Apache jUDDI Guide

Kurt T Stam, Red Hat, Inc.
Alex O'Ree, Apache Software Foundation (ASF), http://juddi.apache.org

Apache jUDDI Guide

by Kurt T Stam and Alex O'Ree

Dedication

We’d like to dedicate this guide to Steve Viens and Andy Cutright who started this project back in 2003.

Preface

The jUDDI project maintains a UDDIv3 registry that can be deployed to most modern JEE application servers. The jUDDI project is part of the Apache Software Foundation and encourages participation. It is easy to participate and if you discover a simple typo or would like to contribute to this guide in general please read the README page (add link).

Chapter 1. Universal Description, Discovery and Integration (UDDI)

1.1. UDDI Protocol, Specification

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 (or Intranet). 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-oasis-open-org]. UDDI has gone through 3 revisions and the latest version is 3.0.2 [uddi-v3]. Additional information regarding UDDI can be found at http://uddi.xml.org [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 (or Intranet) 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.

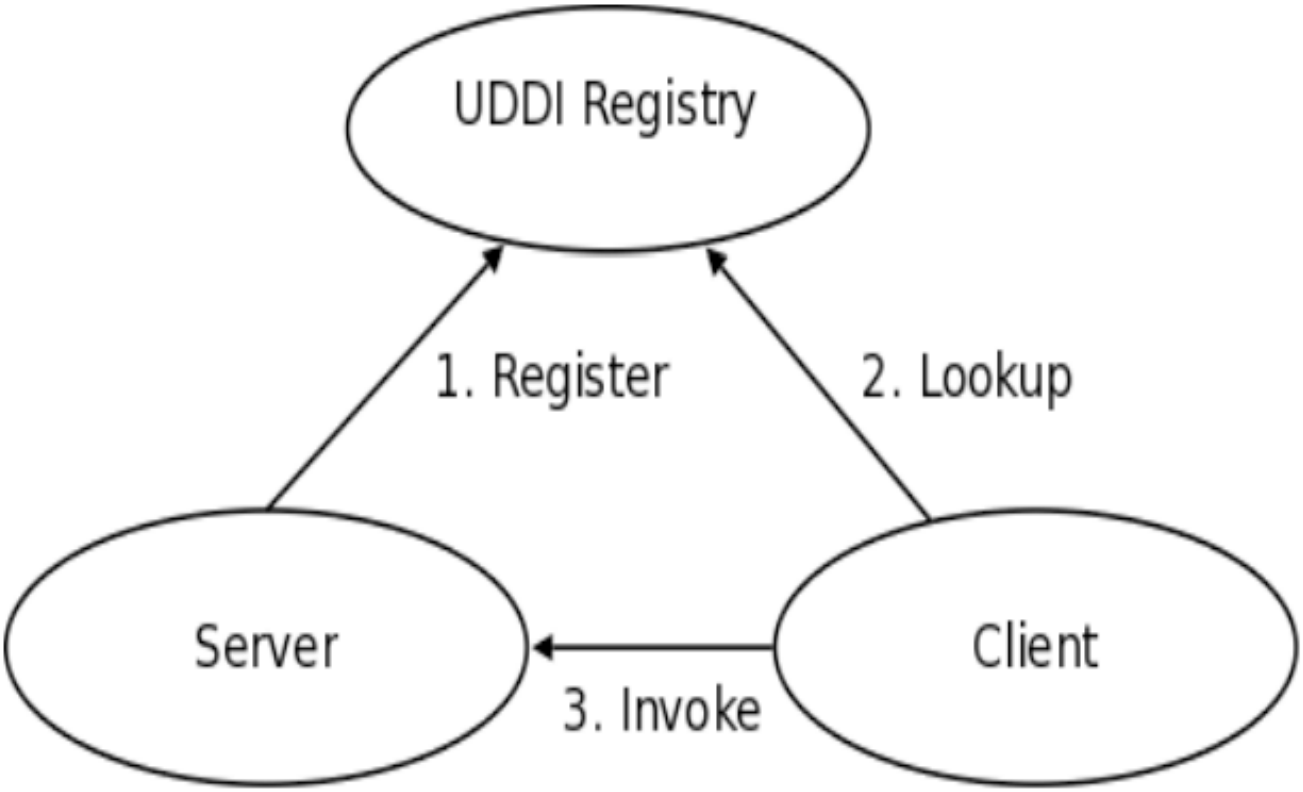


Figure 1.1. Invocation Pattern using the UDDI Registry

In step 1 of Figure 1.1, “Invocation Pattern using the UDDI Registry” it is shown how a business publishes services to the UDDI registry. In step 2, a client looks up the service in the registry and receives service binding information. Finally in step 3, the client then uses the binding information to invoke the service. The UDDI APIs are SOAP based for interoperability reasons. In this example we’ve three APIs specified in the UDDI v3 specification, Security, Publication and Inquiry. 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.

1.3. jUDDI Project

Apache jUDDI is server and client-side implementation of the UDDI v3 specification. The server side is the UDDI Registry, the client side are the juddi-client libraries. There is a Java as well as a C# version of the client libraries. The jUDDI GUI uses the client libraries to connect to a UDDI Registry. For more details please see the Chapter 2, *Getting Started*.

The following is a list of all supported UDDI interfaces provided by this release of jUDDI

Table 1.1. Supported UDDI Interfaces

API	Spec	Supported	Notes
Inquiry [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908076]	Required	All Methods	
Inquiry HTTP GET [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908158]	Optional	All Methods	Plus a number of additional methods
Publication [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908095]	Required	All Methods	
Security [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908115]	Optional	All Methods	Pluggable authentication
Subscription [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908128]	Optional	All Methods	HTTP, SMTP delivery implemented, pluggable
Subscription Listener [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908336]	Optional	All Methods	Client and Server side implementations

API	Spec	Supported	Notes
Value Set Caching [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908141]	Optional	Partial	Scheduled for 3.3
Value Set Validation [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908141]	Optional	Implemented	Scheduled for 3.3
Replication [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908180]	Optional	Partial	Scheduled for 3.3
Custody and Ownership Transfer [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908118]	Optional	All Methods	Only supports user to user transfers on the same node
UDDIv2 Inquiry [http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm#_Toc25137711]	Required	BETA	Supported via API translator
UDDIv2 Publish [http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm#_Toc25137722]	Required	BETA	Supported via API translator

The following is a list of other features of interest that was either defined in the UDDI specifications or in technical notes.

Table 1.2. jUDDI Features

API	Spec	Supported	Notes
Digital Signatures	Server req	Full support	Java and .NET clients and in browser signing
Client side Subscription Listener	Optional	Full support	Java and .NET clients
WSDL to UDDI [https://www.oasis-open.org/committees/uddi-spec/doc/tn/	Recommendation	Full support	Java, .NET clients and web GUI

API	Spec	Supported	Notes
uddi-spec-tc-tn-wsdl-v2.htm]			
WADL to UDDI [https://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm]	Recommendation	Full support	Java, .NET clients and web GUI
BPEL to UDDI [https://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-bpel-20040725.htm]	Recommendation	Full support	Java client
UDDI Technical Compliance Kit	-	Full support	Provides a standalone UDDI testing capability
Internationalization	Recommendation	Yes	Both end user interfaces (User and Admin web apps) are supported. Error messages from the server are external and can be overwritten.
Registration via Annotations	-	Full support	Provides automated registration of classes via Java/.NET Annotations

UDDI defines a number of sorting mechanisms [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908080].

Table 1.3. Supported Sort Orders

Find Qualifier	Spec	Supported	Notes
binarySort [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#sortOrd]	Required	yes	
caseInsensitiveSort [http://uddi.org/	Required	party	Only when using caseInsentitiveMatch, JIRA opened [https://

Find Qualifier	Spec	Supported	Notes
pubs/uddi-v3.0.2-20041019.htm#caseInsensSort]			issues.apache.org/jira/browse/JUDDI-785]
caseSensitiveSort [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908355]	Required	yes	
sortByNameAsc [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908356]	Required	yes	
sortByNameDesc [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908357]	Required	yes	
sortByDateAsc [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908358]	Required	yes	
sortByDateDesc [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908359]	Required	yes	
JIS-X4061 [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc42047570]	Optional	no	Japanese Character Strings

UDDI also defines a number of Find Qualifiers [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908080], which modify the default search behavior of the Inquiry [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908076] Find* APIs.

Table 1.4. Supported Find Qualifiers

Find Qualifier	Spec	Supported
andAllKeys [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908360]	Required	yes
approximateMatch [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908346]	Required	yes

Find Qualifier	Spec	Supported
bindingSubset [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908365]	Required	yes
caseInsensitiveMatch [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908348]	Required	yes
caseSensitiveMatch [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908349]	Required	yes
combineCategoryBags [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908363]	Required	yes
diacriticInsensitiveMatch [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908350]	Optional	yes
diacriticSensitiveMatch [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908351]	Required	yes
exactMatch [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908347]	Required	yes
signaturePresent [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908367]	Required	yes
orAllKeys [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908361]	Required	yes
orLikeKeys [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908362]	Required	yes
serviceSubset [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908364]	Required	yes
suppressProjectedServices [http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908366]	Required	yes

Chapter 2. Getting Started

The jUDDI project is an open source implementation of the UDDI specification. The registry implementation is a WebArchive (war) *juddiv3.war* which is deployable to any JEE container. The application exposes a WebService API which can be accessed using any generic SOAP client, the *juddi-gui* or, if you are looking to integrate the UDDI api in your application, the Java or .NET version of the *juddi-client*.

2.1. Prerequisites

jUDDI is written in Java and minimally requires

- JDK1.6+, although jUDDI should run on JDK1.6, please use the latest JDK if possible

optionally

- Maven 3.0.3+ if you want to run the examples
- A Relation Database, to replace Derby

The versions mentioned above are minimal versions and it is recommended to use the latest version available. By default jUDDI ships and uses a *Derby* database. After evaluation you probably want to move to a more full featured database.

2.2. What should I Download?

At the jUDDI download page http://juddi.apache.org/releases.html, you have the chioce of two distributions; the *juddi-client* distro or the juddi-distro, where the latter includes both client and server. Each distribution contains signed binaries, source, examples and documentation. It you are not sure which distribution to download, then take the *juddi-distro* since it contains everything which is by far the easiest way to get going.

2.3. Running jUDDI

After downloading and unpacking of the *juddi-distro*, you can start the preconfigured tomcat server by going into the *juddi-distro*-<*version*> directory and running startup

```
$ cd apache-tomcat-<version>/bin
$ ./startup.sh
```

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”, the jUDDI Welcome Page.

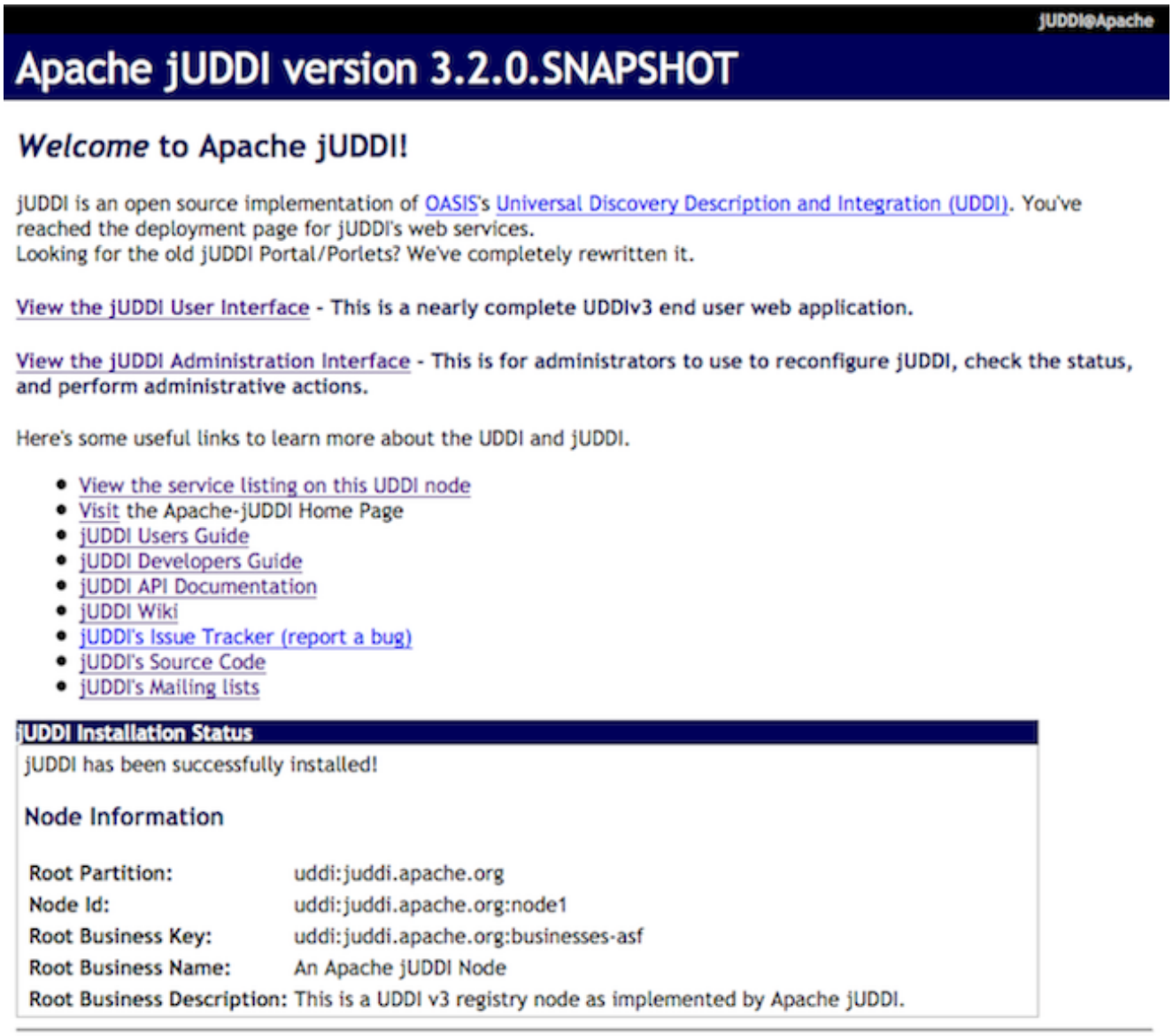


Figure 2.1. jUDDI welcome page

Before continuing please check the jUDDI instalation Status on this page and make sure it says: "jUDDI has been successfully installed!". If the page won't load or the status is anything else please check the *apache-tomcat-x.x.x/logs/juddi.log* and if you need help you can contact us via the jUDDI user mailing list. Also note that it created a *root* partition, using seed data. You can modify or add to the seed, for that see Chapter 7, *UDDI Seed Data*.

2.4. Using the jUDDI Administrative Interface

The juddi admin console runs at http://localhost:8080/juddiv3/admin and requires a login with the role of uddiadmin via the basic authentication popup dialog box. Check the *apache-tomcat-x.x.x/conf/tomcat-users.conf* file for the password of the *uddiadmin* user. Please change the password before going live.

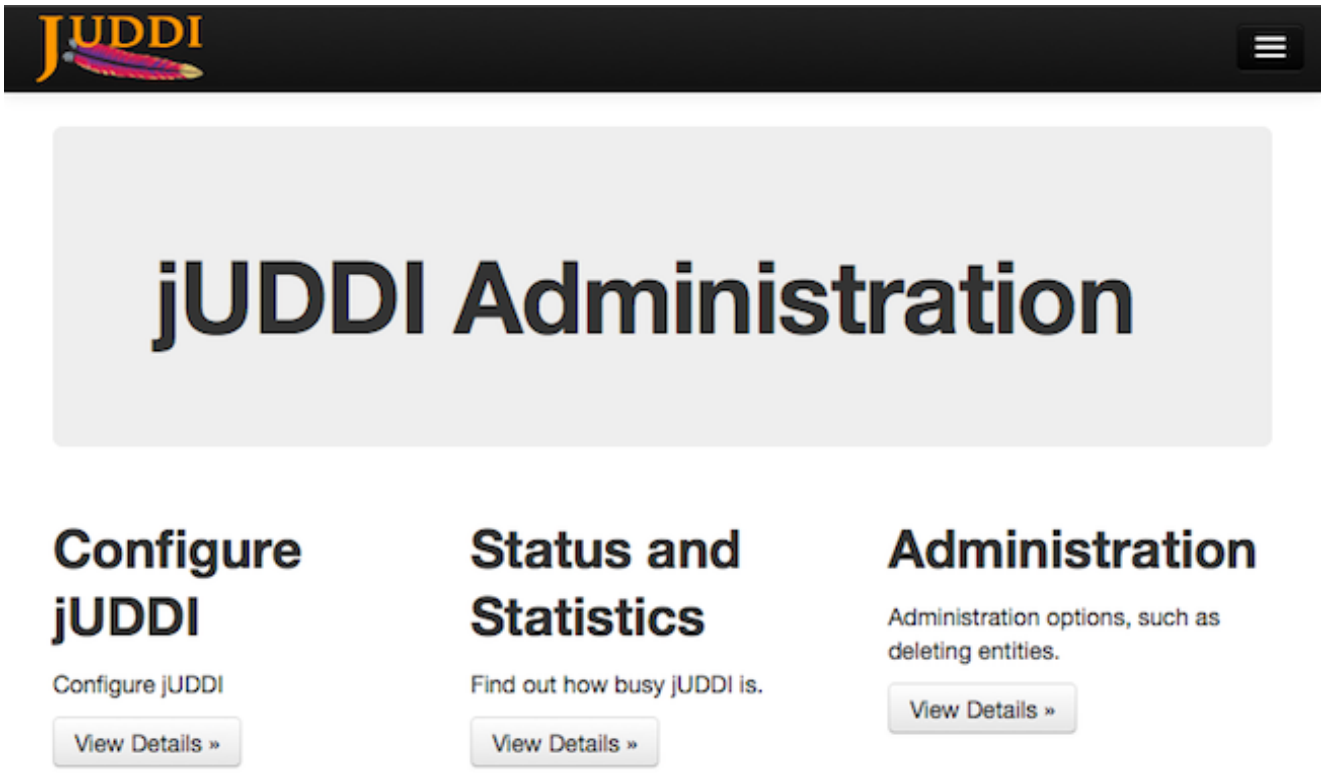


Figure 2.2. jUDDI admin

By popular demand we brought back the happy jUDDI!' page. Just click on *Status and Statistics* page. By default we run on CXF, so it is normal if says the AxisServlet is not found. There should be no other red on this page.

