jUDDI User Guide

A guide to using jUDDI

by Tom Cunningham, Kurt Stam, and The jUDDI Community

and thanks to Darrin Mison

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts* [https://fedorahosted.org/liberation-fonts/] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F1** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F7** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `Mono-spaced Bold`. For example:

> File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or ***`Proportional Bold Italic`***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type `ssh` *`username@domain.name`* at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type `ssh john@example.com`.
>
> The `mount -o remount` *`file-system`* command remounts the named file system. For example, to remount the `/home` file system, the command is `mount -o remount /home`.
>
> To see the version of a currently installed package, use the `rpm -q` *`package`* command. It will return a result as follows: *`package-version-release`*.

Note the words in bold italics above username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as

a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }

}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

A Note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.

Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you!

For any issues you find, or improvements you have, please sign up for a JIRA account at https://issues.apache.org/jira/secure/Dashboard.jspa and file a bug under the "jUDDI" component.

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

UDDI Registry

1.1. Introduction

The Universal Description, Discovery and Integration (UDDI) protocol is one of the major building blocks required for successful Web services. 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. UDDI is a cross-industry effort driven by major platform and software providers, as well as marketplace operators and e-business leaders within the OASIS standards consortium. UDDI has gone through 3 revisions and the latest version is 3.0.2. Additional information regarding UDDI can be found at *http://uddi.xml.org*.

1.2. UDDI Registry

The UDDI Registry implements the UDDI specification. UDDI is a Web-based distributed directory that enables businesses to list themselves on the Internet and discover each other, similar to a traditional phone book's yellow and white pages. The UDDI registry is both a white pages business directory and a technical specifications library. The Registry is designed to store information about Businesses and Services and it holds references to detailed documentation.

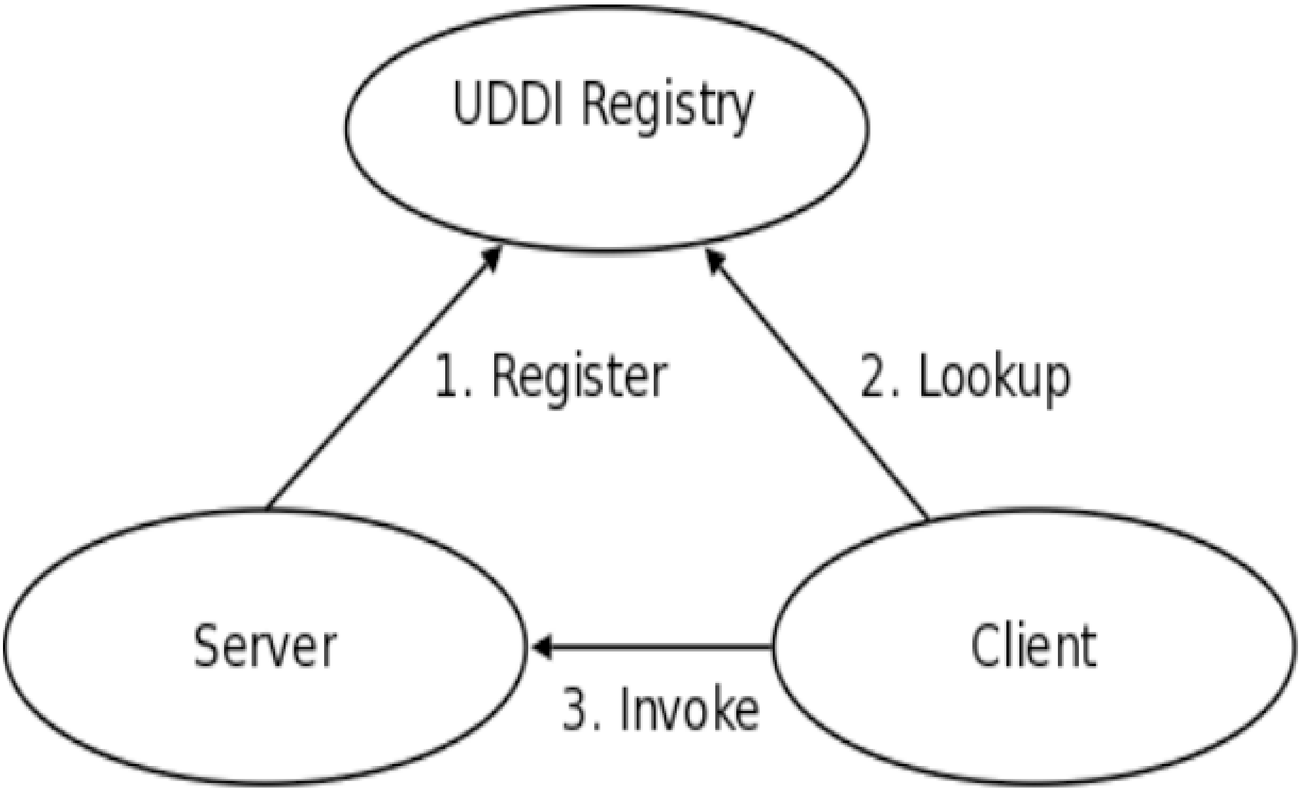


Figure 1.1. Invocation Pattern using the UDDI Registry

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. The UDDI v3 specification defines 9 APIs:

1. `UDDI_Security_PortType`, 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.

2. `UDDI_Publication_PortType`, defines the API to publish business and service information to the UDDI registry.

3. `UDDI_Inquiry_PortType`, defines the API to query the UDDI registry. Typically this API does not require a security token.

4. `UDDI_CustodyTransfer_PortType`, this API can be used to transfer the custody of a business from one UDDI node to another.

5. `UDDI_Subscription_PortType`, defines the API to register for updates on a particular business of service.

6. `UDDI_SubscriptionListener_PortType`, defines the API a client must implement to receive subscription notifications from a UDDI node.

7. `UDDI_Replication_PortType`, defines the API to replicate registry data between UDDI nodes.

8. `UDDI_ValueSetValidation_PortType`, by nodes to allow external providers of value set validation. Web services to assess whether keyedReferences or keyedReferenceGroups are valid.

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

Getting Started

2.1. What Should I Download?

The jUDDI server deploys as a WebARchive (war) named `juddiv3.war`. Within jUDDI, there are three downloadable files (`juddi-core.jar`, `juddi.war`, and `juddi-tomcat.zip`). You should determine which one to use depending on what level of integration you want with your application and your platform / server choices.

JUDDI also ships with client side code, the `juddi-client.jar`. The jUDDI server depends on the `juddi-client.jar` in situations where one server communicates to another server. In this setup one server acts as a client to the other server. The juddi-client.

2.2. Using the JAR

The juddi-core module produces a JAR which contains the jUDDI source and a jUDDI persistence.xml configuration. jUDDI's persistence is being actively tested with both OpenJPA and with Hibernate. If you are going to use only the JAR, you would need to directly insert objects into jUDDI through the database back end or persistence layer, or configure your own Web Service provider with the provided WSDL files and classes.

2.3. Using the WAR File

As with the JAR, you need to make a decision on what framework you would like to use when building the WAR. jUDDI's architecture supports any JAX-WS compliant WS stack (Axis, CXF, etc). The jUDDI 3.0.GA release ships with CXF in the Tomcat bundle, but any docs or descriptors to support other WS stacks would be welcome contributions. Simply copy the WAR to the deploy folder of your server (this release has been tested under Apache Tomcat 6.0.20), start your server, and follow the directions under “using jUDDI as a Web Service”.

2.4. Using the Tomcat Bundle

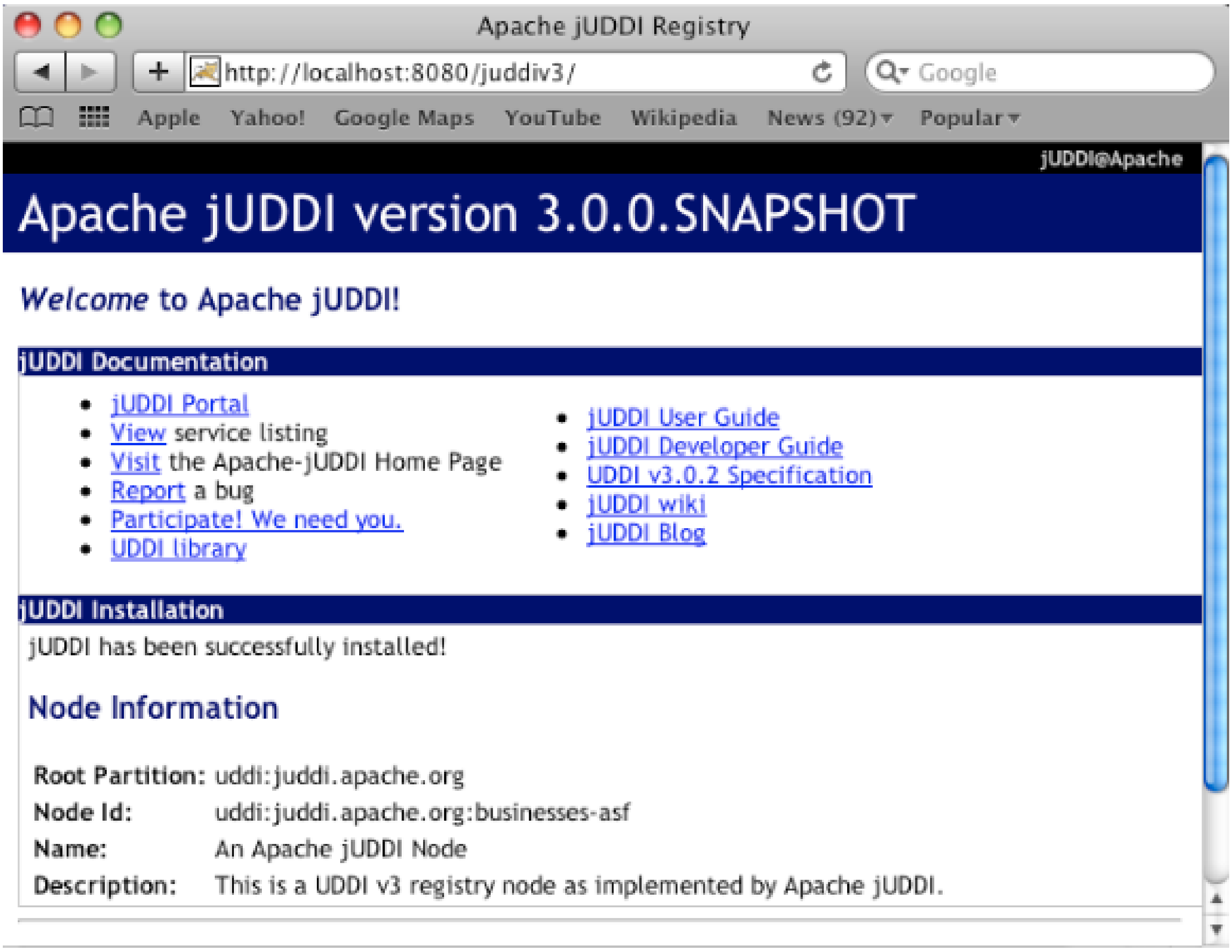
The jUDDI Tomcat bundle packages up the jUDDI WAR, Apache Derby, and a few necessary configuration files and provides the user with a pre-configured jUDDI instance. By default, Hibernate is used as the persistence layer and CXF is used as a Web Service framework. To get started using the Tomcat bundle, unzip the `juddi-tomcat-bundle.zip`, and start Tomcat :

```
% cd apache-tomcat-6.0.20/bin
% ./startup.sh
```

It is suggested that you use JDK 1.6 with the Tomcat 6 bundle. On Mac OS X you can either change your JAVA_HOME settings or use `/Applications/Utilities/Java Preferences.app` to change your current JDK.

