
17. Open the **uddi-portlet.war/WEB-INF/classes/META_INF/uddi.xml** file in a text editor and add the following properties:

```
<name>default</name>
<properties>
    <property name="serverName" value="marketing"/>
    <property name="serverPort" value="8080"/>
    <property name="rmiPort" value="1099"/>
</properties>
```

18. Save the file and exit.

19. Launch a web browser and go to this address: http://marketing:8080/pluto.

20. Log in with both a user name and password of **sales** and observe what is displayed onscreen.



NOTE

The subscriptionlistener is owned by the Marketing Node business, not the Root Marketing Node. The Marketing Node Business is managed by the marketing publisher.

Report a bug

5.4. TUTORIAL: DEPLOY THE HELLOSALES SERVICE

Procedure 5.2. Task

1. Open the **juddiv3-samples.war/WEB-INF/classes/META-INF/uddi.xml** file in a text editor and add the following property values to sales:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<uddi>
    <reloadDelay>5000</reloadDelay>
    <manager name="example-manager">
        <nodes>
            <node>
                <name>default</name>
                <description>Sales jUDDI node</description>
                <properties>
                    <property name="serverName"  value="sales"/>
                    <property name="serverPort"  value="8080"/>
                    <property name="keyDomain"
value="sales.apache.org"/>
                    <property name="department"  value="sales" />
                </properties>
                <proxyTransport>
```

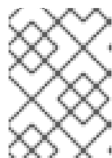
```
org.apache.juddi.v3.client.transport.InVMTransport
                </proxyTransport>
                <custodyTransferUrl>

org.apache.juddi.api.impl.UDDICustodyTransferImpl
                </custodyTransferUrl>

<inquiryUrl>org.apache.juddi.api.impl.UDDIInquiryImpl</inquiryUrl>

<publishUrl>org.apache.juddi.api.impl.UDDIPublicationImpl</publishUr
l>

<securityUrl>org.apache.juddi.api.impl.UDDISecurityImpl</securityUrl
>
                <subscriptionUrl>
                    org.apache.juddi.api.impl.UDDISubscriptionImpl
                </subscriptionUrl>
                <subscriptionListenerUrl>

org.apache.juddi.api.impl.UDDISubscriptionListenerImpl
                </subscriptionListenerUrl>

<juddiApiUrl>org.apache.juddi.api.impl.JUDDIApiImpl</juddiApiUrl>
            </node>
        </nodes>
    </manager>
</uddi>
```

2. Save the file and exit.

3. Now deploy the **`juddiv3-samples.war`** to the sales registry.

   `ant deploy`

   `ant runtest`



NOTE

The HelloSales service is provided in the **`juddiv3-samples.war`** archive file and it is annotated, so it will automatically register itself.

4. Subscribe to the Sales UDDI node's HelloWord service from within the Marketing UDDI node.

Report a bug

5.5. ALLOW USERS TO CREATE THEIR OWN SUBSCRIPTIONS

Prerequisites

- For a user to create and save subscriptions, they must have valid "publisher" log-ins for both the sales and marketing nodes.

- If the marketing publisher is going to create registry objects in the marketing node, the marketing publisher needs to own the **`sales keygenerator tModel`**.

  It is important to understand that the marketing publisher in the marketing registry owns the following **`tModels`**:

```
<save_tModel xmlns="urn:uddi-org:api_v3">

    <tModel tModelKey="uddi:marketing.apache.org:keygenerator"
xmlns="urn:uddi-org:api_v3">
        <name>marketing-apache-org:keyGenerator</name>
        <description>Marketing domain key generator</description>
        <overviewDoc>
            <overviewURL useType="text">
                http://uddi.org/pubs/uddi_v3.htm#keyGen
            </overviewURL>
        </overviewDoc>
        <categoryBag>
            <keyedReference
tModelKey="uddi:uddi.org:categorization:types"
                keyName="uddi-org:types:keyGenerator"
                keyValue="keyGenerator" />
        </categoryBag>
    </tModel>

    <tModel
tModelKey="uddi:marketing.apache.org:subscription:keygenerator"
        xmlns="urn:uddi-org:api_v3">
        <name>marketing-apache-org:subscription:keyGenerator</name>
        <description>Marketing Subscriptions domain key
generator</description>
        <overviewDoc>
            <overviewURL useType="text">
                http://uddi.org/pubs/uddi_v3.htm#keyGen
            </overviewURL>
        </overviewDoc>
        <categoryBag>
            <keyedReference
tModelKey="uddi:uddi.org:categorization:types"
                keyName="uddi-org:types:keyGenerator"
                keyValue="keyGenerator" />
        </categoryBag>
    </tModel>

    <tModel tModelKey="uddi:sales.apache.org:keygenerator"
xmlns="urn:uddi-org:api_v3">
        <name>sales-apache-org:keyGenerator</name>
        <description>Sales Root domain key generator</description>
        <overviewDoc>
            <overviewURL useType="text">
                http://uddi.org/pubs/uddi_v3.htm#keyGen
            </overviewURL>
        </overviewDoc>
        <categoryBag>
            <keyedReference
tModelKey="uddi:uddi.org:categorization:types"
```