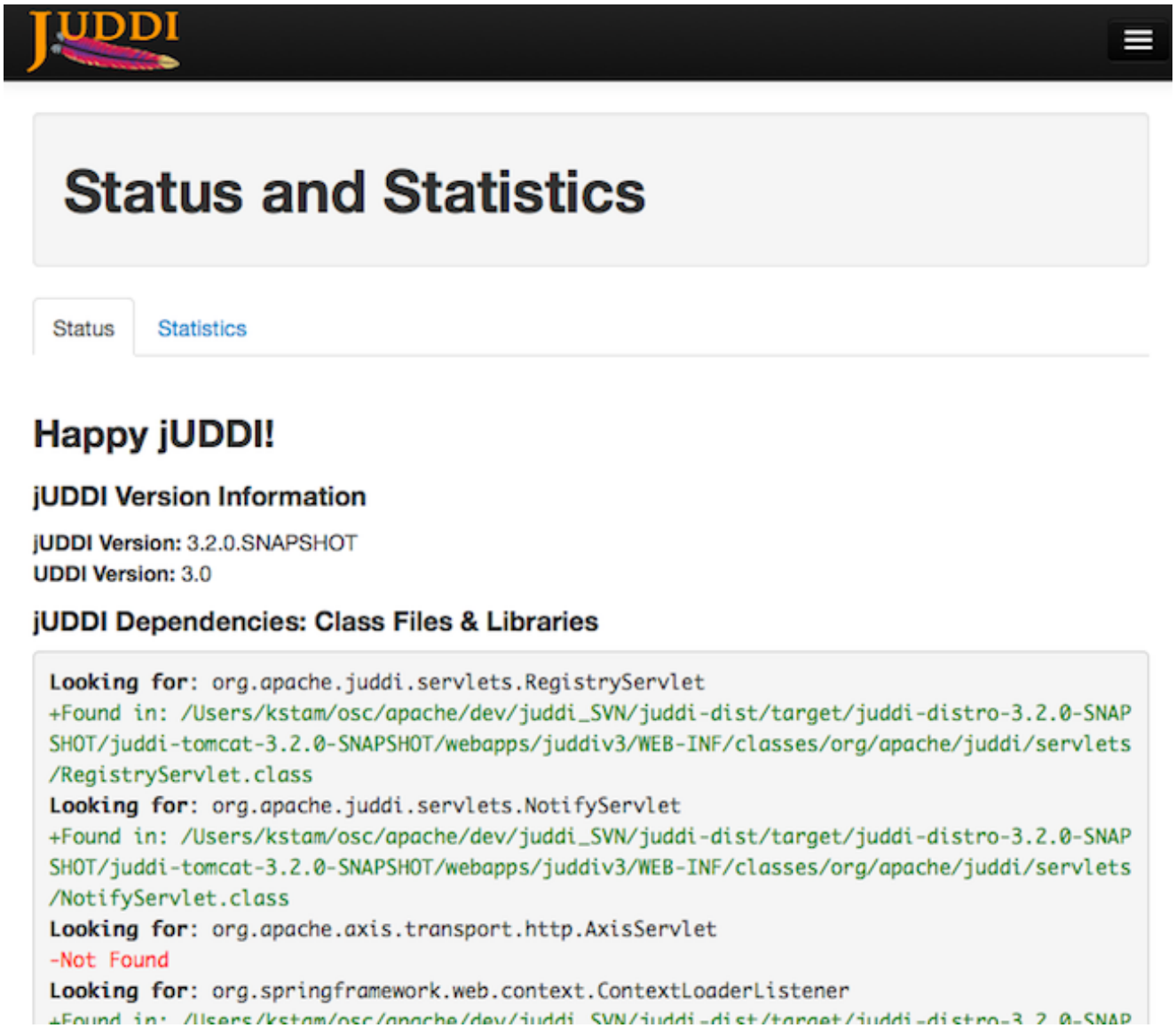


Figure 2.3. Happy jUDDI.

By default jUDDI ships with 2 publishers: *root* and *uddi*. Root is the owner of the repository, while the *uddi* user is the owner of all the default tmodels and categorizations. Please use the *root* user to log into the form login in the admin console.

Important

Please use the *root* user to log into the form login in the admin console.

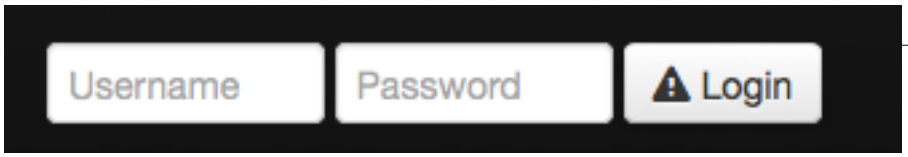


Figure 2.4. Form login

You will now be able to do more then simple browsing. Navigate to the Administration and select *save_publisher* from the dropdown. This will allow you to add your own publisher.

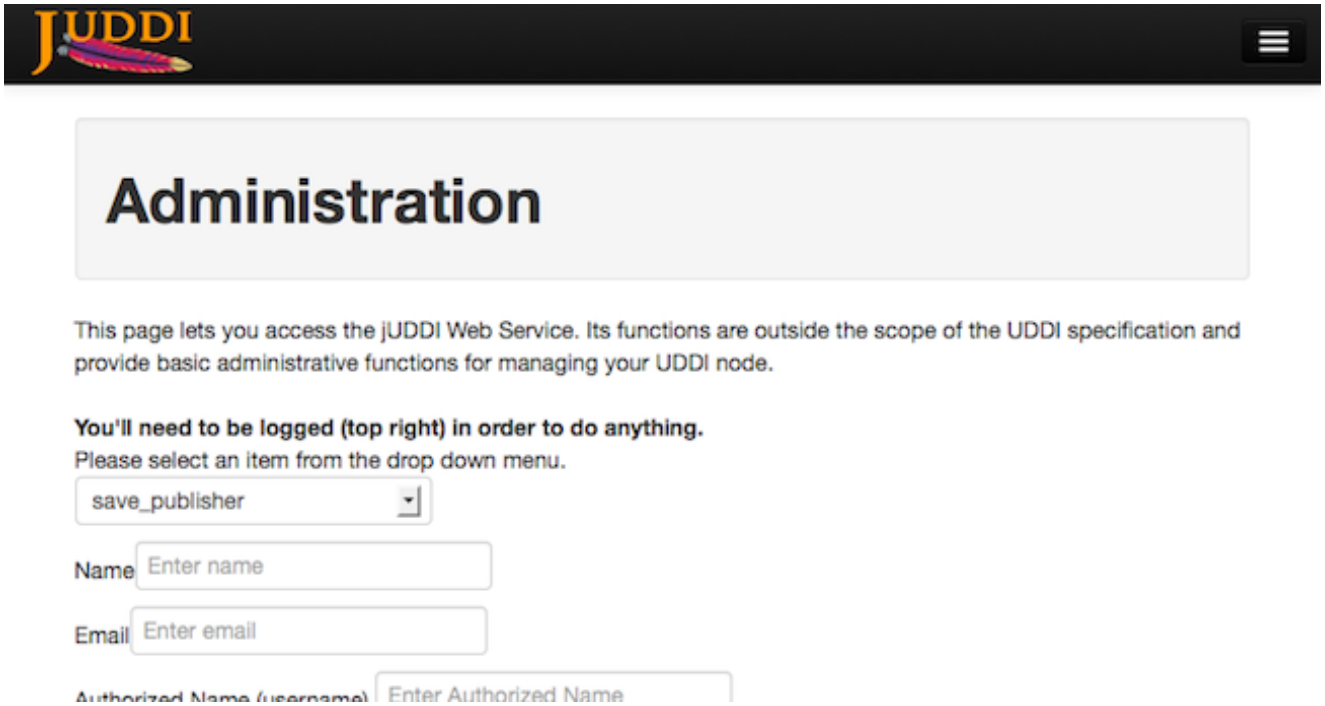
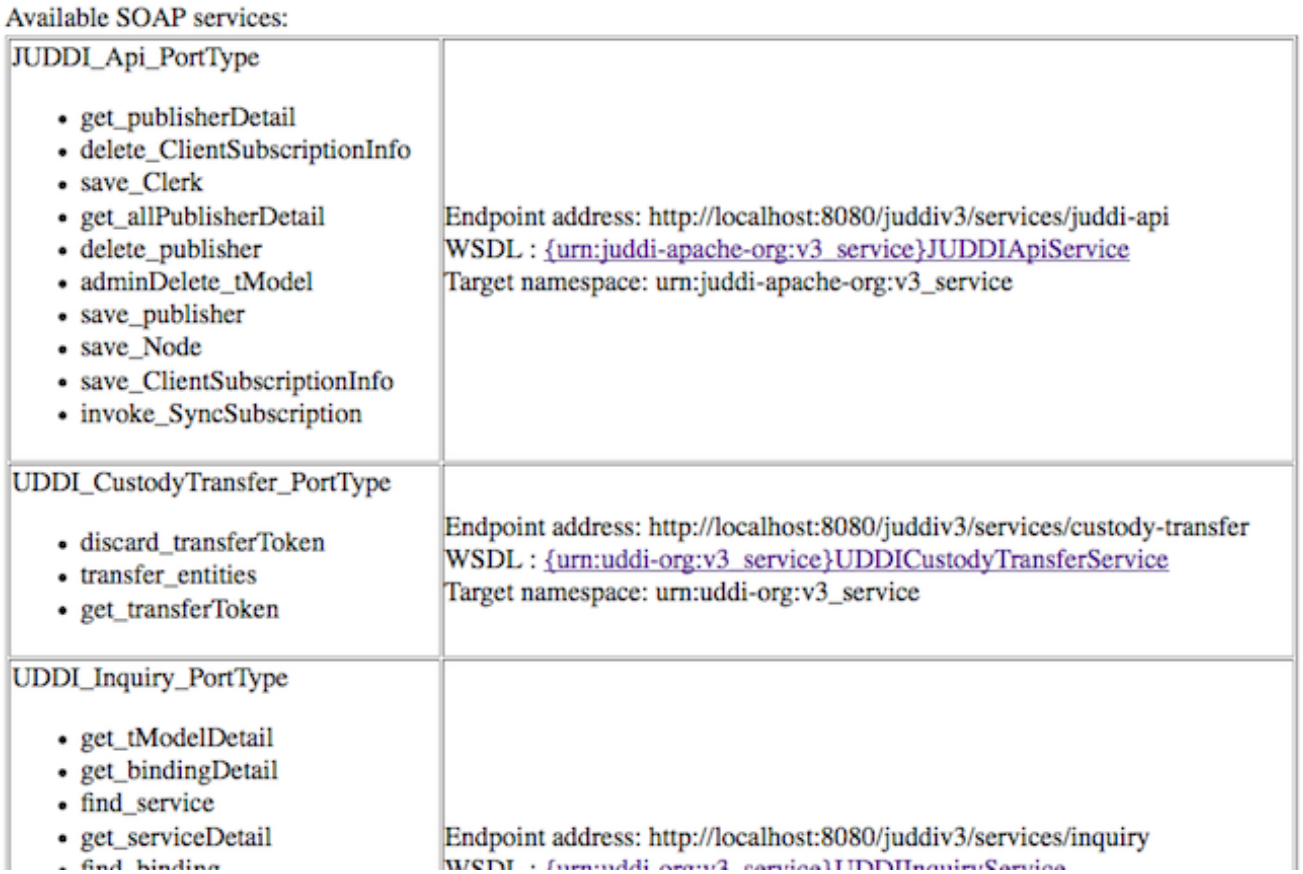


Figure 2.5. Add Publisher

2.5. Using jUDDI Web Services

OK now that we have verified that jUDDI is good to go we can inspect the UDDI WebService API by browsing to http://localhost:8080/juddiv3/services

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


Available SOAP services:

JUDDI_Api_PortType • get_publisherDetail • delete_ClientSubscriptionInfo • save_Clerk • get_allPublisherDetail • delete_publisher • adminDelete_tModel • save_publisher • save_Node • save_ClientSubscriptionInfo • invoke_SyncSubscription	Endpoint address: http://localhost:8080/juddiv3/services/juddi-api WSDL : {urn:juddi-apache-org:v3_service}JUDDIApiService Target namespace: urn:juddi-apache-org:v3_service
UDDI_CustodyTransfer_PortType • discard_transferToken • transfer_entities • get_transferToken	Endpoint address: http://localhost:8080/juddiv3/services/custody-transfer WSDL : {urn:uddi-org:v3_service}UDDICustodyTransferService Target namespace: urn:uddi-org:v3_service
UDDI_Inquiry_PortType • get_tModelDetail • get_bindingDetail • find_service • get_serviceDetail • find_binding	Endpoint address: http://localhost:8080/juddiv3/services/inquiry WSDL : {urn:uddi-org:v3_service}UDDIInquiryService

Figure 2.6. jUDDI Services

The services page shows you the available endpoints and methods available. Using any SOAP client, you should be able to import the wsdls into a tool like SoapUI as shown in Figure 2.7, “Getting an authToken using SoapUI” and send some sample requests to jUDDI to test:

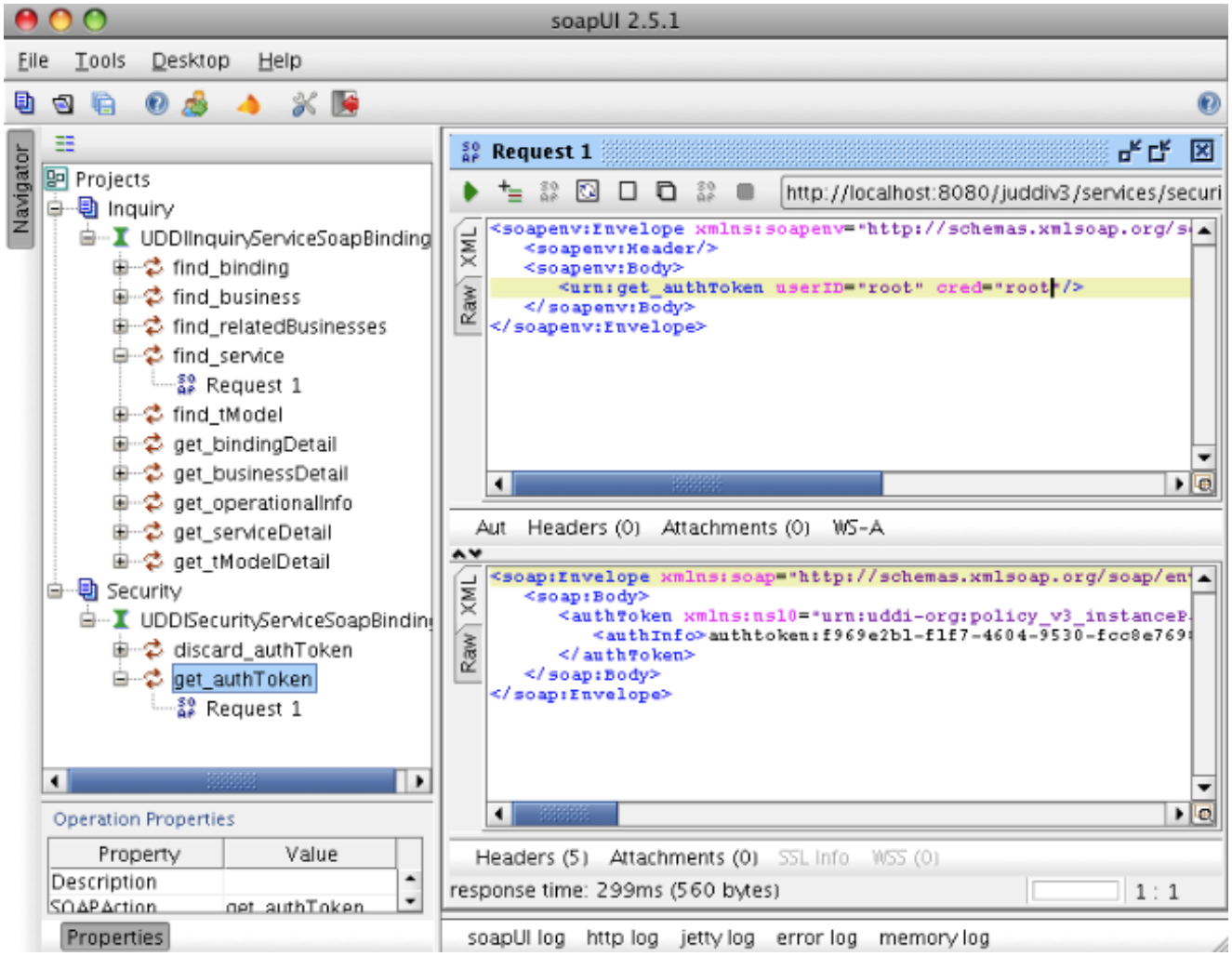


Figure 2.7. Getting an authToken using SoapUI

Tip

Try obtaining an authToken for the publisher you created earlier.

2.6. Using jUDDI GUI to create your keygenerator and business

Navigate to http://localhost:8080/juddi-gui/ to get to the jUDDI-GUI. Please use the Form Login and use the credentials of the publisher you created above. You can browse around, but really the first thing that needs to be done is to create a Key Generator or Partition at http://localhost:8080/juddi-gui/tmodelPartitions.jsp. A Key Generator is needed to save human readable, universally unique UDDIv3 keys. Please read more about UDDI v3 formatted keys, but the short story is that UDDI v3 keys are formatted like: *uddi:<domain>:name*. For example, if you wanted a tModel defined

as "uddi:www.mycompany.com:serviceauthenticationmethod", you would first have to create a tModel key generator with value "uddi:www.mycompany.com:keygenerator".

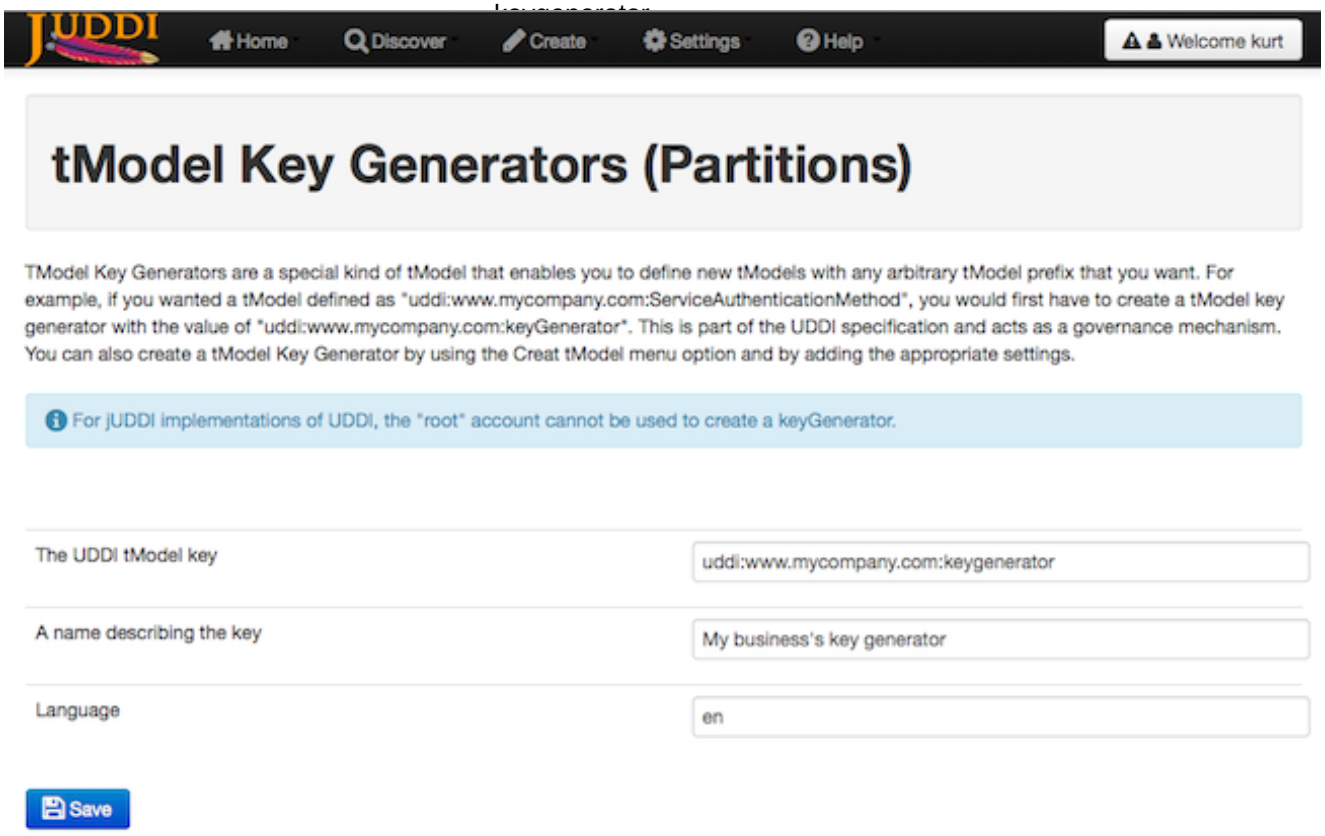


Figure 2.8. Create Key Generator

Next create your business using the key generator format you just registered. For example in Figure 2.9, “Create Business” we use a businessKey of *uddi:www.mycompany.com:mybusiness*.

