SL02
Web Services Implementation on iSeries Lab

ITSO iSeries Technical Forum 2003

Presented by
Yessong Johng

International Technical Support Organization

Web Services Implementation on iSeries Lab

February 2003

Take Note! Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 57.

First Edition (February 2003)

This edition applies to WebSphere Application Server Version V5.0, Program Number 5733-WS5 for use with the OS/400 Version 5 Release 2.

This document created or updated on February 5, 2003.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Lab 1

Enabling Web Services on iSeries

In this lab you will be using both the IBM WebSphere Development Studio Client for iSeries, Version 4.0 (WDVc V5 is not available at the time of the production of this lab materials) and the IBM WebSphere Application Server for iSeries Version 5.0. The WebSphere Development Studio Client (WDSc) is the tool you will use to build and test the Web service. The WebSphere Application Server (WAS) is the server where you will deploy and test the Web service you will create.

The Web service you will be creating will provide inventory query into a DB2 UDB database on the iSeries to retrieve the quantity and availability of the product requested by an auto dealer. The Web service will either return a string stating the quantity that is available and the date of availability, or it will return a string stating “Item not found in inventory, Please try again.”

Resource Table

Use the following table as a resource reference throughout the lab.

Resource name	Value
Team user ID	WEBSVCxx
password	java1a
WAS instance	IWEBxxSVC
User library	WEBSVCxx
User directory	<iSeries root>/WEBSVCxx
iSeries host name	
iSeries IP address	

Team number: Throughout these lab exercises, replace *xx* with the team number that has been assigned to you.

Objectives

This lab shows you how to create, deploy, and test a Web service. To accomplish this objective, this lab teaches you:

- How to create a Java program in the WDSc client that provides the Web service
- How to use the Web service wizard in WDSc to create the contents of the Web service
- How to deploy the Web service application to WAS
- How to test deployment of the Web service

Time required

The time required to efficiently complete this lab project is about 2 and a quarter hours.

Task summary

This lab consists of the following four sub labs:

1. Create a Java program that provides the Web service
2. Use the Web Service wizard in WDSc to create the following files:
   a. WSDL files
   b. DDS.xml file
   c. SOAP client proxy
   d. SOAP requester test program
3. Deploy the Web Service application to WAS
   a. Export the Web Project to an EAR file
   b. Deploy the EAR file as an Enterprise Application
   c. Start the Web service
4. Test deployment of the Web service

Task 1: General setup

This task is not directly related with the main objective of this lab, Creating your first Web service application, but required for the way the whole lab is set up.

Host table entry

__ 1. Open the file, **C:\WINNT\system32\drivers\etc\hosts**, with your NotePad or EditPad Lite.

__ 2. Add the entry with the IP address and the host name of your iSeries machine as illustrated in Example 1-1.

Example 1-1 Host table entry

```
# Copyright (c) 1993-1999 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
```

```
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

127.0.0.1       localhost
10.10.10.21     AS06
```

Copying InventoryAvailabityService Java file into student's directory

One of the tasks of the lab is to import the Java file and modify it. Again, a Java file is already created and provided for you. You just need to copy it from the setup folder. To create a work folder and populate it:

__ 1. Create a **tmp** directory under C:\WDSC, like **C:\WDSC\tmp**.

__ 2. Map a network drive on your workstation to the `root` share on iSeries. Use your team's user ID (WEBSVCxx).

> **Note:** In order to use a different user ID, click **different user name** link in the Map Network Drive window.

__ 3. Copy the following two files From your iSeries setup folder, **SL02V5Files** to **C:\WDSC\tmp**:

- InventoryAvailabilityService.java
- CODE.LOG (This file could be already there. If that's the case, just copy the Java file.)

Task 2: Create Java program that provides the Web service

> **Note:** This is the 'read' only task. You will make some of the changes on the Java file, the highlighted portion as in Figure 1-1, at later part of the lab.

The first step involves creating a Java program that contains the application logic that the Web service will provide. We have already created the Java program, so you will not need to do that in this step. The code is shown in the two following figures Figure 1-1 and Figure 1-2.

```
package com.vck.rst;


import java.util.*;
import javax.sql.*;
import java.sql.*;

public class InventoryAvailabilityService {

public String getInventoryAvailability(String partNo){

      Connection connection   = null;
      String system = "AS06";
      String userName = "yessong";
      String password = "ong7yes";
      String collectionName = "IWEBSVCLAB";

      ResultSet resultSet =null;
      String quantityAvailable = null;
      String byDate = null;
      String availability = null;
        try {

        //


           // This will register the driver with DriverManager.


         Class.forName("com.ibm.as400.access.AS400JDBCDriver");

            // Get a connection to the database.
            System.err.println("\nabout to get connection ");
            connection = DriverManager.getConnection ("jdbc:as400://"
                + system+"/"+collectionName,userName,password);

      System.err.println("\nconnection obtained "+connection);

          // Prepare a statement for inserting rows. Since we
            // execute this multiple times, it is best to use a
            // PreparedStatement and parameter markers.
            PreparedStatement invAvlpStmt = connection.prepareStatement ("SELECT
INVQNTY,INVBKDTE FROM INVAVLPF WHERE INVPRTNO=? ");

              //set the prepared statement for part No
              invAvlpStmt.setString(1,partNo);

            //execute the query string
             resultSet = invAvlpStmt.executeQuery();
```

Figure 1-1 InventoryAvailabilityService.java (part 1 of 2)

The code in Figure 1-1 creates the InventoryAvailabilityService class. The code in this figure connects to the iSeries using the native JDBC driver. Once connected, a prepared statement is created and executed to select the data from the DB2 UDB database, INVAVLPF.

```
            //this will also advance
            if(resultSet!=null && resultSet.next() ){

            quantityAvailable = resultSet.getString(1);
            byDate = resultSet.getString(2);

            availability = "This item has available quantity "+new
String(quantityAvailable).trim()+" by Date "+byDate;

            }
            else {

            availability = "Item not found in inventory, Please try again";
            }

         if(connection!=null){
         connection.close();
         }


        }//end of try

        catch (Exception e) {

            System.out.println ("Exception in getInventoryAvailability, ERROR: " +
e.getMessage());
         e.printStackTrace();
        }
        finally {

            // Clean up.
            try {
                if (connection != null)
                    connection.close ();
            }
            catch (SQLException e) {
                // Ignore.
            }
        }//end finally

   return availability;

     }//end of method



}//end of class
```

Figure 1-2 InventoryAvailabilityService.java (part 2 of 2)

The result set from the query is fetched and the available quantity for the part number selected in the SQL statement is returned to the calling application. If the part number cannot be found, the message “Item not found in inventory, Please try again.” is sent back to the invoking application. If any, errors are monitored for and the last step is to close the connection.

This Java program will be the basis for our Web service. You will see in the resulting tasks how to take this Java program and convert it into a Web service and how to deploy it as a Web service.

Lab 2

Create the Web Service

Now that we have a Java program that will be the basis of our Web Service, we are ready to actually create the Web Service. We first create a Web Service within a Web Project on WDSc. This section will walk you through all of the steps necessary to do this. We will be using the IBM WebSphere Development Studio Client for iSeries (WDSc) for this task. This section is broken into sub sections.

Task 1: Creating a Web Project

First of all, we need to create a Web Project on WDSc.

__ 1. To begin, you will need to open the WDSc client. To do this select **Start -> Programs -> IBM WebSphere Development Studio Client for iSeries -> IBM WebSphere Studio Site Developer Advanced**.

Important: This step may take a while to complete. This is normal. This is a very robust development environment and can take a little while to load.

__ 2. We are now ready to create a project that will be the package to contain our Web service. To do this select **File -> New -> Project** as seen in Figure 2-1.

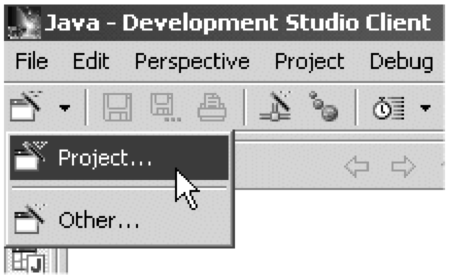


Figure 2-1 Create new project in WDSc.

__ 3. The New Project wizard is presented. Select **Web** in the left pane.

__ 4. Select **Web Project** in the right pane. See Figure 2-2 for details.

__ 5. Press **Next** to continue.

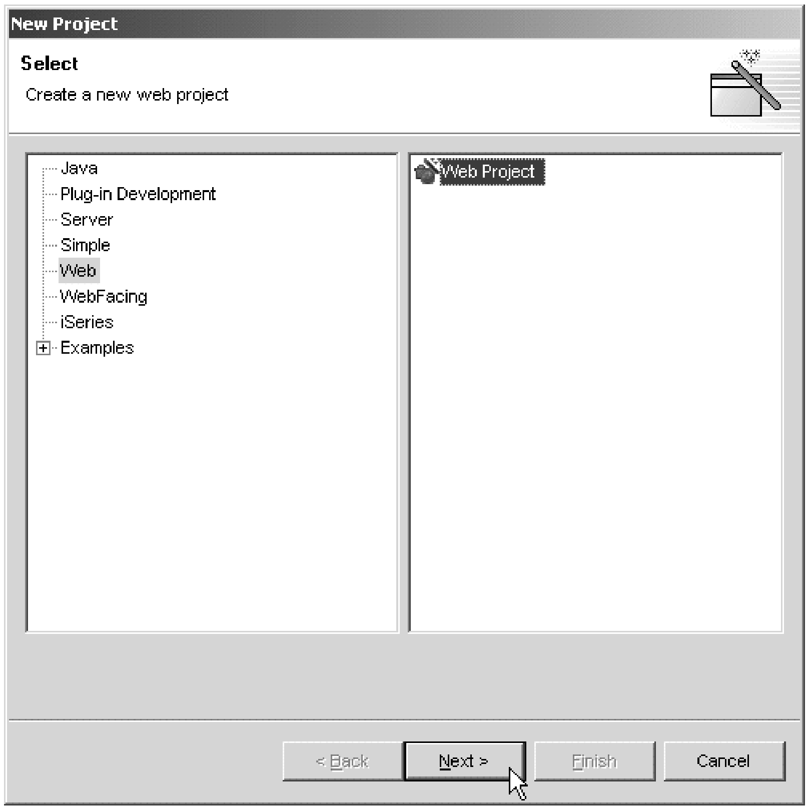


Figure 2-2 Select New Project type "Web Project" in WDSc.

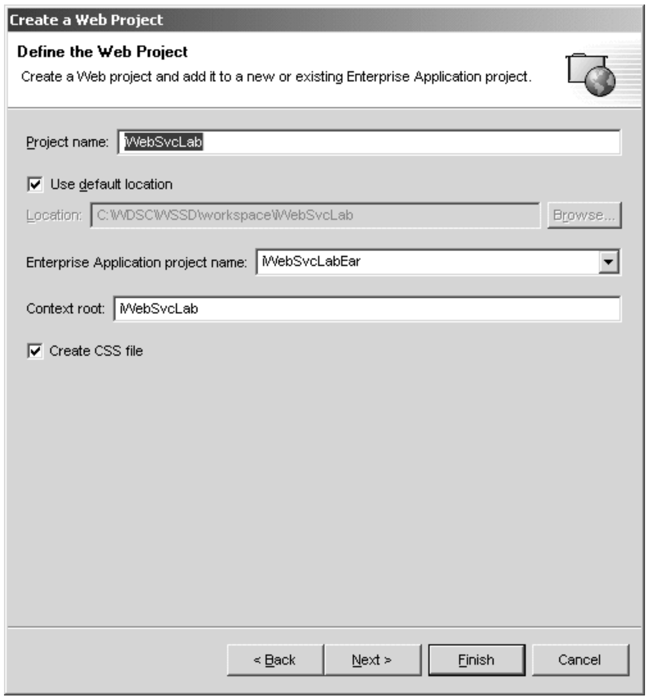
__ 6. The next screen prompts you for details about the project you are creating. For Project Name enter **WebSvcLab**.

__ 7. For Enterprise Application project name, replace the text of **DefaultEAR** with the text **WebSvcLabEar**.

__ 8. Check the box next to **Create CSS file**. This will create a cascading style sheet for the project.

__ 9. Refer to Figure 2-3 on page 9 to ensure all parameters are filled in correctly. When satisfied, press **Next** to continue.

Note: Do not press Finish in step 10. If you do you will need to delete your project and start over.


Create a Web Project

Define the Web Project

Create a Web project and add it to a new or existing Enterprise Application project.

Project name: WebSvcLab

☑ Use default location

Location: C:\WDSC\WSSD\workspace\WebSvcLab

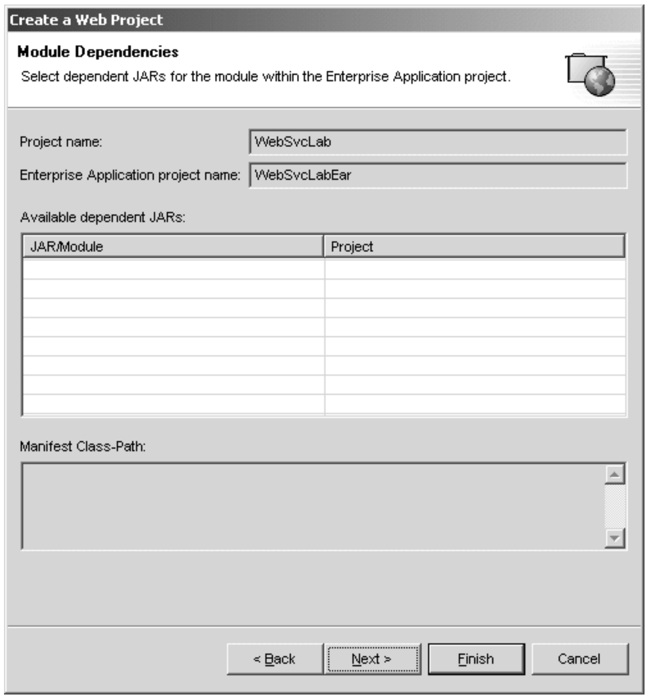
Enterprise Application project name: WebSvcLabEar

Context root: WebSvcLab

☑ Create CSS file

Figure 2-3 Specify project name, enterprise application project name, and context root.

__ 10. In the Module Dependencies selection screen as shown in Figure 2-4, make sure the Project name and Enterprise Application project names are correct as shown in. Press **Next** to continue.


Create a Web Project

Module Dependencies

Select dependent JARs for the module within the Enterprise Application project.

Project name: WebSvcLab

Enterprise Application project name: WebSvcLabEar

Available dependent JARs:

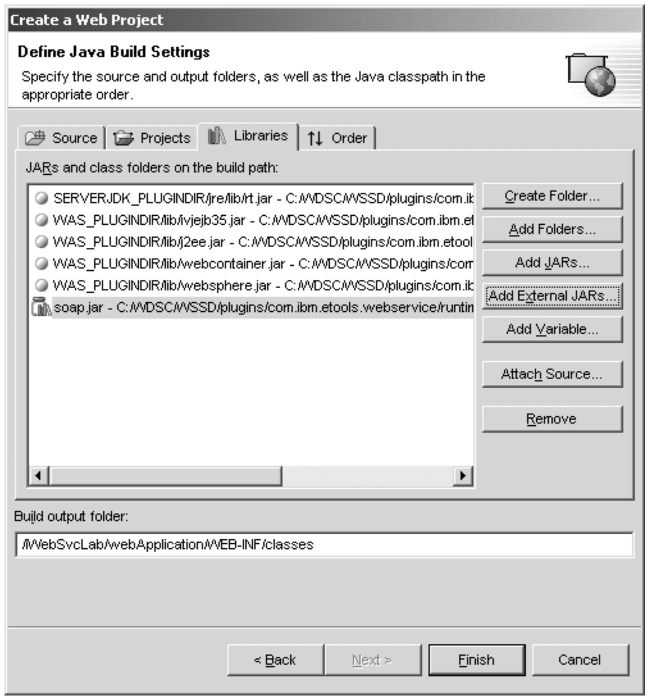
JAR/Module	Project

Manifest Class-Path:

Figure 2-4 Project module dependencies wizard.

__ 11. We are now ready to add the dependent jar files to the classpath of the Web project. To do this, click on the **Libraries** tab on the Define Java Build Settings screen.

__ 12. Next click the **Add External Jars...** button. See Figure 2-5.


Create a Web Project

Define Java Build Settings

Specify the source and output folders, as well as the Java classpath in the appropriate order.

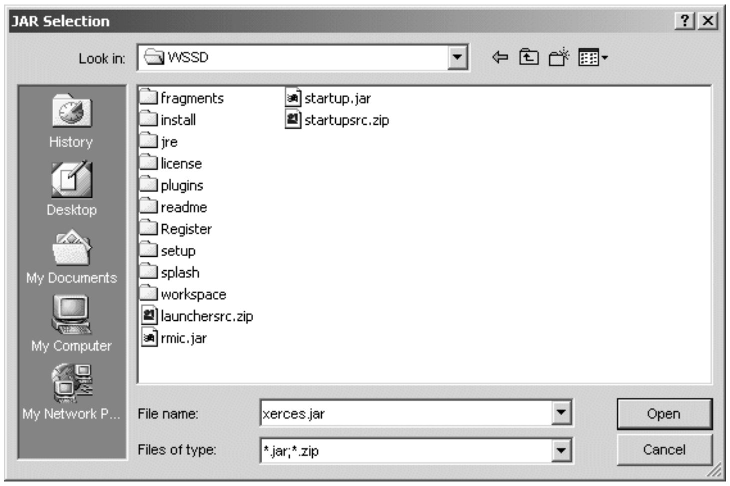
JARs and class folders on the build path:

SERVERJDK_PLUGINDIR/jre/lib/rt.jar - C:/WDSC/WSSD/plugins/com.ib
WAS_PLUGINDIR/lib/ivjejb35.jar - C:/WDSC/WSSD/plugins/com.ibm.et
WAS_PLUGINDIR/lib/j2ee.jar - C:/WDSC/WSSD/plugins/com.ibm.etool
WAS_PLUGINDIR/lib/webcontainer.jar - C:/WDSC/WSSD/plugins/com
WAS_PLUGINDIR/lib/websphere.jar - C:/WDSC/WSSD/plugins/com.ib
soap.jar - C:/WDSC/WSSD/plugins/com.ibm.etools.webservice/runtin

Build output folder: /WebSvcLab/webApplication/WEB-INF/classes

Figure 2-5 Adding external jar files to Web project classpath in WDSc.

__ 13. In the Jar Selection window, you will need to navigate the directory structure to the following directory: **C:\WDSC\WSSD\Plugins\org.apache.xerces\**. To do this navigation, double click the **plugins** directory, as seen in Figure 2-6.


JAR Selection

Look in: WSSD

fragments
install
jre
license
plugins
readme
Register
setup
splash
workspace
launchersrc.zip
rmic.jar
startup.jar
startupsrc.zip

File name: xerces.jar

Files of type: *.jar;*.zip

Figure 2-6 Jar selection window, double click plugins

__ 14. Now scroll way over to the right and double click the directory **org.apache.xerces**.

__ 15. Click on the **xerces.jar** file and click **Open**.

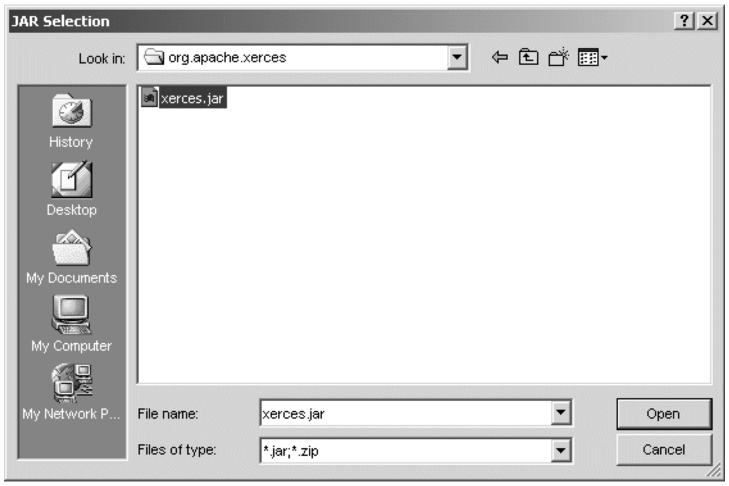


Figure 2-7 Select xerces.jar file to be added to the Web project classpath

__ 16. You will now see the xerces.jar file added to library folder of the Define Java Build Settings wizard.

__ 17. We have three more jar files to include, the next jar file is the soap.jar file. Press the **Add External JARs...** button again.

__ 18. Navigate to the directory that contains the soap.jar file, **C:\WDSC\WSSD\plugins\com.ibm.etools.websphere.runtime\lib.**

__ 19. Click on the **soap.jar** file and click **Open.**

__ 20. Press the **Add External JARs...** button again.

__ 21. Repeat this step to copy **xml4j.jar** file from the same directory.

__ 22. We are now ready to select the final jar file to added to the project classpath. Press the **Add External JARs...** button again.

__ 23. Navigate to the directory **C:\WDSC\WSSD\plugins\com.ibm.etools.iseries.toolbox\runtime.**

__ 24. Click on the **jt400.jar** file and click **Open.**

__ 25. Now that all of the appropriate jar files have been imported, you are ready to create the project. To do this, press the **Finish** button.

