H02

IMS Connectivity: IMS Connect and WSAD IE Enterprise Services

Ken Blackman and Suzie Wendler

Las Vegas, NV **September 15 - September 18, 2003**

RETURN TO INDEX

Contents

This workshop features hands-on labs using IMS Connect and the new WebSphere Application Developer Integration Edition (WSAD IE) toolkit. During the exercises, you will have the opportunity to:

- configure and work with your own IMS Connect system
- get first-hand experience on how it interacts with IMS
- and create a Web Service to access an IMS transaction using the WSAD IE tool.

The workshop provides step-by-step instructions and is designed to be self-paced. Instructors are available to assist in any problems encountered.

Notice

References in this publication to IBM products (including programs or services), do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product in this document is not intended to state or imply that only IBM's product may be used. Any functionally equivalent product may be used instead, although the services described as part of this offering may vary based on non-IBM Vendor agreements. Evaluation is the responsibility of the customer.

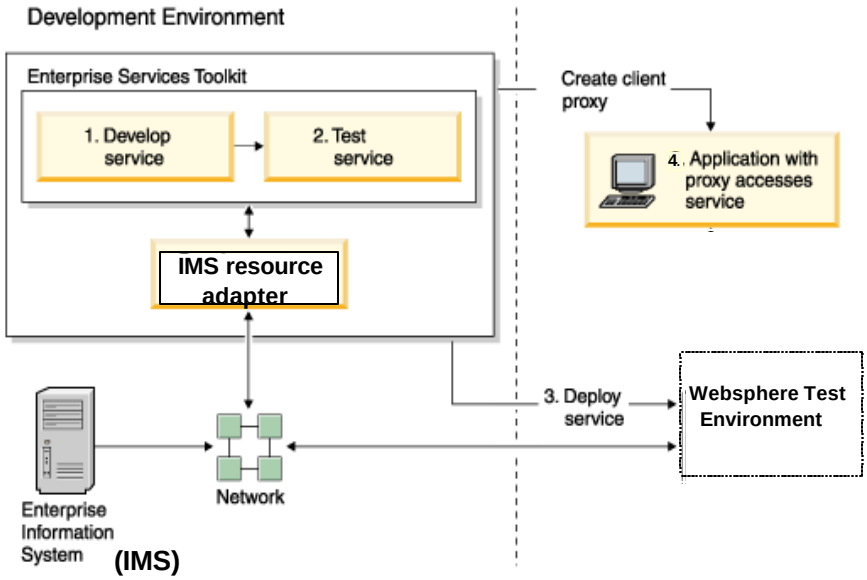
IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

The goal of this document is to provide an overview of the configuration of IMS Connect and to allow IMS customers with the opportunity to work with some of the available products that can assist in web-enabling their IMS systems.

Objectives of this Workshop

This workshop is designed to provide hands-on experience in the configuration of IMS Connect and the creation of a service to drive a transaction in IMS.

A “service”, more specifically, an “enterprise service” is the term used to refer to code that runs on an application server such as Websphere that drives a request to invoke access to an Enterprise System such as IMS. The creation of the “service” follows standard rules. The following diagram shows the process that we will use. It begins by developing the service with the set of tools in WSAD IE. After testing, the service can be moved to the production environment on an application server. In this workshop we will simulate the deployment by using the WebSphere(R) Test Environment in WSAD IE. This part of the tool emulates the actual Websphere EE application server within the development environment. Finally, we will use the tool to create a client proxy and add that proxy to the client application accessing the service.



Exercise 1. Review the Environment.

What This Exercise is About

We will begin by reviewing the workstation environment and then proceed to a familiarization with the host.

Tasks:

1. If your workstation has not already been started, proceed through the following:
   a. Start your system.
   b. When the logon panel appears, use the following **USERID ==> admin, PASSWORD ==> admin**

2. Identify your workstation hostname:
   a. Start a DOS command window.
   b. Type **hostname** on the command prompt. This is your workstation hostname and should be the same name as **teamxx** where **'xx'** is the number assigned to your team.
   c. An alternative way to identify your workstation hostname is:
      - Select (Highlight Only) Network ICON from Control Panel or Desktop.
      - Right Mouse Click, Select Properties. Under the Identification Tab, check Computer Name.

3. Determine your IP address and those of other workstations in the class.

   On a DOS command window prompt, enter **'PING teamxx'** where **'xx'** is your team number. Repeat this for other teams.

4. To logon to the MVS host, the following has been set up for you.

   Userid ==> TEAMxx, Password ==> TEAMxx where **'xx'** is the number assigned to you.

MVS Information

Left mouse click the CM650 icon. This will establish the connection to the MVS system.

Logon information: Userid=TEAMxx Password=TEAMxx

```
                CUSTOMPAC MASTER APPLICATION MENU
OPTION ===>                                          SCROLL ===> PAGE

  IS  ISMF     - Interactive Storage Management Facility
  P   PDF      - ISPF/Program Development Facility
  IP  IPCS     - Interactive Problem Control Facility
  HC  HCD      - Hardware Configuration Definition
  SM  SMP/E    - SMP/E Dialogs
  R   RACF     - Resource Access Control Facility
  SD  SDSF     - System Display and Search Facility
  OS  Support  - OS/390 ISPF System Support Options
  OU  User     - OS/390 ISPF User Options
  DI  DITTO    - Data Interfile Transfer, Testing and Operations
  S   SORT     - DF/SORT Dialogs
  BMR BMR READ - BOOKMANAGER READ (Read Online Documentation)
  BMI BMR INDX - BOOKMANAGER READ (Create Bookshelf Index)
  BMB BMR BLD  - BOOKMANAGER BUILD (Create Online Documentation)
  IC  ICSF     - Integrated Cryptographic Service Facility
```

Use OPTION ‘P’ to access the ISPF/PDF environment

Use OPTION ‘SD’ to access the Console

Exercise 2. Configure IMS Connect

What This Exercise is About

In this exercise you will logon to the MVS system and work with IMS Connect. The activities will involve configuration, set-up and starting an IMS Connect address space and ensuring that it can communicate with IMS.

Tasks:

1. Logon to MVS. Your USERID ==> TEAMxx, PASSWORD ==> TEAMxx.

2. Access “TEAMxx.WORKSHOP.JCL” and edit member BPECFGXX. Make sure the profile is “NUMS OFF”. The TSO command is “unnum”. Check that columns 72-80 are blank.

3. Create a new member BPECFGxx where xx is your team number. This will be used as your own BPE configuration member for tracing. For purposes of this workshop, delete all TRCLEV statements. We will not be requesting any internal traces.

4. Copy your BPECFGxx member to IMS710.PROCLIB.

5. Browse IMS710.PROCLIB(DFSPBIM7).

   a. Identify GRNAME, APPLID1 and IMSID values.

   b. Note all three values because you will need them for the IMS Connect Configuration. Recommended DATASTORE ID is the value in IMSID, though not required (easier to monitor which IMS the Client is communicating with).

6. Edit TEAMXX.WORKSHOP.JCL(ICONNXX)

   a. Change XX to your team value.

   b. Leave Security OFF.

```
************************************************************
* IMS CONNECT CONFIGURATION FILE
************************************************************
HWS (ID=ICONNXX)
TCPIP (HOSTNAME=TCPIP,PORTID=(34XX),EXIT=(HWSIMSO0,HWSSMPL0))
```

c. Add a DATASTORE statement.

DATASTORE statement	FUNCTION	DFSPBxx member
GROUP	XCF group name	GRNAME
ID	Name that client specifies for an IMS	Any value, e.g., IMSID
TMEMBER	XCF name for the target IMS	APPLID1, or USERVAR, or OTMANM
MEMBER	XCF name for this IMS Connect	------

```
DATASTORE (ID=IM7P, MEMBER=ICONNxx,GROUP=IM7POTMA,TMEMBER=IM7P)
```

d. Copy File to IMS710.PROCLIB as ICONNxx where xx is your team number. Again make sure that columns 72-80 are blank.

7. Edit TEAMXX.WORKSHOP.JCL(ICONNJXX)

a. Change XX to your team number.

b. Submit the job.

```
//ICONNXX  JOB CLASS=A,MSGCLASS=O,REGION=4M
// USER=TEAMXX,NOTIFY=TEAMXX
//*********************************************
//*  BRING UP AN IMS CONNECT SYSTEM
//*********************************************
//STEP1    EXEC PGM=HWSHWS00,TIME=1440,
//             PARM='BPECFG=BPECFGXX,HWSCFG=ICONNXX'
//STEPLIB  DD  DSN=IMSCON.SHWSRESL,DISP=SHR
//PROCLIB  DD  DSN=IMS710.PROCLIB,DISP=SHR
//SYSPRINT DD  SYSOUT=O
//SYSUDUMP DD  SYSOUT=O
//HWSSVSTC DD  DSN=TEAMXX.ICONN.TRACE,DISP=SHR
//HWSRCORD DD  DSN=TEAMXX.ICONN.HWSRCORD,DISP=SHR
```

8. View the status of your IMS Connect (ICONNxx) job.

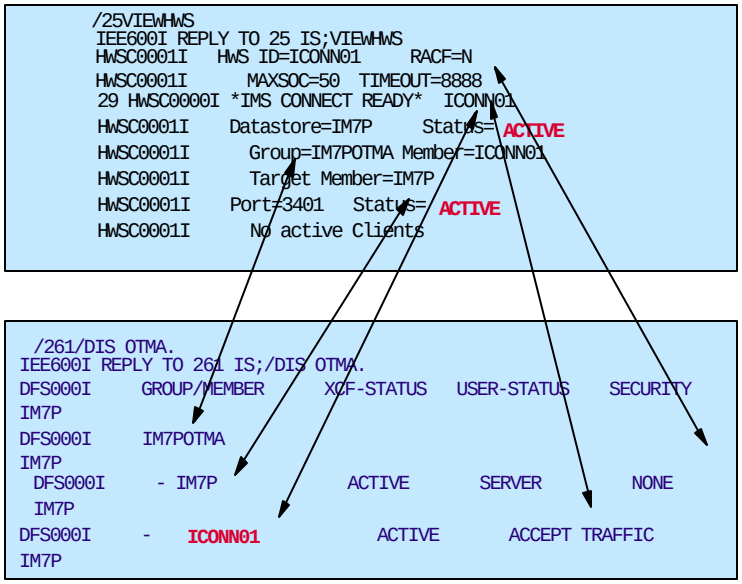
   From the SDSF - LOG panel, enter '**/rr**VIEWHWS' where '**rr'** is the outstanding prompt for your ICONNxx job. Note that the commands on this console must always start with a '/'. This is a requirement for this system and does not imply an IMS command.

```
/25VIEWHWS
IEE600I REPLY TO 25 IS;VIEWHWS
HWSC0001I   HWS ID=ICONN01     RACF=N
HWSC0001I      MAXSOC=50  TIMEOUT=8888
29 HWSC0000I *IMS CONNECT READY*  ICONN01
HWSC0001I    Datastore=IM7P     Status=ACTIVE
HWSC0001I      Group=IM7POTMA Member=ICONN01
HWSC0001I      Target Member=SCSIM7P
HWSC0001I    Port=3401   Status=ACTIVE
HWSC0001I      No active Clients
```

9. View the status of your connection to IMS from the IMS side. Enter '/**rr**/DIS OTMA' where '**rr'** is the outstanding prompt for IMS. Note also that there is a '/' after **rr** since this is an IMS command.

```
/261/DIS OTMA.
IEE600I REPLY TO 261 IS;/DIS OTMA.
DFS000I     GROUP/MEMBER      XCF-STATUS   USER-STATUS    SECURITY
IM7P
DFS000I     IM7POTMA
IM7P
DFS000I     -IM7P              ACTIVE       SERVER         FULL
IM7P
DFS000I     -ICONN01           ACTIVE       ACCEPT TRAFFIC
IM7P
```

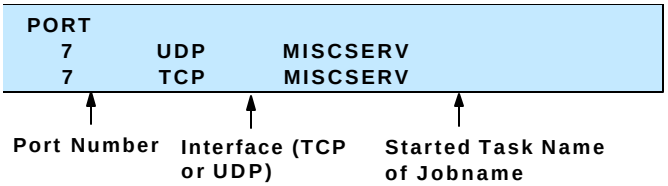
10. Review IMS Connect and OTMA Relationships.



IMS Connect Display	OTMA Display
Datastore=IM7P	IM7P
RACF=N RACF-Y	SECURITY=NONE SECURITY=Profile, CHECK or FULL
Status=ACTVE	USER-STATUS Accept Traffic
MEMBER=ICONN01	GROUP/MEMBER ICONN01
Target Member = SCSIM7P	GROUP/MEMBER SCSIM7P
IM7POTMA	GROUP/MEMBER IM7POTMA

11. Things to review and try.

a. In your own environment, you might want to consider reserving the TCP/IP port that you specified in the TCPIP statement of the configuration file (HWSCFG=). This would be done by the TCP/IP administrator. You can check this by looking in the 'TCPIP.PROFILE.TCPIP' dataset. This is the default TCP/IP data set name and may be different in your particular environment. For this class, the dataset is 'TCPIP.TCPIP.TCPPARMS(TCPPROF)'. Browse this data set (do not edit it). Issue a 'F PORT 1' command. You will notice that this will bring you to the list of reserved ports.



To add your IMS Connect ports to this environment, the TCP/IP administrator would add:

(note - do not do this for this environment)

```
3401 TCP ICONN1          ; IMS Connect Port 1
3402 TCP ICONN2          ; IMS Connect Port 2
...                        ...
```

b. Review and test the following commands:

The format is: **'/rr**COMMAND**'** where **rr** is the outstanding prompt of your IMS Connect address space, e.g., /**45**VIEWHWS.

IMS Connect command	**Description**
CLOSEHWS	Terminates IMS Connect
OPENDS *datastore id*	Starts communication with an IMS
OPENPORT *port id*	Reestablishes communications with TCP/IP
SETRACF *ON* | *OFF*	Defines the security environment
STOPCLNT *port id client id*	Terminates communications with a specific client on a port
STOPDS *datastore id*	Terminates communications with an IMS
STOPPORT *port id*	Terminates listening on a specific TCP/IP port
VIEWDS *datastore id*	Displays current activity with an IMS
VIEWHWS	Displays current activity of IMS Connect from both the port and datastore perspectives
VIEWPORT *port id*	Displays the current activity of a port

Exercise 3. Bring up WSAD IE.

