Apache jUDDI Client and GUI Guide

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

Apache jUDDI Client and GUI 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 vii
1. Simple Publishing Using the jUDDI API 1
1.1. UDDI Data Model 1
1.2. jUDDI Additions to the Model 2
1.3. UDDI and jUDDI API 2
1.4. Getting Started 3
1.4.1. Simple Publishing Example 3
1.4.2. About UDDI Entity Keys 8
1.5. A few tips on adding Binding Templates 9
1.6. Conclusion 9
2. jUDDI Client Configuration Guide 11
2.1. Introduction 11
2.2. Client Settings 11
2.3. Nodes 11
2.3.1. Transport Options 12
2.4. Clerks 12
2.5. Clerk 12
2.6. Digital Signatures 13
2.7. Subscription Callbacks 14
2.8. XtoWsdl 14
2.9. Embedded jUDDI server 14
2.9.1. Requirements 14
2.9.2. Changes in configuration compared to non-embedded 14
3. Key Format Templates 17
3.1. UDDIv3 key format 17
3.2. jUDDI key format templates 17
3.2.1. Advantages of using a template 17
3.2.2. Default UDDIKeyConvention Key Templates 17
3.2.3. How to use the templates? 17
3.2.4. Where to define to properties? 18
4. Using the jUDDI GUI 19
4.1. Requirements 19
4.2. Tasks 19
4.2.1. Your first sign on 19
4.3. The Menu Bar 23
4.4. Logging in to UDDI Services 23
4.5. Logging Out 24
4.6. Discover (Browse UDDI) 24
4.6.1. Business Browser 24
4.6.2. Service Browser 26
4.6.3. tModel Browser 27
4.6.4. Search 28
4.7. Creating new Entities 30
4.7.1. Create a tModel 30

4.7.2. Create a tModek Key Generator (Partition) 30
4.7.3. Create a Business 31
4.7.4. Create a Service 33
4.7.5. Import from WSDL or WADL 35
4.8. Custody Transfers 36
4.9. Publisher Assertions 36
4.10. Subscriptions 37
4.10.1. Create a new subscription 37
4.10.2. View My Subscriptions 41
4.10.3. View the News Feed 42
4.11. Using Digital Signatures in juddi-gui 42
4.11.1. Sign a Business, Service or tModel 42
4.11.2. Verify a signed UDDI entity 44
4.12. Configuration 46
4.13. Language Codes 46
4.14. Switching Nodes 47
4.15. Adding Additional Language Translations 48
5. Mapping WSDL and WSDL to UDDI 51
5.1. Introduction 51
5.2. Use Case - WSDL 51
5.2.1. Sample Code 51
5.2.2. Links to sample project 52
5.3. Use Case - WADL 52
5.3.1. Sample Code 52
5.3.2. Links to sample project 53
6. Using UDDI Annotations 55
6.1. UDDI Service Annotation 55
6.2. UDDIServiceBinding Annotation 56
6.2.1. Java Web Service Example 56
6.2.2. Wiring it all together 57
6.3. .NET Web Service Example 58
6.3.1. Wiring it all together 58
6.4. CategoryBag Attribute 58
6.5. Considerations for clustered or load balanced web servers and automated registration 59
7. Using the UDDI v2 Services and Adapters 61
7.1. Introduction 61
7.2. Accessing UDDI v2 services using the jUDDI v3 Client 61
7.3. Accessing UDDI v2 services using UDDI v2 APIs 61
7.4. Accessing jUDDI v3 services from an existing UDDI v2 based client, plugin or tool. 62
7.5. Additional Information 62
8. UDDI Migration and Backup Tool 63
8.1. Using the tool 63
8.1.1. Get help 63

8.1.2. Use case: basic import and export 64
8.1.3. Use case: Import and Export while preserving ownership information 64
9. Using the jUDDI REST Services 67
9.1. URL Patterns and methods 67
9.1.1. Endpoints 67
9.1.2. Methods 67
9.2. Example Output 69
9.2.1. XML 69
9.2.2. JSON 69
9.3. More information 72
10. jUDDI Client for NET 73
10.1. Procedure 73
11. Using the UDDI Technology Compatibility Kit 75
11.1. Using the TCK Runner 75
11.1.1. Configuration 75
11.1.2. Running the TCK Runner 76
11.2. Analyzing the Results 77
Index 79

Preface

The jUDDI client framework facilitates interaction with any UDDI v3 compliant registry. In addition to providing a client framework for both Java and .NET, it also provides a self proclaimed Technical Compatilibity Test (TCK) Suite. The jUDDI community encourages collabration of other vendors on the TCK or on the client framework in general.

Chapter 1. Simple Publishing Using the jUDDI API

One of the most common requests we get on the message board is "How do I publish a service using jUDDI?" This question holds a wide berth, as it can result anywhere from not understanding the UDDI data model, to confusion around how jUDDI is set up, to the order of steps required to publish artifacts in the registry, to general use of the API - and everything in between. This article will attempt to answer this "loaded" question and, while not going into too much detail, will hopefully clear some of the confusion about publishing into the jUDDI registry.

1.1. UDDI Data Model

Before you begin publishing artifacts, you need to know exactly how to break down your data into the UDDI model. This topic is covered extensively in the specification, particularly in section 3, so I only want to gloss over some for details. Readers interested in more extensive coverage should most definitely take a look at the UDDI specification.

Below is a great diagram of the UDDI data model (taken directly from the specification): http://juddi.apache.org/docs/3.x/userguide/html/images/uddi_core_datastructures.gif As you can see, data is organized into a hierarchical pattern. Business Entities are at the top of the pyramid, they contain Business Services and those services in turn contain Binding Templates. TModels (or technical models) are a catch-all structure that can do anything from categorize one of the main entities, describe the technical details of a binding (ex. protocols, transports, etc), to registering a key partition. TModels won't be covered too much in this article as I want to focus on the three main UDDI entities.

The hierarchy defined in the diagram is self-explanatory. You must first have a Business Entity before you can publish any services. And you must have a Business Service before you can publish a Binding Template. There is no getting around this structure; this is the way UDDI works.

Business Entities describe the organizational unit responsible for the services it publishes. It generally consist of a description and contact information. How one chooses to use the Business Entity is really dependent on the particular case. If you're one small company, you will likely just have one Business Entity. If you are a larger company with multiple departments, you may want to have a Business Entity per department. (The question may arise if you can have one uber-Business Entity and multiple child Business Entities representing the departments. The answer is yes, you can relate Business Entities using Publisher Assertions, but that is beyond the scope of this article.)

Business Services are the cogs of the SOA landscape. They represent units of functionality that are consumed by clients. In UDDI, there's not much to a service structure; mainly descriptive information like name, description and categories. The meat of the technical details about the service is contained in its child Binding Templates.

Binding Templates, as mentioned above, give the details about the technical specification of the service. This can be as simple as just providing the service's access point, to providing the location of the service WSDL to more complicated scenarios to breaking down the technical details of the WSDL (when used in concert with tModels). Once again, getting into these scenarios is beyond the scope of this article but may be the subject of future articles.

1.2. jUDDI Additions to the Model

Out of the box, jUDDI provides some additional structure to the data model described in the specification. Primarily, this is the concept of the Publisher.

The UDDI specification talks about ownership of the entities that are published within the registry, but makes no mention about how ownership should be handled. Basically, it is left up to the particular implementation to decide how to handle "users" that have publishing rights in the registry.

Enter the jUDDI Publisher. The Publisher is essentially an out-of-the-box implementation of an identity management system. Per the specification, before assets can be published into the registry, a "publisher" must authenticate with the registry by retrieving an authorization token. This authorization token is then attached to future publish calls to assign ownership to the published entities.

jUDDI's Publisher concept is really quite simple, particularly when using the default authentication. You can save a Publisher to the registry using jUDDI's custom API and then use that Publisher to publish your assets into the registry. jUDDI allows for integration into your own identity management system, circumventing the Publisher entirely if desired. This is discussed in more detail in the documentation, but for purposes of this article, we will be using the simple out-of-the-box Publisher solution.

Tip

In UDDI, ownership is essentially assigned to a given registry entity by using its "authorizedName" field. The "authorizedName" field is defined in the specification in the operationalInfo structure which keeps track of operational info for each entity. In jUDDI, the authorizedName field translates to the person's username, also know as the publisher id,

1.3. UDDI and jUDDI API

Knowing the UDDI data model is all well and good. But to truly interact with the registry, you need to know how the UDDI API is structured and how jUDDI implements this API. The UDDI API is covered in great detail in chapter 5 of the specification but will be summarized here.

UDDI divides their API into several "sets" - each representing a specific area of functionality. The API sets are listed below:

- Inquiry - deals with querying the registry to return details on entities within
- Publication - handles publishing entities into the registry
- Security - open-ended specification that handles authentication
- Custody and Ownership Transfer - deals with transferring ownership and custody of entities
- Subscription - allows clients to retrieve information on entities in a timely manner using a subscription format
- Subscription Listener - client API that accepts subscription results
- Value Set (Validation and Caching)- validates keyed reference values (not implemented by jUDDI)
- Replication - deals with federation of data between registry nodes (not implemented by jUDDI) The most commonly used APIs are the Inquiry, Publication and Security APIs. These APIs provide the standard functions for interacting with the registry.

The jUDDI server implements each of these API sets as a JAX-WS compliant web service and each method defined in the API set is simply a method in the corresponding web service. The client module provided by jUDDI uses a "transport" class that defines how the call is to be made. The default transport uses JAX-WS but there are several alternative ways to make calls to the API. Please refer to the documentation for more information.

One final note, jUDDI defines its own API set. This API set contains methods that deal with handling Publishers as well as other useful maintenance functions (mostly related to jUDDI's subscription model). This API set is obviously proprietary to jUDDI and therefore doesn't conform to the UDDI specification.

1.4. Getting Started

Now that we've covered the basics of the data model and API sets, it's time to get started with the publishing sample. The first thing that must happen is to get the jUDDI server up and running. Please refer to this http://apachejuddi.blogspot.com/2010/02/getting-started-with-juddi-v3.html article that explains how to start the jUDDI server.

1.4.1. Simple Publishing Example

We will now go over the "simple-publish" examples. These examples expand upon the HelloWorld example in that after retrieving an authentication token, a BusinessEntity and BusinessService are published to the registry. There are two examples:

- simple-publish-portal - This is how to perform the publish operations in a way that's portable, meaning that the code logic should apply to any UDDIv3 client application library.

- simple-publish-clerk - This shows you how to perform the same actions using the helper functions in jUDDI's Client library, which greatly reduces the code required and makes things simple. This uses the UDDIClerk's functions.

1.4.1.1. Simple Publishing using Portable Code

The complete source for this example can be found here: - Portable http://svn.apache.org/repos/asf/juddi/trunk/juddi-examples/simple-publish-portable/

```
        public SimplePublishPortable() {
                try {
                        // create a client and read the config in the
 archive;
                        // you can use your config file name
                        UDDIClient uddiClient = new UDDIClient("META-INF/
uddi.xml");
                        // a UddiClient can be a client to multiple UDDI
 nodes, so
                        // supply the nodeName (defined in your uddi.xml.
                        // The transport can be WS, inVM, RMI etc which is
 defined in the uddi.xml
                        Transport transport =
 uddiClient.getTransport("default");
                        // Now you create a reference to the UDDI API
                        security = transport.getUDDISecurityService();
                        publish = transport.getUDDIPublishService();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

The constructor uses the jUDDI client API to retrieve the transport from the default node. You can refer to the documentation if you're confused about how clerks and nodes work. Suffice it to say, we are simply retrieving the default client transport class which is designed to make UDDI calls out using JAX-WS web services.

Once the transport is instantiated, we grab the two API sets we need for this demo: 1) the Security API set so we can get authorization tokens and 2) the Publication API set so we can actually publish entities to the registry.

All the magic happens in the publish method. We will look at that next.

Here are the first few lines of the publish method:

```
                        // Login aka retrieve its authentication token
                        GetAuthToken getAuthTokenMyPub = new GetAuthToken();
                        getAuthTokenMyPub.setUserID("bob");
   //your username
```

```
                        getAuthTokenMyPub.setCred("bob");
      //your password
                        AuthToken myPubAuthToken =
security.getAuthToken(getAuthTokenMyPub);
                        System.out.println(getAuthTokenMyPub.getUserID() +
"'s AUTHTOKEN = " + "******* never log auth tokens!");
```

Important

Don't log authentication tokens. In addition, whenever you're done with it, it should be *discarded*. Think of it as a logout function.

This code simply gets the authorization token for the *bob* user.

Tip

jUDDI includes two reserved usernames, *uddi* and *root*. Root acts as the "administrator" for jUDDI API calls. Additionally, the *root* user is the owning publisher for all the initial services installed with jUDDI. You may be wondering what those "initial services" are. Well, since the UDDI API sets are all implemented as web services by jUDDI, every jUDDI node actually registers those services inside itself. This is done per the specification. The user *uddi* owns the remaining preinstalled data.

Now that we have Bob's authorization, we can start publishing.

Tip

You'll note that no credentials have been set on both authorization calls. This is because we're using the default authenticator (which is for testing purposes doesn't require credentials). Most UDDI servers will require authentication.

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

            // Adding the business entity to the "save" structure, using our
publisher's authentication info
            // and saving away.
            SaveBusiness sb = new SaveBusiness();
```

```
            sb.getBusinessEntity().add(myBusEntity);
            sb.setAuthInfo(myPubAuthToken.getAuthInfo());
            BusinessDetail bd = publish.saveBusiness(sb);
            String myBusKey =
bd.getBusinessEntity().get(0).getBusinessKey();
            System.out.println("myBusiness key:  " + myBusKey);

            // Creating a service to save.  Only adding the minimum data:
the parent business key retrieved
            //from saving the business above and a single name.
            BusinessService myService = new BusinessService();
            myService.setBusinessKey(myBusKey);
            Name myServName = new Name();
            myServName.setValue("My Service");
            myService.getName().add(myServName);
            // Add binding templates, etc...
            // <snip> We removed some stuff here to make the example
shorter, check out the source for more info</snip>

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

            //and we're done, don't forget to logout!
            security.discardAuthToken(new
DiscardAuthToken(myPubAuthToken.getAuthInfo()));
```

To summarize, here we have created and saved a BusinessEntity and then created and saved a BusinessService. We've just added the bare minimum data to each entity. Obviously, you would want to fill out each structure with greater information, particularly with services. However, this is beyond the scope of this article, which aims to simply show you how to programmatically publish entities.

1.4.1.2. Simple Publishing using Clerks

The complete source for this example can be found here: - Clerk http://svn.apache.org/repos/asf/juddi/trunk/juddi-examples/simple-publish-clerk/

The sample consists of only one class: SimplePublishPortable. Let's start by taking a look at the constructor:

```
        public SimplePublishClerk() {
```

Simple Publishing Example

```
                try {
                        // create a client and read the config in the
 archive;
                        // you can use your config file name
                        UDDIClient uddiClient = new UDDIClient("META-INF/
uddi.xml");
                        //get the clerk
                        clerk = uddiClient.getClerk("default");
                        if (clerk==null)
                                throw new Exception("the clerk wasn't found,
 check the config file!");
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

Notice that this is already much more streamlined than the previous example. In this scenario, all configuration settings and credentials are stored in "META-INF/uddi.xml".

Tip

The configuration file used by clients can be overridden via the system property "uddi.client.xml". E.g. java -Duddi.client.xml=/usr/local/uddi.xml -jar MyCoolProgram.jar

UDDIClient's job is to read the configuration file and initialize the data structures for working with 1 or more UDDI nodes (or servers). It also handles automatic registration of endpoints using WSDL documents or using class annotations. UDDIClerk's job is to manage credentials and to perform a number of common tasks. Feel free to use them in your programs and help you simplify things.

The UDDIClerk also handle credentials and authentication to UDDI for you. If you didn't want to store credentials (it can be encrypted) then you can specify them at runtime very easily.

Moving on, the next function is Publish. Here's the short short version.

```
public void publish() {
              try {
                        // Creating the parent business entity that will
 contain our service.
                        BusinessEntity myBusEntity = new BusinessEntity();
                        Name myBusName = new Name();
                        myBusName.setValue("My Business");
                        myBusEntity.getName().add(myBusName);
                        //<snip>we removed a bunch of useful stuff here, see
 the full example for the rest of it</snip>
```

```
                        //register the business, if the return value is
null, something went wrong!
                        BusinessEntity register =
clerk.register(myBusEntity);
                        //don't forget to log out!
                        clerk.discardAuthToken();
                        if (register == null) {
                                System.out.println("Save failed!");
                                System.exit(1);
                        }
                        // Now you have a business and service via
                        // the jUDDI API!
                        System.out.println("Success!");

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

The UDDIClerk has a register and unregister function for nearly everything for UDDI. Between the UDDIClient and UDDIClerk, there's enough helper functions to significantly reduce the amount of code needed to work with UDDI. Here's a quick list of things they can do for you:

- Create a tModel Partition, also know as a Key Generator
- Resolve endpoints from WSDLs, Hosting directors, and other binding template references from Access Points http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908385
- Get Bindings by Version
- Add REST or SOAP tModels to a binding template
- Setup asynchronous callbacks for subscriptions
- Compare the values of a tModel Instance Info, such as Quality of Service Metrics
- Create and register services using a WADL or WSDL document
- And more…

We're also looking for the next thing to add to the client library. Have an idea? Please open a ticket on jUDDI's Issue Tracker at https://issues.apache.org/jira/browse/JUDDI.

1.4.2. About UDDI Entity Keys

There are a couple important notes regarding the use of entity keys. Version 3 of the specification allows for publishers to create their own keys but also instructs implementers to have a default method. Here we have gone with the default implementation by leaving each entity's "key" field blank in the save call. jUDDI's default key generator simply takes the node's partition and appends a GUID. In a default installation, it will look something like this:

A few tips on adding Binding Templates

uddi:juddi.apache.org:(generated GUID/UUID)

You can, of course, customize all of this, but that is left for another article. The second important point is that when the BusinessService is saved, we have to explicitly set its parent business key (retrieved from previous call saving the business). This is a necessary step when the service is saved in an independent call like this. Otherwise you would get an error because jUDDI won't know where to find the parent entity. I could have added this service to the BusinessEntity's service collection and saved it with the call to saveBusiness. In that scenario we would not have to set the parent business key.

1.5. A few tips on adding Binding Templates

Arguably, the most useful useful part of UDDI is advertising your services similar to a phone book or directory. The important part really isn't that Bob's Business exists (BusinessEntity), it's that Bob provides a service (BusinessService) and it's located at this address. This is where the Binding Template comes it. It identifies an instance of a service, its location and any other metadata associated with the endpoint that someone may want to know.

This article skips the binding Template data because it can be lengthy, but the full source for these examples shows you exactly how to build and add binding templates.

1.6. Conclusion

Hopefully this added clarity to the question, "How do I publish a service using jUDDI?".

Chapter 2. jUDDI Client Configuration Guide

2.1. Introduction

The jUDDI Java and .NET clients use an XML configuration file that dictates settings and behaviors. This guide provides an overview of the settings. Both .NET and jUDDI use the same configuration file schema. This schema is located within the source tree and with the client distribution packages of jUDDI.

