Introduction to Web Services Practical

This practical will guide you through the creation and deployment of a Java web service and client. We will be using the following software packages:

- **Java 2 Enterprise Edition (J2EE) 1.4** (http://java.sun.com/j2ee/index.jsp)
- **Java Web Services Developer Pack (JWSDP) 1.1** (http://java.sun.com/web services/jwsdp/index.jsp)
- **Tomcat-JWSDP** (http://java.sun.com/web services/containers/tomcat_for_JWSDP_1_4.html)

These software packages should already be installed on your tutorial machines. If not, or you are installing all the software on your own machine, please ensure that you install the JWSDP package **last**, and that when asked to select an application server you make sure that the JWSDP version of Tomcat that you have installed is in the list and selected (if not, use the browse button).

Contents

Environment

Developing and using Java web services requires a relatively complex set of environment variables. For occasional developers, it is convenient to set these variables in a batch file that you run every time you open a command-prompt window. Alternatively, these settings can be permanently added to the command-prompt shortcut. Here is what you should add to a “classpath.bat” file in your tutorial directory (**note that only “set” commands are in this listing, any line that doesn’t start with “set” is a continuation of the previous line**):

```
set WS_HOME=\Sun
set JWSDP_HOME=\tomcat-jwsdp-1.4

set PATH=%WS_HOME%\Appserver\bin;
%WS_HOME%\Appserver\jdk\jre;
%WS_HOME%\Appserver\jdk\bin;
%JWSDP_HOME%\apache-ant;
%JWSDP_HOME%\apache-ant\bin;
%JWSDP_HOME%\jwsdp-shared\bin;
%JWSDP_HOME%\bin;
%JWSDP_HOME%\jaxr\bin;
%PATH%

set JAVA_HOME=%WS_HOME%\Appserver\jdk
set J2EE_HOME=%WS_HOME%\Appserver

set JAXRPC_HOME=%JWSDP_HOME%\jaxrpc
set JAXR_HOME=%JWSDP_HOME%\jaxr
set ANT_HOME=%JWSDP_HOME%\apache-ant

set JAVA_XML_HOME=%JWSDP_HOME%
set JAXM_HOME=%J2EE_HOME%\imq\demo
set JAXM_LIB=%J2EE_HOME%\imq\lib
set JAXP_HOME=%JWSDP_HOME%\jaxp
set JAXP_LIB=%JWSDP_HOME%\jaxp\lib
set JAXRPC_LIB=%JWSDP_HOME%\jaxrpc\lib
set
XERCES_JAR=%JAXP_HOME%\lib\endorsed\xercesImpl.jar
set JSSE_HOME=%J2EE_HOME%\jdk\jre\lib
set SAAJ_HOME=%JWSDP_HOME%\saaj

set
CLASSPATH=.;%WS_HOME%\Appserver\jdk\lib\tools.jar;
%WS_HOME%\Appserver\jdk\lib\rt.jar;
```

```
     %JAVA_HOME%\jre\javaws\javaws.jar;
     %J2EE_HOME%\lib\activation.jar;
     %JWSDP_HOME%\jwsdp-shared\lib\mail.jar;
     %JAXRPC_HOME%;
     %JAXR_HOME%;
     %JAXRPC_HOME%\lib\jaxrpc-api.jar;
     %JAXRPC_HOME%\lib\jaxrpc-impl.jar;
     %JAXRPC_HOME%\lib\jaxrpc-spi.jar;
     %SAAJ_HOME%\lib\saaj-api.jar;
     %SAAJ_HOME%\lib\saaj-impl.jar;
     %JAXP_HOME%;
     %JAXP_LIB%\jaxp-api.jar;
     %JAXP_LIB%\endorsed\xalan.jar;
     %JAXP_LIB%\endorsed\sax.jar;
     %JAXP_LIB%\endorsed\dom.jar;
     %J2EE_HOME%\lib\j2ee.jar;
     %XERCES_JAR%
```

This environment setup is specific to the application server and tool kit that you are using, although any Java-based web services setup will likely suffer from the same problem. If you get errors during compilation or deployment, the first thing to check is usually that your CLASSPATH variable is correctly configured in the command prompt that you are currently using.