What This Exercise is About

The purpose of this exercise is to provide a familiarization with the WSAD IE (Websphere Studio Application Developer - Integration Edition) toolkit.

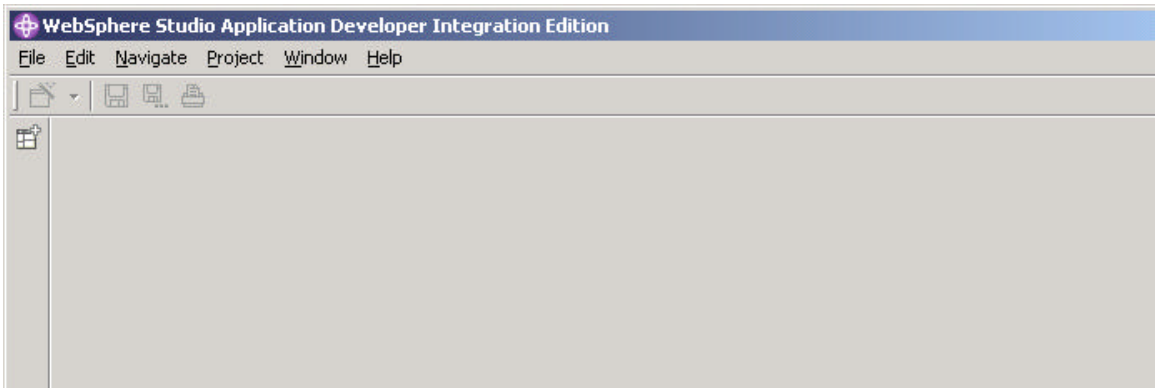
Application Developer Integration Edition consists of the following tools and components:

- **A set of tools to develop a service, collectively called the Enterprise Services Toolkit. These tools include a subset of tools to let you work with multiple services, creating another service out of the many. It is called a service flow. A service flow connects the output of one service to the input of another.**
- **A set of resource adapters. Resource adapters let services access Enterprise Information Systems (EIS). Examples of EIS systems are IMS and CICS.**

WSAD IE is a general-purpose application development environment and also includes the capabilities provided by its predecessor toolkits. For example, the wizards to create EJBs, JSPs, servlets, XML-based documents, Web pages, and so forth. Additionally, WSAD IE has access to the operating system resources such as the file system, databases, and application servers. The toolkit, therefore, is one of the most complete development environments for those designing services for enterprise applications.

1. At the bottom left of your workstation screen, select **Start ==> Programs ==> IBM Websphere Studio Application Developer Integration Edition ==> IBM Websphere Studio Application Developer Integration Edition 5.0.** This brings up the toolkit

2. Once the tool has been initialized, your workspace is created and presented for your use.



You are now ready to begin creating an IMS Service.

(This page intentionally left blank)

Exercise 4. Create an enterprise service for an MFS-based IMS Transaction.

There are a few basic steps that need to be done to successfully create an IMS Service. These include:

2a. Create a service project
2b. Import an MFS source file
2c. Generate the enterprise service
2d. Test the generated enterprise service
2e. Generate the deploy code (EJB session bean) for the enterprise service
2f . Bind the resource reference (allows the generation of a connection during runtime)
2g. Configure the server and deploy the project

2a. Create a service project

****
Some terminology - “project” and “perspective”.

A project is a mechanism for organizing resources. It is composed of folders (directories) and files. There are different kinds of projects in the WSAD IE tool. For example: Java projects are for stand-alone applications; Web projects for Web applications; EJB (Enterprise Java Bean) projects for EJB development; EAR projects tie together Web and EJB (and Client) projects into a J2EE hierarchy; Server projects are used to define application servers for testing.

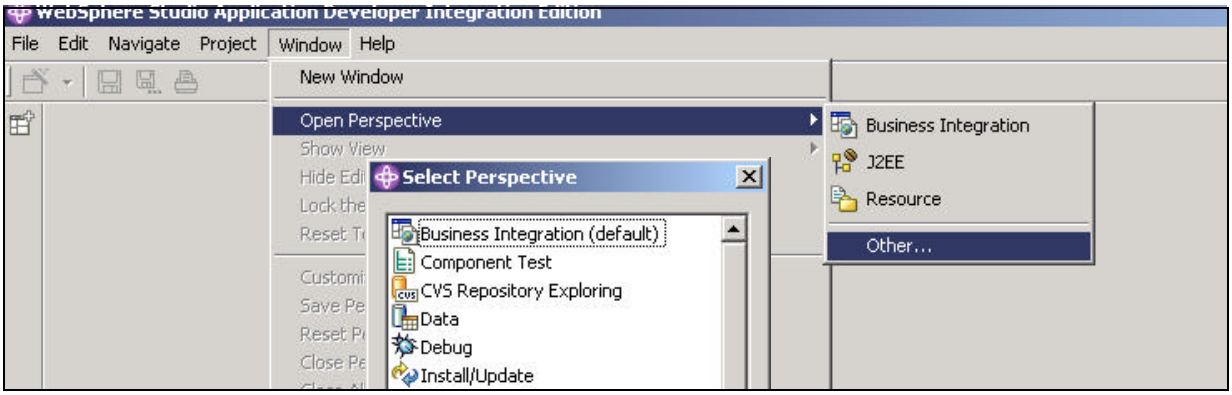
A perspective is the way a developer sees the projects. A perspective is a set of views, arranged into the Workbench window, and a set of editors and tools that are used to manipulate the resources.Perspectives are tailored for certain tasks, based on the role of the developer. For example, in the Java perspective you can compile Java code; in the Web perspective, you can edit and customize Web applications; in the J2EE perspective, you develop J2EEhierarchies and EJBs.

As we go through the remainder of the workshop, you will be told which type of project to create and which perspective to open.
****

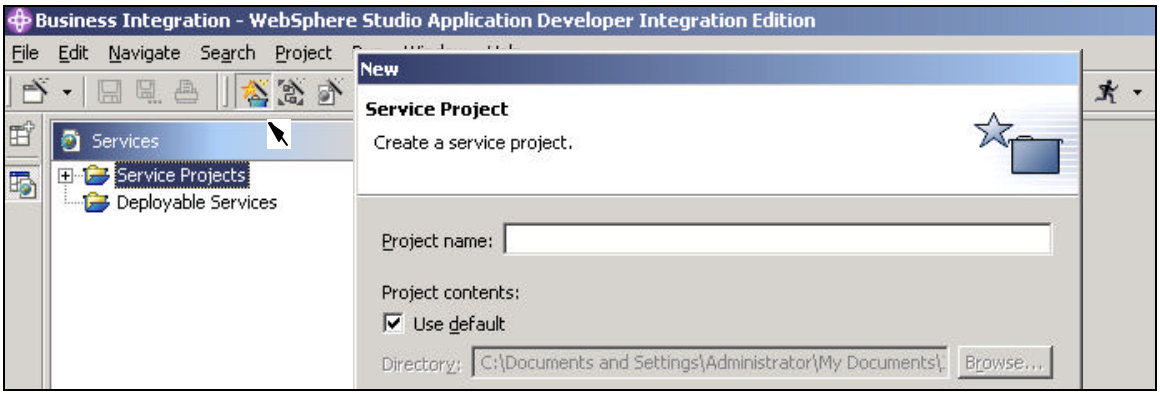
In this exercise, we will create an enterprise service that will be contained in a Java project. All the resources required by this service, e.g., the MFS source and service settings will be kept in this project.

To create a project for the service, follow these steps:

1. Open the Business Integration perspective. Go to the tool bar at the top of the screen and click on **Window > Open Perspective > Other.** Select **Business Integration** and click **OK**.



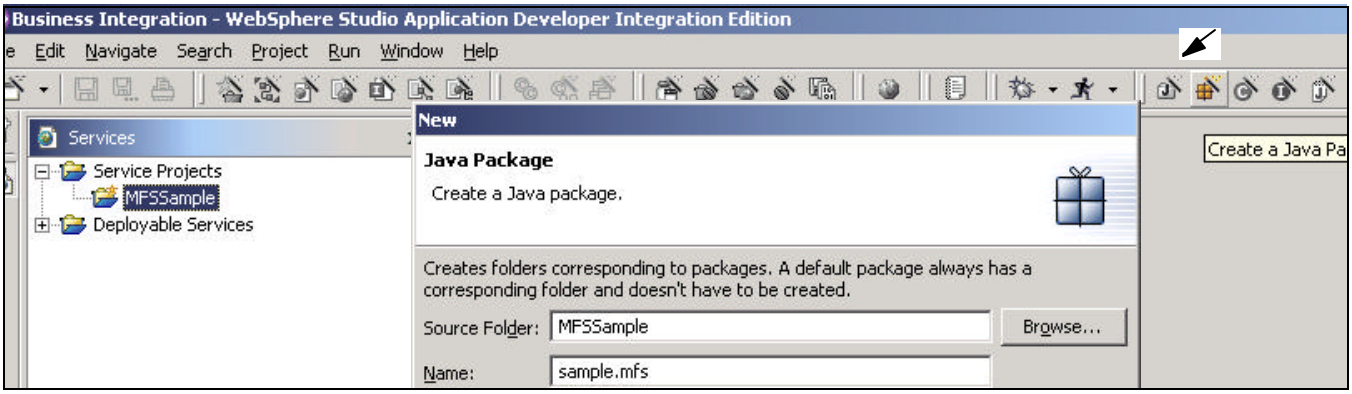
2. From the toolbar, click **Create a service project** icon. This opens the Service project wizard.



3. Type **MFSSample** for the name of the project. Use the default location to store the new project. Click **Finish** to create the project.

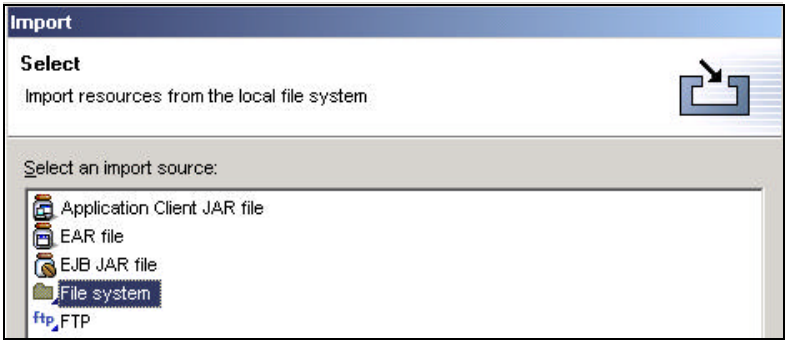
2b. Import the MFS File

1. Create a **Java package** by expanding the **Service Projects** folder. Select **MFSSample** and click on the **Create a Java package** icon from the toolbar. On the Java Package page, make sure that **MFSSample** is displayed as the name of the folder. **(A package is a component in a project that contains a group of files.)**



2. Type **sample.mfs** for the package name. Click Finish.

3. Go back to the Services perspective window. Expand the **MFSSample** project and select the **sample.mfs** package that you just created. Go to the top of the screen and click **File** in the toolbar. In the pop-up window, select **Import** to open up the Import wizard.



At this point we are now ready to work with the MFS definitions from the IMS transaction. A sample MFS file for the IMS IVP transaction - IVTNO - is shipped with the WSAD IE toolkit. What we need out of this file is the structure of the input and output messages that the IMS transaction needs. We will begin by importing the file into our WSAD IE workspace. The next set of steps show you how to access the file.

4. Select **File system** to import the resources from the local file system and then click **Next.**

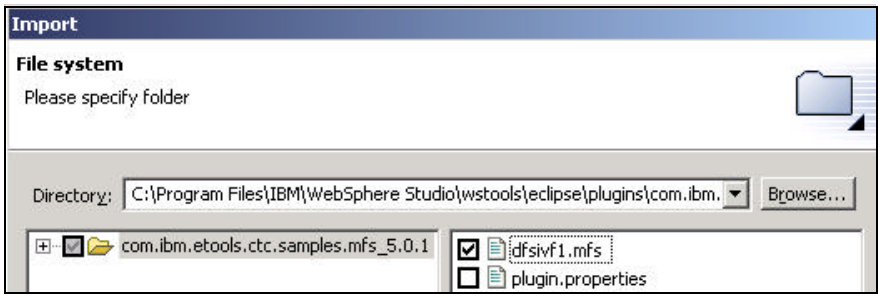
5. To access the directory, click **Browse** and select the following folder:

   C:\Program Files\IBM\WebSphere Studio\wstools\eclipse\plugins\com.ibm.etools.ctc.samples.mfs_5.0.1

   Click **OK.**

6. On the **File System** page of the Import wizard, highlight the **com.ibm.etools.ctc.samples.mfs_5.0.1** folder **and make sure the check box is cleared.**

   .



Select the **dfsivf1.mfs** check box to import the MFS file (this will cause a check to be placed in the left pane on the com.ibm.etools.ctc.sample.mfs_5.0.1 box). Make sure that **MFSSample/sample/mfs** is the name of the destination folder for the imported resource. Click **Finish** to import the file and close the wizard. If you are successful in importing the file, the **Tasks** view (bottom panel of your screen) will not contain any (red) errors and the sample.mfs package in the Packages view will contain **dfsivf1.mfs**.