Once the server is up and running can make sure the root data was properly installed by browsing to *http://localhost:8080/juddiv3*

You should see the screen show in *Figure 2.1, “jUDDI Welcome Page”*.



Apache jUDDI version 3.0.0.SNAPSHOT

Welcome to Apache jUDDI!

jUDDI Documentation

- jUDDI Portal
- View service listing
- Visit the Apache-jUDDI Home Page
- Report a bug
- Participate! We need you.
- UDDI library
- jUDDI User Guide
- jUDDI Developer Guide
- UDDI v3.0.2 Specification
- jUDDI wiki
- jUDDI Blog

jUDDI Installation

jUDDI has been successfully installed!

Node Information

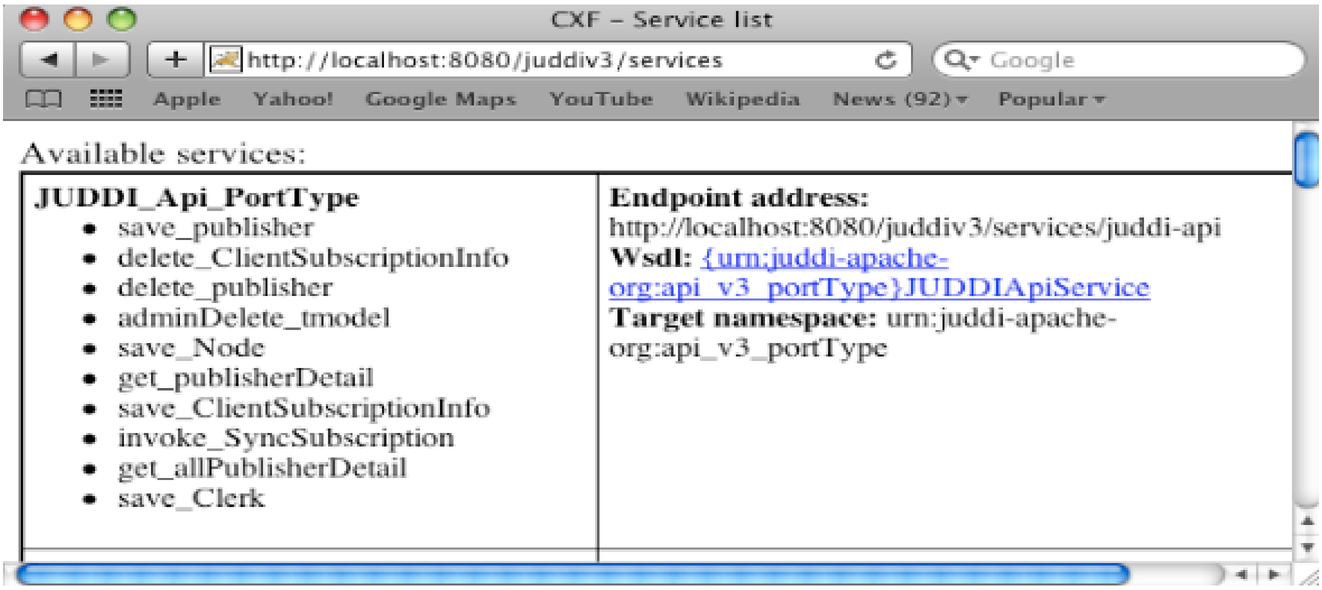
Root Partition:	uddi:juddi.apache.org
Node Id:	uddi:juddi.apache.org:businesses-asf
Name:	An Apache jUDDI Node
Description:	This is a UDDI v3 registry node as implemented by Apache jUDDI.

Figure 2.1. jUDDI Welcome Page

2.5. Using jUDDI Web Services

Once the jUDDI server is started, you can inspect the UDDI WebService API by browsing to *http://localhost:8080/juddiv3/services*

You should see an overview of all the Services and their WSDLs.

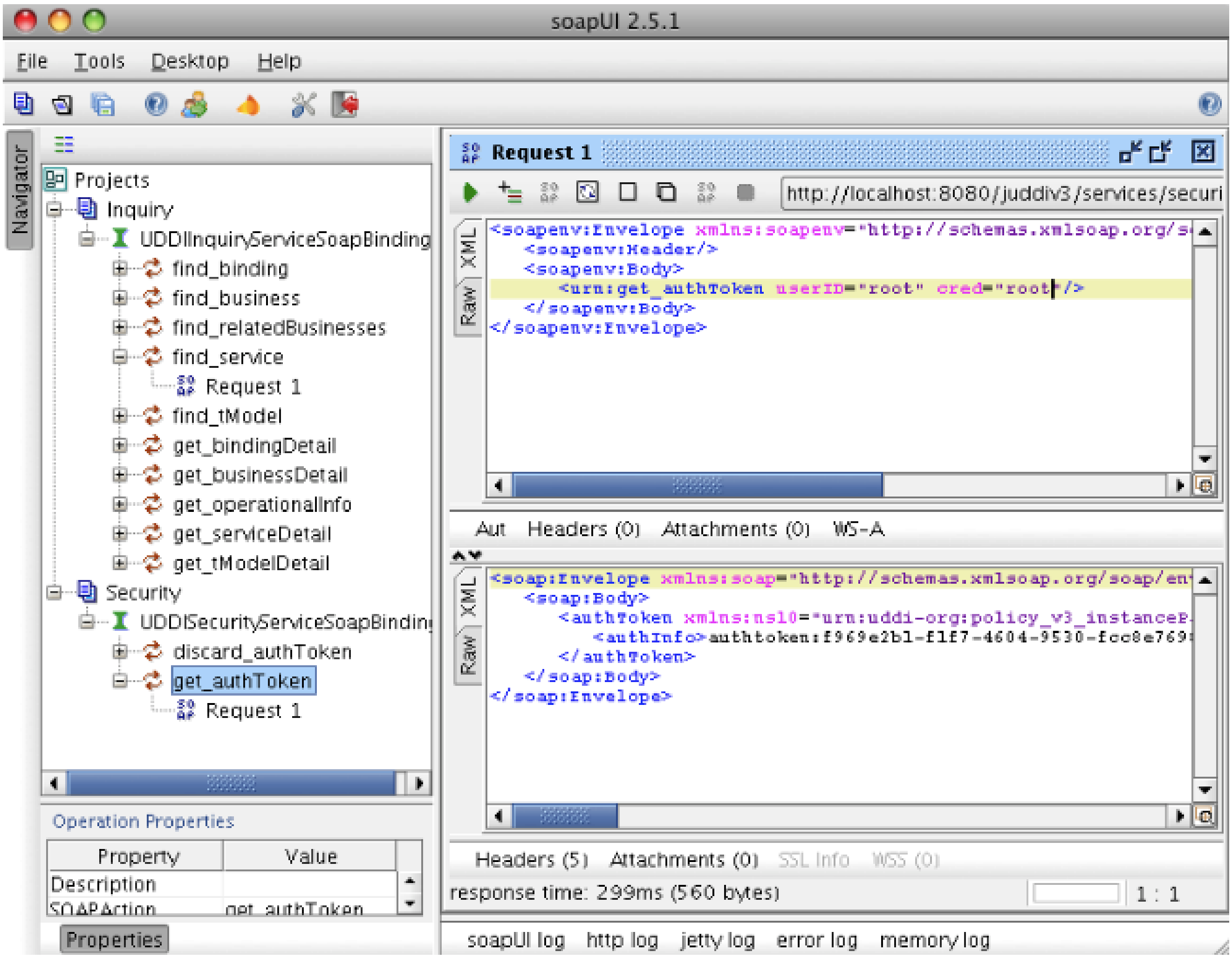


Available services:

JUDDI_Api_PortType • save_publisher • delete_ClientSubscriptionInfo • delete_publisher • adminDelete_tmodel • save_Node • get_publisherDetail • save_ClientSubscriptionInfo • invoke_SyncSubscription • get_allPublisherDetail • save_Clerk	**Endpoint address:** http://localhost:8080/juddiv3/services/juddi-api **Wsdl:** {urn:juddi-apache-org:api_v3_portType}JUDDIApiService **Target namespace:** urn:juddi-apache-org:api_v3_portType

Figure 2.2. UDDI Services Overview

The services page shows you the available endpoints and methods available. Using any SOAP client, you should be able to send some sample requests to jUDDI to test:



http://localhost:8080/juddiv3/services/securi

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/s
   <soapenv:Header/>
   <soapenv:Body>
      <urn:get_authToken userID="root" cred="root"/>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/en
   <soap:Body>
      <authToken xmlns:ns10="urn:uddi-org:policy_v3_instanceP
         <authInfo>authtoken:f969e2b1-f1f7-4604-9530-fcc8e769
      </authToken>
   </soap:Body>
</soap:Envelope>
```

Property	Value
Description	
SOAPAction	get_authToken

response time: 299ms (560 bytes)

Figure 2.3. Getting an authToken using SoapUI

Authentication

3.1. Introduction

In order to enforce proper write access to jUDDI, each request to jUDDI needs a valid `authToken`. Note that read access is not restricted and therefore queries into the registries are not restricted.

To obtain a valid `authToken` a `getAuthToken()` request must be made, where a `GetAuthToken` object is passed. On the `GetAuthToken` object a userid and credential (password) needs to be set.