```
                  keyName="uddi-org:types:keyGenerator"
                  keyValue="keyGenerator" />
          </categoryBag>
      </tModel>
</save_tModel>
```

Procedure 5.3. Task

1. If you are going to use the marketing publisher to subscribe to updates in the sales registry, then you must provide this publisher with two clerks. You do so by opening the **`uddi-portlet.war/uddi.xml`** file in a text editor and adding the following lines:

```
    <clerks registerOnStartup="false">
    <clerk  name="MarketingCratchit"    node="default"
            publisher="marketing"       password="marketing"/>

    <clerk  name="SalesCratchit"        node="sales-ws"
            publisher="marketing"       password="marketing"/>
    <!--  optional
    <xregister>
        <servicebinding
            entityKey="uddi:marketing.apache.org:servicebindings-
subscriptionlistener-ws"
            fromClerk="MarketingCratchit" toClerk="SalesCratchit"/>
    </xregister>
    -->
</clerks>
```

   In the code above, you created two clerks for this publisher, namely MarketingCratchit and SalesCratchit. These allow the publisher to check on the subscriptions it owns in each of the two systems.

2. Save the file and exit.

3. Log in as the marketing publisher on the marketing portal and select the **`UDDISubscription Portlet`**.

4. When both nodes turn green, click on the **new subscription** icon (found on the toolbar.) This will be a synchronous subscription, so only leave the **`Binding Key`** and **`Notification Interval`**

5. Click the **Save** icon to store the subscription.

6. Make sure that the subscription key uses the marketing publisher's **`keyGenerator`**'s conventions. You should see the orange subscription icon appear under the **`sales-ws`** UDDI node.

7. To invoke a synchronous subscription, click the **Green Arrows** icon. This will give you the opportunity to set the coverage period.

8. Click the **Green Arrows** icon again to invoke the synchronous subscription request.

   The example finder request will search the sales node looking for updates to the **`HelloWorld`** service. The raw XML response is then posted to the **`UDDISubscriptionNotification`** portlet:

9. The response is processed by the marketing node. This node then imports the `HelloWorld` subscription information, as well as the `sales business`. If they synchronise successfully, three businesses will be visible in the marketing node's browser portlet.

Report a bug

CHAPTER 6. SUPPORT FOR M-BEANS

6.1. M-BEAN

An M-Bean (Managed Bean) is a Java object that represents a manageable resource, such as a service or application. All of the registered services in the application server's micro-kernel are represented as M-Beans.

Report a bug

6.2. JUDDI M-BEANS

You can query jUDDI M.-Beans in the JMX console. Doing so allows you to observe Service Registry operations. These are the M.-Beans available:

- org.apache.juddi.api.impl.UDDIServiceCounter
- org.apache.juddi.api.impl.UDDICustodyTransferCounter
- org.apache.juddi.api.impl.UDDIInquiryCounter
- org.apache.juddi.api.impl.UDDIPublicationCounter
- org.apache.juddi.api.impl.UDDISecurityCounter
- org.apache.juddi.api.impl.UDDISubscriptionCounter

Each UDDI operation under the API supplies the following functionality for each method:

- successful queries
- failed queries
- total queries
- processing time
- an aggregate count of total/successful/failed per API

Only one operation is available: resetCounts.

Report a bug

CHAPTER 7. USING THE JUDDI CLIENT

7.1. JUDDI CLIENT

The jUDDI Client connects you to your Service Registry. The client is found in `SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/lib/juddi-client-VERSION.jar`.

Report a bug

7.2. JUDDI CLIENT DEPENDENCIES

The jUDDI Client depends on the following files:

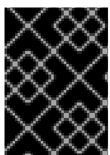
- `uddi-ws-3.0.0.jar`
- `commons-configuration-1.5.jar`
- `commons-collection-3.2.1.jar`
- `log4j-1.2.13.jar`

In addition to these, you may require the following, depending on your configuration choices:

- libraries for JDK6.
- JAXWS client libraries (if you are using a JAXWS transport like CXF.)
- RMI and JNDI client libraries (if you are using the RMI Transport.)

Report a bug

7.3. JUDDI CLIENT AND THE JBOSS ENTERPRISE SOA PLATFORM



IMPORTANT

The jUDDI Client uses the UDDI v3 API so it should be able to connect to any UDDI v3-compliant registry. However, Red Hat only supports it with the jUDDI v3 Registry.

Report a bug

7.4. TRANSPORT SETTINGS

The transport settings are the connection settings that the jUDDI client uses to talk to its server.

Report a bug

7.5. UDDI.XML

The **META-INF/uddi.xml** file contains the configuration settings for the jUDDI client. You can include this file in the deployment archive that is interacting with the UDDI client code.

Report a bug

7.6. DEPLOY A CUSTOM JUDDI CLIENT CONFIGURATION

Procedure 7.1. Task

1. Open META-INF/uddi.xml in your text editor.
2. Configure the settings.
3. Save and exit.
4. Include the uddi.xml file in your archive when you deploy it.
5. Call this code:

```
UDDIClerkManager clerkManager = new
UDDIClerkManager("META/myuddi.xml");
clerkManager.start();
```

6. Alternatively, if your application deploys as a WAR archive, add your client configuration to **yourwar/META-INF/myuddi.xml** and, in the web.xml file, specify these context parameters: **uddi.client.manager.name** and **uddi.client.xml**.

   In this example, both context parameters are set and on deployment the UDDIClerkServlet takes care of reading the configuration:

