

jUDDI 3.0 - Developer Guide

Developer Guide

ASF-JUDDI-DEVGUIDE-11/02/09

Contents

Table of Contents

Contents.....	2	Dev Environment Setup.....	5
About This Guide.....	3	Introduction.....	5
What This Guide Contains.....	3	Building the project.....	5
Audience.....	3	Setting up Eclipse.....	7
Prerequisites.....	3	Building the JAR.....	8
Organization.....	3	Building the WAR.....	9
Documentation Conventions.....	3	Building the Tomcat Bundle.....	10
Additional Documentation.....	4	Running and Developing tests.....	12
Contacting Us.....	4	Index.....	13



About This Guide

What This Guide Contains

The Developer Guide document describes

Audience

This guide is most relevant to engineers who are responsible for setting up jUDDI 3.0 - Developer Guide installations.

Prerequisites

None.

Organization

This guide contains the following chapters:

- **Chapter 1, Dev Environment Setup**
- **Chapter 2, Testing**
- **Chapter 3,**

Documentation Conventions

The following conventions are used in this guide:

Convention	Description
<i>Italic</i>	In paragraph text, italic identifies the titles of documents that are being referenced. When used in conjunction with the Code text described below, italics identify a variable that should be replaced by the user with an actual value.
Bold	Emphasizes items of particular importance.
Code	Text that represents programming code.
Function Function	A path to a function or dialog box within an interface. For example, "Select File Open." indicates that you should select the Open function from the File menu.
() and	<p>Parentheses enclose optional items in command syntax. The vertical bar separates syntax items in a list of choices. For example, any of the following three items can be entered in this syntax:</p> <pre>persistPolicy (Never OnTimer OnUpdate NoMoreOftenThan)</pre>
Note:	A note highlights important supplemental information.
Caution:	A caution highlights procedures or information that is necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

Table 1 Formatting Conventions

Additional Documentation

None on the subject.

Contacting Us

Email: juddi-dev@ws.apache.org

Dev Environment Setup

Introduction

Within jUDDI source, there are the following modules:

uddi-ws - JAXWS stubs built from the wsdl

juddi-core - the jUDDI jar containing the model, api and core jUDDI functionality

juddi-cxf - a WAR module that uses CXF as the web service framework, chosen by default

juddi-axis - a WAR module that uses Axis 2 as the web service framework, this is an alternate to using CXF

juddi-tomcat - a module which builds a Tomcat bundle with juddi-cxf installed and derby as a backend database

juddi-cargo - a module to help test jUDDI web services (currently no tests included)

jUDDI v3 is set up to produce a number of different deliverables – a JAR, a WAR, and a tomcat bundle. Depending on the scope of your application, or your interest in the project, you might want to use the Tomcat server bundle packaged with the Derby database and jUDDI, or you may just want to use the jUDDI JAR and make your own database and Web Service choices. jUDDI is set up so that it can support a range of environments.

Building the project

First, check out the jUDDI sources:

```
% svn co http://svn.apache.org/repos/asf/webservices/juddi/branches/v3\_trunk
```

We suggest using a settings.xml to set your persistence choice on a permanent basis, otherwise you will have to provide it on the command line every time you build.

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
    http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <activeProfiles>
    <activeProfile>hibernate</activeProfile>
  </activeProfiles>
  <profiles>
    <profile>
      <id>hibernate</id>
      <properties>
        <persistence>hibernate</persistence>
      </properties>
    </profile>
  </profiles>
</settings>
```

Then build the entire project:

```
% cd v3_trunk  
% mvn clean install
```

Setting up Eclipse

In order to setup eclipse, first set the M2_REPO property within your Eclipse instance:

Choose “Eclipse” -> “Preferences”

In the preference dialog, select “Java” -> “Build Path” -> “Classpath Variables”

Add a new classpath variable :

Name: M2_REPO

Path : /<path-to-.m2>/m2 (example : /home/tcunning/.m2)