```
org.uddi.api_v3.GetAuthToken ga = new org.uddi.api_v3.GetAuthToken();
ga.setUserID(pubId);
ga.setCred("");

org.uddi.api_v3.AuthToken token = securityService.getAuthToken(ga);
```

The property juddi.auth in the `juddi.properties` configuration file can be used to configure how jUDDI is going to check the credentials passed in on the `GetAuthToken` request. By default jUDDI uses the `JUDDIAuthenticator` implementation. You can provide your own authentication implementation or use any of the ones mention below. The implementation needs to implement the `org.apache.juddi.auth.Authenticator` interface, and juddi.auth property should refer to the implementation class.

There are two phases involved in Authentication. The authenticate phase and the identify phase. Both of these phases are represented by a method in the `Authenticator` interface.

The authenticate phase occurs during the `GetAuthToken` request as described above. The goal of this phase is to turn a user id and credentials into a valid publisher id. The publisher id (referred to as the “authorized name” in UDDI terminology) is the value that assigns ownership within UDDI. Whenever a new entity is created, it must be tagged with ownership by the authorized name of the publisher. The value of the publisher id can be completely transparent to jUDDI – the only requirement is that one exists to assign to new entities. Thus, the authenticate phase must return a non-null publisher id. Upon completion of the `GetAuthToken` request, an authentication token is issued to the caller.

In subsequent calls to the UDDI API that require authentication, the token issued from the `GetAuthToken` request must be provided. This leads to the next phase of jUDDI authentication – the identify phase.

The identify phase is responsible for turning the authentication token (or the publisher id associated with that authentication token) into a valid `UddiEntityPublisher` object. The `UddiEntityPublisher` object contains all the properties necessary to handle ownership of UDDI entities. Thus, the token (or publisher id) is used to “identify” the publisher.

The two phases provide compliance with the UDDI authentication structure and grant flexibility for users that wish to provide their own authentication mechanism. Handling of credentials and publisher properties can be done entirely outside of jUDDI. However, jUDDI provides the Publisher entity, which is a sub-class of `UddiEntityPublisher`, to persist publisher properties within jUDDI. This is used in the default authentication and is the subject of the next section.

3.2. jUDDI Authentication

The default authentication mechanism provided by jUDDI is the `JUDDIAuthenticator`. The authenticate phase of the `JUDDIAuthenticator` simply checks to see if the user id passed in has an associated record in the Publisher table. No credentials checks are made. If, during authentication, the publisher does not exist, it the publisher is added on the fly.

Warning

Do not use jUDDI authentication in production.

The identify phase uses the publisher id to retrieve the Publisher record and return it. All necessary publisher properties are populated as Publisher inherits from `UddiEntityPublisher`.

```
juddi.auth = org.apache.juddi.auth.JUDDIAuthentication
```

3.3. XMLDocAuthentication

The `XMLDocAuthentication` implementation needs a XML file on the classpath. The `juddi.properties` file would need to look like

```
juddi.auth = org.apache.juddi.auth.XMLDocAuthentication
juddi.usersfile = juddi-users.xml
```

where the name of the XML can be provided but it defaults to `juddi-users.xml`, and the content of the file would looks something like

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<juddi-users>
  <user userid="anou_mana" password="password" />
  <user userid="bozo" password="clown" />
  <user userid="sviens" password="password" />
</juddi-users>
```

The authenticate phase checks that the user id and password match a value in the XML file. The identify phase simply uses the user id to populate a new `UddiEntityPublisher`.

3.4. CryptedXMLDocAuthentication

The `CryptedXMLDocAuthentication` implementation is similar to the `XMLDocAuthentication` implementation, but the passwords are encrypted

```
juddi.auth = org.apache.juddi.auth.CryptedXMLDocAuthentication
juddi.usersfile = juddi-users-encrypted.xml
juddi.cryptor = org.apache.juddi.cryptor.DefaultCryptor
```

where the name user credential file is `juddi-users-encrypted.xml`, and the content of the file would looks something like

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<juddi-users>
   <user userid="anou_mana" password="+j/kXkZJftwTFTBH6Cf6IQ=="/>
   <user userid="bozo" password="Na2Ait+2aW0="/>
   <user userid="sviens" password="+j/kXkZJftwTFTBH6Cf6IQ=="/>
</juddi-users>
```

The `DefaultCryptor` implementation uses `BEWithMD5AndDES` and `Base64` to encrypt the passwords. Note that the code in the `AuthenticatorTest` can be used to learn more about how to use this Authenticator implementation. You can plugin your own encryption algorithm by implementing the `org.apache.juddi.cryptor.Cryptor` interface and referencing your implementation class in the juddi.cryptor property.

The authenticate phase checks that the user id and password match a value in the XML file. The identify phase simply uses the user id to populate a new `UddiEntityPublisher`.

3.5. JBoss Authentication

Finally is it possible to hook up to third party credential stores. If for example jUDDI is deployed to the JBoss Application server it is possible to hook up to it's authentication machinery. The `JBossAuthenticator` class is provided in the `docs/examples/auth` directory. This class enables jUDDI deployments on JBoss use a server security domain to authenticate users.

To use this class you must add the following properties to the `juddi.properties` file:

```
juddi.auth=org.apache.juddi.auth.JBossAuthenticator
juddi.securityDomain=java:/jaas/other
```

The juddi.auth property plugs the `JbossAuthenticator` class into the jUDDI the Authenticator framework. The `juddi.sercuity.domain`, configures the `JBossAuthenticator` class where it can lookup the application server's security domain, which it will use to perform the authentication. Note that JBoss creates one security domain for each application policy element on the `$JBOSS_HOME/server/default/conf/login-config.xml` file, which gets bound to the server JNDI tree with name `java:/jaas/<application-policy-name>`. If a lookup refers to a non existent application policy it defaults to a policy named other.

Database Setup

4.1. Derby Out-of-the-Box

By default jUDDI uses an embedded Derby database. This allows us to build a downloadable distribution that works out-of-the-box, without having to do any database setup work. We recommend switching to an enterprise-level database before going to production. JUDDI uses the Java Persistence API (JPA) in the back end and we've tested with both OpenJPA and Hibernate. To configure which JPA provider you want to use, you will need to edit the configuration in the `persistence.xml`. This file can be found in the `juddi.war/WEB-INF/classes/META-INF/persistence.xml`

For Hibernate the content of this file looks like

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
  version="1.0">
  <persistence-unit name="juddiDatabase" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:comp/env/jdbc/JuddiDS</jta-data-source>
    <!-- entity classes -->
    <class>org.apache.juddi.model.Address</class>
    <class>org.apache.juddi.model.AddressLine</class>
    <class>org.apache.juddi.model.AuthToken</class>
    <class>org.apache.juddi.model.BindingCategoryBag</class>
    <class>org.apache.juddi.model.BindingDescr</class>
    <class>org.apache.juddi.model.BindingTemplate</class>
    <class>org.apache.juddi.model.BusinessCategoryBag</class>
    <class>org.apache.juddi.model.BusinessDescr</class>
    <class>org.apache.juddi.model.BusinessEntity</class>
    <class>org.apache.juddi.model.BusinessIdentifier</class>
    <class>org.apache.juddi.model.BusinessName</class>
    <class>org.apache.juddi.model.BusinessService</class>
    <class>org.apache.juddi.model.CategoryBag</class>
    <class>org.apache.juddi.model.Contact</class>
    <class>org.apache.juddi.model.ContactDescr</class>
    <class>org.apache.juddi.model.DiscoveryUrl</class>
    <class>org.apache.juddi.model.Email</class>
    <class>org.apache.juddi.model.InstanceDetailsDescr</class>
    <class>org.apache.juddi.model.InstanceDetailsDocDescr</class>
```

```
        <class>org.apache.juddi.model.KeyedReference</class>
        <class>org.apache.juddi.model.KeyedReferenceGroup</class>
        <class>org.apache.juddi.model.OverviewDoc</class>
        <class>org.apache.juddi.model.OverviewDocDescr</class>
        <class>org.apache.juddi.model.PersonName</class>
        <class>org.apache.juddi.model.Phone</class>
        <class>org.apache.juddi.model.Publisher</class>
        <class>org.apache.juddi.model.PublisherAssertion</class>
        <class>org.apache.juddi.model.PublisherAssertionId</class>
        <class>org.apache.juddi.model.ServiceCategoryBag</class>
        <class>org.apache.juddi.model.ServiceDescr</class>
        <class>org.apache.juddi.model.ServiceName</class>
        <class>org.apache.juddi.model.ServiceProjection</class>
        <class>org.apache.juddi.model.Subscription</class>
        <class>org.apache.juddi.model.SubscriptionChunkToken</class>
        <class>org.apache.juddi.model.SubscriptionMatch</class>
        <class>org.apache.juddi.model.Tmodel</class>
        <class>org.apache.juddi.model.TmodelCategoryBag</class>
        <class>org.apache.juddi.model.TmodelDescr</class>
        <class>org.apache.juddi.model.TmodelIdentifier</class>
        <class>org.apache.juddi.model.TmodelInstanceInfo</class>
        <class>org.apache.juddi.model.TmodelInstanceInfoDescr</class>
        <class>org.apache.juddi.model.TransferToken</class>
        <class>org.apache.juddi.model.TransferTokenKey</class>
        <class>org.apache.juddi.model.UddiEntity</class>
        <class>org.apache.juddi.model.UddiEntityPublisher</class>

        <properties>
            <property name="hibernate.archive.autodetection" value="class"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="false"/>
            <property name="hibernate.dialect"
                value="org.hibernate.dialect.DerbyDialect"/>
        </properties>
    </persistence-unit>
</persistence>
```

The `persistence.xml` reference a datasource “java:comp/env/jdbc/JuddiDS”. Datasource deployment is Application Server specific. If you are using Tomcat, then the datasource is defined in `juddi/META-INF/context.xml` which by default looks like

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
```

```
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <!-- -->
    <Resource name="jdbc/JuddiDS" auth="Container"
        type="javax.sql.DataSource" username="" password=""
        driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
        url="jdbc:derby:juddi-derby-test-db;create=true"
        maxActive="8"
        />
</Context>
```

4.2. Switch to MySQL

To switch over to MySQL you need to add the mysql driver (i.e. The `mysql-connector-java-5.1.6.jar`) to the classpath and you will need to edit the `persistence.xml`

```
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
```

and the datasource. For tomcat you the `context.xml` should look something like

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <Resource name="jdbc/JuddiDS" auth="Container"
        type="javax.sql.DataSource" username="root" password=""
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://localhost:3306/juddiv3"
        maxActive="8"/>
</Context>
```

Warning

Tomcat copies the `context.xml` to `conf/CATALINA/localhost/juddiv3.xml`, and if you update the `context.xml` it may not update this copy. You should simply delete the `juddiv3.xml` file after updating the `context.xml`.

To create a MySQL database name `juddiv3` use

```
mysql> create database juddiv3
```

and finally you probably want to switch to a user which is a bit less potent then '`root`'.

4.3. Switch to Postgres

This was written from a JBoss - jUDDI perspective. Non-JBoss-users may have to tweak this a little bit, but for the most part, the files and information needed is here.