```
       <!-- required -->
<context-param>
    <param-name>uddi.client.manager.name</param-name>
    <param-value>example-manager</param-value>
 </context-param>

<!-- optional override -->
<context-param>
    <param-name>uddi.client.xml</param-name>
    <param-value>META-INF/myuddi.xml</param-value>
</context-param>

<servlet>
    <servlet-name>UDDIClerkServlet</servlet-name>
    <display-name>Clerk Servlet</display-name>
    <servlet-
class>org.apache.juddi.v3.client.config.UDDIClerkServlet</servlet-
class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Report a bug

7.7. SAMPLE JUDDI CLIENT CONFIGURATION FILE

Here is a simple jUDDI client configuration:

```
        <?xml version="1.0" encoding="ISO-8859-1" ?>
<uddi>
  <reloadDelay>5000</reloadDelay>
  <manager name="example-manager">
    <nodes>
      <node isHomeJUDDI="true">
        <name>default</name>
        <description>jUDDI node</description>
        <properties>
          <property name="serverName"  value="www.myuddiserver.com"/>
          <property name="serverPort"  value="8080"/>
          <property name="keyDomain"
value="mydepartment.mydomain.org"/>
          <property name="department"  value="mydepartment" />
        </properties>
        <!-- InVM -->

<proxyTransport>org.apache.juddi.v3.client.transport.InVMTransport</proxyT
ransport>

<custodyTransferUrl>org.apache.juddi.api.impl.UDDICustodyTransferImpl</cus
todyTransferUrl>
        <inquiryUrl>org.apache.juddi.api.impl.UDDIInquiryImpl</inquiryUrl>

<publishUrl>org.apache.juddi.api.impl.UDDIPublicationImpl</publishUrl>

<securityUrl>org.apache.juddi.api.impl.UDDISecurityImpl</securityUrl>

<subscriptionUrl>org.apache.juddi.api.impl.UDDISubscriptionImpl</subscript
ionUrl>

<subscriptionListenerUrl>org.apache.juddi.api.impl.UDDISubscriptionListene
rImpl</subscriptionListenerUrl>
        <juddiApiUrl>org.apache.juddi.api.impl.JUDDIApiImpl</juddiApiUrl>
        <!-- JAX-WS Transport

<proxyTransport>org.apache.juddi.v3.client.transport.JAXWSTransport</proxy
Transport>

<custodyTransferUrl>http://${serverName}:${serverPort}/juddiv3/services/cu
stody-transfer</custodyTransferUrl>

<inquiryUrl>http://${serverName}:${serverPort}/juddiv3/services/inquiry</i
nquiryUrl>

<publishUrl>http://${serverName}:${serverPort}/juddiv3/services/publish</p
ublishUrl>

<securityUrl>http://${serverName}:${serverPort}/juddiv3/services/security<
/securityUrl>

<subscriptionUrl>http://${serverName}:${serverPort}/juddiv3/services/subsc
```

```
ription</subscriptionUrl>

<subscriptionListenerUrl>http://${serverName}:${serverPort}/juddiv3/servic
es/subscription-listener</subscriptionListenerUrl>

<juddiApiUrl>http://${serverName}:${serverPort}/juddiv3/services/juddi-
api?wsdl</juddiApiUrl>
        -->
        <!-- RMI Transport Settings

<proxyTransport>org.apache.juddi.v3.client.transport.RMITransport</proxyTr
ansport>

<custodyTransferUrl>/juddiv3/UDDICustodyTransferService</custodyTransferUr
l>
        <inquiryUrl>/juddiv3/UDDIInquiryService</inquiryUrl>
        <publishUrl>/juddiv3/UDDIPublicationService</publishUrl>
        <securityUrl>/juddiv3/UDDISecurityService</securityUrl>

<subscriptionUrl>/juddiv3/UDDISubscriptionService</subscriptionUrl>

<subscriptionListenerUrl>/juddiv3/UDDISubscriptionListenerService</subscri
ptionListenerUrl>
        <juddiApiUrl>/juddiv3/JUDDIApiService</juddiApiUrl>

<javaNamingFactoryInitial>org.jnp.interfaces.NamingContextFactory</javaNam
ingFactoryInitial>

<javaNamingFactoryUrlPkgs>org.jboss.naming</javaNamingFactoryUrlPkgs>

<javaNamingProviderUrl>jnp://${serverName}:1099</javaNamingProviderUrl>
        -->
      </node>
    </nodes>
    <clerks registerOnStartup="true">
      <clerk name="BobCratchit" node="default" publisher="bob"
password="bob">
         <class>org.apache.juddi.samples.HelloWorldImpl</class>
      </clerk>
    </clerks>
  </manager>
</uddi>
```

The manager element is required. The example-manager name attribute should be unique to your deployment environment. The nodes element may contain one or more sub-node elements. Normally, you would only need one node, unless you are using subscriptions to transfer updates of entities from one UDDI registry to another. For the 'local' registry you would set **isHomeJUDDI="true"**, while for the 'remote' registries you would set **isHomeJUDDI="false"**.