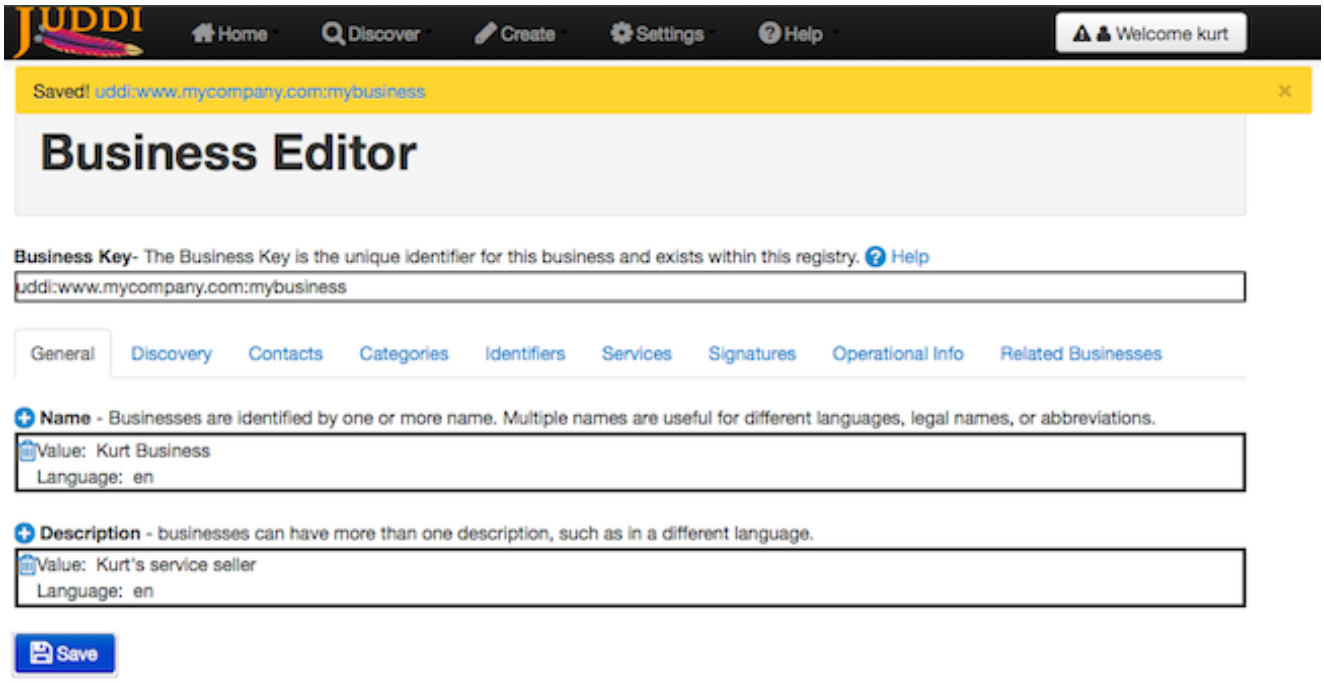


Figure 2.9. Create Business

See the Client and GUI Guide [stam-oree] for more details on how to use the GUI.

2.7. Running the demos in the disto

The jUDDI distribution ships with a lot of demos to get yourself more familiarized with the features of jUDDI. You are encouraged to go over the demos and follow the instructions in the README files. To ensure the demos work they use the root publisher. In practice you should not be using the root publisher for this, but rather your own publisher you created above. To reference your own publisher simply update the uddi.xml file in each demo. For more details on running the demos see the Client and GUI Guide [stam-oree].

2.8. Examples on the jUDDI blog

The jUDDI blog at http://apachejuddi.blogspot.com/ has examples as well as screencasts. This can be a useful resource to learn about some new feature or to simply get started.

2.9. What is new in jUDDI 3.2?

Here's the change log for version 3.2

- A new end user interface based on Twitter's Bootstrap
- A new administrative user interface based on Twitter's Bootstrap with in browser monitoring

What is new in jUDDI 3.2?

- A client side subscription callback API
- Client distribution package
- Many more examples
- WADL to UDDI mappings
- All credentials are now encryptable with command line tools
- Removal of the porlet services
- Deployment templates for Jboss EAP 6+
- Client side digital signature support
- REST style interface for Inquiry API
- Added many more tModels to the base install
- More documentation

Chapter 3. jUDDI Architecture

3.1. jUDDI Server

The jUDDI Architecture leverages well known frameworks to minimize the codebase we need to maintain. The API layer uses JAX-WS, while the persistence layer uses JPA. The entire server is packages as a war archive that can be deployed to different servlet containers with minimal configuration changes. The JPA layer uses JDBC to communicate to a relational database. Figure 3.1, “jUDDI Architecture” shows the different components, where the implementation providers marked with a blue dot are the implementations we use by default.

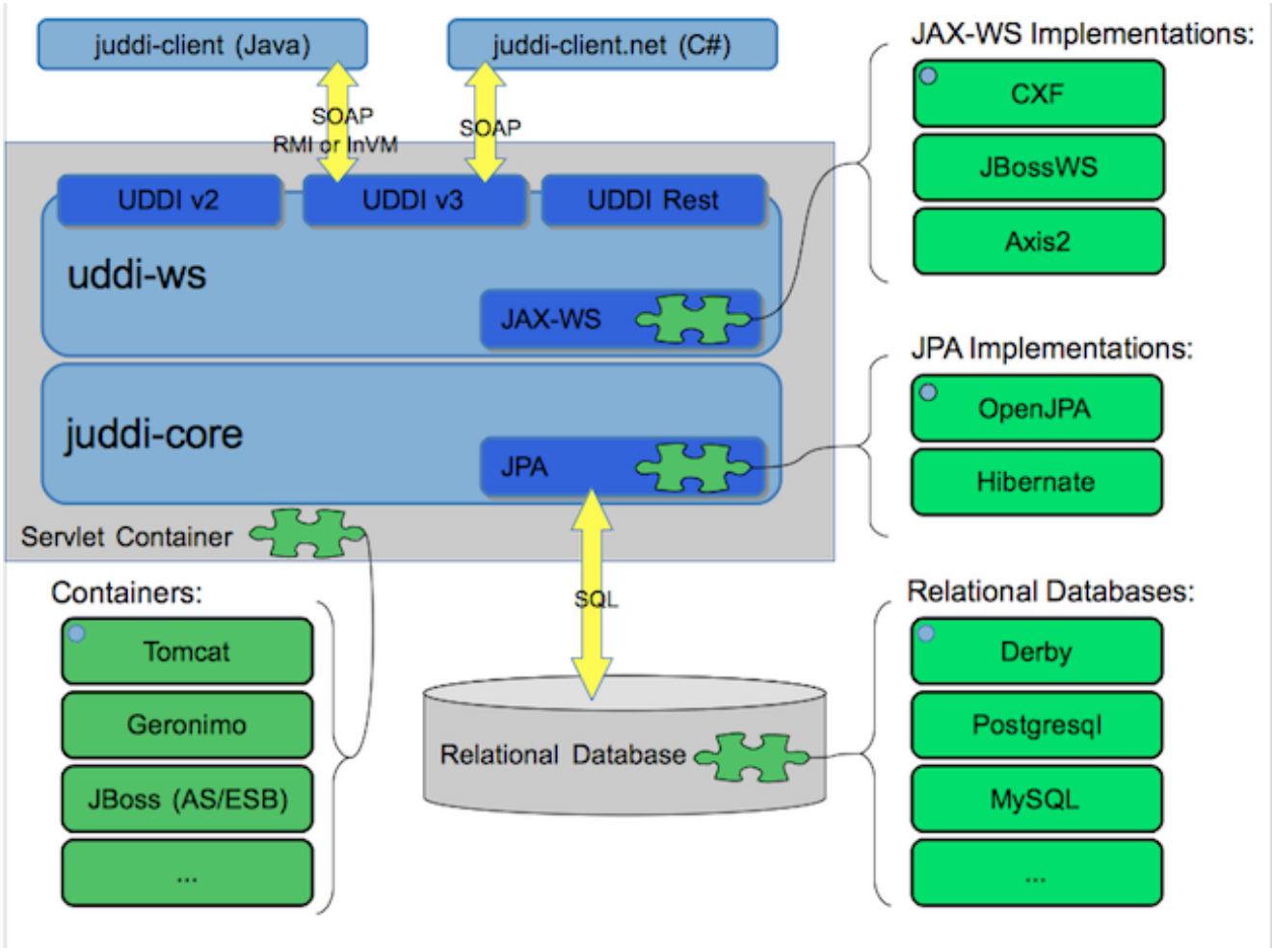


Figure 3.1. jUDDI Architecture

3.1.1. UDDI API layer `uddi-ws` using JAX-WS

The API layer is generated from the WSDL files provided with the UDDI specification. Since the 3.2 release we support both the UDDIv2 as well as the UDDIv3 API. The `uddi-ws` components

Core UDDI juddi-core using JPA

leverages `JAX-WS` annotations to bring up the UDDIv2 and v3 Endpoints. In addition to these two sets of SOAP based services, we also support a REST based API. The REST based API is a subset of the SOAP API. The default JAX-WS implemention used is Apache CXF, but we also offer scripted deployments for JBossWS and Axis2. Each WebService stack relies on the web.xml as well as vendor specific configuration files. For example, CXF uses a beans.xml file in the WEB-INF directory. For more details on this see ???.

The `juddi-client.jar` can be used on the client side to communicate with the API layer. The juddi-client can be configured to use either SOAP, RMI or and inVM protocol, where the inVM protocol is the most performant. For more details on the juddi-client configuration options see the Client Guide [stam-oree].

3.1.2. Core UDDI `juddi-core` using JPA

The jUDDI server logic is packaged in the `juddi-core.jar`. It implements all of the server side behavior defined in the UDDI specification. For persistence it uses the Java Peristence Api (JPA). The default JPA implemenation used is OpenJPA, but Hibernate is supported as well. The configuration for JPA implementations lives in the `WEB-INF/classes/META-INF/persistence.xml` file. This file also references the datasource that is used to connect to the datasource.

Important

It is important to note that there are two JARs provided through maven. If you will be using Hibernate, please use the juddi-core JAR, if you are using OpenJPA, use juddi-core-openjpa.

The difference between these JARs is that the persistence classes within juddi-core-openjpa have been enhanced (http://people.apache.org/~mprudhom/openjpa/site/openjpa-project/manual/ref_guide_pc_enhance.html). Unfortunately, the Hibernate classloader does not deal well with these enhanced classes, so it it important to note not to use the juddi-core-openjpa JAR with Hibernate.

3.1.3. Relational Databases

By default we ship jUDDI preconfigured with a Java based Database called `Derby`. This database persists to the local file system, typically from where the application was started.

Note

To switch databases, you need to change the JDBC driver configuration in the datasource as well as the database dialect setting in the persistence.xml.

For details on switching database see the Section 4.3, “Configuration Database Connections”.

3.1.4. Servlet Containers

The jUDDI server is packaged up a WebArchive (`juddiv3.war`). This war archive can be deployed to different servlet containers with minimal configuration changes. By default we ship on Apache Tomcat but we also have scripted deployment support for GlassFish and JBoss.

Tip

Most open source EE6 containers (JBoss, Geronimo, Glassfish) ship with jUDDI preconfigured to pass the JAXR tests in the TCK.

When switching containers you may need to use different configuration to create a datasource. Some containers already package up a WebServices stack which can be used instead of the CXF packages up in juddiv3.war/WEB-INF/lib. In that case the number of dependent jars in the `juddiv3.war` can be reduced significantly. For details on switching containers see the Chapter 8, *How to deploy jUDDI To?*.

3.2. jUDDI GUI `juddi-gui.war`

The jUDDI GUI is also a Web Archive that is deployed along side the `juddiv3` server in the same servlet container. The GUI uses the `juddi-client` to communicate to the UDDI API Endpoints. It can use a SOAP, RMI or an inVM transport protocol, so the GUI can be deployed in a different location then the server as long as it can connect to the UDDI SOAP API.

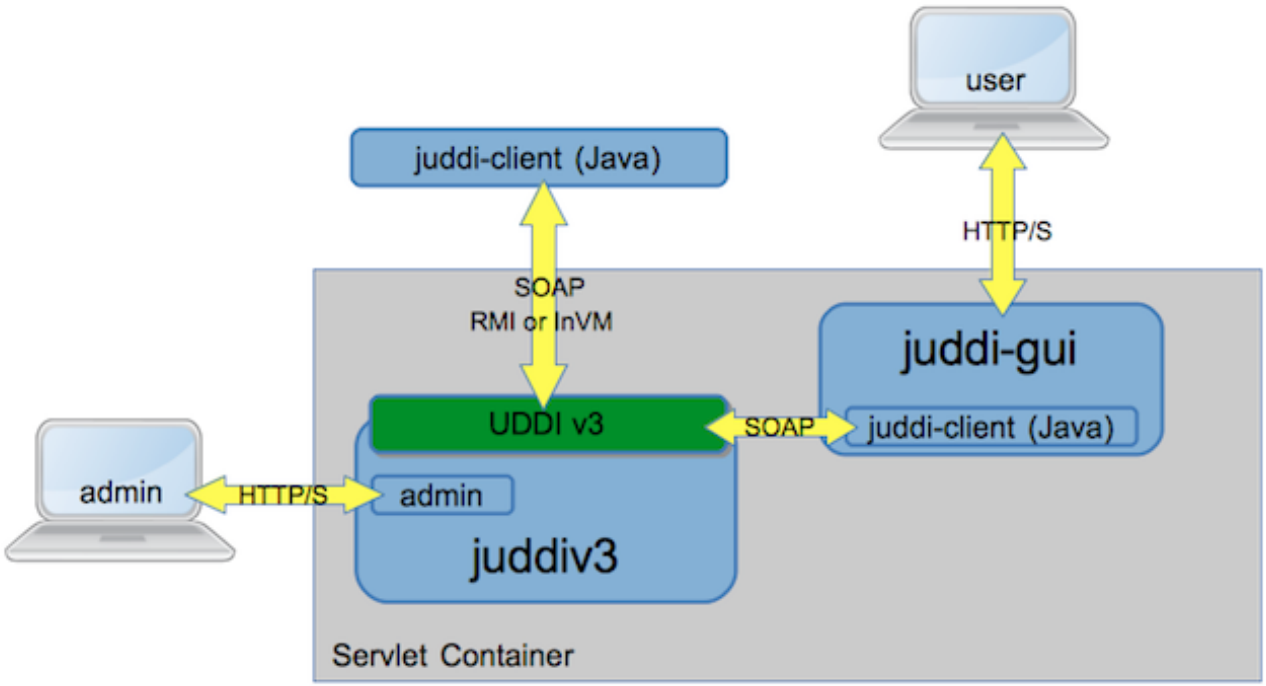


Figure 3.2. jUDDI Client and Console Architecture

jUDDI
GUI
`juddi-gui.war`

Figure 3.2, “jUDDI Client and Console Architecture” shows the admin console and the juddi-gui. Typically one one run the admin console behind a firewall. The admin console interacts over a jUDDI WS API and, among other things, it can be used to create and delete publishers.

The `juddi-gui` can be configured to connect to any UDDIv2 or UDDIv3 compliant UDDI server.

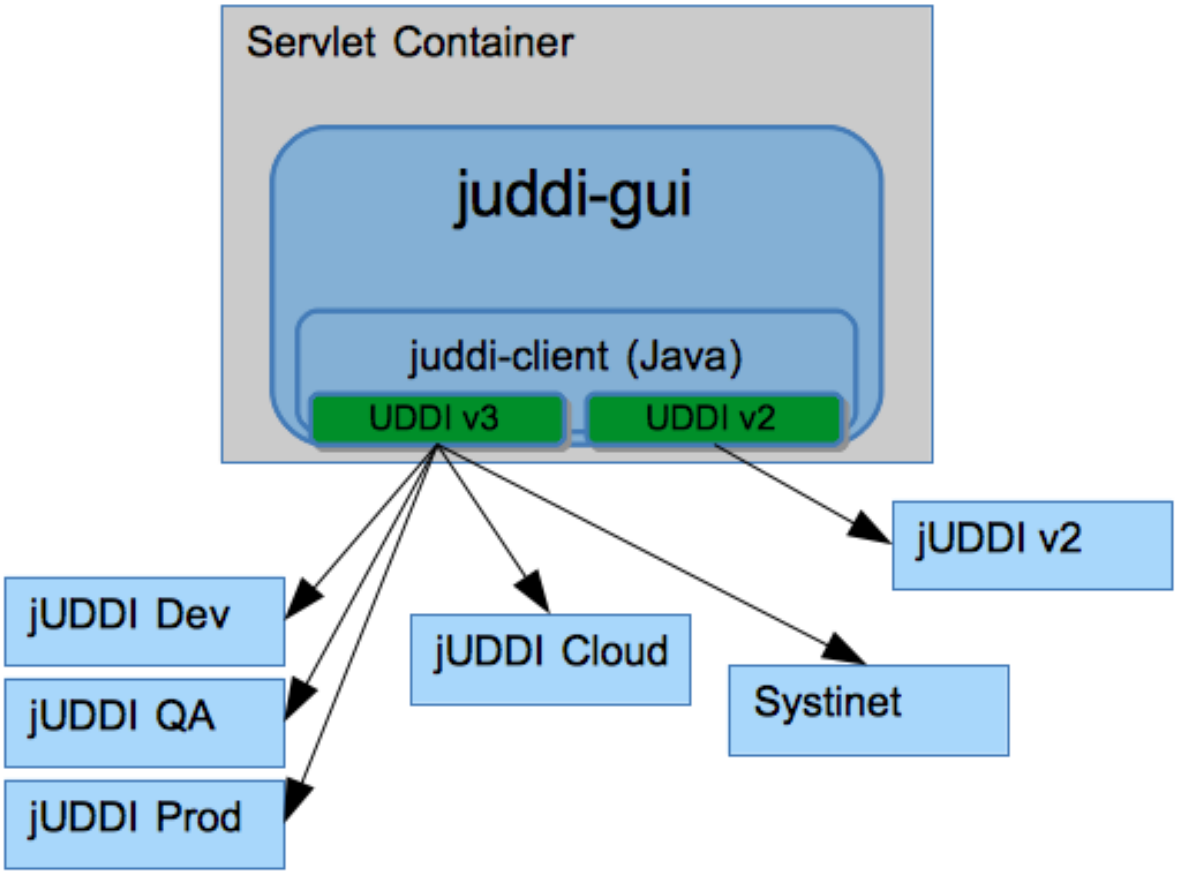


Figure 3.3. jUDDI Console Architecture

You may have a jUDDI v3 Server for each type of environment (Dev, QA and Prod) and you would only need one console to connect to each one of them.

For details on using the GUI see the Client and GUI Guide [stam-oree].

Chapter 4. Administration

4.1. Changing the Web Server Listen Port

If you want to change the port Tomcat listens on to something non-standard (something other than 8080), use the following guidance.

jUDDI Server (Tomcat) - This assumes you are using the jUDDI server bundled with Apache Tomcat. For other application servers, consult their documentation, however the juddiv3.xml must still be altered.

- Edit `conf/server.xml` and change the port within the <Connector> element.
- Edit `webapps/juddiv3/WEB-INF/classes/juddiv3.xml` and change the port number jUDDI Server Baseurl.
- Edit `webapps/juddiv3/WEB-INF/config.properties` and change the port numbers for "securityurl" and "juddipapi".
- Edit `webapps/juddi-gui/META-INF/config.properties` and change the port numbers for all of the URLs listed.

If these changes are made before jUDDI has been started for the first time, then no further action is required. If jUDDI has been previously started, you'll need to either A) update the URL information for the Node's root business entity URLs or B) turn on "Seed Always" in the juddiv3.xml file to auto update the changes.

4.2. Administering Users and Access Control

As of version 3.2, jUDDI Authentication is handled from two perspectives, administrator and end user access.

4.2.1. Administrative Users

Administrative users have special access to juddi-gui's remote configuration page at http://localhost:8080/juddi-gui/settings.jsp and to the Administrative Console at http://localhost:8080/juddiv3/admin. Access to both of these is configured at the container level (i.e. Jboss, Tomcat, etc). By default, users that need to access these pages need to have the "uddiadmin" role (which is defined in the WEB-INF/web.xml of both web application archives). When you are running on tomcat this configuration can be found in the `<tomcat>/conf/tomcat-users.conf` file.

4.2.2. End Users

End users typically will either access jUDDI's services directly at http://localhost:8080/juddiv3/ or via the user interfaces http://localhost:8080/juddi-gui. In both cases, authentication is handled

via jUDDI's Authentication providers which is configured in `juddiv3.war/WEB-INF/classes/juddiv3.xml`.