Logged in as postgres user, access psql:

```
# psql

postgres= CREATE USER juddi with PASSWORD 'password';
postgres= CREATE DATABASE juddi;
postgres= GRANT ALL PRIVILEGES ON DATABASE juddi to juddi;
```

Note, for this example, my database is called juddi, as is the user who has full privileges to the database. The user 'juddi' has a password set to 'password'.

```
<datasources>
    <local-tx-datasource>
        <jndi-name>JuddiDS</jndi-name>
        <connection-url>jdbc:postgresql://localhost:5432/juddi</connection-url>
        <driver-class>org.postgresql.Driver</driver-class>
        <user-name>juddi</user-name>
        <password>password</password>
        <!-- sql to call when connection is created.  Can be anything,
        select 1 is valid for PostgreSQL
        <new-connection-sql>select 1</new-connection-sql>
        -->
        <!-- sql to call on an existing pooled connection when it is obtained
        from pool.  Can be anything, select 1 is valid for PostgreSQL
        <check-valid-connection-sql>select 1</check-valid-connection-sql>
        -->
        <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml -->
        <metadata>
            <type-mapping>PostgreSQL 8.0</type-mapping>
        </metadata>
    </local-tx-datasource>
</datasources>
```

In `persistence.xml`, reference the correct JNDI name of the datasource and remove the derby Dialect and add in the postgresql Dialect:

```
<jta-data-source>java:comp/env/jdbc/JuddiDS</jta-data-source>
    .....
```

```
<property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
```

Be sure to have `postgresql-8.3-604.jdbc4.jar` in the `lib` folder!

4.4. Switch to <other db>

If you use another database, please document, and send us what you had to change to make it work and we will include it here.

Chapter 5.

Root Seed Data

5.1. Introduction

As of UDDI v3, each registry need to have a “`root`” publisher. The root publisher is the owner of the UDDI services (inquiry, publication, etc). There can only be one root publisher per node. JUDDI ships some default seed data for the root account. The default data can be found in the `juddi-core-3.x.jar`, under `juddi_install_data/`. By default jUDDI installs two Publishers: “`root`” and “`uddi`”. Root owns the root partition, and uddi owns all the other seed data such as pre-defined tModels.

5.2. Seed Data Files

For each publisher there are four seed data files that will be read the first time you start jUDDI:

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

For example the content of the `root_Publisher.xml` looks like

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

Each publisher should have its own key generator schema so that custom generated keys cannot end up being identical to keys generated by other publishers. It is therefor that the each publisher need to define their own KenGenerator tModel. The tModel Key Generator is defined in the file `root_tModelKeyGen.xml` and the content of this file is

```
<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 legal format of keys used by the root publisher need to be in the form `uddi:juddi.apache.org:<text-of-chioce>` The use of other types of format will lead to an 'illegal key' error. The root publisher can only own one KeyGenerator while any other publisher can own more then one KeyGenerator. KeyGenerators should not be shared unless there is a good reason to do so. If you want to see your publisher with more then just the one KeyGenerator tModel, you can use the `<publisher>_tModels.xml` file.

Finally, in the `<publisher>_BusinessEntity.xml` file can be used to setup Business and Service data. In the `root_BusinessEntity.xml` we specified the ASF Business, and the UDDI services; Inquiry, Publish, etc.:

```
<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://${juddi.server.name}:${juddi.server.port}/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://${juddi.server.name}:${juddi.server.port}/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>
```

Note that the seeding process only kicks off if no publishers exist in the database. So this will only work with a clean database.

5.3. Tokens in the Seed Data

You may have noticed the tokens in the `root_BusinessEntity.xml` file (`${juddi.server.name}` and `${juddi.server.port}`) . The value of these tokens can set in the `juddiv3.properties` file, and you can add your own tokens.

5.4. Customer Seed Data

In your deployment you probably do not want to use the Seed Data shipped with the default jUDDI install. The easiest way to overwrite this data is to add it to a directory call `juddi_custom_install_data` in the `juddiv3.war/WEB-INF/classes/` directory. That way you don't have to modify the `juddi-core-3.x.jar`. Additionally if your root publisher is not called “root” you will need to set the juddi.root.publisher property in the `juddiv3.properties` file to something other then

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

The `juddiv3.war` ships with two example data directory. One for the Sales Affiliate, and one for the Marketing Affiliate. To use the Sales Seed Data, in the `juddiv3.war/WEB-INF/classes/`, rename the directory

```
mv RENAME4Sales_juddi_custom_install_data juddi_custom_install_data
```

before you start jUDDI the first time. It will then use this data to populate the database. If you want to rerun you can trash the database it created and restart tomcat. Don't forget to set the tokens in the `juddiv3.properties` file.

jUDDI_Configuration

6.1. Introduction

jUDDI will look for a juddiv3.properties file on the root of the classpath. In the `juddiv3.war` you can find it in `juddiv3.war/WEB_INF/classes/juddiv3.properties`

6.2. Startup

6.3. Subscription

```
# Days before a subscription expires
juddi.subscription.expiration.days=365
```

This is the upper boundary set by the registry. Between the user defined endDate of a Subscription and this value, the registry will pick the earliest date.

```
# Minutes before a "chunked" subscription call expires
juddi.subscription.chunkexpiration.minutes=5
```

This is the expiration time of a subscription “chunk”.

Using the jUDDI-Client

7.1. Introduction

The jUDDI project includes a jUDDI-Client (`juddi-client-3.0.0.jar`) which can be use to connect to the Registry. The client uses the UDDI v3 API so it should be able to connect to any UDDI v3 compliant registry, however we have only tested it with jUDDIv3. It maybe useful to take a look at the unit-tests in the jUDDIv3-uddi-client module to see how the client can be used.

7.2. Configuration

The UDDI client has a configuration file called `uddi.xml`. In this file you can set the type “Transport” used by the client to talk to the registry. The client tries to locate this file on the classpath and uses Apache Commons Configuration [COM-CONFIG] to read it. By default the `uddi.xml` file looks like

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

7.3. JAX-WS Transport

Using the settings in the `uddi.xml` file from above, 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 WS stack (such as CXF, Axis2 or JBossWS). Make sure to point the JAXWS URLs to 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/services/custody-transfer?wsdl
    </custodyTransferUrl>
    <inquiryUrl>
        http://${serverName}:${serverPort}/juddiv3/services/inquiry?wsdl
    </inquiryUrl>
    <publishUrl>
        http://${serverName}:${serverPort}/juddiv3/services/publish?wsdl
    </publishUrl>
    <securityUrl>
        http://${serverName}:${serverPort}/juddiv3/services/security?wsdl
    </securityUrl>
    <subscriptionUrl>
        http://${serverName}:${serverPort}/juddiv3/services/subscription?wsdl
    </subscriptionUrl>
    <subscriptionListenerUrl>
        http://${serverName}:${serverPort}/juddiv3/services/subscription-listener?wsdl
    </subscriptionListenerUrl>
    <juddiApiUrl>
        http://${serverName}:${serverPort}/juddiv3/services/juddi-api?wsdl
    </juddiApiUrl>
```

7.4. RMI Transport

If jUDDIv3 is deployed to an Application Server it is possible to register the UDDI Services as RMI services. If this is desired you need to edit the `juddiv3.war/WEB-INF/classes/juddiv3.properties` file, on the server. Add the following setting

```
juddi.jndi.registration=true
```

Now at deployment time, the RMI based UDDI services are bound into the Global JNDI namespace.

juddi (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`)

Next, on the client side you need to comment out the JAXWS section in the `uddi.properties` file and use the RMI Transport section instead. Optionally you can set the `java.naming.*` properties. In this case we specified setting for connecting to jUDDIv3 deployed to a JBoss Application Server. If you like you can set the `java.naming.*` properties in a `jndi.properties` file, or as System parameters.

7.5. InVM Transport

If you choose to use InVM Transport this means that the jUDDIv3 server is running in the same VM as you client. If you are deploying to `juddi.war` the server will be started by the `org.apache.juddi.RegistryServlet`, but if you are running outside any container, you are responsible for starting and stopping the `org.apache.juddi.Registry` Service yourself. Make sure to call

```
Registry.start()
```

before making any calls to the Registry, and when you are done using the Registry (on shutdown) call

```
Registry.stop()
```

so the Registry can release any resources it may be holding. To use InVM Transport uncomment this section in the `uddi.properties` while commenting out the JAXWS and RMI Transport sections.

7.6. Dependencies

The UDDI client depends on `uddi-ws-3.0.0.jar`, `commons-configuration-1.5.jar`, `commons-collection-3.2.1.jar` and `log4j-1.2.13.jar`, plus

- libraries for JAXB if you are not using JDK5.
- JAXWS client libraries when using JAXWS transport (like CXF).
- RMI and JNDI client libraries when using RMI Transport.