Service

The first part of this tutorial will show you how to construct and deploy a web service using Sun’s Java Web Services Developer Pack (JWSDP) and the JWSDP version of the Tomcat web application server.

Setup directory structure

In your home directory, create a "tutorial" folder. Inside this folder, create a "service" folder. This will be the base folder for the remainder of this tutorial, and all relative paths will assume that you are somewhere within this folder.
Create the following directories:

```
          src/
          war/
```

Write interface and implementation .java files

Create a "qotd" folder inside your "src/" directory. Firstly, we have to create a Java interface class for the service. Open a new text file, called "Qotd.java". The first thing we have to do is specify the package and import the classes that we need:

```
     package qotd; // This is just to keep the code neat

     // This class enables remote method invocation
     import java.rmi.Remote;

     // This is for handling remote exceptions
     import java.rmi.RemoreException;
```

Now we must define the interface itself. For our example service, this is very simple indeed:

```
     public interface Qotd extends Remote {
          public String getQuote() throws
     RemoteException;
     }
```

That's it for the interface. Save this file. Now, we must write the class that implements the functionality that we have just described in the interface. Open another new text file, called "Qotd_Impl.java". This file is also very short:

```
     package qotd; // Same package as our interface

     public class Qotd_Impl implements Qotd {
          private static String quote = "Your quote";

          public String getQuote() {
                return this.quote;
            }
        }
```
Save this file. That is all of the programming that we need to do for this (rather trivial) web service.

Create the WAR structure

The various files that we have created so far now have to be bundled up into a WAR file, which is a JAR file with a specific folder structure so that web application servers can deploy the software automatically.

First, create a "\service\war" folder. Then, create a "WEB-INF" subfolder (note the capitalisation - very important).

- WEB-INF\
- WEB-INF\classes\

Compile the code

Compile the two .java files by running this command in the "\src" folder:

```
javac -d ..\war\WEB-INF\classes\ qotd\*.java
```

The "-d" flag and path tells the compiler to place the compiled ".class" files in the appropriate directory of our war file structure.

Write the "WEB-INF\jaxrpc-ri.xml" file

The jaxrpc-ri.xml file tells the wsdeploy utility how to generate the various pieces of code and configuration information in order for the portable WAR to be converted and deployed on a specific web application server.

Open a new text file called jaxrpc-ri.xml under your WEB-INF folder. Enter the following XML:

```
<webServices
       xmlns="http://java.sun.com/xml/ns/jax-
rpc/ri/dd"
       version="1.0"

targetNamespaceBase="http://nesc.ed.ac.uk/wsdl"

typeNamespaceBase="http://nesc.ed.ac.uk/types"
       urlPatternBase="/ws">
       <endpoint
           name="Qotd"
           displayName="Quote Of The Day Service"
           description="A simple web service"
```

```
                interface="qotd.Qotd"
                implementation="qotd.Qotd_Impl"/>
            <endpointMapping
                endpointName="Qotd"
                urlPattern="/qotd"/>
     </webServices>
```

This tells wsdeploy about the namespaces that we want to use, and defines a web service endpoint in terms of the Java interface that it exposes, and an endpoint mapping which tells the application server how to translate the endpoint into a URL.

Write the "WEB-INF\web.xml" file

Next we have to write the web.xml file which is used by the application server to setup our web service.

Open a new text file under your WEB-INF folder, called "web.xml", and enter the following XML:

```
     <?xml version="1.0" encoding="UTF-8" ?>

     <!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
     Inc.//DTD Web Application
     2.3//EN" "http://java.sun.com/j2ee/dtds/web-
     app_2_3.dtd">

     <web-app>
             <display-name>Quote of the day
     service</display-name>
             <description>Returns some philosophical
     dribble</description>
     </web-app>
```

This simply tells the application server what to call our service, and the description of what it does, both in a user-readable form (this is used primarily for the management interface).