4.2.2.1. Under the Hood

In order to enforce proper write access to jUDDI, each request to jUDDI needs a valid authToken. Note that read access is not restricted (by default, but can be enabled) 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("username");
ga.setCred("password");
org.uddi.api_v3.AuthToken token = securityService.getAuthToken(ga);
```

The property `juddi/auth/*` in the `juddiv3.xml` 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/authenticator/class` 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.

4.2.2.2. Choosing a Cryptographic Provider

jUDDI provides a number of cryptographic providers. Some of them may not be available in your region of the world due to export restrictions. All of these providers are provides that are included with the Oracle Java Runtime Environment.

4.2.2.2.1. jUDDI's Cryptographic Providers

Tip

The AES256Cryptor requires the Sun Java unlimited strength Crypograhpic Extensions to be installed. OpenJDK users are not affected by this.

In the following section, Authentication, a Cryptographic Provider must be selected using the following property in juddiv3.xml:

```
juddi/cryptor
```

4.2.2.2.2. jUDDI Server Providers

- org.apache.juddi.cryptor.DefaultCryptor - Password Based Encryption With MD5 and DES
- org.apache.juddi.cryptor.TripleDESCrytor - Triple DES 168 bit
- org.apache.juddi.cryptor.AES128Cryptor - Advanced Encryption Standard 128 bit
- org.apache.juddi.cryptor.AES256Cryptor - Advanced Encryption Standard 256 bit

4.2.2.3. jUDDI Client Providers (Java and .NET)

- org.apache.juddi.v3.client.crypto.DefaultCryptor - Password Based Encryption With MD5 and DES
- org.apache.juddi.v3.client.crypto.TripleDESCrytor - Triple DES 168 bit
- org.apache.juddi.v3.client.crypto.AES128Cryptor - Advanced Encryption Standard 128 bit
- org.apache.juddi.v3.client.crypto.AES256Cryptor - Advanced Encryption Standard 256 bit

4.2.2.3.1. Encrypting a Password

To encrypt a password, the jUDDI Tomcat server comes with a basic Windows Batch file and a Unix Bash script which will fire off the correct Java command. It is located at the following path:

```
{tomcat_home}/bin/juddi-cryptor.bat/sh
```

Tip

The jUDDI-Client (Java only) uses the same encryption keys and the jUDDI Server, therefore encrypted passwords using this tool will work with the jUDDI-client's configuration file.

In addition, an MD5 hashing program is included to assist with setting users passwords for the MD5XMLDocAuthenticator.

```
{tomcat_home}/bin/juddi-md5.bat/sh
```

Tip

You can generate new encryption keys using this utility by specifying the System Property -Dgenerate=true option. You can then use them using the System Property -Djuddi.encryptionKeyFile.TripleDESCrytor=path/to/key

4.2.2.4. 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 Default Authenticator in production. It does not compare passwords to anything!

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/authenticator/class = org.apache.juddi.auth.JUDDIAuthentication
```

4.2.2.5. XMLDocAuthentication

The XMLDocAuthentication implementation needs a XML file on the classpath. The juddiv3.xml file would need to look like

```
juddi/auth/authenticator/class = org.apache.juddi.auth.XMLDocAuthentication
```

```
juddi/auth/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.

4.2.2.6. CryptedXMLDocAuthentication

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

```
juddi/auth/authenticator/class =
 org.apache.juddi.auth.CryptedXMLDocAuthentication
juddi/auth/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.

4.2.2.7. MD5XMLDocAuthenticator

The MD5XMLDocAuthenticator implementation is similar to the XMLDocAuthentication implementation, but the passwords are hashed using MD5.

```
juddi/auth/authenticator/class =
 org.apache.juddi.auth.MD5XMLDocAuthenticator
juddi/auth/usersfile = juddi-users-hashed.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.

4.2.2.8. LDAP Authentication

LdapSimpleAuthenticator provides a way of authenticating users using LDAP simple authentication. It is fairly rudimentary and more LDAP integration is planned in the future, but this class allows you to authenticate a user based on an LDAP prinicipal, provided that the principal (usually the distinguished name) and the juddi publisher ID are the same.

To use this class you must add the following properties to the juddi3v.xml file:

```
juddi/auth/authenticator/class=org.apache.juddi.auth.LdapSimpleAuthenticator
juddi/auth/authenticator/url=ldap://localhost:389
juddi/auth/authenticator/style=simple
```

The juddi/authenticator/url property configures the LdapSimpleAuthenticator class so that it knows where the LDAP server resides. Future work is planned in this area to use the LDAP uid rather than the LDAP principal as the default publisher id.

LdapExpandedAuthenticator provides a slightly more flexible way to authenticate users via LDAP.

```
juddi/auth/authenticator/
class=org.apache.juddi.v3.auth.LdapSimpleAuthenticator
juddi/auth/authenticator/url=ldap://localhost:389
juddi/auth/authenticator/style=simple
juddi/auth/authenticator/ldapexp=CN=%s, OU=Users,DC=Domain, etc
```

4.2.2.9. JBoss Authentication

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.

Tip

The JBoss authentication is not distributed with jUDDI. It can be found here: http://svn.apache.org/viewvc/juddi/extras/jbossauthenticator/src/org/apache/juddi/auth/JBossAuthenticator.java?view=markup

To use this class you must add the following properties to the juddiv3.xml file:

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

The juddi/auth/authenticator/class property plugs the JbossAuthenticator class into the jUDDI the Authenticator framework. The juddi/sercuityDomain, 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></application-policy-name>. If a lookup refers to a non existent application policy it defaults to a policy named other.

4.2.2.10. Container Based Authentication

Certain security configurations may use HTTP based authentication. In this scenario, jUDDI simply trust's that the container will authenticate the user via some mechanism and uses that username for interactions with jUDDI. To configure this setup, use the following configuration settings in juddiv3.xml:

```
juddi/auth/authenticator/
class=org.apache.juddi.auth.HTTPContainerAuthenticator
juddi/auth/authenticator@useAuthToken=false
```

In addition, you'll have to make whatever changes necessary to the juddiv3.war/WEB-INF/web.xml file in order to use the chosen authentication mechanism. See your appliation server's documentation for details on this.

4.2.2.11. Authentication by Proxy (HTTP Header)

Certain security configurations that enforce authentication before requests come to the web application, such as via Apache HTTPD or a reverse SSL proxy. In these cases, the proxy provided

authenticates the user, then passes along the user's identity via a HTTP header. To configure this setup, use the following configuration settings in juddiv3.xml:

```
juddi/auth/authenticator/class=org.apache.juddi.auth.HTTPHeaderAuthenticator
juddi/auth/authenticator/header=(Some HTTP Header)
juddi/auth/authenticator@useAuthToken=false
```

4.3. Configuration Database Connections

4.3.1. 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 *juddiv3.war/WEB-INF/classes/META-INF/persistence.xml*. The content of this file is pretty standard between JPA implementations, however there can be slight differences. To make it easy we created different versions for different JPA implementations and target platforms. All JPA implementation have an enhancement phase, where the persistence *model* classes are enhanced. Hibernate does this at runtime, OpenJPA prefers doing this at compile time. This is the reason we ship two versions of *juddi-core*, where the *juddi-core-openjpa.jar* contains classes (byte-code) enhanced by OpenJPA. This is the reason this jar is larger then the *juddi-core.jar*.

For Hibernate, for testing 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.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>
```

For OpenJPA the persistence.xml 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.apache.openjpa.persistence.PersistenceProviderImpl</
provider>
    <non-jta-data-source>java:comp/env/jdbc/JuddiDS</non-jta-data-source>
    <!-- entity classes -->
    <class>org.apache.juddi.model.Address</class>
    <class>org.apache.juddi.model.AddressLine</class>
   ...
    <class>org.apache.juddi.model.UddiEntity</class>
    <class>org.apache.juddi.model.UddiEntityPublisher</class>
    <properties>
      <property name="openjpa.jdbc.SynchronizeMappings"
 value="buildSchema(SchemaAction='add')"/>
      <property name="openjpa.Log" value="DefaultLevel=WARN, Tool=INFO"/>
      <property name="openjpa.jdbc.UpdateManager" value="operation-order"/>
      <property name="openjpa.jdbc.DBDictionary" value="derby"/>
      <!-- dialects: derby, postgres, mysql, oracle, sybase, sqlserver
           for a complete list check the OpenJPA documentation -->
      <property name="openjpa.RuntimeUnenhancedClasses" value="warn"/>
      <property name="openjpa.Compatibility"
 value="CheckDatabaseForCascadePersistToDetachedEntity=true"/>
    </properties>
  </persistence-unit>
</persistence>
```

In this case we reference a *jta-data-source* called *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>
```

By default the juddiv3.war is configured to be used on Tomcat using OpenJPA. However the download bundle lets you specify different target platforms resulting in a different setup. In all cases it will point to the embedded Derby database.

4.3.2. Switching to another Database

We recommend switching to an enterprise-level database before going to production. Most JPA providers support a large number of Databases and switching to another database is achieved by updating the configuration settings in both the persistence.xml and datasource files. The recipe is:

- change the database dialect in the persistence.xml.
- change the database connection information in the datasource.
- add the database specific driver to your classpath.
- in some cases (Oracle is one such case) you will need to use sequences for the ID generation, in this case you will need an *orm.xml* file. We ship a *orm.xml.example* along side the *persistence.xml*. Rename this file and update this to your liking.

Some examples for specific databases are given below.

Warning

Tomcat copies the *context.xml* to *<tomcat>/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*.

4.3.3. Switch to MySQL on Tomcat using OpenJPA

Check if you have are using Hibernate of OpenJPA, by looking at the jars in the *juddiv3.war/WEB-INF/lib*. Edit the dialect in the *persistence.xml* For OpenJPA:

```
<property name="openjpa.jdbc.DBDictionary" value="mysql"/>
```

Next edit the datasource. For tomcat you need to update the *juddiv3/META-INF/context.xml* which 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>
```

Finally you need to add the MySQL mysql driver (i.e. The *mysql-connector-java-5.1.6.jar*) to the classpath. Note that this jar may already by in the tomcat/lib directory, in which case you can move on to the step and create the mysql juddiv3 database. 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*, and delete the *<tomcat>/conf/CATALINA/localhost/juddiv3.xml* file.

4.3.4. Switch to Postgres on Tomcat using OpenJPA

Check if you have are using Hibernate of OpenJPA, by looking at the jars in the *juddiv3.war/WEB-INF/lib*. Edit the dialect in the *persistence.xml* For OpenJPA:

```
<property name="openjpa.jdbc.DBDictionary" value="postgres"/>
```

Next edit the datasource. For tomcat you need to update the *juddiv3/META-INF/context.xml* which 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="juddi" password="juddi"
            driverClassName="org.postgresql.Driver"
            url="jdbc:postgresql://localhost:5432/juddi"
            maxActive="8"/>
</Context>
```

To create a MySQL database name *juddi* use

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

Be sure to have *postgresql-8.3-604.jdbc4.jar* to the classpath. Note that this jar may already by in the tomcat/lib directory, in which case the final step is to delete the *<tomcat>/conf/CATALINA/localhost/juddiv3.xml* file.

4.3.5. Switch to Postgres on JBoss using Hibernate

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:

```
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*. Next edit the juddi-ds.xml datasource file with the settings for the postgres connection info:

```
<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, for Hibernate on JBoss use:

```
<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.

Switch to Oracle on Tomcat using Hibernate

4.3.6. Switch to Oracle on Tomcat using Hibernate

To switch over to Oracle you need to add the oracle driver (i.e. the_classes12.jar_) to the classpath and you will need to edit the *persistence.xml*

```
<property name="hibernate.dialect"
 value="org.hibernate.dialect.Oracle10gDialect"/>
```

To create a Oracle database name juddiv3 with the ultimate in minimalism use

```
sqlplus> create database juddiv3;
```

then you probably want to switch to a user which is a bit less potent then *root* and set the appropriate password, and delete the *<tomcat>/conf/CATALINA/localhost/juddiv3.xml*

4.3.6.1. 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 orm.xml.example specifies a "juddi_sequence" sequence to be used with jUDDI. Rename this file and update it to your liking.

4.3.7. Switch to HSQL on Tomcat using Hibernate

First make sure you have a running hsqldb. For a standalone server startup use:

```
java -cp hsqldb.jar org.hsqldb.server.Server --port 1747 --database.0
 file:juddi --dbname.0 juddi
```

Next, connect the client manager to this instance using:

```
java -classpath hsqldb.jar org.hsqldb.util.DatabaseManagerSwing --driver
 org.hsqldb.jdbcDriver --url jdbc:hsqldb:hsql://localhost:1747/juddi  -user
 sa
```

and create the juddi user:

```
CREATE USER JUDDI PASSWORD "password"  ADMIN;
CREATE SCHEMA JUDDI AUTHORIZATION JUDDI;
SET DATABASE DEFAULT INITIAL SCHEMA JUDDI;
ALTER USER juddi set initial schema juddi;
```

Switch to other db

From now on, one can connect as JUDDI user to that database and the database is now ready to go. To switch jUDDI over to HSQL you need to add the hsql driver (i.e. The *hsqldb.jar*) to the classpath and you will need to edit the *persistence.xml*

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

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>
    <!-- HSQL data source -->
    <Resource name="jdbc/JuddiDS" auth="Container"
            type="javax.sql.DataSource" username="JUDDI" password="password"
            driverClassName="org.hsqldb.jdbcDriver"
            url="jdbc:hsqldb:hsql://localhost:1747/juddi"
            maxActive="8"/>
</Context>
```

4.3.8. 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.

4.3.9. Override persistence properties in the juddiv3.xml

The juddiv3.xml file can be externalized; if you give the path of juddiv3.xml in the JVM args, the juddiv3.xml will not be picked up from the WAR. To use this set the *juddi.propertiesFile* to a location of your configuration file. This allows the user to change the jUDDI properties without having to open up the juddiv3.war file. For this use case it makes sense that also persistence properties can be overridden as well in the juddiv3.xml file. The following properties can be set:

Table 4.1. Hibernate properties that can be referenced in the *juddiv3.xml* file

property name	description	example value
persistenceProvider	JPA Implementation	Hibernate
hibernate.connection.datasource	datasource name	java:/jdbc/JuddiDS
hibernate.hbm2ddl.auto	hibernate to ddl setting	java:/jdbc/JuddiDS
hibernate.default_schema	Schema name	JuddiSchema
hibernate.dialect	DataBase vendor name	org.hibernate.dialect.DB2Dialect

4.4. Logging

The jUDDI codebase uses the *commons-logging-api*, and *log4j* as the default logging implementation. The *juddiv3/WEB-INF/classes/commons-logging.properties* sets the logging to

log4j. The default *log4j* configuration logs to a *juddi.log* file in the *tomcat/logs* directory. The *log4j* configuration lives in the *juddiv3/WEB-INF/classes/log4j.properties* file, which is referenced in the *web.xml*

```
<context-param>
    <param-name>log4jConfigLocation</param-name>
    <param-value>/WEB-INF/classes/log4j.properties</param-value>
</context-param>
```

The *commons-logging* and *log4j* jars are shipped in the *juddiv3/WEB-INF/lib* directory.

If you are using CXF for the webservice stack you can log the request/response xml by adding

```
log4j.category.org.apache.cxf=INFO
```

to your log4j.properties and the cxf.xml file should contains this:

```
<cxf:bus>
    <cxf:features>
        <cxf:logging/>
    </cxf:features>
</cxf:bus>
```

The jUDDI beans.xml specifies the location of this file at *META-INF/cxf/cxf.xml*.

4.5. Administering the GUI (juddi-gui.war)

There are a few things worth mentioning for administering the jUDDI Graphical User Interface. The first is user authentication, which is covered in the authentication chapter. The other the the Digital Signature Applet. This applet enables users to digitally signed UDDI entities via the GUI. There are a number of requirements in order for this to work.

- The applet must be digitally signed. It is recommended that this signed by the administrator using the SSL certificate of the jUDDI instance. If it is not signed, it may not be able to digital certificates.
- The Oracle Java browser plugin must be installed. For details on this, visit Oracle's website.
- The end user must have a digital certificate installed that is accessible to the browser. On Windows computers, this is supported by Internet Explorer, Opera and Chrome which use the Windows Certificate Store (Start > Run > MMC, Add Certificates). Firefox uses its own certificate store. On MacOS, Safari uses the Mac Keychain.

4.6. Task: Signing the Digital Signature Applet jar file

```
jarsigner -keystore your.keystore -storepass yourpass -keypass keypass
 <pathto>/juddi-gui.war/applets/juddi-gui-dsig-all.jar
```

Note: Jarsigner comes with most JDKs and has many command line options.

4.7. Administrating your jUDDI Instance using the Administrative Console

Your instance of the jUDDI (juddiv3.war) can be managed via the administration console. It can be access url the following URL:

```
http://localhost:8080/juddiv3/admin
```

By default, only users with the role "uddiadmin" are allowed to access this page. In addition, it must be accessed from the same computer hosting juddiv3.war (this can be changed if needed). When accessing the URL, you should be prompted for login via username/password (this can also be changed to another mechanism).

After authenticating, you will be prompted with a very similar interface to the juddi-gui.war. From here, you can perform a number of tasks.

- Access Status and Statistics of jUDDI
- Configure jUDDI (juddiv3.war)
- Access the jUDDIv3 API, which provides a number of administrative tasks and functions (requires an additional login)*

*Why is there another login required for the jUDDIv3 API functions?

The answer is because the admin console will be directly accesses a web service and it requires a user account with juddi admin rights. This may be the same username you use to access the admin console (juddiv3.war/admin) but unfortunately, this double login is unavoidable.

4.8. Configure jUDDI

From the browser, it is possible to configure jUDDI's web services via the web browser. All of the settings available from the chapter on configuring jUDDI can be set there.

4.8.1. Enabling Remote Access

The jUDDI Configuration page by default is only accessible via the same host that is hosting the server. To enable remote access, change the setting

```
config/props/configLocalHostOnly=true
```

To false.

Enabling
Remote
Access

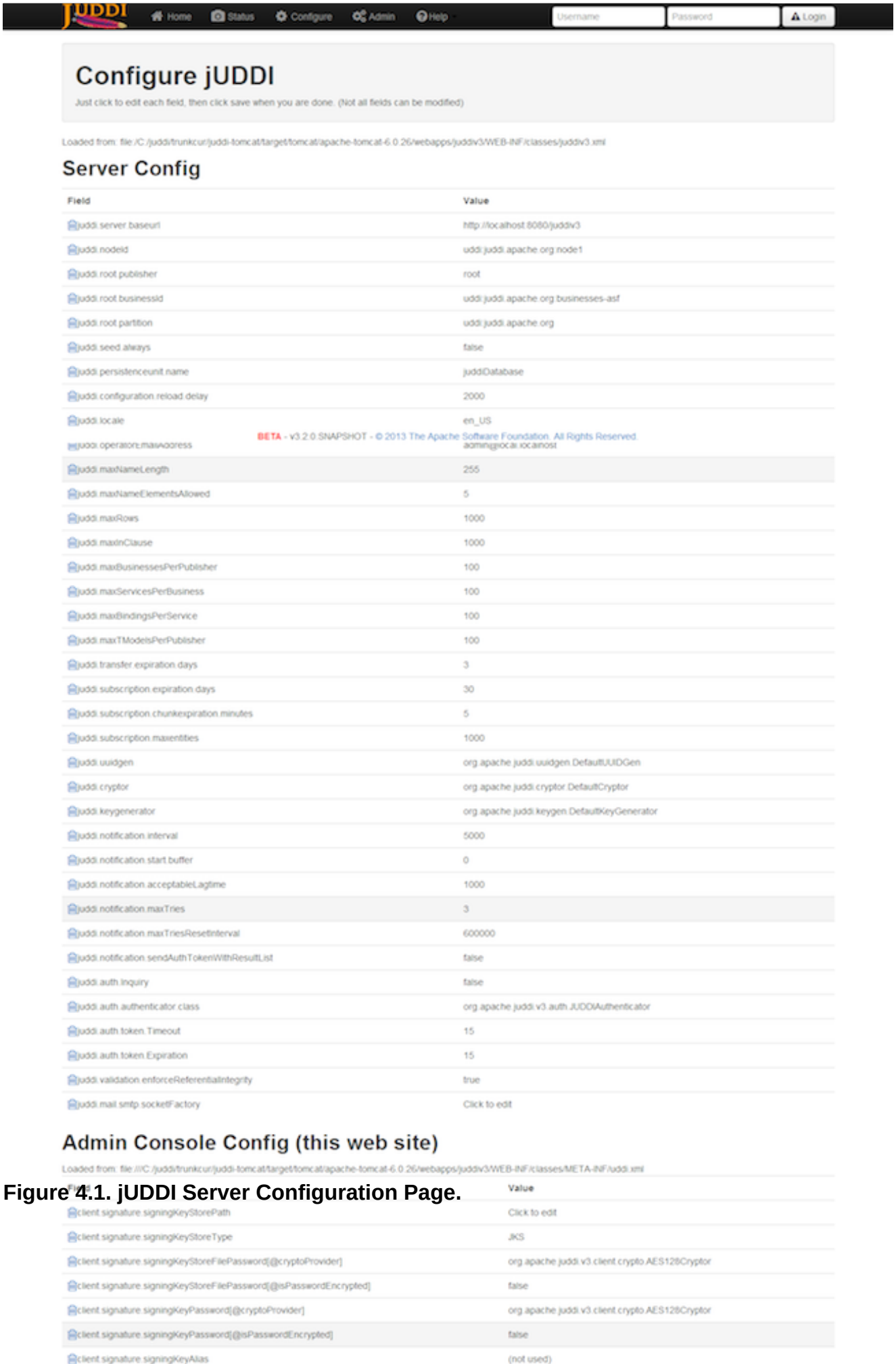


Figure 4.1. jUDDI Server Configuration Page.