7.7. Sample Code

Sample code on how to use the UDDI client can be found in the `uddi-client` module on the jUDDIv3 project. Usually the first thing you want to is to 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();
    }
}
```

Make sure that the publisher, in this case “root” is an existing publisher in the Registry and that you are supplying the correct credential to get a successful response. If needed check *Chapter 3, Authentication* to learn more about this subject.

Another place to look for sample code is the `docs/examples/helloword` directory. Alternatively you can use annotations.

UDDI Annotations

8.1. Introduction

8.2. @UDDIService

8.3. @UDDIServiceBinding

Subscription

9.1. Introduction

Subscriptions come to play in a multi-registry setup. Within your company you may have the need to run with more then one UDDI, let's say one for each department, where you limit access to the systems in each department to just their own UDDI node. However you may want to share some services cross departments. The subscription API can help you cross registering those services and keeping them up to date by sending out notifications as the registry information in the parent UDDI changes.

There are two type of subscriptions:

asynchronous
: Save a subscription, and receive updates on a certain schedule.

synchronous
: Save a subscription and invoke the get_Subscription and get a synchronous reply.

The notification can be executed in a synchronous and an asynchronous way. The asynchronous way requires a listener service to be installed on the node to which the notifications should be sent.

9.2. Two node example setup: Sales and Marketing

In this example we are setting up a node for 'sales' and a node for 'marketing'. For this you need to deploy jUDDI to two different services, then you need to do the following setup:

Procedure 9.1. Setup Node 1: Sales

1. Create `juddi_custom_install_data`.

   ```
   cd juddiv3/WEB-INF/classes
   mv RENAME4SALES_juddi_custom_install_data juddi_custom_install_data
   ```

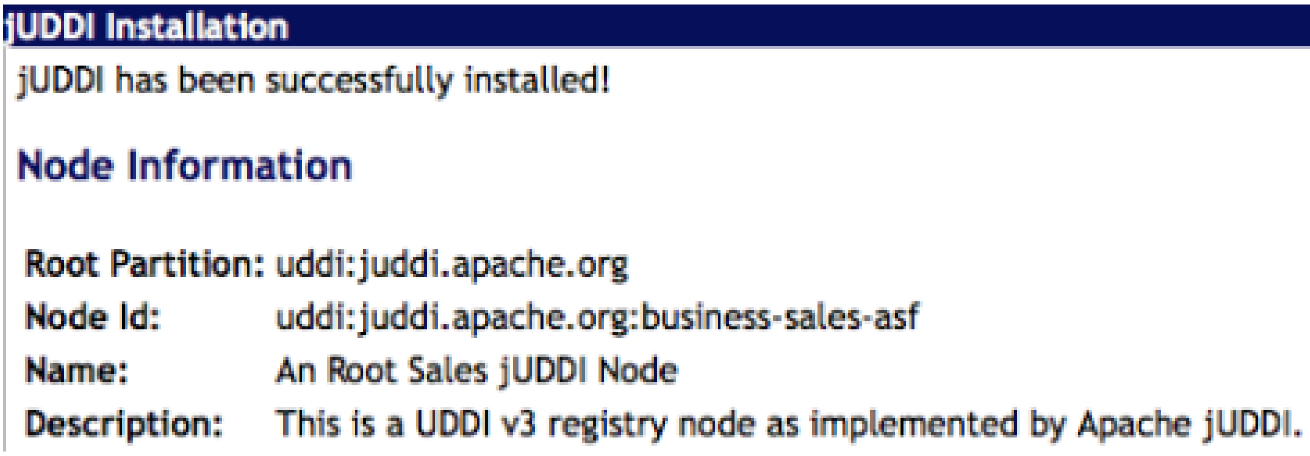
2. edit: `webapps/juddiv3/WEB-INF/classes/juddiv3.properties` and set the following property values where 'sales' is the DNS name of your server.

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

3. Start the server (tomcat), which will load the UDDI seed data (since this is the first time you're starting jUDDI, see *Chapter 5, Root Seed Data*)

   ```
   bin/startup.sh
   ```

4. Open your browser to *http://sales:8080/juddiv3*. You should see:



jUDDI Installation

jUDDI has been successfully installed!

Node Information

Root Partition: uddi:juddi.apache.org
Node Id: uddi:juddi.apache.org:business-sales-asf
Name: An Root Sales jUDDI Node
Description: This is a UDDI v3 registry node as implemented by Apache jUDDI.

Figure 9.1. Sales Node Installation

Procedure 9.2. Setup Node 2: Marketing

1. Create `juddi_custom_install_data`.

   ```
   cd juddiv3/WEB-INF/classes
   mv RENAME4MARKETING_juddi_custom_install_data juddi_custom_install_data
   ```

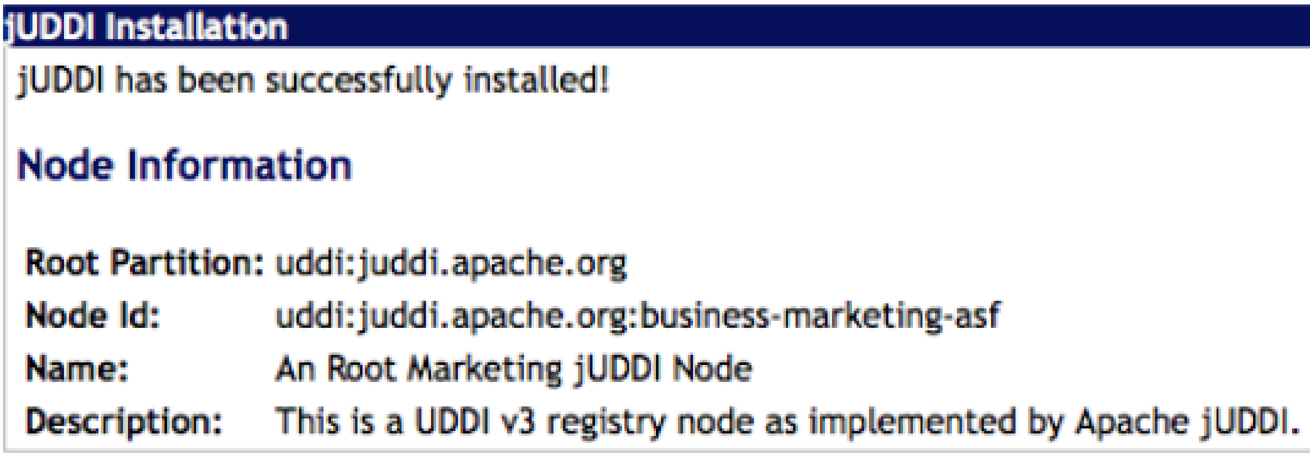
2. edit: `webapps/juddiv3/WEB-INF/classes/juddiv3.properties` and set the following property values where 'marketing' is the DNS name of your server.

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

3. Start the server (tomcat), which will load the UDDI seed data (since this is the first time you're starting jUDDI, see *Chapter 5, Root Seed Data*)

   ```
   bin/startup.sh
   ```

4. Open your browser to *http://marketing:8080/juddiv3* . You should see:



jUDDI Installation

jUDDI has been successfully installed!

Node Information

Root Partition:	uddi:juddi.apache.org
Node Id:	uddi:juddi.apache.org:business-marketing-asf
Name:	An Root Marketing jUDDI Node
Description:	This is a UDDI v3 registry node as implemented by Apache jUDDI.

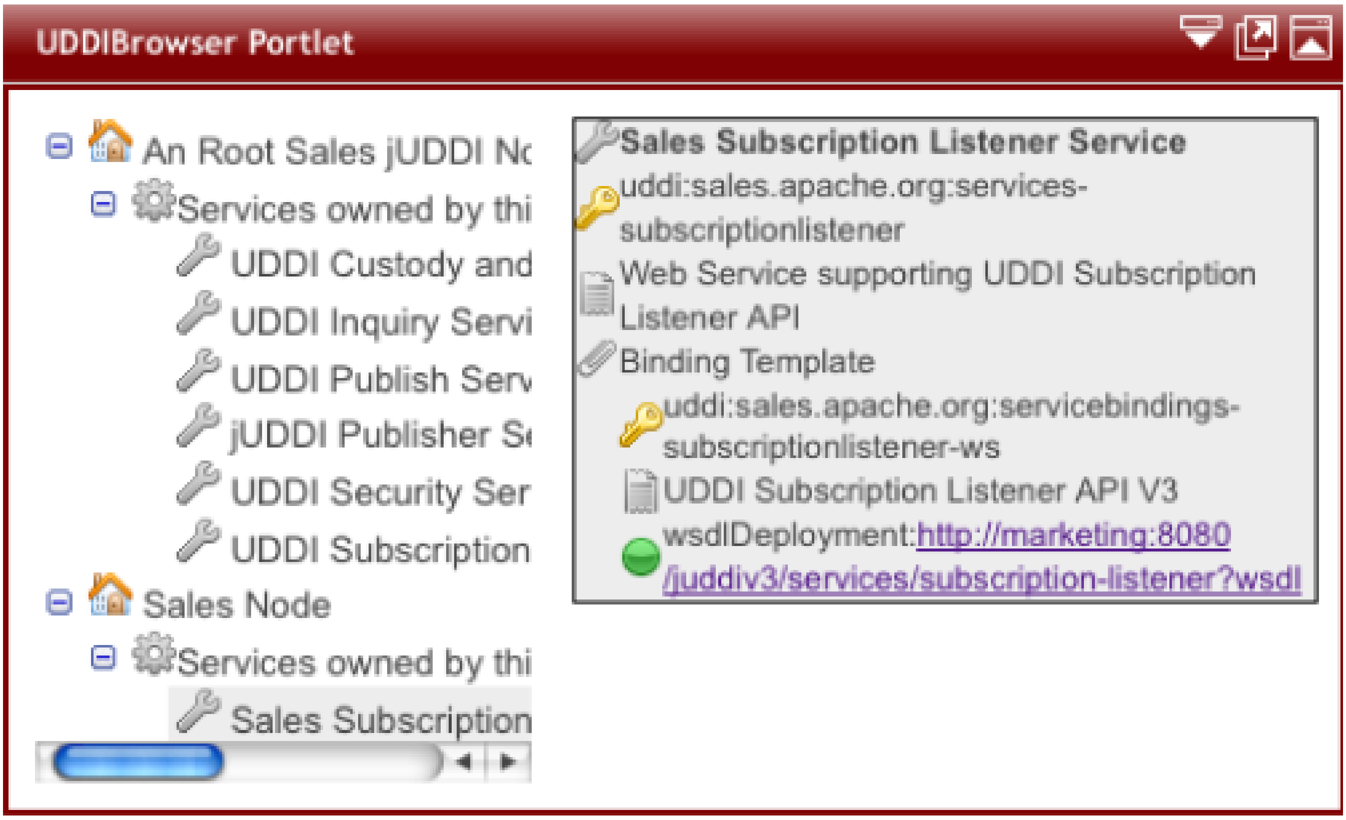
Figure 9.2. Marketing Node Installation

Note that we kept the root partition the same as sales and marketing are in the same company, however the Node Id and Name are different and reflect that this node is in 'sales' or 'marketing'.

Finally you will need to replace the sales server's `uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml` with `uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml.sales`. Then, edit the `uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml` and set the following properties:

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

Log into the sales portal: *http://sales:8080/pluto* with username/password: sales/sales.



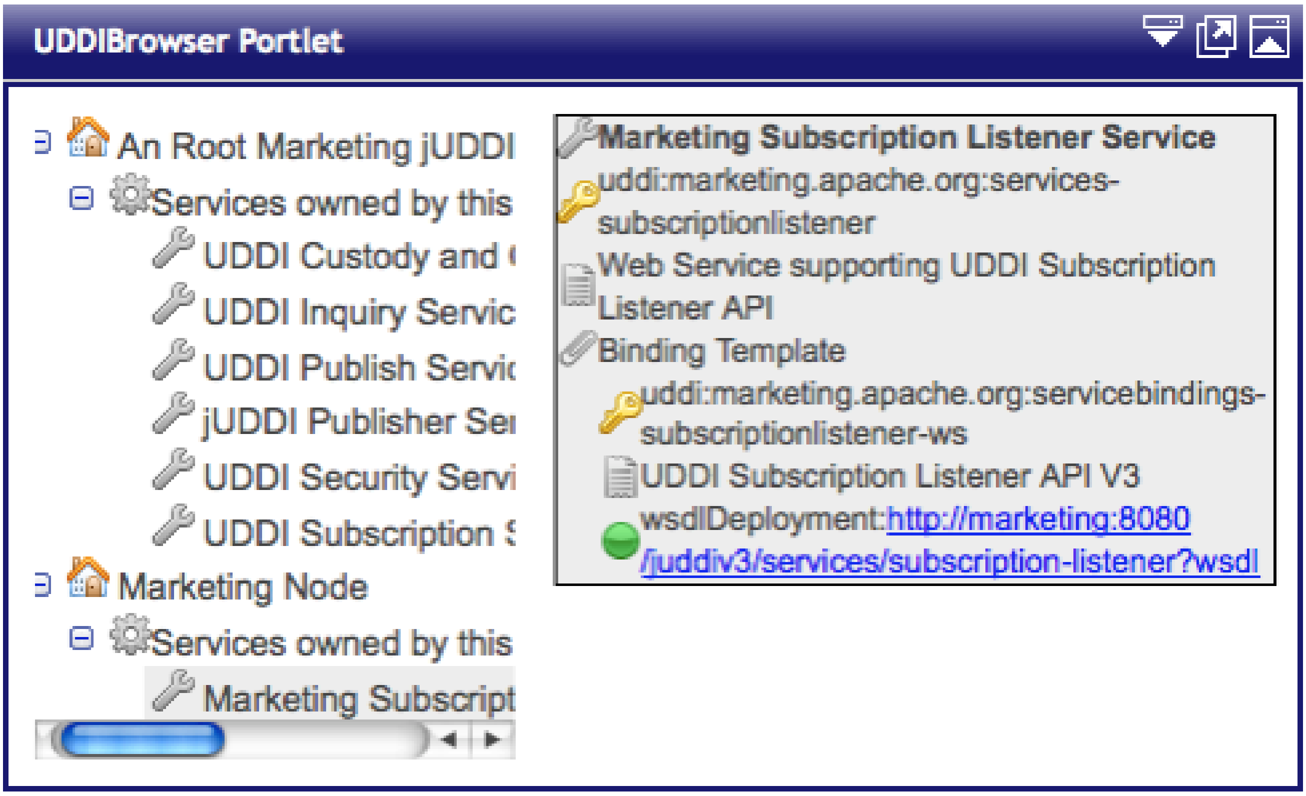
Sales Subscription Listener Service
uddi:sales.apache.org:services-subscriptionlistener
Web Service supporting UDDI Subscription Listener API
Binding Template
uddi:sales.apache.org:servicebindings-subscriptionlistener-ws
UDDI Subscription Listener API V3
wsdlDeployment:http://marketing:8080/juddiv3/services/subscription-listener?wsdl

Figure 9.3. Sales Services

Before logging into the marketing portal, replace marketing's `uddi-portlet.war/WEB-INF/classes/META-INF/uddi.xml` with `udd-portlet.war/WEB-INF/classes/META-INF/uddi.xml.marketing`. Then you will need to edit the `uddi-portlet.war/WEB-INF/classes/META_INF/uddi.xml` and set the following properties:

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

Now log into the marketing portal *http://marketing:8080/pluto* with username/password: marketing/marketing. In the browser for the marketing node we should now see:



Marketing Subscription Listener Service
uddi:marketing.apache.org:services-subscriptionlistener
Web Service supporting UDDI Subscription Listener API
Binding Template
uddi:marketing.apache.org:servicebindings-subscriptionlistener-ws
UDDI Subscription Listener API V3
wsdlDeployment:http://marketing:8080/juddiv3/services/subscription-listener?wsdl

Figure 9.4. Marketing Services

Note that the subscriptionlistener is owned by the Marketing Node business (and not the Root Marketing Node). The Marketing Node Business is managed by the marketing publisher.

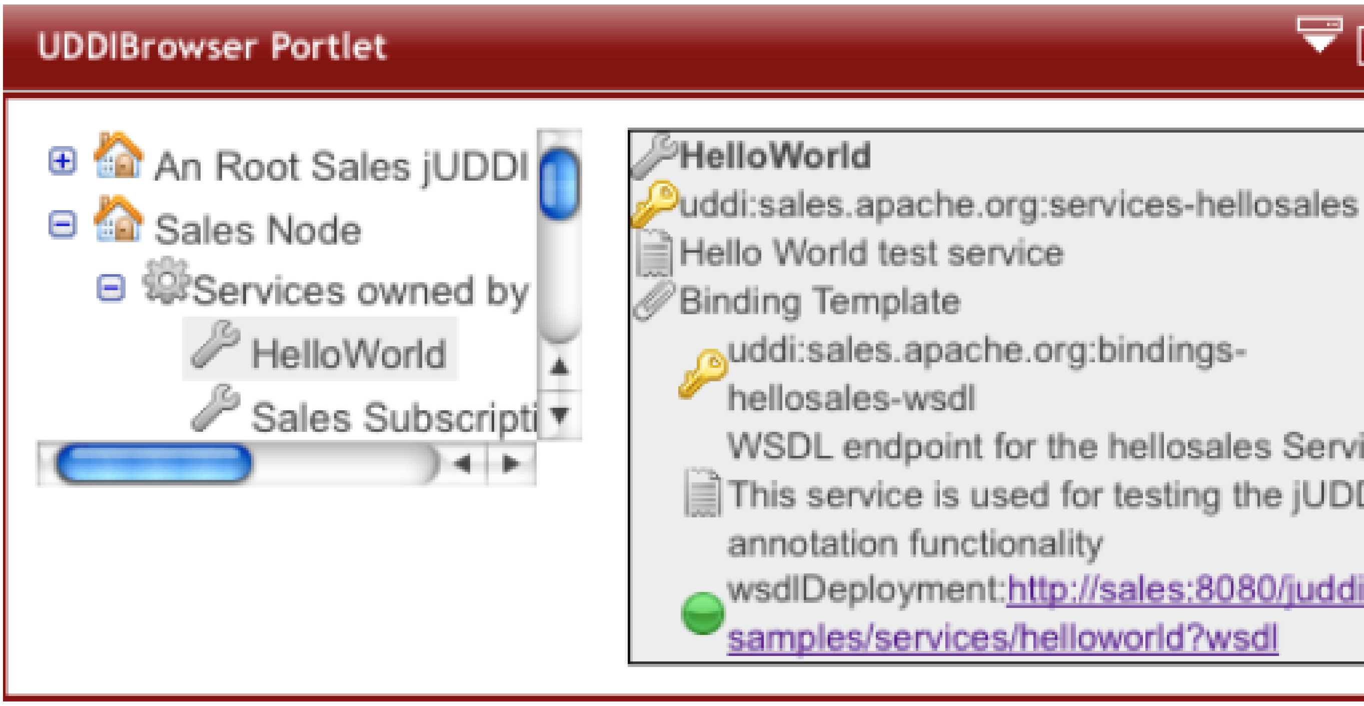
9.3. Deploy the HelloSales Service

The sales department developed a service called HelloSales. The HelloSales service is provided in the `juddiv3-samples.war`, and it is annotated so that it will auto-register. Before deploying the war, edit the `juddiv3-samples.war/WEB-INF/classes/META-INF/uddi.xml` file to set some 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</publishUrl>
        <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>
```