2.2. Client Settings

The root XML node for the jUDDI client configuration file is always "uddi".

```
<!-- applies to Java clients only -->
uddi/reloadDelay
```

Multiple clients can be defined in each configuration file.

```
uddi/client@name="someName"
```

2.3. Nodes

At least one node is required per client. A node represents a one logical UDDI server (or cluster of servers). Each UDDI node should host at least the inquiry service. A client using the jUDDI client package can be configured to access multiple nodes at the same time.

```
<!-- if isHomeJUDDI is true, then this node is loaded by default, when no
 node is specified in client code -->
uddi/client/nodes[]/node@isHomeJUDDI=true/false
<!-- the name of the node is referenced in client code -->
uddi/client/nodes[]/node/name
<!-- the description of the node -->
uddi/client/nodes[]/node/description
<!-- the properties section defines HTTP style credentials and a runtime
 tokenizer for URLs -->
uddi/client/nodes[]/node/properties
<!-- The transport represents the class name of the transport mechanism that
 the client will use to connect
to the UDDI node. The most commonly used is
 org.apache.juddi.v3.client.transport.JAXWSTransport, however
RMITransport, InVMTransport and JAXWSv2TranslationTransport are also defined
 -->
```

```
uddi/client/nodes[]/node/proxyTransport

<!-- endpoint URLs -->
uddi/client/nodes[]/node/custodyTransferUrl
uddi/client/nodes[]/node/inquiryUrl
uddi/client/nodes[]/node/publishUrl
uddi/client/nodes[]/node/securityUrl
uddi/client/nodes[]/node/subscriptionUrl
uddi/client/nodes[]/node/subscriptionListenerUrl
<!-- note: this is for jUDDI v3.x servers only and is not part of the UDDI
 standard -->
uddi/client/nodes[]/node/juddiApiUrl
```

2.3.1. Transport Options

The Proxy Transport defines which mechanism is used for *on the wire* transport of the UDDI XML messages. JAXWS Transport is the most commonly used and assumes SOAP messaging protocol over HTTP transport layer.

RMI Transport using the Java Remote Method Invocation for transport. It's more commonly used for communicating within a J2EE container, but could be used in other cases. It's not required by the UDDI specification and is considered a jUDDI add on.

InVM Transport is for hosting jUDDI services without a J2EE container.

JAXWSv2TranslationTransport is a bridge for accessing UDDIv2 web services using the UDDIv3 data structures and APIs. Only the Inquiry and Publish services are required and they must point to SOAP/HTTP endpoints for a UDDI v2 node.

2.4. Clerks

Clerks are responsible for mapping stored user credentials to a Node and for automated registration.

```
<!-- if true, the contents of the child node xregister are registered
 when the UDDIClient object is created, using the credential and node
 reference.-->
uddi/client/clerks/registerOnStartup=true/false
```

2.5. Clerk

Clerks store credentials and map to a specific node.

```
<!-- the name is a reference to the Node that these credentials apply to-->
uddi/client/clerks[]/clerk@node - This is reference to uddi/client/node/
name, it must exist
uddi/client/clerks[]/clerk@name - This is a unique identifier of the clerk
```

```
uddi/client/clerks[]/clerk@publisher  - This is the username
uddi/client/clerks[]/clerk@password
uddi/client/clerks[]/clerk@isPasswordEncrypted=true/false
uddi/client/clerks[]/clerk@cryptoProvider=(see Crypto providers)
```

Credentials can be encrypted using the included batch/bash scripts and then appended to the configuration. Example with encryption:

```
<clerk name="default" node="default" publisher="root" password="(cipher text
 removed)"
                          isPasswordEncrypted="true"
 cryptoProvider="org.apache.juddi.v3.client.cryptor.AES128Cryptor" />
```

Clerks also have settings for the automated cross registration of Businesses and Services on start up.

```
uddi/client/clerks[]/xregister/service@bindingKey
uddi/client/clerks[]/xregister/service@fromClerk
uddi/client/clerks[]/xregister/service@toClerk
```

2.6. Digital Signatures

The Signature section contains settings that map to the Digital Signature Utility that makes working with UDDI digital signatures simple. The section contains all of the settings for both signing and validating signatures.

```
uddi/client/signature/signingKeyStorePath
uddi/client/signature/signingKeyStoreFilePassword
uddi/client/signature/signingKeyStoreFilePassword@isPasswordEncrypted
uddi/client/signature/signingKeyStoreFilePassword@cryptoProvider
uddi/client/signature/signingKeyPassword
uddi/client/signature/signingKeyPassword@isPasswordEncrypted
uddi/client/signature/signingKeyPassword@cryptoProvider
uddi/client/signature/signingKeyAlias
uddi/client/signature/canonicalizationMethod
uddi/client/signature/signatureMethod=(default RSA_SHA1)
uddi/client/signature/XML_DIGSIG_NS=(default http://www.w3.org/2000/09/
xmldsig#)
uddi/client/signature/trustStorePath
uddi/client/signature/trustStoreType
uddi/client/signature/trustStorePassword
uddi/client/signature/trustStorePassword@isPasswordEncrypted
uddi/client/signature/trustStorePassword@cryptoProvider
<!-- checks signing certificates for timestamp validity -->
uddi/client/signature/checkTimestamps
<!-- checks signing certificates for trust worthiness -->
uddi/client/signature/checkTrust
```

```
<!-- checks signing certificates for revocation -->
uddi/client/signature/checkRevocationCRL
uddi/client/signature/keyInfoInclusionSubjectDN
uddi/client/signature/keyInfoInclusionSerial
uddi/client/signature/keyInfoInclusionBase64PublicKey
<!-- default is http://www.w3.org/2000/09/xmldsig#sha1 -->
uddi/client/signature/digestMethod
```

2.7. Subscription Callbacks

The subscriptionCallbacks section defines settings uses by the subscription callback API. This enables developers to create capabilities that need to be notified immediately when something in UDDI changes through the use of subscriptions.

```
uddi/client/subscriptionCallbacks/keyDomain
uddi/client/subscriptionCallbacks/listenUrl this is the URL that is used for
 callbacks, should be externally resolvable
uddi/client/subscriptionCallbacks/autoRegisterBindingTemplate=true/false
uddi/client/subscriptionCallbacks/autoRegisterBusinessServiceKey=(key)
 append callback endpoint to this service
uddi/client/subscriptionCallbacks/
signatureBehavior=(AbortIfSigned,Sign,DoNothing,SignOnlyIfParentIsntSigned),
 default DoNothing. Applies when auto registering the endpoint to a business
 or service that is already signed.
```

2.8. XtoWsdl

XtoWsdl represents configuration parameters for importing WSDL or WADL files. Currently, the only setting is for ignoring SSL errors when fetching remote WSDL or WADL files.

```
uddi/client/XtoWsdl/IgnoreSSLErrors=true or false
```

2.9. Embedded jUDDI server

jUDDI has the ability to run in embedded mode. This means that the jUDDI services can operate without a web servlet container, such as Tomcat or JBoss. Typically, this is something that application developers would use for more advanced scenarios and for operation without network connectivity.

2.9.1. Requirements

A database server, if one is not available, the embedded Derby engine can be used.

2.9.2. Changes in configuration compared to non-embedded

- META-INF/embedded-uddi.xml needs to contain the connection settings for InVmTransport.

Changes
in
configuration

```
   <!-- In VM Transport Settings -->
   <proxyTransport>org.apache.juddi.v3.client.transport.InVMTransport</
proxyTransport>
   <custodyTransferUrl>org.apache.juddi.api.impl.UDDICustodyTransferImpl</
custodyTransferUrl>
   <inquiryUrl>org.apache.juddi.api.impl.UDDIInquiryImpl</inquiryUrl>
   <publishUrl>org.apache.juddi.api.impl.UDDIPublicationImpl</publishUrl>
   <securityUrl>org.apache.juddi.api.impl.UDDISecurityImpl</securityUrl>
   <subscriptionUrl>org.apache.juddi.api.impl.UDDISubscriptionImpl</
subscriptionUrl>

 <subscriptionListenerUrl>org.apache.juddi.api.impl.UDDISubscriptionListenerImpl</
subscriptionListenerUrl>
   <juddiApiUrl>org.apache.juddi.api.impl.JUDDIApiImpl</juddiApiUrl>
```

- The serverside config file juddiv3.xml needs to be added to the classpath.
- A META-INF/persistence.xml needs to be added.
- Add the juddi-core (UDDI server) and derby (Embedded Database) dependencies to the pom. Use the juddi-core-openjpa jar for OpenJPA.

See also the hello-world-embedded example.

Chapter 3. Key Format Templates

3.1. UDDIv3 key format

The UDDI v3 keys are formatted such that they are human readable. 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".

3.2. jUDDI key format templates

The jUDDI client has taken the key format approach one step further so the name part of the key actually helps you understand what the key is for, or even better in the case of a binding template what server registered the key.

3.2.1. Advantages of using a template

Using a binding Key with format `uddi:${keyDomain}:binding_${serverName}_${serviceName}_${portName}_${serverPort}` contains valuable information for two reasons - you can tell from the bindingKey what server registered it. This is helpful information if you want to debug connectivity issues between a client and the UDDI server. - if a server goes to register a bindingTemplate it registered before it won't create a second bindingTemplate, so it won't leave behind dead or duplicate bindingTemplates.

3.2.2. Default UDDIKeyConvention Key Templates

The default templates setup by the jUDDI client are:

```
uddi:${keyDomain}:business_${businessName}
uddi:${keyDomain}:service_${serviceName}
uddi:${keyDomain}:service_cache_${serverName}
uddi:${keyDomain}:binding_${serverName}_${serviceName}_${portName}_
${serverPort}
```

where tokens are expressed using `${}`. The templates are defined in the UDDIKeyConvention class.

3.2.3. How to use the templates?

At runtime a serviceKey can be obtained using

```
String serviceKey = UDDIKeyConvention.getServiceKey(properties,
 serviceName);
```

Where to define to properties?

The serviceName can be specified in as a property in the first argument, or it can explicitly passed as the second argument. Using the second argument overrides what's specified in the properties. By default it will use the service template `uddi:${keyDomain}:service_${serviceName}`, but if you wish to use a different template you can simple specify that as a property, for example

```
String myCustomServiceFormat = "uddi:${keyDomain}:s_${serviceName}";
properties.add(Property.SERVICE_KEY_FORMAT, myCustomServiceFormat);
String myCustomFormattedServiceKey =
 UDDIKeyConvention.getServiceKey(properties, serviceName);
```

3.2.4. Where to define to properties?

You can define the properties in your code, or you can obtain and pass in the properties defined in your `uddi.xml`. For an exmaple of this you can check out the `META-INF/wsdl2uddi-uddi.xml` of the `wsdl2uddi` example where for the `default` node we set

```
        ...
        <nodes>
            <node>
                <name>default</name>
                <properties>
                    <property name="serverName" value="localhost"/>
                    <property name="serverPort" value="8080"/>
                    <property name="keyDomain"
 value="uddi.joepublisher.com"/>
                    <property name="businessName" value="WSDL-Business"/>
                </properties>
   ...
     </node>
           ...
        </nodes>
        ...
```

and you can obtain the properties like

```
UDDIClient uddiClient = new UDDIClient("META-INF/wsdl2uddi-uddi.xml");
Properties properties =
 uddiClient.getClientConfig().getUDDINode("default").getProperties();
```

This is exactly what the WSDL2UDDI implementation does, and it the reason the class requires the properties in the constructor.

Chapter 4. Using the jUDDI GUI

Starting with jUDDI v3.2, a nearly full featured UDDI v3 compliant web user interface is included called the jUDDI Graphical User Interface, or jUDDI GUI. It is also referred to as the jUDDI Console, or jUDDI User Console. The jUDDI GUI is a web application that can run in most web servlet containers, such as Tomcat and can be deployed along side of the jUDDI Web Services war (juddiv3.war). From the jUDDI GUI, users can browse, search, create, update, digitally sign and delete most UDDI entities, as well as preform custody transfers, configure subscriptions.

As of version 3.2, the jUDDI GUI supports the complete functionality of the following UDDI services

- Inquiry
- Publish
- Security
- Custody Transfer
- Subscription

4.1. Requirements

The jUDDI GUI needs two things in order to operate.

- UDDI v3 compliant services
- A J2EE application server, such as Tomcat, Jboss, Jetty or maybe even in Winstone
- Optionally, a container level authentication mechanism that supports role based authentication (for remote configuration)

4.2. Tasks

The following sections detail how to perform basic tasks using the jUDDI GUI. Hopefully, the user interface is intuitive enough that thorough guidance isn't necessary.

4.2.1. Your first sign on

Typically, the jUDDI GUI is accessed via a URL in a web browser, such as this: http://localhost:8080/juddi-gui. This URL will probably be different from this if someone else set up jUDDI GUI for you (such as a system administrator), in which case, you'll want to ask them for the correct URL. Once loading the page, you should see something similiar to this.

Figure 4.1. Welcome to jUDDI, Please select a language.

Select a language, then click the button labeled "Go".

Tip

Would you like to see the jUDDI-GUI in a different language that the one's listed and want to offer some translation help? Please contact us!

Important

The juddi-gui stores your language preference as a cookie. No personally identifiable information is stored there.

After clicking on "Go", you should see something similar to the next two screen shots.

Your
first
sign

Tip

Why would it be different? The jUDDI GUI is based on the Twitter Bootstrap API and is designed to automatically rearrange the user interface based on screen size and resolution. Small form factor devices, such as tablets and smart phones generally function as normal, except that the upper navigation bar becomes condensed into a single button.

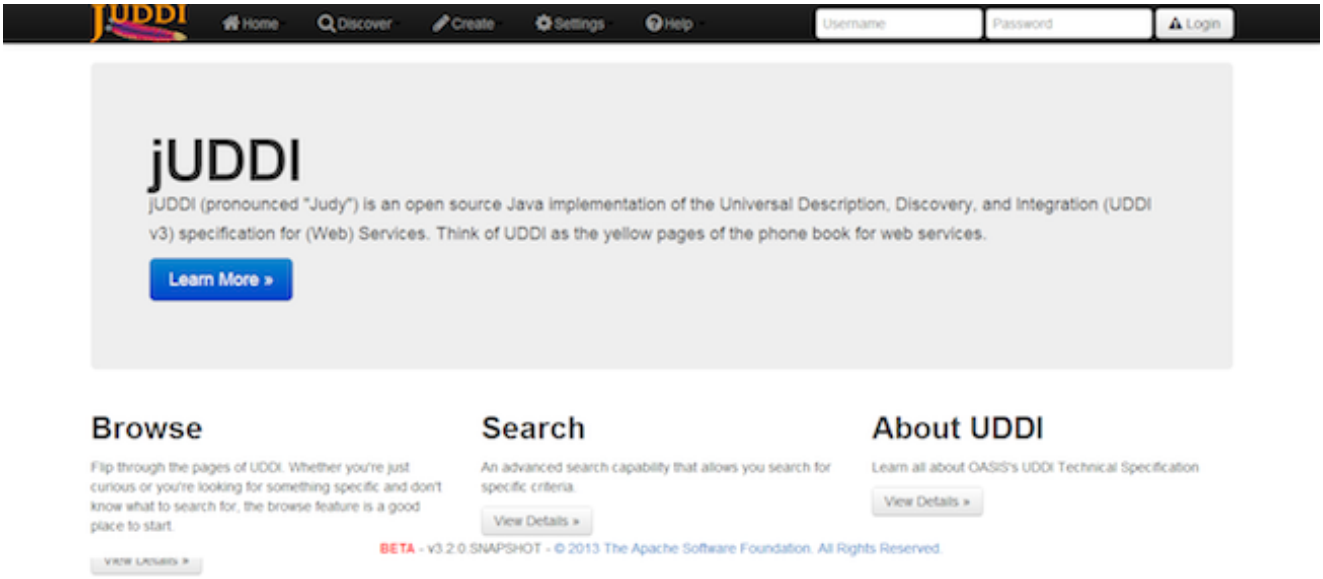
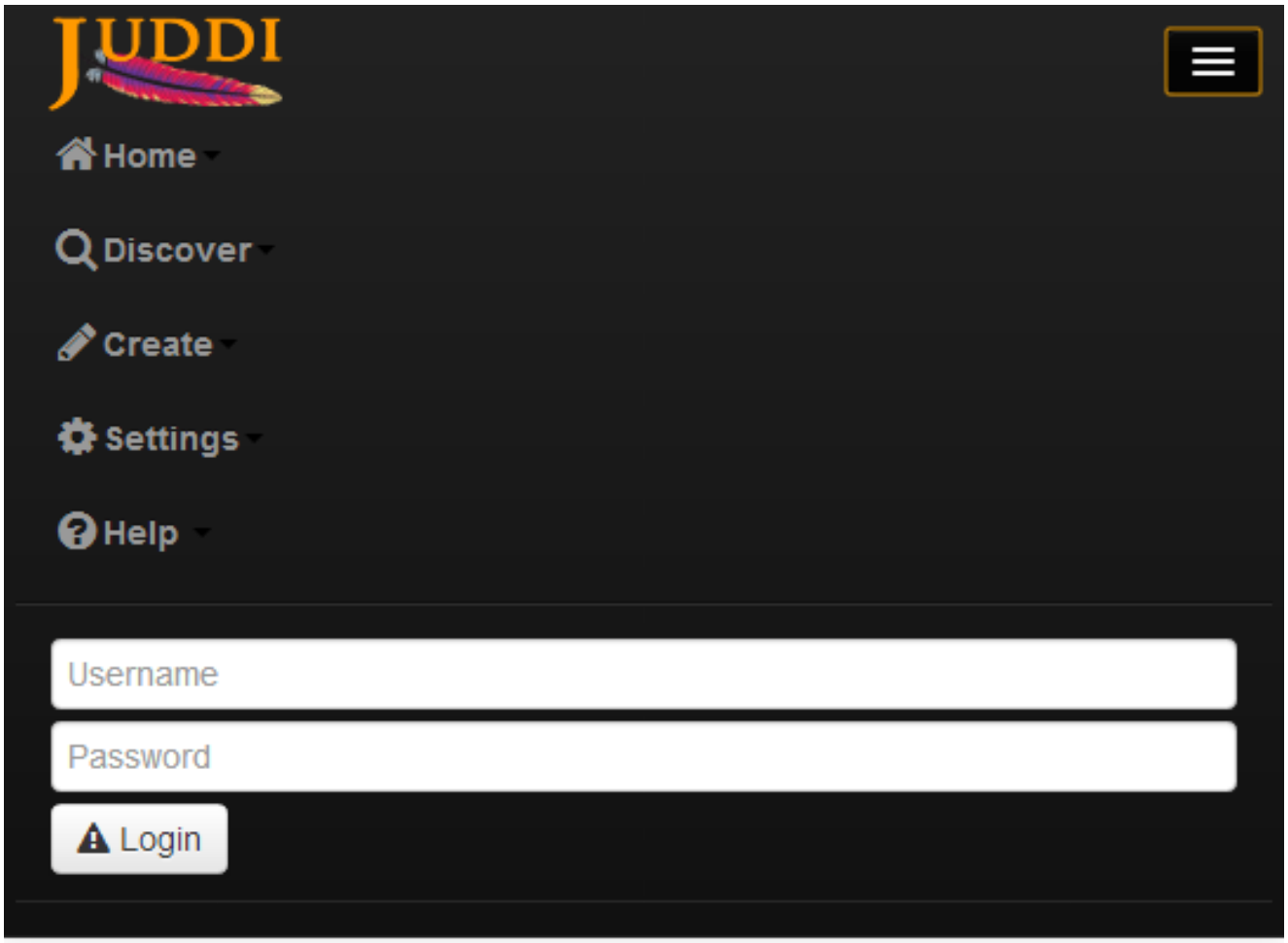


Figure 4.2. Full menu bar for computers or large displays.