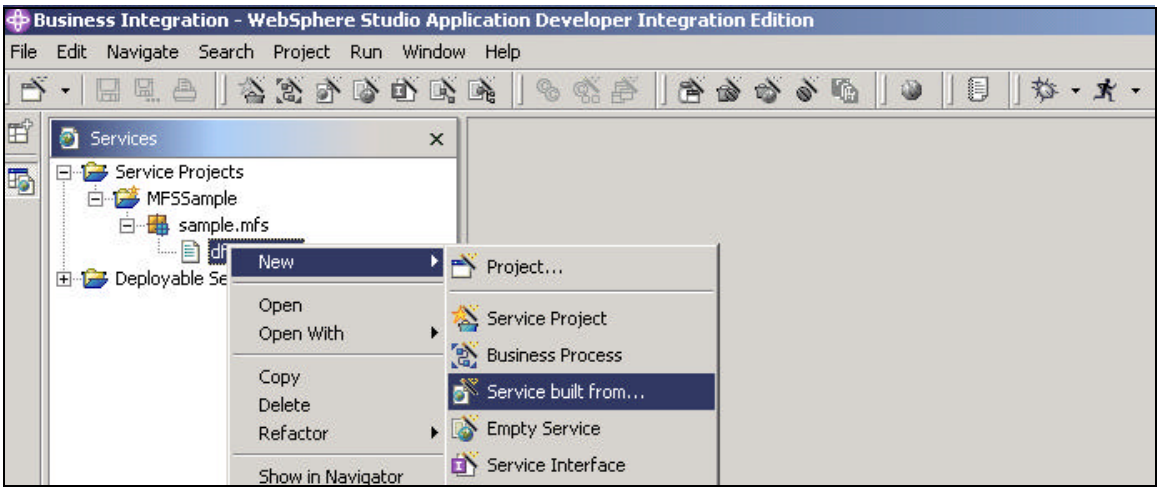
You are now ready to create an enterprise service definition from that file.

2c. Generate the enterprise service definition

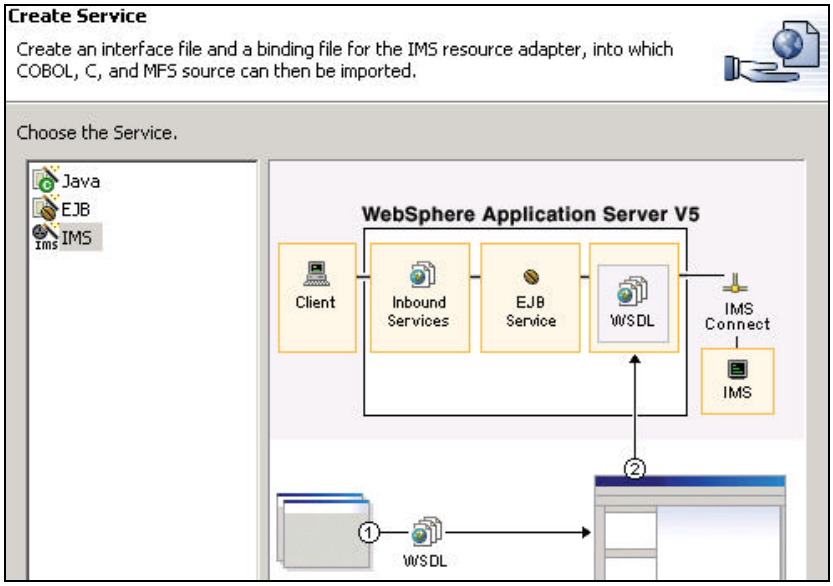
A service definition describes the basic format of system requests - where the service is deployed and what operations this service provides. It is described in the Web Services Description Language (WSDL) which is a standard for describing networked, XML-based services. WSDL provides a simple way to describe the basic format of system requests regardless of the underlying run-time implementation.

In this workshop, you will create an IMS enterprise service definition.

1. Expand **Service Projects > MFSSample > sample.mfs.**
2. Right click on dfsivf1.mfs and in the window that pops up select **New > Service built from...**



3. In the Create Service page, select **IMS** and click **Next**. (Note: IMS should appear on the list, if it does not then you will need to first import the IMS resource adapter, ims.rar, before continuing. You would do this by clicking on the button “import resource adapter” located below the pane listing available services. This will bring up a “Connector Import” page. Browse for the connector RAR file in C:\Program Files\IBM\WebSphere Studio\resource adapters\ims.rar and key in a project name, e.g., IMS.)

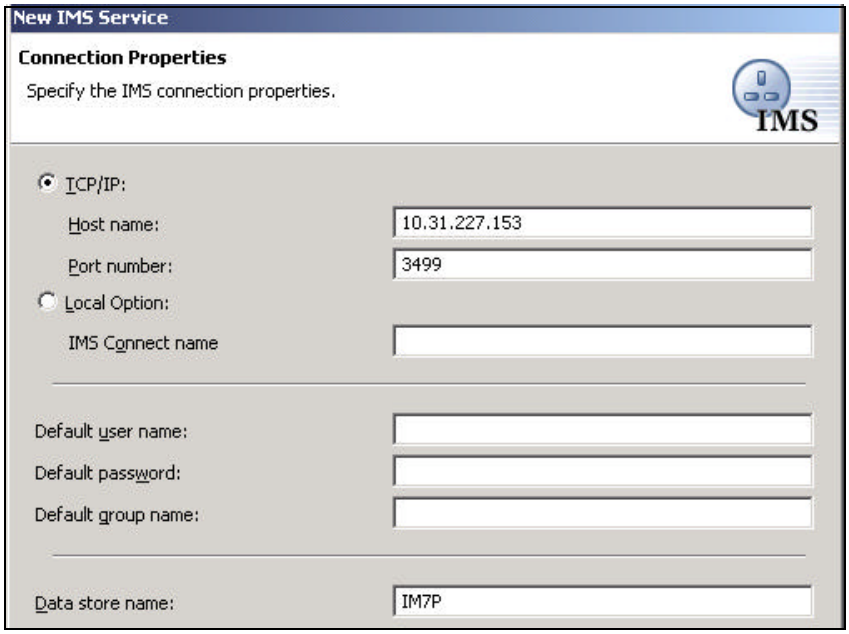


4. In the Connection Properties page which follows, type the values appropriate for your environment.

   a. Type **10.31.227.153** for the Host name.

   b. Type the Port number for your IMS Connect system, e.g/., **34**xx where xx is your team number.

   c. Type **IM7P** for the Data store name. Position the cursor on any empty field - this is to ensure that all the values you have keyed in are actually accepted.

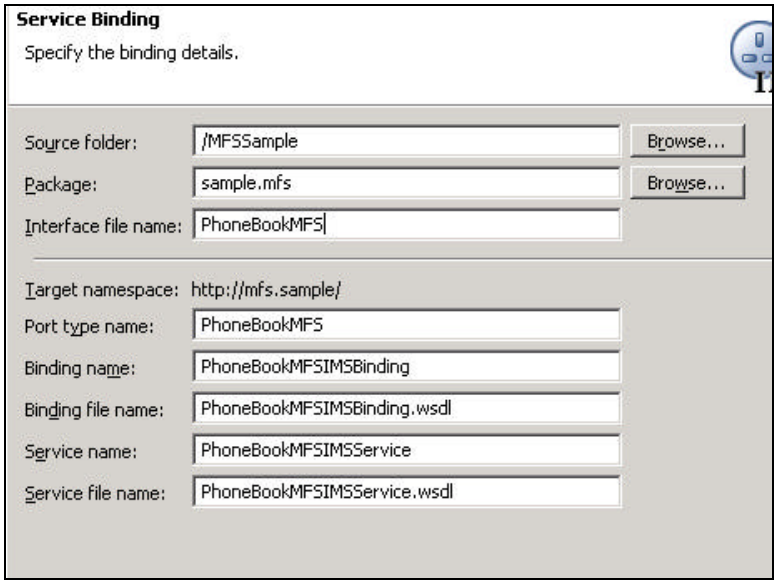


5. Click **Next.**

6. In the Service Binding Properties page, ensure that the following are set:

   – The **Source folder** field is set to **/MFSSample**
   – The **Package** field contains **sample.mfs**.
   – The **Target namespace** is **http://mfs.sample**.

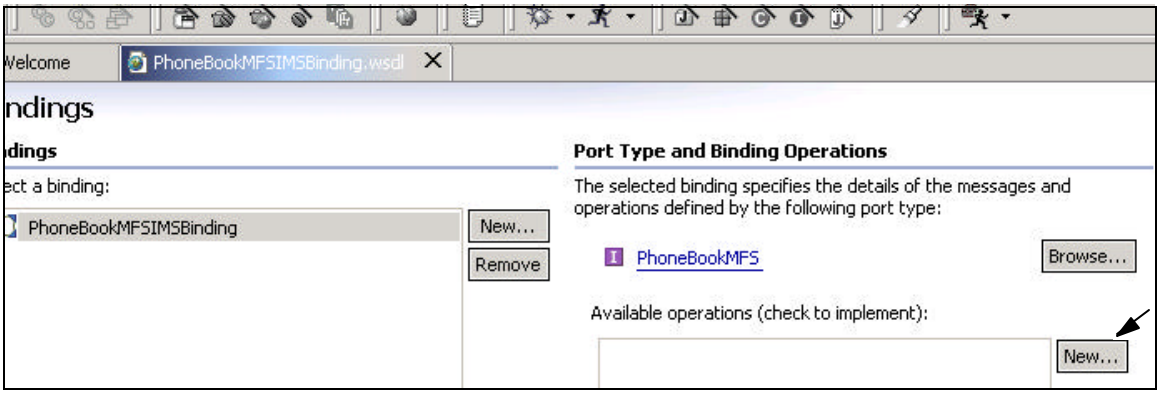
7. In the **Interface file name** field, type **PhoneBookMFS.** This file contains the interface the service uses to send input and get output from the IMS transaction. In this sample, the service gets a request from the client to run the application, sends the request tothe IMS PhoneBook transaction, and returns the response to the client. When you type the interface file name, the wizard automatically enters values for the remaining fields in the Service Binding page.



8. Click **Finish** to accept all other default names. The wizard generates three WSDL files for the service you are creating:

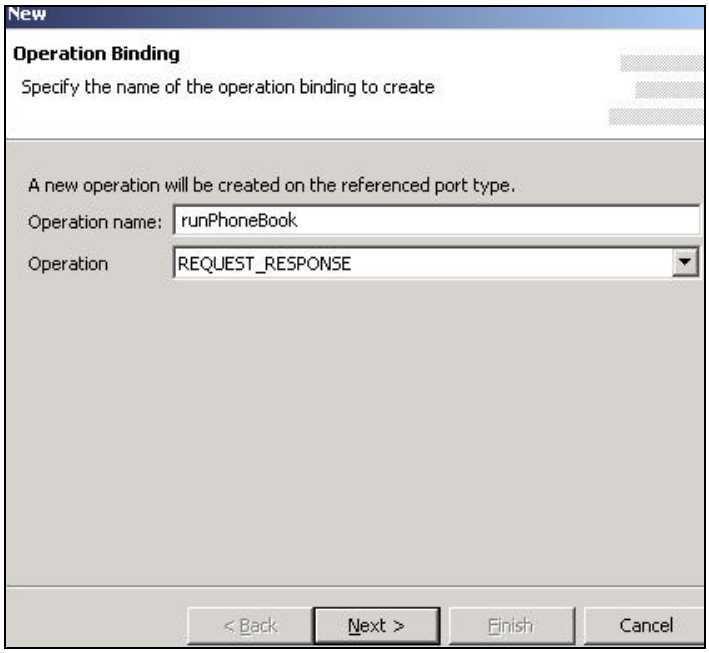
   – The interface file, PhoneBookMFS.wsdl which contains the port types and message elements.
   – The binding file, PhoneBookMFSIMSBinding.wsdl which stores the operation and binding information.
   – The service file, PhoneBookMFSIMSService.wsdl which stores the host information.

   A WSDL editor opens with the Overview of PhoneBookMFSIMSBinding.wsdl. You will now add operations to the service. Make sure you click on the tab called “Bindings”.



9. In the Bindings page under **Port Type and Binding Operations,** click **New...**

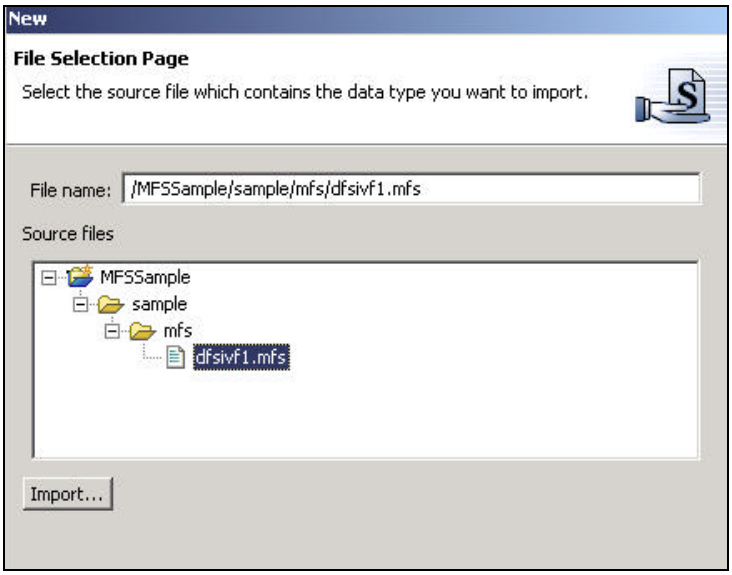
10. In the **Operation Binding** page, specify a new operation. In the **Operation name** field, key in **runPhoneBook.** Leave the type of operation as REQUEST_RESPONSE because there will be two messages, one for the request to run the IMS transaction and one for the reply from the IMS transaction. Click **Next.**



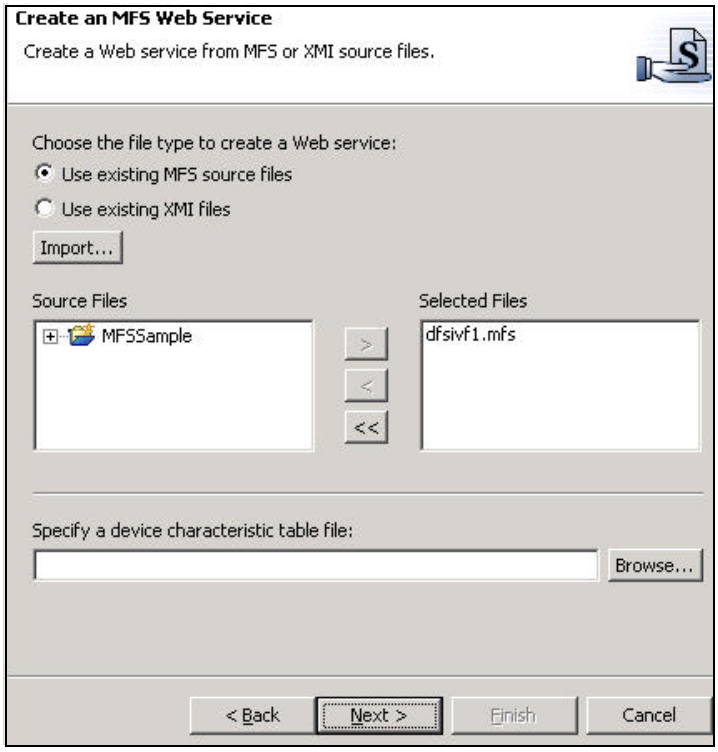
11. i In the **IMS Operation Binding Properties** page, the properties of the interaction with the IMS application program are shown. For this exercise it will not be necessary to change any of the default values. Click **next.**

