Create a J2C application for IMS Phonebook transaction using IMS TM Resource Adapter

Level: Intermediate

Evgueni Liakhovitch (evgueni@us.ibm.com)
Software Developer, IBM®

Yee-Rong Lai (ylai@us.ibm.com)
Information Developer, IBM

Shahin Mohammadi-Rashedi (shahin@us.ibm.com)
IMS™ SOA Demonstration Team Lead, IBM

15 March 2009

Abstract

This tutorial takes you through the steps of using the J2EE Connector (J2C) component of the IBM Rational® Developer for System z® Version 7.5, and IBM IMS TM Resource Adapter Version 10 to access IMS transactions through a J2C Java™ bean. After the J2C bean is created, it can be easily consumed by a Web application (JavaServer Pages component, or JSP), Web service, or Enterprise Java Beans (EJB) component.

About this tutorial

This tutorial will take you through the steps of using the J2EE Connector (J2C) components of the IBM Rational Developer for System z, version 7.5 and IBM IMS TM Resource Adapter Version 10.

Many customers have invested heavily in IMS to perform critical transactions for their businesses. The J2C tools available in Rational Developer for System z can expose the interaction specification properties of the IMS TM Resource Adapter and generate a Java class that accesses IMS transactions. Once the J2C bean is created, it can easily be consumed by a Web application (JavaServer Pages component, or JSP), Web service, or Enterprise Java Beans (EJB) component.

This tutorial will help to familiarize you with the Java Platform, Enterprise Edition (Java EE, previously known as J2EE) development environment and the J2C. In this tutorial, you will be working with the sample Phonebook IMS transaction (IVTNO), which is one of the IMS installation verification programs shipped with IMS. As an optional exercise you will also experiment with creating a Web service from the generated J2C bean class, and testing that Web service with the Web Services Explorer provided with Rational Developer for System z.

Rational Developer for System z Version 7.5 has two packages. The Rational Developer for System z with Java is the package that is a superset of IBM Rational Application Developer, Therefore, the J2EE perspective, J2C wizard, and Web tools that are offered in Rational Application Developer are also available in Rational Developer for System z and other IDEs that also include Rational Application Developer, such as IBM Rational Software Architect and IBM WebSphere® Integration Developer. The tutorial instructions also apply to these IDEs, with perhaps minor naming and interface differences.

Objectives

To understand and gain hands on experience extending IMS applications to Web as a part of Web pages or Web services. Software tools that are part of Rational Developer for System z and IMS TM Resource Adapter make the transformation processes easy, as the tutorial will demonstrate. The total cost of ownership for customers, therefore, is reduced.

Upon completion of this study, you will be able to perform these tasks:

- Use Rational Developer for System z and its built-in J2C tools.
- Configure IMS TM Resource Adapter connection factory
- Enable IMS applications as JSP components and Web services

System requirements for the tutorial:

- Software installed on Windows®
    - Rational Developer for System z Version 7.5
- System software installed on IBM z/OS®
    - IMS Version 9 or Version 10
    - IMS Connect Version 9 or Version 10 with XML Adapter configured
    - OTMA
    - TCP/IP

Checklist for the first time implementation

You may find it helpful to have the following checklist available before proceeding with your own implementation for the first time.
A tutorial checklist is provided for this exercise.

Table 1. Implementation checklist

	Your environment	This tutorial:
COBOL copybook	This can be obtained from IMS application programmers.	C:\IMS PhoneBook\IMSPHBK.cpy
IMS Connect host name (or IP address) and port number.	This can be obtained from IMS system programmers.	**Host name:** ZSERVEROS.DFW.IBM.COM **Port number:** 9999
IMS Data store.	This can be obtained from IMS system programmers.	IMSC
Workspace directory and project name will be used by Rational Developer for System z when generating artifacts.	A naming standard is recommended.	Workspace directory: C:\Workspaces7.5\SANDBOX Project name: J2CPhoneBook

Overview of development tasks

To complete this tutorial you will perform the following tasks:

1. Install and configure the IMS TM Resource Adapter. Import the IMS TM Resource Adapter from the file system.
2. Use the J2C Java Bean wizard to create a bean that executes a transaction in IMS. Use the J2C Java Bean wizard to set up the J2EE project, Java interface, and implementations. Create sample JSP client for testing.
3. Deploy and test the sample application. Deploy and test the sample application within the IBM WebSphere Application Server.
4. (Optional) Generate Web Service implementation. Use Rational Developer for System z to generate a Web service implementation from the sample application and test it by using the Web services explorer.

Figure 1 shows how these tools interact to help you accomplish the tutorial tasks.

