

## **ESB Registry Guide**

**4.2**

# **JBoss Enterprise SOA Platform**



**ISBN:**

**Publication date: February, 2008**

The SOA Platform edition of the JBoss ESB Registry Guide

---

# ESB Registry Guide: JBoss Enterprise SOA Platform

Copyright © 2008 Red Hat, Inc.

Copyright © 2008 Red Hat, Inc.. This material may only be distributed subject to the terms and conditions set forth in the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License (which is presently available at <http://creativecommons.org/licenses/by-nc-sa/3.0/>).

Red Hat and the Red Hat "Shadow Man" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

The GPG fingerprint of the security@redhat.com key is:

CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

1801 Varsity Drive  
Raleigh, NC 27606-2072  
USA  
Phone: +1 919 754 3700  
Phone: 888 733 4281  
Fax: +1 919 754 3701  
PO Box 13588  
Research Triangle Park, NC 27709  
USA

---



---

Preface .....	vii
1. Document Conventions .....	vii
2. We Need Feedback .....	viii
1. What is the Registry? .....	1
1. Introduction .....	1
1.1. Why do I need it ? .....	1
1.2. How do I use it ? .....	1
1.3. Registry Vs Repository .....	1
1.4. SOA Components .....	2
1.5. UDDI .....	3
1.6. The Registry and JBossESB .....	3
2. Configuring the Registry .....	5
1. Introduction .....	5
2. The components involved .....	6
3. The Registry Implementation Class .....	6
4. Using JAXR .....	7
5. Using Scout and jUDDI .....	7
3. Configuration Examples .....	9
1. Introduction .....	9
2. Embedded .....	9
3. RMI using the juddi.war or jbossesb.sar .....	10
4. RMI using your own JNDI Registration of the RMI Service .....	11
5. SOAP .....	14
4. UDDI Browser .....	17
1. Introduction .....	17
2. UB setup .....	17
5. Troubleshooting .....	21
1. Scout and jUDDI pitfalls .....	21
2. More Information .....	21

---

---

## Preface

# 1. Document Conventions

Certain words in this manual are represented in different fonts, styles, and weights. This highlighting indicates that the word is part of a specific category. The categories include the following:

*Courier font*

Courier font represents `commands, file names and paths, and prompts`.

When shown as below, it indicates computer output:

```
Desktop      about.html    logs          paulwesterberg.png
Mail         backupfiles   mail          reports
```

***Courier font***

Bold Courier font represents text that you are to type, such as: `service jonas start`

If you have to run a command as root, the root prompt (`#`) precedes the command:

```
# gconftool-2
```

*italic Courier font*

Italic Courier font represents a variable, such as an installation directory:

```
install_dir/bin/
```

**bold font**

Bold font represents **application programs** and **text found on a graphical interface**.

When shown like this: **OK**, it indicates a button on a graphical application interface.

Additionally, the manual uses different strategies to draw your attention to pieces of information. In order of how critical the information is to you, these items are marked as follows:



### Note

A note is typically information that you need to understand the behavior of the system.



### Tip

A tip is typically an alternative way of performing a task.



### Important

Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.



### Caution

A caution indicates an act that would violate your support agreement, such as recompiling the kernel.



### Warning

A warning indicates potential data loss, as may happen when tuning hardware for maximum performance.

## 2. We Need Feedback

If you find a typographical error in the *ESB Registry Guide*, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in JIRA: <http://jira.jboss.com/jira/> against the Documentation component of the *SOA Platform* project.

When submitting a bug report, be sure to mention the manual's identifier:

ESB\_REG

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.

# What is the Registry?

## 1. Introduction

In the context of SOA, a registry provides applications and businesses a central point to store information about their services. It is expected to provide the same level of information and the same breadth of services to its clients as that of a conventional market place. Ideally a registry should also facilitate the automated discovery and execution of e-commerce transactions and enabling a dynamic environment for business transactions. Therefore, a registry is more than an “e-business directory”. It is an inherent component of the SOA infrastructure.

### 1.1. Why do I need it ?

It is not difficult to discover, manage and interface with business partners on a small scale, using manual or ad hoc techniques. However, this approach does not scale as the number of services, the frequency of interactions, and the physical distributed nature of the environment, increase. A standards-based registry solution provides a way to publish and discover services. It offers a central place where you query whether a partner has a service that is compatible with in-house technologies or to find a list of companies that supports shipping services on the other side of the globe.

Service registries are central to most service oriented architectures and at runtime act as a contact point to correlate service requests to concrete behaviors. A service registry has meta-data entries for all artifacts within the SOA that are used at both runtime and design time. Items inside a service registry may include service description artifacts (e.g., WSDL), Service Policy descriptions, various XML schema used by services, artifacts representing different versions of services, governance and security artifacts (e.g., certificates, audit trails), etc. During the design phase, business process designers may use the registry to link together calls to several services to create a workflow or business process.



#### Note

The registry may be replicated or federated to improve performance and reliability. It need not be a single point of failure.

### 1.2. How do I use it ?

From a business analyst's perspective, it is similar to an Internet search engine for business processes. From a developers perspective, they use the registry to publish services and query the registry to discover services matching various criteria.

### 1.3. Registry Vs Repository

A registry allows for the registration of services, discovery of metadata and classification of

entities into predefined categories. Unlike a repository, it does not have the ability to store business process definitions or WSDL or any other documents that are required for trading agreements. A registry is essentially a catalogue of items, whereas a repository maintains those items.

### 1.4. SOA Components

An SOA is a specific type of distributed system in which the agents are "services"<sup>1</sup>.

The key components of a Service Oriented Architecture are the messages that are exchanged, agents that act as service requesters and service providers, and shared transport mechanisms that allow the flow of messages. A description of a service that exists within an SOA is essentially just a description of the message exchange pattern between itself and its users. Within an SOA there are thus three critical roles: requester, provider, and broker.