12. The **Operation Binding** page comes up next. This is where you will create new input and output messages. Click **Import** next to the input message. The importer Wizard opens. You will use this wizard to import the dfsivf1.mfs file and populate the XML schema definition in the interface WSDL file:

    a. Expand **MFSSample > sample > mfs** and select **dfsivf1.mfs.** Click **Next.** The MFS Importer wizard opens.



    b. In the **Create an MFS Web Service** page, you can choose to create the Web service based on existing MFS source files or XMI files. For this sample, select **Use existing MFS source files** and ensure that **dfsivf1.mfs** appears in the Selected Files box. You do not need to specify a device characteristic table file. This field is optional. Click **Next.**



c. The **Parse MFS Files** page comes up next. This is where you can specify a host code page and indicate the encoding (binary or text) of the MFS files downloaded from the host. This is used to parse the selected MFS source files. When Binary is selected, the host code page is used during parsing. When Text is selected, you must identify a text code page to use. Ensure that the Host code page is **Cp037** and that **Text** is selected with **Cp1252** (the text code page for Windows Latin 1) as the Text code page. Click **Parse** to parse the MFS source file. Ensure that the parsing succeeded in the Parse Log:

   IXFWZ10I: Parse succedded

   IXFWZ15I: Device characteristic table file selected: None

   "C:...../MFSSample/sample/mfs/dfsivf1.mfs" parsed successfully

d. Click **Next.**

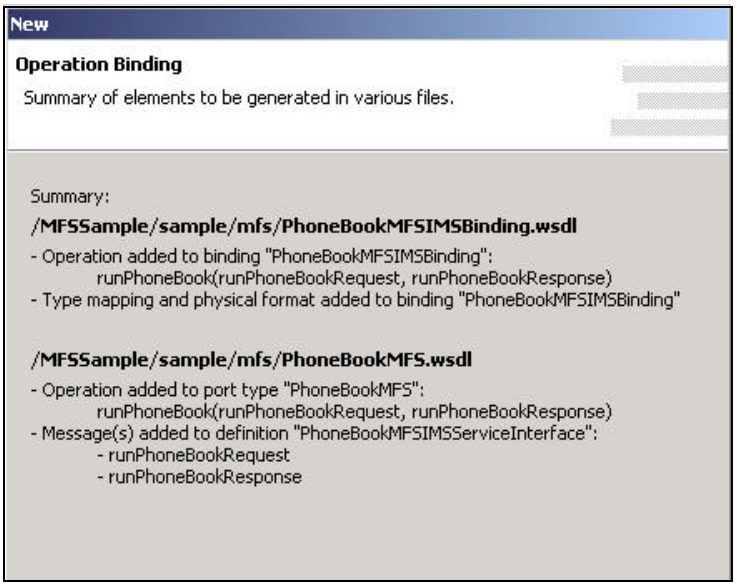
e. In the **Device Type** page, you can select different combinations of device types and device features specified in the MFS source. The device type **3270-A02** and device feature **Ignore** are automatically selected by default. Click **Next.**

f. In the **Mid and Mod** selection page, select the MID and MOD names to use as the input and output types. The Message Input Descriptor (MID) defines how data should be formatted for presentation to the IMS application program. The Message Output Descriptor (MOD) defines content for the output message and, optionally, literal data to be considered part of the output message. The system default MODs (DFSMO1, DFSMO2, DFSMO3, and DFSMO5) are selected by default. Leave them selected and use the arrow keys to select **IVTNOMI1** for the MID name (left mouse click to highlight and press **>** to move it to the selected column) and **IVTNO** for the MOD name. (Accept the default, 1, for Logical page number).

g. Ensure that **Overwrite existing XSD types** is selected. Click **Finish.** The wizard generates XMI (XML Metadata Interchange) files for the selected MID and MOD and populates WSDL files with the input and output information for the new operation.

h. The **Operation Binding** page comes up next with defaults of **runPhoneBookRequest** as the input message and **runPhoneBookResponse** as the output message. Click **Next.**

i. In the **Operation Binding** summary page, the new operation information is displayed. Click **Finish**. The wizard populates WSDL files with operation information.



Now you are ready to create a proxy to test the service that you just created.

2d. Test the generated enterprise service

It is now time to test the generated service you created previously. In Step 1, you will build a Java service proxy to access the service and in Step 2, you will test it.

****

A proxy provides a remote procedure call interface to the service. Using a proxy, the application can call a remote method on the service as if the method were a local one. Once the application makes the remote callk the proxy handles all of the communication details between the application and the service.

****

1. Create the Java Service proxy.
   a. Expand the **MFSSample** project and the **sample.mfs** package. Select the service file **PhoneBookMFSIMSService.wsdl**.
   b. Right click the file and select **Enterprise Services > Generate Service Proxy.** The Service Proxy wizard opens.
   c. In the **Service Proxy** page, ensure that the service you want to create the proxy for is shown. Because you selected a file in the first step, most fields are populated with default values. These default values are generated based on the contents of the selected service file.

      - Ensure that the class name for the proxy is **PhoneBookMFSProxy.java**.

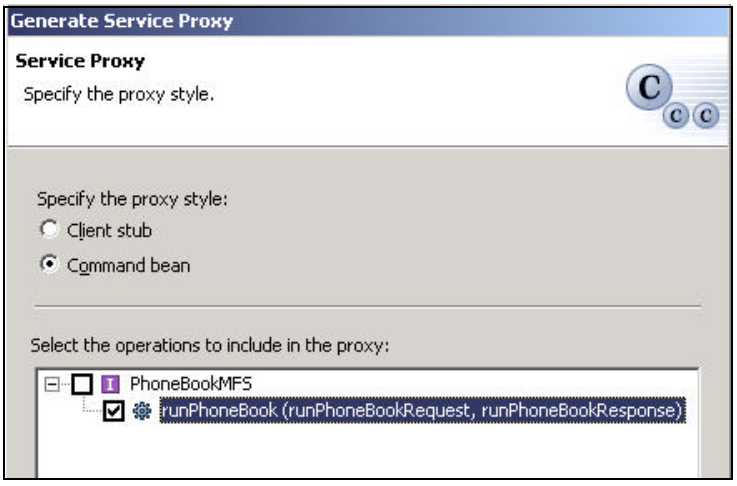
      - Ensure that the package name is **sample.mfs**.

      - Ensure that **Generate helper classes** is selected. These Java helper classes are required by your service.

      Click **Next**.
   d. In the **Service Proxy** page, specify the style of the proxy and the operations to expose in the proxy:

      - Select the **Command bean** proxy style.

      - Select the **runPhoneBook** check box to select the operations to include in the proxy.

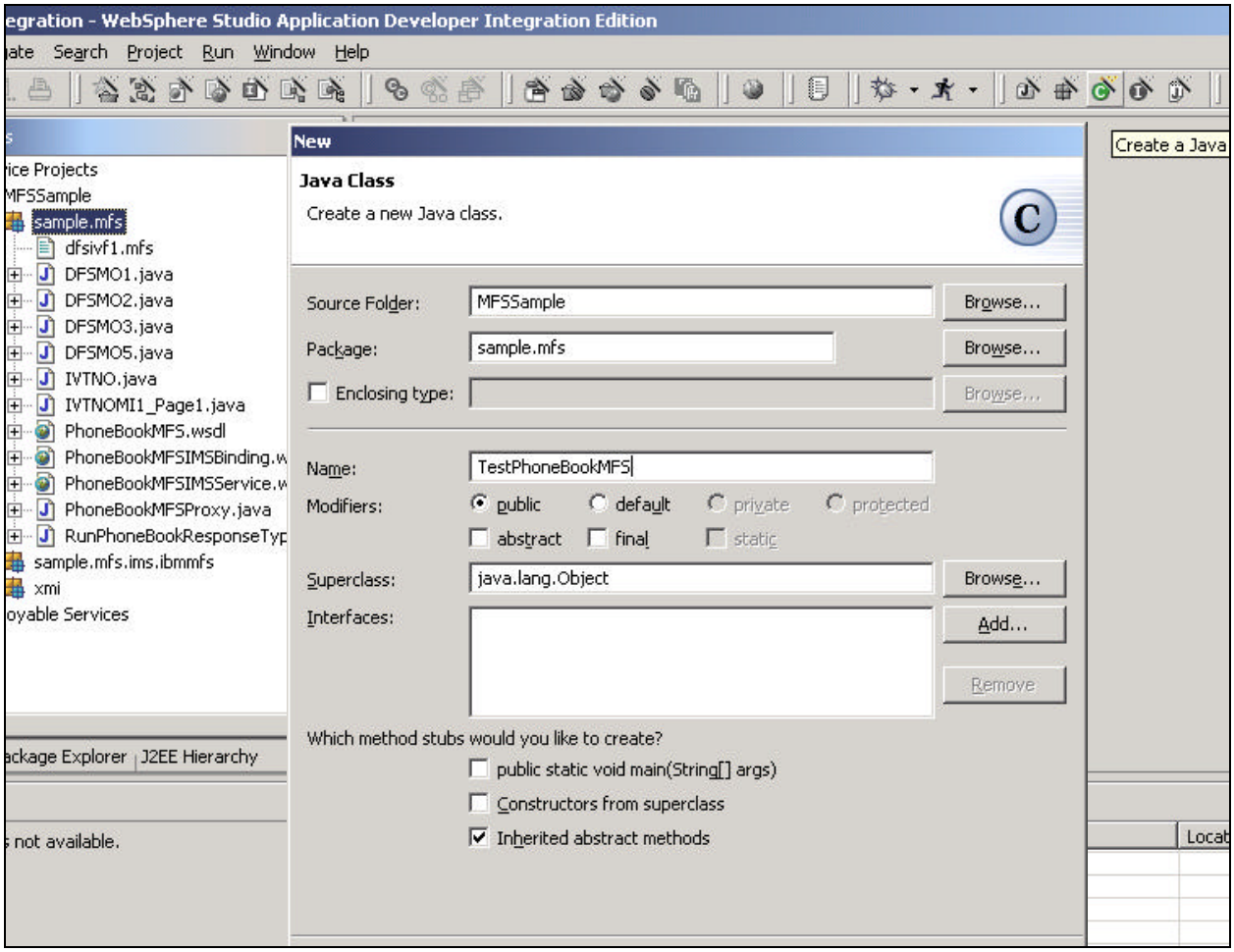


   e. Click **Finish**. The Java service proxy (PhoneBookMFSProxy) is generated in the MFSSample project.

2. To test the proxy that you just created, you will need to write client code that can execute it. The code will need to set parameters for the input message, invoke the proxy, pass the input message to the proxy, receive an output message back from the proxy, and then display the message on the console. You will use the service project you created named MFSSample to store your code.

   a. Expand the **MFSSample** project and then select the **sample.mfs** package. From the toolbar click the **Create a Java class** icon .

   b. Ensure that **MFSSample** is the source folder and that **sample.mfs** is the package name.



   c. Type **TestPhoneBookMFS** for the name of the new class.

   d. Accept all other defaults and click **Finish.**

   e. Replace the code in the editor with the following code (also found in softcopy in C:\class\svc501.txt):