__ 26. You will see a message at the bottom of the Define Web Project wizard screen saying "Creating Java Project...". When the project has been created, you will be brought back to the main perspective in the WDSc client.

Task 2: Creating Java class file in the Web Project

Next task is to create InventoryAvailabilityService class file in the Web Project we have just created in the previous step. Java file is already provided for you. We will import this provided Java file into our Web Project.

__ 1. We will need to open the Web perspective in the WDSc client. Select **Perspective -> Open -> Web** as shown in Figure 2-8.

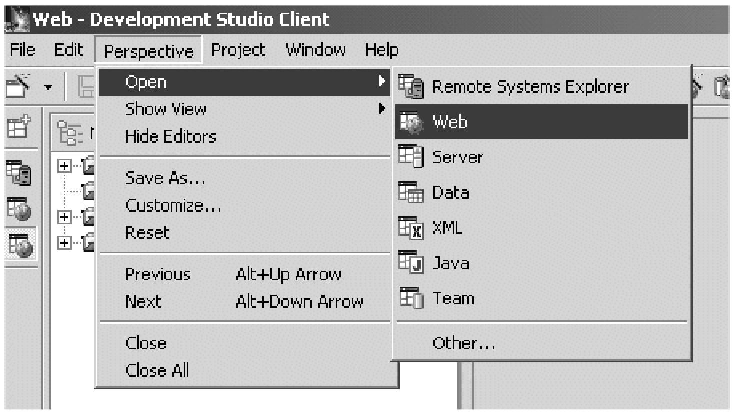


Figure 2-8 Opening the Web perspective

__ 2. Expand the "+" sign next to both the **WebSvcLab** and the **WebSvcLabEar** project. Expand the directory structures for both projects as shown in Figure 2-9 on page 12.

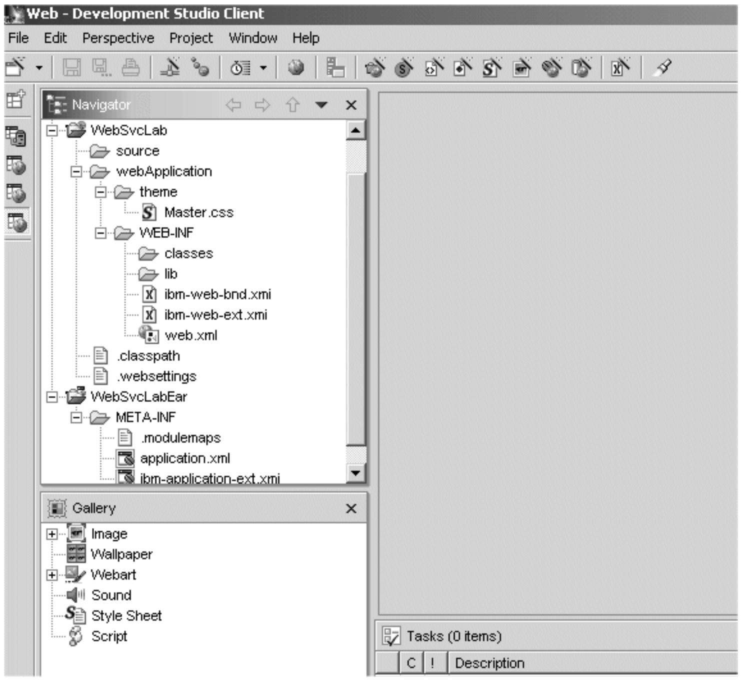


Figure 2-9 Expand the WebSvcLab and WebSvcLabEar projects

__ 3. We can now import the InventoryAvailableService class file into the project. Highlight **source** under the **WebSvrLab** project.

__ 4. Select **File -> Import**.

__ 5. On the Import Select wizard, select **File system** as shown in Figure 2-10.

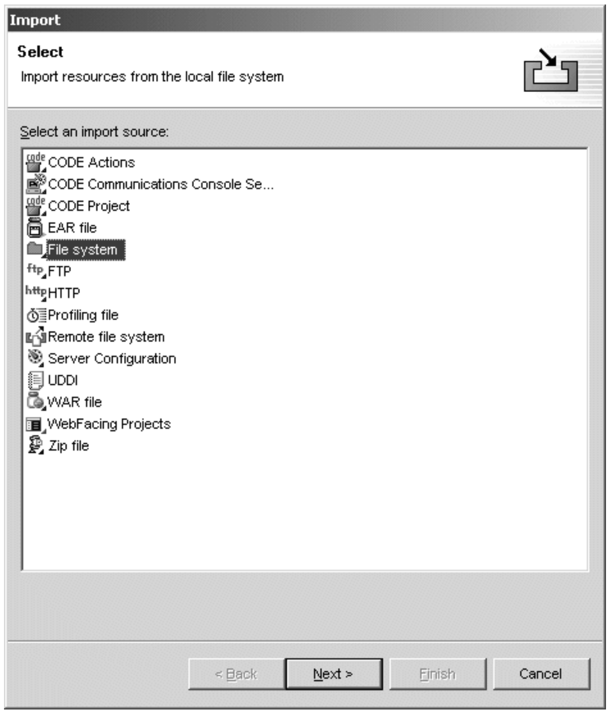


Figure 2-10 Select File System for import option

__ 6. Click the **Next** button to continue.

__ 7. On the File system selection screen, click the **Browse** button.

__ 8. Navigate through the directory structure selecting **C:\WDSC\tmp** and select **tmp**.

__ 9. Press **OK**.

__ 10. Back at the Import File System screen, check the box next to tmp.

__ 11. Click on **tmp**.

__ 12. Deselect all of the check boxes for the files that you now see in the right pane. Only file **InventoryAvailabilityService.java** file should have a check box next to it. See Figure 2-11.

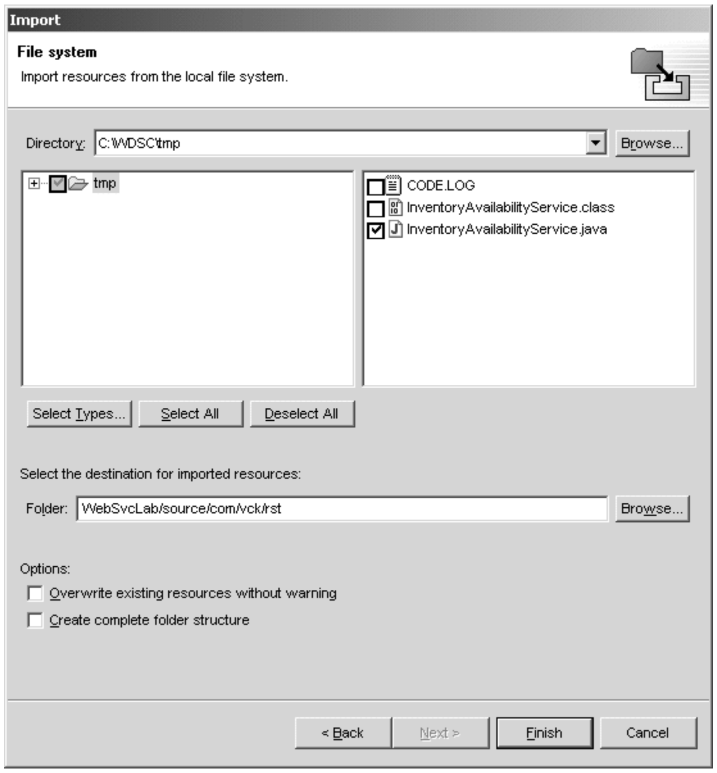

Import
File system
Import resources from the local file system.
Directory: C:\WDSC\tmp
tmp
CODE.LOG
InventoryAvailabilityService.class
✓ InventoryAvailabilityService.java
Select the destination for imported resources:
Folder: WebSvcLab/source/com/vck/rst
Options:
Overwrite existing resources without warning
Create complete folder structure

Figure 2-11 Select only file InventoryAvailabilityService.java for Import File System selection

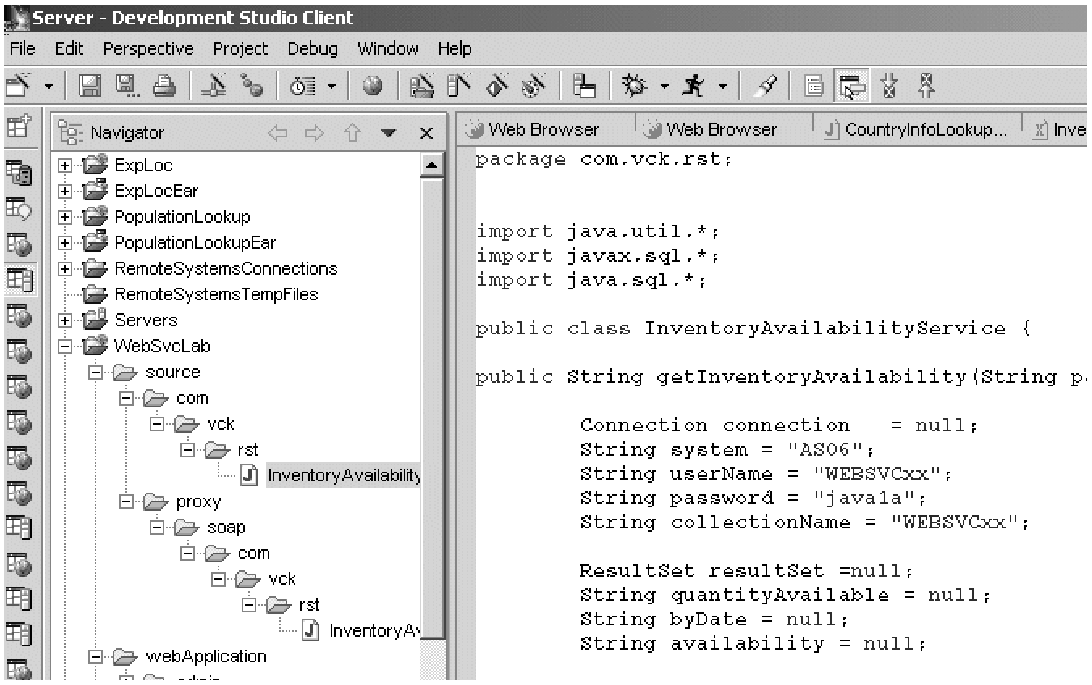
__ 13. The destination folder for the imported resource needs to be changed to match the directory structure that the Java class file package was created with. You will need to append **/com/vck/rst** to the \WebSvrLab\source directory selection. The final directory structure selection for the destination folder should be **WebSvrLab/source/com/vck/rst**.

__ 14. Click the **Finish** button to continue. This will import the java source file into your Web service project.

Modifying InventoryAvailabilityService.java

We will modify the Java file for the consumption of your own team.

__ 1. On WDSc, open InventoryAvailabilityService.java file you just imported from the previous task as shown on Figure 2-12.



```
package com.vck.rst;


import java.util.*;
import javax.sql.*;
import java.sql.*;

public class InventoryAvailabilityService {

public String getInventoryAvailability(String p

        Connection connection   = null;
        String system = "AS06";
        String userName = "WEBSVCxx";
        String password = "java1a";
        String collectionName = "WEBSVCxx";

        ResultSet resultSet =null;
        String quantityAvailable = null;
        String byDate = null;
        String availability = null;
```

Figure 2-12 Modifying InventoryAvailabilityService.java file

__ 2. Modify the portion of Connection values:

- String system = “<iSeries host name>”, for example “AS06”
- String user name = “WEBSVCxx”
- String password = “java1a”
- String collectionName = “WEBSVCxx”

The portion of the source code is highlighted in Figure 1-1 on page 4.

__ 3. Save the changes by File -> Save as shown in Figure 2-13.

Note: WDSc creates a class file automatically when you save the changes made in java file. It is, therefore, ***totally important you ‘save’ the file***!

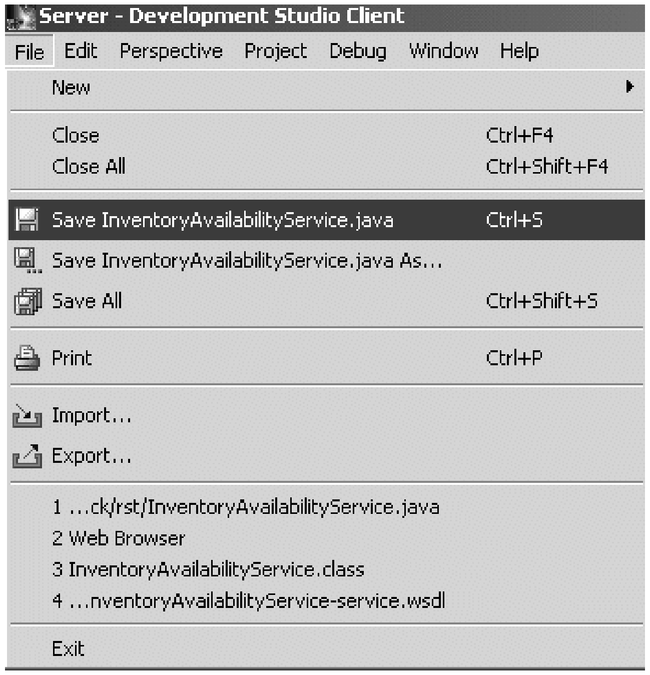

Server - Development Studio Client
File Edit Perspective Project Debug Window Help
New
Close Ctrl+F4
Close All Ctrl+Shift+F4
Save InventoryAvailabilityService.java Ctrl+S
Save InventoryAvailabilityService.java As...
Save All Ctrl+Shift+S
Print Ctrl+P
Import...
Export...
1 ...ck/rst/InventoryAvailabilityService.java
2 Web Browser
3 InventoryAvailabilityService.class
4 ...nventoryAvailabilityService-service.wsdl
Exit

Figure 2-13 Saving the changes in InventoryAvailabilityService.java file

> **Note:** We are hard coding the system name and other values here for the purpose of the lab but in real world, you might write an application with better flexibility using properties file or Java properties.

Task 3: Creating a Web Service in WDSc Web Project

Now we will create a web Service in the Web Project on WDSc.

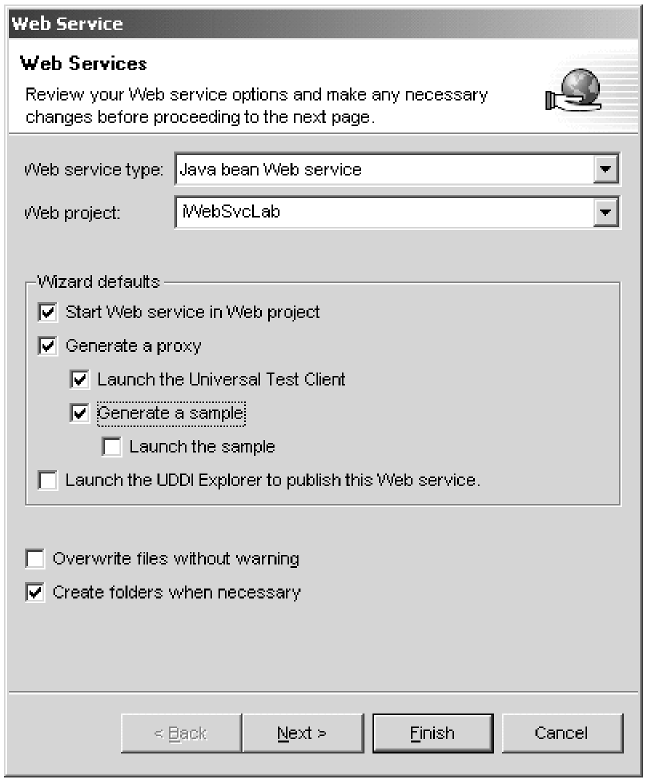
__ 1. On WDSc, select **File -> New -> Other**.

__ 2. Select **Web Services** in the left pane.

__ 3. Select **Web Service** in the right pane.

__ 4. Press **Next** to continue. This step may take a few seconds to complete.

__ 5. The Web Service wizard now appears. As shown in Figure 2-14, make sure the Web service type is **Java bean Web service.**


Web Service

Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Web service type: Java bean Web service

Web project: WebSvcLab

Wizard defaults

- [✓] Start Web service in Web project
- [✓] Generate a proxy
  - [✓] Launch the Universal Test Client
  - [✓] Generate a sample
    - [ ] Launch the sample
- [ ] Launch the UDDI Explorer to publish this Web service.

- [ ] Overwrite files without warning
- [✓] Create folders when necessary

Figure 2-14 Web Services create options wizard in WDSc

__ 6. Validate that the Web project name is **WebSvcLab**.

__ 7. Ensure there is a check box next to each of the following:

```
Start Web service in Web project
Generate a proxy
Launch the Universal Test Client
Generate a sample
Create folders when necessary
```

__ 8. Click **Next** to continue.

__ 9. Every Web service must have a service component which provides core functionality of the Web service. In our InventoryAvailabilityService example, the java bean InventoryAvailabilityBean will encapsulate the core functionality of the Web service. In the Web Service Java Bean selection screen, press the **Browse classes...** button, see figure Figure 2-15 on page 18.

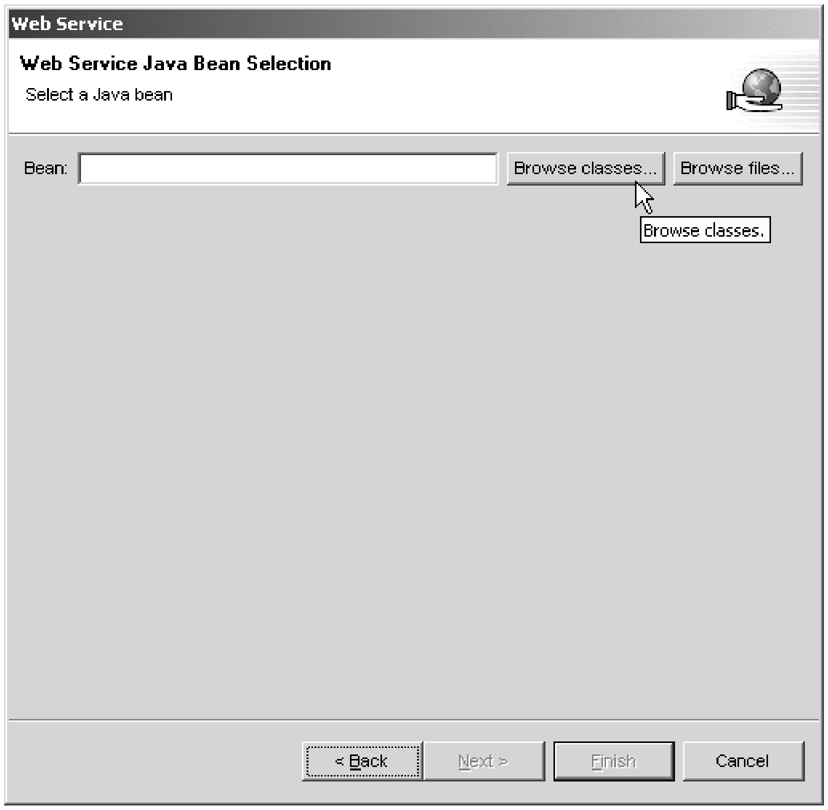


Figure 2-15 Select Java bean on which Web service will be built in WDSc

__ 10. Begin typing **InventoryAvailabilityService**. As you type, the InventoryAvailabilityService bean will appear. See Figure 2-16 on page 19.

