jUDDI User's Guide

1. Requirements

jUDDI is a pure Java web application and as such can be deployed to any application server or servlet engine that supports version 2.1 or later of the servlet API. If you need an application server, we recommend Jakarta Tomcat. Note also that jUDDI requires Java 1.3 or later. As with any Java web application, deployment to your application server or servlet engine will vary on a product-by-product basis.

Instructions for deploying jUDDI to several application servers have been donated and are available in the HOW-TO section of the jUDDI wiki.

jUDDI also requires an external datastore in which to persist the registry data it manages. Typically this is a relational database management system such as MySQL, Oracle or DB2. Support for several open source and commercial database products are included. See the section below titled Persistence for more information.

2. Configuration

To properly configure and deploy jUDDI it will be helpful to understand a bit about it's architecture. jUDDI consist of a core request processor that unmarshalls incoming UDDI requests, invoking the appropriate UDDI function and marshalling UDDI responses (marshalling and unmarshalling is the process of converting XML data to/from Java objects).

To invoke a UDDI function jUDDI employs the services of three configurable sub-components or modules that handle persistence (the DataStore), authentication (the Authenticator) and the generation of UUID's (the UUIDGen). jUDDI is bundled and pre-configured to use default implementations of each of these modules to help your registry up and running quickly. These sub-components and a description of the default implementations are described below.

Several public Java interfaces for creating your own DataStore, Authenticator and UUIDGen module implementations are available. Please see the module development section of this guide for more information regarding jUDDI module development.

3. Persistence (jUDDI DataStore)

jUDDI needs a place to store it's registry data so it should come as no surprise that jUDDI is pre-configured to use JDBC and any one of several different DBMSs to do this.

The process of setting this up is straight forward. Start by creating a new jUDDI database using the instructions for your preferred DBMS which you will find in the 'sql' directory of your jUDDI distribution. At the time of this writing instructions for the following products were available:

- MySQL
- DB2
- HSQLdb (HypersonicSQL)
- Sybase
- PostreSQL
- Oracle
- TotalXML
- JDataStore (Borland)

If support for your DBMS is not listed you might try posting a message to the juddi-user list to see if someone has already developed support for it. You can also try to create the scripts yourself by using the instructions for a supported DBMS as a guide. Please consider contributing your work back to the project so the next person won't have the same issue.

To complete the DataStore set up you'll need to configure a JNDI Datasource with a name of 'jdbc/juddiDB' in the application server or servlet engine that you're deploying to. Datasource setup varies on an product-by-product basis so review documentation for your application server. If you're deploying to Jakarta Tomcat, take a look at Tomcat's JNDI Datasource HOW-TO for assistance.

4. Authentication (jUDDI Authenticator)

Authenticating a jUDDI publisher is a two-step process. The first step confirms that the ID/password combination provided by the user in a get_authToken request is valid. The default Authenticator implementation simply approves any authentication attempt. It is expected that a typical jUDDI deployment will use an external authentication mechanism. It is our hope that additional jUDDI authentication implementations will be developed by jUDDI users as they determine how they would like authentication to take place in their particular environment. See the jUDDI Developers Guide for more information on developing a custom jUDDI authentication module for your environment.

The second step confirms that the publisher has been defined to jUDDI. A publisher is said to be defined when a row identifying the publisher exists in the PUBLISHER table of the jUDDI datastore. At the moment the only way to do this is via SQL. An example of defining a new publisher named John Doe would look like this:

```
INSERT INTO PUBLISHER (PUBLISHER_ID,PUBLISHER_NAME,ADMIN,ENABLED)
VALUES ('jdoe','John Doe','false','true');
```

The PUBLISHER table consists of several columns but only four of them are required and they are defined as follows:

Column Name	Description
PUBLISHER_ID	The user ID the publisher uses when authenticating. IMPORTANT: This should be the same value used to authenticate with the external authentication service.
PUBLISHER_NAME	The publisher's name (or in UDDI speak the Authorized Name).
ADMIN	Indicate if the publisher has administrative privileges. Valid values for this column are 'true' or 'false'. The ADMIN value is currently not used.
ENABLED	Indicate if the publishers account is enabled and eligible for use.

The jUDDI web application will (eventually) be extended to facilitate the Publisher creation process. The value of the ADMIN column in the PUBLISHER table above will be used to determine who has the privilege to create new jUDDI publishers.

5. UUID Generation (jUDDI UUIDGen)

There's nothing for you to do here but I thought I'd offer a little information about how, why and where jUDDI makes use of UUID generation.

The UDDI specification indicates that each Business, Service, Binding and TModel (Technical Model) is to be uniquely identified by a Universally Unique ID (UUID). Additionally, jUDDI also uses the UUID generator to create AuthTokens.

Generation of a UUID typically requires access to hardware level information that (unfortunately) is not easily accessible from Java. Fortunately, the UUID specification offers an alternative method for generating these ID's when this hardware information is not present. By default the jUDDI implements this alternative method.

6. Logging

When deploying jUDDI you may wish to make changes to the juddi.properties and log4j.properties files. These files are located in the juddi webapp's WEB-INF/classes directory. They're here because they need to be in the classpath for jUDDI to locate and load them at runtime. One Log4j property value that you'll most likely want to set is log4j.appender.LOGFILE.File which specifies the name and location of the jUDDI log file.

7. Module Development

Coming Soon.