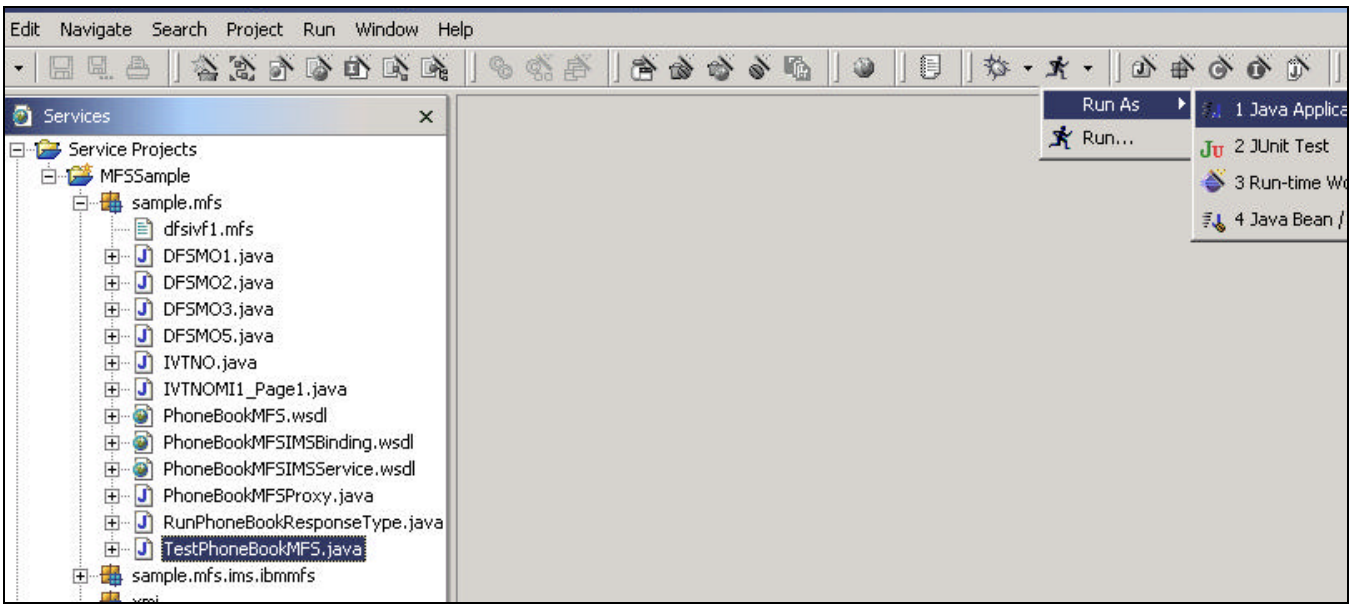
```
package sample.mfs;
public class TestPhoneBookMFS {
public static void main(String[] args)
{ try
{PhoneBookMFSProxy aProxy = new PhoneBookMFSProxy();
IVTNOMI1_Page1 inputMessage = new IVTNOMI1_Page1();
```

```
aProxy.setIVTNOMI1_Page1(inputMessage);
inputMessage.setCMD("DISPLAY");
inputMessage.setNAME1("LAST1");
aProxy.execute();
RunPhoneBookResponseType outputMessage = aProxy.getRunPhoneBookResponseTypePart();
if (outputMessage.getIVTNO() != null) {
         System.out.println("IVTNO");
         IVTNO outputPart = outputMessage.getIVTNO();
         IVTNO.IVTNO_PagesLocal[] outputPages = outputPart.getIVTNO_Pages();
         System.out.println("MFS Phonebook Java Proxy");
         System.out.println();
         IVTNO.IVTNO_PagesLocal.IVTNO_Page1Local outputPage1 = out
                  putPages[0].getIVTNO_Page1();
         System.out.println(outputPage1.getNAME2() + outputPage1.getNAME1());
         System.out.println(outputPage1.getEXTUnicode0023());
         System.out.println(outputPage1.getZIP());
         System.out.println(outputPage1.getMSG().trim() + " - " +
                  outputPage1.getSDATE());
} if (outputMessage.getDFSMO1() != null) {
         System.out.println("DFSMO1");
         DFSMO1 outputPart = outputMessage.getDFSMO1();
         DFSMO1.DFSMO1_PagesLocal[] outputPages = outputPart.getDFSMO1_Pages();
         System.out.println("System error message:");
         System.out.println();
         DFSMO1.DFSMO1_PagesLocal.DFSMO1_Page1Local outputPage1 =
                  outputPages[0].getDFSMO1_Page1();
         System.out.println(outputPage1.getOUTLINE());
} if (outputMessage.getDFSMO2() != null) {
        System.out.println("DFSMO2");
        DFSMO2 outputPart = outputMessage.getDFSMO2();
        DFSMO2.DFSMO2_PagesLocal[] outputPages = outputPart.getDFSMO2_Pages();
        System.out.println("System error message:");
        System.out.println();
        DFSMO2.DFSMO2_PagesLocal.DFSMO2_Page1Local outputPage1 =
                  outputPages[0].getDFSMO2_Page1();
        System.out.println(outputPage1.getPP1_OUTL01());
} if (outputMessage.getDFSMO3() != null) {
        System.out.println("DFSMO3");
        DFSMO3 outputPart = outputMessage.getDFSMO3();
        DFSMO3.DFSMO3_PagesLocal[] outputPages = outputPart.getDFSMO3_Pages();
        System.out.println("System error message:");
```

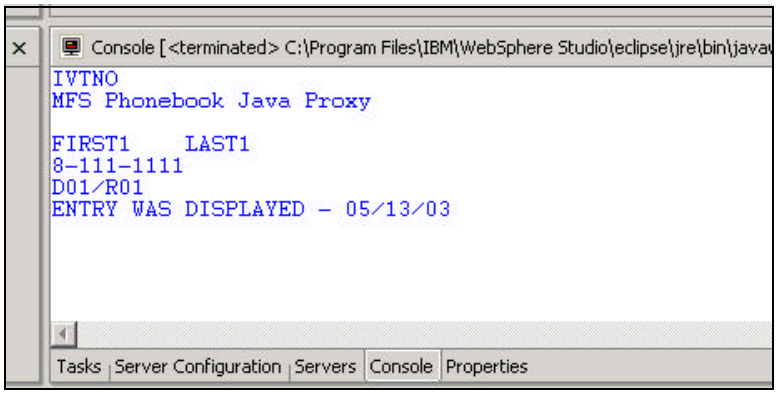
```
            System.out.println();
            DFSMO3.DFSMO3_PagesLocal.DFSMO3_Page1Local outputPage1 =
              outputPages[0].getDFSMO3_Page1();
            System.out.println(outputPage1.getOUTL01());
        } if (outputMessage.getDFSMO5() != null) {
            System.out.println("DFSMO5");
            DFSMO5 outputPart = outputMessage.getDFSMO5();
            DFSMO5.DFSMO5_PagesLocal[] outputPages = outputPart.getDFSMO5_Pages();
            System.out.println("System error message:");
            System.out.println();
            DFSMO5.DFSMO5_PagesLocal.DFSMO5_Page1Local outputPage1 =
              outputPages[0].getDFSMO5_Page1();
            System.out.println(outputPage1.getPP1_OUTL01());
          }
        } catch (Exception e) {
          e.printStackTrace();
        }   }
        }
```

f. Press **Ctrl-S** to save the changes and then close the editor. If you see compilation errors in the code, ensure that you used the correct name when you generated the proxy.

g. Select **TestPhoneBookMFS.java** and expand the **Run** icon on the toolbar by selecting the arrow beside it. From the pop-up menu select **Run as > Java application.**



h. The Java application should run without exceptions, and you should see the following on the Console:



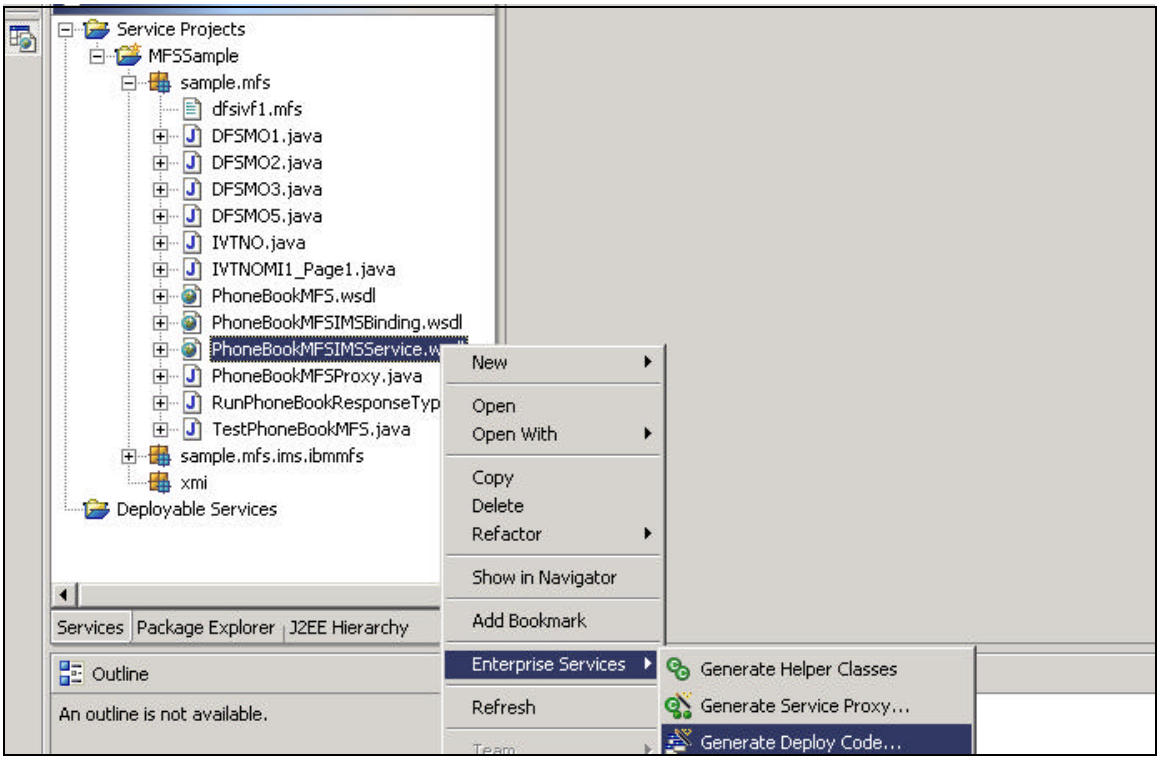
Successfully running the application ensures that your core application logic works correctly.

Note that this example assumes that your IMS system has the non-conversational COBOL version of the IMS INSTALL/IVP program installed and that the pre-loaded entries in the IVPDB2 database have not been modified during previous testing.

2e. Generate deploy code for the enterprise service.

You will now generate an EJB session bean using the Generate Deploy Code wizard. The session bean will be used to hand the client request to the PhoneBook service. A PhoneBook request is a request to run the IMS transaction IVTNO to add, delete, update or display information that is stored in the IMS PhoneBook database. In addition to generating the session bean, the wizard also generates deployed classes that allow the session bean to operate on an EJB server such as the WebSphere Application server.

a. In the **MFSSample** project, expand the **sample.mfs** package and select the **PhoneBookMFSIMSService.wsdl** service file.



b. Right click the file and select **Enterprise Services > Generate Deploy Code.** The wizard opens up.

c. In the **Deployment** page, you will see default values based on the service file that you selected earlier. You should see:

   - **Service file name:** /MFSSample/sample/mfs/PhoneBookMFSIMSService.wsdl
   - **Service name:** PhoneBookMFSIMSService
   - **Port name:** PhoneBookMFSIMSPort

d. Ensure that **Create a new port and binding** is selected.

e. Because you may want to create a service that uses SOAP, ensure that **SOAP** is selected from the **Inbound binding type** list. Note that even if you only plan to deploy the service as a SOAP service, you still need to create two inbound bindings (SOAP and EJB) to access the service. The EJB binding is required because the SOAP binding is built on top of the EJB binding.

f. Default project names (based on your service project name) are provided for you in the following fields:

   - **EAR project:** MFSSampleEAR
   - **EJB project:** MFSSampleEJB
   - **Web project:** MFSSampleWeb

   These are new projects and will be created for you. Accept all other defaults and click **Next.**

g. In the **Inbound Service Files** page, accept the default values. Note that the name of the service is PhoneBookMFSService. Click **Next.**

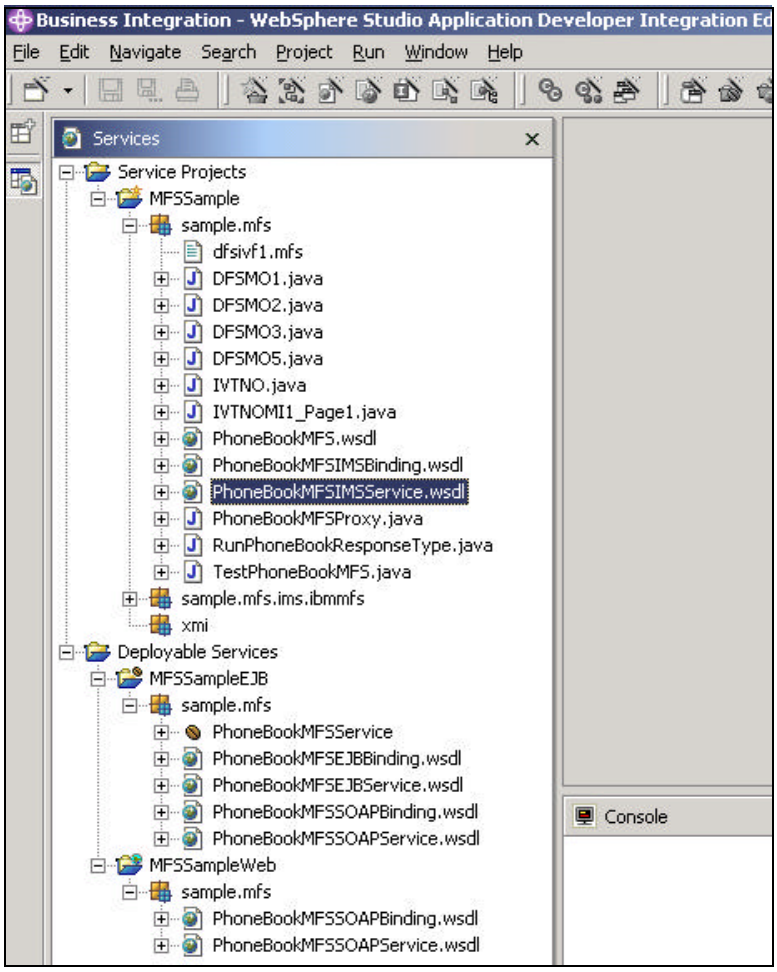
h. In the **EJB Inbound Service Files** page, accept the default values. Click **Next.**

i. In the **EJB Port** page, accept the default value for the **JNDI name**. Click **Next.**

j. In the **SOAP** page, you can accept the default transport, style, action and encoding for the SOAP binding properties. These properties are all specific to the SOAP specification. Click **Next**.

k. In the **SOAP Port** page, you can specify the **SOAP port address**. Accept the default that is provided for you. Click **Finish.** This completes the wizard. Wait while the code is generated.

l. To see what you have generated, look in the **Services** window of the tool (left side). Under **Deployable Services**, the MFSSample Web project contains the service file PhoneBookMFSSOAPService.wsdl and the binding file PhoneBookMFSSOAPBinding.wsdl. The MFSSampleWeb project also includes a SOAP deployment descriptor that tells the SOAP server about the service. The deployment descriptor contains servlet initialization and mapping information, as well as additional settings for running the Web module within an application server. Also under **Deployable Services**, the MFSSampleEJB project contains all of the resources for EJB applications, including the session bean (PhoneBookMFSService), the deployment descriptor, the remote interface, and the EJB home. The service project (MFSSample) contains the service definition, and this project is zipped as a JAR file and placed in the enterprise application project (MFSSampleEAR), which can be viewed in the J2EE view. Here is a screen capture of the Service, EJB, and Web projects in the Services view:

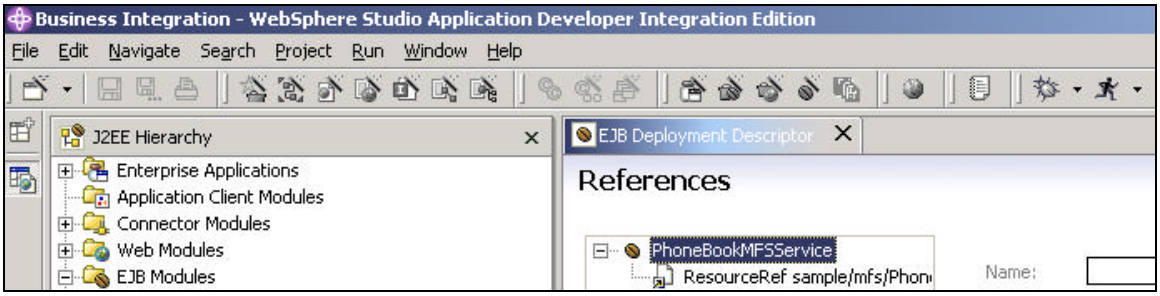


2f. Bind the Resource Reference.