Now deploy the `juddiv3-samples.war` to the sales registry node, by building the `juddiv3-samples.war` and deploying. The HelloWorld service should deploy



HelloWorld
uddi:sales.apache.org:services-hellosales
Hello World test service
Binding Template
uddi:sales.apache.org:bindings-hellosales-wsdl
WSDL endpoint for the hellosales Servi
This service is used for testing the jUDD
annotation functionality
wsdlDeployment:http://sales:8080/juddi
samples/services/helloworld?wsdl

Figure 9.5. Registration by Annotation, deploying the `juddi-samples.war` to the sales Node

On the Marketing UDDI we'd like to subscribe to the HelloWord service, in the Sales UDDI Node. As mentioned before there are two ways to do this; synchronously and asynchronously.

9.4. Configure a user to create Subscriptions

For a user to create and save subscriptions the publisher needs to have a valid login to both the sales and the marketing node. Also if the marketing publisher is going to create registry objects in the marketing node, the marketing publisher needs to own the sales keygenerator tModel. Check the `marketing_*.xml` files in the root seed data of both the marketing and sales node, if you want to learn more about this. 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>
```

If we are going to user the marketing publisher to subscribe to updates in the sales registry, then we need to provide this publisher with two clerks in the `uddi.xml` of the `uddi-portlet.war`.