Your
first
sign



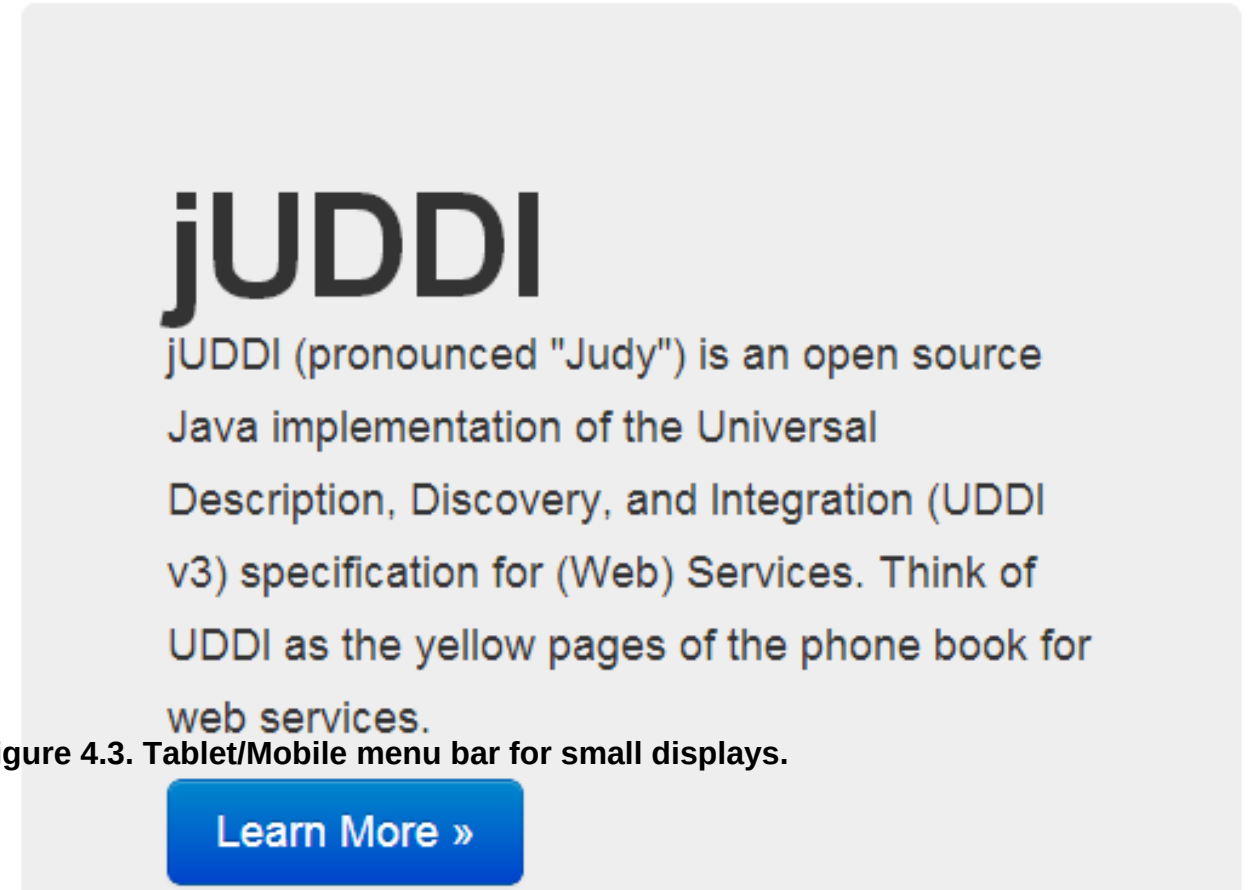


Figure 4.3. Tablet/Mobile menu bar for small displays.

For now, let's just focus on the menu or navigation bar.

4.3. The Menu Bar

Figure 4.4. The Menu Bar.

The menu bar is designed to make navigation simple and intuitive.

- Home - This sub-menu contains links towards information that is tailored towards you, such as all the businesses you own, subscriptions, custody transfer, and publisher assertion (business relationships)
- Browse - This sub-menu makes it simple to find stuff in UDDI by letting you flip the pages of the directory.
- Create - This sub-menu makes it simple to create new UDDI entities, such as businesses, services, tModels, import from WSDL/WADL and some advanced operations.
- Settings - This page is typically access controlled and enables administrators to remotely configure the juddi-gui.
- Help - Contains links to the Internet for more help with jUDDI and source code access
- Login/Logout - many registries require authentication. These buttons support both HTTP and UDDI Auth Token style of authentication.

4.4. Logging in to UDDI Services

Assuming that your UDDI services require authentication, you'll probably want to login with your username and password. This is done using the Login/Logout section the Menu bar (top right of the screen).

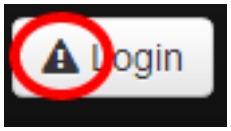


Figure 4.5. Login Warning.

Caution

If you happen to notice that a warning symbol next to the Login button, use caution! Your password may be exposed if you proceed.

Tip

The Warning symbol on the Login portion of the Menu bar will be present unless the following conditions are met: Communication from your browser to juddi-gui is encrypted using SSL/TLS AND the communication from juddi-gui to the UDDI services is encrypted using SSL/TLS.

4.5. Logging Out

Once logged in, just "Welcome <username>" button to log out.

4.6. Discover (Browse UDDI)

All of the Browse pages support pagination, that is you can flip through the pages of the database, as if it were a phone book.

In addition, search results can be filtered by language. On each Discover page, you will see the following

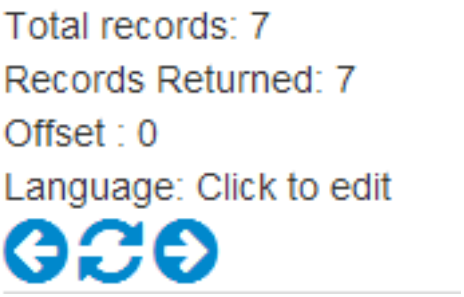


Figure 4.6. Browse Options.

Click on "Click to Edit", enter your desired language code, then either press enter, or click "Ok" and the results will be filtered automatically. See "Language Codes" for more information.

4.6.1. Business Browser

To browse for a UDDI Business, simply click on the word *Discover* from the top navigation bar and select Businesses.

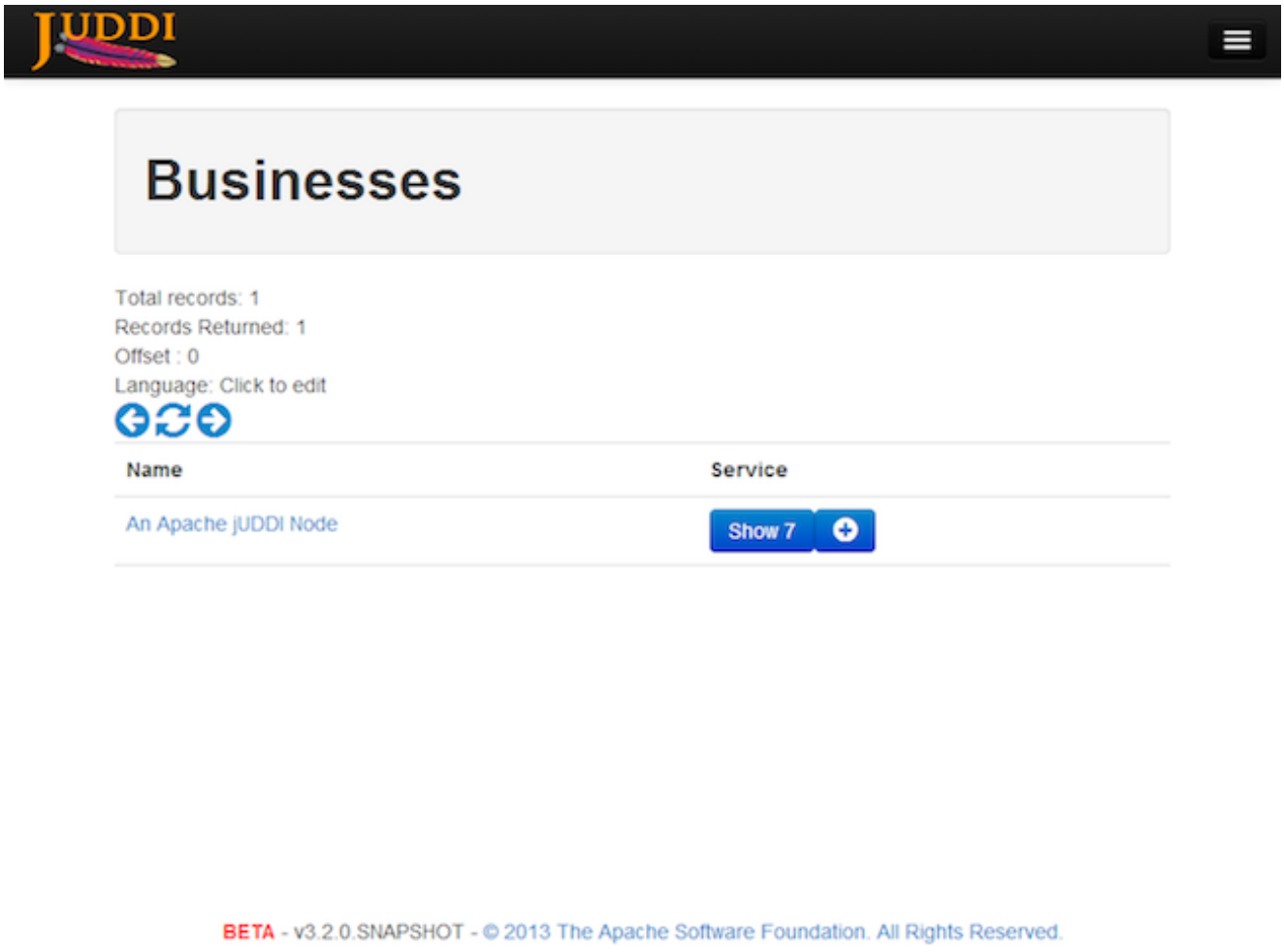


Figure 4.7. Browse Business.

When clicking on "Show XX" (XX is the number of services that business has)

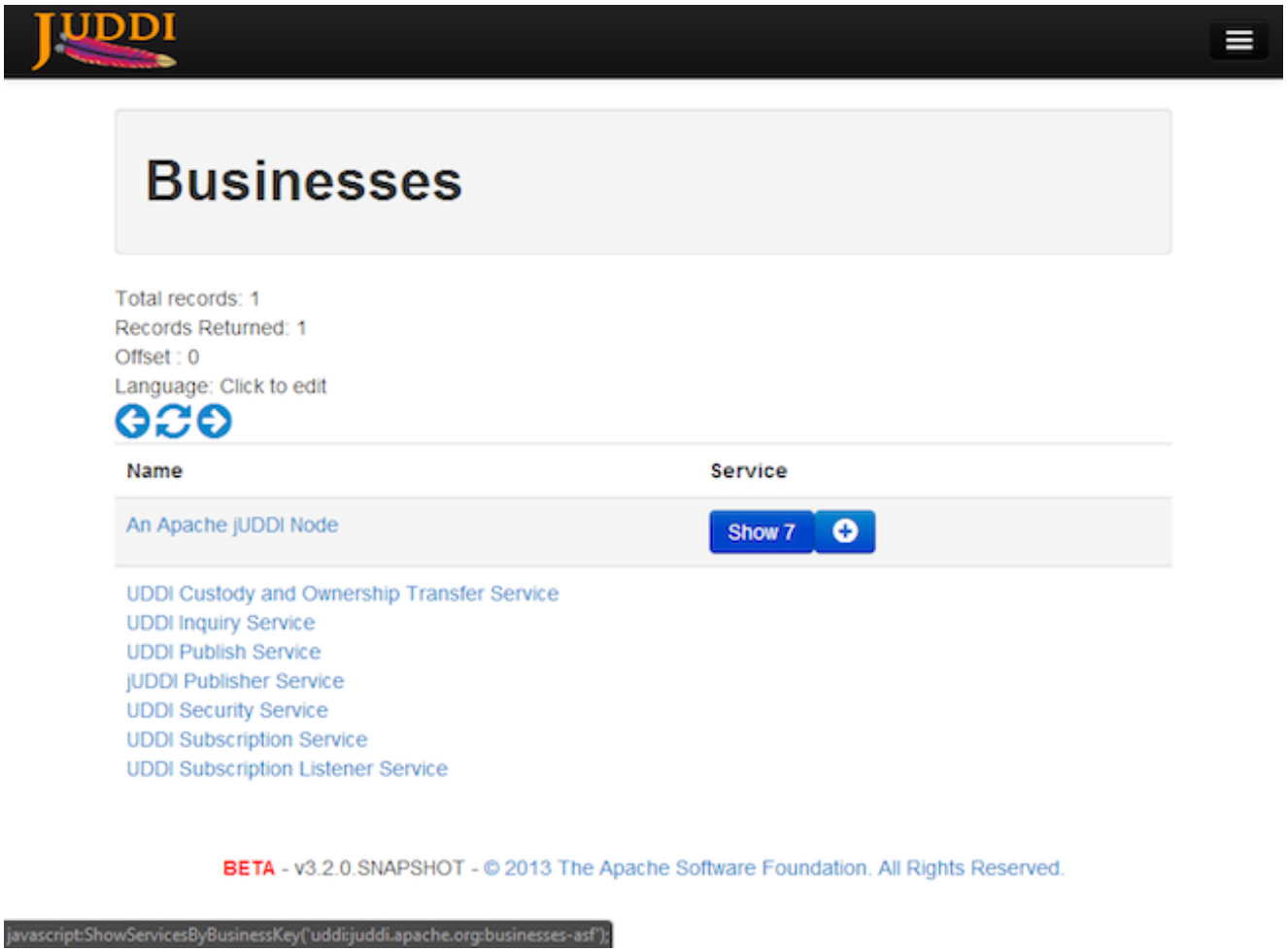


Figure 4.8. Browse Business Zoomed in.

The (+) Plus button will enable you to add a new service that belongs to the business on the same table row.

4.6.2. Service Browser

To browse for a UDDI Service, simply click on the word *Discover* from the top navigation bar and select Services. Clicking on the Name of the service, will bring you to the Service Editor page. Click on the owning Business key to bring you to the Business Editor page.

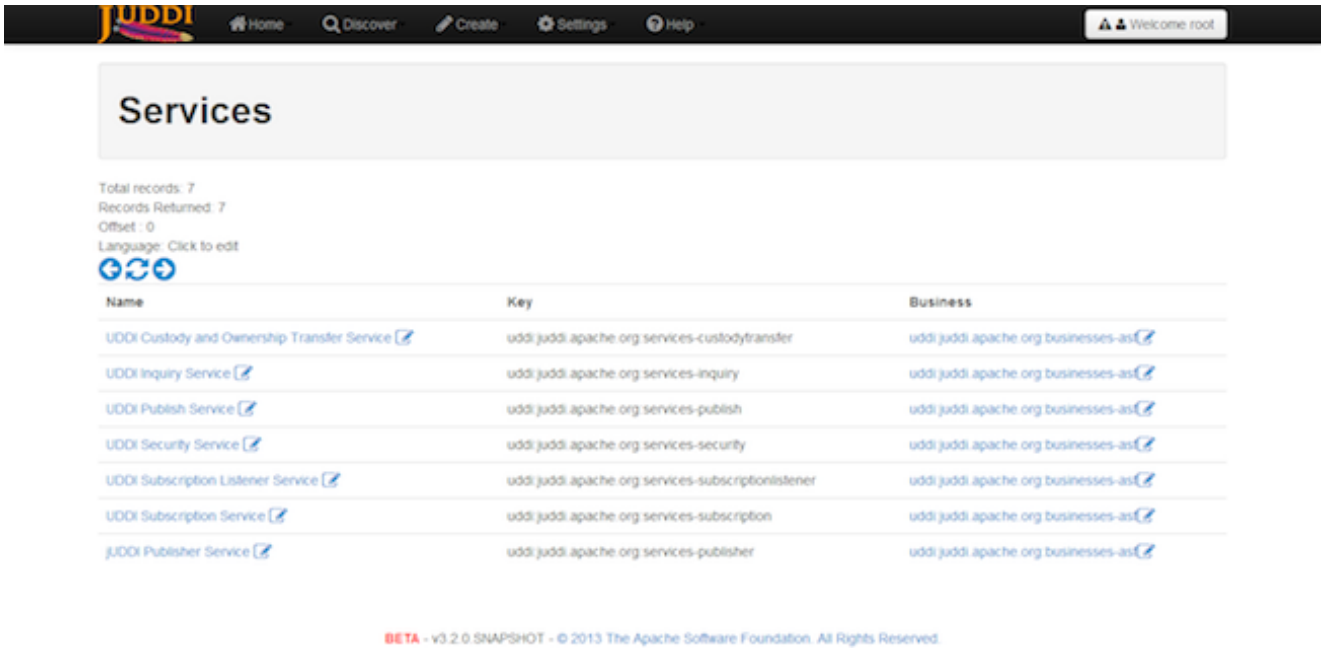


Figure 4.9. Service Browser.

4.6.3. tModel Browser

To browse for a UDDI tModel simply click on the word *Discover* from the top navigation bar and select tModel. Clicking on the Key of the tModel, will bring you to the tModel Editor page.

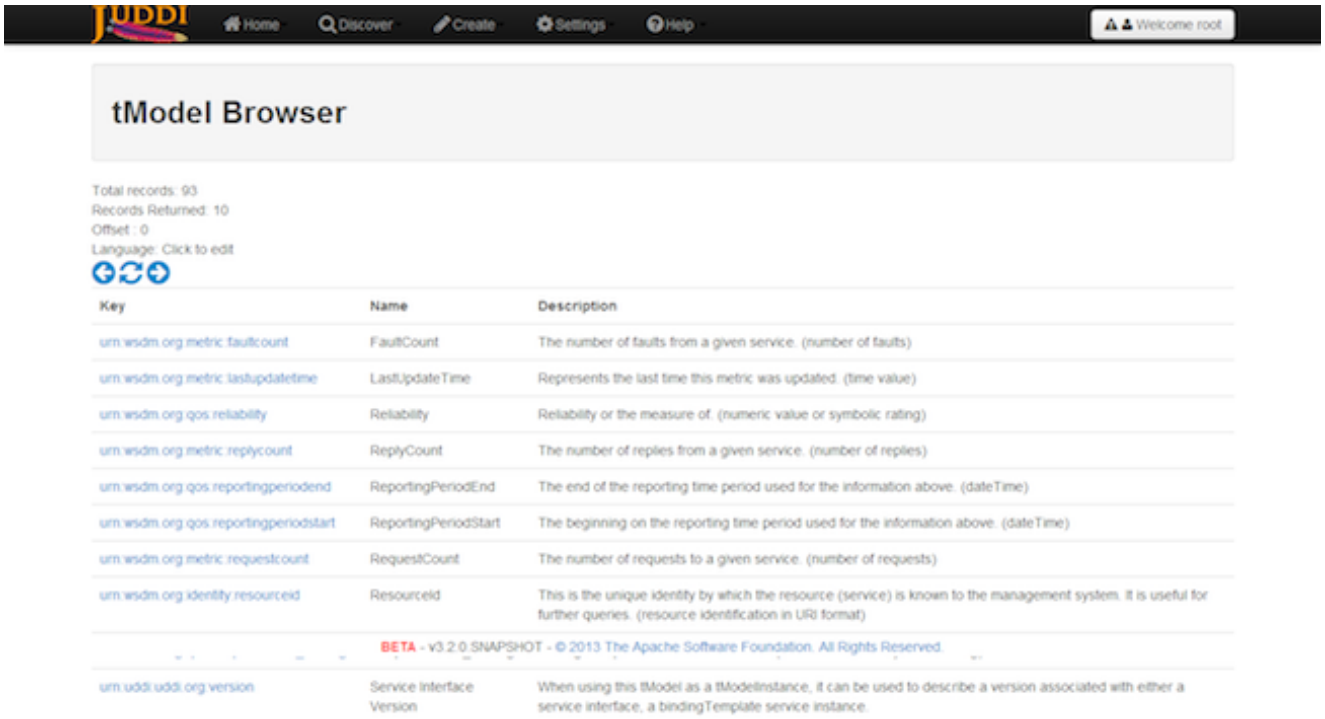


Figure 4.10. tModel Browser.