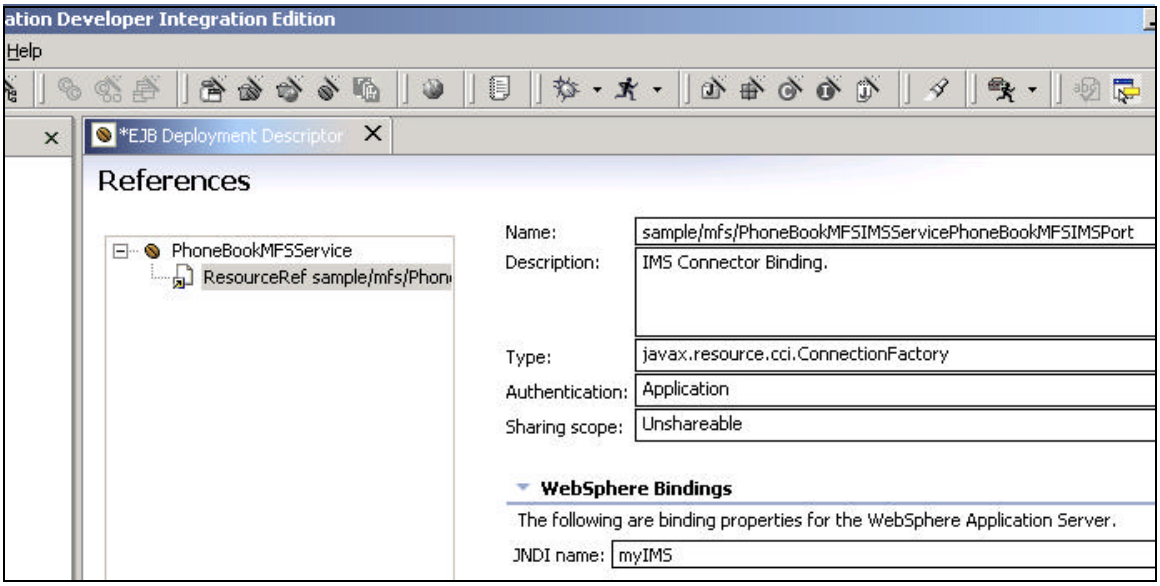
In this step you will create the information that is needed to generate a connection during run time. To do this you will bind a resource reference to a resource. The resource reference includes the Java Naming and Directory Interface (JNDI) related information. This is what the factory object uses at run time to generate a connection when needed by the application.

1. Click the **J2EE Hierarchy tab** in the Business Integration perspective and expand **EJB Modules**. Double click **MFSSampleEJB** to open the Deployment Descriptor Editor. Click the **References** tab.

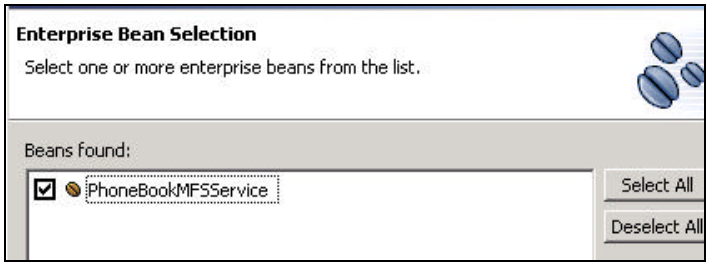
2. Expand **PhoneBookMFSService** and select the **ResourceRef** element.



3. Select **Application** in the Authentication field.
4. Select **Unshareable** for the Sharing scope.
5. Under **Websphere Bindings**, type **myIMS** for the JNDI name.



6. Click the **Assembly Descriptor** tab. Under Container Transactions, click **Add.** In the wizard, select the **PhoneBookMFSService** check box and click **Next.**



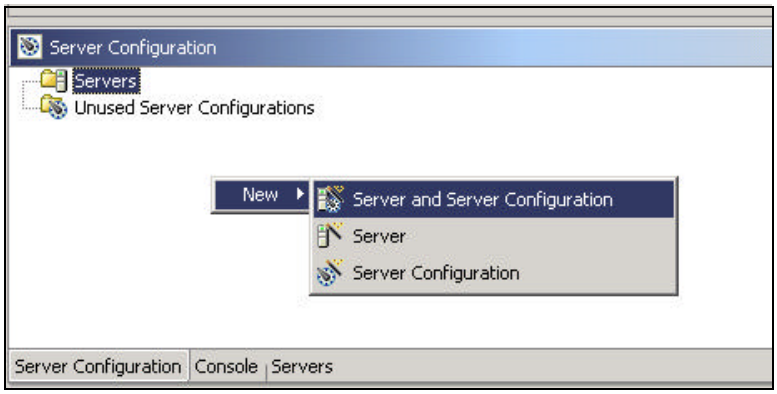
7. In the Container transaction type field, select **NotSupported.** Expand **PhoneBookMFSService** and select the **runPhoneBook** method (make sure it is the only method checked). Click **Finish.**
8. Press **Ctrl-S** to save the changes and then close the editor.

2g. Configure the server and deploy the EAR project

Before running the service, you must deploy the session bean (created in **2e**.) to a server. In this case, the server we will use is a test server that runs in the WSAD IE tool and is called the Websphere Test Environment. This means that the test environment server must be configured and started. For the service that you created, you will need to create one server instance and a corresponding server configuration. A server instance identifies the run time environment that you want to use for testing your project resources. A server configuration contains information that is required to set up and publish to a server. After you configure the server, you will deploy the EAR project containing the service.

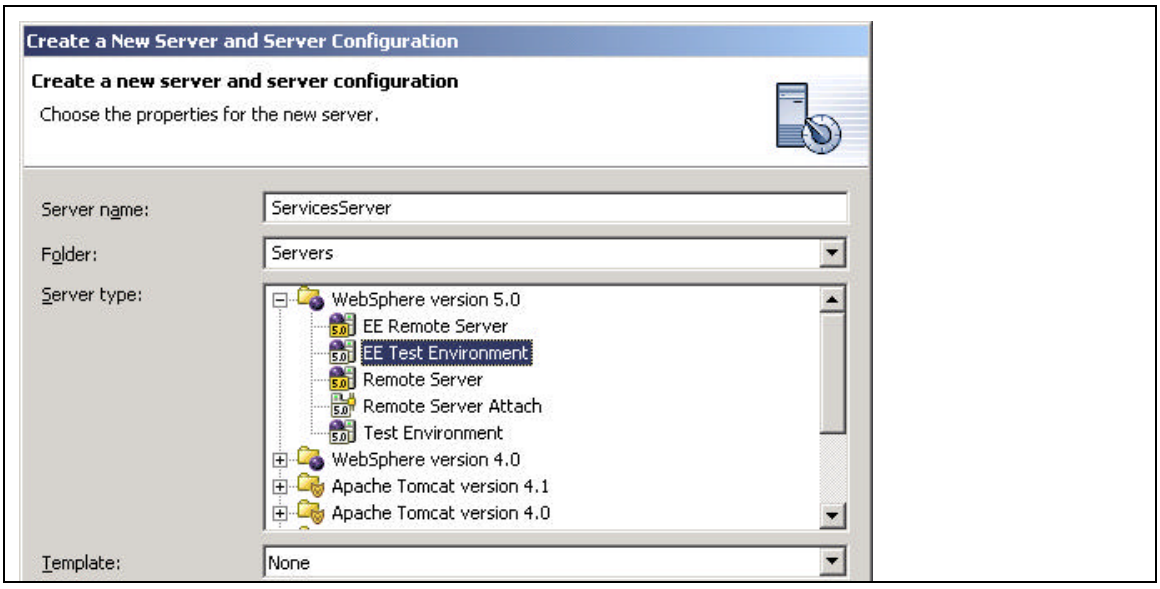
1. Create and configure the server instance.

   a. In the Business Integration perspective, click the **Server Configuration** tab to open the Server Configuration view. Right click anywhere in the Server Configuration view. Select **New > Server and Server Configuration.** The Create a New Server and Server Configuration wizard opens.

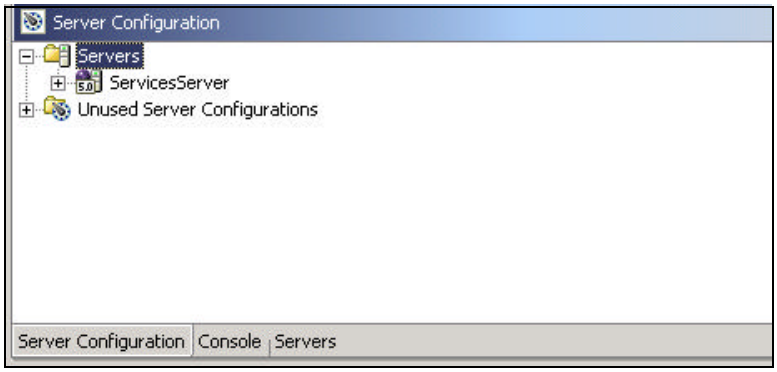


   b. Type **ServicesServer** for the server name. (The default folder name is Servers.)

   c. Expand **WebSphere version 5.0** and select **EE Test Environment.** Leave the template set to **None.** Click **Next.**



d. The server port number defaults to **9080.** The port identifies the location of the service. Click **Finish.** The new server instance appears in the Server Configuration view and in the Servers view.



You have just created a test instance of the WebSphere Application Server that is emulated in WSAD IE WebSphere Test Environment runing on your local host port 9080.

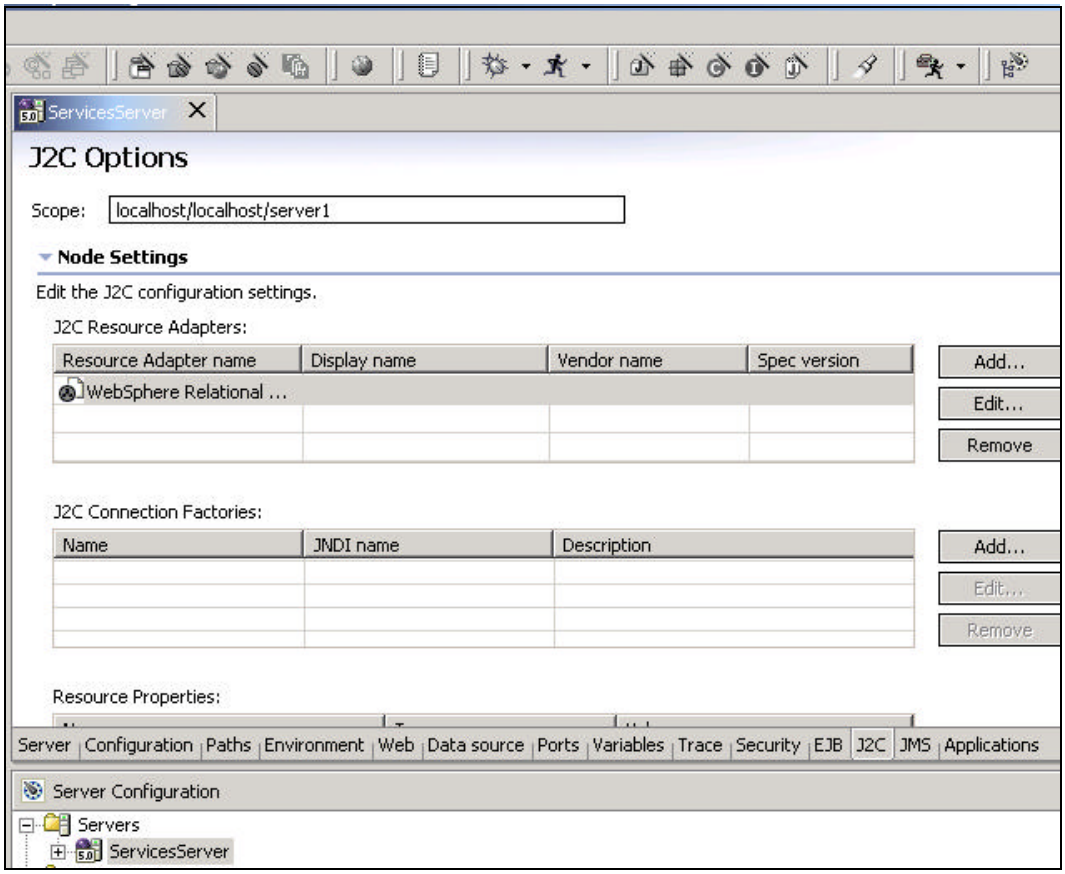
2. Add a Connection Factory to the server configuration.