Table 7.1. Elements

Element Name	Description	Required?
name	name of the node	yes

Element Name	Description	Required?
description	description of the node	no
properties	container for properties that will be passed into the clerk	no
proxyTransport	The Transport protocol used by the client to connect to the UDDI server	yes
custodyTransferUrl	Connection settings for custody transfer	no
inquiryUrl	Connection location settings for inquiries	yes
publishUrl	Connection location settings for publishing	yes
securityUrl	Connection location settings for obtaining security tokens	yes
subscriptionUrl	Connection location settings for registering subscription requests	no
subscriptionListenerUrl	Connection location settings receiving subscription notifications	no
juddiApiUrl	Connection location settings for the jUDDI specific API for things like publisher management	no

Finally, the manager element can contain a 'clerks' element in which you can define one or more clerks.

Table 7.2. Clerks

Attribute Name	Description	Required?
name	name of the clerk	yes
node	name reference to one of the nodes specified in the same manager	yes
publisher	name of an existing publisher	yes

Attribute Name	Description	Required?
password	password credential of the publisher	yes

Report a bug

7.8. JAVA API FOR XML WEB SERVICES (JAX-WS)

The Java API for XML Web Services (JAX-WS) is a Java API that allows you to create web services. The JAX-WS handler mechanism is used by the web service to invoke a user-specified class whenever a message (or fault) is sent or received. The handler is therefore installed in the message pipeline and used to manipulate the message header or body as required.

Report a bug

7.9. SET THE JAX-WS TRANSPORT

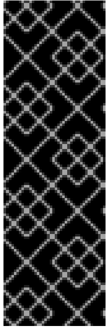
Prerequisites

- Based on the settings in the **uddi.xml** file, the client will use JAX-WS to communicate with the (remote) registry server. This means that the client needs to have access to a JAX-WS-compliant web service stack (such as CXF, Axis2 or JBossWS).

Procedure 7.2. Task

- Ensure that the JAX-WS URLs are pointing to an address where the UDDI client can find the WSDL documents:

```
<!-- JAX-WS Transport -->
<proxyTransport>org.apache.juddi.v3.client.transport.JAXWSTransport<
/proxyTransport>
<custodyTransferUrl>http://${serverName}:${serverPort}/juddiv3/servi
ces/custody-transfer</custodyTransferUrl>
<inquiryUrl>http://${serverName}:${serverPort}/juddiv3/services/inqu
iry</inquiryUrl>
<publishUrl>http://${serverName}:${serverPort}/juddiv3/services/publ
ish</publishUrl>
<securityUrl>http://${serverName}:${serverPort}/juddiv3/services/sec
urity</securityUrl>
<subscriptionUrl>http://${serverName}:${serverPort}/juddiv3/services
/subscription</subscriptionUrl>
<subscriptionListenerUrl>http://${serverName}:${serverPort}/juddiv3/
services/subscription-listener</subscriptionListenerUrl>
<juddiApiUrl>http://${serverName}:${serverPort}/juddiv3/services/jud
di-api?wsdl</juddiApiUrl>
```



IMPORTANT

Pros: This is the standard way of performing UDDI communication and it should should work with all UDDIv3 server implementations.

Cons: If the server is deployed on the same application server this may lead to issues when auto-registration on deployment/undeployment is used, since the WS stack may become unavailable during undeployment. A workaround is to host the UDDI server on a different server.

Report a bug

7.10. REMOTE INVOCATION CLASS

As its name implies, a remote invocation class is a class that can be called from a remote machine. This can be useful for developers but can also lead to potential security risks.

Report a bug

7.11. JUDDI AND REMOTE METHOD INVOCATION

If you deploy the jUDDI to an Application Server, you can register the UDDI services as remote method invocation services.

Report a bug

7.12. ENABLE REMOTE METHOD INVOCATION FOR JUDDI

Procedure 7.3. Task

1. Open the jUDDI configuration file in a text editor: `vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml`.

2. Edit the settings and set the property <entry key="juddi.jndi.registration">true</entry>. If set to true, RMI methods are registered to jndi and can be searched and called. Default is true.

3. Save the file and exit.

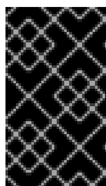
4. Open the UDDI configuration file in a text editor: `vi META-INF/uddi.xml`.

5. "Comment out" the JAX-WS section of the file and uncomment the RMI section instead.

6. As an optional step, you can also set the java.naming.* properties. In this example, you specified the setting for connecting to jUDDI v3 deployed on a **JBoss Application Server**. If you like you can instead set the java.naming.* properties in a `jndi.properties` file, or as system parameters.

```
<!-- RMI Transport Settings -->
<proxyTransport>org.apache.juddi.v3.client.transport.RMITransport</p
roxyTransport>
```

```
<custodyTransferUrl>/juddiv3/UDDICustodyTransferService</custodyTran
sferUrl>
<inquiryUrl>/juddiv3/UDDIInquiryService</inquiryUrl>
<publishUrl>/juddiv3/UDDIPublicationService</publishUrl>
<securityUrl>/juddiv3/UDDISecurityService</securityUrl>
<subscriptionUrl>/juddiv3/UDDISubscriptionService</subscriptionUrl>
<subscriptionListenerUrl>/juddiv3/UDDISubscriptionListenerService</s
ubscriptionListenerUrl>
<juddiApiUrl>/juddiv3/JUDDIApiService</juddiApiUrl>
<javaNamingFactoryInitial>org.jnp.interfaces.NamingContextFactory</j
avaNamingFactoryInitial>
<javaNamingFactoryUrlPkgs>org.jboss.naming</javaNamingFactoryUrlPkgs
>
<javaNamingProviderUrl>jnp://${serverName}:1099</javaNamingProviderU
rl>
```



IMPORTANT

Pros: It is lightweight fast since it does not need a web service stack.