```
% cd v3_trunk
% mvn eclipse:eclipse -Declipse.workspace=/<path-to-workspace>/workspace
```

Then within Eclipse, “Create New Project” and choose “Create from existing source” and choose the source folder that you just checked out from SVN.

Building the JAR

The juddi-core module produces a JAR which contains the jUDDI source and a jUDDI persistence.xml configuration. jUDDI is currently setup so that you can choose between using either OpenJPA or Hibernate as your persistence framework. The juddi-core pom.xml contains two profiles, triggered on the "persistence" property.

OpenJPA

```
% cd juddi-core
% mvn -Dpersistence=openjpa
```

Hibernate

```
% cd juddi-core
% mvn -Dpersistence=hibernate
```

Currently the parent POM uses hibernate as the default persistence layer, but this can be easily changed by modifying the persistence property that is specified in the parent pom.xml.

By default, the project is currently using hibernate as the persistence layer. In the v3trunk, the pom.xml sets the persistence property :

v3trunk pom.xml

```
...cut...
<properties>
  <persistence>hibernate</persistence>
</properties>
...cut...
```


Building the WAR

As with the JAR, you need to make a decision on what framework you would like to use when building the WAR. The `juddi-cxf` module builds a war that uses CXF as a web service framework, and the `juddi-axis` target builds a war that uses Axis 2 as a web service framework.

There are currently some issues with the Axis 2 web service framework in `juddi-axis`.

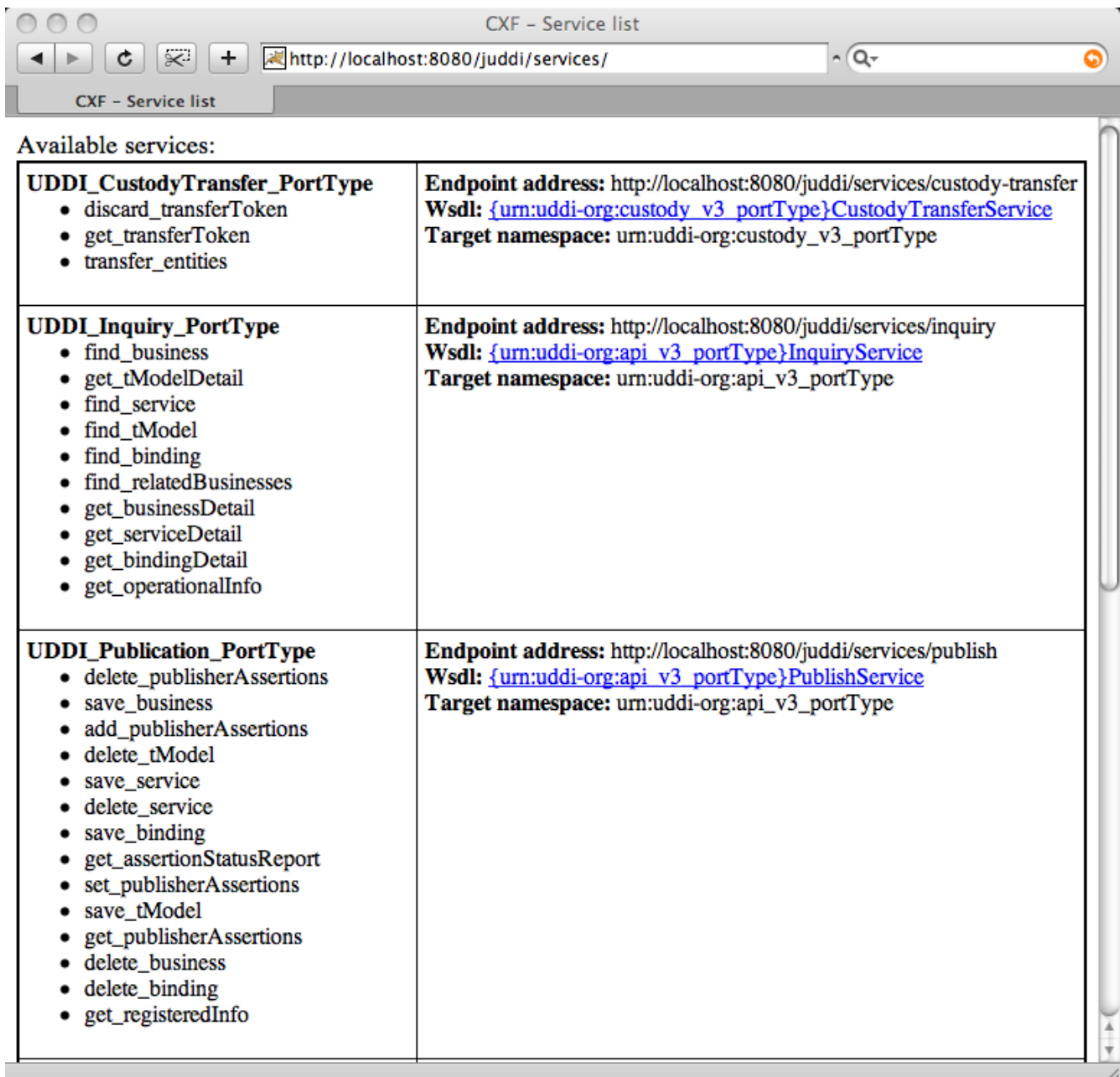
Building the Tomcat Bundle

The jUDDI Tomcat bundle packages up one of the jUDDI WAR files, Apache Derby, and a few necessary configuration files and provides the user with a pre-configured jUDDI instance. By default, the WAR produced by the juddi-cxf module is used – the example below shown uses URLs and endpoints using the jUDDI CXF configuration. If you use the Axis 2 configuration, URLs and endpoints may differ.

To get started using the Tomcat bundle, unzip the juddi-tomcat-bundle.zip, and start Tomcat :