4.9. Monitoring the Status and Statistics

The Statistics and Status page provides valuable information to administrators and developers looking to trouble shoot or debug problems with jUDDI.

4.9.1. Statistics

The Statistics page provides you with access to usage counts and time spent processing on each method of each service that jUDDI provides.

Tip

This information can be pulled and is available in JSON encoded data from the following URL: http://localhost:8080/juddiv3/admin/mbeans.jsp

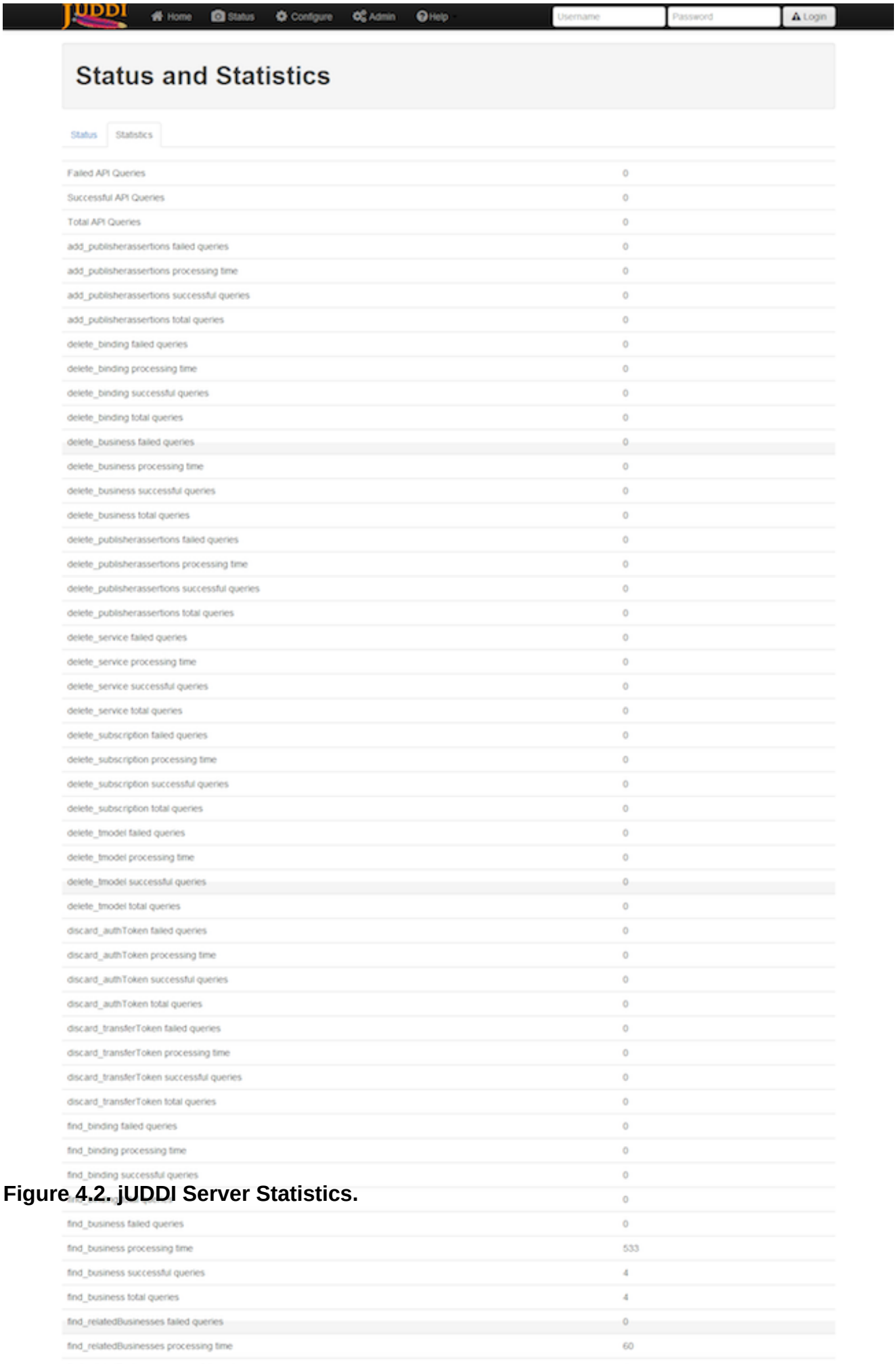


Figure 4.2. jUDDI Server Statistics.

or you can hook up the jconsole to look at the jUDDI mbeans

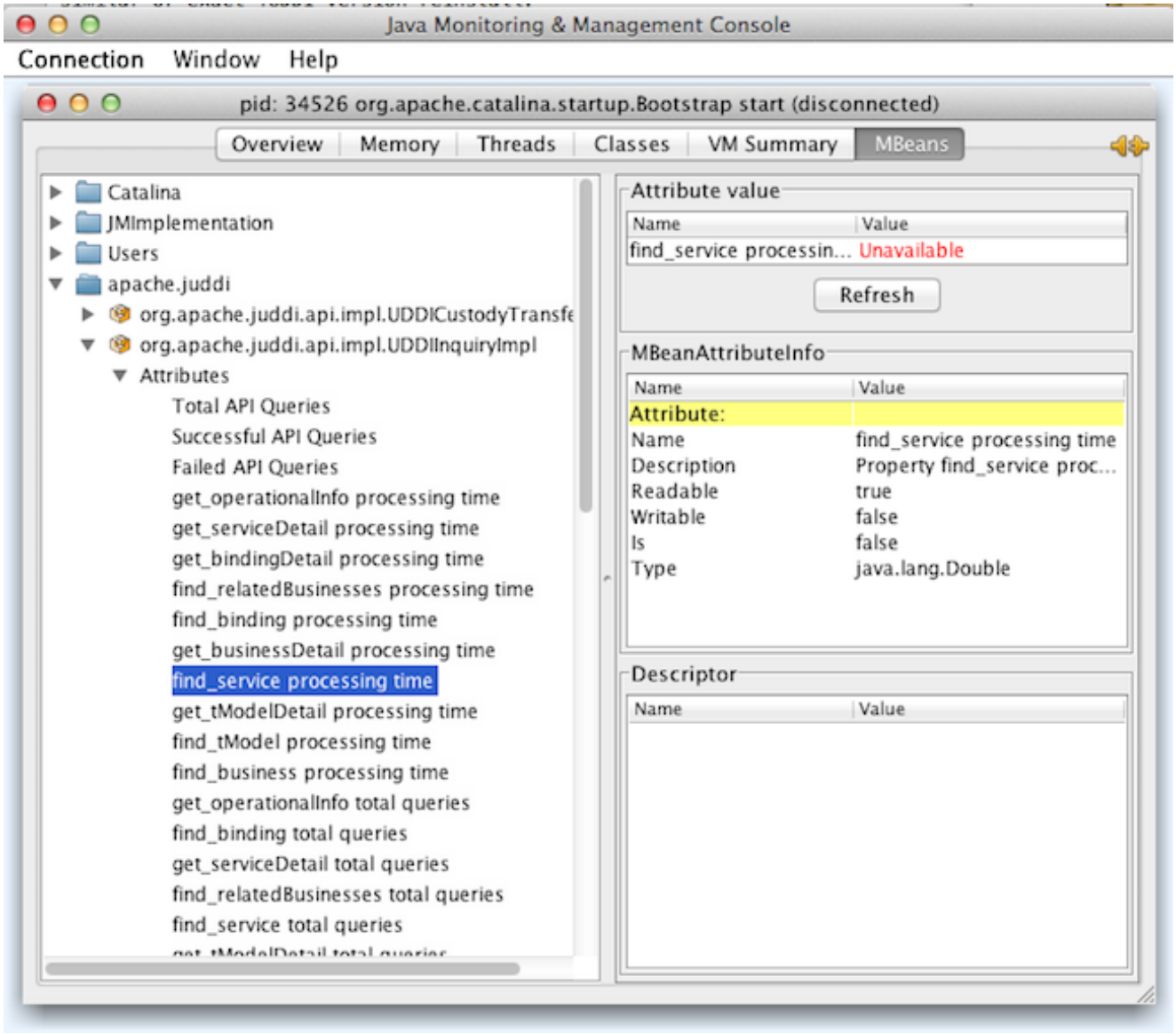


Figure 4.3. jUDDI MBeans.

4.9.2. Status

The Status page gives you the former "Happy jUDDI" page from version 2 of jUDDI.

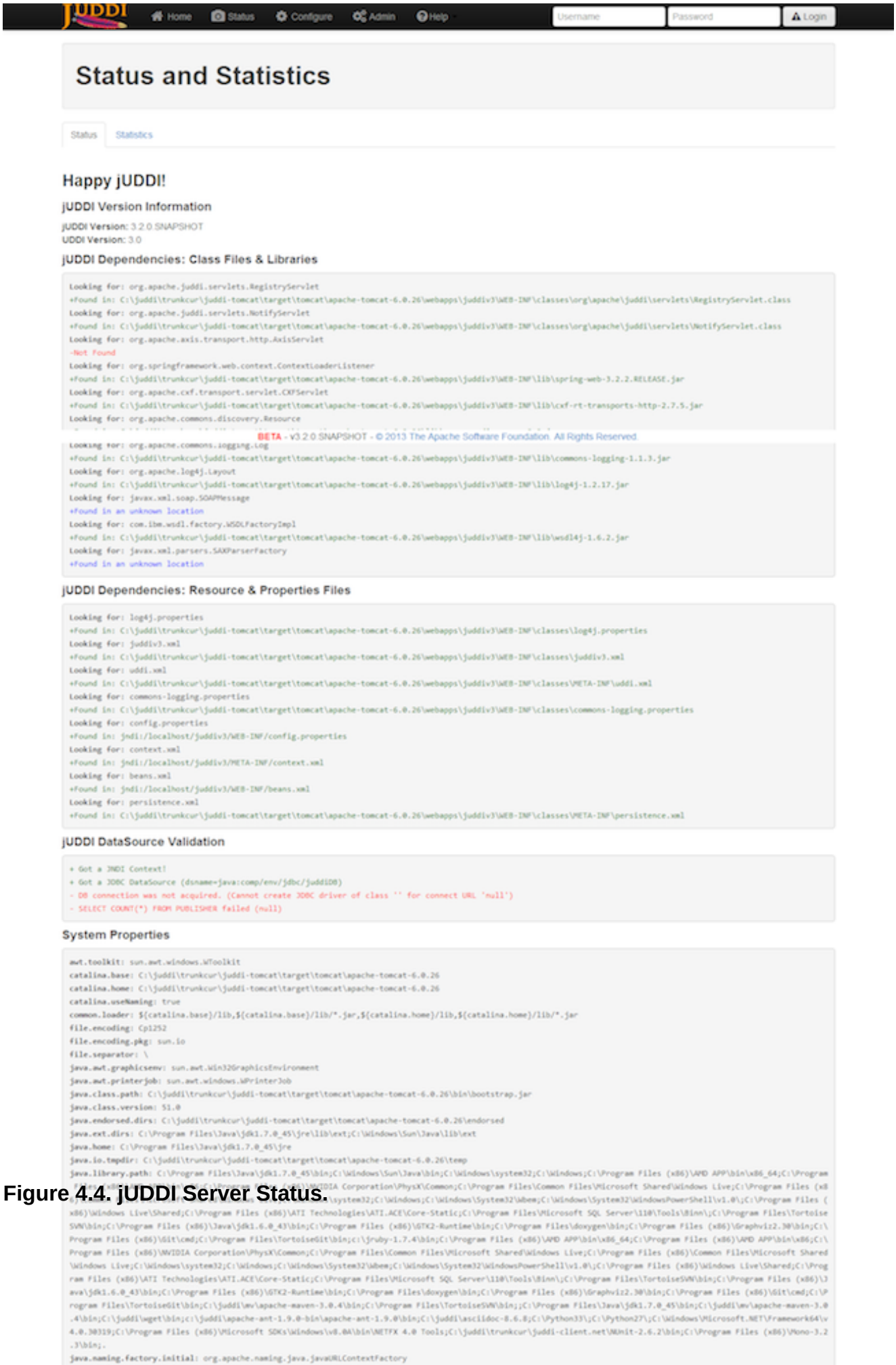


Figure 4.4. jUDDI Server Status.

4.10. Accessing the jUDDIv3 API

The jUDDI API is a web service that extends the UDDI specification. It provides various functions for both configuring the jUDDI server and for performing administrative functions, such as authorizing a new username as a publisher, user rights assignment and so on. This page will let you access the functions from the web browser.

Tip

You must authenticate using the top right hand side login/password box in order to use this.

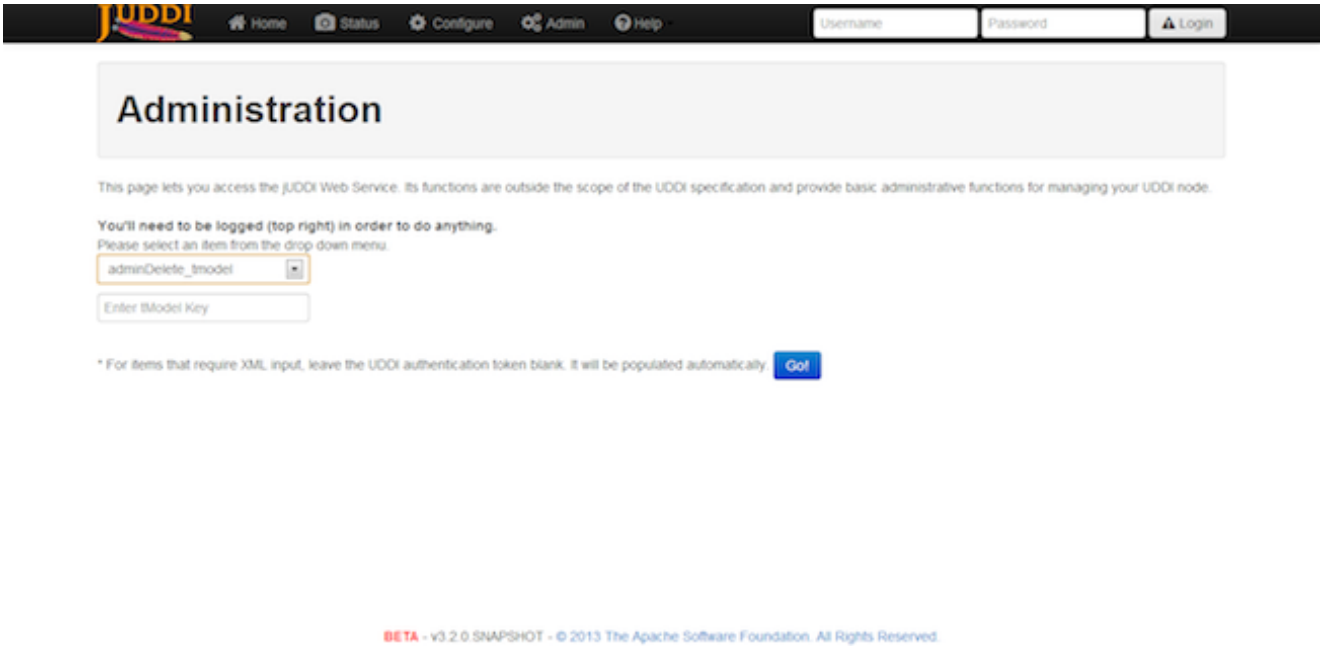


Figure 4.5. jUDDI API.

4.11. Security Guidance

This guide contains general security guidelines to ensure that your jUDDI server and jUDDI Client based application are relatively safe and to prevent authorized users.

This section is broken down into guidance for the jUDDI server and for the jUDDI Client

4.11.1. jUDDI Server

- Always use SSL or TLS for connections to and from the jUDDI server, especially connections where authentication is used. Use encrypted connections to the database server when possible. client configs (uddi.xml), database (juddiv3/WEB-INF/classes/META-INF/persistence.xml)

- If the juddi-gui web app is not on the same server as the juddiv3 web services web app, use SSL or TLS. (juddi-gui/WEB-INF/classes/META-INF/uddi.xml)
- Use UDDI Digital Signatures where appropriate. Enable all validation options. Java/.NET Clients + juddi-gui, uddi.xml uddi/client/signatures, checkTimestamps,checkTrust,checkRevocationCRL
- Require authentication for Inquiry API. (config/juddi/auth/Inquiry=true)
- Use a LDAP user store and set passwords to expire regularly. Enforce the usage of strong passwords of sufficient length and SSL for LDAP connections. (config/juddi/auth/token/authenticator)
- Encrypt all stored credentials (database, key stores, email, etc) with the highest possible encryption available. (config/juddi/cryptor=org.apache.juddi.v3.client.cryptor.AES256Cryptor or AES128)
- Configure Auth Tokens to expire with relatively short intervals. This should meet all automatic logout requirements and help reduce the risk that an intercepted auth token can't be reused by a 3rd party. (config/juddi/auth/token/Expiration) and (config/juddi/auth/token/Timeout)
- Configure Auth Tokens to require Same IP Enforcement. This is a mitigation factor for when a token is intercepted and attempted to be reused from another source. (config/juddi/auth/token/enforceSameIPRule=true)
- Configure Custody Transfer Tokens to expire with relatively short intervals. (config/juddi/transfer/expiration/days)
- Disable sending authentication tokens to subscription notifications (config/juddi/notification/sendAuthTokenWithResultList=false)
- If you're using the replication services, configure your application server to use mutual certification authentication for that deployment (per the specification's recommendation).

4.11.2. jUDDI Client (and developers)

- Never log auth tokens. Protect it as if it was a password
- Encrypt all stored credentials (key stores, UDDI credentials, etc) with the highest possible encryption available (uddi.xml)
- Discard auth tokens when they are no longer needed.

4.11.3. jUDDI GUI (Web user interface)

- Enable automatic logouts (WEB-ING/classes/META-INF/uddi.xml)
- All cached credentials are encrypted in the session tokens using an AES key that is generated at boot up time of the juddi-gui instance.

- Use SSL or TLS when connecting using your web browser to juddi-gui.
- The juddi-gui uses cookies to store user preferences, such as language and the current node.
- The juddi-gui makes heavy use of JavaScript using Jquery and JqueryUI. Without a JavaScript enabled browser that supports AJAX, the juddi-gui will not be functional. This usually implies Firefox 1.6 or higher, IE 6, Chrome/Chromium (nearly all versions), Opera v8 or higher, and Safari v2 or higher.
- The juddi-gui uses a Java applet that is used for Digital Signature support. This runs within your web browser. The Java plugin for your web browser must be enabled in order to use this functionality. In addition, the applet itself must be digitally signed (usually performed by the administrator, see article on this).
- The juddi-gui has built in validation for digital signatures. This requires a trusted key store. Ensure that the passwords are encrypted using the highest available crypto class and that the validation settings are enabled.
- The juddi-gui has a settings pages for altering the uddi.xml configuration file. By default, this is only accessible from the same machine running juddi-gui (i.e. localhost). This behavior can be changed by either using the setting page from localhost or by manually editing the uddi.xml page. Unless required, the recommended setting is to prevent remote configuration changes. If the settings page isn't required, it can be removed.
- The juddi-gui has a settings page that is password protected to prevent unauthorized changes. Use the strongest available mechanism to protect credentials. The default configuration is for HTTP BASIC. It is recommended to use this with SSL/TLS and/or switch to DIGEST based authentication. If the settings page isn't required, it can be removed.