#### Service provider

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

#### Service broker

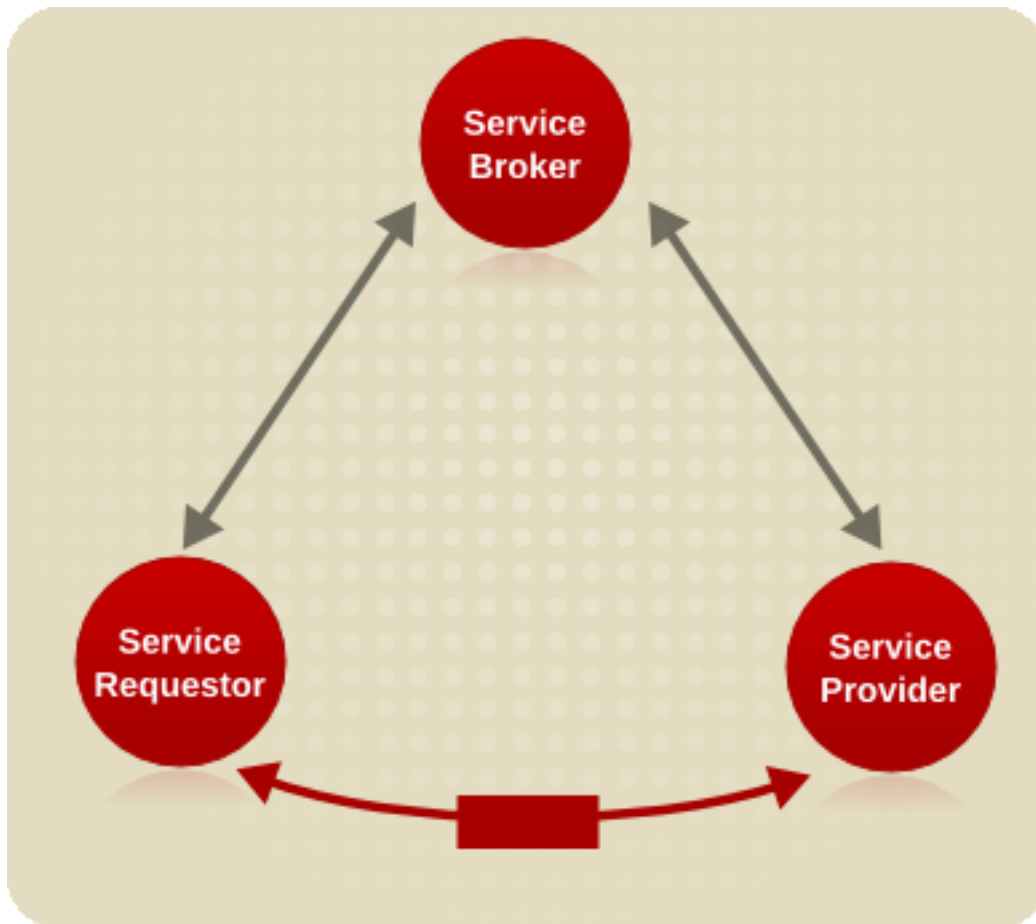
hosts a registry of service descriptions. It is responsible for linking a requestor to a service provider.

#### Service requester

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

---

<sup>1</sup> Refer to the W3C Working Draft on [Web Services Architecture](http://www.w3.org/TR/2003/WD-ws-arch-20030808/#id2617708) [http://www.w3.org/TR/2003/WD-ws-arch-20030808/#id2617708] for a more detailed definition.



## 1.5. UDDI

The Universal Distribution, Discover and Interoperability registry is a directory service for Web Services. It enables service discovery through queries to the UDDI registry at design time or at run time. It also allows providers to publish descriptions of their services to the registry. The registry typically contains a URL that locates the WSDL document for the web services and contact information for the service provider. Within UDDI information is classified into the following categories.

- *White pages*: contain general information such as the name, address and other contact information about the company providing the service.
- *Yellow pages*: categorize businesses based on the industry their services cater to.
- *Green pages*: provide information that will enable a client to bind to the service that is being provided.

## 1.6. The Registry and JBossESB

The registry plays a central role within JBossESB. It is used to store endpoint references (EPRs) for the services deployed within the ESB. It may be updated dynamically when services

first start-up, or statically by an external administrator.

As with all environments within which registries reside, it is not possible for the registry to determine the liveness of the entities its data represents, e.g., if an EPR is registered with the registry then there can be no guarantee that the EPR is valid (it may be malformed) or it may represent a service that is no longer active. At present JBossESB does not perform life-cycle monitoring of the services that are deployed within it. As such, if services fail or move elsewhere, their EPRs that may reside within the registry will remain until they are explicitly updated or removed by an administrator. Therefore, if you get warnings or errors related to EPRs obtained from the registry, you should consider removing any out-of-date items.

# Configuring the Registry

## 1. Introduction

The JBossESB Registry architecture allows for many ways to configure the ESB to use either a Registry or Repository. By default we use a JAXR implementation (Scout) and a UDDI (jUDDI), in an embedded way.

The following properties can be used to configure the JBossESB Registry. In the `jbossesb-properties.xml` there is section called 'registry':