Cons: It only works with a jUDDI v3 server implementation.

7. Save the file and exit.

Result

When you deploy it, the RMI-based UDDI services will bind to the global JNDI namespace:

- juddiv3 (class: `org.jnp.interfaces.NamingContext`)
- UDDIPublicationService (class: `org.apache.juddi.rmi.UDDIPublicationService`)
- UDDICustodyTransferService (class: `org.apache.juddi.rmi.UDDICustodyTransferService`)
- UDDISubscriptionListenerService (class: `org.apache.juddi.rmi.UDDISubscriptionListenerService`)
- UDDISecurityService (class: `org.apache.juddi.rmi.UDDISecurityService`)
- UDDISubscriptionService (class: `org.apache.juddi.rmi.UDDISubscriptionService`)
- UDDIInquiryService (class: `org.apache.juddi.rmi.UDDIInquiryService`)

Report a bug

7.13. INVM TRANSPORT

The InVM ("intra-virtual machine") Transport provides communication between services running on the same JVM.

Report a bug

7.14. INVM AND JUDDI

You have the option of using the InVM Transport. It allows you to run the jUDDI server in the same virtual machine as your client. If you are deploying to the **juddi.war** archive, the server will be started by the **org.apache.juddi.RegistryServlet** class, but if you are running outside any container, you are responsible for starting and stopping the **org.apache.juddi.Registry** service yourself.

Report a bug

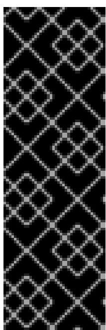
7.15. CONFIGURE THE INVM TRANSPORT FOR JUDDI

Procedure 7.4. Task

1. Open the UDDI configuration file in a text editor: **vi META-INF/uddi.xml**.

2. "Comment out" the JAX-WS and RMI Transport sections of the file and uncomment the InVM Transport section instead.

```
<!-- InVM -->
<proxyTransport>org.apache.juddi.v3.client.transport.InVMTransport</
proxyTransport>
<custodyTransferUrl>org.apache.juddi.api.impl.UDDICustodyTransferImp
l</custodyTransferUrl>
<inquiryUrl>org.apache.juddi.api.impl.UDDIInquiryImpl</inquiryUrl>
<publishUrl>org.apache.juddi.api.impl.UDDIPublicationImpl</publishUr
l>
<securityUrl>org.apache.juddi.api.impl.UDDISecurityImpl</securityUrl
>
<subscriptionUrl>org.apache.juddi.api.impl.UDDISubscriptionImpl</sub
scriptionUrl>
<subscriptionListenerUrl>org.apache.juddi.api.impl.UDDISubscriptionL
istenerImpl</subscriptionListenerUrl>
<juddiApiUrl>org.apache.juddi.api.impl.JUDDIApiImpl</juddiApiUrl>
```



IMPORTANT

Pros: It is lightweight and provides the best performance for communications and there are no deployment order issues when using auto-registration of services during deployment and undeployment.

Cons: It will only work with a jUDDI v3 server implementation. Normally, you would use a jUDDI server for each application server sharing one common database.

3. Save the file and exit.

Report a bug

7.16. START THE SERVICE REGISTRY USING THE INVM TRANSPORT

- Call the following method prior to making any other calls to it: `Registry.start()`

Report a bug

7.17. STOP THE SERVICE REGISTRY USING THE INVM TRANSPORT

Procedure 7.5. Task

- Call the following method prior to making any other calls to it: `Registry.stop()`

Result

The registry releases any resources it may be holding.

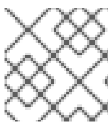
Report a bug

7.18. UDDI ANNOTATION

UDDI annotations automatically register services as they are deployed. By having them automatically handling service registration, it means the data in the Service Registry is less likely to become outdated than if you added and removed each end-point reference manually.

There are two annotations, namely UDDIService and UDDIServiceBinding. You need to use both to register an end-point.

When you undeploy a service, the end-point will be removed from the UDDI Registry but the service information remains. (It makes sense to leave the service level information in the Registry since this reflects that the service is there even though there is no end-point at the moment. This is telling it to "check back later").



NOTE

If you do want to remove service information, you must do so manually.

Report a bug

7.19. USE THE UDDISERVICE ANNOTATION

Procedure 7.6. Task

- Use the UDDIService annotation to register a service under an already-existing business in the Registry. The annotation should be added to the Java class' "class" level.

Report a bug

7.20. UDDISERVICE ATTRIBUTES

Table 7.3.