4.12. Backups, Upgrading and Data Migration

There are several different strategies for managing your jUDDI backups.

4.12.1. Database Backups

Database backups are vendor specific and are effective for backup/restore to a similar or exact jUDDI version reinstall.

4.12.2. Config Backup

Aside from database backups, you should also make backup copies of all jUDDI configuration files and any files that you have customized to meet your operational needs.

4.13. Upgrading jUDDI

Sometimes, the jUDDI development team has no choice but to alter the database schema. In many cases, OpenJPA or Hibernate (both Java Persistence API provides) will automatically alter database columns when a new version is installed. In some cases, there may actually be data loss.

Tip

Check the jUDDI distribution notes before attempting an upgrade.

Important

Always perform a database level backup of your instance before attempting the upgrade.

4.14. Scaling jUDDI and Federation

The capabilities and components provided by jUDDI are designed to scale. The following will describe the options and known limitations of jUDDI.

4.14.1. Scaling the jUDDI Services (multiple servers)

The jUDDI web services (juddiv3.war) is designed to be scaled to multiple servers in a number of ways. The following sub sections outline the available options.

4.14.1.1. Scaling using a common database

The first and simplest mechanism is for the instances of juddiv3.war to share the same database. All of jUDDI's database calls are transactional SQL, meaning that concurrent changes will function just fine from multiple concurrent users. Each instance of juddiv3.war must point to the same database and must use the same Node ID and configuration settings. See the Database Configuration Chapter for more information.

4.14.1.2. Scaling using Subscriptions

The second mechanism is to use the Subscription API to import data and updates from a remote registry. Unfortunately, this scenario isn't quite yet supported for jUDDI, but may be in a future release.

4.14.1.3. Replication API

The third mechanism is the Replication API, which is part of the OASIS UDDIv3 specification. Since version 3.3, jUDDI provides support for synchronizating UDDI servers using the techniques described in the specification as Replication. See the Replication Services chapter for additional information,

4.14.2. Limitations of jUDDI

jUDDI's web services have no explicit upper bound on the volume of businesses and services registered. Load testing has shown that at least 10,000 are supported for each category. The

upper limit is more of a function of both the underlying database implementation and hardware (free disk space). In either case, the likelihood of hitting the limit is low for most instances. If you happen to run into scaling issues, please file a bug report at JUDDI's JIRA site at: https://issues.apache.org/jira/browse/JUDDI

Chapter 5. jUDDI Server Configuration (`juddiv3.xml`)

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

```
Since 3.2 the jUDDI server now uses an XML file for configuration. Previous
 versions uses a properties file.
```

Important

When referring to configuration *properties*, we are really referencing the XPath to specified setting.

5.1. Authentication

Table 5.1. Authentication properties that can be referenced in the juddiv3.xml file

Property Name	Description	Required	Default Value or [Example Value]
juddi/auth/ authenticator/class	The jUDDI authenticator class to use. See Chapter <add ref> of the Userguide for the choices provided.	N	*org.apache.juddi.v3.auth.JUDDIAuthentica*
juddi/auth/Inquiry	This flag determines whether authentication (the presence of a getAuthToken) is required on queries invoking the Inquiry API. By default, jUDDI sets this to false for ease of use.	N	*false*
juddi/auth/token/ Timeout	Time in minutes to expire tokes after inactivity.	N	*15*

Property Name	Description	Required	Default Value or [Example Value]
juddi/auth/token/Expiration	As of 3.1.5 Duration of time for tokens to expire, regardless of inactivity.	N	*15*
juddi/auth/token/enforceSameIPRule	As of 3.2 This setting will enable or disable the auth token check to ensure that auth tokens can only be used from the same IP address that they were issued to..	N	*true*
juddi/auth/authenticator@useAuthToken	Indicates that the authenticator is use requires a UDDI auth token. Set to false when using HTTP based authenticators	N	*true*

5.2. Startup

Table 5.2. Startup properties that can be referenced in the *juddiv3.xml* file

Property Name	Description	Required	Default Value or [Example Value]
juddi/server/baseurl	Token that can be accessed in accessPointURLs and resolved at runtime. Currently this is only used during the Installation process (seeding root data)	N	*http://localhost:8080*
juddi/root/publisher	The username for the jUDDI root publisher. This is usually just set to "root".	N	*root*
juddi/seed/always	Forces seeding of the jUDDI data. This will re-apply all files	N	*false*

Property Name	Description	Required	Default Value or [Example Value]
	with the exception of the root data files. Note that this can lead to losing data that was added to the entities that are re-seeded, since data is not merged.		
juddi/server/name	This token is referenced in the install data. Note that you can use any tokens, and that their values can be set here or as system parameters..	N	*false*
juddi/server/port	This token is referenced in the install data. Note that you can use any tokens, and that their values can be set here or as system parameters..	N	*false*
juddi/nodeId	The Node ID uniquely identifies this server. Use caution when changing the Node ID after jUDDI has been started, you may not be able to edit any existing entities! ..	N	*uddi:juddi.apache.org:node1*
juddi//load/install/data	This property allows you to cancel loading of the jUDDI install data.	N	false
juddi/locale	The default local to use. This currently is not used.	N	*en_US*

Property Name	Description	Required	Default Value or [Example Value]
juddi/operatorEmailAddress	The UDDI Operator Contact Email Address. This currently is not used.	N	*admin@juddi.org*
juddi/persistenceunit.name	The persistence name for the jUDDI database that is specified in the persistence.xml file.	N	juddiDatabase
juddi/configuration/reload/delay	The time in milliseconds in which juddiv3.xmlis polled for changes.	N	5000

Caution

Take caution in changing the jUDDI Node ID. (Updated at 3.3) jUDDI can now change Node IDs via the Admin console. However care must be taken to prevent changes to data while the rename is in progress. It is recommended to use the Admin console to change the Node ID. It will automatically update the database and the *juddiv3.xml* configuration file.

5.3. Email

As of 3.1.5, jUDDI supports Email delivery options for Subscription API functions. Email properties can be referenced in the *juddiv3.xml* file. Starting with 3.2.1, jUDDI can now send a test email via the juddiv3.war/admin console.

Property Name	Description	Required	Default Value or [Example Value]
juddi/mail/smtp/from	The Operator's Email address	Y	[*jUDDI@example.org*]
juddi/mail/smtp/host	The hostname of the SMTP server	Y	[*localhost*]
juddi/mail/smtp/port	The portname of the SMTP server	Y	[*25*]
juddi/mail/smtp/socketFactory.class	If set, specifies the name of a class that implements the *javax.net.SocketFactory*	N	

Property Name	Description	Required	Default Value or [Example Value]
	interface. This class will be used to create SMTP sockets.		
juddi/mail/smtp/ socketFactory/ fallback	If set to true, failure to create a socket using the specified socket factory class will cause the socket to be created using the *java.net.Socket* class. Defaults to true.	N	*true*
juddi/mail/smtp/ starttls/enable	f true, enables the use of the STARTTLS command (if supported by the server) to switch the connection to a TLS-protected connection before issuing any login commands. Note that an appropriate trust store must configured so that the client will trust the server's certificate. Defaults to false.	N	*false*
juddi/mail/smtp/ socketFactory/port	Specifies the port to connect to when using the specified socket factory. If not set, the default port will be used.	N	[*465*]
juddi/mail/smtp/auth	If true, attempt to authenticate the user using the AUTH command. Defaults to false.	N	[*false*]

Property Name	Description	Required	Default Value or [Example Value]
juddi/mail/smtp/user	Username used to authenticate to the SMTP server	Y, if *juddi/mail/smtp/auth* is true	[*juddi@apache.org*]
juddi/mail/smtp/password	Username used to authenticate to the SMTP server	Y, if *juddi/mail/smtp/auth* is true	[*secret*]
juddi/mail/smtp/password@encrypted	If the password is encrypted, the setting juddi/cryptor is the Cryptographic provider used to decrypt at runtime.	Y, if *juddi/mail/smtp/auth* is true	false

5.4. Query Properties

Table 5.3. Query properties that can be referenced in the *juddiv3.xml* file

Property Name	Description	Required	Default Value or [Example Value]
juddi/maxBusinessesPerPublisher	The maximum number of UDDI Businesses that can be registered per publisher. A value of *-1* indicates any number of businesses is allowed (These values can be overridden at the individual publisher level)	N	-1
juddi/maxServicesPerBusiness	The maximum number of UDDI BusinessServices allowed per Business. A value of *-1* indicates any number of artifacts is valid (These values can be # overridden at the	N	-1

Property Name	Description	Required	Default Value or [Example Value]
	individual publisher level).		
juddi/maxBindingsPerService	The maximum number of UDDI TemplateBindings allowed per BusinessService. A value of *-1* indicates any number of artifacts is valid (These values can be overridden at the individual publisher level).	N	-1
juddi/maxTModelsPerPublisher	The maximum number of TModels allowed per publisher. A value of *-1* indicates any number of artifacts is valid (These values can be overridden at the individual publisher level).	N	-1
juddi/maxInClause	The maximum number of "IN" clause parameters. Some RDMBS limit the number of parameters allowed in a SQL "IN" clause.	Y	[*1000*]
juddi/maxNameElementsAllowed	The maximum name size and maximum number of name elements allows in several of the *FindXxxx* and *SaveXxxx* UDDI functions	N	[*5*]

Property Name	Description	Required	Default Value or [Example Value]
juddi/maxNameLength	The maximum name size of name elements	N	[*255*]
juddi/maxRows	The maximum number of rows returned in a find* operation. Each call can set this independently, but this property defines a global maximum. This is related to the *maxInClause* setting (the same?).	N	1000

5.5. RMI Proxy

These properties are used to bring up RMI server socket. The settings allow for registering this service to JNDI. RMI Proxy properties that can be referenced in the *juddiv3.xml* file and is only used by RMITransport.

Property Name	Description	Required	Default Value or [Example Value]
juddi/proxy/factory/initial	JNDI Contect Facory	N	[*org.jnp.interfaces.NamingContextFactory*]
juddi/proxy/provider/url	JNDI Provider Address	N	[*jnp://localhost:1099*]
juddi/proxy/factory/url/pkg	JNDI Naming Convention	N	[*org.jboss.naming*]

5.6. Key Generation and Cryptography

Table 5.4. UDDI Key generation properties that can be referenced in the `juddiv3.xml` file.

Property Name	Description	Required	Default Value or [Example Value]
juddi/cryptor	jUDDI Cryptor implementation class that jUDDI will use to	N	*org.apache.juddi.cryptor.DefaultCryptor*

Property Name	Description	Required	Default Value or [Example Value]
	encrypt and decrypt password settings		
juddi/keygenerator	Key generator implementation that jUDDI will use to create UDDI keys if no key is passed in by the user.	N	*org.apache.juddi.keygen.KeyGenerator*
juddi/uuidgen	UUID generator implementation that jUDDI will use to create UUIDs.	N	*org.apache.juddi.uuidgen.DefaultUUIDGer*

5.7. Subscription

Table 5.5. Subscription properties that can be referenced in the *juddiv3.xml* file.

Property Name	Description	Required	Default Value or [Example Value]
juddi/subscription/ expiration/days	Days before a subscription expires	N	[*30*]
juddi/subscription/ chunkexpiration/ minutes	Minutes before a "chunked" subscription call expires	N	[*5*]
juddi/notification/ interval	Specifies the interval at which the notification timer triggers. 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. (in ms)	N	*3000000*
juddi/notification/start/ buffer	Specifies the amount of time to wait before	N	20000

Property Name	Description	Required	Default Value or [Example Value]
	the notification timer initially fires. (in ms)		
juddi/notification/ acceptableLagtime	Specifies the amount of time (in ms) from which to determine if the server is overload and to skip notifications. Notifications during this cycle will not be repeated (i.e. never be delivered). (in ms)	N	10000
juddi/notification/ maxTries	Specifies the number of times to attempt the delivery of messages to subscribers.	N	3
juddi/notification/ maxTriesResetInterval	Once the maximum delivery attempts have been made, the server will add that endpoint to an ignore list, which is reset every N ms.	N	600000
juddi/notification/ sendAuthTokenWithResult	Sends a valid authentication token for the owning user of the subscription in the subscription notification result message. Unless it is specifically needed, this is recommended to be set to false.	N	false

5.8. Custody Transfer

Table 5.6. Transfer properties that can be referenced in the *juddiv3.xml* file.

Property Name	Description	Required	Default Value or [Example Value]
juddi/transfer/ expiration/days	Days before a transfer request expires.	N	[*3*]

5.9. Validation

Table 5.7. These settings are for validating the data that users store in jUDDI. They can be referenced in the *juddiv3.xml* file.

Property Name	Description	Required	Default Value or [Example Value]
juddi/validation/ enforceReferentialIntegrity	As of 3.1.5 This setting will force referential integrity for all tModels (except keyGenerators), category bags, bindingTemplate/ AccessPoint/ hostingRedirector (referencing another host), tModelInstanceParms and anything else that references a KeyName default value is true. Set to false for backwards compatibility or for a more lax registry.	N	[*true*]
juddi/validation/ rejectInvalidSignatures/ enable	Enables or Disables the validation of signatures when a publisher attempts to save an entity	N	false

Property Name	Description	Required	Default Value or [Example Value]
juddi/validation/ rejectInvalidSignatures/ enable/trustStorePath	Path to the trust store. Can be overridden via system properties. If not specified, the Windows trust store will be used, else the default JRE trust store will be used.	N	[truststore.jks]
juddi/validation/ rejectInvalidSignatures/ trustStoreType	The type of store to use	N	JKS
juddi/validation/ rejectInvalidSignatures/ trustStorePassword	The clear text or encrypted password to the trust store	N	
juddi/validation/ rejectInvalidSignatures/ trustStorePassword@isPasswordEncrypted	True/False	N	false
juddi/validation/ rejectInvalidSignatures/ trustStorePassword@cryptoProvider	A cryptographic provider, representing the one that was used to encrypt		*juddi/validation/ rejectInvalidSignatures/ checkTimestamps*
If true, certificates are checked against the time validity	N	false	*juddi/validation/ rejectInvalidSignatures/ checkTrust*
If true, the certificates trust chain is validated against the trust store	N	false	*juddi/validation/ rejectInvalidSignatures/ checkRevocationCRL*

5.10. Logging

These properties are used to enable additional logging capabilities. Logging properties that can be referenced in the *juddiv3.xml* file.

Property Name	Description	Required	Default Value or [Example Value]
juddi/logging/ logInquirySearchPayloads	Enables request payload logging for the Inquiry Find apis	N	false

5.11. Performance

These properties are used to enable or disable certain capabilities based on performance considerations. Perofrmance properties are referenced in the *juddiv3.xm* file.

Property Name	Description	Required	Default Value or [Example Value]
juddi/performance/enableFindBusinessTModelBagFiltering	UDDI defines a tModelBag filter to filter findBusiness relates based on tModelInstanceInfo within their service's binding templates. This is an expensive operation and will cause significant performance degredation on larger registries. For spec complliance, it should be set to true. We suspect it's not a commonly used feature and recommend setting this to false.	N	true

5.12. Replication

These properties are used to tweak the replication service capabilities. These properties are referenced in the *juddiv3.xml* file.

Property Name	Description	Required	Default Value or [Example Value]
juddi/replication/getChangeRecordsMax	The maximum number of records to return from a getChangeRecord request	N	100
juddi/replication/start/buffer	Specifies the amount of time to wait before the replication timer initially fires. (in ms)	N	5000

Property Name	Description	Required	Default Value or [Example Value]
juddi/replication/interval	Specifies the interval at which the replication timer triggers (in ms).	N	5000

Deploying two or more jUDDI server on the same application server

5.13. Deploying two or more jUDDI server on the same application server

It is possible to deploy one or more jUDDI servers to the same application server. You will need copy the juddiv3.war archive (let's say you copied it to juddiv3a.war), and change the following settings to have it connect to a different database:

1. edit the `juddiv3a/META-INF/context.xml` (and `conf/Catalina/localhost/juddiv3a.xml`) to use the *jdbc/JuddiADS* datasource, and add *a* to the url: `url="jdbc:derby:target/juddi-derby-test-db-v3a;create=true"`
2. edit the *juddiv3a/WEB-INF/classes/META-INF/persistence.xml* to use `<non-jta-data-source>java:comp/env/jdbc/JuddiADS` and `persistence-unit name="juddiADatabase"`
3. edit the *juddiv3a/WEB-INF/classes/juddiv3.xml* to have `<persistenceunit><name>juddiADatabase</name></persistenceunit>`

This will create a new jUDDI server under the http://localhost:8080/juddiv3a url which connects to the juddi-derby-test-db-v3a Derby database.

5.14. jUDDI GUI Configuration

The jUDDI GUI (juddi-gui.war) has one place for configuration settings, the jUDDI Client config file.

5.15. jUDDI Client uddi.xml Settings

Defined in WEB-INF/classes/META-INF/uddi.xml, there are many settings to configure. All of these are clearly defined by the jUDDI Client Configuration Guide. The juddi-gui, uses things a bit differently, so here are the relevant parts to use. Note: this is xpath notation.

- uddi/client/nodes/properties, not used
- uddi/client/clerks, not used
- uddi/client/nodes/node, all URLs except juddiApiUrl (not used)
- uddi/client/signature, all validation related settings
- uddi/client/subscriptionCallbacks, not used

- uddi/client/XtoWsdl, not used

In addition, there a special section added just for the juddi-gui.war

Table 5.8. jUDDI GUI Configuration

Property Name	Description	Required	Default Value or [Example Value]
uddi/config/props/authtype	This controls the authentication mode to connect to a UDDI server. Most implementations of UDDI use the security service, however others use HTTP based authentication. In this case, us the value of *HTTP*, otherwise *UDDI_AUTH*	Y	*UDDI_AUTH*
uddi/config/props/enableAutomaticLogouts	This flag determines whether automatic logouts is enabled. By default, jUDDI-gui sets this to false for ease of use. (true/false)	N	*false*
udddi/config/props/enableAutomaticLogouts/duration	Time in milliseconds to force an automatic logout after inactivity.	N	*900000*
uddi/config/props/configLocalHostOnly	If false, the configuration page will be available from anywhere. If true, it will only be accessible from the server hosting juddi-gui. (true/false)	N	*true*

5.16. Encryption Keys

By default, the juddi-gui will use a randomly generated AES encryption key to help protect user credentials stored in the session object. This key is generated using the "StartupServlet" defined in

the web.xml file of juddi-gui.war/WEB-INF/web.xml and then it is stored at the path juddi-gui.war/META-INF/config.properties@key.

If the start up servlet fails to start, any authenticate operation of the juddi-gui will fail.

Important

The user account that the container for juddi-gui runs as must have write access to the file juddi-gui.war/META-INF/config.properties.

5.17. Customizing the juddi-gui

The juddi-gui has a mechanism that you can use to alter the appearance of every page. This is typically used for organizations that require legal notifications, banners or warnings on every page for one reason or another. To add your own html to every page, edit the file in

```
juddi-gui/user/banner.jsp
```

Chapter 6. Replication Services

6.1. Introduction

The UDDIv3 specification introduced a Replication API that outlines a mechansim for maintaing data ownership and data synchronization across more than one UDDI node. The replication specification has a number of facets that to the casual reader, can see overwhelmingly complex. jUDDI v3 provides support for the majority of the UDDIv3 replication API. This article will attempt to describe the in's and out's of the specification, what jUDDI supports and doesn't, finally, how to use it with your jUDDI instance(s).

6.2. UDDIv3 Replication Overview

The UDDIv3 replication API defines a number of web service methods that are used to manage and replicate UDDI data. Each node is responsible for maintaining a record of all changes made both locally and at all remote nodes. Everytime a Business, Service, Binding, tModel, or Publisher Assertion changes, all nodes are notified of the change. Once receiving the notification of the change, all nodes are then responsible to obtain the change set, apply it locally, and then retransmit (if needed and based on topology). The topology is configured via the Replication Configuration. With jUDDI, this is configured using the adminstration console.

There's one important note to remember. Each piece of data in UDDI is owned by a given node.

6.2.1. UDDIv3 Replication Topology

The specification identifies two scenarios for replication topology.

1. Non-directed Graph: All nodes can talk directly to each other and have direct communication with each other.
2. Directed Graph: Nodes can only talk to subset of the complete set of nodes.

The Non-directed graph is easier to implement and to understand. During the "notify" phase of replication, the node where the change originates, simply tell's everyone it knows about it.