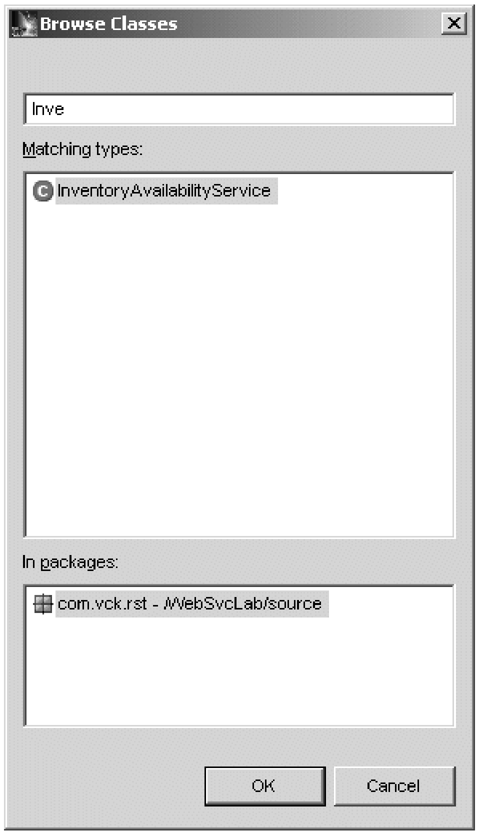


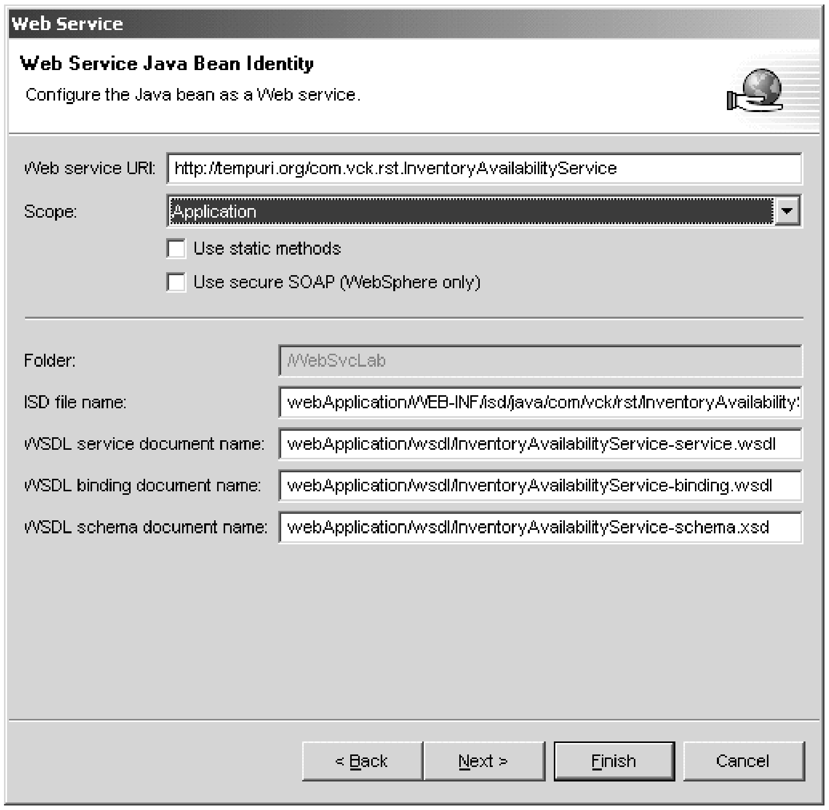
Figure 2-16 Select InventoryAvailabilityService Java bean in the browse classes dialog box

__ 11. Now that the InventoryServiceAvailability bean is shown, **highlight** it.

__ 12. Press **OK** to continue.

__ 13. You are now brought back to the Web Service Java Bean Selection wizard. Press **Next.**

__ 14. The Web Service Java Bean Identity screen contains information about the Web service URI, scope, ISD file name, and WSDL documents. See Figure 2-17 on page 20 for details. All of the parameters are filled in for you. The URI (Universal Resource Indicator) can be in the form of a URL such as **http://tempuri.org/com.vck.rst.InventoryAvailabilityService** or a URN (Universal Resource NameSpace) such as **urn:InventoryAvailabilityService**. There are pros and cons to both, so a decision must be made. By using the http:// format, you can register the domain and guarantee uniqueness of the name of your Web service, by using a urn, it offers you more flexibility in the future for making changes to the Web service name.


Web Service

Web Service Java Bean Identity

Configure the Java bean as a Web service.

Web service URI: http://tempuri.org/com.vck.rst.InventoryAvailabilityService

Scope: Application

[ ] Use static methods

[ ] Use secure SOAP (WebSphere only)

Folder: /WebSvcLab

ISD file name: webApplication/WEB-INF/isd/java/com/vck/rst/InventoryAvailability

WSDL service document name: webApplication/wsdl/InventoryAvailabilityService-service.wsdl

WSDL binding document name: webApplication/wsdl/InventoryAvailabilityService-binding.wsdl

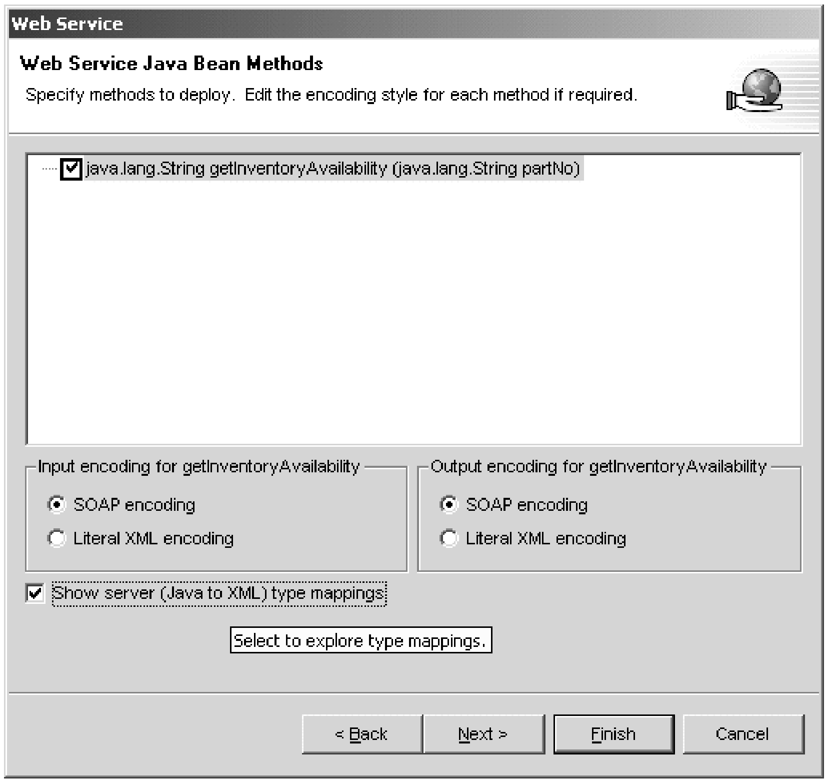
WSDL schema document name: webApplication/wsdl/InventoryAvailabilityService-schema.xsd

Figure 2-17 Specify Web service URI for the Web service Java bean

__ 15. Change the Web service URI to **http://tempuri.org/com.vck.rst.InventoryAvailabilityService** if it is not already selected.

__ 16. Click **Next** to continue.

__ 17. On the Web Service Java Bean Methods wizard, ensure **SOAP encoding** is selected for both the Input encoding for getInventoryAvailability and Output encoding for getInventoryAvailability. This is required because we are building our Web service on the SOAP encoding envelope.


Web Service

Web Service Java Bean Methods

Specify methods to deploy. Edit the encoding style for each method if required.

[✓] java.lang.String getInventoryAvailability (java.lang.String partNo)

Input encoding for getInventoryAvailability: [● SOAP encoding] ○ Literal XML encoding

Output encoding for getInventoryAvailability: [● SOAP encoding] ○ Literal XML encoding

[✓] Show server (Java to XML) type mappings

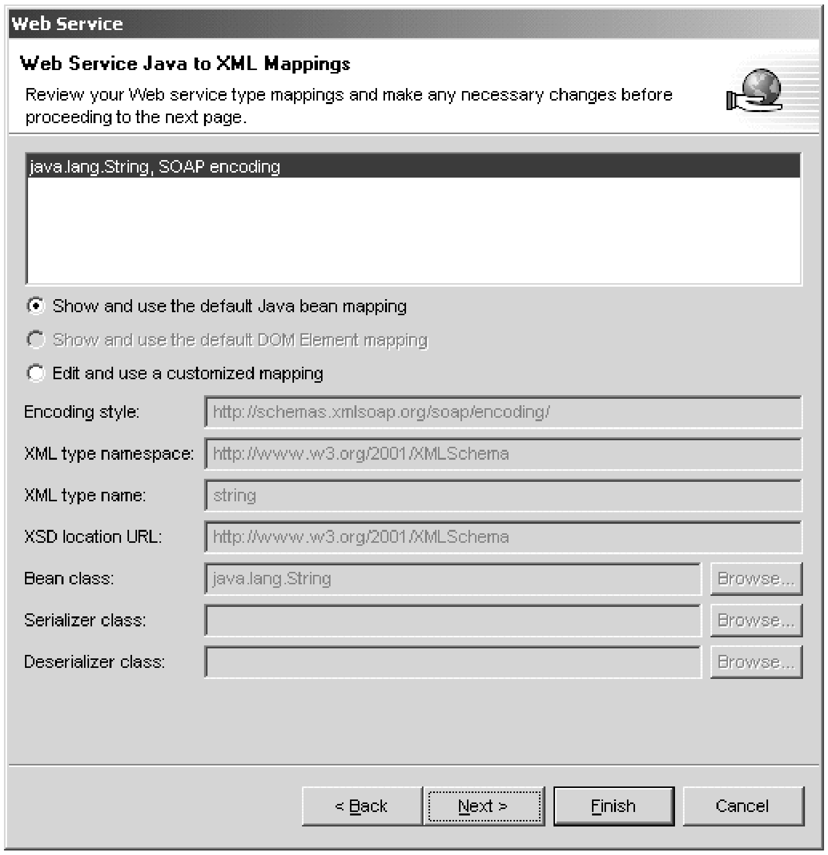
Select to explore type mappings.

Figure 2-18 Specify encodings for the Web service Java bean methods in WDSc

__ 18. Check the box next to **Show server (Java to XML) type mappings**.

__ 19. Click **Next** to continue.

__ 20. Figure 2-19 on page 22 shows the Web Service Java to XML Mappings screen. The Java bean method argument and return type associated with this Web service are shown. In our example, the method argument and return type are the same, they are both strings. This is depicted with **java.lang.String,SOAP encoding**.



Web Service

Web Service Java to XML Mappings

Review your Web service type mappings and make any necessary changes before proceeding to the next page.

java.lang.String, SOAP encoding

(●) Show and use the default Java bean mapping

( ) Show and use the default DOM Element mapping

( ) Edit and use a customized mapping

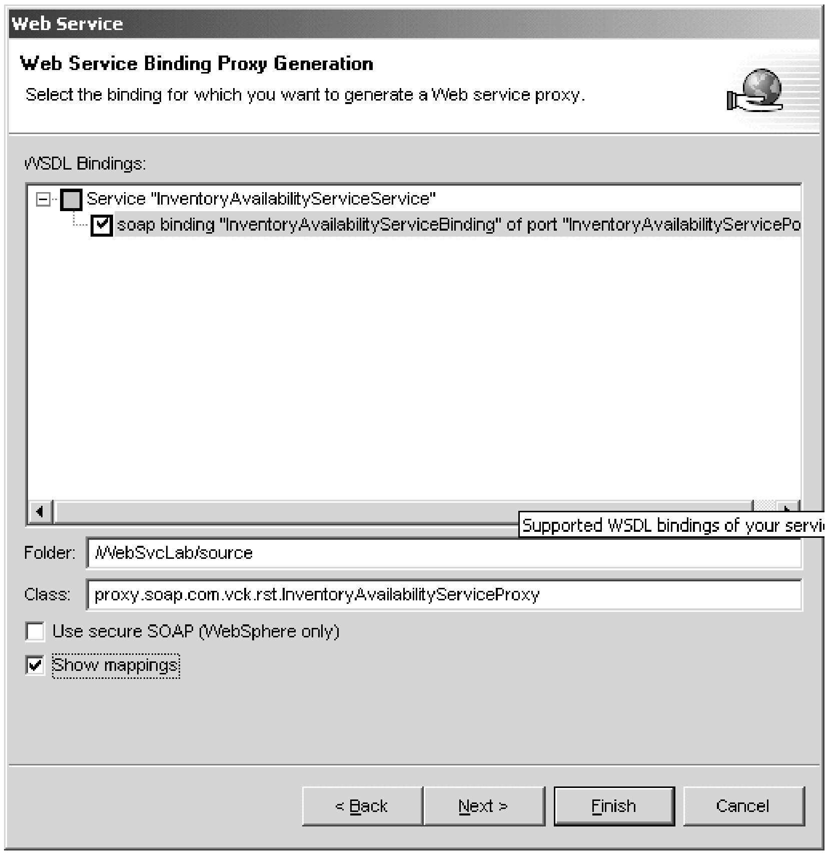
Encoding style:	http://schemas.xmlsoap.org/soap/encoding/
XML type namespace:	http://www.w3.org/2001/XMLSchema
XML type name:	string
XSD location URL:	http://www.w3.org/2001/XMLSchema
Bean class:	java.lang.String
Serializer class:	
Deserializer class:	

Figure 2-19 Verify Java to XML mappings for the Web service in WDSc

__ 21. The default for Java to XML mappings is **Show and use the default Java bean mapping.** If you want to customize these settings, you would select the radio button next to **Edit and use a customized mapping.**

__ 22. Leave all of the defaults on this screen and press **Next** to continue.

__ 23. The next screen is Web Service Binding Proxy Generation, see Figure 2-20 on page 23. Here is where we specify the binding type that will be generated for our InventoryAvailabilityService Web service. The bindings will be stored in the WSDL files and make the framework of the Web service.



Web Service Binding Proxy Generation

Select the binding for which you want to generate a Web service proxy.

WSDL Bindings:

- [ ] Service "InventoryAvailabilityServiceService"
  - [✓] soap binding "InventoryAvailabilityServiceBinding" of port "InventoryAvailabilityServicePo

Supported WSDL bindings of your servi

Folder: /WebSvcLab/source

Class: proxy.soap.com.vck.rst.InventoryAvailabilityServiceProxy

- [ ] Use secure SOAP (WebSphere only)
- [✓] Show mappings

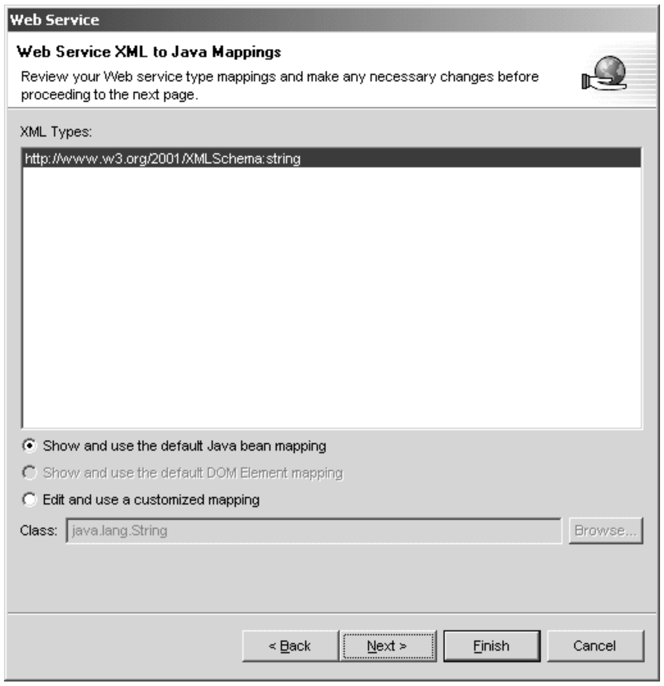
Figure 2-20 Verify Web service binding proxy generation options in WDSc

__ 24. To consume the Web service, a proxy client is also generated. The proxy client class is **proxy.soap.com.vck.rst.InventoryAvailabilityServiceProxy** and will be created in the folder named **/WebSvcLab/source.**

__ 25. Click the **check box** next to Show mappings.

__ 26. Click **Next** to continue.

__ 27. The next screen, Web Service XML to Java Mappings in Figure 2-21 on page 24, lets you verify whether the XML Schema mapping and Java class mapping for a particular input or output parameter is appropriate.



Web Service

Web Service XML to Java Mappings

Review your Web service type mappings and make any necessary changes before proceeding to the next page.

XML Types:

http://www.w3.org/2001/XMLSchema:string

(●) Show and use the default Java bean mapping

( ) Show and use the default DOM Element mapping

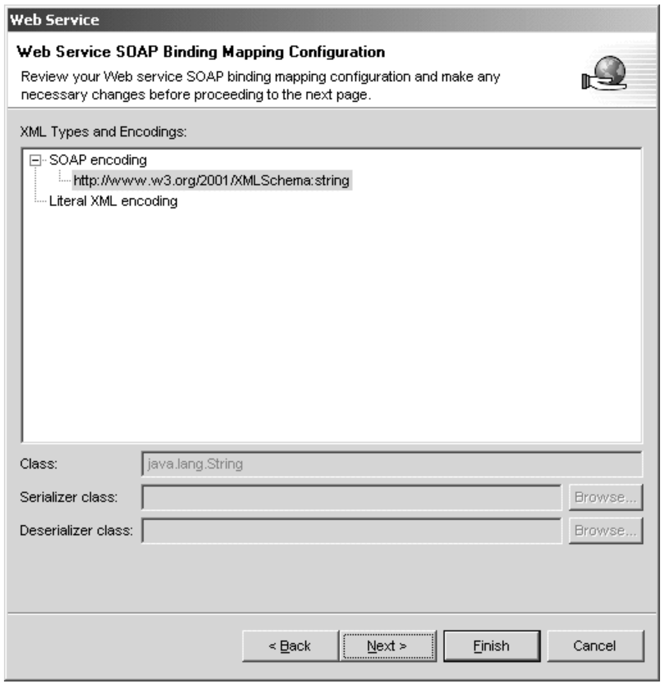
( ) Edit and use a customized mapping

Class: java.lang.String

Figure 2-21 Verify the input and output parameter Schema mappings for the Web service in WDSc

__ 28. Leave all of the defaults and click **Next** to continue.

__ 29. We now see the Web Service SOAP Binding Mapping Configuration screen, Figure 2-22. This is another verification screen. Here we can cross verify the configuration with SOAP encoding types for the parameters and return types of the Web service methods.



Web Service

Web Service SOAP Binding Mapping Configuration

Review your Web service SOAP binding mapping configuration and make any necessary changes before proceeding to the next page.

XML Types and Encodings:

- SOAP encoding
  - http://www.w3.org/2001/XMLSchema:string
- Literal XML encoding

Class: java.lang.String

Serializer class:

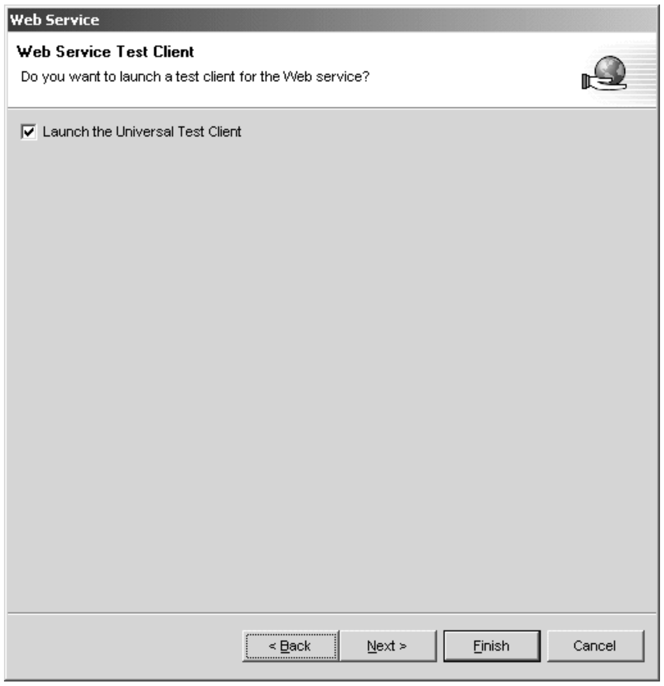
Deserializer class:

Figure 2-22 Verify Web service SOAP binding configuration in WDSc

__ 30. Leave everything at their default settings and press **Next** to continue.

__ 31. The next option we have is to create a Web service test client. This test client is created as **TestClient.jsp** in WDSc and packaged under the **sample** folder.

__ 32. **Click the checkbox** next to **Launch the Universal Test Client** as seen in Figure 2-23.

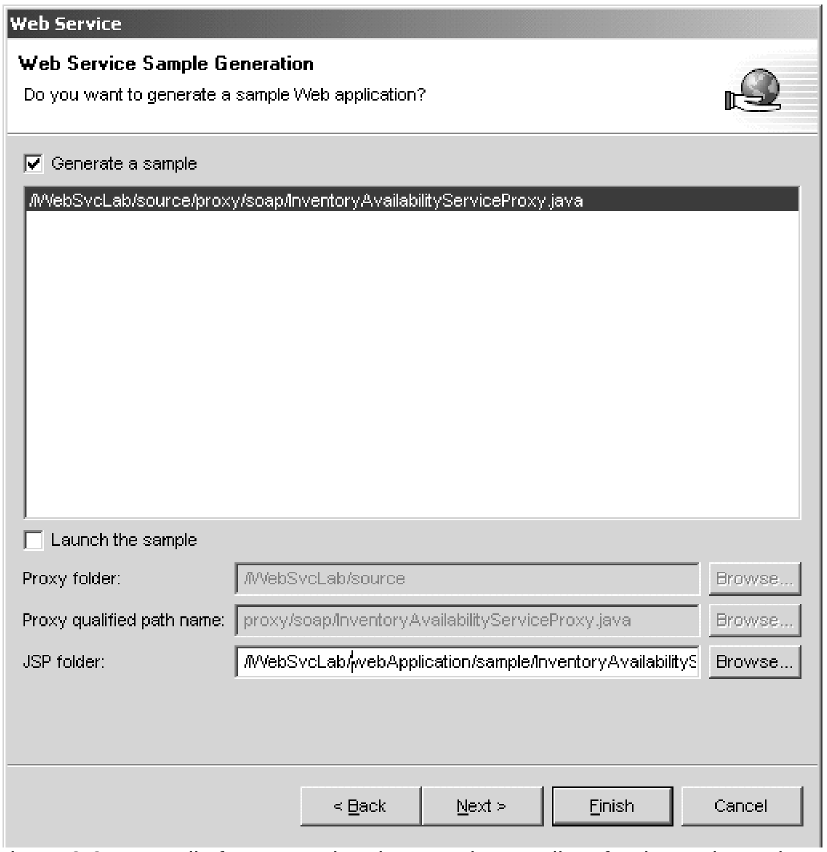


☑ Launch the Universal Test Client

Figure 2-23 Screen to select to create a test client for the Web service in WDSc

__ 33. Click **Next** to continue.

__ 34. Because we have selected to generate a test client, we are now prompted with options for generating the sample that will be used to test our Web service. These options are seen in Figure 2-24 on page 26.


Web Service

Web Service Sample Generation

Do you want to generate a sample Web application?