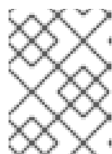
Attribute	Description	Required?
serviceName	This is the name of the service. By default, the clerk will use the one name specified in the WebService annotation	No
description	This is the human-readable description of the service	Yes
serviceKey	This is the service's UDDI v3 key.	Yes
businessKey	This is the UDDI v3 key of the business that should own the service. (The business should exist in the Registry at time of registration.)	Yes
lang	This is the language locale which will be used for the name and description. (It defaults to "en" if you omit it.)	No
categoryBag	This is the definition of a *CategoryBag*.	No

Report a bug

7.21. USE THE UDDISERVICEBINDING ANNOTATION

Procedure 7.7. Task

- Use the UDDIServiceBinding annotation to register an end-point reference in the Service Registry. The annotation should be added to the Java class' "class" level.



NOTE

You cannot use this annotation by itself. It needs to be utilised within an UDDIService annotation

Report a bug

7.22. UDDISERVICEBINDING ATTRIBUTES

Table 7.4. UDDIServiceBinding Attributes

Attribute	Description	Required?
bindingKey	This is the ServiceBinding's UDDI v3 key.	Yes
description	This is a human-readable description of the service	Yes
accessPointType	This is UDDI v3's AccessPointType. (It defaults to wsdlDeployment if you omit it.)	No
accessPoint	This is the end-point reference	Yes
lang	This is the language locale which will be used for the name and description, (It will default to "en" if you omit it.)	No
tModelKeys	This is a comma-separated list of tModelKeys key references.	No
categoryBag	This is the definition of a *CategoryBag*.	No

Report a bug

7.23. EXAMPLE USE OF UDDI ANNOTATIONS

Introduction

You can use the annotations on any class that defines a service. Here they are added to a web service (a POJO with a JAX-WS web service annotation):

```
        package org.apache.juddi.samples;

import javax.jws.WebService;
import org.apache.juddi.v3.annotations.UDDIService;
import org.apache.juddi.v3.annotations.UDDIServiceBinding;

@UDDIService(
  businessKey="uddi:myBusinessKey",
  serviceKey="uddi:myServiceKey",
  description = "Hello World test service")
@UDDIServiceBinding(
  bindingKey="uddi:myServiceBindingKey",
     description="WSDL endpoint for the helloWorld Service. This service
is used for "
      + "testing the jUDDI annotation functionality",
     accessPointType="wsdlDeployment",
     accessPoint="http://localhost:8080/juddiv3-
```

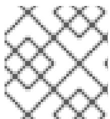
```
samples/services/helloworld?wsdl")
@WebService(
  endpointInterface = "org.apache.juddi.samples.HelloWorld",
        serviceName = "HelloWorld")

public class HelloWorldImpl implements HelloWorld {
    public String sayHi(String text) {
        System.out.println("sayHi called");
        return "Hello " + text;
    }
}
```

When you deploy this web service, the **juddi-client** code will scan this class for UDDI annotations and take care of the registration process.

In the clerk section, you need to refer to the **org.apache.juddi.samples.HelloWorldImpl** service class:

```
            <clerk name="BobCratchit" node="default" publisher="sales"
password="sales">
                 <class>org.apache.juddi.samples.HelloWorldImpl</class>
            </clerk>
```

This code dictates that Bob is using the connection setting of the node named **default**, and that he will be using the **sales** publisher (for which the password is **sales**).



NOTE

This is analogous with how data-sources are defined.

Report a bug

7.24. CATEGORYBAG ATTRIBUTE

Use the CategoryBag attribute to refer to tModels.

Report a bug

7.25. EXAMPLE USE OF THE CATEGORYBAG ANNOTATION

Here is a sample CategoryBag:

```
        <categoryBag>
           <keyedReference tModelKey="uddi:uddi.org:categorization:types"
            keyName="uddi-org:types:wsdl" keyValue="wsdlDeployment" />
           <keyedReference tModelKey="uddi:uddi.org:categorization:types"
            keyName="uddi-org:types:wsdl2" keyValue="wsdlDeployment2" />
        </categoryBag>
```

You can add it like this:

```
categoryBag="keyedReference=keyName=uddi-
org:types:wsdl;keyValue=wsdlDeployment;" +
                     "tModelKey=uddi:uddi.org:categorization:types," +
                     "keyedReference=keyName=uddi-
org:types:wsdl2;keyValue=wsdlDeployment2;" +
                     "tModelKey=uddi:uddi.org:categorization:types2",
```

Report a bug

7.26. USE ANNOTATIONS TO DEVELOP A WEB SERVICE

Procedure 7.8. Task

1. Add an annotation to any class that defines a service. Here is an example where they are added to a web service POJO with a JAX-WS Webservice annotation:

```
package org.apache.juddi.samples;

import javax.jws.WebService;
import org.apache.juddi.v3.annotations.UDDIService;
import org.apache.juddi.v3.annotations.UDDIServiceBinding;

@UDDIService(businessKey="uddi:myBusinessKey",
serviceKey="uddi:myServiceKey", description = "Hello World test
service")
@UDDIServiceBinding(bindingKey="uddi:myServiceBindingKey",
description="WSDL endpoint for the helloWorld Service. This service
is used for "
      + "testing the jUDDI annotation functionality",
    accessPointType="wsdlDeployment",
accessPoint="http://localhost:8080/juddiv3-
samples/services/helloworld?wsdl")
@WebService(endpointInterface =
"org.apache.juddi.samples.HelloWorld", serviceName = "HelloWorld")

public class HelloWorldImpl implements HelloWorld {
    public String sayHi(String text) {
        System.out.println("sayHi called");
        return "Hello " + text;
    }
}
```

   On deployment of this WebService, the juddi-client code will scan this class for UDDI annotations and take care of the registration process.