In a directed graph, the node where the change originates only notifies the nodes designated nodes. This typically forms some kind of ring in which one node notifies the next and so on until the original change ends up at the origin.

6.2.2. Conflict handling

The specification defines a mechanism that is similar to a two step commit (for those familiar with database terminology). Esscentially, when a given change (typically a new record) is created, it then notifies all other nodes to put a block on the new record's keys and waits for all nodes to respond with an "OK" to commit message. This prevents the same record from being created in multiple locations. These types of messages are refered to in the specification as

NewDataConditional. As of the time of this writing jUDDI doesn't support it. When a record is created at the same at two different nodes within the same replication graph, jUDDI will simply reject the change and prevent the modifications or the transfer from happening. Records that fail to apply for one reason or another are stored in the database and can be accessed via the admin console via "Admin" and selected "getFailedReplicationChangeRecords" from the drop down menu.

6.3. Configuring your jUDDI Node for replication

Prerequisites:

1. Each node must have a unique ID associated with it.
2. Each node must have the UDDI v3 Replication service (juddiv3replication.war) deployed and configured for CLIENT-CERT authentication using SSL/TLS.
3. Each node must have a configured JKS key store and trust store.

6.3.1. Changing the Node ID

Forgot to change the Node ID before starting jUDDI for the first time? No problem. Visit the jUDDI Administration console at http://localhost:8080/juddiv3/admin, then go to the Admin page and select "Change Node Id" from the drop down menu.

6.3.2. Setting up CLIENT-CERT authentication

Since a registry can be corrupted via the replication endpoint, it is important to provide adequate security. The UDDI spec recommends using mutual certificate authentication. This is somtimes returned to as "CLIENT-CERT", certificate based authentication, or two-way SSL. All of these terms really refer to the same thing. jUDDI comes prebundled with Apache Tomcat that is configured for mutal certificate authentication out of the box (with self signed certificates). To setup CLIENT-CERT authentication, please see the documentation for your web application server.

6.3.2.1. Special notes on key stores and trust stores

jUDDI's use of key stores and trust stores for replication purposes using the standard system properties - -Djavax.net.ssl.keyStore - -Djavax.net.ssl.keyPassword - -Djavax.net.ssl.trustStore - -Djavax.net.ssl.trustStorePassword

These are used for transport layer security (node to node). On a side node, jUDDI (server) can also use the trust store to verify signed entities (configured though *juddiv3.xml*) and finally, the application server itself needs access to the key store and trust store for providing a certificate for SSL/TLS communication with clients for validating users (or another jUDDI replication node) that provide a client certificate.

For Tomcat, all you need is a connector with "clientAuth=want". Here's an example:

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
```

```
                maxThreads="150" scheme="https" secure="true"
                clientAuth="want" sslProtocol="TLS"
                truststoreFile="truststore.jks" truststorePass="password"
       keystoreFile="conf/keystore.jks" keystorePass="password"/>
```

6.3.2.2. Mapping certificates to roles

For each certificate that is used by a jUDDI node to authenticate to another, you'll have to map the Subject DN of the certificate to a user with the role "replication". In our example, we'll use tomcat's *tomcat-users.xml* file.

```
<user username="CN=localhost, OU=jUDDI Test, O=Apache Software Foundation,
 L=Anytown, ST=MD, C=US" password="null" roles="replication"/>
```

In this example, we've added our test certificate's subject DN to the role of "replication".

Tip

If you run into issues getting things working, try adding the following to the startup parameters for tomcat: -Djavax.net.debug=all

Important

Besides mapping the certificates to the replication role, either the certificate itself or the issuer of the certificate must be in the trust store used by the application server.

Since dealing with certificates can be confusing, consider the following configuration.

- Node 1 sends updates to Node 2
- Node 2 sends updates to Node 1

Then the certificates must be setup as follows (assuming that each node's SSL cert is used for authentication to the other node(s))

- Node 1's public key must be trusted by Node 2 (in Node 2 app server's trust store)
- Node 2's public key must be trusted by Node 1 (in Node 1 app server's trust store)
- Node 1 must have Node 2's certificate's Subject DN mapped to the *replication* role
- Node 2 must have Node 1's certificate's Subject DN mapped to the *replication* role
- Node 1's public and private keys must be in a keystore on Node 1 (and the Java -D properties set)

- Node 2's public and private keys must be in a keystore on Node 2 (and the Java -D properties set)

6.3.3. Setting the Replication Configuration

To set the replication configuration, you'll need to go to http://localhost:8080/juddiv3/admin then click on "Admin" in the top navigation bar and login. Once logged in, select "set_ReplicationNodes" from the drop down menu. The text entry field is actually resizable, so you'll probably want to make it bigger. This text box should be pre-populated with an example replication configuration. Edit the replication as needed, then click the "Go!" button to save it.

Note: when saving the configuration, several of the fields (time stamp, serial number) will be overwritten by the server. This is normal.

Additional notes: jUDDI doesn't currently support maximumTimeToSyncRegistry, maximumTimeToGetChanges, and controlledMessage. Due to the way the specification was written, these fields are mandatory (they must be in the Replication Configuration XML), but jUDDI wont't respect them.

6.3.3.1. Replication Configuration

When using jUDDI's Admin console to set the replication config, here's a few things to keep in mind (using xpath notation).

- replicationConfiguration/operator() - All nodes in the replication graph must be listed in the Operator section, including all directed graph nodes
- replicationConfiguration/registryContact - Must have at least one contact. If one is specified for the node's root business, then jUDDI will include that with the default config.
- replicationConfiguration/communicationGraph - Must be specified with all nodes listed as identified by the NodeID in replicationConfiguration/operator/operatorNodeID.
- replicationConfiguration/communicationGraph/controlledMessage must be specified. jUDDI uses a * to represent all messages.
- replicationConfiguration/maximumTimeToSyncRegistry isn't used and jUDDI will always set it to 1
- replicationConfiguration/maximumTimeToGetChanges - isn't used and jUDDI will always set it to 1
- replicationConfiguration/serialNumber - jUDDI will always set this to the time stamp when the configuration was last changed (time since epoch)
- replicationConfiguration/timeOfConfigurationUpdate - jUDDI will always set this to the time stamp when the configuraiton was last changed in a human readable form. The UDDI specification doesn't state what format it should be in, so we used ISO 8601 as the format.

Everytime the configuration changes, an audit log is required in jUDDI log file.

Here's an example default configuration

```
<?xml version="1.0" encoding="UTF-8"?><replicationConfiguration
 xmlns="urn:uddi-org:repl_v3" xmlns:ns2="urn:uddi-org:api_v3"
 xmlns:ns3="http://www.w3.org/2000/09/xmldsig#">
    <serialNumber>1424114880586</serialNumber>
    <timeOfConfigurationUpdate>201502161428-0500</timeOfConfigurationUpdate>
    <registryContact>
        <ns2:contact>
            <ns2:personName>unknown</ns2:personName>
        </ns2:contact>
    </registryContact>
    <operator>
        <operatorNodeID>uddi:juddi.apache.org:node1</operatorNodeID>
        <operatorStatus>normal</operatorStatus>
        <ns2:contact/>
        <soapReplicationURL>http://localhost:8080/juddiv3/services/
replication</soapReplicationURL>
    </operator>
    <communicationGraph>
        <node>uddi:juddi.apache.org:node1</node>
        <controlledMessage>*</controlledMessage>
    </communicationGraph>
    <maximumTimeToSyncRegistry>1</maximumTimeToSyncRegistry>
    <maximumTimeToGetChanges>1</maximumTimeToGetChanges>
</replicationConfiguration>
```

Here's an example non-directed replicaton graph. In this example, all changes perform on all nodes get set to all the other nodes.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<replicationConfiguration xmlns="urn:uddi-org:repl_v3" xmlns:ns2="urn:uddi-
org:api_v3" xmlns:ns3="http://www.w3.org/2000/09/xmldsig#">
    <serialNumber>0</serialNumber>
    <timeOfConfigurationUpdate></timeOfConfigurationUpdate>
    <registryContact>
        <ns2:contact>
            <ns2:personName>unknown</ns2:personName>
        </ns2:contact>
    </registryContact>
    <operator>
        <operatorNodeID>uddi:juddi.apache.org:node1</operatorNodeID>
        <operatorStatus>normal</operatorStatus>
        <ns2:contact useType="admin">
            <ns2:personName xml:lang="en">bob</ns2:personName>
        </ns2:contact>
        <soapReplicationURL>https://localhost:8443/juddiv3replication/
services/replication</soapReplicationURL>
    </operator>
```

```
    <operator>
        <operatorNodeID>uddi:another.juddi.apache.org:node2</operatorNodeID>
        <operatorStatus>normal</operatorStatus>
        <ns2:contact useType="admin">
            <ns2:personName xml:lang="en">mary</ns2:personName>
        </ns2:contact>
        <soapReplicationURL>https://localhost:9443/juddiv3replication/
services/replication</soapReplicationURL>
    </operator>
    <communicationGraph>
        <node>uddi:juddi.apache.org:node1</node>
        <node>uddi:another.juddi.apache.org:node2</node>
        <controlledMessage>*</controlledMessage>
    </communicationGraph>
    <maximumTimeToSyncRegistry>1</maximumTimeToSyncRegistry>
    <maximumTimeToGetChanges>1</maximumTimeToGetChanges>
</replicationConfiguration>
```

In this example, we have a directed graph where Node 1 sends to Node2, Node 2 to Node 3, and Node 3 to Node 1. Note the addition of the replicationConfiguration/communicationGraph/edge() that defines this interaction pattern. Again all nodes defined in edges must also be defined both in the communicationGraph and as operator() XML elements.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<replicationConfiguration xmlns="urn:uddi-org:repl_v3" xmlns:ns2="urn:uddi-
org:api_v3" xmlns:ns3="http://www.w3.org/2000/09/xmldsig#">
    <serialNumber>0</serialNumber>
    <timeOfConfigurationUpdate></timeOfConfigurationUpdate>
    <registryContact>
        <ns2:contact>
            <ns2:personName>unknown</ns2:personName>
        </ns2:contact>
    </registryContact>
    <operator>
        <operatorNodeID>uddi:juddi.apache.org:node1</operatorNodeID>
        <operatorStatus>normal</operatorStatus>
        <ns2:contact useType="admin">
            <ns2:personName xml:lang="en">bob</ns2:personName>
        </ns2:contact>
        <soapReplicationURL>https://localhost:8443/juddiv3replication/
services/replication</soapReplicationURL>
    </operator>
    <operator>
        <operatorNodeID>uddi:another.juddi.apache.org:node2</operatorNodeID>
        <operatorStatus>normal</operatorStatus>
        <ns2:contact useType="admin">
            <ns2:personName xml:lang="en">mary</ns2:personName>
        </ns2:contact>
```

```
        <soapReplicationURL>https://localhost:9443/juddiv3replication/
services/replication</soapReplicationURL>
    </operator>
    <operator>
        <operatorNodeID>uddi:yet.another.juddi.apache.org:node3</
operatorNodeID>
        <operatorStatus>normal</operatorStatus>
        <ns2:contact useType="admin">
            <ns2:personName xml:lang="en">mary</ns2:personName>
        </ns2:contact>
        <soapReplicationURL>https://localhost:10443/juddiv3replication/
services/replication</soapReplicationURL>
    </operator>
    <communicationGraph>
        <node>uddi:another.juddi.apache.org:node2</node>
        <node>uddi:juddi.apache.org:node1</node>
        <node>uddi:yet.another.juddi.apache.org:node3</node>
        <edge>
            <messageSender>uddi:juddi.apache.org:node1</messageSender>
            <messageReceiver>uddi:another.juddi.apache.org:node2</
messageReceiver>
        </edge>
        <edge>
            <messageSender>uddi:another.juddi.apache.org:node2</
messageSender>
            <messageReceiver>uddi:yet.another.juddi.apache.org:node3</
messageReceiver>
        </edge>
        <edge>
            <messageSender>uddi:yet.another.juddi.apache.org:node3</
messageSender>
            <messageReceiver>uddi:juddi.apache.org:node1</messageReceiver>
        </edge>
    </communicationGraph>
    <maximumTimeToSyncRegistry>1</maximumTimeToSyncRegistry>
    <maximumTimeToGetChanges>1</maximumTimeToGetChanges>
</replicationConfiguration>
```

One last point of interest, Edge's can have a list of alternate message receivers and it is supported by jUDDI.

6.3.4. Performing Custody Transfer between nodes

Custody transfer (from a user's perspective) happens exacty the same way as it would to transfer between two users on the same node. The only change is that the Replication API plays a signficant role in this process and is thus a requirement.

6.3.5. What's Supported and What's Not

Here's a quick summary of what is and isn't supported for jUDDI replication capabilities. Want more support? Open a ticket and contribute.

Supported:

- Directed graph replication with retransmit (primary and alternate message receivers)
- Non-directed graphic replication (no edges defined)
- All UDDI data is replicated (Business, Binding, Serivce, tModels and Publisher Assertions)
- Custody transfer from Node to Node within the replication graph.

Functions not supported:

- Conditional Data Updates
- Configuration Settings:
- maximumTimeToSyncRegistry
- maximumTimeToGetChanges
- OperatorStatus - Node Status (New, Normal, Resigned)
- Controlled Messages (all messages are sent to all nodes)

Chapter 7. UDDI Seed Data

This information is relevant for both understanding how jUDDI's default data is set when jUDDI first runs (i.e. to a new database). It's also useful for scripting or automating the deployment of a jUDDI server within your organization which will enable you to prepopulate the data.

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.

7.1. 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></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></publisher>_tModels.xml file. Finally, in the <publisher></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">${juddi.server.baseurl}/juddiv3</
discoveryURL>
  </discoveryURLs>
  <categoryBag>
    <keyedReference tModelKey="uddi:uddi.org:categorization:nodes"
 keyValue="node" />
  </categoryBag>
  <businessServices>
    <!-- As mentioned above, you may want to provide user-defined keys for
 these (and the services/bindingTemplates below.  Services that you
    don't intend to support should be removed entirely -->
    <businessService serviceKey="uddi:juddi.apache.org:services-inquiry"
 businessKey="uddi:juddi.apache.org:businesses-asf">
      <name xml:lang="en">UDDI Inquiry Service</name>
      <description xml:lang="en">Web Service supporting UDDI Inquiry API</
description>
      <bindingTemplates>
        <bindingTemplate bindingKey="uddi:juddi.apache.org:servicebindings-
inquiry-ws" serviceKey="uddi:juddi.apache.org:services-inquiry">
          <description>UDDI Inquiry API V3</description>
          <!-- This should be changed to the WSDL URL of the inquiry API.
 An access point inside a bindingTemplate will be found for every service
          in this file.  They all must point to their API's WSDL URL -->
```

```
          <accessPoint useType="wsdlDeployment">${juddi.server.baseurl}/
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>
<!-- snip -->
</businessService>
```

Note that the seeding process only kicks off if no publishers exist in the database. So this will only work with a clean database, unless you set juddi/seed/always to true. Then it will re-apply all files with the exception of the root data files. Note that this can lead to losing data that was added to entities that are re-seeded, since data is not merged.

7.2. Tokens in the Seed Data

You may have noticed the tokens in the root_BusinessEntity.xml file (${juddi.server.baseurl}. The value of this tokens can set in the juddiv3.xml file. The value substitution takes place at runtime, so that different nodes can do the substitution with their own value if needed.

7.3. 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.xml 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

```
*nix
mv RENAME4Sales_juddi_custom_install_data juddi_custom_install_data
Win*
ren 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.xml file.

Chapter 8. How to deploy jUDDI To?

The jUDDI distribution ships preconfigured on Tomcat - it runs out of the box. All you have to do in go into the `juddi-distro-<version>/juddi-tomcat-<version>/bin` directory and start up Tomcat. All of this just as described in Chapter 2, *Getting Started*.

By default the `juddiv3.war` is configured to use OpenJPA and CXF. If you want to change your JPA or WS provider, or you'd like to run on a different container then this chapter may come in handy, as there a number of scripted *profiles* to change the configuration and dependencies in the `juddiv3.war`. To run these maven based scripts you need to go into `juddi-distro-<version>/juddiv3-war` directory.

8.1. Tomcat

8.1.1. OpenJPA and CXF

Target platform Tomcat and Derby using OpenJPA and CXF. Both OpenJPA and CXF are packaged up in the juddiv3.war.

```
mvn clean package -P openjpa
```

Then copy the `target/juddiv3.war` to the `<tomcat>/webapps` directory.

8.1.2. Hibernate and CXF

Target platform Tomcat and Derby using Hibernate and CXF. Both Hibernate and CXF are packaged up in the juddiv3.war.

```
mvn clean package -P hibernate
```

Then copy the `target/juddiv3.war` to the `<tomcat>/webapps` directory.

8.1.3. OpenJPA and Axis2

Target platform Tomcat and Derby using OpenJPA and Apache Axis2. Both Hibernate and Axis2 are packaged up in the juddiv3.war.

```
mvn clean package -P axis2
```

Then copy the `target/juddiv3.war` to the `<tomcat>/webapps` directory.

8.2. JBoss

8.2.1. JBossAS 6.0.0.GA

This section describes how to deploy juddi to JBoss 6.0.0.GA.

First, download jboss-6.0.0.GA - the zip or tar.gz bundle may be found at http://www.jboss.org/jbossas/downloads/. Download the bundle and uncompress it.

8.2.1.1. Hibernate and JBossWS-Native

Target platform JBoss-6.x and HSQL using Hibernate and JBossWS-native. The juddiv3.war relies on Hibernate and JBossWS-native in the appserver.

```
mvn clean package -P hibernate-jbossws-native
```

Then copy the `target/juddiv3.war` to the `<jboss>/server/default/deploy` directory.

8.2.1.2. Hibernate and JBossWS-CXF

Target platform JBoss-6.x and HSQL using Hibernate and JBossWS-cxf. The juddiv3.war relies on Hibernate and JBossWS-cxf in the appserver.

```
mvn clean package -P hibernate-jbossws-cxf
```

KNOWN ISSUES

Issue 1

```
15:14:37,275 SEVERE [RegistryServlet] jUDDI registry could not be
 started. org.apache.commons.configuration.ConfigurationException:
 java.util.zip.ZipException: error in opening zip file:
 org.apache.commons.configuration.ConfigurationException:
 org.apache.commons.configuration.ConfigurationException:
 java.util.zip.ZipException: error in opening zip file
```

Workaround: deploy juddiv3.war as a directory (not a zip file).

Issue 2

JBoss-5.x Note that configuration 3 and 4 will also run on JBoss-5.x, but you may run into the following

```
ERROR [org.jboss.ws.metadata.wsdl.xmlschema.JBossXSErrorHandler]
 (main) [domain:http://www.w3.org/TR/xml-schema-1]::[key=src-
resolve]::Message=src-resolve: Cannot resolve the name ns1:Signature to a
 element declaration component.
```

Workaround: Unzip the deployers/jbossws.deployer/jbossws-native-core.jar and add the xmldsig-core-schema.xsd in the schema directory,

```
10293 Fri May 27 14:40:40 EDT 2011 schema/xmldsig-core-schema.xsd
```

Edit the file META-INF/jbossws-entities.properties by adding a line at the bottom saying:

```
http\://www.w3.org/2000/09/xmldsig#=schema/xmldsig-core-schema.xsd
```

Copy juddiv3.war to server/default/deploy and unpack it.

Insert jboss-web.xml into the juddiv3.war/WEB-INF directory , should look like the following :

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE jboss-web PUBLIC
 "-//JBoss//DTD Web Application 2.3V2//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-web_3_2.dtd">

<jboss-web>

    <resource-ref>
        <res-ref-name>jdbc/JuddiDS</res-ref-name>
        <jndi-name>java:JuddiDS</jndi-name>
    </resource-ref>
    <depends>jboss.jdbc:datasource=JuddiDS,service=metadata</depends>

</jboss-web>
```

8.2.1.3. Change web.xml

Replace the WEB-INF/web.xml with the jbossws-native-web.xml within docs/examples/appserver.