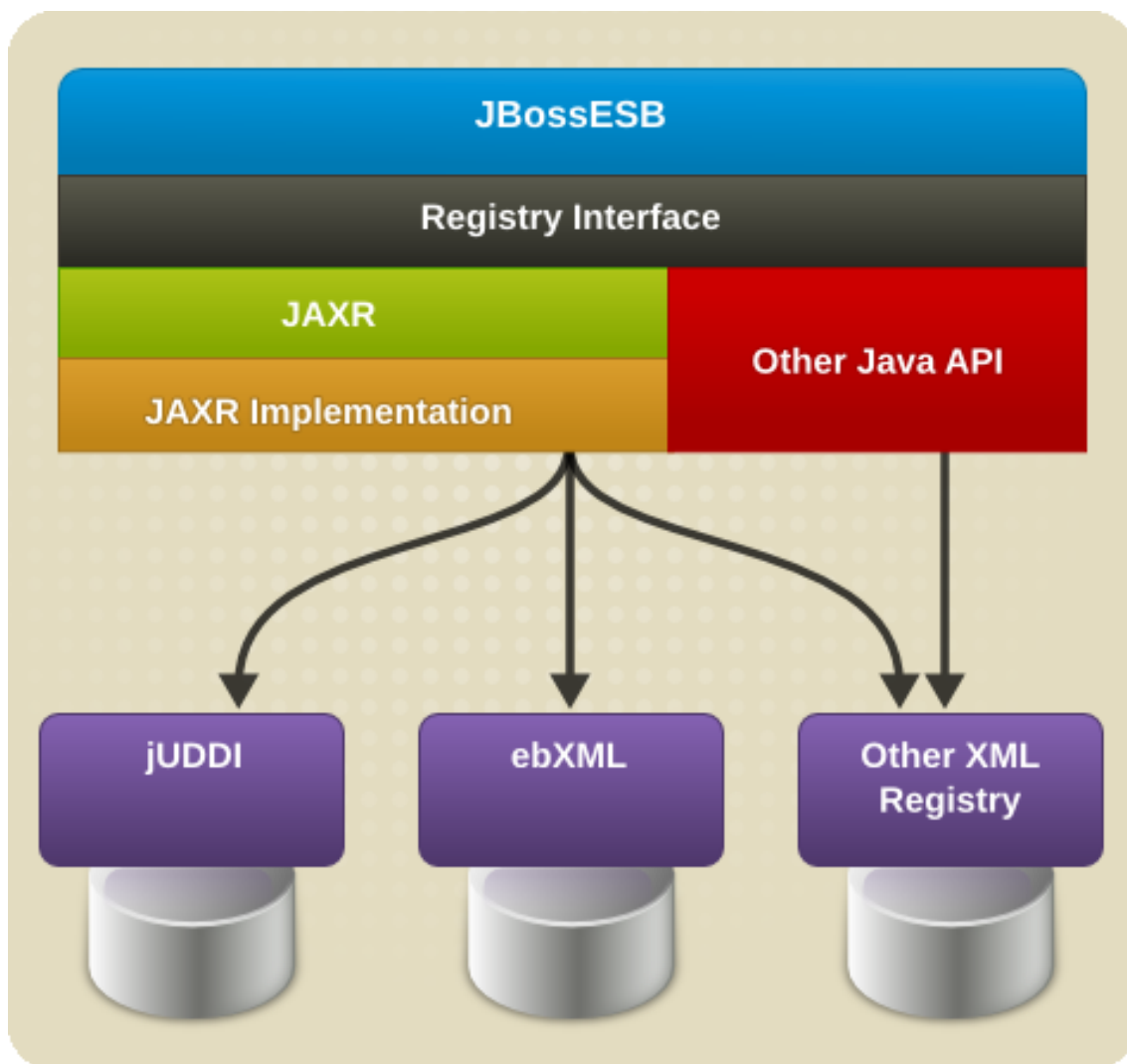
```
<properties name="registry">
  <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
  <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
  <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="org.apache.juddi.registry.local.InquiryService#inquire"/>
  <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="org.apache.juddi.registry.local.PublishService#publish"/>
  <property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
  <property name="org.jboss.soa.esb.registry.password"
value="password"/>
  <!-- the following parameter is scout specific to set the type of
communication
between scout and the UDDI (embedded, rmi, soap) -->
  <property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.LocalTransport"/>
</properties>
```

In short, the properties are:

1. `org.jboss.soa.esb.registry.implementationClass`: a class that implements the `jbossesb Registry` interface. We have provided one implementation (`JAXRRegistry` interface).
2. `org.jboss.soa.esb.registry.factoryClass`: the class name of the JAXR `ConnectionFactory` implementation.
3. `org.jboss.soa.esb.registry.queryManagerURI`: the URI used by JAXR to query.
4. `org.jboss.soa.esb.registry.lifeCycleManagerURI`: the URI used by JAXR to edit.
5. `org.jboss.soa.esb.registry.user`: the user used for edits.
6. `org.jboss.soa.esb.registry.password`: the password to go along with the user.
7. `org.jboss.soa.esb.scout.proxy.transportClass`: the name of the class used by scout to do the transport from scout to the UDDI.

## 2. The components involved

The registry can be configured in many ways. [Figure 2.1, “Blue print of the Registry component architecture”](#) shows a blue print of all the registry components. From the top down we can see that the JBossESB funnels all interaction with the registry through the Registry Interface. By default it then calls into a JAXR implementation of this interface. The JAXR API needs an implementation, which by default is Scout. The Scout JAXR implementation calls into a jUDDI registry. However there are many other configuration options.



**Figure 2.1. Blue print of the Registry component architecture**

## 3. The Registry Implementation Class

Property: `org.jboss.soa.esb.registry.implementationClass`

By default we use the JAXR API. The JAXR API is a convenient API since it allows us to

connect any kind of XML based registry or repository. However, if for example you want to use Systinet's Java API you can do that by writing your own `SystinetRegistryImplementation` class and referencing it in this property.

## 4. Using JAXR

Property: `org.jboss.soa.esb.registry.factoryClass`

If you decided to use JAXR then you will have to pick which JAXR implementation to use. This property is used to configure that class. By default we use Scout and therefore it is set to the scout factory `'org.apache.ws.scout.registry.ConnectionFactoryImpl'`. The next step is to tell the JAXR implementation the location of the registry or repository for querying and updating, which is done by setting the `org.jboss.soa.esb.registry.queryManagerURI`, and `org.jboss.soa.esb.registry.lifeCycleManagerURI` respectively, along with the username (`org.jboss.soa.esb.registry.user`) and password (`org.jboss.soa.esb.registry.password`) for the UDDI.