4.6.4. Search

Searching UDDI provides you with the capabilities to make both simple and complex queries. To search, simply click on the word *Discover* from the top navigation bar and select *Search*.

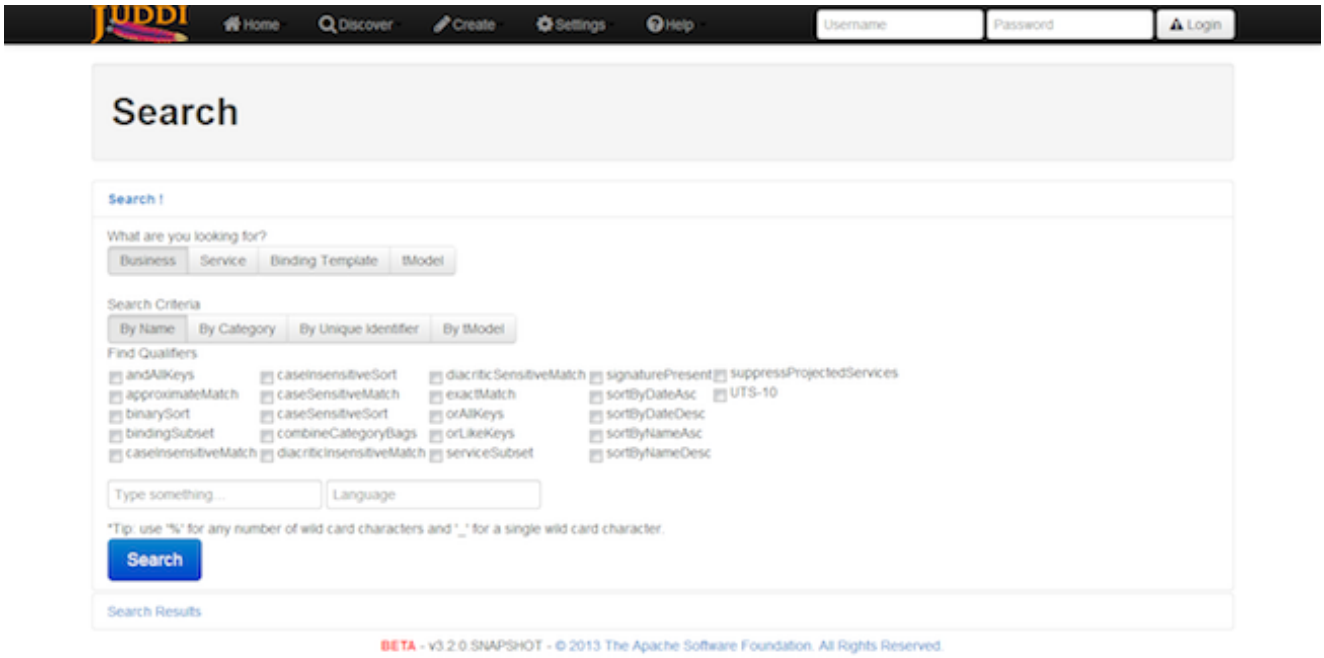


Figure 4.11. Search.

You first need to select what you're looking for. You can either search a Business, Service, Binding Template, or tModel.

Tip

Not all combinations are valid. For instance, you can't search for a Binding Template by Name because UDDI's binding templates do not have names.

Important

UDDI offers a wider, richer search capability. The juddi-gui's search page is in comparison, limited. If you have the need for more complex searches, you'll probably have to write some code to do so.

Tip

When using the wildcards (%, ?), you have to add the find qualifier, approximateMatch.

4.7. Creating new Entities

The jUDDI GUI has the ability to create and register new UDDI entities.

4.7.1. Create a tModel

From the menu, select Create, then tModel. For tModels, the only required item is the Name element.

4.7.2. Create a tModek Key Generator (Partition)

Important

If you want to create your own UDDI keys (recommended) rather than use the not so user friendly server generated GUID values, then you'll have to make a Key Generator first! Read on!

A tModel Key Generator is a special kind of tModel that enables you to define your own keys (for anything in UDDI) for your own "domain". A "domain" is similar to the Domain Name System used by the Internet to resolve user friendly names, such as www.apache.org, to an IP address. This effectively allows you to define any arbitrary UDDI key prefix that you want. For example, if you wanted a UDDI key defined as "uddi:www.mycompany.com:salesbusiness1", you would first have to create a tModel key generator (partition) with the value of "uddi:www.mycompany.com:keygenerator". TModel keys must start with "uddi:" and end with ":keygenerator". This is part of the UDDI specification and acts as a governance mechanism. You can also create a tModel Key Generator by using the Create tModel menu option and by adding the appropriate settings (assuming you know the secret sauce) or you can simply click on the word *Create* from the top navigation bar and select *tModel Partition (Key Generator)*.

Create
a
Business

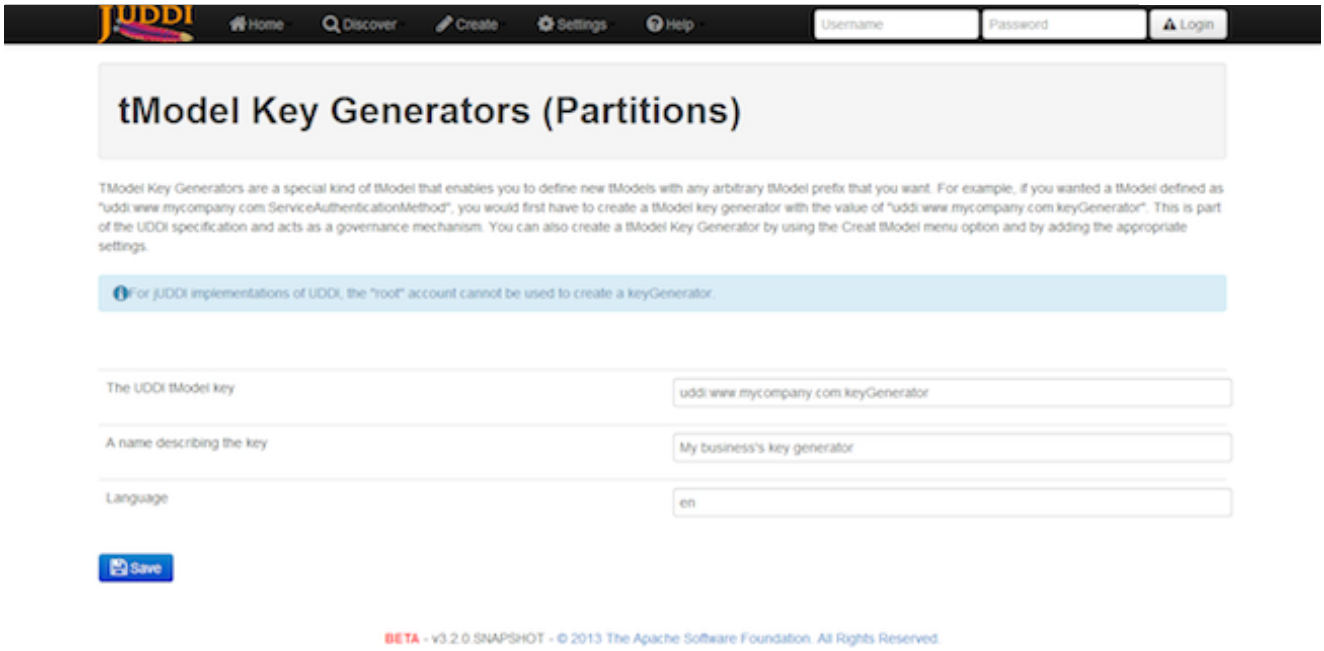


Figure 4.12. Create a tModel Key Generator (Partition).

Tip

You can also use nested partitions such as "uddi:www.mycompany.com:keygenerator" and "uddi:www.mycompany.com:sales:keygenerator". UDDI uses the colon ":" as a separator for partitions. This will enable you to make UDDI keys such as "uddi:www.mycompany.com:biz1" and "uddi:www.mycompany.com:sales:biz2".

Tip

UDDI key names can be at MOST 255 characters long!

4.7.3. Create a Business

The UDDI Business entity enables you to define and advertise your business with a variety of ways. To create a new business, simply click on the word *Create* from the top navigation bar and select *Business*.

Tip

The "Create", "Business" page is also the same page to use when editing an existing business.

Create
a
Business

Figure 4.13. Create Business.

Businesses in UDDI only require you to define at least one name. All of fields are optional. Business entities can have 0 or more of just about everything. For now, let's just make a Name, give it a Value and then save our new business. To add a new Name, click the "+" button next to the "Name". Then click on "Click to edit" next to "Value". If you make a mistake or wish to remove the "Name" or any other element, click on the trash can.

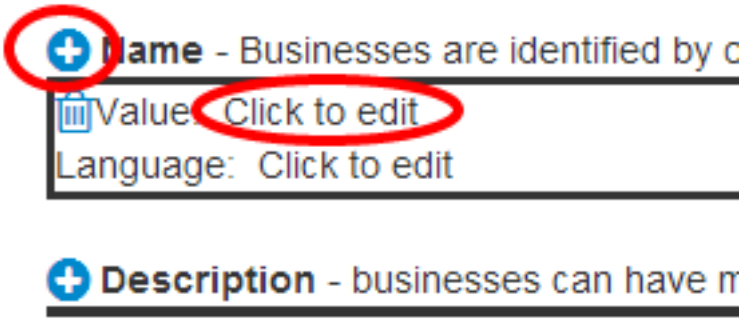


Figure 4.14. How to Add and Delete items.

If you read the previous section on tModel Key Generators, then you know all about UDDI keys. This is your one and only chance to get it right. Once your done, click "Save". Congrats! You've just made your first UDDI business!

Important

When working with UDDI entities, you cannot change the UDDI key after it has been created.

The Business Editor/Creator web page, along with the other editor/creator pages, has a ton of other interesting things that you can do. Since there's way too much stuff to look at, we broke them up into logical sections.

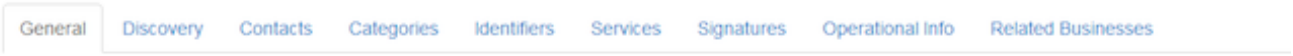


Figure 4.15. Business Editor's Section.

In case you can't see the picture above:

- General - Names and Descriptions
- Discovery URLs - Usually a link to a web page
- Contacts - Points of Contact, such as Sales, Tech Support, etc
- Categories - These reference tModels and act as a way to categorize your business.
- Identifiers - Can be used for Tax IDs, DUNS Number, or anything else that you can think of.
- Services - This is the meat and potatoes of UDDI, advertising all the great services that your business provides.
- Signatures - Digital Signatures prevent tampering
- Operational Info - Who created it and when
- Related Businesses - This is where people can find out if you have a business relationship with someone else. It's also called Publisher Assertions.

Tip

Clicking on each tab will supply additional information.

Tip

If a business, service, or tModel is signed, the juddi-gui will automatically attempt to validate the signature. You'll see a thumbs up or thumbs down icon to let you know.

4.7.4. Create a Service

Creating a new service is simple so long as you remember that a service must be attached to a business. There are a few ways to create a new business.

The first option is to locate the business that you wish to add a service to via the Business Browser and then click the Plug button.

Create
a
Service

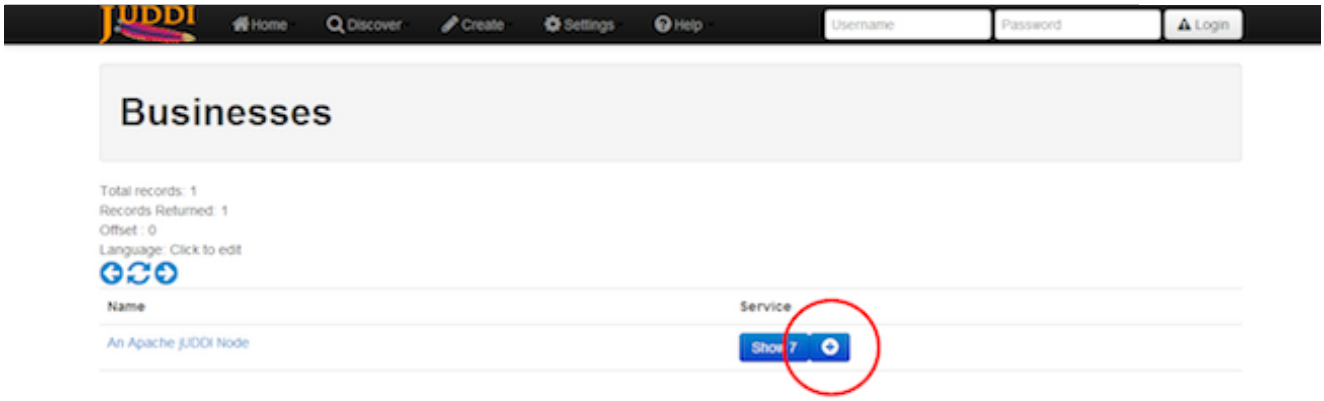


Figure 4.16. Add a Service via Business Browser.

The second option is to bring up the Business Editor for the business you want to add a service to, then click on the Services tab, then "Add a Service".

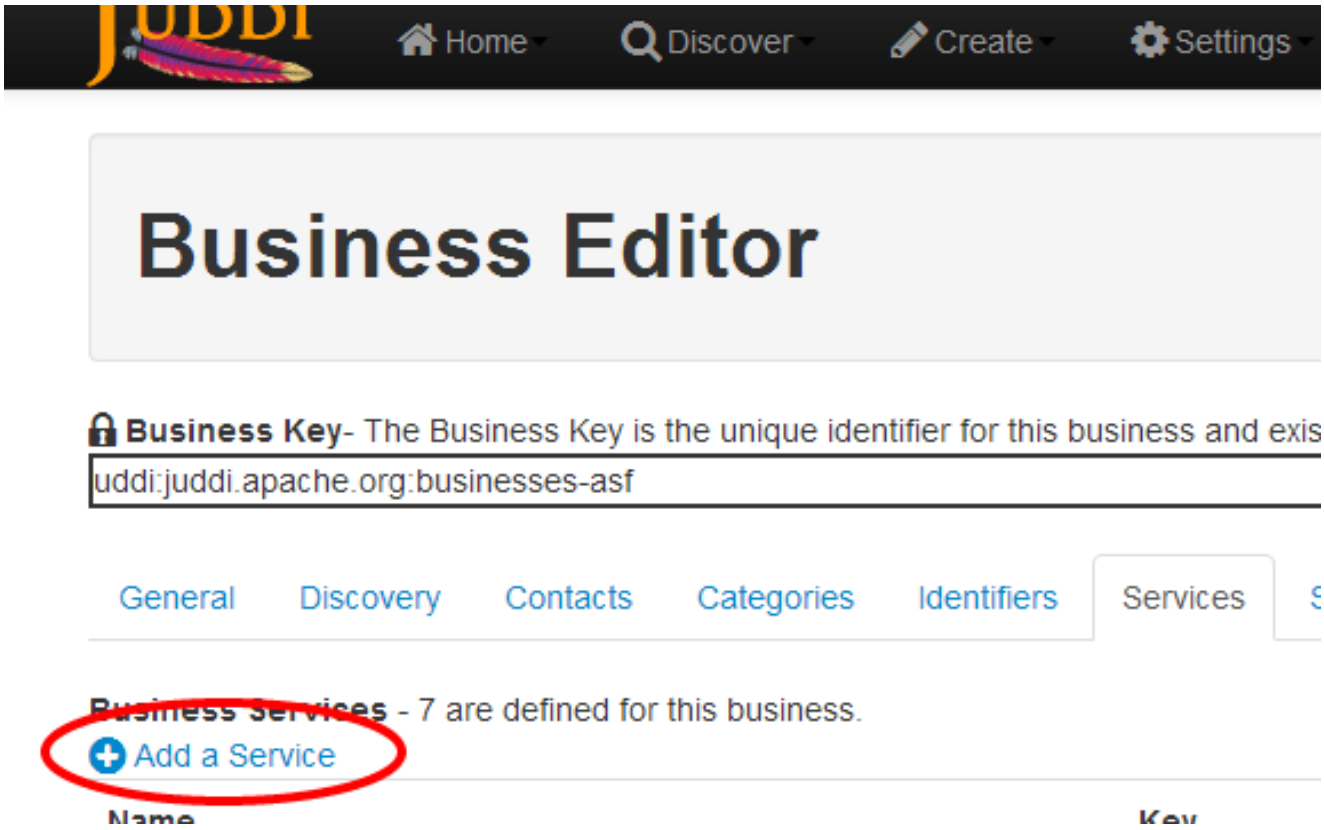


Figure 4.17. Add a Service via Business Editor.

Import
from
WSDL

Tip

Services require at least one name. Everything else is considered optional.

4.7.5. Import from WSDL or WADL

The jUDDI client provides programmatic access to convert a WSDL or WADL document into UDDI Service and tModel entries. The juddi-gui takes advantage of this and provides a simple to use interface to quickly and easily import your SOAP and REST services into UDDI.

From the Create menu, select Register Services from WSDL or WADL.

The process is pretty straight forward.

1.Provide the location of the WSDL or WADL file. It must be web accessible from the server hosting juddi-gui.war. If it is password protected (such as Digest, Basic or NTLM) provide a password to access the WSDL or WADL. Your credentials will not be saved anywhere. 1.The key domain. The imported UDDI service, binding, and tModels will all use this key partition/domain for key generation. The juddi-gui will populate this field with the domain of the URL entered in step 1. If you don't like, go ahead and change it. One will be automatically created for you. 1.Pick a business to attach the imported data to. 1.Review and Approve. The Preview button will do all of the processing except saving the content, so it is a good way to get a preview of what will happen. Save will do the processing and save it.

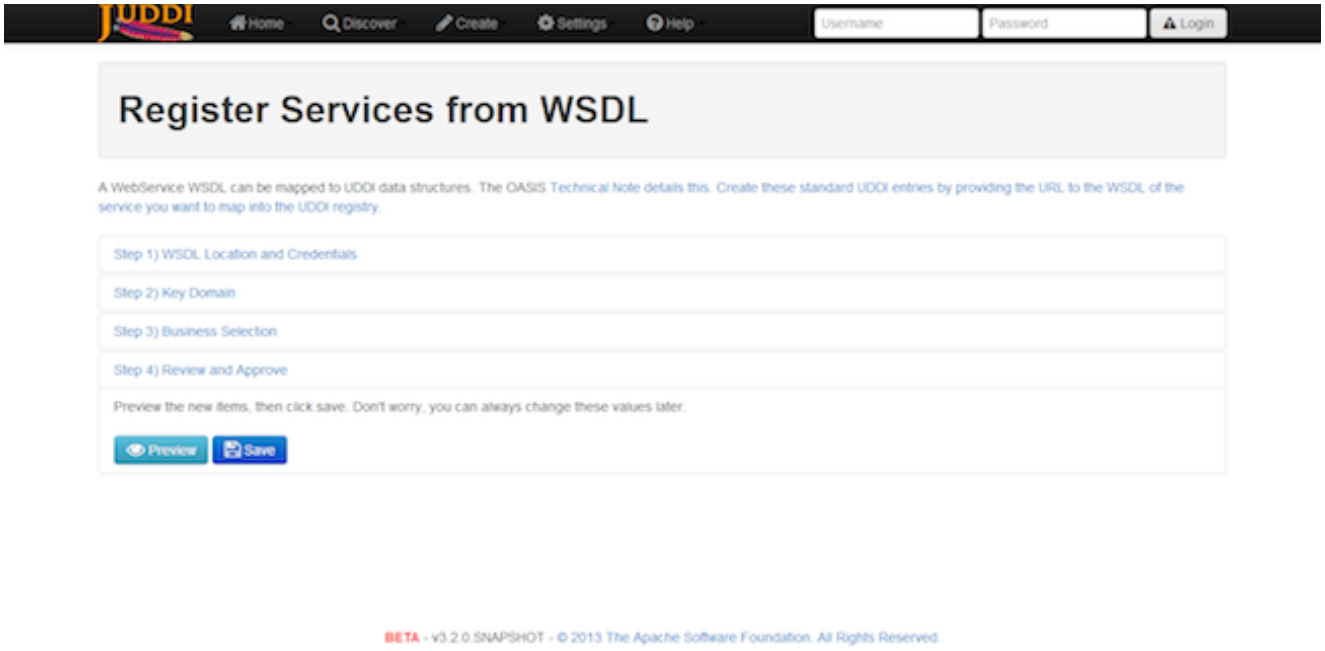


Figure 4.18. Importing a Service from WSDL or WADL.