A Connection Factory is the facility that provides connections to the target EIS (IMS in our case) on demand or as needed. To do this, you will need to add an instance of the JCA connection factory and configure its properties. In the next set of steps you will specify the HostName, DataStoreName, and PortNumber properties that determine which specific IMS should be accessed. You will also specify the JNDI lookup name (myIMS - refer back to Step 2f, number 5) under which the new conection factory instance will be available to components. The components can use this lookup name to quickly make a connection to IMS.

a. Click the **Server Configuration** tab to see the Server Configuration view. Expand **Servers.**

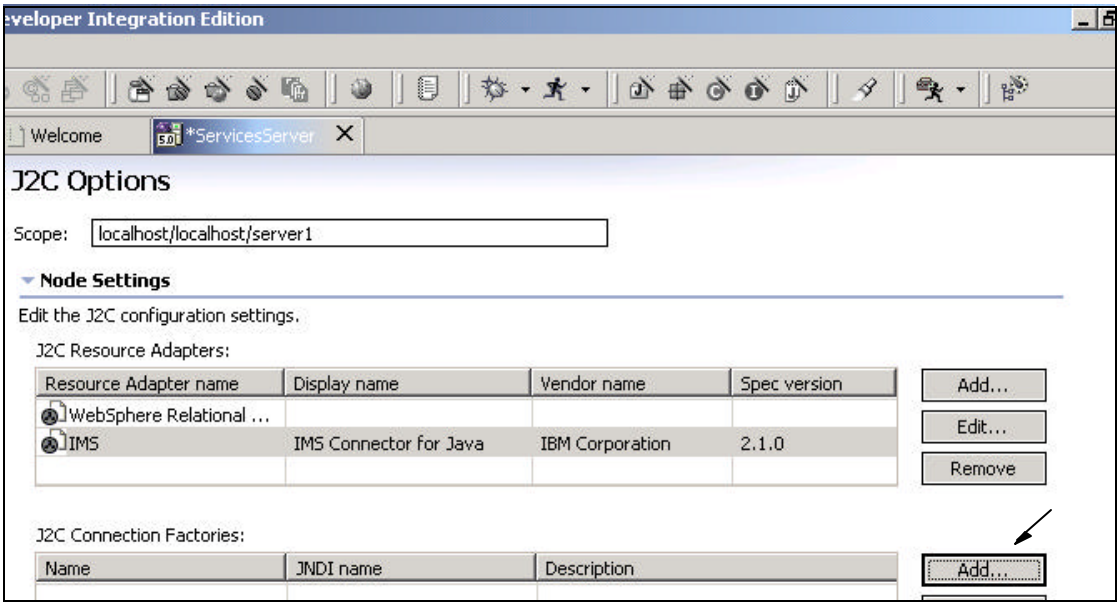
b. Double click the server configuration ServicesServer which will cause an editor to open.

c. Click the **J2C** tab then click **Add** beside the J2C Resource Adapters table.



d. From the Resource Adapter Name list that pops up, select the IMS resource adapter. Click **OK.** This adds the IMS resource adapter into the table.

e. In the same J2C Resource Adapter table, select the newly added IMS resource adapter (make sure it is highlighted), then go to the next table which is the J2C Connection Factories table and click **Add.** The application client will look up this connection factory instance using the JNDI interface. The application client will then use this connection factory instance to get a connection to the underlying IMS.



f. In the Create Connection Factory window, type **ims_cf** for the Name and **myIMS** for the JNDI name. Click **OK.**

g. Scroll down to the next table which is the Resource Properties table.

   - for **HostName**, key in **10.31.227.153**

   - for **DataStoreName,** key in **IM7P** (make sure you use capital letters where appropriate)

   - for **PortNumber**, key in **34**xx - where xx is your team number

   This is important - position your cursor to another field (e.g., UserName) but don't key in anything. This is done to make sure that the actual values you keyed in have been accepted.

h. Press **Ctrl-S** to save the changes and then close the editor.

3. Add the EAR project to the server configuration. This will allow the server access to the application.

   a. In the Server Configuration view under Servers, right click **ServicesServer.**

   b. Select **Add > MFSSampleEAR**. MFSSampleEAR is the name of the Enterprise Application Project that you created earlier.

   You have now successfully generated an enterprise service from an MFS-based IMS transaction and deployed that service to the WebSphere Test Environment.

Exercise 5. Create the client application to invoke the enterprise service.

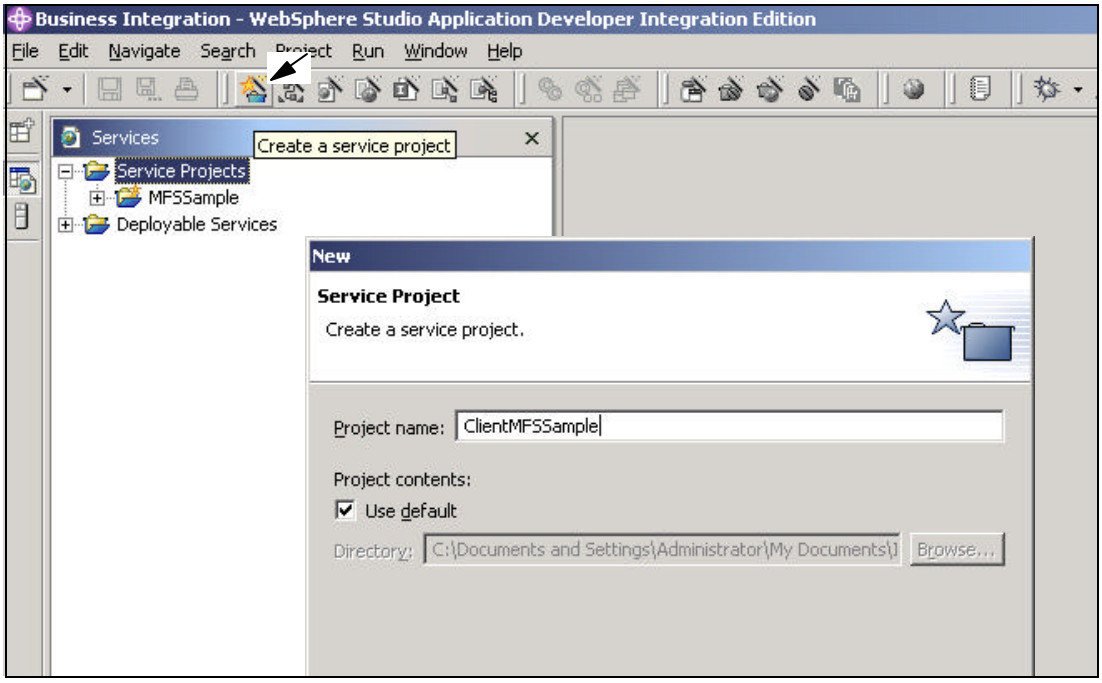
You will now go through the exercise of creating a client application that can invoke the service you created in the previous step. The client application you will generate uses a SOAP proxy.

****

A proxy is needed to provide a remote procedure call interface to the service. This is useful because oftentimes a client application will end up residing on a different platform than the service it needs. The proxy provides the way for the application to call a remote method on the service as if the method were local to the client application. When the client makes the remote call, the proxy handles all of the communication details between the application and the service.

****

1. Create a Service Project for the client.
   a. From the toolbar, click the **Create a service project** icon to start the wizard.
   b. Type **ClientMFSSample** for the project name.



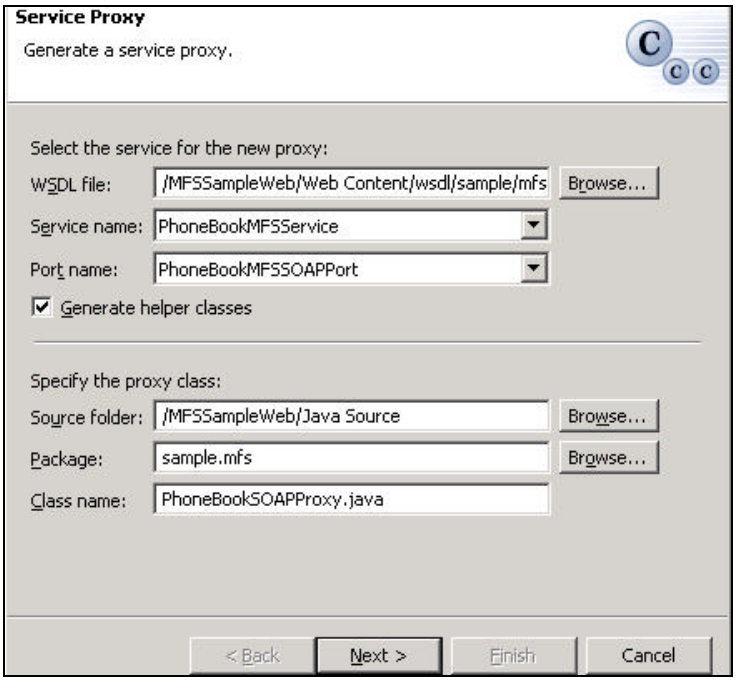
   c. Select **Use default** to use the default location to store the new roject.
   d. Click **Finish** to create the project. You do not have to specifry Java Build Path settings or dependent JARs (In subsequent pages of the wizard) because these are automatically set for you.

2. Generate a SOAP proxy

   a. Earlier in this sample, you created an EAR file that included the SOAP inbound binding type. Click the **Services** tab of the Business Integration perspective. Expand **Deployable Services > MFSSampleWeb > sample.mfs** and select **PhoneBookMFSSOAPService.wsdl.**

b. Right click the file and select **Enterprise Services > Generate Service Proxy.** The Generate Proxy wizard opens.

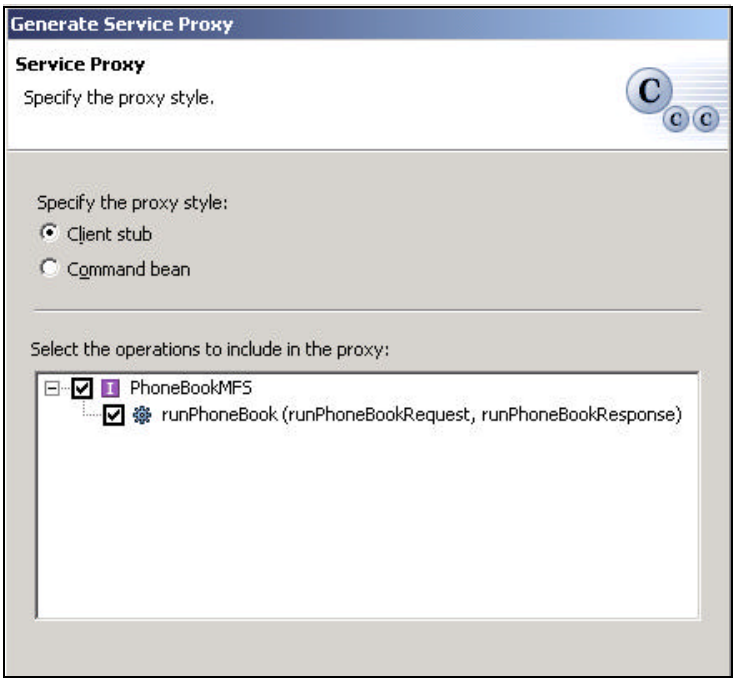
c. In the Service Proxy page, the fields contain default values that are based on the service file you selected. Change the **Source folder** field to place the proxy in the client project. Click **Browse** or type **/MFSSampleWeb/Java Source** for the **Source folder.**



d. Ensure the package name is **sample.mfs** and the **Generate helper classes** check box is selected.

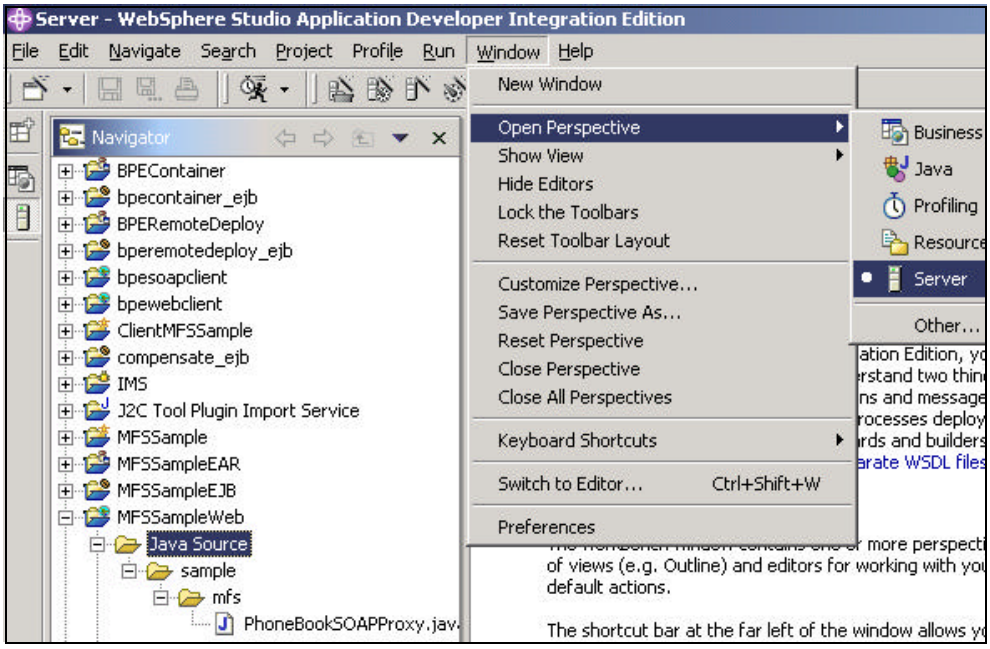
e. Change the **Class name** to **PhoneBookSOAPProxy.java.** Click **Next.**

f. In the next panel, select **client stub** for the **Proxy style** and then select the **runPhoneBook** check box.



g. Click **finish.**

The SOAP proxy, PhoneBookDSOAPProxy, is generated in the /MFSSampleWeb/Java Source folder. You can see this by opening the Server Perspective and looking in the Navigator panel.



3. Create the JSP files for the client application.

   You will now write the JSP (Java Server Pages) files to bind the client to the service and to actually invoke the service. Within your Java package, you will create three new JSP files to test the SOAP proxy that you just created.