[✓] Generate a sample

/WebSvcLab/source/proxy/soap/InventoryAvailabilityServiceProxy.java

[ ] Launch the sample

Proxy folder: /WebSvcLab/source

Proxy qualified path name: proxy/soap/InventoryAvailabilityServiceProxy.java

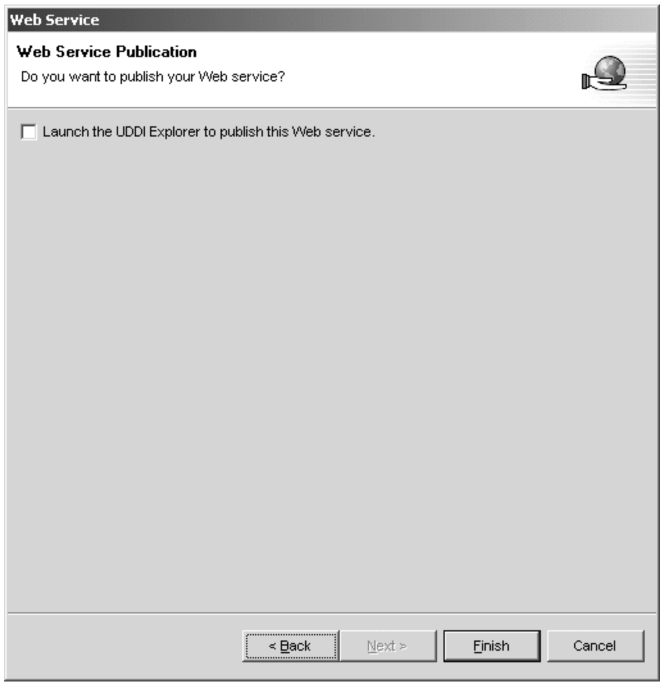
JSP folder: /WebSvcLab/webApplication/sample/InventoryAvailabilityS

Figure 2-24 Details for generating the sample test client for the Web service

__ 35. Click the check box next to **Generate a sample**. This will cause the TestClient.jsp to be generated in the JSP folder of **/WebSvcLab/webApplication/sample/InventoryAvailabilityService.**

__ 36. Press **Next** to continue.

__ 37. In the next screen, Figure 2-25, we have the option to launch the UDDI Explorer to publish the Web service.


Web Service

Web Service Publication

Do you want to publish your Web service?

☐ Launch the UDDI Explorer to publish this Web service.

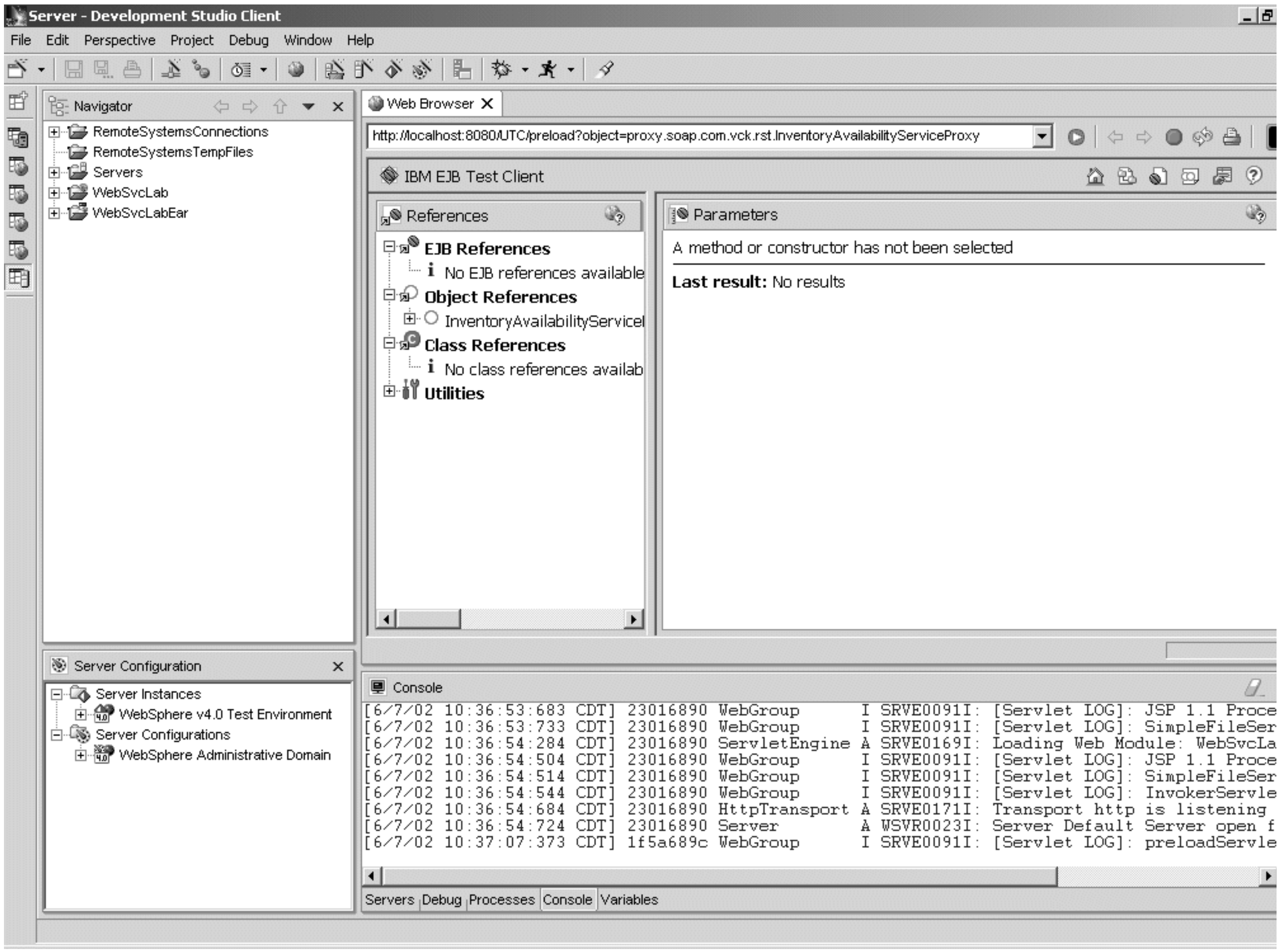
Figure 2-25 Web service publication option in WDSc

__ 38. Leave the Launch the UDDI Explorer to publish this Web service ***unchecked***.

> **Note:** UDDI is for publishing the Web Service. We are not using UDDI in this lab so be sure to leave this box unchecked.

__ 39. Click **Finish** to generate the Web service. You will see a number of status messages with progress bar indicators flash across the bottom of the window. This step may take a few minutes, be patient and let it complete normally.

__ 40. You will be brought back to the Server perspective and will see the IBM EJB Test Client references and parameters in the two right most panes of the workspace. See Figure 2-26 on page 28.


Server - Development Studio Client
File Edit Perspective Project Debug Window Help
Navigator
RemoteSystemsConnections
RemoteSystemsTempFiles
Servers
WebSvcLab
WebSvcLabEar
Web Browser
http://localhost:8080/UTC/preload?object=proxy.soap.com.vck.rst.InventoryAvailabilityServiceProxy
IBM EJB Test Client
References
EJB References
No EJB references available
Object References
InventoryAvailabilityServiceI
Class References
No class references availab
Utilities
Parameters
Server Configuration
Server Instances
WebSphere v4.0 Test Environment
Server Configurations
WebSphere Administrative Domain
Console
Servers Debug Processes Console Variables

A method or constructor has not been selected

Last result: No results

```
[6/7/02 10:36:53:683 CDT] 23016890 WebGroup      I SRVE0091I: [Servlet LOG]: JSP 1.1 Proce
[6/7/02 10:36:53:733 CDT] 23016890 WebGroup      I SRVE0091I: [Servlet LOG]: SimpleFileSer
[6/7/02 10:36:54:284 CDT] 23016890 ServletEngine A SRVE0169I: Loading Web Module: WebSvcLa
[6/7/02 10:36:54:504 CDT] 23016890 WebGroup      I SRVE0091I: [Servlet LOG]: JSP 1.1 Proce
[6/7/02 10:36:54:514 CDT] 23016890 WebGroup      I SRVE0091I: [Servlet LOG]: SimpleFileSer
[6/7/02 10:36:54:544 CDT] 23016890 WebGroup      I SRVE0091I: [Servlet LOG]: InvokerServle
[6/7/02 10:36:54:684 CDT] 23016890 HttpTransport A SRVE0171I: Transport http is listening
[6/7/02 10:36:54:724 CDT] 23016890 Server        A WSVR0023I: Server Default Server open f
[6/7/02 10:37:07:373 CDT] 1f5a689c WebGroup      I SRVE0091I: [Servlet LOG]: preloadServle
```

Figure 2-26 Server perspective showing IBM EJB test client references and parameters

This will conclude the task of creating a Web Service in a Web Project on WDSc.

Lab 3

Testing under WDSc's WebSphere Test Environment

We have completed the process of creating a Web Project and a Web Service on WDSc. Next task would be exporting the set to WAS for deployment. Before that, WDSc provides you a local test option to verify if the set was created all correctly.

__ 1. To test the Web Service in WDSc's WebSphere Test Environment, we will first need to switch to the Web perspective. Select **Perspective -> Open -> Web.**

__ 2. **Expand the WebSvcLab** folder by clicking the "+" next to WebSvcLab. The **source** folder contains the original java file we used to generate the Web service. See Figure 3-1 on page 30 for details.

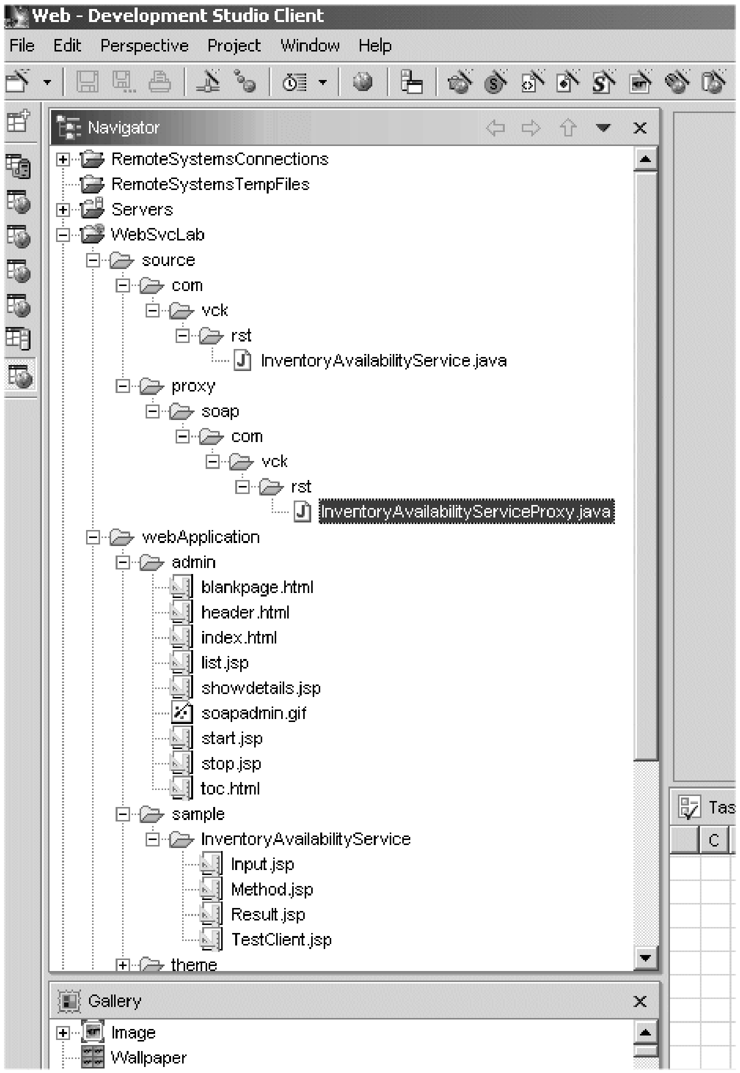


Figure 3-1 Web perspective showing new files created

__ 3. Notice the new file **InventoryAvailabilityServiceProxy.java** which has been created under the subfolder structure **source/proxy/soap/com/vck/rst.**

__ 4. The **webApplication** folder contains the executable class files and other resource files for our newly created Web service.

__ 5. The **Admin** subfolder under webApplications contains a file, **index.html**, which gives the details on all of the running Web services within WDSc.

__ 6. The **sample** subfolder under webApplications contains the TestClient.jsp file and other .jsp resources. We will use this jsp to test our Web service.

__ 7. Expand the directory structure of **WEB-INF** under webApplication. You will see the file **InventoryAvailabilityService.class** at the end of the directory structure. This class file is the resulting bytecodes that were produced by the underlying Java compilation that took place when the Web service was created. These bytecodes are portable to other systems that have a Java Virtual Machine (JVM) environment installed and configured.

__ 8. Now expand the **WSDL** folder. You will see the WSDL XML files, **InventoryAvailabilityService-binding.wsdl** and **InventoryAvailabilityService-service.wsdl**. These WSDL XML files define the framework of the Web service in terms of web serving methods, the parameters in these methods, their return types, and the XML encodings.

__ 9. Your server (that is a local server which lives on WDSc, not your WAS server on iSeries) needs jt400.jar file in its classpath. Open Server perspective by clicking on **Perspective -> Open -> Server**.On Start the server configuration pane on the lower left corner, double click on WebSphere v4.0 Test Environment as shown in Figure 3-2.

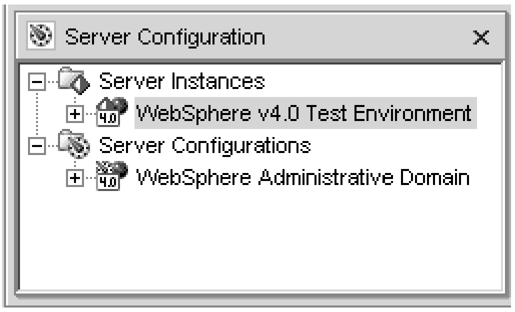
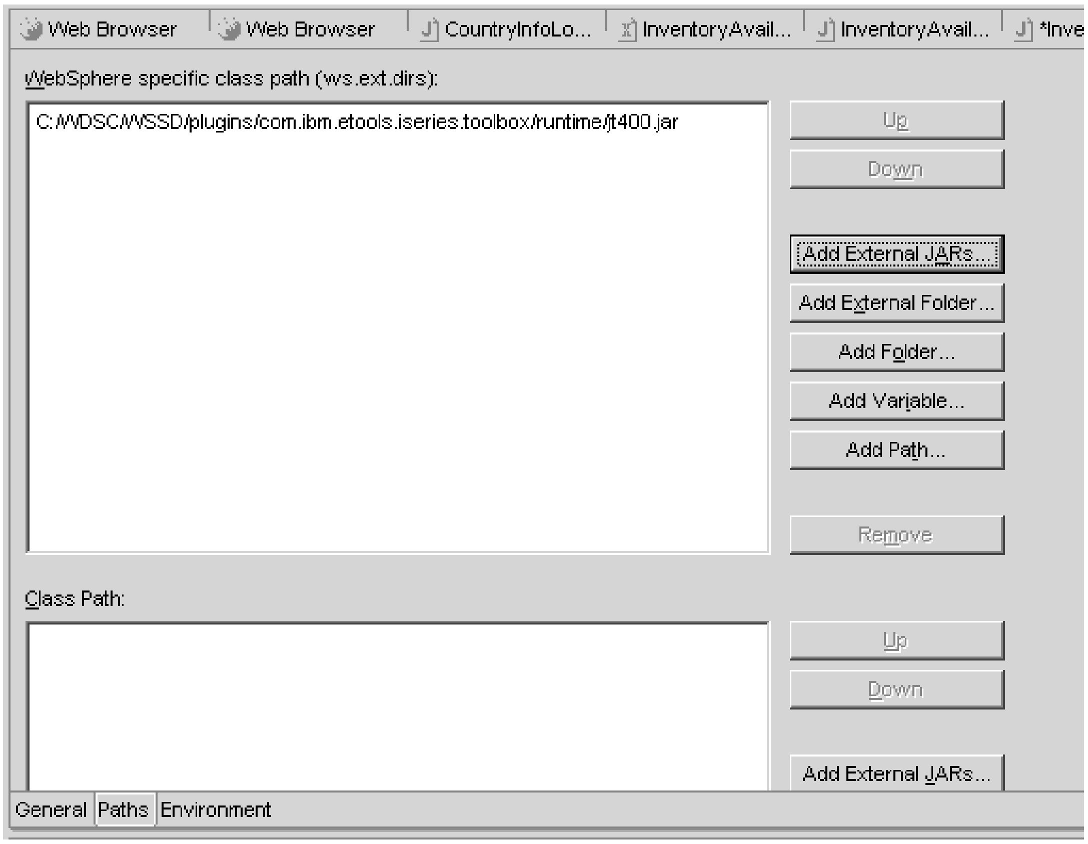

Server Configuration
Server Instances
WebSphere v4.0 Test Environment
Server Configurations
WebSphere Administrative Domain

Figure 3-2 WDSc WebSphere text environment server configuration

__ 10. Click on **Path** tab, then click on **Add External JARs...** button. Then open **C:\WDSC\WSSD\plugins\com.ibm.etools.iseries.toolbox\runtime\jt400.jar** file to get the screen as shown in Figure 3-3.



WebSphere specific class path (ws.ext.dirs):

C:/WDSC/WSSD/plugins/com.ibm.etools.iseries.toolbox/runtime/jt400.jar

Class Path:

Figure 3-3 Adding jt400.jar file to the server

__ 11. Save your changes.

__ 12. Switch to the Server perspective and stop the server if needed:

i. Click the Server tab in the lower right pane of the workbench window
ii. Right-click the server instance and select Stop from the pop-up window (see Figure 3-4)

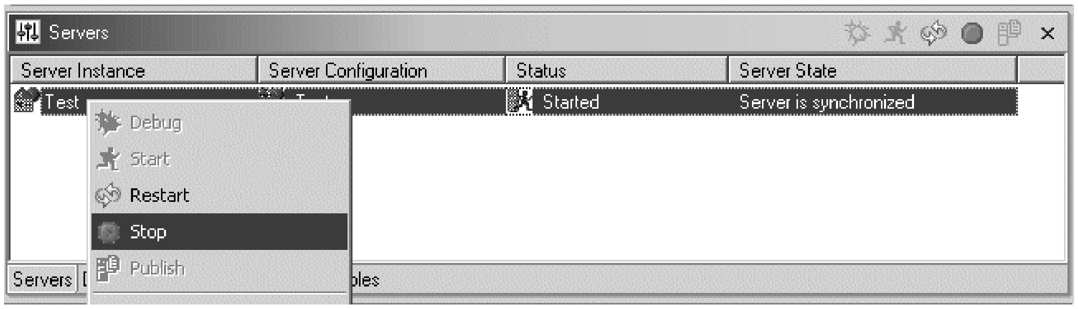


Figure 3-4 Stopping WebSphere Test Environment

__ 13. Now that we have explored the files that were generated for us during the Web Service creation task, we are now ready to test the Web Service on the WDSc client. **Right click** on the **TestClient.jsp** file and select **Run On Server** as shown in Figure 3-5.