8.2.1.4. Configure Datasource

The first step for configuring a datasource is to copy your JDBC driver into the classpath. Copy your JDBC driver into ${jboss.home.dir}/server/${configuration}/lib, where configuration is the profile you wish to start with (default, all, etc.). Example :

```
cp mysql-connector-java-5.0.8-bin.jar /opt/jboss-5.1.0.GA/server/default/lib
```

Next, configure a JBoss datasource file for your db. Listed below is an example datasource for MySQL :

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
 <local-tx-datasource>
```

```
    <jndi-name>JuddiDS</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/juddiv3</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>root</user-name>
    <password></password>
    <exception-sorter-class-
name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-
sorter-class-name>

    <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml
 (optional) -->
    <metadata>
       <type-mapping>mySQL</type-mapping>
    </metadata>
 </local-tx-datasource>
</datasources>
```

Next, make a few changes to the juddiv3.war/classes/META-INF/persistence.xml. Change the "hibernate.dialect" property to match the database you have chosen for persistence. For MySQL, change the value of hibernate.dialect to "org.hibernate.dialect.MySQLDialect". A full list of dialects available can be found in the hibernate documentation (https://www.hibernate.org/hib_docs/v3/api/org/hibernate/dialect/package-summary.html). Next, change the <jta-data-source> tags so that it reads <non-jta-data-source>, and change the value from java:comp/env/jdbc/JuddiDS to java:/JuddiDS.

8.2.2. JBossAS 7.x/JBossEAP-6.x

This section describes how to deploy juddi to JBossAS 7, WildFly and JBossEAP 6

8.2.2.1. Hibernate and JBossWS-CXF

Target platform Wildfly/EAP and H2 using Hibernate and JBossWS-cxf. The juddiv3.war relies on Hibernate and JBossWS-cxf modules in the appserver. To build the correct juddiv3.war run

```
mvn clean package -P jboss7up
```

Use the JBoss add-user.sh script to create an application user with the uddiadmin role.

8.3. Deploying to Glassfish

This section describes how to deploy juddi to Glassfish 2.1.1. These instructions will use CXF as a webservice framework.

First, download the glassfish-v2.1.1 installer JAR. Once downloaded,install using the JAR and then run the ant setup script :

```
java -jar glassfish-installer-v2.1.1-b31g-linux.jar
```

```
cd glassfish
ant -f setup.xml
```

8.3.1. Glassfish jars

Copy the following JARs into domains/domain1/lib/ext. Note that for the purposes of this example, we have copied the MySQL driver to domains/domain1/lib/ext :

```
antlr-2.7.6.jar
cglib-nodep-2.1_3.jar
commons-collections-3.2.1.jar
commons-logging-1.1.jar
dom4j-1.6.1.jar
hibernate-3.2.5.ga.jar
hibernate-annotations-3.3.0.ga.jar
hibernate-commons-annotations-3.0.0.ga.jar
hibernate-entitymanager-3.3.1.ga.jar
hibernate-validator-3.0.0.ga.jar
javassist-3.3.ga.jar
jboss-common-core-2.0.4.GA.jar
jta-1.0.1B.jar
mysql-connector-java-5.0.8-bin.jar
persistence-api-1.0.jar
```

8.3.2. Configure the JUDDI datasource

First, using the asadmin administration tool, import the following file :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//Sun Microsystems Inc.//DTD Application Server
 9.0 Domain//EN" "*<install directory>/lib/dtds/sun-resources_1_3.dtd*">
<resources>
<jdbc-connection-pool name="mysql-pool" datasource-
classname="com.mysql.jdbc.jdbc2.optional.MysqlDataSource" res-
type="javax.sql.DataSource">
<property name="user" value="juddi"/>
<property name="password" value="juddi"/>
<property name="url" value="jdbc:mysql://localhost:3306/juddiv3"/>
</jdbc-connection-pool>
<jdbc-resource enabled="true" jndi-name="jdbc/mysql-resource" object-
type="user" pool-name="mysql-pool"/>
</resources>
```

```
asadmin add-resources resource.xml
```

Then use the Glassfish administration console to create a "jdbc/juddiDB" JDBC datasource resource based on the mysql-pool Connection Pool.

8.3.3. Add juddiv3-cxf.war

Unzip the juddiv3-cxf WAR into domains/domain1/autodeploy/juddiv3.war .

Add a sun-web.xml file into juddiv3.war/WEB-INF. Make sure that the JNDI references matches the JNDI location you configured in the Glassfish administration console.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-web-app PUBLIC '-//Sun Microsystems, Inc.//DTD
Application Server 9.0 Servlet 2.5//EN'
'http://www.sun.com/software/appserver/dtds/sun-web-app_2_5-0.dtd'>
<sun-web-app>
<resource-ref>
<res-ref-name>jdbc/juddiDB</res-ref-name>
<jndi-name>jdbc/juddiDB</jndi-name>
</resource-ref>
</sun-web-app>
```

Next, make a few changes to juddiv3.war/WEB-INF/classes/META-INF/persistence.xml . Change the "hibernate.dialect" property to match the database that you have chosen for persistence. For MySQL, change the value of hibernate.dialect to "org.hibernate.dialect.MySQLDialect". A full list of dialects available can be found in the hibernate documentation (https://www.hibernate.org/hib_docs/v3/api/org/hibernate/dialect/package-summary.html). Next, change the <jta-data-source> change the value from java:comp/env/jdbc/JuddiDS to java:comp/env/jdbc/JuddiDB.

8.3.4. Run jUDDI

Start up the server :

```
cd bin
asadmin start-domain domain1
```

Once the server is deployed, browse to http://localhost:8080/juddiv3

Chapter 9. Extending UDDI

jUDDI has extensively uses the Interface/Factory pattern to enable configuration runtime options and to provide you, the developer easy insertion points to customize the behavior of jUDDI. The remaining sections of this chapter outline the different technology insertion points.

9.1. Authentication modules

Authentication modules are used when the UDDI's AuthToken is utilized on the Security web service. It's function is to point to some kind of user credential store to validate users. See the User Guide for details on what's available out of the box.

All of the provided classes implement the interface .org.apache.juddi.v3.auth.Authenticator.. So, if you wanted something a bit more functional than what's provided out of the box. you'll need to implement your own Authenticator. To wire it in, edit the juddiv3.xml file, specifying your class name as the value to the property "juddi/auth/authenticator/class" and then add the class or jar containing your implementation to juddiv3.war/WEB-INF/classes or judiv3.war/WEB-INF/lib respectively.

9.2. Subscription Notification Handlers

Subscription Notification Handlers are used to asynchronously notify users that something has changed in UDDI. In order to do this, a UDDI Subscription is created that references a specific Binding Template key which represents the service that will be called whens something changes. jUDDI comes with support for Email delivery and the UDDI Subscription Listener Web Service (HTTP) delivery. In addition, jUDDI comes with an example for publishing to an Apache Qpid AMQP pub/sub server, which can be used to further disseminate the change. The following is an exert from the jUDDI Blog posting on this.

1. Make a new Java library projects in your IDE of choice. Reference the juddi-core, and uddi-ws projects or JAR files or the Maven dependency equivalent
2. Create a class of your own within the following package name: org.apache.juddi.subscription.notify
3. The class name MUST follow this pattern: PROTOCOLNotifier Where PROTOCOL is the prefix of whatever URL you want users to be able to use. Here's an example using Apache Qpid. Example URL: amqp://guest:guest@client1/development?brokerlist=*tcp://localhost:5672* Class Name: AMQPNotifier. The Notification class basically takes the protocol of the Access Point's value, splits it on the character ":" and then grabs the first token "amqp" and converts to upper case. Using this pattern you should be able to insert anything you want.
4. Our new shinny class, AMQPNotifier, must implement the interface org.apache.juddi.subscription.notify.Notifier. From there, all you need to do is to add in the jars necessary for your transport mechanism and wire in your own code.

5. Deployment is simple. Add your PROTOCOL Notifier jar and its dependencies to the juddiv3.war/WEB-INF/lib folder.

Note: be careful and watch for conflicting jar file versions. In general, usually moving up a version is ok, but moving down may cause the services to fail unexpectedly.

To test, create a Service with the BindingTemplate's Access Point's value equal to whatever you need. Next, setup a subscription and reference the BindingTemplate key that represents your call back handler's end point. Finally, change an item that is covered by the subscription's filter and monitor the log files. Hopefully, you won't see an unexpected errors.

9.3. KeyedReference Value Set Validation Services

Since jUDDI 3.2.1, we now have support for the Value Set Validation Service. This allows you to define a validator that will check when a user saves a UDDI entity that references a given tModel that contains a keyed reference to uddi:uddi.org:identifier:validatedby (which points to the VSV service).

To defined your own validator, use the following steps # Create you tModel with a named key # Implement the org.apache.juddi.validation.vsv.ValueSetValidator interface # Name your implementation class using the naming schema defined in the ConvertKeyToClass function of UDDIValueSetValidationImpl (first letter is upper, all else is lower. Numbers and letters only. Class must be in the package org.apache.juddi.validation.vsv # Update your saved tModel and add a keyed reference for uddi:uddi.org:identifier:validatedby using the value of uddi:juddi.apache.org:servicebindings-valueset-cp # Get your class in the class path of jUDDI and give it a shot

9.4. Cryptographic Providers

jUDDI provides cryptographic functions via (Java) juddi-client.jar/ org.apache.juddi.v3.client.cryptor and implement the Cryptor interface which provides two simple functions, encrypt and decrypt. (Note: .NET has similar functionality).

9.5. jUDDI Client Transport

The juddi-client's Transport class is an abstract class that you can you alter the transport mechanism used by jUDDI's client APIs. Included is what would be used in most cases, such as JAXWS, RMI, and InVM (Embedded mode). This can be extended to use whatever you may need.

Chapter 10. Digital Signatures

Users of UDDI can use digital signatures to ensure that no unauthorized users alter the content of UDDI. We're sure that one of the first questions one would ask is "can't access control rules handle this problem for us?" The answer is yes, however it does not mitigate the risk of a number of opportunities for electronic attack.

10.1. Requirements

UDDI supports both the XML Digital Signature specification, which effectively means that you can use PGP Keys and X509 certificates. jUDDI provides out of the box support for X509 certificates and the Public Key Infrastructure (PKI). If you require direct PGP signing support, please open a JIRA ticket.

10.2. Using Digital Signatures using the jUDDI GUI

Please see ???.

10.3. Frequently Asked Questions

Doesn't UDDI access control rules prevent alteration of the content?

Yes, however it does not mitigate the man in the middle attack vectors. Since UDDI is used to determine the location of the thing you want, it's possible that falsified endpoints can be interjected in transport. The target service requires authentication, then the end user's credentials could be compromised without their knowledge.

How can I sign a business, service, tModel or binding?

Use the juddi-gui's digital signature applet by first located the item in the juddi-gui interface, then click on the "Sign" button. You need write access to the entity.

The digital signature applet doesn't run. Now what?

The applet requires the Java browser plugin. Unfortunately, due to recent (2013) security vulnerabilities, many places of business have heeded Oracle's advice and have disabled the browser plugin. There are other options, however.

What other tools can I use to sign a UDDI entity?

TBD

What is a signature?

It's basically a cryptographic (a fancy math equation) using a set a keys (one is public and everyone can see/know it, the other only is held by the owner) that proves that the owner signed a piece of data.

How is a signature verified?

There's a few ways, we can prove mathematically that the signature is valid (the content hasn't been modified). From there we can also verify that the signing key is valid.

How do we know the signing key is valid?

Most certificates (key pairs) have some kind of mechanism in it to verify if the certificate has been revoked. If your certificate has it, it will be labeled with something like OCSP or CRL. Both of these are supported in both .NET and Java juddi-clients as well as via the juddi-gui.

Chapter 11. Troubleshooting jUDDI

Here are some tips to help you troubleshoot problems with jUDDI, jUDDI-GUI, jUDDI Client and more.

11.1. jUDDI Web Services, juddiv3.war

11.1.1. Enable debugging logging

You can adjust the logging level to provide additional output for troubleshooting purposes. To do so, see the Administration Guide, Logging.

11.2. jUDDI GUI, juddi-gui.war

Problem: Can't authentication from juddi-gui's top right hand side login box to juddiv3.war services Solutions:

- Check the server's log files for both juddi, juddi-gui and the server itself for error messages. This can sometimes be caused by the lack of Java Crypto Extensions (Oracle/Sun JRE/JDK only).
- Check juddi-gui's configuration page at http://localhost:8080/juddi-gui/settings.jsp, confirm that the URL's that are referenced for the UDDI services are correct and accessible from the server hosting juddi-gui.
- Make sure you're using a valid username/password ;)
- Increase the logging level of jUDDI by changing the commons-logging.properties file
- If you're having problems with Email delivery of subscription updates, enable debug logging by setting config/uddi/mail/debug=true in juddiv3.xml

11.3. jUDDI Client Java

11.3.1. Enable debugging logging

You can adjust the logging level to provide additional output for troubleshooting purposes. To do so, see the Administration Guide, Logging.

11.4. jUDDI Client .NET

Components based on jUDDI's Client for the .NET Framework can configure logging from their application's config file. This is usually app.config or web.config. To configure logging, the following three settings must appear in the configuration/appSetttings section.

```
<!-- DEBUG, INFO, WARN, ERROR -->
```

```
<add key="org.apache.juddi.v3.client.log.level" value="INFO" />
   <!-- options are CONSOLE, EVENTLOG, FILE multiple values can be
specified, comma delimited.
 Notes for EVENTLOG, you must run the juddi-installer as admin before
running-->
   <add key="org.apache.juddi.v3.client.log.target" value="CONSOLE" />
<!-- only used when target=FILE -->
   <add key="org.apache.juddi.v3.client.log.logger.file"
value="pathToOutputFile" />
```

If nothing is defined, the default log level is "WARN" and the target is "CONSOLE" which is standard out.

11.5. Getting help

There are many different ways to get help with your jUDDI instance. Please refer to the following URLs for more information.

- jUDDI Home Page http://juddi.apache.org/
- User Guide http://juddi.apache.org/docs/3.x/userguide/html/index.htmlnavbar.help.userguide
- Developer Guide http://juddi.apache.org/docs/3.x/devguide/html/index.htmlnavbar.help.devguide
- Developer API Documentation http://juddi.apache.org/docs.html
- jUDDI Wiki http://wiki.apache.org/juddi
- jUDDI Issue/Bug Tracker http://juddi.apache.org/issue-tracking.html
- jUDDI User and Developer Mailing List http://juddi.apache.org/mailing-list.html
- jUDDI Source Code http://svn.apache.org/viewvc/juddi/

Chapter 12. Contributing to jUDDI

We welcome contributions to jUDDI. Visit the jUDDI web set at http://juddi.apache.org for more information.

12.1. License guidance

Apache jUDDI is released under the Apache Software Foundation v2.0 License. Details on the license is located at the following link: http://apache.org/licenses/LICENSE-2.0.

If you wish to bring in 3rd libraries, please keep in mind that certain libraries cannot be used due to license restrictions. See http://www.apache.org/legal/3party.html for details.

12.2. SVN access

Source code is accessible at the following link: https://svn.apache.org/viewvc/juddi/trunk/.

12.3. Project structure

jUDDI, from a developer's perspective, is divided into a number of smaller, more manageable modules. In general, each module contains all of the necessary unit tests in order to ensure functionality.

12.4. Building and testing jUDDI

jUDDI has a number of components, however it is mostly Java based. The following sections describe the particulars for each language.

12.4.1. All Java Components

Procedure

1. Acquire a Subversion client.
2. Execute svn co https://svn.apache.org/viewvc/juddi/trunk/
3. Acquire a JDK5 or higher and setup the JAVA_HOME environment variable.
4. Acquire Apache Maven. Known working version: 3.0.4
5. Setup an environment variable, MAVEN_OPTS=-Xmx768m -XX:MaxPermSize=512m
6. Make sure the Maven/bin folder and the JDK/bin folders are in the current path
7. Execute "mvn clean install"

This will build, test and package all of the Java components of jUDDI. This includes the Technical Conformance Kit (TCK), a live Tomcat server, the user interfaces, and more.

For additional build output, add -Ddebug-true for Java.

To prepare a deployable jUDDI war for an alternate deployment scenario (other than Tomcat with CXF and OpenJPA), use the following procedure:

1. Execute "mvn clean package -P<packageName>"

Where <packageName> is one of the following

1. openjpa-jboss7up for EAP 6 and up, GA 7 and up
2. hibernate-jbossws-native for EAP 5, Jboss GA 6 and down with the JbossWS Native soap stack
3. hibernate-jbossws-cxf for EAP 5, Jboss GA 6 and down with the JbossWS Native soap stack
4. hibernate (includes CXF in the war, used for Tomcat)
5. openjpa (includes CXF in the war, used for Tomcat)
6. axis2 (includes Axis2 in the war)

Tip

When altering the TCK based modules, make sure you clean install in the root check out location. Due to the build order, you may end up with strange results when just executing the tests, even with clean install.

Tip

To attach the debugger to the build process try "mvn -Dmaven.surefire.debug clean install". It listens on port 5005 by default. More info on debugging maven projects is here http://maven.apache.org/surefire/maven-surefire-plugin/examples/debugging.html

12.4.2. .NET

jUDDI also has a .NET based jUDDI Client. To build this, only the .NET Framework needs to be installed, version 3.5 or higher. A Visual Studio solution file is included, but it is not required for building.

Procedure - Windows * Add MSBuild.exe to your system path. It’s usually in %SYSTEMROOT %\Microsoft.NET\Framework(64)\v4.x.x. If you haven’t installed .NET 4 yet, replace v4.x.x with

v2.x.x * Build the solution. This will build the juddi-client.net.dll, the same application(s) and the test project(s).

```
MSBuild.exe juddi-client.net.sln /p:Configuration=Debug /p:Platform="Any
 CPU"
```

For additional debug output set the environment variable *debug=true*

```
set debug=true
```

Procedure - *nix using Mono

Tip

Support on Mono is very experimental. There are still many APIs that have no yet been implements on Mono that may cause compilation failure.

To build the .NET assemblies on a Linux or Unix based computer: * Install Mono (apt-get install mono-complete mono-develop * Build it

```
cd juddi-client.net
xbuild judddi-client.net-mono.sln
cd juddi-client.net-sample/bin/Debug/
mono juddi-client.net-sample.exe
```

12.5. Other ways to Contribute to jUDDI

There are many ways you can contribute to jUDDI. We welcome all kinds and types contributions.

12.5.1. Bug Reports

Bug reports and feature requests are low effort tasks that do not require a high level of technical proficiency.

12.5.2. Internationalization

The jUDDI GUI user interface is designed to be multi-lingual. For the 3.2 release, English and Spanish are provided for the user interface. The jUDDI server administration user interface is also available in English and Spanish.

12.5.3. Contributing Source code

When contributing source code, you must own the code and be will to donate the code to the Apache Software Foundation. For those without SVN access, the process is as follows: . Open a

JIRA on the jUDDI Issue Tracker . Write your code and test it (mvn clean install) . Use Subversion to create a patch (svn patch) . Upload the patch as an attachment for the JIRA

Once accepted, your code will be added to the baseline. Code submissions may be modified for style, content, documentation and any other reason that we see fit.

12.5.3.1. Coding Standards

The majority of jUDDI's source code is formatted using 8 space tabs and using Javadoc style documentation. In general, test cases are often more useful and more valuable that the code being tested.

12.5.4. Releases

For the latest information on jUDDI's release process, visit http://juddi.apache.org/committers.html

12.6. What the?

Having ran into a number of strange issues when developing with jUDDI, we decided to write a few of them down.

1. I added a new class to juddi-core but it doesn't end up in the packaged tomcat instance? A: Modify the pom and make sure the package name is added to juddi-core-openjpa
2. Some unit tests fail, but only under windows. A: This is specifically for the SubscriptionListerner Tests and most likely has something to do with ports getting locked up by the Java process.

Bibliography

Books

[graham-Davis-et-all] Steve Graham, Doug Davis et all. *Building Web Services with Java - Making sense of XML, SOAP, WSDL, and UDDI*. Second Edition. Sams Publishing. 2005. ISBN 0-672-32641-8.

[stam-oree] Kurt Stam, Alex O'Ree et all. *jUDDI Client and GUI Guide*. 2014.

Articles

[uddi-v3] OASIS. *https://www.oasis-open.org/standards#uddiv3.0.2*. 2003.

WebSites

[uddi-xml-org] UDDI XML.org Editorial Board. *http://uddi.xml.org/*. 2014.

[uddi-oasis-open-org] OASIS. *https://www.oasis-open.org/standards#uddiv3.0.2*. 2003.

Index