## 5. Using Scout and jUDDI

Property: `org.jboss.soa.esb.scout.proxy.transportClass`

When using Scout and jUDDI there is an additional parameter that one can set. This is the transport class that should be used for communication between Scout and jUDDI. Thus far there are 4 implementations of this class which are based on SOAP, SAAJ, RMI and Local (embedded java). Note that when you change the transport, you will also have to change the query and lifecycle URIs. For example:

queryManagerURI	<code>http://localhost:8080/juddi/inquiry</code>
lifeCycleManagerURI	<code>http://localhost:8080/juddi/publish</code>
transportClass	<code>org.apache.ws.scout.transport.AxisTransport</code>

**Table 2.1. SOAP Transport Classes**

queryManagerURI	<code>jnp://localhost:1099/InquiryService?org.apache.juddi.registry.rmi.InquiryService</code>
lifeCycleManagerURI	<code>jnp://localhost:1099/PublishService?org.apache.juddi.registry.rmi.PublishService</code>
transportClass	<code>org.apache.ws.scout.transport.RMITransport</code>

**Table 2.2. RMI Transport Classes**

queryManagerURI	<code>org.apache.juddi.registry.local.InquiryService#inquire</code>
lifeCycleManagerURI	<code>org.apache.juddi.registry.local.PublishService#publish</code>
transportClass	<code>org.apache.ws.scout.transport.LocalTransport</code>

### Table 2.3. Local Transport Classes

For jUDDI we have two requirements that need to be fulfilled:

1. access to the jUDDI database. You will need to create a schema in your database, and add the jbossesb publisher. The `product/install/jUDDI-registry` directory contains db create scripts for you favorite database.
2. `esb.juddi.xml`. The configuration of jUDDI itself. If you do not use a datasource you need to take special care to set the following properties:

```
<entry key="juddi.isUseDataSource">false</entry>
<entry key="juddi.jdbcDriver">com.mysql.jdbc.Driver</entry>
<entry key="juddi.jdbcUrl">jdbc:mysql://localhost/juddi</entry>
<entry key="juddi.jdbcUsername">juddi</entry>
<entry key="juddi.jdbcPassword">juddi</entry>
```

if you do use a datasource you need something like:

```
<entry key="juddi.isUseDataSource">true</entry>
<entry key="juddi.dataSource">java:comp/env/jdbc/juddiDB</entry>
```

The database can be automatically created if the user you have created has enough rights to create tables. Next make sure the `isCreateDatabase` flag is set to true, and that the `sqlFiles` parameter settings point to the database you are using. The jUDDI create scripts are located in the `juddi.jar` and jUDDI supports daffodildb, db2, derby, firebird, hsqldb, informix, jdatastore, mysql, oracle, postgresql, sybase (can be used for ms-sqlserver) and totalxml.

```
<!-- <entry key="juddi.tablePrefix">JUDDI_</entry> -->
<entry key="juddi.isCreateDatabase">true</entry>
<entry key="juddi.databaseExistsSql">select * from
    ${prefix}BUSINESS_ENTITY</entry>
<entry key="juddi.sqlFiles">juddi-sql/mysql/create_database.sql,
    juddi sql/mysql/insert_publishers.sql</entry>
```

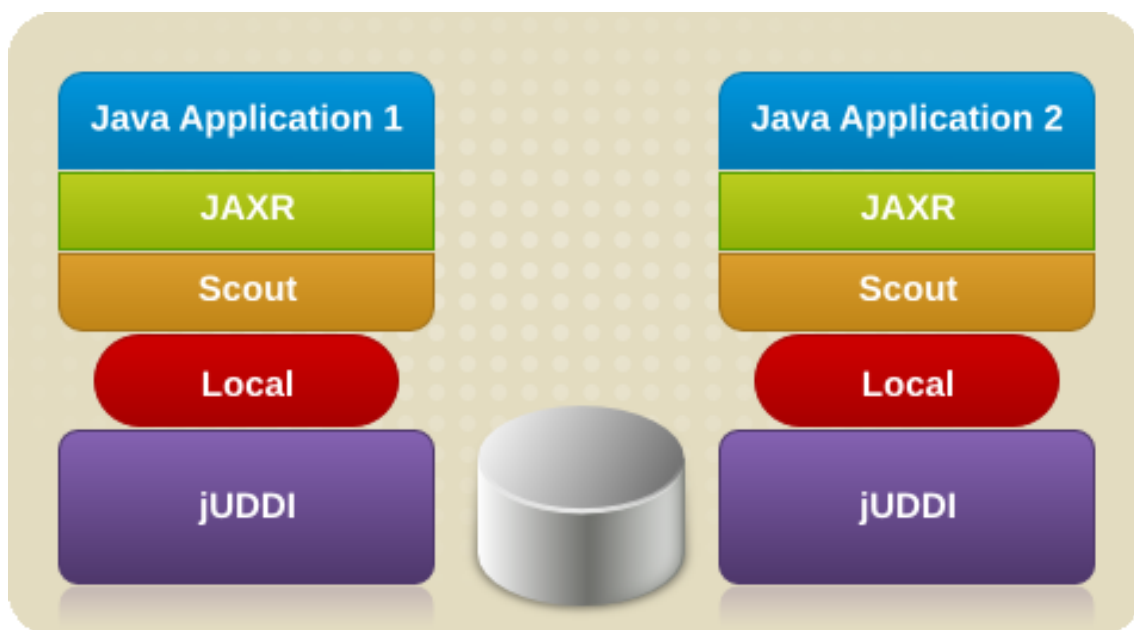
# Configuration Examples

## 1. Introduction

As mentioned before, by default the JBossESB is configured to use the JAXR API using Scout as its implementation and jUDDI as the registry. Here are some examples of how you can deploy this combo.

## 2. Embedded

All ESB components (with components we really mean JVMs in this case) can embed the registry and they all can connect to the same database (or different once if that makes sense).