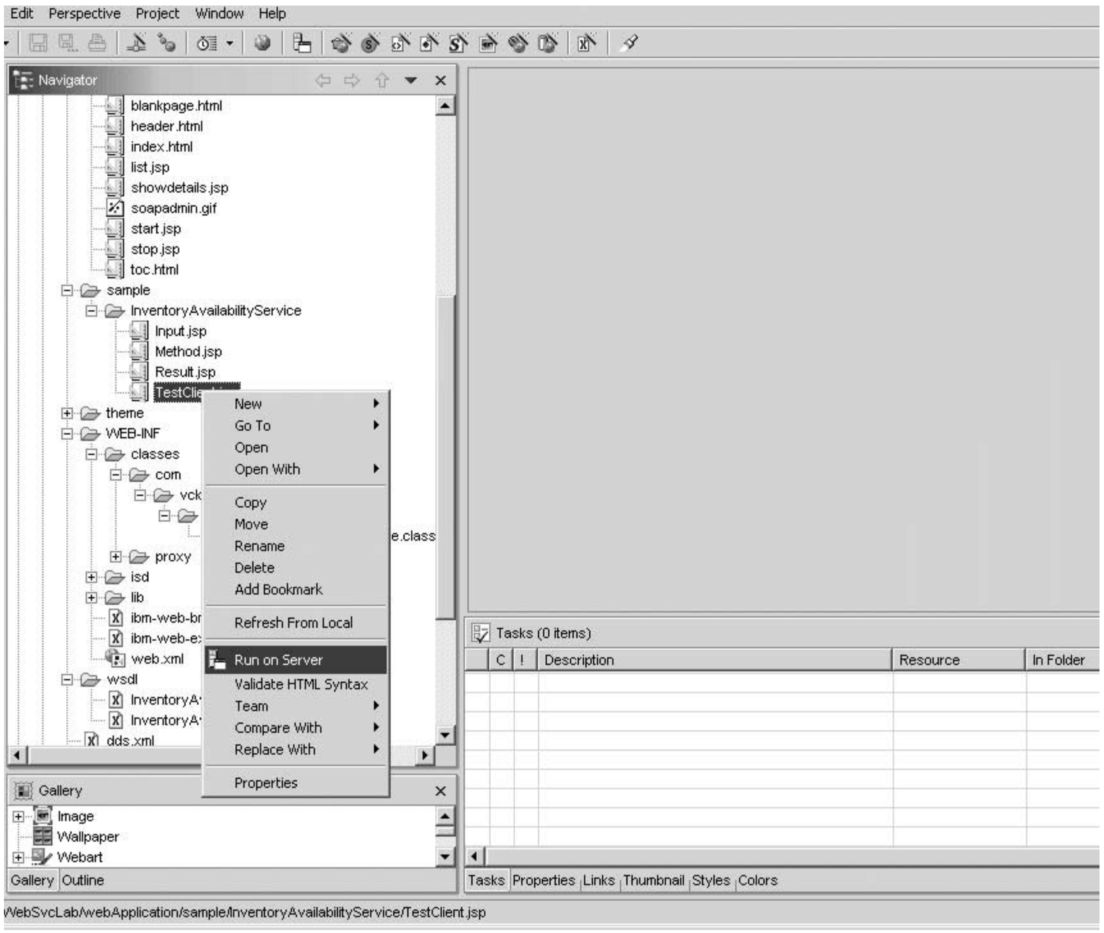


Figure 3-5 Test the Web service by selecting Run On Server

__ 14. You will now see the Methods, Inputs, and Results for your Web service. See Figure 3-6 for details. **Click** on the method **getInventoryAvailability.**

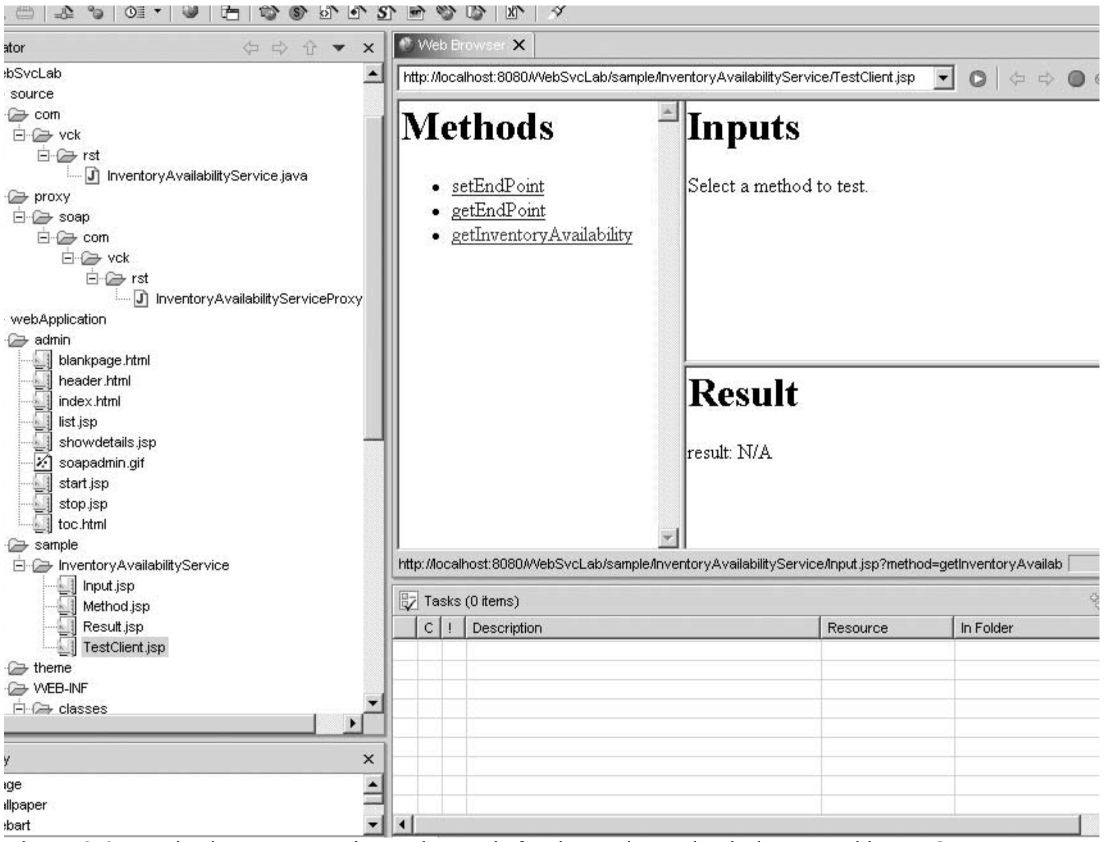

Web Browser
http://localhost:8080/WebSvcLab/sample/InventoryAvailabilityService/TestClient.jsp
Methods
setEndPoint
getEndPoint
getInventoryAvailability
Inputs
Select a method to test.
Result
result: N/A
http://localhost:8080/WebSvcLab/sample/InventoryAvailabilityService/Input.jsp?method=getInventoryAvailab
Tasks (0 items)
Description
Resource
In Folder

Figure 3-6 Methods, Inputs, and Result panels for the Web service being tested in WDSc

__ 15. Enter the value **A001** in the **partNo** field as shown in Figure 3-7. The input parameter is case sensitive.

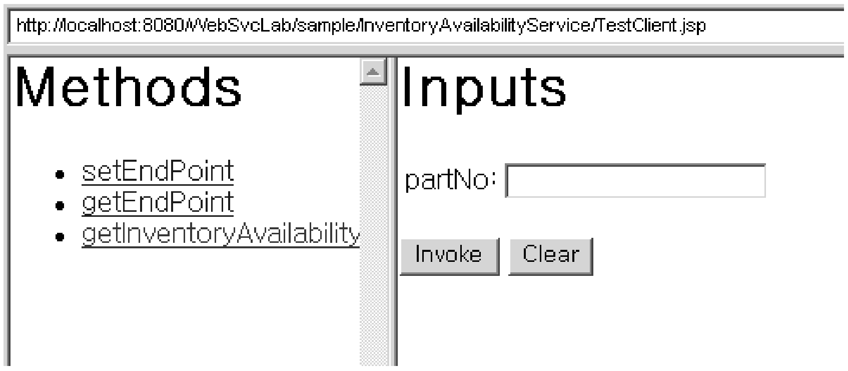

http://localhost:8080/WebSvcLab/sample/InventoryAvailabilityService/TestClient.jsp
Methods
setEndPoint
getEndPoint
getInventoryAvailability
Inputs
partNo:
Invoke
Clear

Figure 3-7 Part number input screen

__ 16. Press the **Invoke** button.

__ 17. You should receive the results of **The item has available quantity 25 by Date 10/15/2002**.

__ 18. You will also see messages on the console showing that a connection has been established to the iSeries to do the inventory lookup using the JDBC connection that is coded in the underlying Java program of your Web service. These messages will be **about to get connection** and **connection obtained (systemname)** where (systemname) is the name of the iSeries server the Java program is connecting into.

__ 19. Stop the WebSphere Test Environment.

Lab 4

Deploying your Web Service in the WebSphere Application Server environment

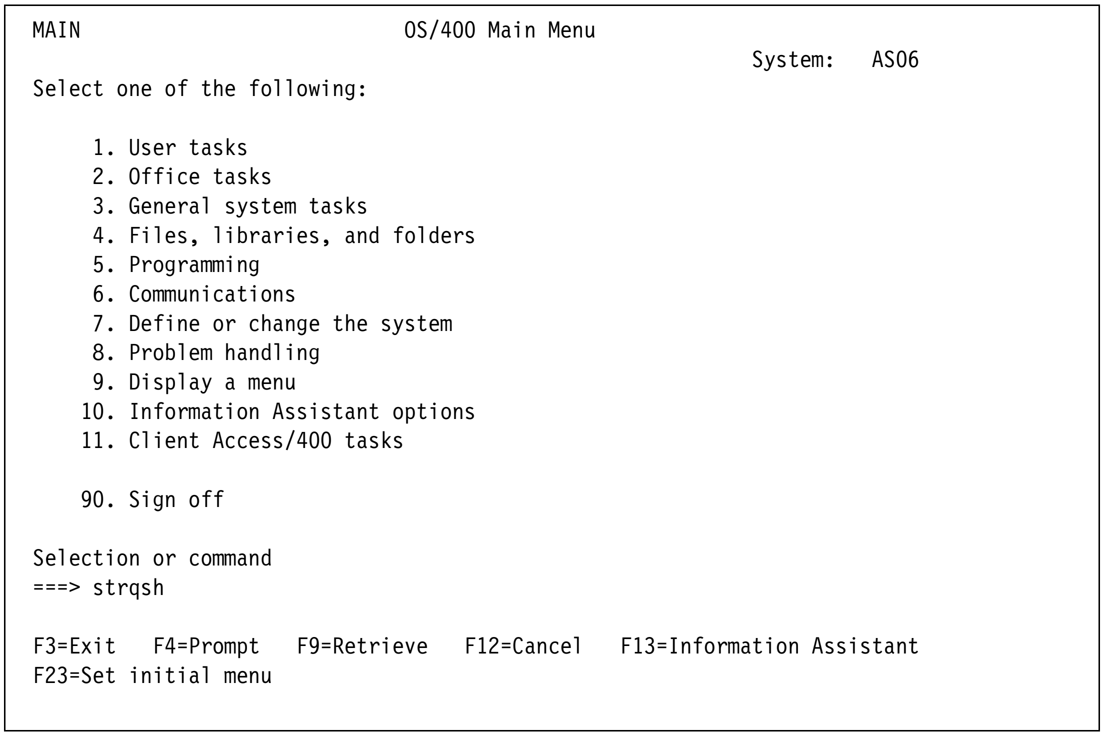
Now we are at the final stage of Web Service creation: Deploying our first Web Service application on WebSphere Application Server (WAS) and testing it from the Web browser. It also is the last sub lab.

Task 1: Creating your own WAS instance

To deploy your Web service on WAS, you will create a new instance called **IWEBxxSVC** on WAS. To create the new instance on iSeries WAS you will need to signon to the iSeries machine.

__ 1. Double-click on the icon for your iSeries to **start a 5250 emulation session**.

__ 2. On the command line, type **strqsh** as shown in Figure 4-1.



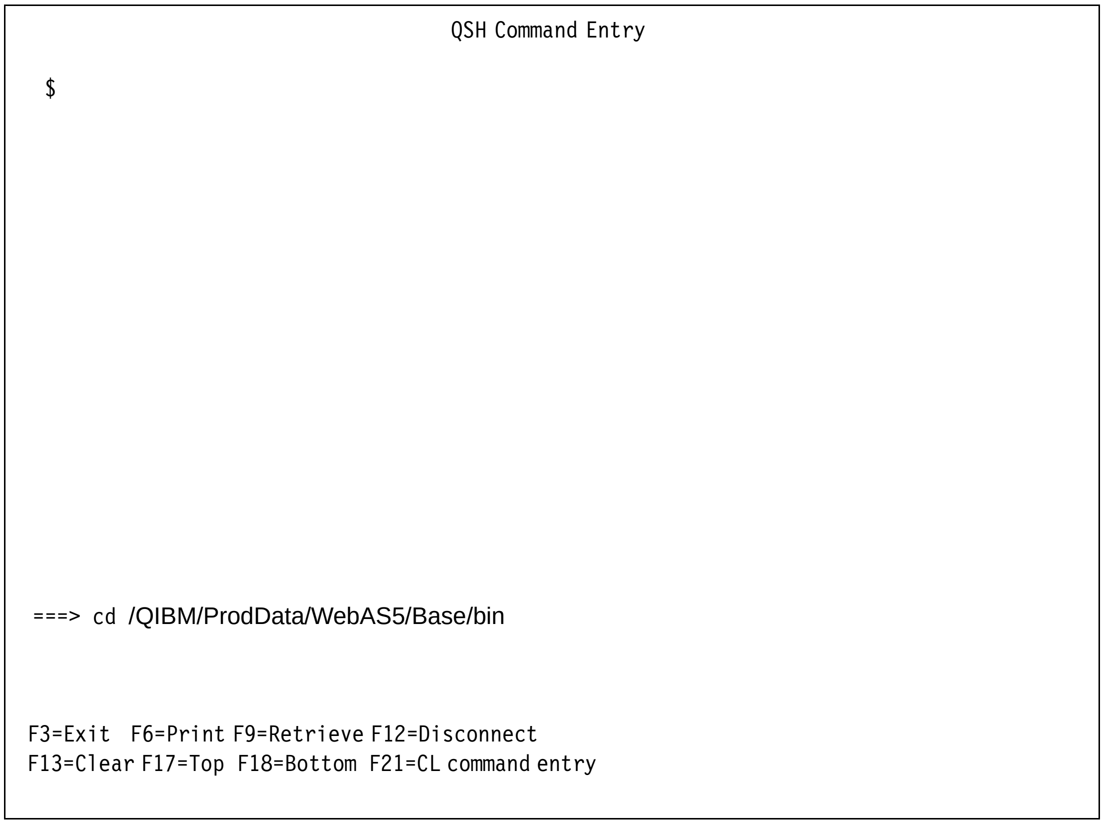
```
MAIN                           OS/400 Main Menu
                                                             System:   AS06
Select one of the following:

     1. User tasks
     2. Office tasks
     3. General system tasks
     4. Files, libraries, and folders
     5. Programming
     6. Communications
     7. Define or change the system
     8. Problem handling
     9. Display a menu
    10. Information Assistant options
    11. Client Access/400 tasks

    90. Sign off

Selection or command
===> strqsh

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel   F13=Information Assistant
F23=Set initial menu
```

Figure 4-1 Type strqsh at the command line to start QSHELL interpreter

__ 3. Press **Enter.**

__ 4. This will open the QSHELL interpreter. The interpreter facilitates the running of Java programs on the iSeries. Type **cd /QIBM/ProdData/WebAS5/Base/bin** as shown in Figure 4-2. This will change the current directory to /QIBM/ProdData/WebAS5/Base/bin.

__ 5. Press **Enter.**



```
                              QSH Command Entry

 $
















===> cd /QIBM/ProdData/WebAS5/Base/bin



F3=Exit  F6=Print F9=Retrieve F12=Disconnect
F13=Clear F17=Top F18=Bottom F21=CL command entry
```

Figure 4-2 Change the directory in QSHELL to /QIBM/ProdData/WebAsAdv4/bin

__ 6. Make sure after entering command that you receive a **"$"** prompt. Type in command **crtwasinst -instance IWEBxxSVC -server IWEBxxSVC -exthttp 88xx -inthttp 55xx -portblock 4xx01** as shown in Figure 4-3.

> **Note:** Substitute your team number for xx in IWEBxxSVC and for each of the ports, 88xx, 55xx, and 4xx01.

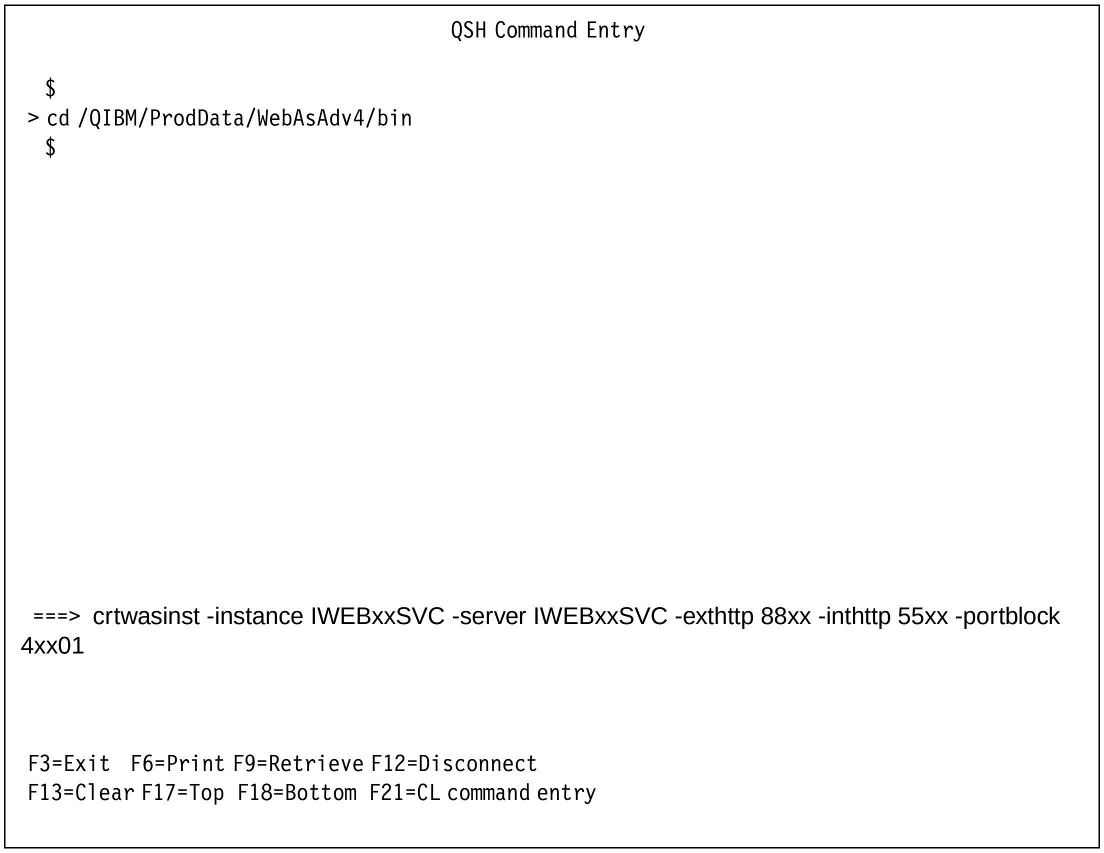

```
                                QSH Command Entry

  $
 > cd /QIBM/ProdData/WebAsAdv4/bin
  $











 ===>  crtwasinst -instance IWEBxxSVC -server IWEBxxSVC -exthttp 88xx -inthttp 55xx -portblock
4xx01



 F3=Exit  F6=Print F9=Retrieve F12=Disconnect
 F13=Clear F17=Top  F18=Bottom  F21=CL command entry
```

Figure 4-3 Use CRTWASINST command in QSHELL to create a new instance of a WAS server

__ 7. Press **Enter.** This process creates an instance on WAS named IWEBxxSVC. This step will take a few minutes. Great time to take a tea break!

__ 8. As the WAS server instance is being created, you will receive the following two messages. It is important to wait for both before continuing.

Creating instance IWEBxxSVC...

Instance IWEBxxSVC created.

Remember: When entering commands in the QSHELL environment, wait for the "**$**" prompt to ensure the command has completed.

__ 9. Now we can started the newly created WAS instance. Issue the command **startServer -instance IWEBxxSVC IWEBxxSVC** on the QSHELL prompt (it's not a typo: you need to key in IWEBxxSVC twice). This will start the IWEBxxSVC application server in the IWEBxxSVC instance.

__ 10. Press **Enter.**

__ 11. You will receive four messages when the WAS server instance is starting. Again, wait for the "$" prompt before continuing. You should get the message similar to:

EJB6123: Application server started.

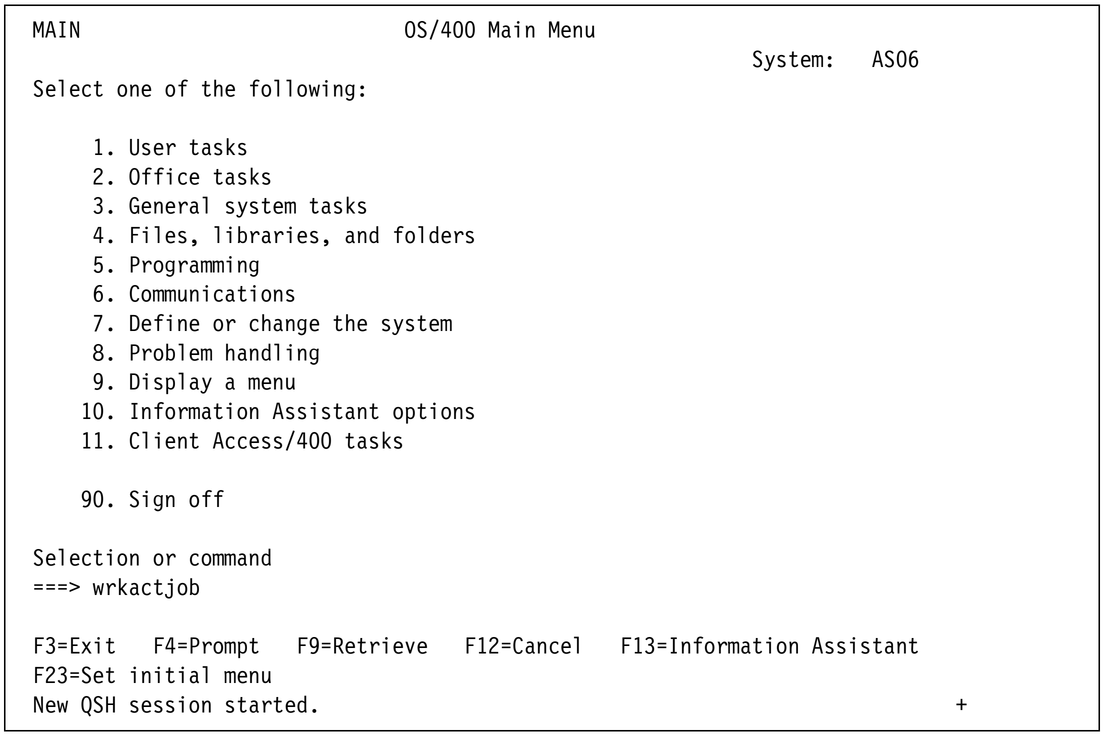
Cause . . . . . : Application server IWEBxxSVC in Base instance IWEBxxSVCj has started

and is ready to accept connections on admin port 4xx10.