4.8. Custody Transfers

Custody Transfers are used to give away ownership and edit permission for UDDI business and tModels. It's not used very often, but the workflow is simple.

1.Two business representatives agree to transfer either a business(s) or tModel(s) from business A to business B. 1.Business A creates a transfer token 1.Business A gives the transfer token data to Business B's representative (perhaps via email?) 1.Business B accepts the token and transfers the ownership over.

All of these actions are processed at the Transfer Ownership page from the Home menu.

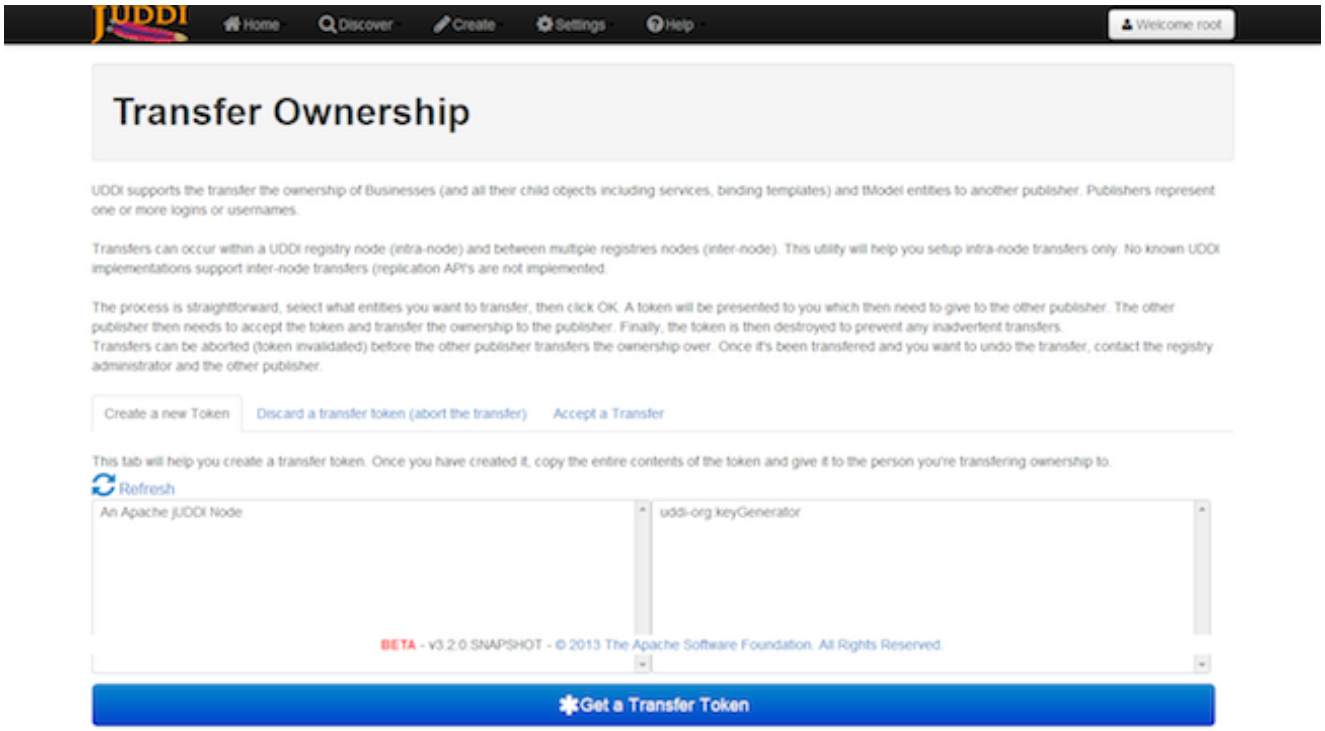


Figure 4.19. Custody Transfer.

4.9. Publisher Assertions

Publisher Assertions are how two different businesses can setup a UDDI Business Relationship. This essentially means that other users can see that this is a relationship between business A and B and that they can perform queries based on the relationship.

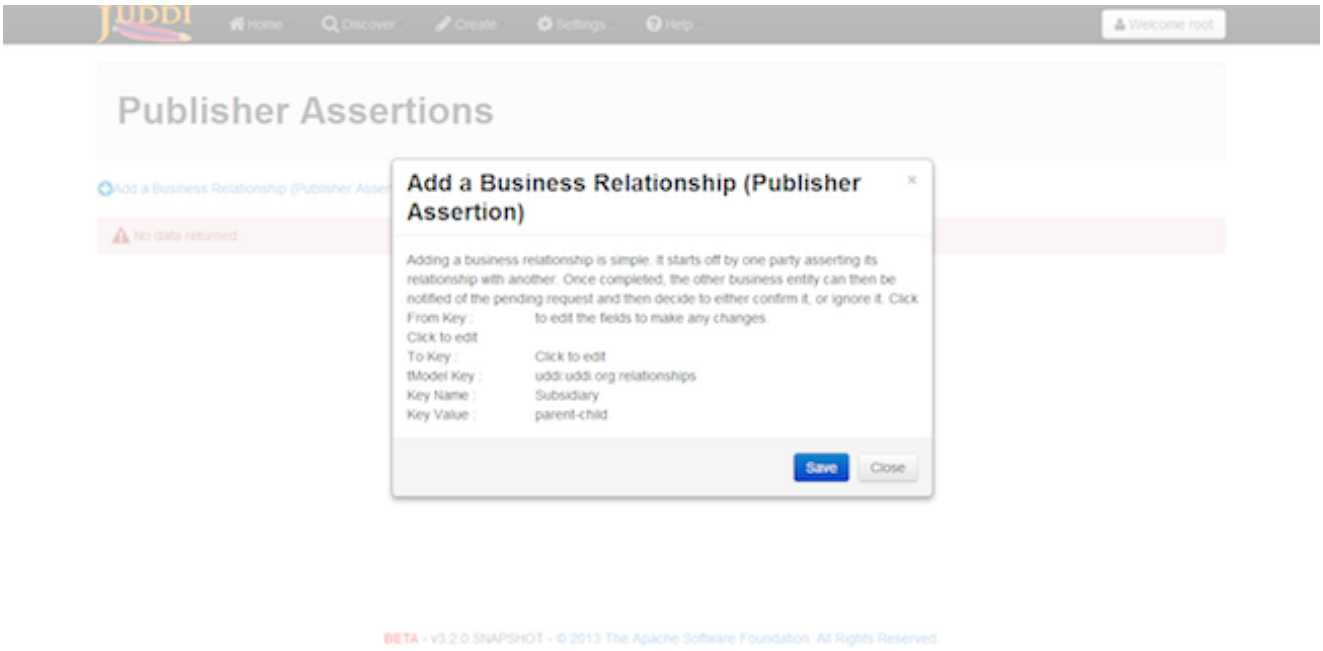


Figure 4.20. Publisher Assertion.

4.10. Subscriptions

Subscriptions in UDDI are used to easily detect when changes are made to a UDDI node.

4.10.1. Create a new subscription

To create a new subscription, you must first be logged in. Click on Home, then Create Subscription.

Subscriptions can either be for a set of specific items or for search results.

Create
a
new

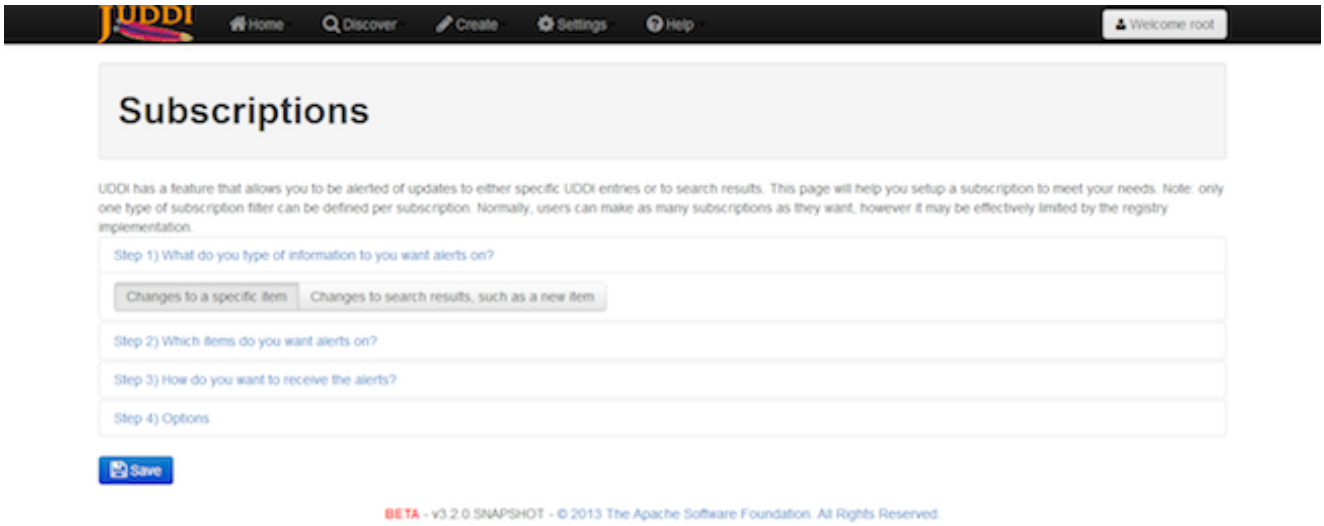


Figure 4.21. Create a Subscription, Specific Item or Search Results.

In our example, we've selected a set of specific items.

Create a Subscription, Select Items. image::images/juddi-gui-subscription2.png[Create a Subscription, Select Items

To add an item to the list, click on Add. The item chooser will appear. Check each item to add them to the list. To remove, select the item, then click remove.

Create
a
new

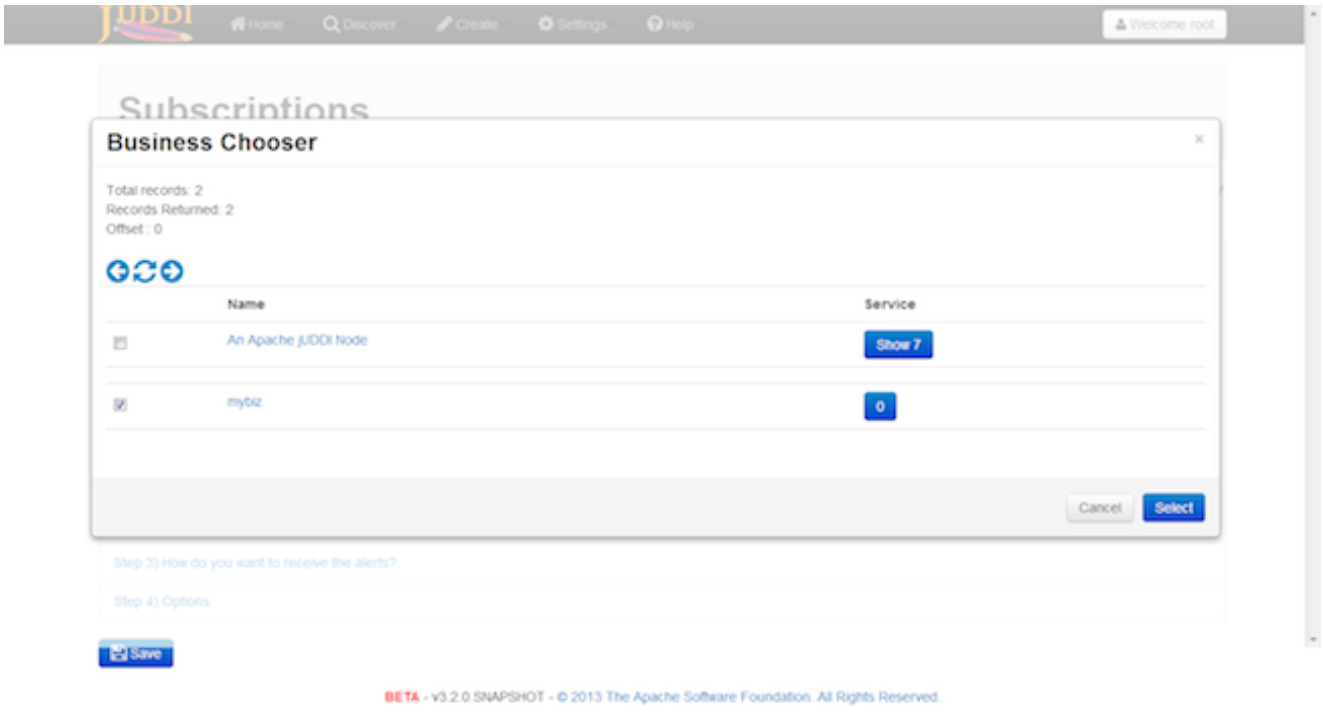


Figure 4.22. Create a Subscription, Add an item using the chooser.

Specific items are added by entity keys.

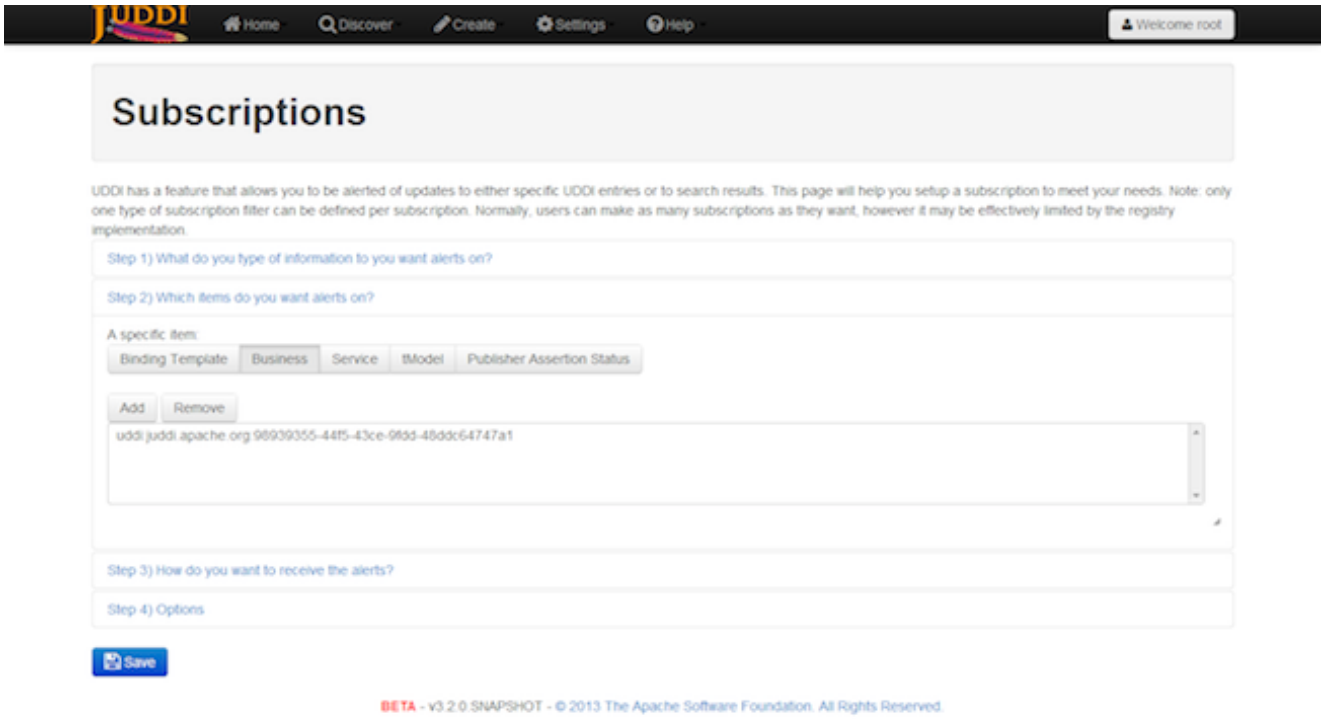


Figure 4.23. Create a Subscription, Item Added.

Create a new subscription

Next is the delivery mechanism. The UDDI node can deliver the notifications to you if you have your own implementation of the UDDI Subscription Listener service. (The juddi-client contains this for you if you were looking to develop a solution). In addition, the UDDI node can email the results to you (in XML format).

Tip

Since jUDDI 3.2.1, you can also configure jUDDI to send you a more human readable version of the subscription notification. To configure, all that is needed is to add a special transport tModel instance to your subscription binding, uddi:uddi.org:transport:userfriendlysmtp.

The other option is to periodically poll the UDDI server and get your subscription results (see the News Feed).

Here, we've selected the, I'll pick them up, option.

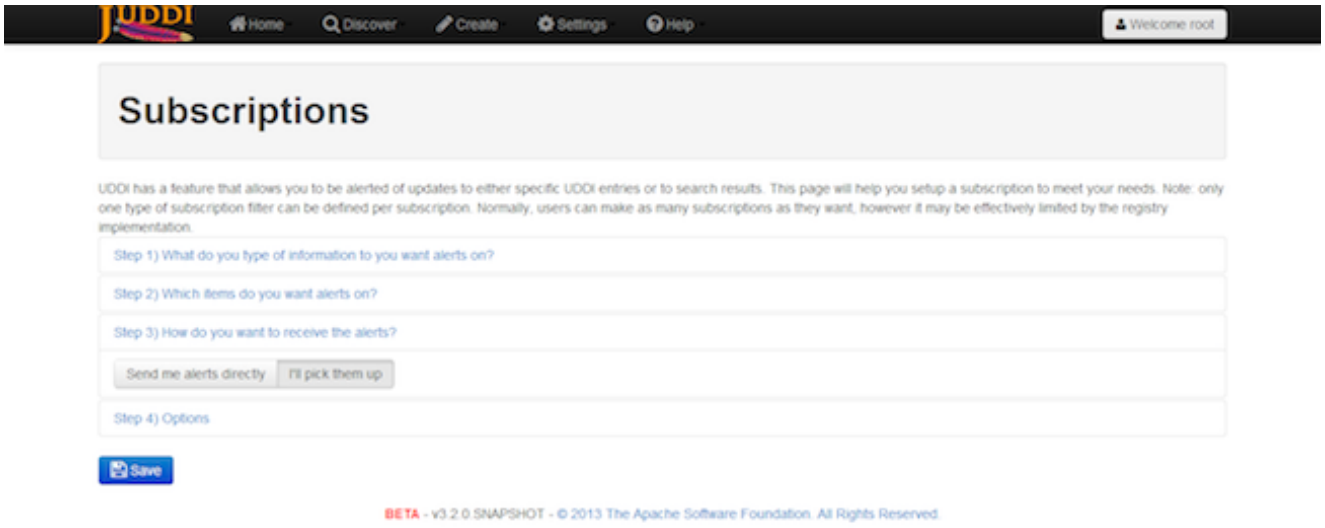


Figure 4.24. Create a Subscription, Delivery Mechanism.

The final slider provides subscription options. * Expiration - a date where the subscription expires * Notification Interval - this is only used when the UDDI node sends the notifications to you via the Subscription Listener Service * Brief - If true, the UDDI node will only tell you which items have changed, not what the change was.