Package it all up into a portable WAR file

Use this command line in your "war\" directory:

```
          jar cfv temp.tar *
```

This packages up all of the files and directory structure into a portable WAR file. You should be able to migrate this file to other web services environments if necessary.

Make the WAR file deployable

You should now run the wsdeploy command in your "\war" folder:

```
          wsdeploy -o Qotd.war temp.war
```

This tool converts the portable WAR file into an implementation specific WAR file by generating all of the tie classes and a WSDL file from your Java interface class.

Deploy your web service

The final step is to copy your deployable WAR file, Qotd.war, to the "webapps" folder of your JWSDP Tomcat installation. This should automatically deploy the web service contained in the WAR file into the server. You will have to restart the server in order for these changes to take effect. You can inspect your service by going to "http://localhost:8080/manager/html". Your service should be listed here.

If you now go to "http://localhost:8080/Qotd/qotd", you should be able to inspect the services WSDL file. You should save the WSDL file so that you can use it to write your client software in the next tutorial.

Client

A web service is not much use without a client with which it can interact. The second part of this tutorial will show you how to construct a simple command-line client for the service that you have just written and deployed.

WSDL

Create a directory for you to work in. Call it "client", and make it in your tutorial directory.

Get the WSDL from your service running in your container, using the following URL:

```
http://localhost:8080/Qotd/qotd?wsdl
```

Copy this into your "client" directory.

Generate the glue components

First, create a file called config.xml, and add the following lines to it:

```
<configuration
xmlns="http://java.sun.com/xml/ns/jax-
rpc/ri/config">
     <wsdl location="Qotd.wsdl" packageName="qotd"
/>
</configuration>
```

This is the configuration file for the wscompile utility that will generate all of the glue code. The file tells wscompile which WSDL file to use, and what package name we would like the glue code placed in.

Next, we run the wscompile utility itself:

```
wscompile -gen:client -keep -d . config.xml
```

This gets wscompile to generate the client code (-gen:client), keep the generated source code so we can examine it later (-keep), work in the current directory (-d .), using the configuration described in config.xml.

This will create a "qotd" directory, which will contain a number of .class and .java files that allow us to talk to the web service.

Write the client

Create a new file called qotdClient.java in your "client" directory.

First, we must import the various RPC libraries and glue code that we have just generated:

```
// Standard libraries
import javax.xml.rpc.Service;
import javax.xml.rpc.Stub;

// Our glue code
import qotd.Qotd_Service_Impl;
import qotd.Qotd_PortType;
```

Next, we create our client class and give it two methods:

```
public class qotdClient {

     public static String getQuote() throws
Exception
     {

     }

     public static void main(String[] args)
     {

     }
}
```

Now, we'll write the getQuote function so that it calls a remote function via web services. Add the following to the getQuote function:

```
     Qotd_Service_Impl service = new
Qotd_Service_Impl();
     Qotd_PortType port = service.getQotdPort();
```

The Qotd_Impl class can be seen as the "class" of the web service, and QotdPortType can be seen as an "instance" of that class.
Now we call the remote "getQuote" method, as defined in our WSDL, and return it to our caller:

```
     String quote = port.getQuote();
     return (quote);
```

And that is all that our getQuote client method needs. Now, we'll write some code in our "main" method so that it calls our getQuote method:

```
          try{
               String quote = getQuote();
               System.out.println(quote);
          }
          catch (Exception e) {e.printStackTrace();}
```

That should now be all of the code that is required. Compile the code with:

```
     javac qotdClient.java
```

Run the Client

Try the client out with:

```
     java qotdClient
```

If the client seems to hang, it could be because the remote service is down (welcome to the world of web services! ;-)