```
% cd apache-tomcat-5.5.23/bin
% ./startup.sh
```

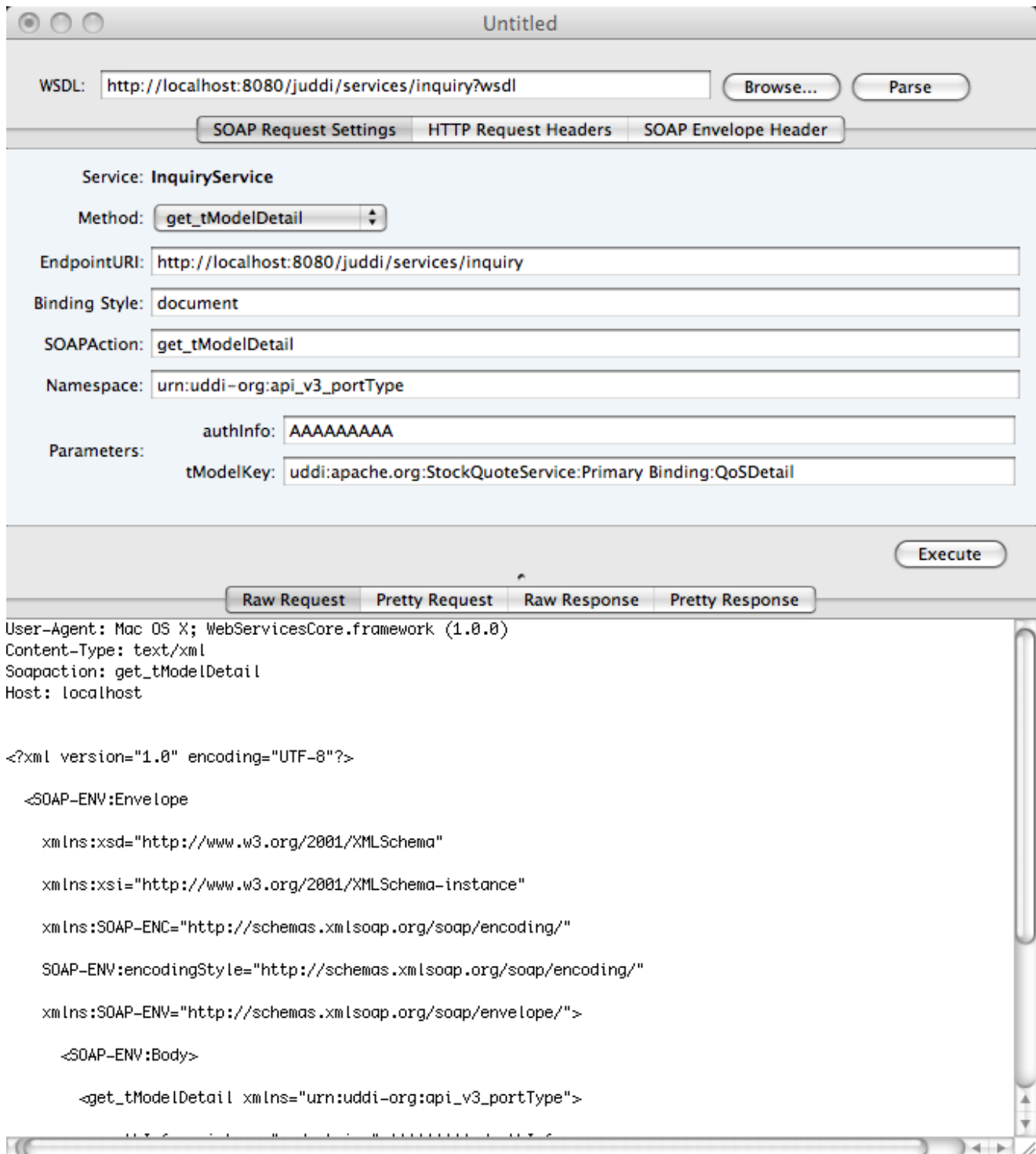
Browse to <http://localhost:8080/juddi/services>



Available services:

UDDI_CustodyTransfer_PortType <ul style="list-style-type: none">• discard_transferToken• get_transferToken• transfer_entities	Endpoint address: http://localhost:8080/juddi/services/custody-transfer WSDL: {urn:uddi-org:custody_v3_portType}CustodyTransferService Target namespace: urn:uddi-org:custody_v3_portType
UDDI_Inquiry_PortType <ul style="list-style-type: none">• find_business• get_tModelDetail• find_service• find_tModel• find_binding• find_relatedBusinesses• get_businessDetail• get_serviceDetail• get_bindingDetail• get_operationalInfo	Endpoint address: http://localhost:8080/juddi/services/inquiry WSDL: {urn:uddi-org:api_v3_portType}InquiryService Target namespace: urn:uddi-org:api_v3_portType
UDDI_Publication_PortType <ul style="list-style-type: none">• delete_publisherAssertions• save_business• add_publisherAssertions• delete_tModel• save_service• delete_service• save_binding• get_assertionStatusReport• set_publisherAssertions• save_tModel• get_publisherAssertions• delete_business• delete_binding• get_registeredInfo	Endpoint address: http://localhost:8080/juddi/services/publish WSDL: {urn:uddi-org:api_v3_portType}PublishService Target namespace: urn:uddi-org:api_v3_portType

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



Running and Developing tests

Currently the only unit tests are in juddi-core. We plan to add a suite of web service tests automated against the juddi-cargo module.

Running the tests:

```
% cd juddi-core
% mvn -Dpersistence=hibernate test
```

The tests are run through a maven-surefire-plugin within the juddi-core pom.xml :

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.4.2</version>
  <configuration>
    <suiteXmlFiles>
      <suiteXmlFile>src/test/resources/suite-
init.xml,src/test/resources/suite-subscribe.xml,src/test/resources/suite-
clean.xml</suiteXmlFile>
    </suiteXmlFiles>
  </configuration>
</plugin>
```

The NUnit suite files listed here determine what tests are run with what data, and what order they are run in. suite-init.xml initializes the jUDDI database with data, suite-subscribe.xml runs a subscription test, and suite-clean.xml cleans the database and removes the test data.

To develop your own tests, please add another maven-surefire-plugin segment and the same ordering of XML files (suite-init.xml, your custom suite, and then suite-clean.xml).



Index