View
My
Subscriptions

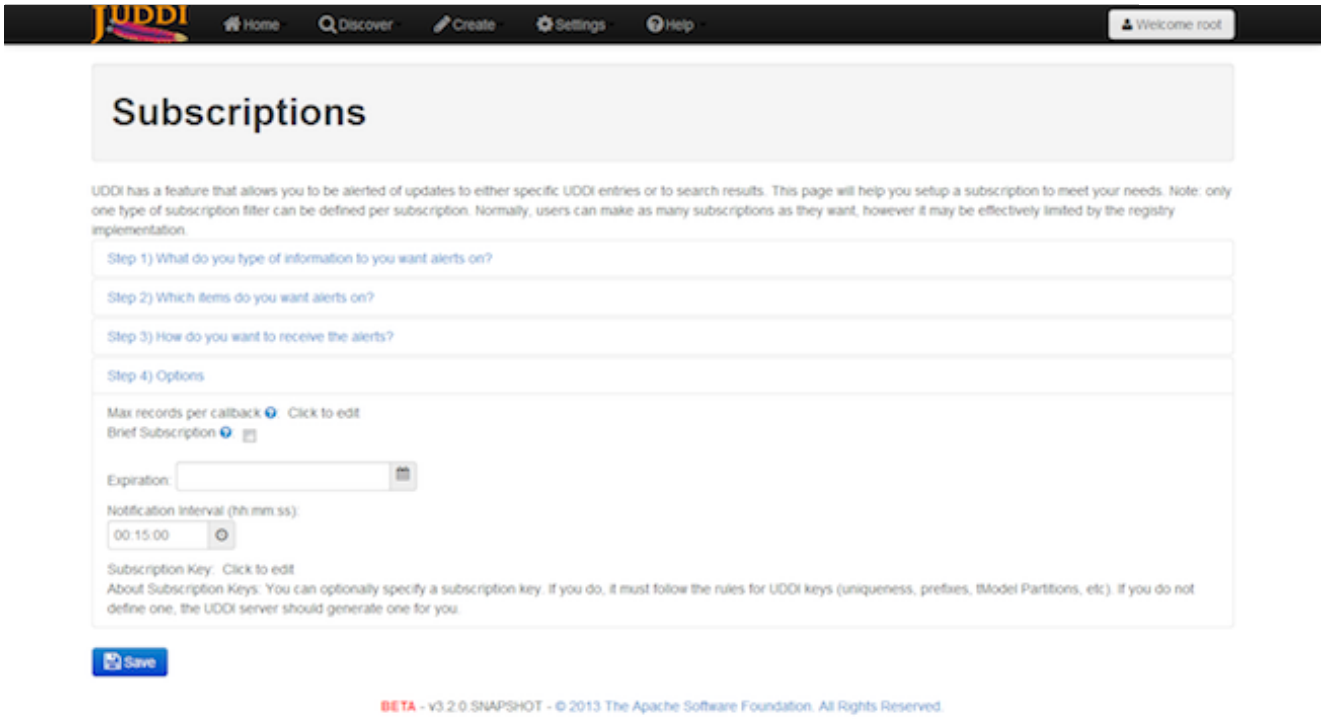


Figure 4.25. Create a Subscription, Options.

4.10.2. View My Subscriptions

To view, edit, and delete existing subscriptions, click on Home, then View Subscriptions.

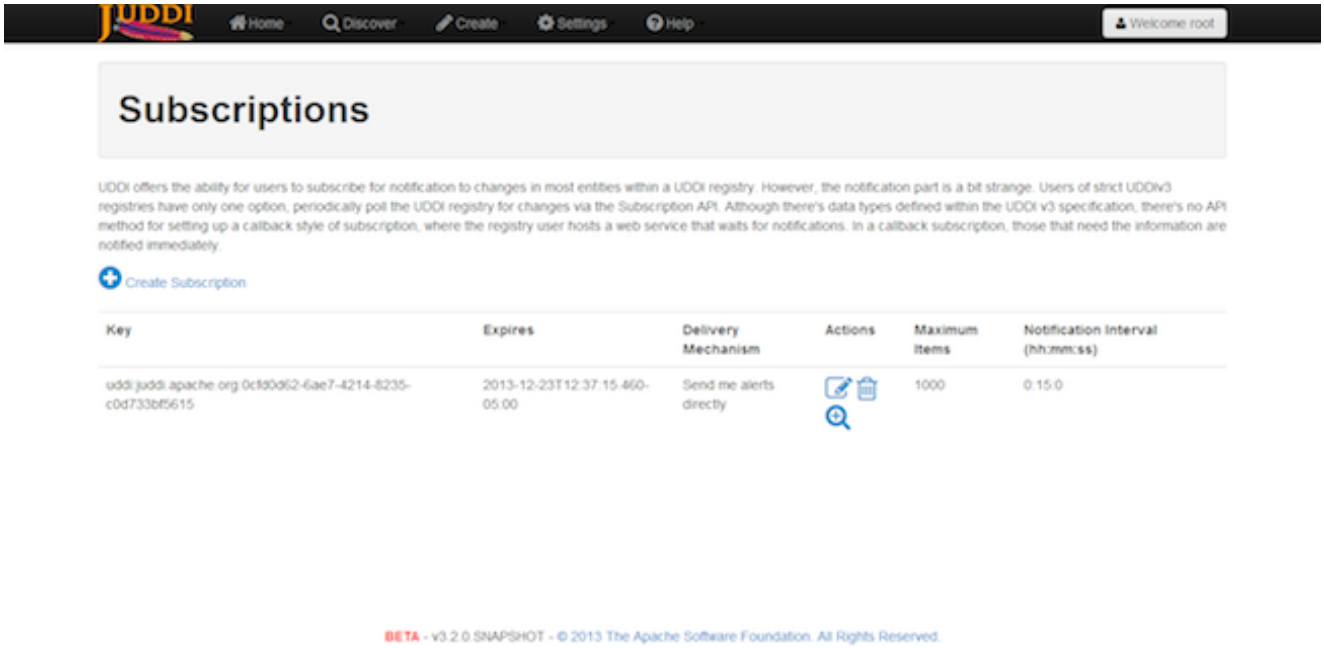


Figure 4.26. View Subscriptions.

View the News Feed

4.10.3. View the News Feed

The New Feed is a simple page designed to show you subscription results for the past 24 hrs. To view the news feed, click on Home, then News Feed.

4.11. Using Digital Signatures in juddi-gui

The juddi-gui makes working with digital signatures simple and enables you do digitally sign and protect entities right from the web browser. It allows you to sign business, services and tModels.

Tip

Digital signatures are performed using the jUDDI Digital Signature Applet which requires a Java plugin for your web browser, as well as a digital certificate (X509).

Tip

You also need to have an X509 Certificate installed in either your Windows My/ Current User Certificate Store or your MacOS Key Chain certificate store.

4.11.1. Sign a Business, Service or tModel

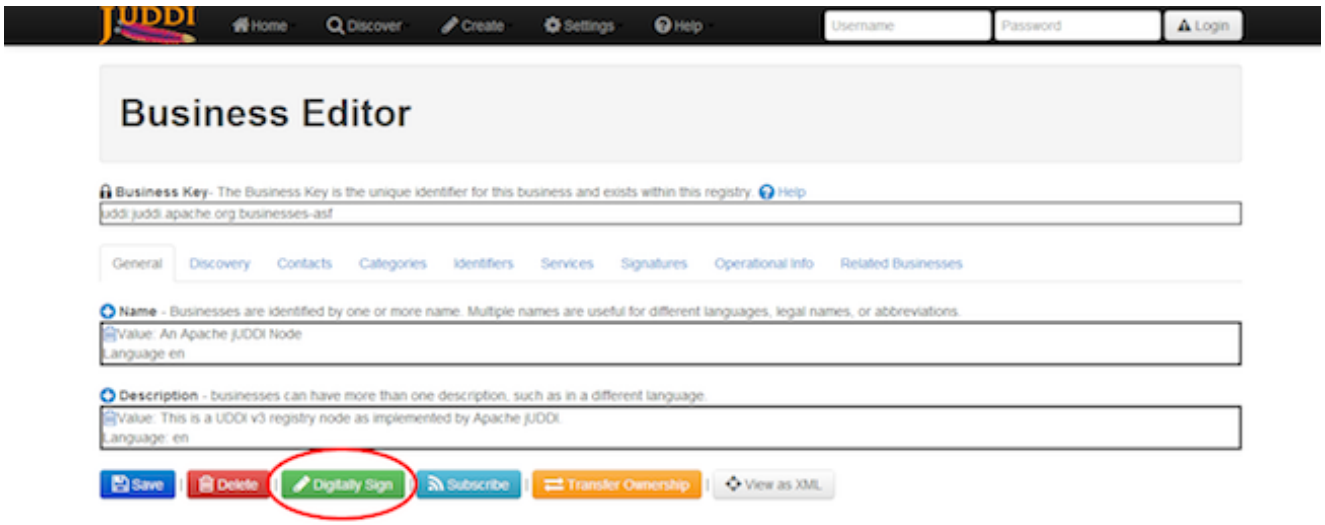


Figure 4.27. Select an entity, then click Digitally Sign.

Sign
a
Business,

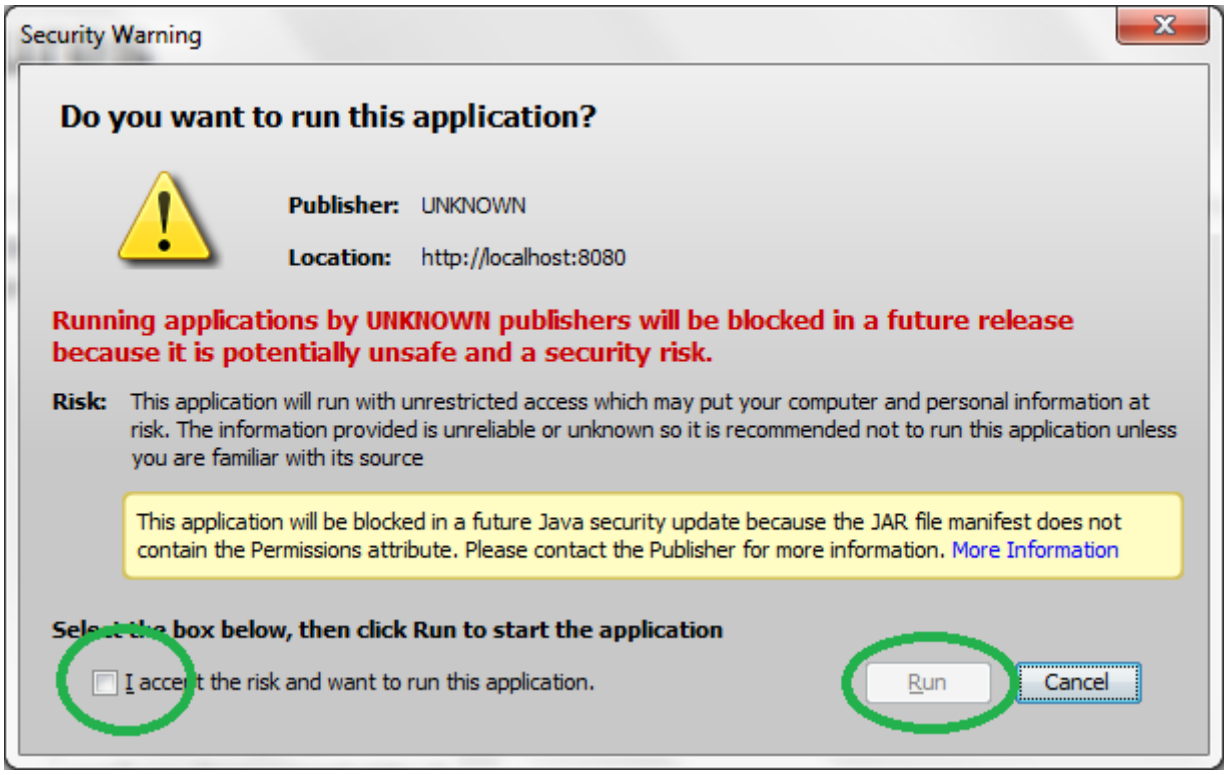


Figure 4.28. Java Plugin Warnings.

Select a certificate, then if you're ready to sign, click on "Digitally Sign". This will automatically generate the signature and save it in the UDDI server.

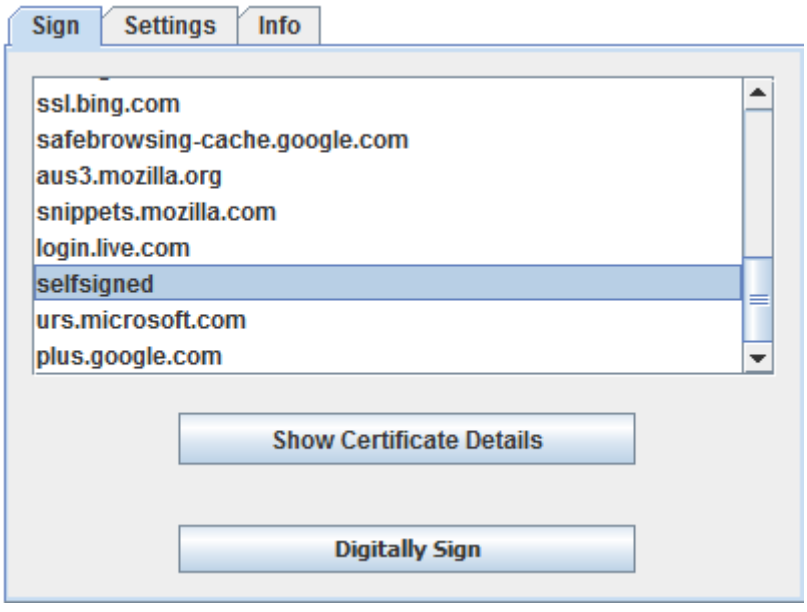


Figure 4.29. Select a Signing Certificate.

Click on Certificate Info will display the following panel. This is useful if you have a few certificates that are similarly named.

Verify a signed

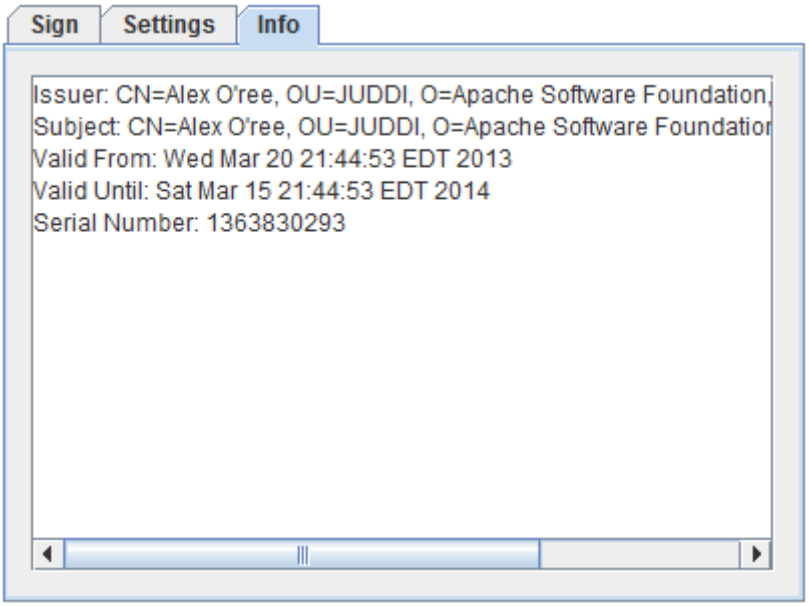


Figure 4.30. Certificate Details.

The settings tab gives you a number of options for advanced users. We recommend that you leave the defaults as is.

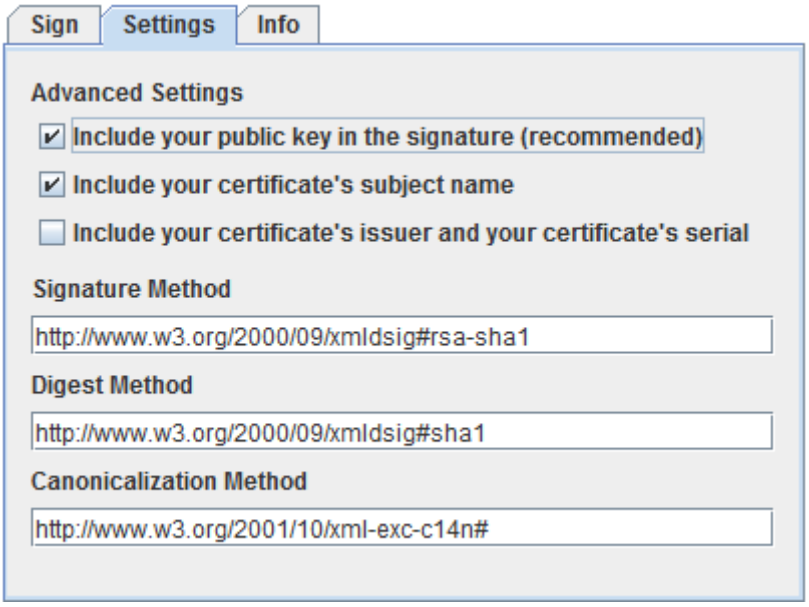


Figure 4.31. Signature Settings.

4.11.2. Verify a signed UDDI entity

Once an entity is signed, the juddi-gui will always attempt to validate the signature and will notify you if its signed and whether or not its valid.

In the following example, the business was signed.

Verify a signed

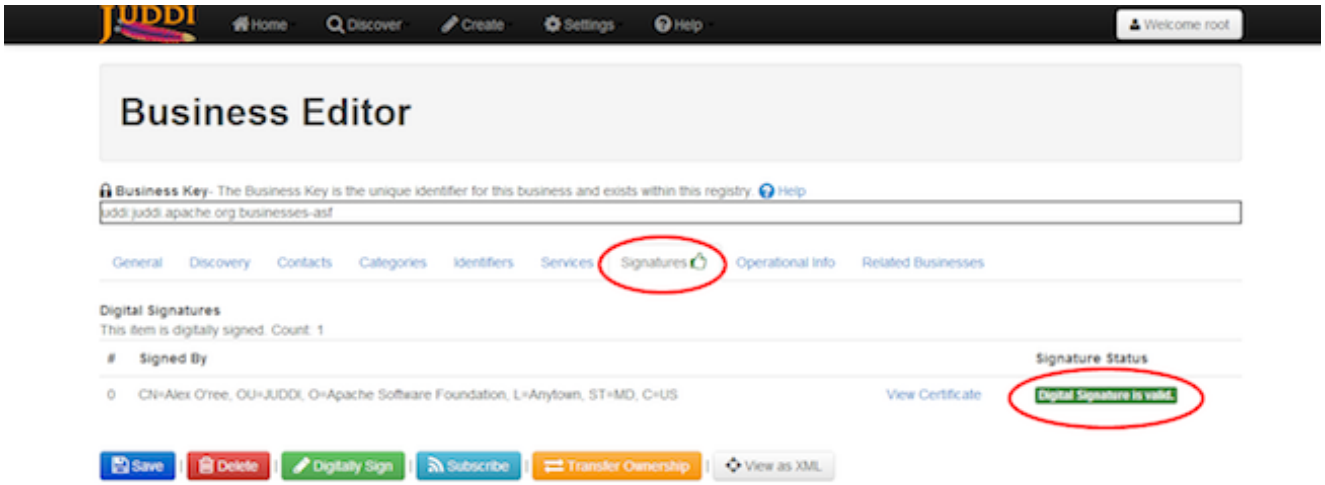


Figure 4.32. Valid Signed Entity.