Embedded jUDDI

Properties example

```
<properties name="registry">
  <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
  <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
  <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="org.apache.juddi.registry.local.InquiryService#inquire"/>
  <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="org.apache.juddi.registry.local.PublishService#publish"/>
  <property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
  <property name="org.jboss.soa.esb.registry.password"
value="password"/>
  <!-- the following parameter is scout specific to set the type of
```

```
communication between scout and the UDDI (embedded, rmi, soap) -->
  <property name="org.jboss.soa.esb.scout.proxy.transportClass"
    value="org.apache.ws.scout.transport.LocalTransport"/>
</properties>
```

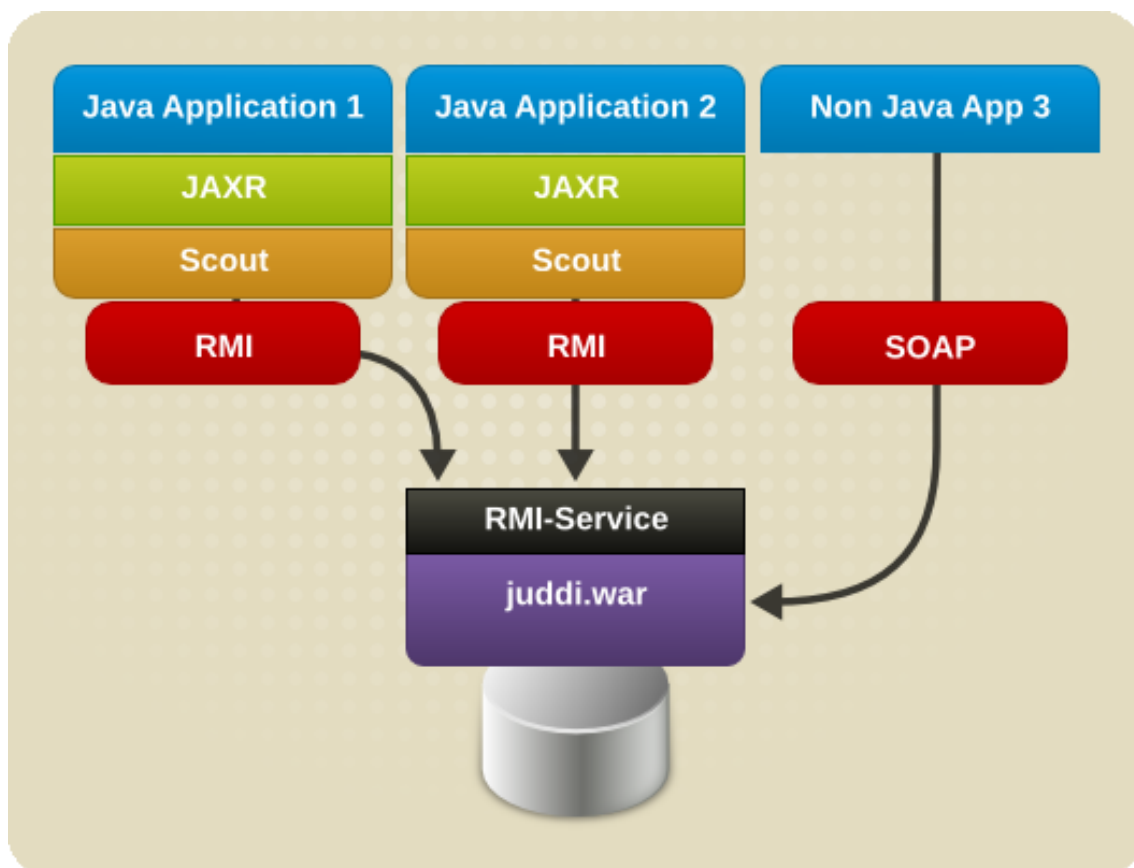
### 3. RMI using the juddi.war or jbossesb.sar

Deploy a version of the jUDDI that brings up an RMI service. The JBossESB comes with a `juddi.war` in the `product/install/jUDDI-registry` directory. This war brings up the regular webservises but also an RMI service. Along with the `juddi.war` you need to deploy a datasource which points to your jUDDI database. An example file is supplied for MySQL.



#### Note

The `jbossesb.sar` also registers a RMI service. So you would only need to deploy the `juddi.war` if you need webservice access.



RMI using the juddi.war

Properties example:

```
<properties name="registry">
  <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
  <property name="org.jboss.soa.esb.registry.factoryClass"
  value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
  <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="jnp://localhost:1099/InquiryService?org.apache.juddi.registry.rmi.Inquiry#inquire"/>
  <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="jnp://localhost:1099/PublishService?org.apache.juddi.registry.rmi.Publish#publish"/>
  <property name="org.jboss.soa.esb.registry.user"
  value="jbossesb"/>
  <property name="org.jboss.soa.esb.registry.password"
  value="password"/>
  <!-- the following parameter is scout specific to set the type of
communication between scout and the
      UDDI (embedded, rmi, soap) -->
  <property name="org.jboss.soa.esb.scout.proxy.transportClass"
  value="org.apache.ws.scout.transport.RMITransport"/>
</properties>
```

The `juddi.war` is configured to bring up a RMI Service, which is triggered by the following setting in the `web.xml`

```
<!-- uncomment if you want to enable making calls in juddi with rmi -->
<servlet>
  <servlet-name>RegisterServicesWithJNDI</servlet-name>
  <servlet-class>org.apache.juddi.registry.rmi.RegistrationService</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Make sure to include (for example) the following JNDI settings in your `juddi.properties`:

```
# JNDI settings (used by RMITransport)
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
java.naming.factory.url.pkgs=org.jboss.naming
```



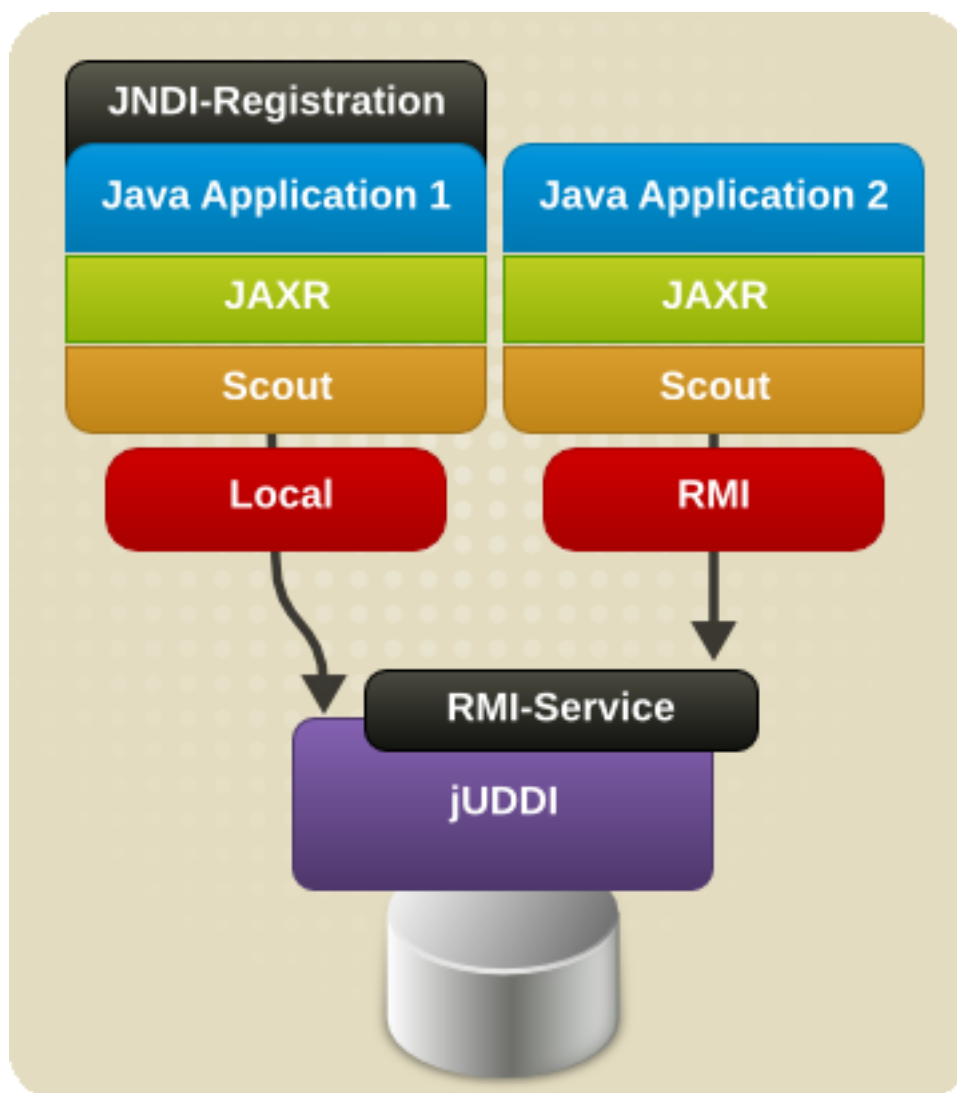
### Note

The RMI clients need to have the `scout-client.jar` in their classpath.

## 4. RMI using your own JNDI Registration of the RMI Service

If you don't want to deploy the `juddi.war` you can setup one of the ESB components that runs in the the same JVM as jUDDI to register the RMI service. While the other applications need to

be configured to use RMI.



RMI using your own JNDI registration

Properties example: For application 1 you need the Local settings:

```
<properties name="registry">
  <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
  <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
  <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="org.apache.juddi.registry.local.InquiryService#inquire"/>
  <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="org.apache.juddi.registry.local.PublishService#publish"/>
  <property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
  <property name="org.jboss.soa.esb.registry.password"
value="password"/>
  <!-- the following parameter is scout specific to set the type of
communication
```

```

        between scout and the UDDI (embedded, rmi, soap) -->
        <property name="org.jboss.soa.esb.scout.proxy.transportClass"
            value="org.apache.ws.scout.transport.LocalTransport"/>
    </properties>

```

while for application2 you need the RMI settings:

```

<properties name="registry">
    <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
    <property name="org.jboss.soa.esb.registry.factoryClass"
        value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
    <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="jnp://localhost:1099/InquiryService?org.apache.juddi.registry.rmi.Inquiry#inquire"/>
    <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="jnp://localhost:1099/PublishService?org.apache.juddi.registry.rmi.Publish#publish"/>
    <property name="org.jboss.soa.esb.registry.user"
        value="jbossesb"/>
    <property name="org.jboss.soa.esb.registry.password"
        value="password"/>
    <!-- the following parameter is scout specific to set the type of
communication between scout
        and the UDDI (embedded, rmi, soap) -->
    <property name="org.jboss.soa.esb.scout.proxy.transportClass"
        value="org.apache.ws.scout.transport.RMITransport"/>

```

Where the hostname of the `queryManagerURI` and `lifeCycleManagerURI` need to point to the hostname on which jUDDI is running (which would be where application1 is running). Obviously application1 needs to have access to a naming service. To do the registration process you need to do something like:

```

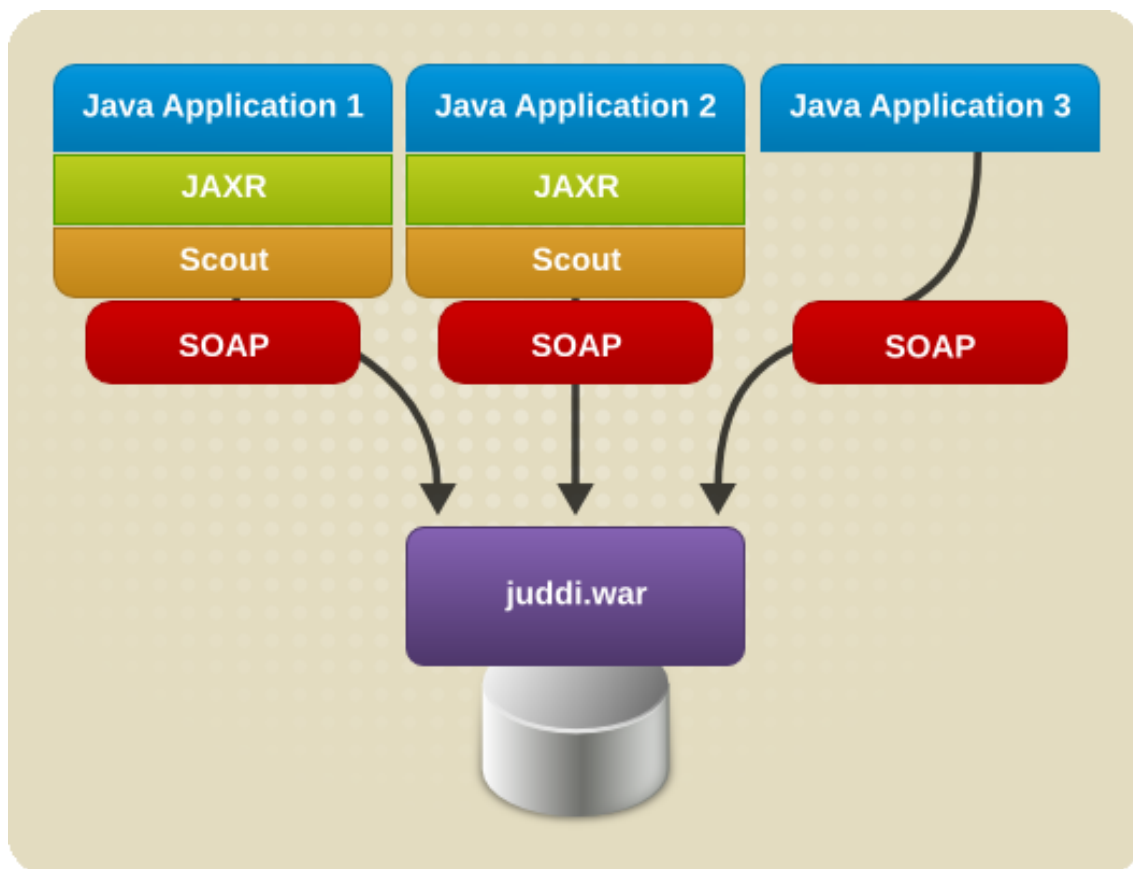
//Getting the JNDI setting from the config
String factoryInitial = Config.getStringProperty(
Properties env = new Properties();
env.setProperty(RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_INITIAL,factoryInitial);
env.setProperty(RegistryEngine.PROPNAME_JAVA_NAMING_PROVIDER_URL,
providerURL);
env.setProperty(RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_URL_PKGS,
factoryURLPkgs);
log.info("Creating Initial Context using: \n"
+ RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_INITIAL + "=" + factoryInitial
+ "\n"
+ RegistryEngine.PROPNAME_JAVA_NAMING_PROVIDER_URL      + "=" + providerURL +
"\n"
+ RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_URL_PKGS + "=" +
factoryURLPkgs + "\n");
InitialContext context = new InitialContext(env);
Inquiry inquiry = new InquiryService();
log.info("Setting " + INQUIRY_SERVICE + ", " +
inquiry.getClass().getName());
mInquiry = inquiry;
context.bind(INQUIRY_SERVICE, inquiry);
Publish publish = new PublishService();

```

```
log.info("Setting " + PUBLISH_SERVICE + ", " +
publish.getClass().getName());
mPublish = publish;
context.bind(PUBLISH_SERVICE, publish);
```

## 5. SOAP

Finally, you can make the communication between Scout and jUDDI SOAP based. Again you need to deploy the `juddi.war` and configure the datasource. You probably want to shutdown the RMI service by commenting out the `RegisterServicesWithJNDI` servlet in the `web.xml`.



SOAP

Properties example:

```
<properties name="registry">
  <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
  <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
  <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="http://localhost:8080/juddi/inquiry"/>
  <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
```