   - A JSP file for the input named **MFSSampleController.jsp**
   - A JSP file to display results named **Display.jsp**
   - A JSP file to display errors named **error.jsp**

   The steps to create the files are as follows:

   a. Create a JSP file to accept the user input.

      i. In the Package explorer, right-click **MFSSampleWeb** and select **New > Other,**

      ii. Select **Web** and then select **JSP File.** Click **Next**. The **New JSP File** wizard opens.

      iii. Make sure that the folder is **/MFSSampleWeb/Web Content** and type **MFSSampleController.jsp** for the File Name. Click **Finish.**

      iv. **CLICK THE "SOURCE" TAB** and replace the code in the editor with the following code (also available for cut and paste in file C:\class\jsp1.txt.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
```

```
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">

<TITLE>MFSSampleController.jsp</TITLE>
</HEAD>
<BODY>
<FORM METHOD="POST" ACTION="Display.jsp"><BR>
<H1>Query the PhoneBook application!</H1>
<P>To display a phone book entry, type DISPLAY in the
command field and type a last name.
To add or delete an entry, type ADD or DELETE in the command field and
fill in other fields with the required information. </P>
<TABLE>
<TR><TD>Command: </TD>
<TD><INPUT TYPE="TEXT" NAME="Cmd" VALUE="" SIZE="10" MAXLENGTH="50"></TD>
</TR>
<TR><TD>Last name: </TD>
<TD><INPUT TYPE="TEXT" NAME="LName" VALUE="" SIZE="10" MAXLENGTH="50"></TD>
</TR>
<TR><TD>First name: </TD>
<TD><INPUT TYPE="TEXT" NAME="FName" VALUE="" SIZE="10" MAXLENGTH="50"></TD>
</TR>
<TR><TD>Extension: </TD><TD>
<INPUT TYPE="TEXT" NAME="Extn" VALUE="" SIZE="10" MAXLENGTH="50"></TD>
</TR>
<TR><TD>Zip code: </TD>
<TD><INPUT TYPE="TEXT" NAME="Zip" VALUE="" SIZE="10" MAXLENGTH="50">
</TD></TR>
</TABLE><Br>
<INPUT TYPE="SUBMIT" NAME="Submit">
</FORM>
</BODY>
</HTML>
```

v. Press **Ctrl-S** to save the file and then close the editor. (Note: you can ignore any broken link messages because they will be resolved when you build the next two JSP files).

b. Create a JSP file to display the output.

i. In the Package explorer, right-click **MFSSampleWeb** and select **New > Other,**

ii. Select **Web** and then select **JSP File.** Click **Next**. The **New JSP File** wizard opens.,

iii. Make sure that the folder is **/MFSSampleWeb/Web Content** and type **Display.jsp** for the File Name. Click **Finish.**

iv. **CLICK THE "SOURCE" TAB** and replace the code in the editor with the following code (also available for cut and paste in file C:\class\jsp2.txt.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
import="javax.naming.*, sample.mfs.*"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
errorPage="error.jsp"
%>

<META HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE">
<TITLE>Display.jsp</TITLE>
</HEAD>
<BODY>
<%
response.setHeader("Cache-Control","no_store"); //HTTP 1.1
response.setHeader("Cache-Control","must-revalidate"); //HTTP 1.1
response.setHeader("Pragma","no-cache"); //HTTP 1.0
response.setDateHeader ("Expires", 0); //prevents caching at the proxy server
response.setHeader("Cache-Control","no-store"); //HTTP 1.1

PhoneBookSOAPProxy phoneBookProxy = new PhoneBookSOAPProxy();

IVTNOMI1_Page1 inputMessage = new IVTNOMI1_Page1();
inputMessage.setCMD(request.getParameter("Cmd"));
inputMessage.setNAME1(request.getParameter("LName"));
inputMessage.setNAME2(request.getParameter("FName"));
inputMessage.setEXTUnicode0023(request.getParameter("Extn"));
inputMessage.setZIP(request.getParameter("Zip"));

RunPhoneBookResponseType outputMessage = phoneBookProxy.runPhoneBook(inputMessage);

if (outputMessage.getDFSMO1() != null) {
out.println("DFSMO1");
DFSMO1 outputPart = outputMessage.getDFSMO1();
DFSMO1.DFSMO1_PagesLocal[] outputPages = outputPart.getDFSMO1_Pages();
out.println("System error message:");
out.println();
DFSMO1.DFSMO1_PagesLocal.DFSMO1_Page1Local outputPage1 =
outputPages[0].getDFSMO1_Page1();
out.println(outputPage1.getOUTLINE());
}
```

```
if (outputMessage.getDFSMO2() != null) {
out.println("DFSMO2");
DFSMO2 outputPart = outputMessage.getDFSMO2();
DFSMO2.DFSMO2_PagesLocal[] outputPages = outputPart.getDFSMO2_Pages();
out.println("System error message:");
out.println();
DFSMO2.DFSMO2_PagesLocal.DFSMO2_Page1Local outputPage1 =
outputPages[0].getDFSMO2_Page1();
out.println(outputPage1.getPP1_OUTL01());
}
if (outputMessage.getDFSMO3() != null) {
out.println("DFSMO3");
DFSMO3 outputPart = outputMessage.getDFSMO3();
DFSMO3.DFSMO3_PagesLocal[] outputPages = outputPart.getDFSMO3_Pages();
out.println("System error message:");
out.println();
DFSMO3.DFSMO3_PagesLocal.DFSMO3_Page1Local outputPage1 =
outputPages[0].getDFSMO3_Page1();
out.println(outputPage1.getOUTL01());
}
if (outputMessage.getDFSMO5() != null) {
out.println("DFSMO5");
DFSMO5 outputPart = outputMessage.getDFSMO5();
DFSMO5.DFSMO5_PagesLocal[] outputPages = outputPart.getDFSMO5_Pages();
out.println("System error message:");
out.println();
DFSMO5.DFSMO5_PagesLocal.DFSMO5_Page1Local outputPage1 =
outputPages[0].getDFSMO5_Page1();
out.println(outputPage1.getPP1_OUTL01());
}

%>
<H1>Query Results</H1>
<hr WIDTH="50%" ALIGN="LEFT">
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="50%">
<TR ALIGN="left" VALIGN="middle">
<TH>Field</TH>
<TH>Value</TH></TR>
<TR ALIGN="left" VALIGN="middle">
<TD>Last name</TD>
<TD>

<%
    IVTNO.IVTNO_PagesLocal.IVTNO_Page1Local outputPage1;
    outputPage1 = null;
```

```
    IVTNO outputPart = outputMessage.getIVTNO();
    IVTNO.IVTNO_PagesLocal[] outputPages = outputPart.getIVTNO_Pages();
    outputPage1 = outputPages[0].getIVTNO_Page1();
%>
<%= outputPage1.getNAME1() %>
</TD></TR>
<TD>First name</TD>
<TD>
<%= outputPage1.getNAME2() %>

</TD></TR>
<TR ALIGN="left" VALIGN="middle">
<TD>Extension</TD>
<TD>
<%= outputPage1.getEXTUnicode0023() %>

</TD></TR>
<TR ALIGN="left" VALIGN="middle">
<TD>Zip code</TD>
<TD>
<%= outputPage1.getZIP() %>
</TD>
<TR ALIGN="left" VALIGN="middle">
<TD>Date</TD>
<TD>
<%= outputPage1.getSDATE() %>
</TD></TR>
</TD></TR>
</TABLE>
<hr WIDTH="50%" ALIGN="LEFT">
<P>Status: <%= outputPage1.getMSG().trim() %></P>
</BODY>
<HEAD>
 <META HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE">
</HEAD>
</HTML>
```

v. Press **Ctrl-S** to save the file and then close the editor.

c. Create a JSP file to display any errors encountered during the query.

i. In the Package explorer, right-click **MFSSampleWeb** and select **New > Other,**

ii. Select **Web** and then select **JSP File.** Click **Next**. The **New JSP File** wizard opens.

iii. Make sure that the folder is **/MFSSampleWeb/Web Content** and type **error.jsp** for the File Name. Click **Finish.**

iv. **CLICK THE "SOURCE" TAB** and replace the code in the editor with the following code (also available for cut and paste in file C:\class\jsp3.txt.

```
<%@ page isErrorPage="true" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<head>
 <title>Error Page</title>
</head>
<BODY text="#000000" bgcolor="#FFFFC0" link="#0000FF" vlink="#800080"
alink="#FF00FF">
<center>
<h2>Error</h2></center>
<P>Application <B>PhoneBook</B> reported the following error:
<P>
<font color="#3333FF"><%=exception.toString() %></FONT>
<P>This problem occurred in the following place:
<P>
<PRE>
<% exception.printStackTrace(new java.io.PrintWriter(out));  %>
</PRE>
</BODY>
</HTML>
```

v. Press **Ctrl-S** to save the file and then close the editor. Now you can use the JSP files to test the SOAP proxy.

4. Test the SOAP proxy.

   Now you are ready to test the SOAP proxy by using the input JSP to send requests tot he PhoneBook application. To test the proxy, complete the following steps:

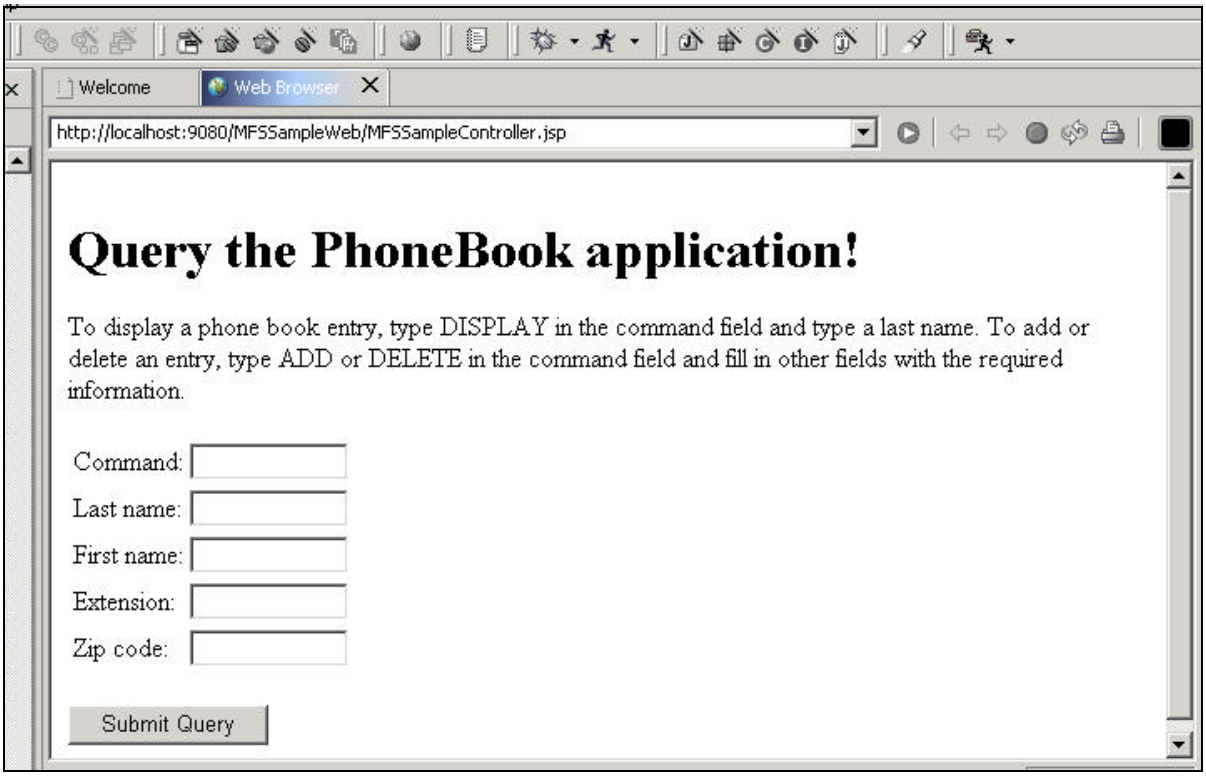
   a. Open the server perspective by going to the top of the screen and selecting **Window > Open Perspective > Server.**

   b. In the **Navigator** window, highlight **MFSSampleWeb,** right-mouse click and select **Rebuild Project.**

   c. Go to the top of the screen and selecting **Window > Open Perspective > Business Integration.** In the **Package Explorer** window, expand **MFSSampleWeb > Web Content.**

   d. Right-click **MFSSampleController.jsp** amd click **Run on server.**

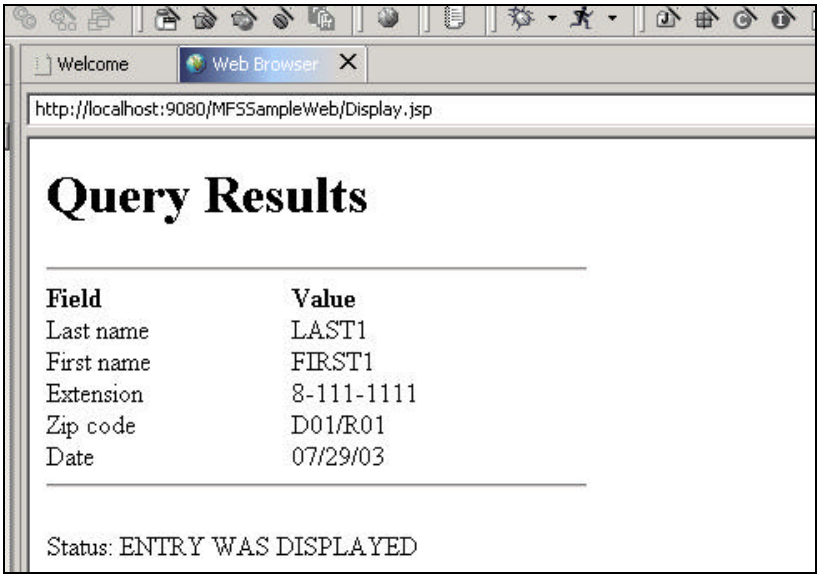
   e. If the Server Selection wizard displays, select **Use an existing server** and select the server you configured for this sample, **ServicesServer**. Click **Finish.** The server starts and launches the JSP. YOu should see the following in the display.



Issue a query to display a phone book entry. Type the following values in the fields on the input JSP:

Field	Value
Command	**DISPLAY**
Last name	**LAST1**

f. Click **Submit Query**. You should see the following:



g. Click the back button in the browser to return to the input JSP.

h. Issue a query to add a phone book entry. Type the following values in the fields on the input JSP:

Field	Value
Command	**ADD**
Last name	your last name
First name	your first name
Extension	your extension
Zip code	your zip code

Click **Submit Query.**

You can also try other commands such as: **DELETE and DISPLAY**

RETURN TO INDEX