```
<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>
```

Here we created two clerks for this publisher called 'MarketingCratchit' and 'SalesCratchit'. This will allow the publisher to check the existing subscriptions owned by this publisher in each of the two systems.

9.5. Synchronous Notifications

While being logged in as the marketing publisher on the marketing portal, we should see the following when selecting the UDDISubscription Portlet.

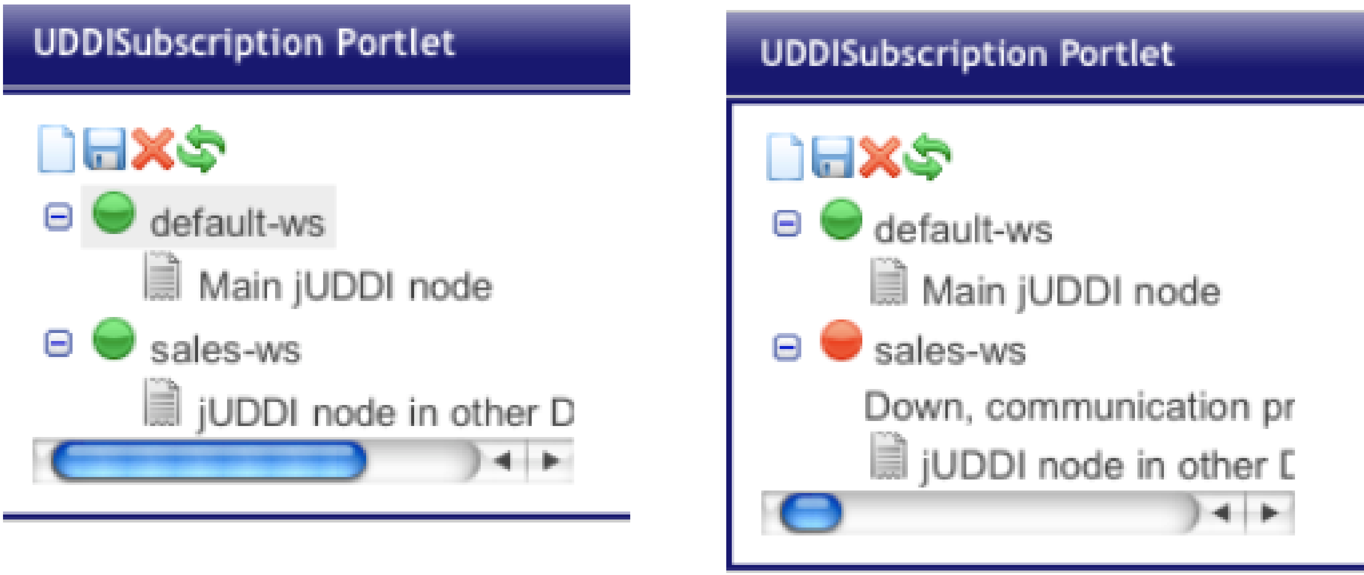
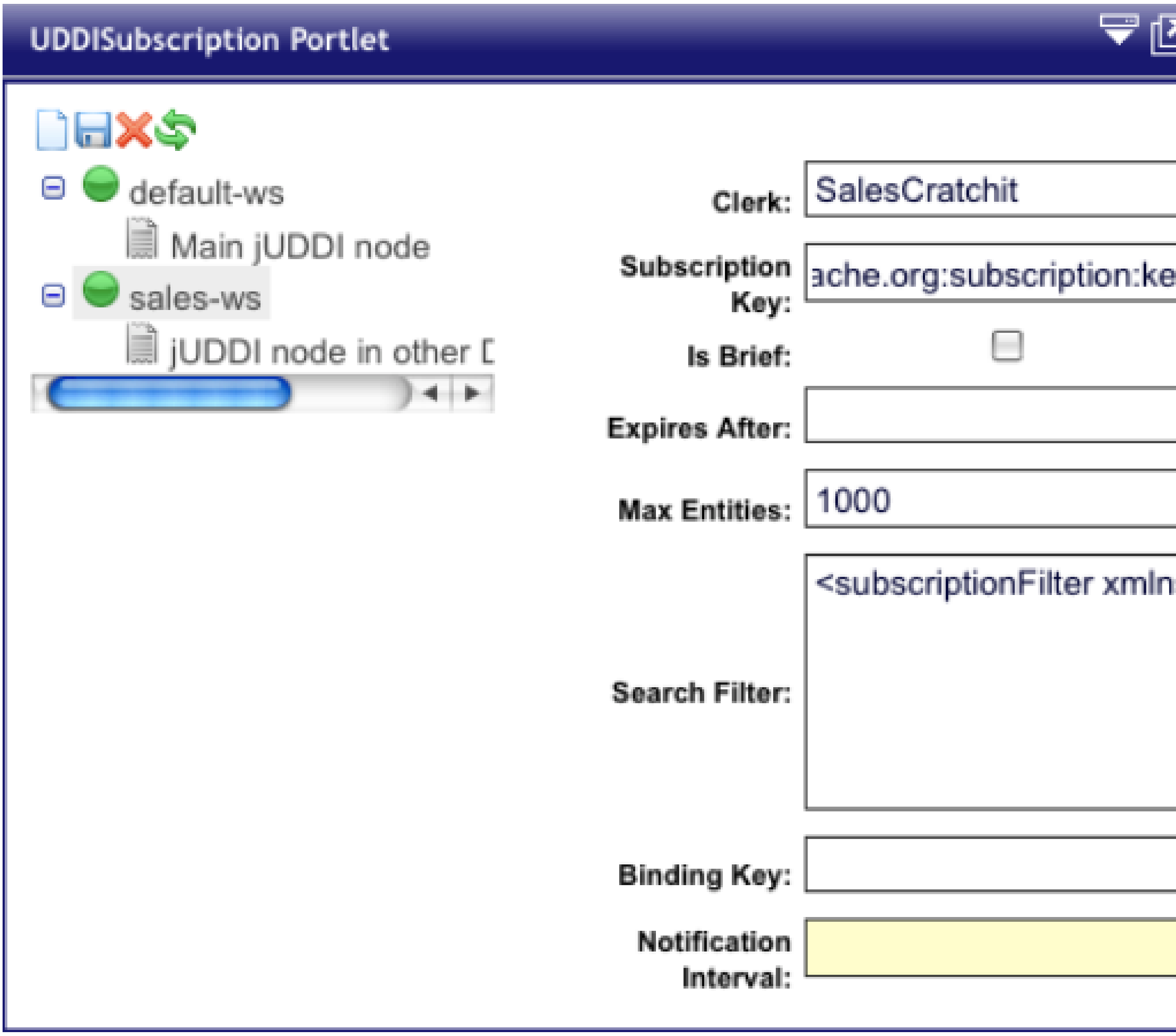


Figure 9.6. Subscriptions. In (a) both nodes are up while in (b) the sales node is down

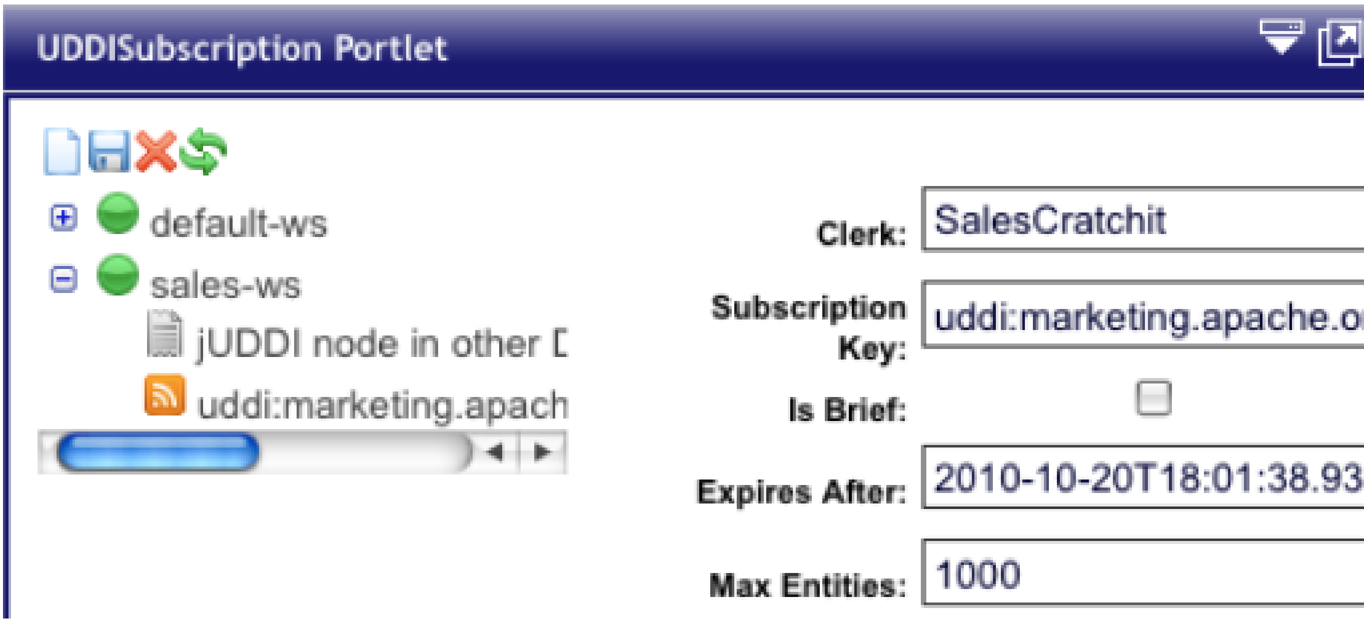
When both nodes came up green you can lick on the 'new subscription' icon in the toolbar. Since we are going to use this subscription synchronously only the Binding Key and Notification Interval should be left blank, as shown in *Figure 9.7, “Create a New Subscription”*. Click the save icon to save the subscription.



Clerk: SalesCratchit
Subscription Key: ache.org:subscription:ke
Is Brief: ☐
Expires After:
Max Entities: 1000
Search Filter: <subscriptionFilter xmln
Binding Key:
Notification Interval:

Figure 9.7. Create a New Subscription

Make sure that the subscription Key uses the convention of the keyGenerator of the marketing publisher. You should see the orange subscription icon appear under the “sales-ws” UDDI node.



Clerk: SalesCratchit

Subscription Key: uddi:marketing.apache.o

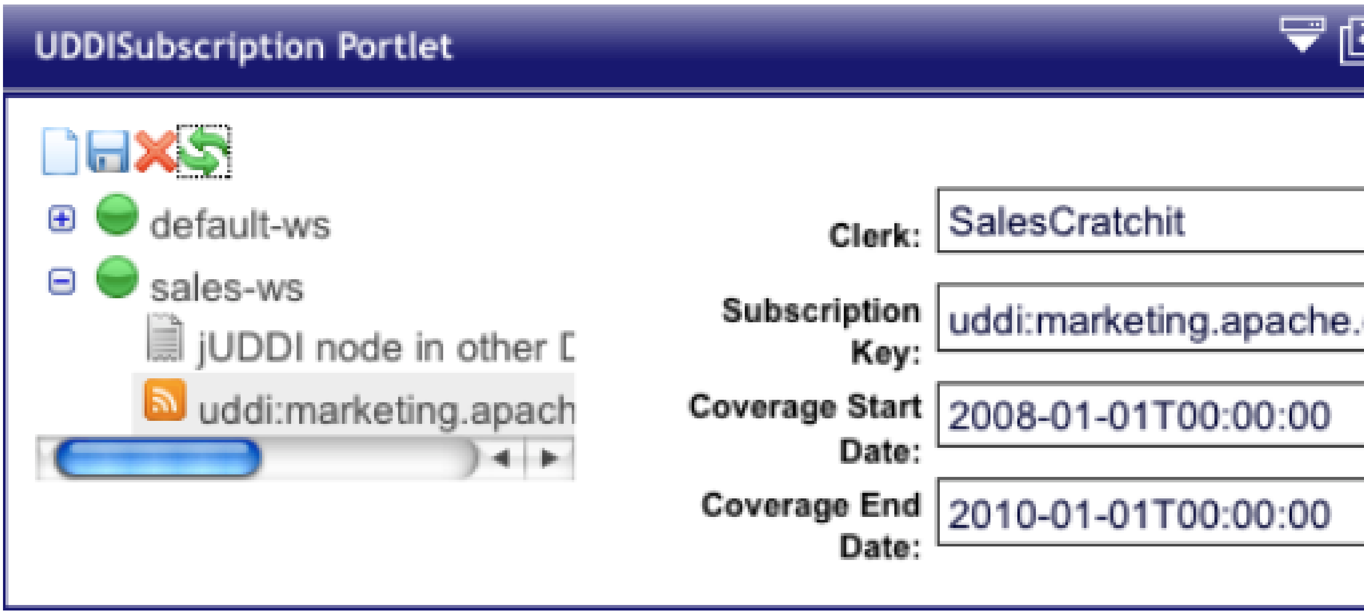
Is Brief: ☐

Expires After: 2010-10-20T18:01:38.93

Max Entities: 1000

Figure 9.8. A Newly Saved Subscription

To invoke a synchronous subscription, click the icon with the green arrows. This will give you the opportunity to set the coverage period.



Clerk: SalesCratchit

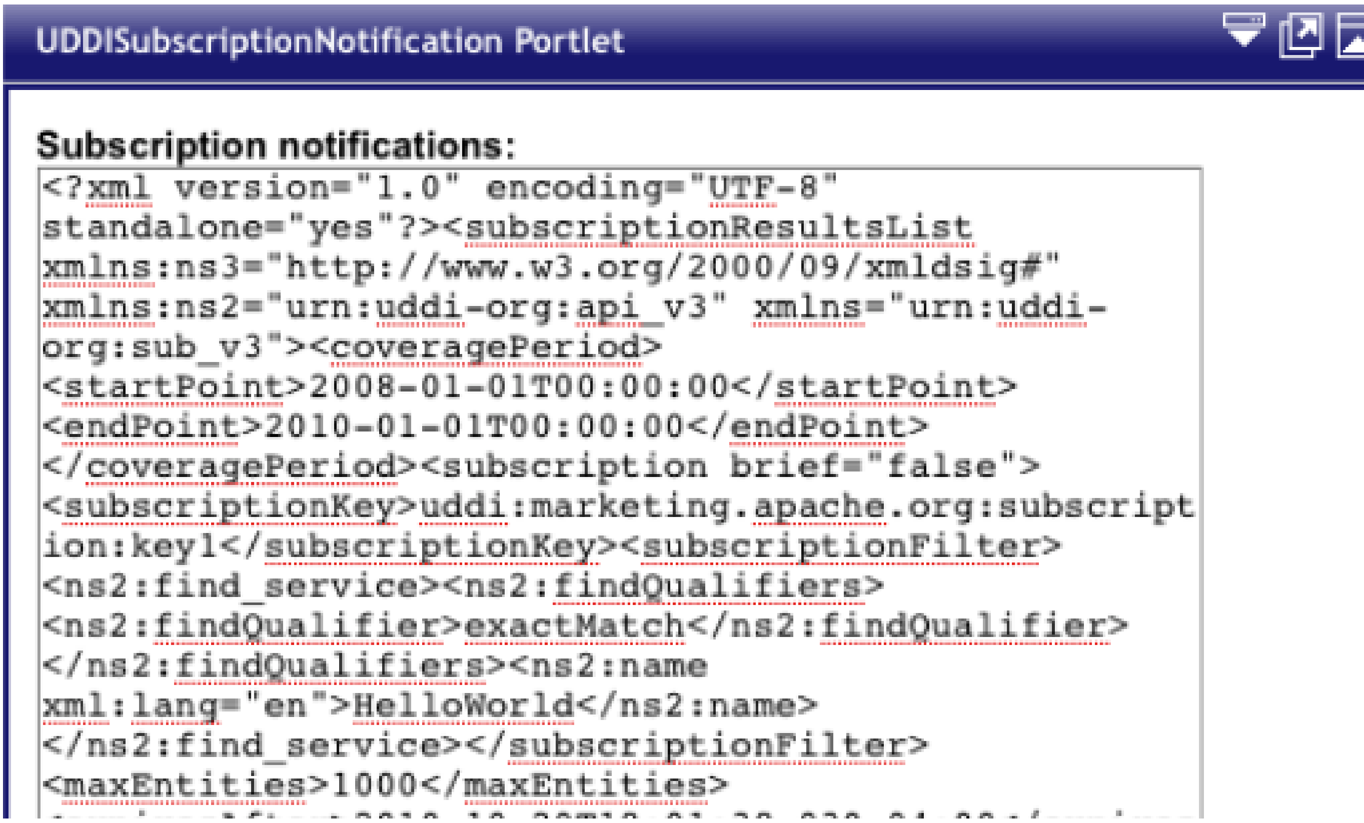
Subscription Key: uddi:marketing.apache.

Coverage Start Date: 2008-01-01T00:00:00

Coverage End Date: 2010-01-01T00:00:00

Figure 9.9. Set the Coverage Period

Click the green arrows icon again to invoke the synchronous subscription request. The example finder request will go out to the sales node and look for updates on the HelloWorld service. The raw XML response will be posted in the UDDISubscriptionNotification Portlet.

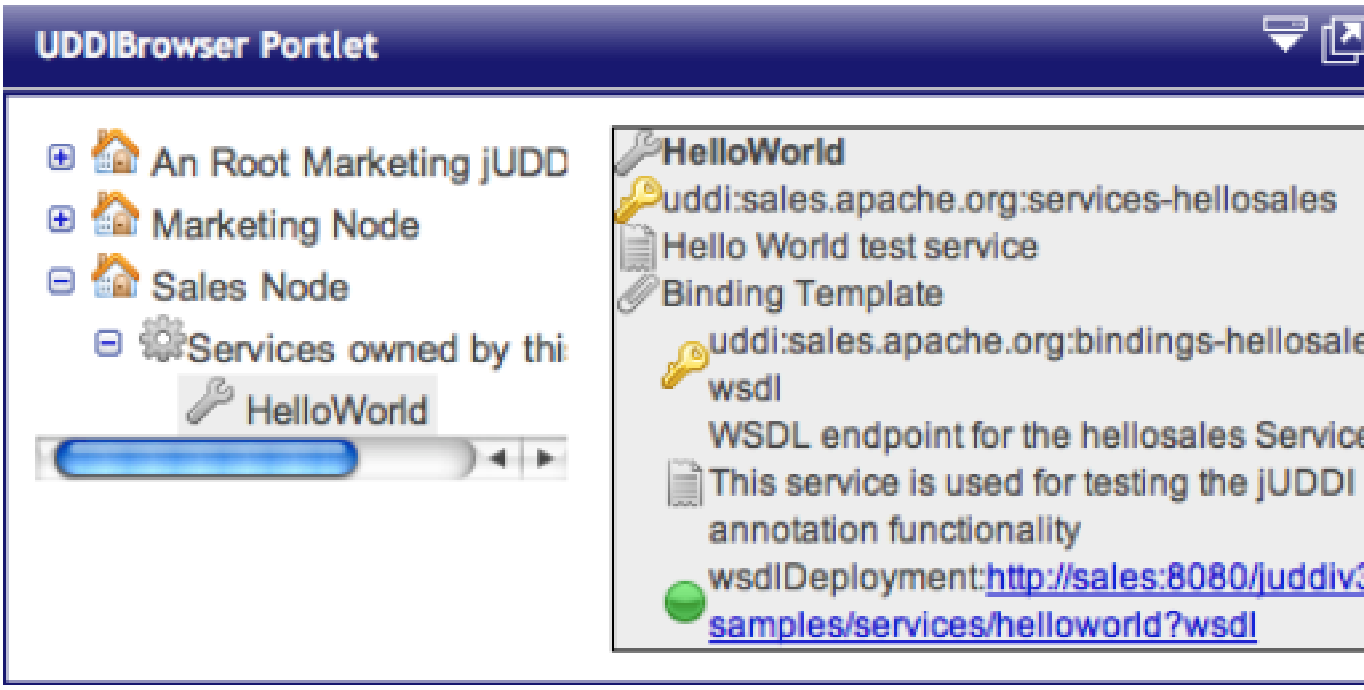


Subscription notifications:

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><subscriptionResultsList
xmlns:ns3="http://www.w3.org/2000/09/xmldsig#"
xmlns:ns2="urn:uddi-org:api_v3" xmlns="urn:uddi-
org:sub_v3"><coveragePeriod>
<startPoint>2008-01-01T00:00:00</startPoint>
<endPoint>2010-01-01T00:00:00</endPoint>
</coveragePeriod><subscription brief="false">
<subscriptionKey>uddi:marketing.apache.org:subscript
ion:key1</subscriptionKey><subscriptionFilter>
<ns2:find_service><ns2:findQualifiers>
<ns2:findQualifier>exactMatch</ns2:findQualifier>
</ns2:findQualifiers><ns2:name
xml:lang="en">HelloWorld</ns2:name>
</ns2:find_service></subscriptionFilter>
<maxEntities>1000</maxEntities>
```