__ 12. Press **F3** to exit the QSHELL interpreter and return to the iSeries command line.

__ 13. On the iSeries command line, type **WRKACTJOB** as shown in Figure 4-4.

__ 14. Press **Enter.**



```
MAIN                           OS/400 Main Menu
                                                             System:   AS06
Select one of the following:

     1. User tasks
     2. Office tasks
     3. General system tasks
     4. Files, libraries, and folders
     5. Programming
     6. Communications
     7. Define or change the system
     8. Problem handling
     9. Display a menu
    10. Information Assistant options
    11. Client Access/400 tasks

    90. Sign off

Selection or command
===> wrkactjob

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel   F13=Information Assistant
F23=Set initial menu
New QSH session started.                                                    +
```

Figure 4-4 WRKACTJOB command typed at iSeries command line

__ 15. On the Work With Active Jobs screen, scroll down (use the Page Down key or press the Shift button while pressing the down arrow key) until you see the subsystem named **QEJBAS5**. You should see the job **IWEBxxSVC** with the status of **JVAW**.

We have now ensured that our WAS server instance has been created and started properly on the iSeries.

Task 2: Modifying WSDL files and Proxy application

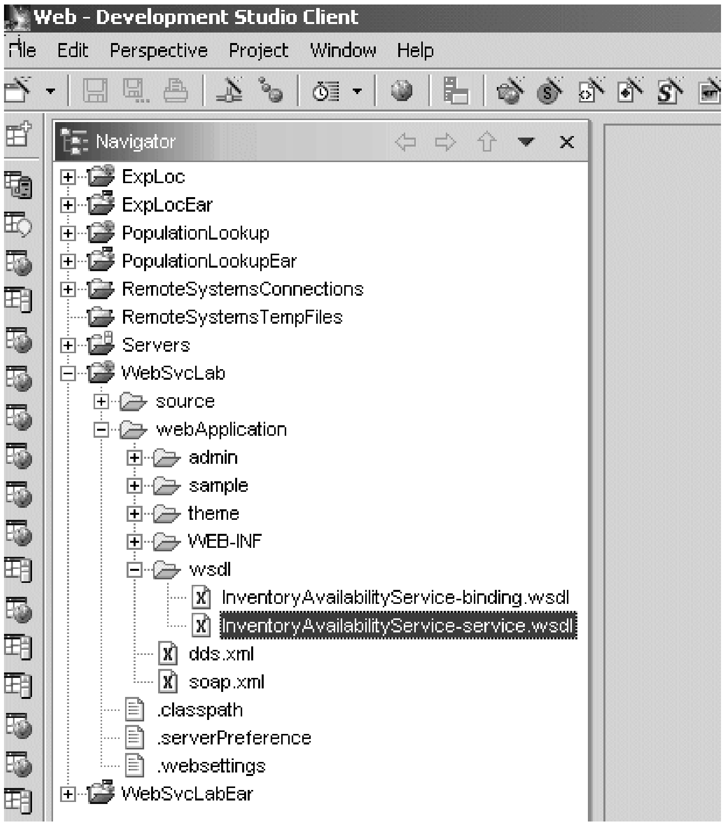
Now we will return to the WDSc client to make some customizations to our InventoryAvailabilityService Web Service.

__ 1. Open the WDSc client if it is not already open on your desktop.

__ 2. Open the **Web perspective** view by selecting **Perspective -> Open -> Web.**

__ 3. Expand the directory structure **WebSvcLab, webApplication, wsdl**.

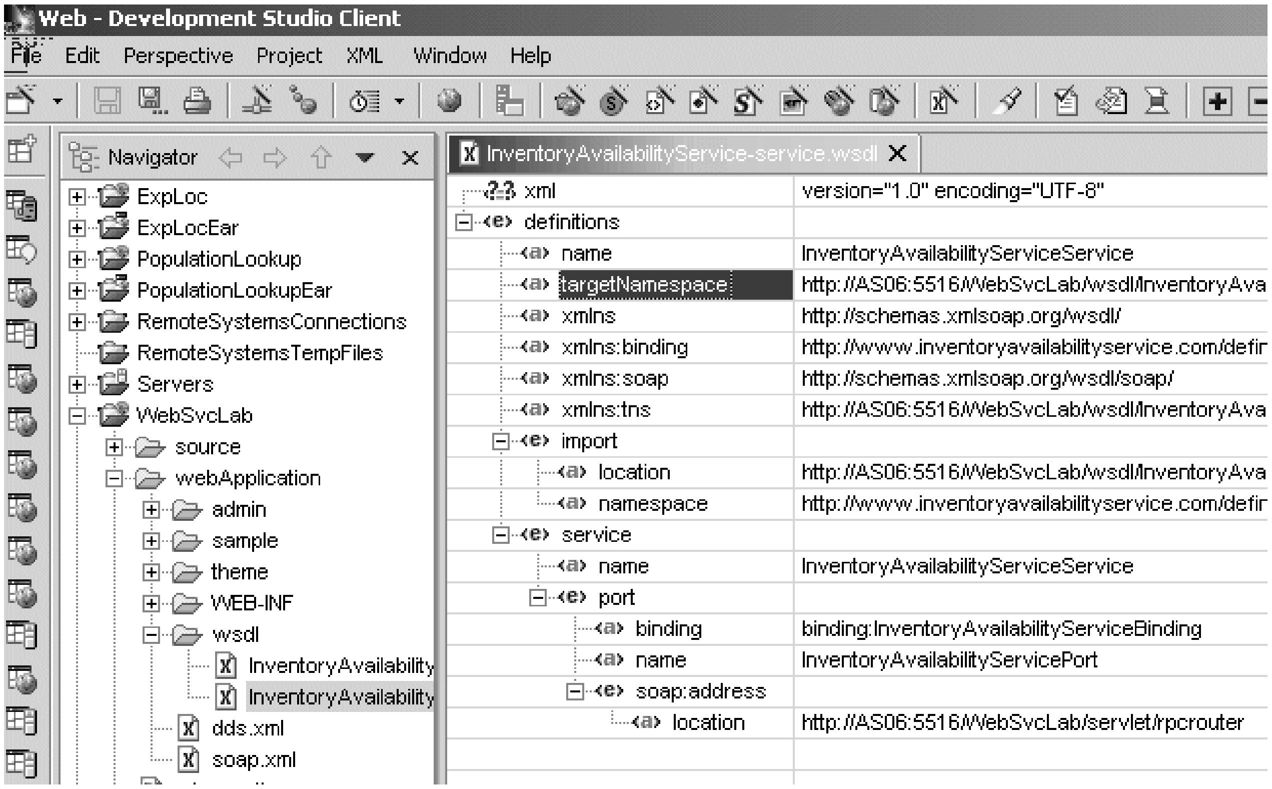
__ 4. Double click the **InventoryAvailabilityService-service.wsdl** file as shown in Figure 4-5.


Web - Development Studio Client
File Edit Perspective Project Window Help
Navigator

ExpLoc
ExpLocEar
PopulationLookup
PopulationLookupEar
RemoteSystemsConnections
RemoteSystemsTempFiles
Servers
WebSvcLab
source
webApplication
admin
sample
theme
WEB-INF
wsdl
InventoryAvailabilityService-binding.wsdl
InventoryAvailabilityService-service.wsdl
dds.xml
soap.xml
.classpath
.serverPreference
.websettings
WebSvcLabEar

Figure 4-5 InventoryAvailabilityService-service.wsdl file in Web perspective view of WebSvcLab project

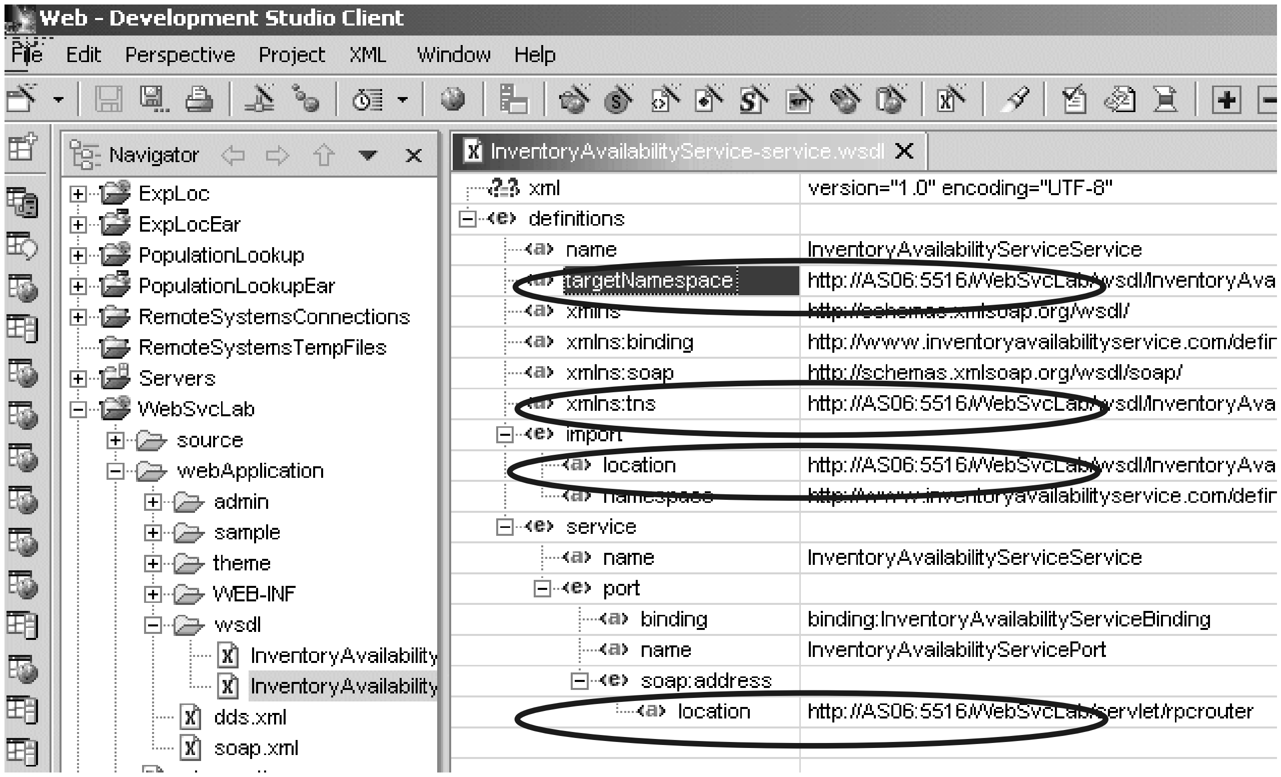
__ 5. In the right pane of the WDSc client, expand **definitions, import,** and **service > port > soap:address** portions of the WDSL file as shown in Figure 4-6.



xml	version="1.0" encoding="UTF-8"
definitions	
name	InventoryAvailabilityServiceService
targetNamespace	http://AS06:5516/WebSvcLab/wsdl/InventoryAva
xmlns	http://schemas.xmlsoap.org/wsdl/
xmlns:binding	http://www.inventoryavailabilityservice.com/defir
xmlns:soap	http://schemas.xmlsoap.org/wsdl/soap/
xmlns:tns	http://AS06:5516/WebSvcLab/wsdl/InventoryAva
import	
location	http://AS06:5516/WebSvcLab/wsdl/InventoryAva
namespace	http://www.inventoryavailabilityservice.com/defir
service	
name	InventoryAvailabilityServiceService
port	
binding	binding:InventoryAvailabilityServiceBinding
name	InventoryAvailabilityServicePort
soap:address	
location	http://AS06:5516/WebSvcLab/servlet/rpcrouter

Figure 4-6 Expanded view of InventoryAvailabilityService-service.wsdl file

__ 6. We need to make changes in four places as highlighted in Figure 4-7.


Web - Development Studio Client
File Edit Perspective Project XML Window Help
Navigator
InventoryAvailabilityService-service.wsdl

```
+ ExpLoc
+ ExpLocEar
+ PopulationLookup
+ PopulationLookupEar
+ RemoteSystemsConnections
  RemoteSystemsTempFiles
+ Servers
- WebSvcLab
    + source
    - webApplication
        + admin
        + sample
        + theme
        + WEB-INF
        - wsdl
            InventoryAvailability
            InventoryAvailability
        dds.xml
        soap.xml
```

xml	version="1.0" encoding="UTF-8"
definitions	
name	InventoryAvailabilityServiceService
targetNamespace	http://AS06:5516/WebSvcLab/wsdl/InventoryAva
xmlns	http://schemas.xmlsoap.org/wsdl/
xmlns:binding	http://www.inventoryavailabilityservice.com/defir
xmlns:soap	http://schemas.xmlsoap.org/wsdl/soap/
xmlns:tns	http://AS06:5516/WebSvcLab/wsdl/InventoryAva
import	
location	http://AS06:5516/WebSvcLab/wsdl/InventoryAva
namespace	http://www.inventoryavailabilityservice.com/defir
service	
name	InventoryAvailabilityServiceService
port	
binding	binding:InventoryAvailabilityServiceBinding
name	InventoryAvailabilityServicePort
soap:address	
location	http://AS06:5516/WebSvcLab/servlet/rpcrouter

Figure 4-7 Changes in InventoryAvailabilityService-service.wsdl file

__ 7. We now need to change the Web Project to point to the WSDL files on the iSeries HTTP server rather than the local HTTP server. Again, in this lab, we will use the HTTP stack which is in WAS, rather than an external HTTP server instance. We will hardcode the host name of the iSeries where WAS is running, for example "**AS06**". Replace this with your iSeries host name. The local server, **localhost:8080**, is specified by default. We need to change the WSDL file to point to the HTTP server on the iSeries (**AS06:55xx** where xx is your team number**)** rather than the localhost.

Note: Replace AS06 with the host name of your lab iSeries machine.

a. Click on the string **http://localhost:8080/WebSvcLab/wsdl/InventoryAvailabilityService-service.wsdl** across from **targetNamespace**

- Change this string to **http://AS06:55xx/WebSvcLab/wsdl/InventoryAvailabilityService-service.wsdl**

b. Now click on the string **http://localhost:8080/WebSvcLab/wsdl/InventoryAvailabilityService-service.wsdl** across from **xnlns:tns**

- Change this string to **http://AS06:55xx/WebSvcLab/wsdl/InventoryAvailabilityService-service.wsdl**

c. Click on the string **http://localhost:8080/WebSvcLab/wsdl/InventoryAvailabilityService-binding.wsdl** across from **import > location**

- Change this string to **http://AS06:55xx/WebSvcLab/wsdl/InventoryAvailabilityService-binding.wsdl**

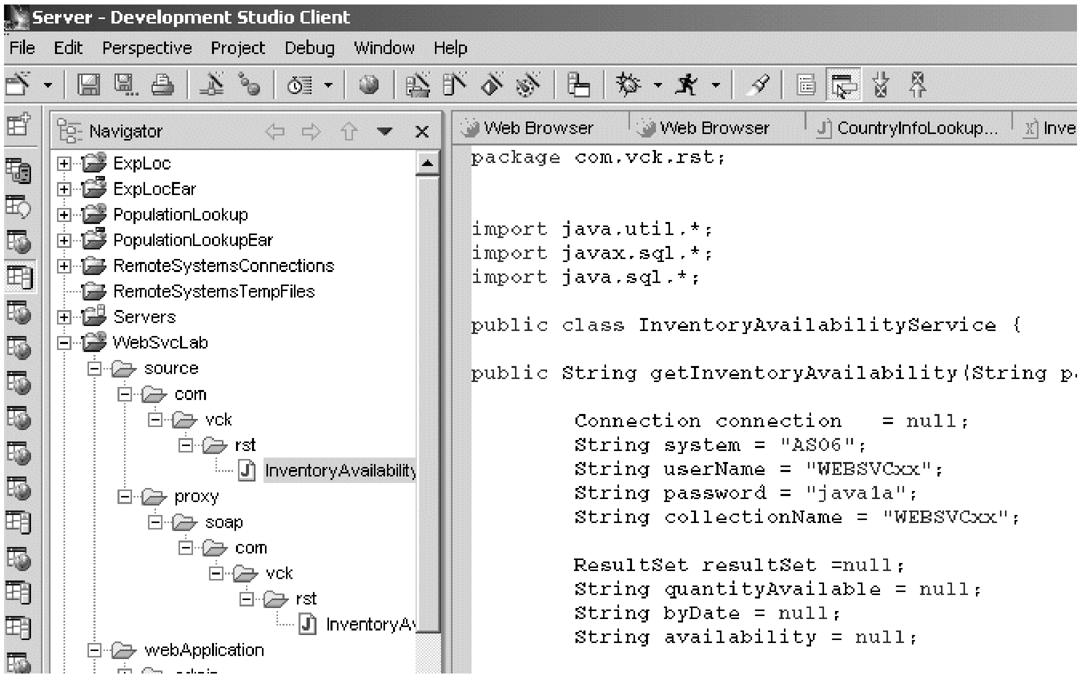
d. Click on the string **http://localhost:8080/WebSvcLab/servlet/rpcrouter** across from **service > port > soap:address > location**

- Change this string to **http://AS06:55xx/WebSvcLab/servlet/rpcrouter**

__ 8. We can now save our changes. Select **File -> Save InventoryAvailabilityService-service.wsdl** from the pull-down menu.

__ 9. You will see a **Progress Information** dialog box with a message of **Setting Contents** and a progress indicator bar. Wait for the updates to be saved before continuing.

__ 10. We also need to change the **InventoryAvailabilityServiceProxy.java** file in source directory. Expand the directory structure in the left pane for **WebSvcLab > source > proxy > soap > com > vck > rst** as shown in Figure 4-8.

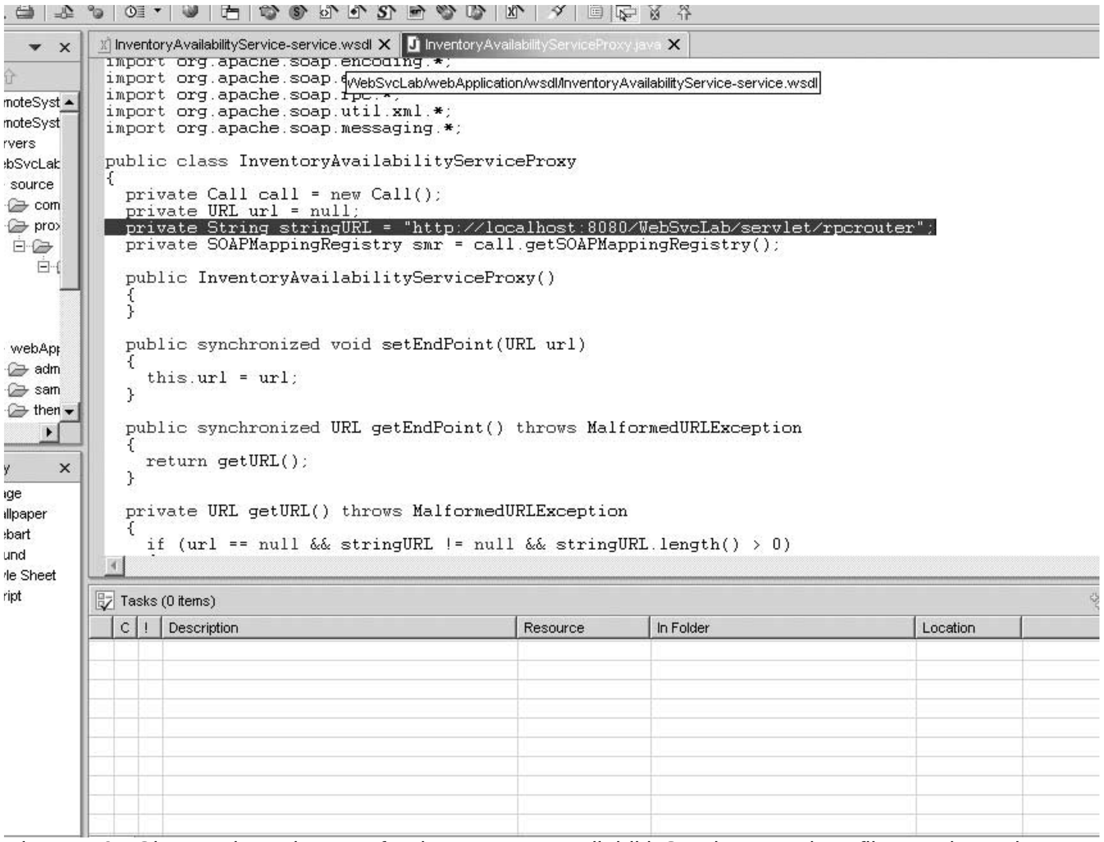


```
package com.vck.rst;


import java.util.*;
import javax.sql.*;
import java.sql.*;

public class InventoryAvailabilityService {

public String getInventoryAvailability(String p

        Connection connection   = null;
        String system = "AS06";
        String userName = "WEBSVCxx";
        String password = "java1a";
        String collectionName = "WEBSVCxx";

        ResultSet resultSet =null;
        String quantityAvailable = null;
        String byDate = null;
        String availability = null;
```

Figure 4-8 Expand the WebSvcLab project directory structure to locate the InventoryAvailabilityServiceProxy.java source file

__ 11. Double click the **InventoryAvailabilityServiceProxy.java** file to open it in the right pane.

__ 12. You will see the stringURL in blue, see Figure 4-9.


InventoryAvailabilityService-service.wsdl
InventoryAvailabilityServiceProxy.java
WebSvcLab/webApplication/wsdl/InventoryAvailabilityService-service.wsdl
Tasks (0 items)
C
!
Description
Resource
In Folder
Location

```
import org.apache.soap.encoding.*;
import org.apache.soap.e
import org.apache.soap.rpc.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.messaging.*;

public class InventoryAvailabilityServiceProxy
{
  private Call call = new Call();
  private URL url = null;
  private String stringURL = "http://localhost:8080/WebSvcLab/servlet/rpcrouter";
  private SOAPMappingRegistry smr = call.getSOAPMappingRegistry();

  public InventoryAvailabilityServiceProxy()
  {
  }

  public synchronized void setEndPoint(URL url)
  {
    this.url = url;
  }

  public synchronized URL getEndPoint() throws MalformedURLException
  {
    return getURL();
  }

  private URL getURL() throws MalformedURLException
  {
    if (url == null && stringURL != null && stringURL.length() > 0)
```

Figure 4-9 Change the stringURL for the InventoryAvailabiltiyServiceProxy.java file to point to the iSeries http server

__ 13. Change the stringURL from **http://localhost:8080/WebSvcLab/servlet/rpcrouter** to **http://AS06:55xx/WebSvcLab/servlet/rpcrouter.**

Note: Again, replace AS06 with the host name of your lab iSeries machine.