Important

UDDI entities are hierarchical. A signed business entities includes all of the data for its services and binding templates. Any change to a service or binding template will cause the business's signature to be invalid. TModels are not affected by this.

In the following example, one of the services own by the business was changed. Note that the signature is now invalid due to the alteration.

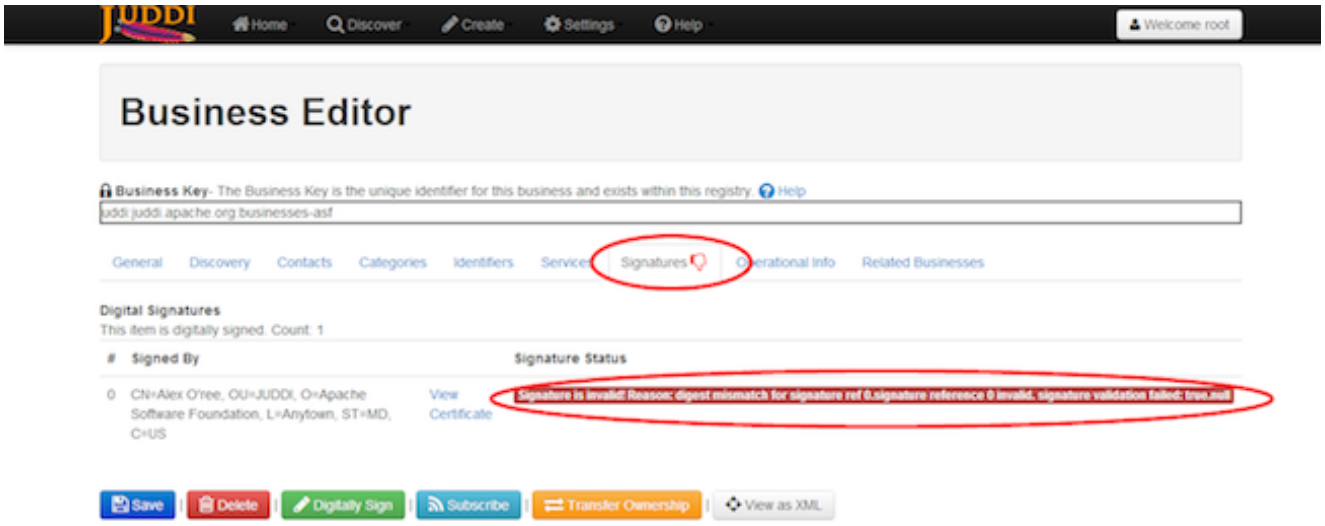


Figure 4.33. Invalid Signed Entity.

4.12. Configuration

The configuration page is usually restricted to system administrators. There are many fields that are displayed. Some of the are editable, others are not. In general, the following settings can be changed (the others are just for troubleshooting and informational purposes).

Details on configuration is located in chapter, jUDDI Server Configuration.

Warning

Saving updates to the console configuration in *juddi-gui/WEB-INF/classes/META-INF/uddi.xml* will only work if the juddi-gui is deployed as a folder.

4.13. Language Codes

The Language Code is a field supported by UDDI that is inherited from the errata for XML Schema, which references RFC 3066, which can be read here: http://www.ietf.org/rfc/rfc3066.txt/. In general, Language Codes are either 2 or 5 characters but can by up to 26 characters. Here's a few examples

- en
- en_US
- es_US

More can be found here: http://www.i18nguy.com/unicode/language-identifiers.html

4.14. Switching Nodes

The jUDDI GUI supports connectivity to multiple UDDI nodes. A UDDI Node is simple a collection of UDDI services that are all connected to the same data source. Another way to put it this, *a UDDI server*. Each browser session to the jUDDI GUI has the ability to select the current Node connection. The current Node select is always saved as a cookie. To avoid any potential confusion, the currently selected Node is available both from the drop down Settings menu, and on the bottom of every page.

To switch nodes, simply select the desired node from the Settings menu.

Important

When switching nodes, any unsaved work that you have will be lost. You will also be logged out of the old node if you were signed in.

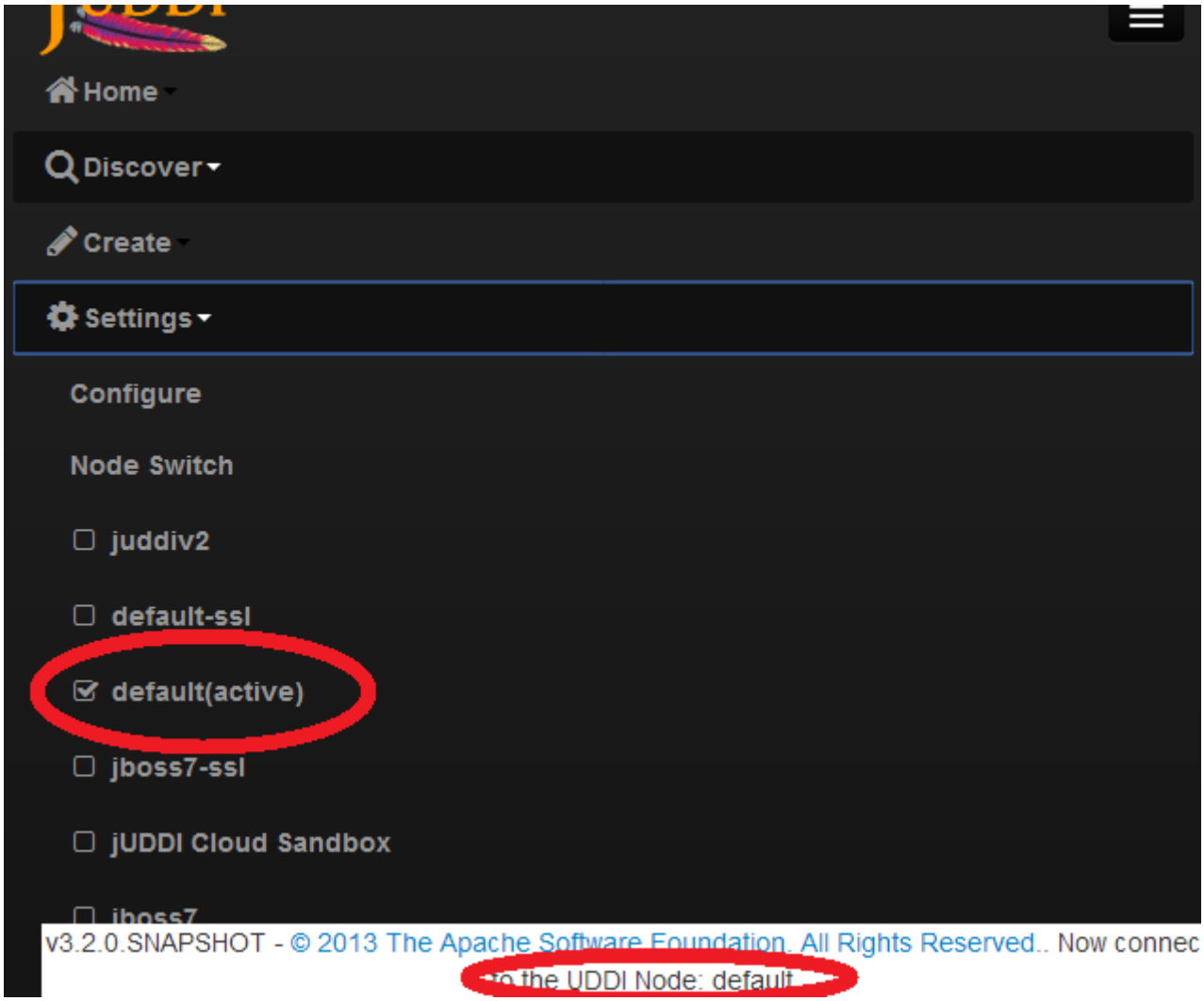


Figure 4.34. Node Switcher.

Tip

Administrators can define the default node via the Setting, Configuration page.

4.15. Adding Additional Language Translations

Adding support for a new translation is relatively simple.

1. Identify what language you want to add (i.e. French)
2. Identify the two letter language code (i.e. fr)
3. Copy the default language resource file "web.properties" and name it "web_fr.properties".

4. Edit "web_fr.properties" and translating each phrase as needed. It’s a standard Java properties file that uses the format key=value.
5. Place this new file within "juddi-gui/WEB-INF/classes/org/apache/juddi/webconsole/resources"
6. Edit the "index.jsp" file in "juddi-gui" to add the new line for the "fr" langauge code. It’s towards the bottom.
7. Restart Tomcat or your application server and test.

Finally, contributions are always welcome. Either open a ticket with our issue tracker and attached the new translation or open a pull request.

Chapter 5. Mapping WSDL and WSDL to UDDI

5.1. Introduction

OASIS published a technical article which describes the recommended way to map the entries from a WSDL (Web Service Description Language) document into UDDI (https://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v202-20040631.htm). The jUDDI-client provides a convenient mechanism for this mapping. In addition, the jUDDI team provides a similar API for mapping a WADL (Web Application Description Language) document to UDDI. This guide will help you use the APIs to register a WSDL or WADL document that describes a service within a UDDI registry.

5.2. Use Case - WSDL

The most basic use case is that we have one or more SOAP/WSDL based services from a 3rd party that was just stood up on our network and we wish to now advertise to our user base that this service exists. We could manually punch in the information, but what fun is that? Let's import it using some code! This can be expanded to import services in bulk.

5.2.1. Sample Code

```
URL url = new URL("http://someURLtoYourWSDL");
ReadWSDL rw = new ReadWSDL();
Definition wsdlDefinition = rw.readWSDL(url);
Properties properties = new Properties();
properties.put("keyDomain", domain);
properties.put("businessName", domain);
properties.put("serverName", url.getHost());
properties.put("serverPort", url.getPort());
wsdlURL = wsdlDefinition.getDocumentBaseURI();
WSDL2UDDI wsdl2UDDI = new WSDL2UDDI(null, new URLLocalizerDefaultImpl(),
 properties);
//This creates a the services from WSDL
BusinessServices businessServices =
 wsdl2UDDI.createBusinessServices(wsdlDefinition);
//This creates the tModels from WSDL
Map<QName, PortType> portTypes = (Map<QName, PortType>)
 wsdlDefinition.getAllPortTypes();
//This creates more tModels from WSDL
Set<TModel> portTypeTModels = wsdl2UDDI.createWSDLPortTypeTModels(wsdlURL,
 portTypes);
Map allBindings = wsdlDefinition.getAllBindings();
//This creates even more tModels from WSDL
```

Links
to
sample

```
Set<TModel> createWSDLBindingTModels =
 wsdl2UDDI.createWSDLBindingTModels(wsdlURL, allBindin

//Now just save the tModels, then add the services to a new or existing
 business
```

5.2.2. Links to sample project

SVN Links to sample projects

- http://svn.apache.org/repos/asf/juddi/trunk/juddi-examples/
- http://svn.apache.org/repos/asf/juddi/trunk/juddi-examples/wsdl2uddi/
- http://svn.apache.org/repos/asf/juddi/trunk/juddi-examples/uddi-samples/

The examples are also available in both jUDDI distributions.

5.3. Use Case - WADL

The most basic use case is that we have one or more REST/WADL based services from a 3rd party that was just stood up on our network and we wish to now advertise to our user base that this service exists. We could manually punch in the information, but what fun is that? Let's import it using some code! This can be expanded to import services in bulk.

5.3.1. Sample Code

```
Application app = WADL2UDDI.ParseWadl(new URL("URL to WADL file"));
List<URL> urls = WADL2UDDI.GetBaseAddresses(app);
URL url = urls.get(0);
String domain = url.getHost();
Properties properties = new Properties();
properties.put("keyDomain", domain);
properties.put("businessName", domain);
properties.put("serverName", url.getHost());
properties.put("serverPort", url.getPort());
WADL2UDDI wadl2UDDI = new WADL2UDDI(null, new URLLocalizerDefaultImpl(),
 properties);

//creates the services
BusinessService businessServices = wadl2UDDI.createBusinessService(new
 QName("MyWasdl.namespace", "Servicename"), app);
//creates tModels (if any)
Set<TModel> portTypeTModels = wadl2UDDI.createWADLPortTypeTModels(wsdlURL,
 app);
//Now just save the tModels, then add the services to a new or existing
 business
```

5.3.2. Links to sample project

SVN Links to sample projects

- http://svn.apache.org/repos/asf/juddi/trunk/juddi-examples/
- http://svn.apache.org/repos/asf/juddi/trunk/juddi-examples/uddi-samples/

The examples are also available in both jUDDI distributions.

Chapter 6. Using UDDI Annotations

Conventionally Services (BusinessService) and their EndPoints (BindingTemplates) are registered to a UDDI Registry using a GUI, where an admin user manually adds the necessary info. This process tends to make the data in the Registry rather static and the data can grow stale over time. To make the data in the UDDI more dynamic it makes sense to register and EndPoint (BindingTemplate) when it comes online, which is when it gets deployed. The UDDI annotations are designed to just that: register a Service when it get deployed to an Application Server. There are two annotations: UDDIService, and UDDIServiceBinding. You need to use both annotations to register an EndPoint. Upon undeployment of the Service, the EndPoint will be de-registered from the UDDI. The Service information stays in the UDDI. It makes sense to leave the Service level information in the Registry since this reflects that the Service is there, however there is no EndPoint at the moment ("Check back later"). It is a manual process to remove the Service information. The annotations use the juddi-client library which means that they can be used to register to any UDDIv3 registry.

6.1. UDDI Service Annotation

The UDDIService annotation is used to register a service under an already existing business in the Registry. The annotation should be added at the class level of the java class.

Table 6.1. UDDIService attributes

attribute	description	required
serviceName	The name of the service, by default the clerk will use the one name specified in the WebService annotation	no
description	Human readable description of the service	yes
serviceKey	UDDI v3 Key of the Service	yes
businessKey	UDDI v3 Key of the Business that should own this Service. The business should exist in the registry at time of registration	yes
lang	Language locale which will be used for the name and description, defaults to "en" if omitted	no
categoryBag	Definition of a CategoryBag, see below for details	no

6.2. UDDIServiceBinding Annotation

The UDDIServiceBinding annotation is used to register a BindingTemplate to the UDDI registry. This annotation cannot be used by itself. It needs to go along side a UDDIService annotation.

Table 6.2. UDDIServiceBinding attributes

attribute	description	required
bindingKey	UDDI v3 Key of the ServiceBinding	yes
description	Human readable description of the service	yes
	accessPointType UDDI v3 AccessPointType, defaults to wsdlDeployment if omitted	no
accessPoint	Endpoint reference	yes
lang	Language locale which will be used for the name and description, defaults to "en" if omitted	no
tModelKeys	Comma-separated list of tModelKeys key references	no
categoryBag	Definition of a CategoryBag, see below for further details	no

6.2.1. Java Web Service Example

The annotations can be used on any class that defines a service. Here they are added to a WebService, a POJO with a JAX-WS *WebService* annotation.

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

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

@UDDIService(
  businessKey="uddi:myBusinessKey",
  serviceKey="uddi:myServiceKey",
  description = "Hello World test service")
@UDDIServiceBinding(
  bindingKey="uddi:myServiceBindingKey",
  description="WSDL endpoint for the helloWorld Service. This service is
 used for "
```

```
      + "testing the jUDDI annotation functionality",
  accessPointType="wsdlDeployment",
  accessPoint="http://localhost:8080/juddiv3-samples/services/helloworld?
wsdl")
@WebService(
  endpointInterface = "org.apache.juddi.samples.HelloWorld",
  serviceName = "HelloWorld")

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

On deployment of this WebService, the juddi-client code will scan this class for UDDI annotations and take care of the registration process. The configuration file uddi.xml of the juddi-client is described in the chapter, Using the jUDDI-Client. In the clerk section you need to reference the Service class *org.apache.juddi.samples.HelloWorldImpl*:

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

which means that Bob is using the node connection setting of the node with name "default", and that he will be using the "sales" publisher, for which the password it "sales". There is some analogy here as to how datasources are defined.

6.2.2. Wiring it all together

The mechanism that triggers the automated registration is the UDDIClient. For each class you annotation, the class needs to be listed in the jUDDI Client Configuration file. When the client reads in the configuration, it will read the uddi.xml config file for the following location:

```
client/clerks/clerk[].class
```

In addition, the following flag must be set to true.

```
client/clerks@registerOnStartup="true"
```

Here's a full example

```
<clerks registerOnStartup="false" >
    <clerk name="default" node="default" publisher="userjoe"
 password="******" cryptoProvider="" isPasswordEncrypted="false">
        <class>com.mybiz.services.Service1</class>
    </clerk>
```

```
</clerks>
```

The next step is to automate the "starting" and "stopping" of UDDIClient. In Java, anything that runs in a servlet container and use the following servlet class:

```
org.apache.juddi.v3.client.config.UDDIClerkServlet
```

It will automatically handle registration on start up and it will remove binding Templates on shutdown (this ensuring clients that discover the endpoint won't find a dead link).

Clients that run elsewhere simply need to "start" the UDDIClient.

```
//start up
UDDIClient clerkManager = new UDDIClient("META-INF/uddi.xml");
// register the clerkManager with the client side container
UDDIClientContainer.addClient(clerkManager);
clerkManager.start(); //will create business/services/bindings as necessary

//shutdown down
clerkManager.stop(); //will unregister binding templates
```

6.3. .NET Web Service Example

In .NET, the procedure is almost identical to Java. Annotate your web service classes, append the classnames to your uddi.xml client config file. .NET annotations work with any WCF, ASP.NET or any other class.

6.3.1. Wiring it all together

In .NET, there's a few options, each with pro's and con's for automating registration.

6.3.1.1. Use UDDIClient in your service's constructor

Pro: It's simple Con: Services often get multiple instances depending on the number of worker threads in the server and thus can cause some concurrency issues.

6.3.1.2. Use UDDIClient in Global.asa Application_Start

Pro: It's simple Con: You need .NET 4.0 and ASP.NET enabled in order to utilize this function

More information about this can be found here: http://weblogs.asp.net/scottgu/archive/2009/09/15/auto-start-asp-net-applications-vs-2010-and-net-4-0-series.aspx

6.4. CategoryBag Attribute

The CategoryBag attribute allows you to reference tModels. For example the following categoryBag

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

can be put in like

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

6.5. Considerations for clustered or load balanced web servers and automated registration

Most production environments have primary and failover web servers and/or an intelligent load balancer that routers traffic to whichever server is online. When using automated registration with the jUDDI client, care must be taken when enabling automated registration.

Chapter 7. Using the UDDI v2 Services and Adapters

7.1. Introduction

Starting with jUDDI version 3.2, a number of adapters are provided to help you use or access UDDI version 2 based services. There are a multitude of options and will be discussed in the following sections.

7.2. Accessing UDDI v2 services using the jUDDI v3 Client

Accessing UDDI v2 services via the jUDDI v3 client is quite simple. All that's required is modification of the uddi.xml client configuration file. Simply set the transport to:

```
org.apache.juddi.v3.client.transport.JAXWSv2TranslationTransport
```

…and adjust the *inquiryUrl* and *publishUrl* URL endpoints.

Tip

When accessing UDDI v2, Custody Transfer, Subscription, Replication and Value Set APIs will not be available and may generate unexpected behavior. The UDDIv3 Inquiry getOperationalInfo method is only partially mapped.

That's it. No code changes are required other than to avoid Custody Transfer, Subscription, Replication and Value Set APIs. In addition, digital signatures are not mapped.