Figure 9.10. The Raw XML response of the synchronous Subscription request

The response will also be consumed by the marketing node. The marketing node will import the HelloWorld subscription information, as well as the sales business. So after a successful sync you should now see three businesses in the Browser Portlet of the marketing node, see *Figure 9.11, “The registry info of the HelloWorld Service information was imported by the subscription mechanism.”*.



HelloWorld
uddi:sales.apache.org:services-hellosales
Hello World test service
Binding Template
uddi:sales.apache.org:bindings-hellosale
wsdl
WSDL endpoint for the hellosales Service
This service is used for testing the jUDDI annotation functionality
wsdlDeployment:http://sales:8080/juddiv samples/services/helloworld?wsdl

Figure 9.11. The registry info of the HelloWorld Service information was imported by the subscription mechanism.

Administration

10.1. Introduction

General Stuff about administration.

10.2. Changing the Listener Port

If you want to change the port Tomcat listens on to something non-standard (something other than 8080):

jUDDI Server

1. edit `conf/server.xml` and change the port within the `<Connector>` element
2. edit `webapps/juddiv3/WEB-INF/classes/juddiv3.properties` and change the port number

jUDDI Portal

1. edit `webapps/uddi-portlets/WEB-INF/classes/META-INF/uddi.xml` and change the port numbers within the endpoint URLs
2. edit `pluto/WEB-INF/classes/server.xml` and change the port within the `<Connector>` element

10.3. Changing the Oracle Sequence name

If you are using Hibernate as a persistence layer for jUDDI, then Oracle will generate a default sequence for you ("HIBERNATE_SEQUENCE"). If you are using hibernate elsewhere, you may wish to change the sequence name so that you do not share this sequence with any other applications. If other applications try to manually create the default hibernate sequence, you may even run into situations where you find conflicts or a race condition.

The easiest way to handle this is to create an `orm.xml` file and place it within the classpath in a META-INF directory, which will override the jUDDI persistence annotations and will allow you to specify a specific sequence name for use with jUDDI. The following `orm.xml` specifies a "juddi_sequence" sequence to be used with jUDDI.

```
<entity-mappings
  xmlns="http://java.sun.com/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm  http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
  version="1.0">
```

```
<sequence-generator name="juddi_sequence" sequence-name="juddi_sequence"/>

<entity class="org.apache.juddi.model.Address">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.AddressLine">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.BindingDescr">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.BusinessDescr">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.BusinessIdentifier">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.BusinessName">
  <attributes>
```

```
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.CategoryBag">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.Contact">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.ContactDescr">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.DiscoveryUrl">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.Email">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.InstanceDetailsDescr">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.InstanceDetailsDocDescr">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

 <entity class="org.apache.juddi.model.KeyedReference">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.KeyedReferenceGroup">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.OverviewDoc">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.OverviewDocDescr">
    <attributes>
```

```
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.PersonName">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.Phone">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.ServiceDescr">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.ServiceName">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
  </entity>

  <entity class="org.apache.juddi.model.SubscriptionMatch">
    <attributes>
      <id name="id">
        <generated-value generator="juddi_sequence" strategy="AUTO"/>
      </id>
    </attributes>
```

```
</entity>

<entity class="org.apache.juddi.model.TmodelDescr">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.TmodelIdentifier">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.TmodelInstanceInfo">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.TmodelInstanceInfoDescr">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.TransferTokenKey">
  <attributes>
    <id name="id">
      <generated-value generator="juddi_sequence" strategy="AUTO"/>
    </id>
  </attributes>
</entity>

<entity class="org.apache.juddi.model.BindingTemplate">
  <attributes>
```

```
   <basic name="accessPointUrl">
    <column name="access_point_url" length="4000"/>
   </basic>
  </attributes>
 </entity>
</entity-mappings>
```

Appendix A. Revision History

Revision History

Revision 1.1	Thu Jan 07 2010	TomCunningham`<tcunning@apache.org>`
Translated Dev Guide to docbook		
Revision 1.0	Mon Nov 16 2009	DarrinMison`<dmison@redhat.com>`
Created from community jUDDI Guide		