__ 14. Save your changes by selecting **File -> Save InventoryAvailabilityServiceProxy.java** from the pull-down menu.

Task 3: Exporting Ear file to WAS

We are now ready to export the Web service for deployment to our WAS instance on the iSeries. To deploy the InventoryAvailabilityService Web service on the instance **IWEBxxSVC** we need to export it to WAS.

__ 1. On WDSc, highlight **WebSvcLab** project.

__ 2. Select **File -> Export** from the pull-down menu.

__ 3. In the Select Export an Enterprise Application project into an EAR file, select **EAR** file as shown in Figure 4-10.

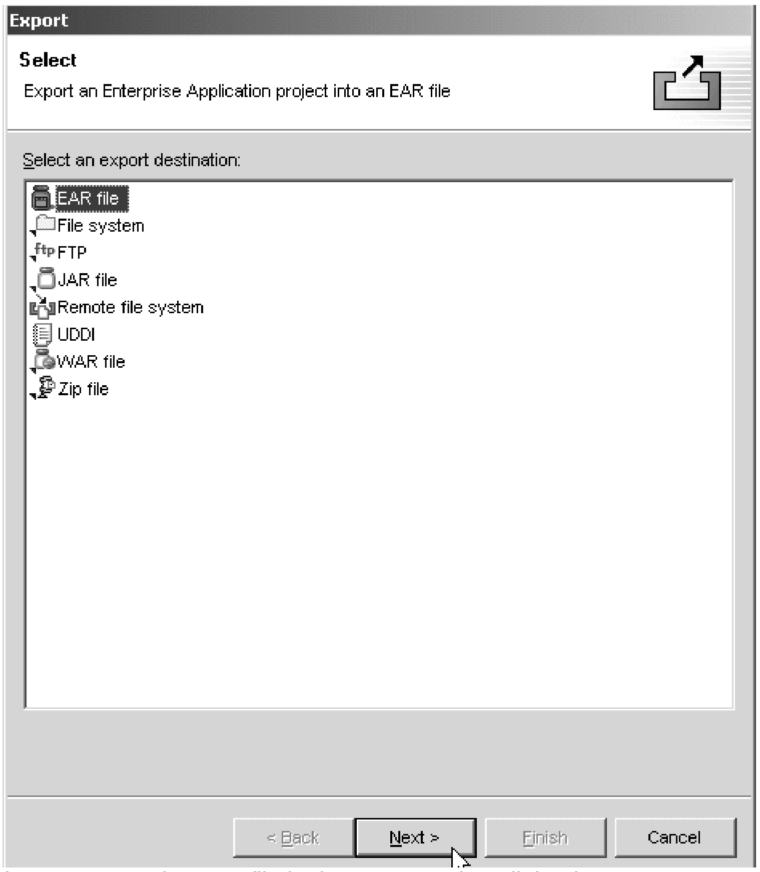
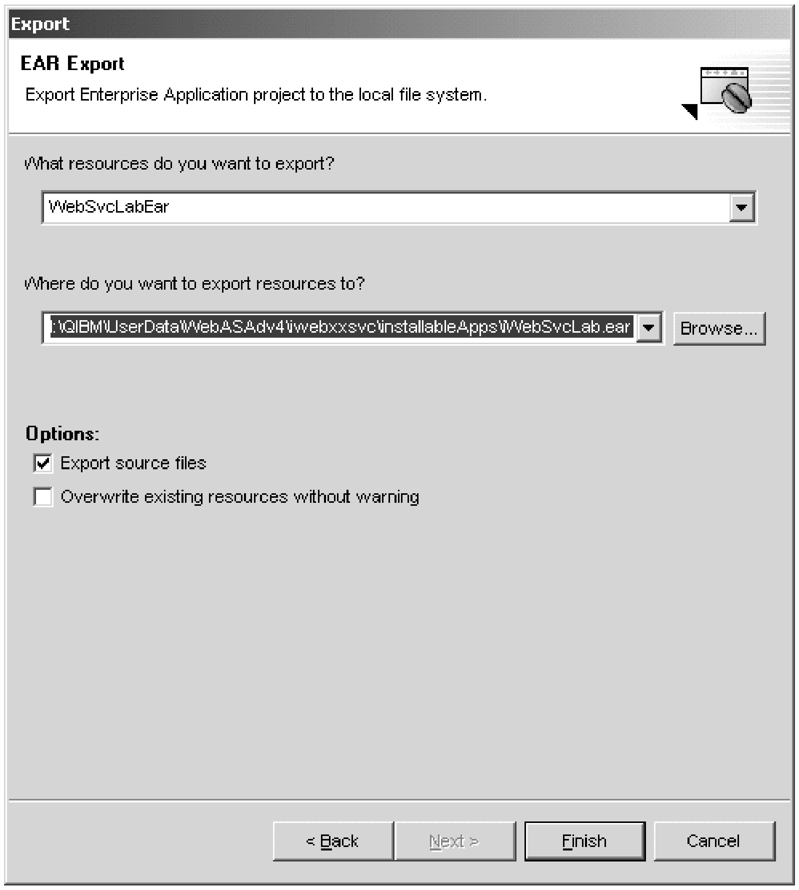


Figure 4-10 Select Ear file in the Export Select dialog box

__ 4. Press **Next** to continue.

__ 5. In the EAR Export dialog box, click the pull-down tab for the “What resources do you want to export?” box and select **WebSvcLabEar.** Refer to Figure 4-11 for this step in addition to the following steps.


Export
EAR Export
Export Enterprise Application project to the local file system.
< Back
Next >
Finish
Cancel
Browse...

What resources do you want to export?

WebSvcLabEar

Where do you want to export resources to?

:\QIBM\UserData\WebASAdv4\iwebxxsvc\installableApps\WebSvcLab.ear

Options:

- [✓] Export source files
- [ ] Overwrite existing resources without warning

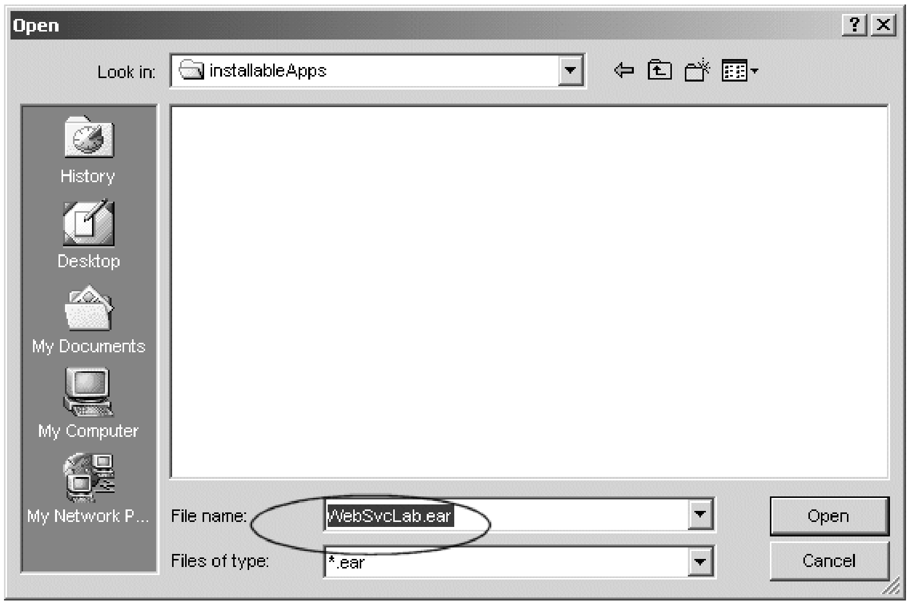
Figure 4-11 EAR export options

__ 6. In the input box for “Where do you want to export resources to?”, click the **Browse** button.

__ 7. Select the directory structure of **H:\QIBM\UserData\WebAS5\Base\IWEBxxSVC\installableApps**

> **Note:** Select **installableApps** in the directory structure rather than **installedApps.**

__ 8. Type in **WebSvcLab.ear** in the file name prompt as shown in Figure 4-12.



File name: WebSvcLab.ear

Files of type: *.ear

Figure 4-12 Type in the name of the ear file to be created with the export and press Open

__ 9. Press **Open.**

__ 10. Click the check box next to **Export source files** as shown in Figure 4-11.

__ 11. Press **Finish** to export the project to your WAS instance on the iSeries.

Task 4: Verifying that the application server has started

Before you start the administrative console (which we do in the next lab exercise), verify that the environment started successfully. When the WebSphere Application Server environment is ready for use, a message is written to the job log of the application server job indicating that the WebSphere Application Server environment is ready.

To determine if the WebSphere Application Server environment is ready, perform these steps from an OS/400 command line:

__ 1. Run the Work with Active Jobs (WRKACTJOB) command, specifying the appropriate subsystem on the subsystem (SBS) parameter. For WebSphere Application Server (Base), use the QEJBAS5 subsystem:

```
WRKACTJOB SBS(QEJBAS5)
```

__ 2. Find your team’s application server job.

__ 3. Specify option 5 (Work with Job) on the option line next to the job.

__ 4. Press Enter.

__ 5. On the command line, specify option `10` (Display job log).

__ 6. Press Enter.

__ 7. Press F10

__ 8. Look for this message:

```
WebSphere application server application_server ready.
```

Here *application_server* is the name of your application server.

If the message is not displayed, press F5 to refresh the job log messages until the message is displayed. When the message is displayed, the WebSphere Application Server environment has successfully started. It may take up to 20 minutes for the message to be displayed, depending on your iSeries server. If the message is not displayed, see your instructor.

__ 9. To display the port number on which the application server is listening for the administrative console, position the cursor on the last line of the message.

__ 10. Press F1. This message is displayed:

```
WebSphere application server application_server in job app_server_job is ready to
handle administrative requests on port port_number
```

Here *application_server* is the name of your application server, *app_server_job* is the OS/400 job name for your application server, and *port_number* is the number of the port used by the administrative console.

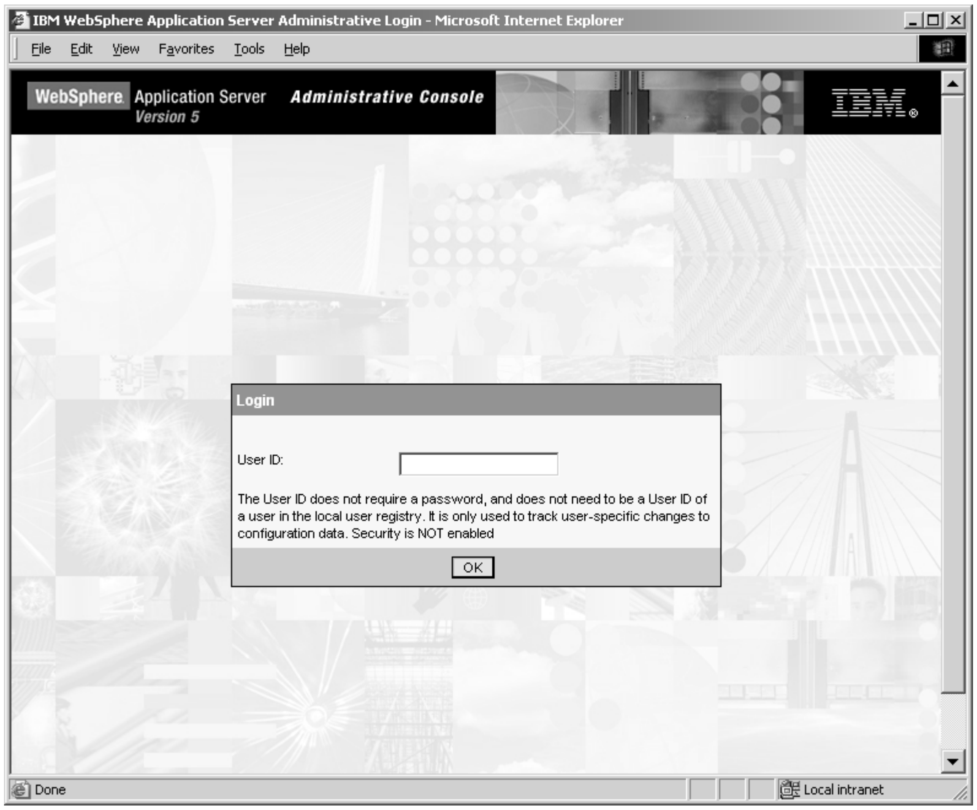
__ 11. Press F3 twice to exit.

Task 5: Web Service application installation on WAS

Next task is installing Web Service application on WAS. For the next set of steps we need to connect to the adminconsole application running in WAS (this is a replacement for the administrative console):

__ 1. Point your Web browser to http://<serverName>:<portNumber>/admin, where *serverName* is your class iSeries, and *portNumber* is the port on which the adminconsole is listening for incoming requests.

__ 2. You should see screen similar to Figure 4-13.


IBM WebSphere Application Server Administrative Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
WebSphere Application Server Version 5 Administrative Console
IBM
Login
User ID:
OK
Done
Local intranet

The User ID does not require a password, and does not need to be a User ID of a user in the local user registry. It is only used to track user-specific changes to configuration data. Security is NOT enabled

Figure 4-13 Login page

__ 3. Enter your team's user ID and click **OK**.

__ 4. Expand **Applications** in the navigation tree.

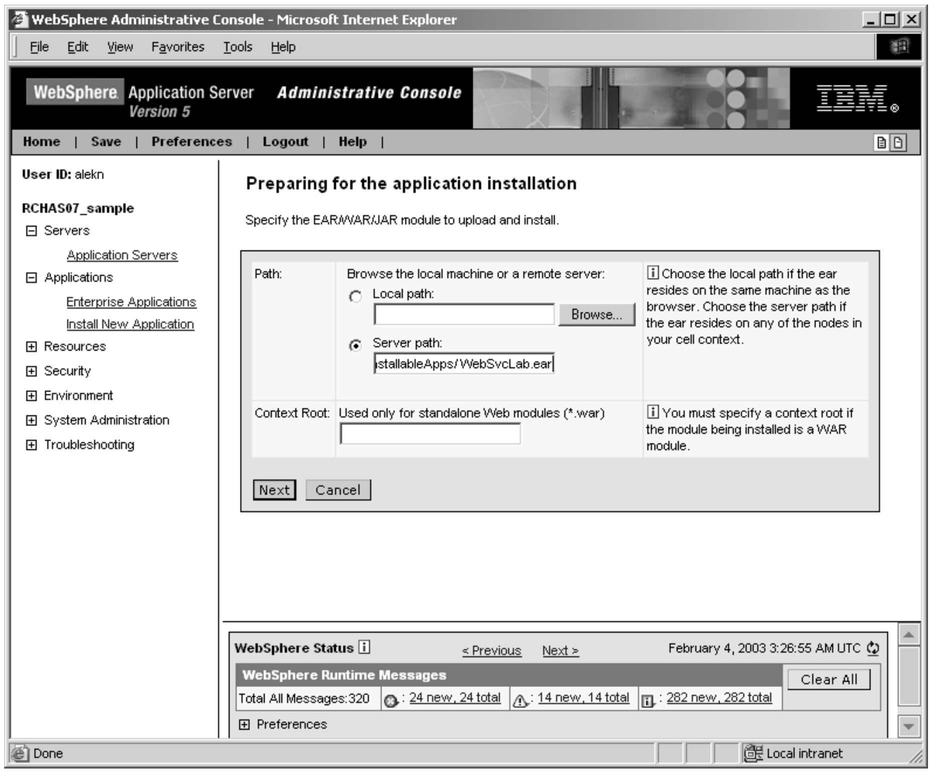
__ 5. Click **Install New Application**.

__ 6. On the next screen provide the location of the .EAR file. You should select the ***Server path*** radio button and provide the directory name to where you have exported your application. Again, click on Server path radio button and key in:

/QIBM/UserData/WebAS5/Base/IWEBxxSVC/installableApps/WebSvcLab.ear

Important: This is the directory and file name on iSeries.

__ 7. Click **Next** (see Figure 4-14).



Preparing for the application installation

Specify the EAR/WAR/JAR module to upload and install.

Path:	Browse the local machine or a remote server: ( ) Local path: [✓ Server path:] stallableApps/WebSvcLab.ear	Choose the local path if the ear resides on the same machine as the browser. Choose the server path if the ear resides on any of the nodes in your cell context.
Context Root:	Used only for standalone Web modules (*.war)	You must specify a context root if the module being installed is a WAR module.

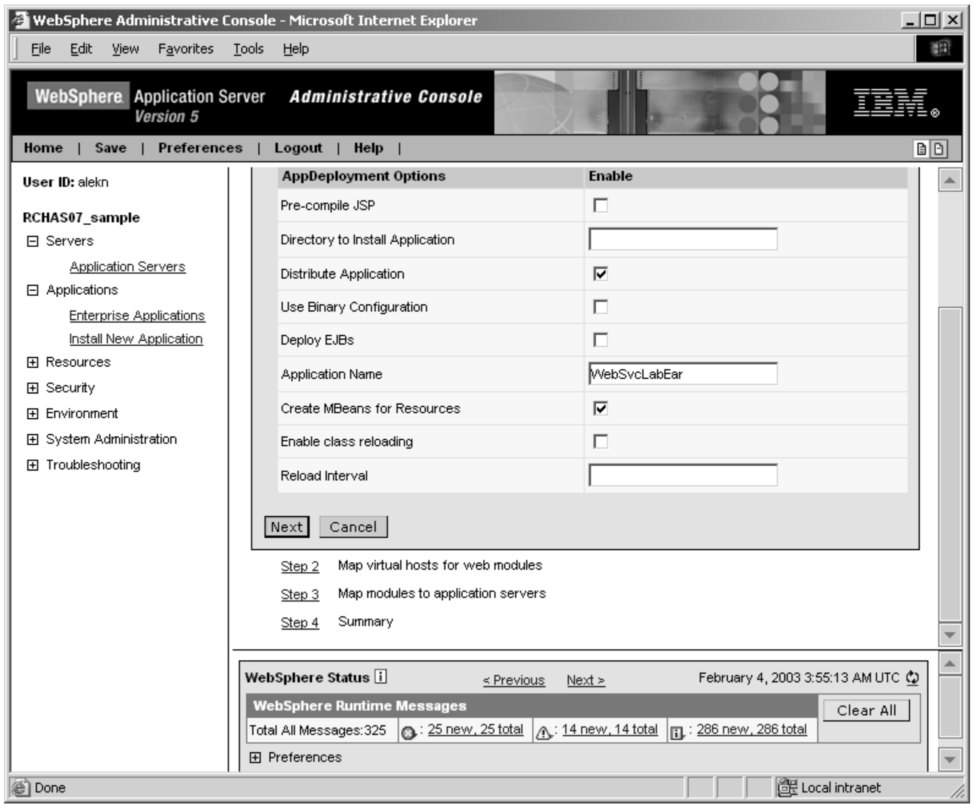
February 4, 2003 3:26:55 AM UTC

WebSphere Runtime Messages

Total All Messages:320 : 24 new, 24 total : 14 new, 14 total : 282 new, 282 total

Figure 4-14 Selecting the file to install

__ 8. The installation wizard starts. Take the default values and click **Next** to reach the screen looking like Figure 4-15



AppDeployment Options	Enable
Pre-compile JSP	☐
Directory to Install Application	
Distribute Application	✓
Use Binary Configuration	☐
Deploy EJBs	☐
Application Name	WebSvcLabEar
Create MBeans for Resources	✓
Enable class reloading	☐
Reload Interval	

Step 2 Map virtual hosts for web modules
Step 3 Map modules to application servers
Step 4 Summary

WebSphere Status < Previous Next > February 4, 2003 3:55:13 AM UTC
WebSphere Runtime Messages
Total All Messages:325 : 25 new, 25 total : 14 new, 14 total : 286 new, 286 total

Figure 4-15 Installation wizard

__ 9. Take the default values again and clike on **Step 4 Summary**.

__ 10. Review the summary page and click **Finish**.

__ 11. After a few moments you should see a new page with the following message: `Application WebSvcLabEar installed successfully.`

__ 12. Click **Save to Master Configuration** to save your changes

__ 13. Click the **Save** button on the confirmation page.

__ 14. We need to add one more jar file to the classpath. This jar file contains the class which implements the JDBC driver. Expand **Servers** in the navigation tree.