7.3. Accessing UDDI v2 services using UDDI v2 APIs

The jUDDI v3 client now contains the UDDI 2 web service clients. Although, there isn't currently a configuration/transport/client/clerk wrapper for it, you can still get access to web service clients with the following code:

```
org.apache.juddi.v3.client.UDDIServiceV2 svc = new UDDIServiceV2();
Inquire port = svc.getInquire();
((BindingProvider)
 port).getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
 "http://localhost:8080/juddiv3/services/inquiryv2");
Publish pub= svc.getPublish();
```

```
((BindingProvider)
 pub).getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
 "http://localhost:8080/juddiv3/services/publishv2");
```

All you need to reference the following projects/jars from jUDDI:

- juddi-client
- uddi-ws

7.4. Accessing jUDDI v3 services from an existing UDDI v2 based client, plugin or tool

When UDDI v2 was release, many application developers jumped on board to support it. As such, there are many UDDI v2 tools that exist, such as IDE plugins like Eclipse's Web Services Explorer. To support legacy tools, jUDDI now offers UDDI v2 endpoints. Simple point your tool at the following URLs. You'll have to alter them to match your environment.

```
http://localhost:8080/juddiv3/services/inquiryv2
http://localhost:8080/juddiv3/services/publishv2
```

7.5. Additional Information

The UDDI v2 adapters provide basic mappings from to and from UDDI v3. The juddi-client has several mapping functions that are used both client and service side to convert from UDDI v2 to UDDI v3. In addition, the client has as several interface adapters to help with a seamless transition.

Chapter 8. UDDI Migration and Backup Tool

The UDDI Migration and Backup Tool can be used to perform a number of administrative tasks such as

- Backup the contents of a UDDI server (business, services, binding templates and tModels)
- Import contents into a UDDI server (business, services, binding templates and tModels)

In addition, the migration tool has a few features that serve as job aids.

- Ability to remove digital signatures on Import or Export
- Ability to maintain ownership properties of UDDI entries
- Ability to export and import Publishers (jUDDI only)
- Automatically skip an item on import if the entity key already exists

The UDDI Migration and Backup Tool is Command Line Interface program and has a number of use cases such as:

- Copying data from one registry to another
- Migrating from one vendor to another
- Periodic backups
- Upgrades to jUDDI

Tip

The migration tool will not overwrite data when importing.

8.1. Using the tool

There are many configuration options and settings for the migration tool. This tool is distributed with the uddi client distribution package.

8.1.1. Get help

```
>java -jar juddi-migration-tool-3.2.0-SNAPSHOT-jar-with-dependencies.jar
This tool is used to export and import UDDI data from a UDDI v3 registry
```

Use case: basic

```
Random TIP: Without the preserveOwnership flag, all imported data will be
 owned by the username that imported it.

usage: java -jar juddi-migration-tool-(VERSION)-jar-with-dependencies.jar
 -business <arg>      Im/Export option, file to store the business data,
                      default is 'business-export.xml'
 -config <arg>        Use an alternate config file default is 'uddi.xml'
 -credFile <arg>      Import option with -preserveOwnership, this is a
                      properties file mapping with user=pass
 -export              Exports data into a UDDIv3 registry
 -import              Imports data into a UDDIv3 registry
 -isJuddi             Is this a jUDDI registry? If so we can in/export
                      more stuff
 -mappings <arg>      Im/Export option, file that maps keys to owners,
                      default is 'entityusermappings.properties'
 -myItemsOnly         Export option, Only export items owned by yourself
 -node <arg>          The node 'name' in the config, default is 'default'
 -pass <arg>          Password, if not defined, those is uddi.xml will be
                      used
 -preserveOwnership   Im/Export option, saves owership data to the
                      'mappings' file
 -publishers <arg>    jUDDI only - In/Export option, file to store
                      publishers, default is 'publishers-export.xml'
 -stripSignatures     Im/Export option, removes digital signatures from
                      all signed items, default is false
 -tmodel <arg>        Im/Export for tmodels, file to store tmodel data,
                      default is 'tmodel-export.xml'
 -user <arg>          Username, if not defined, those is uddi.xml will be
                      used
```

8.1.2. Use case: basic import and export

To export everything without preserving ownership information:

```
java -jar juddi-migration-tool-(VERSION)-jar-with-dependencies.jar -export
```

To import everything without preserving ownership information:

```
java -jar juddi-migration-tool-(VERSION)-jar-with-dependencies.jar -import
```

8.1.3. Use case: Import and Export while preserving ownership information

To export everything with preserving ownership information:

```
java -jar juddi-migration-tool-(VERSION)-jar-with-dependencies.jar -export -
preserveOwnership
```

Use case: Import and Export while preserving

To import everything with preserving ownership information, first edit the mappings file which is entityusermappings.properties by default. Once every user has a password, run the following command

```
java -jar juddi-migration-tool-(VERSION)-jar-with-dependencies.jar -import -preserveOwnership
```

Tip

When preserving ownership information, upon import, you'll need every UDDI entity owner's password. If you don't have this and you're using jUDDI, you can temporarily switch jUDDI to the *DefaultAuthenticator* which doesn't validate passwords (just put anything in the mappings file for each user). Once the import is complete, you can then switch back to whatever authenticator you were using before.

Chapter 9. Using the jUDDI REST Services

jUDDI includes a Inquiry API adapter that exposes some of the basic functionality of UDDI via REST. Data can be retrieved in both XML and JSON encoding for all methods.

9.1. URL Patterns and methods

1. All jUDDI Inquiry REST service are accessible via HTTP GET.
2. Authentication is not yet supported. This also implies that the Inquiry API must be configured for anonymous access (i.e. do not turn on Inquiry Authentication Required).
3. The jUDDI Inquiry REST service is not currently portable as an adapter to other UDDI instances (but it could be adapted to it in the future)

9.1.1. Endpoints

All endpoints must be prefixed with http(s)://server:port/juddicontext/ where *juddicontext* is typically *juddiv3*.

WADL Document: http://localhost:8080/juddiv3/services/inquiryRest?_wadl

Tip

All of the examples in this document reference JSON encoded messages. To switch to XML messages, just replace the *JSON* with *XML* in the URL. That's it!

9.1.2. Methods

Each method is accessible using the following pattern:

```
http://localhost:8080/juddiv3/services/inquiryRest/{encoding}/{method}/
{parameters}
//or
http://localhost:8080/juddiv3/services/inquiryRest/{encoding}/{method}?
{name=value}
```

Notes

- Encoding - Can be *XML* or *JSON*
- Methods - See below
- Parameters - usually a unique UDDI key

9.1.2.1. xxxList

Returns up to 100 items within a KeyBag object, containing a list of all keys for the given object type.

1. serviceList - http://localhost:8080/juddiv3/services/inquiryRest/JSON/serviceList
2. businessList - http://localhost:8080/juddiv3/services/inquiryRest/JSON/businessList
3. tModelList - http://localhost:8080/juddiv3/services/inquiryRest/JSON/tModelList

9.1.2.2. endpointsByService/key

Returns all executable endpoints for a given service key, including all binding Templates. This also resolves hosting redirector and a number of other accessPoint useType specifics.

Example:

```
http://localhost:8080/juddiv3/services/inquiryRest/JSON/endpointsByService/
uddi:juddi.apache.org:services-custodytransfer
```

9.1.2.3. getDetail

Return the details of a specific item using query parameters. This implements the UDDI recommendation for HTTP GET services for UDDI. See http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908158 for further information.

Example:

```
http://localhost:8080/juddiv3/services/inquiryRest/XML/getDetail?
businessKey=uddi:juddi.apache.org:businesses-asf
```

The following query parameters are supported. Only one can be specified at a time

1. businessKey/(key) - http://localhost:8080/juddiv3/services/inquiryRest/JSON/getDetail?businessKey=uddi:juddi.apache.org:businesses-asf
2. tModelKey/(key) - http://localhost:8080/juddiv3/services/inquiryRest/JSON/getDetail?tModelKey=uddi:uddi.org:categorization:types
3. bindingKey/(key) - http://localhost:8080/juddiv3/services/inquiryRest/JSON/getDetail?bindingKey=uddi:juddi.apache.org:servicebindings-inquiry-ws
4. serviceKey/(key) - http://localhost:8080/juddiv3/services/inquiryRest/JSON/getDetail?serviceKey=uddi:juddi.apache.org:services-inquiry

9.1.2.4. xxxKey

Return the details of a specific item. This is similar to getDetail except that it is not based on query parameters. The underlying code of this function is the same as getDetail.

Example:

1. businessKey/(key) - http://localhost:8080/juddiv3/services/inquiryRest/JSON/businessKey/uddi:juddi.apache.org:businesses-asf
2. tModelKey/(key) - http://localhost:8080/juddiv3/services/inquiryRest/JSON/tModelKey/uddi:uddi.org:categorization:types
3. bindingKey/(key) - http://localhost:8080/juddiv3/services/inquiryRest/JSON/bindingKey/uddi:juddi.apache.org:servicebindings-inquiry-ws
4. serviceKey/(key) - http://localhost:8080/juddiv3/services/inquiryRest/JSON/serviceKey/uddi:juddi.apache.org:services-inquiry
5. opInfo/(key) - http://localhost:8080/juddiv3/services/inquiryRest/JSON/opInfo/uddi:juddi.apache.org:businesses-asf

9.1.2.5. xxxSearch

Returns the search results for registered entities in XML or JSON using a number of query parameters.

Supported entities: . searchService . searchBusiness . searchTModel

Supported query parameters . name - Filters by the name element. If not specified, the wildcard symbol is used *%*. . lang - Filters by language. If not specified, null is used. . findQualifiers - Adds sorting or additional find parameters. comma delimited. If not specified, *approximateMatch* is used . maxrows - Maximum rows returned. If not specified, 100 is used. . offset - Offset for paging operations. If not specified, 0 is used.

9.2. Example Output

9.2.1. XML

The output of all XML encoded messages is identical to the UDDI schema specifications. There should be no surprises.

9.2.2. JSON

The output of JSON encoded messages is obviously different than XML. The following is an example of what it looks like.

```
{
    "businessEntity": {
        "@businessKey": "uddi:juddi.apache.org:businesses-asf",
        "discoveryURLs": {
            "discoveryURL": {
                "@useType": "homepage",
                "$": "http://localhost:8080/juddiv3"
            }
```

```
        },
        "name": {
            "@xml.lang": "en",
            "$": "An Apache jUDDI Node"
        },
        "description": {
            "@xml.lang": "en",
            "$": "This is a UDDI v3 registry node as implemented by Apache
 jUDDI."
        },
        "businessServices": {
            "businessService": [
                {
                    "@serviceKey": "uddi:juddi.apache.org:services-
custodytransfer",
                    "@businessKey": "uddi:juddi.apache.org:businesses-asf",
                    "name": {
                        "@xml.lang": "en",
                        "$": "UDDI Custody and Ownership Transfer Service"
                    },
                    "description": {
                        "@xml.lang": "en",
                        "$": "Web Service supporting UDDI Custody and
 Ownership Transfer API"
                    },
                    "bindingTemplates": {
                        "bindingTemplate": [
                            {
                                "@bindingKey":
 "uddi:juddi.apache.org:servicebindings-custodytransfer-ws",
                                "@serviceKey":
 "uddi:juddi.apache.org:services-custodytransfer",
                                "description": "UDDI Custody and Ownership
 Transfer API V3",
                                "accessPoint": {
                                    "@useType": "wsdlDeployment",
                                    "$": "http://localhost:8080/juddiv3/
services/custody-transfer?wsdl"
                                },
                                "tModelInstanceDetails": {
                                    "tModelInstanceInfo": {
                                        "@tModelKey":
 "uddi:uddi.org:v3_ownership_transfer",
                                        "instanceDetails": {
                                            "instanceParms": "\n
      \n                <?xml version=\"1.0\" encoding=\"utf-8\" ?>\n
           <UDDIinstanceParmsContainer\n                 xmlns=\"urn:uddi-
org:policy_v3_instanceParms\">\n                  <authInfoUse>required</
```

```
authInfoUse>\n                </UDDIinstanceParmsContainer>\n
 \n                "
                                        }
                                    }
                                },
                                "categoryBag": {
                                    "keyedReference": {
                                        "@tModelKey":
 "uddi:uddi.org:categorization:types",
                                        "@keyName": "uddi-org:types:wsdl",
                                        "@keyValue": "wsdlDeployment"
                                    }
                                }
                            },
                            {
                                "@bindingKey":
 "uddi:juddi.apache.org:servicebindings-custodytransfer-ws-ssl",
                                "@serviceKey":
 "uddi:juddi.apache.org:services-custodytransfer",
                                "description": "UDDI Custody and Ownership
 Transfer API V3 SSL",
                                "accessPoint": {
                                    "@useType": "wsdlDeployment",
                                    "$": "https://localhost:8443/juddiv3/
services/custody-transfer?wsdl"
                                },
                                "tModelInstanceDetails": {
                                    "tModelInstanceInfo": [
                                        {
                                            "@tModelKey":
 "uddi:uddi.org:v3_ownership_transfer",
                                            "instanceDetails": {
                                                "instanceParms": "\n
        \n                <?xml version=\"1.0\" encoding=\"utf-8\" ?>\n
            <UDDIinstanceParmsContainer\n                xmlns=\"urn:uddi-
org:policy_v3_instanceParms\">\n                  <authInfoUse>required</
authInfoUse>\n                </UDDIinstanceParmsContainer>\n
 \n                "
                                            }
                                        },
                                        {
                                            "@tModelKey":
 "uddi:uddi.org:protocol:serverauthenticatedssl3"
                                        }
                                    ]
                                },
                                "categoryBag": {
                                    "keyedReference": {
```

```
                                        "@tModelKey":
  "uddi:uddi.org:categorization:types",
                                        "@keyName": "uddi-org:types:wsdl",
                                        "@keyValue": "wsdlDeployment"
                                    }
                                }
                            }
                        ]
                    }
                }
            ]
        },
        "categoryBag": {
            "keyedReference": {
                "@tModelKey": "uddi:uddi.org:categorization:nodes",
                "@keyName": "",
                "@keyValue": "node"
            }
        }
    }
}
```

9.3. More information

For more information, please check out the source code: http://svn.apache.org/repos/asf/juddi/trunk/juddi-rest-cxf/

Chapter 10. jUDDI Client for NET

Since 3.2, the majority of the functions in the jUDDI Client for Java have been ported to .NET. This guide will show you how to use it and integrate it with your own .NET based applications.

10.1. Procedure

1. Add a reference to jUDDI-Client.NET.dll
2. Add a reference to System.Web.Services
3. Add a reference to System.ServiceModel
4. Add a reference to System.Xml
5. Add a reference to System.Runtime.Serialization
6. Add a reference to System.Configuration
7. Add a reference to System.Security
8. Add a copy of the sample uddi.xml file. Modify it to meet your environment and operational needs.
9. Note, many of the settings are identical to the Java jUDDI-client. The APIs are also nearly identical, so example code should be easily portable from one language to another.

Sample Code

```
/*
 * Copyright 2001-2013 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 */
using org.apache.juddi.v3.client;
using org.apache.juddi.v3.client.config;
```

```
using org.apache.juddi.v3.client.transport;
using org.uddi.apiv3;
using System;
using System.Collections.Generic;
using System.Text;


namespace juddi_client.net_sample
{
    class Program
    {
        static void Main(string[] args)
        {
            UDDIClient clerkManager = new UDDIClient("uddi.xml");
            UDDIClientContainer.addClient(clerkManager);
            Transport transport = clerkManager.getTransport("default");

            org.uddi.apiv3.UDDI_Security_SoapBinding security =
 transport.getUDDISecurityService();
            org.uddi.apiv3.UDDI_Inquiry_SoapBinding inquiry =
 transport.getUDDIInquiryService();

            UDDIClerk clerk = clerkManager.getClerk("default");


            find_business fb = new find_business();
            fb.authInfo = clerk.getAuthToken(security.Url);
            fb.findQualifiers = new string[]
 { UDDIConstants.APPROXIMATE_MATCH };
            fb.name = new name[1];
            fb.name[0] = new name(UDDIConstants.WILDCARD, "en");
            businessList bl = inquiry.find_business(fb);
            for (int i = 0; i < bl.businessInfos.Length; i++)
            {
                Console.WriteLine(bl.businessInfos[i].name[0].Value);
            }
            Console.Read();

        }
    }
}
```

The sample code above should print out a list of all businesses currently registered in the registry. If credentials are stored in the uddi.xml file and are encrypted, they will be decrypted automatically for you.

Within the jUDDI Source Tree, there are many different examples of how to use the jUDDI Client for .NET. They are available here: http://svn.apache.org/repos/asf/juddi/trunk/juddi-client.net/juddi-client.net-sample/

Chapter 11. Using the UDDI Technology Compatibility Kit

Since UDDI is a specification with many complex rules in it, we (the jUDDI team) have had to write test cases to exercise each of the rules and restrictions in UDDI. Knowing that there are a number of open source and commercial UDDI v3 implementations, the jUDDI team took this as an opportunity to create a reusable benchmark for testing the compatibility of UDDI v3 implementations.

Important

Although the TCK covers a large number of test cases, the UDDI specification is long and complex. It's more than possible that we missed a few scenarios or test cases. If you run across any, please let us know.

11.1. Using the TCK Runner

The TCK Runner requires a few files to operate:

1. juddi-tck-runner-version.jar - This is the executable
2. uddi.xml - This file sets the location of the UDDI services
3. tck.properties - This file controls what tests are ran.
4. truststore and keystore.jks - These files are for digital signature tests

11.1.1. Configuration

- Edit the uddi.xml file and update all of the UDDI endpoint locations for all supported endpoints of UDDI server being tested. Ignore all credentials and other settings
- Edit tck.properties and update the usernames and passwords of the test users. Enable or disable tests based on the whether or not the UDDI server supports the optional listed capabilities.

Tip

Do not use usernames and passwords that already have data associated with it.

A few of the test cases, such as RMI transport, are not identified by the UDDI specification, therefore the results may be skewed if unsupported tests are attempted. In addition, the UDDI specification identifies a number of APIs and features that are considered optional.

Although it is possible to run the TCK against a UDDIv2 registry using the UDDIv2 transport adapters, this is not supported. The TCK's test cases and rules apply to the business rules defined in UDDIv3. Unsupported and unmapped functions defined in UDDIv3 that are not supported in UDDIv2 fail ultimately fail.

11.1.1.1. tck.properties

The TCK properties file contains settings for all of the TCK tests.

1. Credentials - You'll need credentials for a number of user accounts
2. jUDDI optional tests - If you're running the tests against jUDDI, there's a number of additional tests ran to exercise things like user accounts.
3. Load testing - These settings enable you to tweak or disable the load testing.
4. Key stores - These are needed to run the digital signature tests
5. Supported transports - jUDDI supports a number of transports, such as RMI and HTTP (for UDDI service interaction) and SMTP and HTTP for subscription callbacks. RMI is actually not in the spec and SMTP is considered optional, so you'll want to adjusted these based on the available documentation from the vendor.

11.1.1.2. uddi.xml

The only parts used from uddi.xml are the following

1. The endpoints of the UDDI services
2. The client subscription callback settings

11.1.2. Running the TCK Runner

Executing the TCK runner is simple.

```
java (options) -Duddi.client.xml=uddi.xml -jar juddi-tck-runner-{VERSION}-
SNAPSHOT-jar-with-dependencies.jar
```

Optional parameters

- -Ddebug=true - this turns up the logging output, typically including the XML payloads of each message.

- -Duddi.client.xml=uddi.xml - Use this file as the jUDDI Client config file. This specifies where all of the UDDI endpoints are.
- -Dtck.properties=file.properties - Use this to use an alternate tck properties file.

11.2. Analyzing the Results

There are two ways to identify the result of the tests.

- Analyze the console output
- Review the test results in uddi-tck-results-[DateTime].txt

The results are summarized in the uddi-tck-results file along with the specific error conditions and stack traces that will enable you to find out the root cause of the failure. It may be necessary to obtain UDDI server logs to help with root cause identification.

Index