2. Open the uddi.xml file in your text editor: **`vi uddi.xml`**

3. Scroll down to the "clerk" section and add a reference to the **`org.apache.juddi.samples.HelloWorldImpl`** service class:

```
<clerks registerOnStartup="true">
  <clerk name="BobCratchit" node="default" publisher="bob"
password="bob">
```

```
      <class>org.apache.juddi.samples.HelloWorldImpl</class>
    </clerk>
  </clerks>
```

In this example, Bob is using the connection setting for the node named "default". Furthermore, he will be using a publisher named "bob", for which the password is also "bob".

4. Save the file and exit.

Report a bug

7.27. KEY TEMPLATE

A key template is a template that can be set in a business, service or binding key's "annotation" attribute. Both the WSDL and BPEL registration code use the key format o construct UDDI v3 keys. The format of the keys is defined in the properties section of the **uddi.xml** file.

Both the key's serviceName and portName are obtained from the RegistrationInfo. The nodeName can be obtained automatically from the environment, or you can set it manually in the **uddi.xml** file. The values you set for the properties in this template will be used when the key is registered.

Report a bug

7.28. KEY TEMPLATE PROPERTIES

Table 7.5. Key Template Properties

Property	Description	Required?	Default Value
lang	The language setting used by the registration	no	en
businessName	The business name which is used by the registration.	yes	-
keyDomain	The key domain key part (used by the key formats)	yes	-
businessKeyFormat	Key format used to construct the Business Key	no	uddi:${keyDomain}:business_${businessName}
serviceKeyFormat	Key format used to construct the BusinessService Key	no	uddi:${keyDomain}:service_${serviceName}

Property	Description	Required?	Default Value
bindingKeyFormat	Key format used to construct the TemplateBinding Key	no	uddi:${keyDomain}:binding_${nodeName}_${serviceName}_${portName}
serviceDescription	Default BusinessService description	no	Default service description when no <wsdl:document> element is defined inside the <wsdl:service> element.
bindingDescription	Default BindingTemplate description	no	Default binding description when no <wsdl:document> element is defined inside the <wsdl:binding> element.

Report a bug

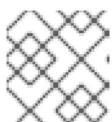
7.29. USE THE JUDDI CLIENT CODE IN YOUR APPLICATION

Procedure 7.9. Task

1. Open the jUDDI client configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml`**. Use this file as a template for your own **`uddi.xml`** file.

2. Make sure the clerk manager points to your custom **`uddi.xml`** when it is instantiated:

```
UDDIClerkManager clerkManager = new
UDDIClerkManager("META/myuddi.xml");
clerkManager.start();

UDDIClerk clerk =
clerkManager.getClientConfig().getUDDIClerks().get(clerkName);
```



NOTE

A UDDIClerk will allow you do make authenticated requests to a UDDI server.

Report a bug

7.30. WSDL AND THE UDDI REGISTRY

The jUDDI client implements the UDDI v3 version of the WSDL2UDDI mapping as described in the OASIS technical note found at http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm. The registration process registers a BindingTemplate for each WebService EndPoint and, if the BusinessService for this BindingTemplate does not yet exist, it also registers the BusinessService along with a WSDLPortType TModel for each portType, and a WSDLBinding TModel for each binding.

Report a bug

7.31. SPECIFICATION FOR THE SERVICE AND BINDING DESCRIPTIONS IN THE WSDL

The UDDI specification allows you to set a human-readable description on both the BusinessService and TemplateBinding. These description fields are important if humans are browsing the registry. A default global description is found in the **uddi.xml** file. However it makes a lot more sense to have a specific description for each service and binding and so the registration code tries to obtain these descriptions from the <wsdl:document> tags in the WSDL, which can be nested as a child element inside the <wsdl:service> and <wsdl:binding> elements.

Report a bug

7.32. REGISTER A WSDL PROCESS IN JUDDI

Prerequisites

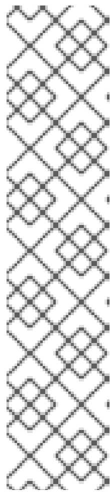
- org.apache.juddi.v3.client.mapping

Procedure 7.10. Task

1. Use the code in the **org.apache.juddi.v3.client.mapping** package.

2. Make the following call to asynchronously register your web service end point:

```
//Add the properties from the uddi.xml
properties.putAll(clerk.getUDDINode().getProperties());
RegistrationInfo registrationInfo = new RegistrationInfo();
registrationInfo.setServiceQName(serviceQName);
registrationInfo.setVersion(version);
registrationInfo.setPortName(portName);
registrationInfo.setServiceUrl(serviceUrl);
registrationInfo.setWsdlUrl(wsdlURL);
registrationInfo.setWsdlDefinition(wsdlDefinition);
registrationInfo.setRegistrationType(RegistrationType.WSDL);
registration = new AsyncRegistration(clerk, urlLocalizer,
properties, registrationInfo);
Thread thread = new Thread(registration);
thread.start();
```



NOTE

This does assume that you can pass in a URL to the WSDL file in addition to the WSDLDefinition. In most cases you will need to package the WSDL file you wish to register in your deployment.