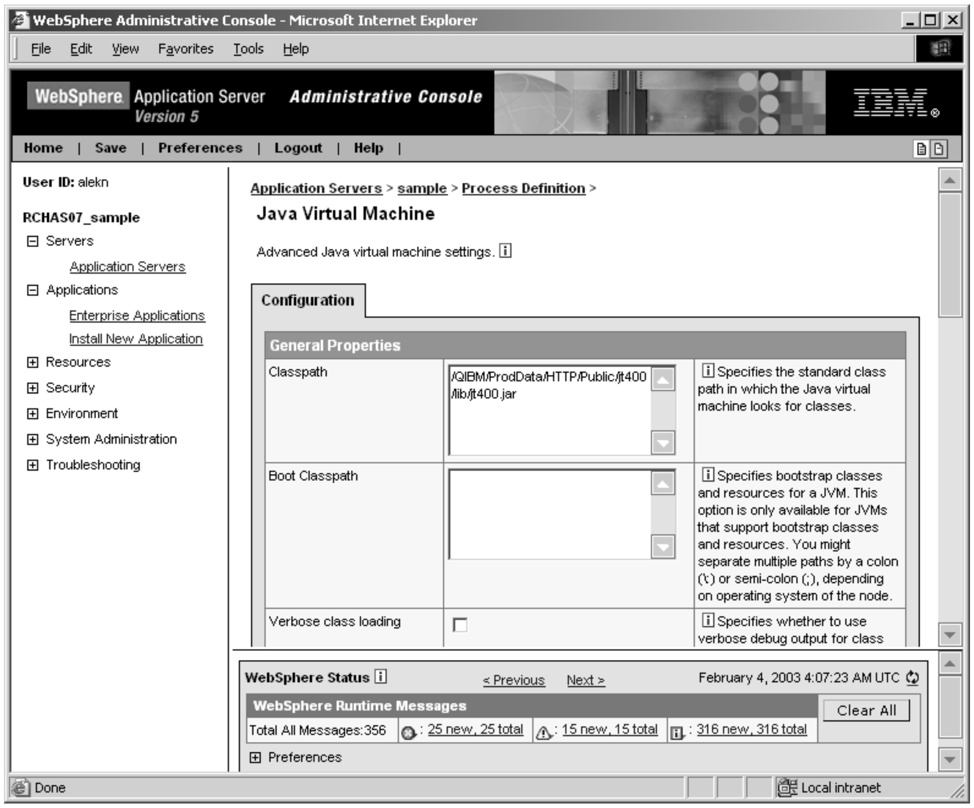
__ 15. Click **Application Servers**.

__ 16. Click your server name in the main frame.

__ 17. Scroll down and click **Process Definition**.

__ 18. A process definition pane appears. Scroll down and click **Java Virtual Machine**.

__ 19. Add /QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar to the class path as shown in Figure 4-16.


WebSphere Administrative Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
WebSphere Application Server Version 5 Administrative Console
Home | Save | Preferences | Logout | Help |
User ID: alekn
RCHAS07_sample
Servers
Application Servers
Applications
Enterprise Applications
Install New Application
Resources
Security
Environment
System Administration
Troubleshooting
Application Servers > sample > Process Definition >
Configuration
General Properties
Done
Local intranet

Java Virtual Machine

Advanced Java virtual machine settings.

Classpath	/QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar	Specifies the standard class path in which the Java virtual machine looks for classes.
Boot Classpath		Specifies bootstrap classes and resources for a JVM. This option is only available for JVMs that support bootstrap classes and resources. You might separate multiple paths by a colon (':') or semi-colon (;), depending on operating system of the node.
Verbose class loading	☐	Specifies whether to use verbose debug output for class

WebSphere Status < Previous Next > February 4, 2003 4:07:23 AM UTC

WebSphere Runtime Messages — Clear All

Total All Messages:356 : 25 new, 25 total : 15 new, 15 total : 316 new, 316 total

Preferences

Figure 4-16 Updating classpath

__ 20. Click **OK**.

__ 21. Save your changes.

__ 22. Click Enterprise Applications in the navigation tree. You should see your application stopped.

__ 23. Select a checkbox on the left of the application name and click the **Start** button.

__ 24. Your application should start: the icon for the application status will change to green arrow.

__ 25. You need to restart the application server. Go to 5250 screen and start Qshell utility. Then run stopServer and startServer scripts:

i. cd /QIBM/ProdData/WebAS5/Base/bin

ii. stopServer -instance IWEBxxSVC IWEBxxSVC

iii. startServer -instance IWEBxxSVC IWEBxxSVC

Task 6: Test Web Service application from Web browser

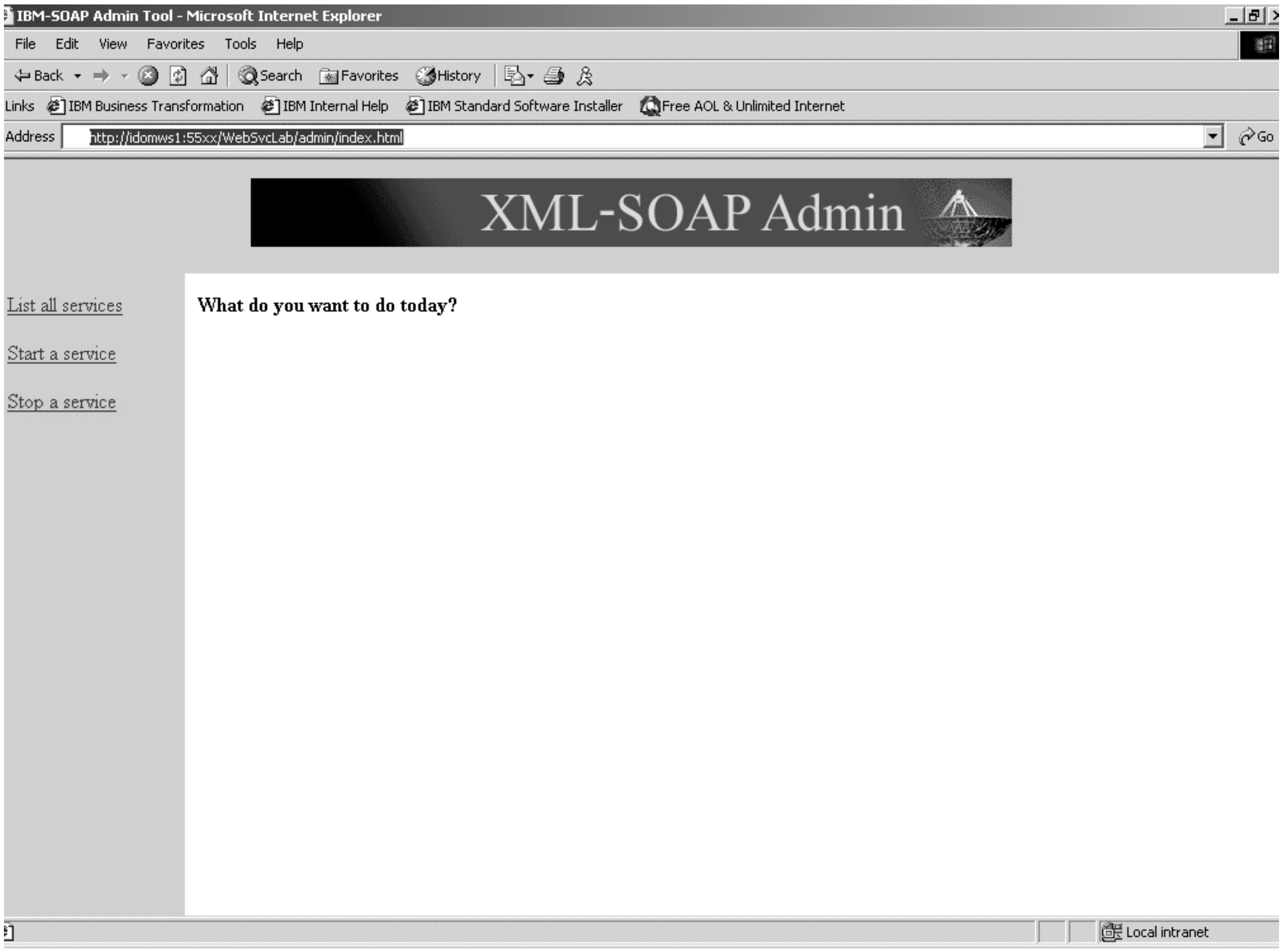
We can now verify that the Web Service has been deployed successfully on our iSeries WAS server instance. We will go to a browser session and access our Web service from the iSeries HTTP server which will route request for our Web service to our specific WAS server instance.

__ 1. Open an **Internet Explorer** browser session.

__ 2. Type **http://AS06:55xx/WebSvcLab/admin/index.html** for the URL. Replace “AS06” with the host name of your iSeries. Remember to replace xx with your team number for the port number (55xx).

__ 3. Press **Enter** to access the Web Service.

__ 4. You should see the XML-SOAP Admin as shown in Figure 4-17.


IBM-SOAP Admin Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Links IBM Business Transformation IBM Internal Help IBM Standard Software Installer Free AOL & Unlimited Internet
Address
Go
XML-SOAP Admin
List all services
Start a service
Stop a service
Local intranet

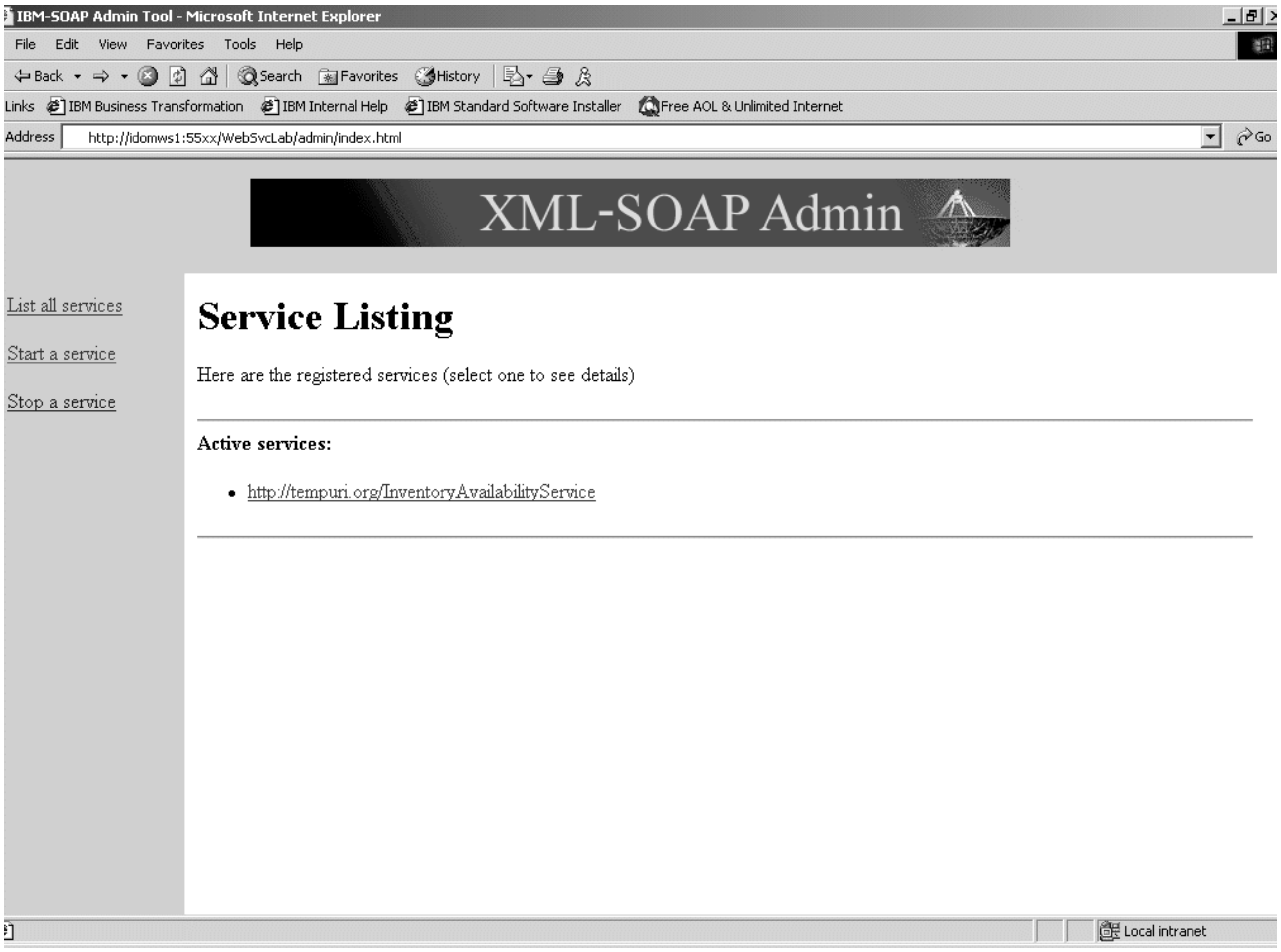
http://idomws1:55xx/WebSvcLab/admin/index.html

What do you want to do today?

Figure 4-17 XML-SOAP Admin welcome screen

__ 5. On the XML-SOAP Admin welcome page, click on **List all services**.

__ 6. The resulting page can be seen in Figure 4-18. We know our Web Service is available to be invoked.


IBM-SOAP Admin Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Links IBM Business Transformation IBM Internal Help IBM Standard Software Installer Free AOL & Unlimited Internet
Address http://idomws1:55xx/WebSvcLab/admin/index.html
Go
XML-SOAP Admin
List all services
Start a service
Stop a service
Local intranet

Service Listing

Here are the registered services (select one to see details)

Active services:

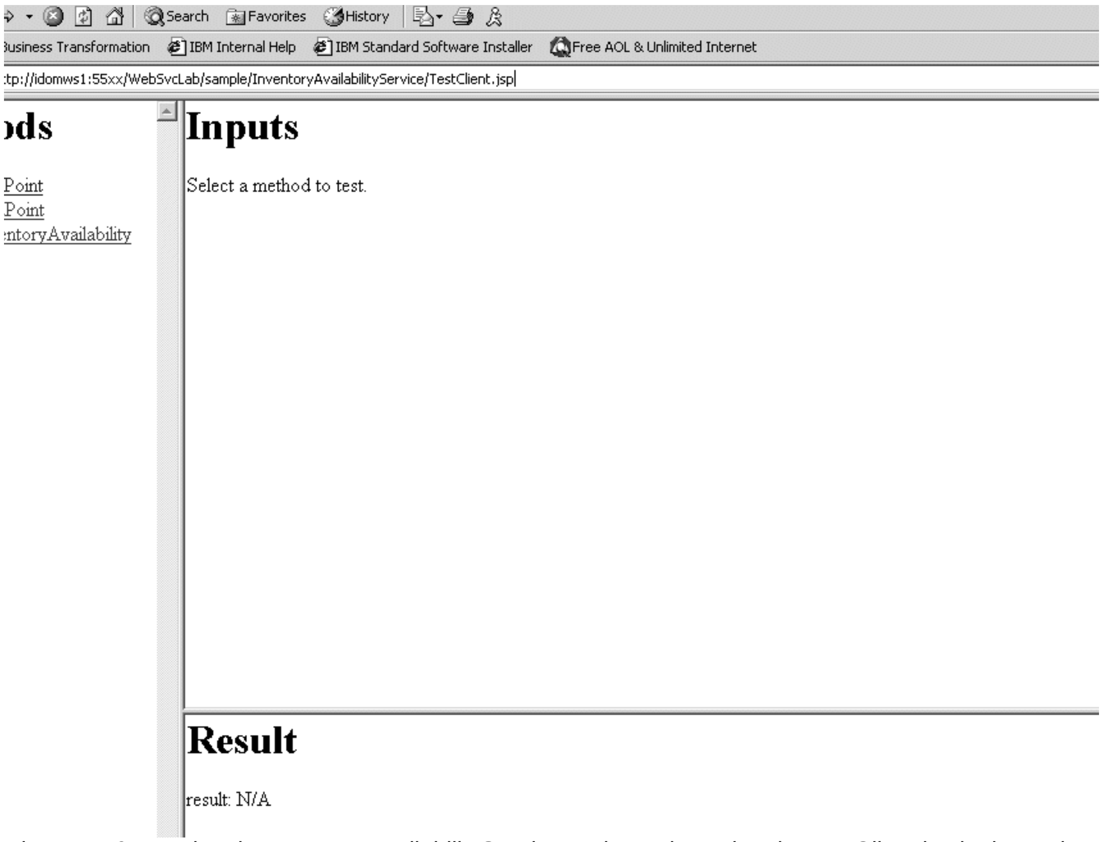
- http://tempuri.org/InventoryAvailabilityService

Figure 4-18 A listing of the active Web services

__ 7. To invoke your Web Service, change the URL in the browser to: **http://AS06:55xx/WebSvcLab/sample/InventoryAvailabilityService/TestClient.jsp**. Replace “AS06” with the host name of your iSeries and xx with your team number for the port number (55xx).

Note: If you get a “Page not found” error, make sure the URL typed in is correct.

__ 8. Press **Enter.** The result is shown in Figure 4-19.



:tp://idomws1:55xx/WebSvcLab/sample/InventoryAvailabilityService/TestClient.jsp

ods

Point
Point
entoryAvailability

Inputs

Select a method to test.

Result

result: N/A

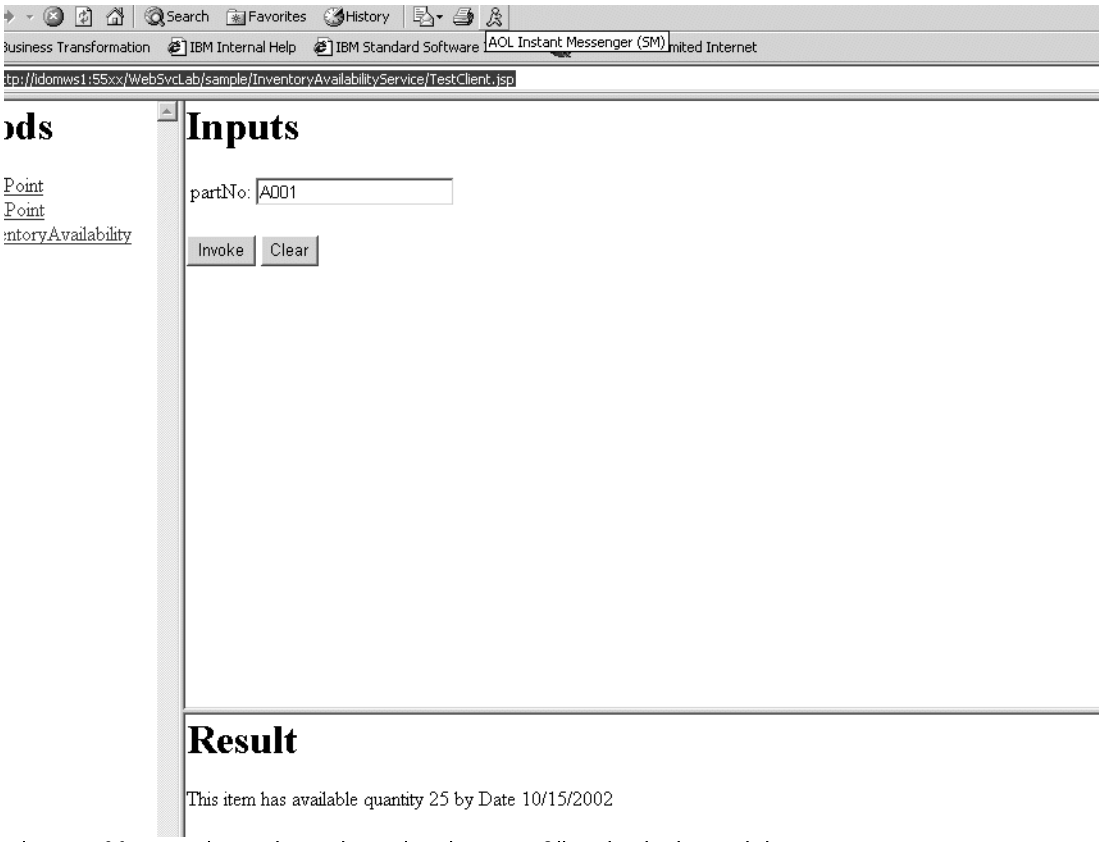
Figure 4-19 Testing the InventoryAvailabilityService Web service using the TestClient.jsp in the Web browser

__ 9. Click the **getInventoryAvailability** method.

__ 10. Enter the value **A001** in the **Inputs parameter**.

__ 11. Press the **Invoke** button.

__ 12. You should receive the results of **The item has available quantity 25 by Date 10/15/2002** as shown in Figure 4-20.



ttp://idomws1:55xx/WebSvcLab/sample/InventoryAvailabilityService/TestClient.jsp

Inputs

partNo: A001

Result

This item has available quantity 25 by Date 10/15/2002

Figure 4-20 Test the Web service using the eTestClient.jsp in the Web browser

Congratulations! You have successfully deployed your Web service in the WebSphere Application Server environment running on the iSeries!

Special notices