```
value="http://localhost:8080/juddi/publish"/>
<property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
<property name="org.jboss.soa.esb.registry.password"
value="password"/>
<!-- the following parameter is scout specific to set the type of
communication between scout
and the UDDI (embedded, rmi, soap) -->
<property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.AxisTransport"/>
```



### Note

JBossAS 4.2 ships with older versions of Scout and jUDDI. It is recommended to remove the `juddi.sar` to prevent versioning issues



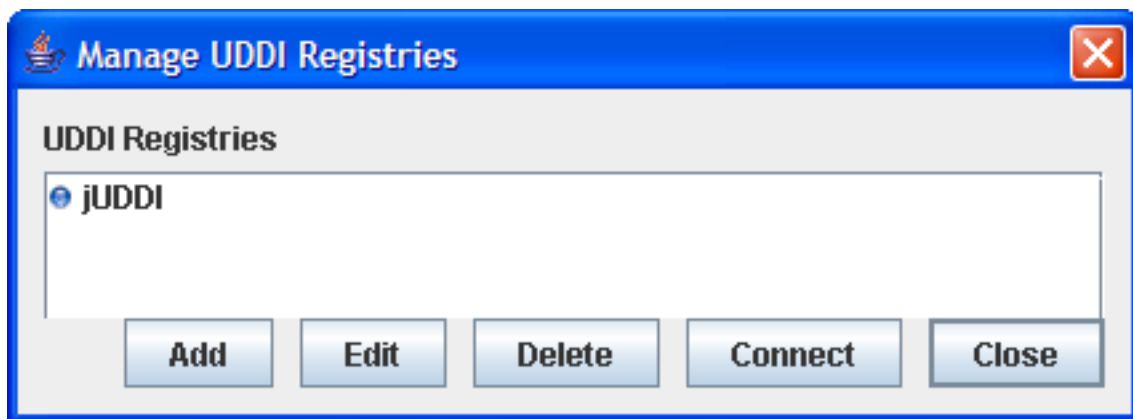
# UDDI Browser

## 1. Introduction

We are looking for a good web based console for jUDDI. In the meantime you can browse the jUDDI registry using the uddibrowser. The uddibrowser 'ub' can be downloaded from <http://www.uddibrowser.org>. Before configuring the ub make sure the `juddi.war` is deployed to the `jbossesb.sar`, to enable WS communication to jUDDI.

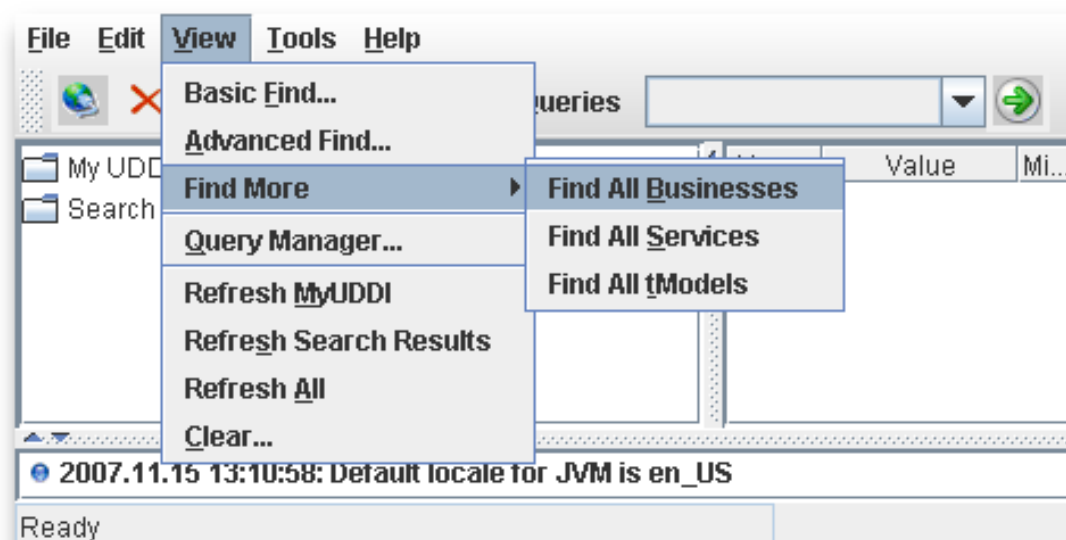
## 2. UB setup

The ub is a standalone Java application. Start the ub and select **Edit > UDDI Registries**, and add an entry called jUDDI



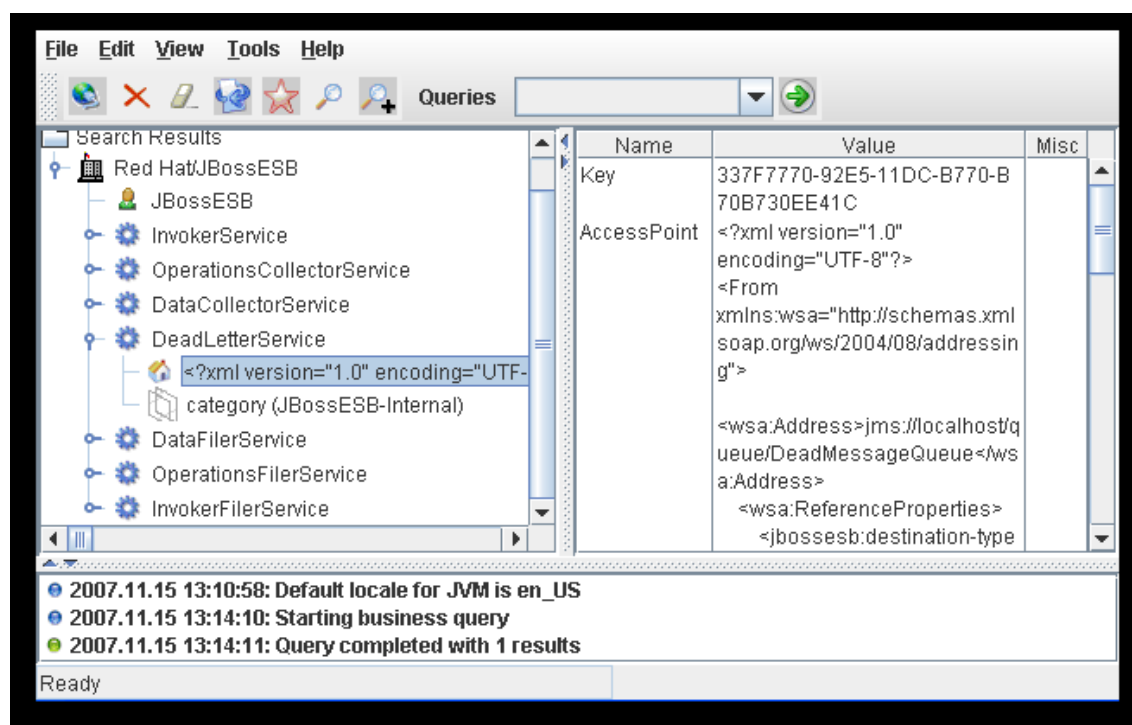
**Figure 4.1. Add a connection**

Click on 'connect' and select **View > Find More > Find All Businesses**



**Figure 4.2. View all Businesses**

and in the left pane you should see the Red Hat/JBossESB organization. You can navigate into the individual services and their ServiceBindings.



**Figure 4.3. View Services and ServiceBindings**

Each ServiceBinding contains an EPR in its AccessPoint.

Some features of the uddibrowser may not work, but it should give enough functionality to maintain jUDDI. As mentioned before we are looking for a good web based console for jUDDI.



# Troubleshooting

## 1. Scout and jUDDI pitfalls

- Make sure to put our version of the `jaxr-api-1.0.jar`, `scout-0.7rc2-embedded.jar` and the `juddi-embedded.jar` first. Other versions of these libraries are present in the JBossAS libraries and they are, for the time being, incompatible. This should get resolved in future release of the Application Server.
- If you use RMI you need the `juddi-client.jar`.
- Make sure the `jbossesb-properties.xml` file is on the classpath and read or else the registry will try to instantiate classes with the name of 'null'.
- Make sure you have a `juddi.properties` file on your classpath for jUDDI to configure itself. JBossESB uses `esb.juddi.xml`, but generates the `juddi.properties` file for jUDDI to read.
- Make sure to read the `README` in the `product/install/jUDDI-registry` directory, to prepopulate your own jUDDI database.
- In the event that a service fails or does not shut down cleanly, it is possible that stale entries may persist within a registry. You will have to remove these manually.

## 2. More Information

- For more information see the wiki <http://labs.jboss.com/wiki/JudyEvaluation>
- JBossESB user forum: <http://www.jboss.com/index.html?module=bb&op=viewforum&f=246>
- When you deploy the `juddi.war` that ship with our ESB, you enable access through SOAP, which means that you can use any UDDI browser. You can use <http://uddibrowser.org>