To obtain a WSDLDefinition use this code:

```
ReadWSDL readWSDL = new ReadWSDL();
Definition definition =
readWSDL.readWSDL("wsdl/HelloWorld.wsdl");
```

You need to pass the path information to the WSDL that is on the classpath.

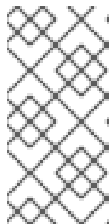
Report a bug

7.33. REMOVE A WSDL BINDING FROM THE SERVICE REGISTRY

Procedure 7.11. Task

- To remove a WSDL binding from the service registry, use this code:

```
WSDL2UDDI wsdl2UDDI = new WSDL2UDDI(clerk, urlLocalizer,
properties);
String serviceKey = wsdl2UDDI.unRegister(serviceName, portName,
serviceURL);
```



NOTE

If this is the last BindingTemplate for the BusinessService it will also remove the BusinessService itself along with the WSDLPortType and WSDLBinding TModels. The lifecycle is set to "registration" on deployment of the end point and set to "unregistration" on removal of the end point.

Report a bug

7.34. REGISTER A BPEL PROCESS IN JUDDI

Prerequisites

- org.apache.juddi.v3.client.mapping

Procedure 7.12. Task

1. Use the code in the `org.apache.juddi.v3.client.mapping` package.
2. Make the following call to asynchronously register your web service end point:

```
//Add the properties from the uddi.xml
properties.putAll(clerk.getUDDINode().getProperties());
```

```
RegistrationInfo registrationInfo = new RegistrationInfo();
registrationInfo.setServiceQName(serviceQName);
registrationInfo.setVersion(version);
registrationInfo.setPortName(portName);
registrationInfo.setServiceUrl(serviceUrl);
registrationInfo.setWsdlUrl(wsdlURL);
registrationInfo.setWsdlDefinition(wsdlDefinition);
registrationInfo.setRegistrationType(RegistrationType.BPEL);
registration = new AsyncRegistration(clerk, urlLocalizer,
properties, registrationInfo);
Thread thread = new Thread(registration);
thread.start();
```

Set the RegistrationInfo.RegistrationType to RegistrationType.BPEL.

Report a bug

7.35. URLLOCALIZER INTERFACE AND THE JBOSS ENTERPRISE SOA PLATFORM

The end-point URL setting is obtained from WSDL <wsdl:port>'s <soap:addressbinding>. Unfortunately, this URL is static, and it is useful if it can be made dynamic. In the version of the URLLocalizer that ships with the JBoss Enterprise SOA Platform, the settings obtained from the local WebService Stack override the protocol and host parts of the URL, mitigating this problem.

Report a bug

7.36. DYNAMIC UDDI SERVICE LOOK-UP

"Dynamic UDDI service look-up" refers to the process of periodically interrogating the UDDI registry to obtain fresh binding information. This makes the system more dynamic than if it simply looked at the actual binding information for the web service's end point that is stored on the client-side. Furthermore, it allows clients to simply following the changes that occur in the service deployment topology.

Report a bug

7.37. USE THE SERVICE LOCATOR TO FIND A SERVICE BINDING

Prerequisites

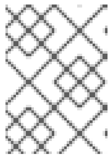
- You must already know the names of the service and port.

Procedure 7.13. Task

- To locate a service binding using the service locator, use this code:

```
ServiceLocator serviceLocator = new ServiceLocator(clerk,
urlLocalizer, properties);
String endPointURL = serviceLocator.lookupEndpoint(serviceQName,
```

```
String portName);
```



NOTE

The downside of doing a look-up prior to each service invocation is that it will be detrimental to performance.

Report a bug

7.38. TUTORIAL: USING THE JUDDI CLIENT

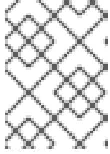
Prerequisites

- Make sure the publisher (in this case, **root**) already exists in the Service Registry

Procedure 7.14. Task

1. Obtain the sample code located in the **uddi-client** module.
2. Make a call to the Registry to obtain an authentication token. (The following code is taken from the unit tests in this module):

```
public void testAuthToken() {
    try {
        String clazz = ClientConfig.getConfiguration().getString(

Property.UDDI_PROXY_TRANSPORT,Property.DEFAULT_UDDI_PROXY_TRANSPORT)
;
        Class<?> transportClass = Loader.loadClass(clazz);
        if (transportClass!=null) {
            Transport transport = (Transport)
transportClass.newInstance();
            UDDISecurityPortType securityService =
transport.getSecurityService();
            GetAuthToken getAuthToken = new GetAuthToken();
            getAuthToken.setUserID("root");
            getAuthToken.setCred("");
            AuthToken authToken =
securityService.getAuthToken(getAuthToken);
            System.out.println(authToken.getAuthInfo());
            Assert.assertNotNull(authToken);
        } else {
            Assert.fail();
        }
    } catch (Exception e) {
        e.printStackTrace();
        Assert.fail();
    }
}
```



NOTE

Ensure that you supply the correct credentials for the root publisher to obtain a successful response.

Report a bug

APPENDIX A. REVISION HISTORY

Revision 5.3.1-23.400	**2013-10-31**	**Rüdiger Landmann**

Rebuild with publican 4.0.0

Revision 5.3.1-23	**Thu Jan 10 2013**	**David Le Sage**

Built from Content Specification: 6847, Revision: 365903 by dlesage