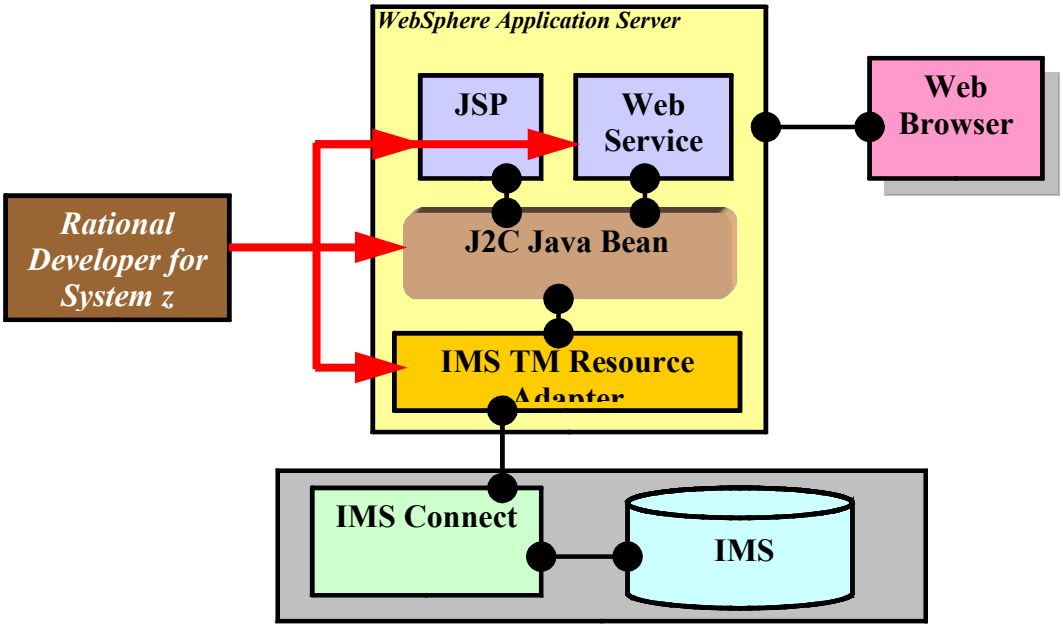


Figure 1. Using Rational Developer for System z to accomplish lab objectives

Task 1. Install the IMS TM Resource Adapter

In this section, you will validate that the WebSphere Application Server runtime environment is available, and then import the IMS TM Resource Adapter.

Using Rational Developer for System z and the J2EE Projects perspective

Start Rational Developer for System z if it is not already started.

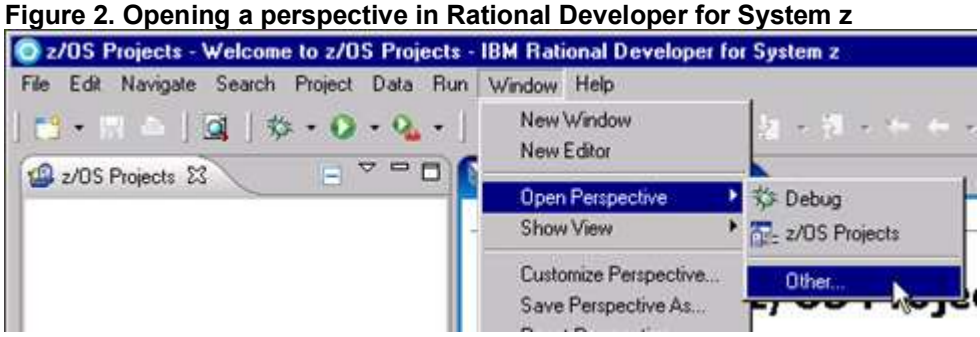
1. Select **Start** > **Programs** > **IBM Software Development Platform** > **IBM Rational Developer for System z** > **IBM Rational Developer for System z**

Switch to the J2EE perspective

Switch from the default z/OS Projects perspective to the J2EE perspective. Within Eclipse, there are several ways to change perspectives.

1. From the **Window** pull down, select **Open Perspective > Other**, as shown in the following Figure.

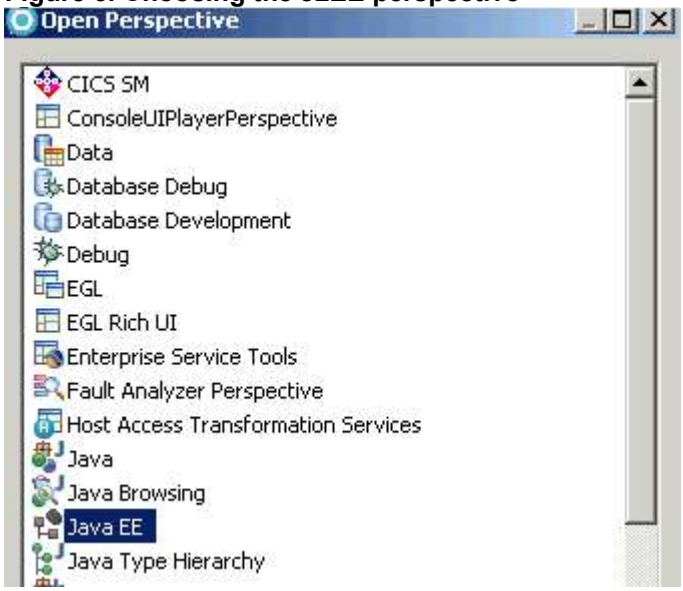
Figure 2. Opening a perspective in Rational Developer for System z



What is a perspective?
A perspective defines the initial set and layout of views in the Workbench window. Within the window, each perspective shares the same set of editors. Each perspective provides a set of functionality aimed at accomplishing a specific type of task, or that works with specific types of resources. For example, the Java perspective combines views that you would commonly use while editing Java source files, while the Debug perspective contains the views that you would use while debugging Java programs.

2. Scroll down and select **J2EE** from the Open Perspective dialog box, as shown in the following Figure:

Figure 3. Choosing the J2EE perspective



What is Java EE?
The Java Platform, Enterprise Edition (previously known as Java Platform, Enterprise Edition, or JEE) provides a standard for developing component-based, multi-tier, enterprise applications.
A Java EE application system typically includes the following tiers:

- **Client tier**: In the client tier, Web components (such as servlets, JSP components, or standalone Java applications) provide a dynamic interface to the middle tier.
- **Middle tier**: In the server tier, or middle tier, enterprise beans and Web services encapsulate reusable, distributable business logic for the application. These server-tier components are contained on a Java EE Application Server, which provides the platform for these components to perform actions and store data.
- **Enterprise data tier**: In the data tier, the enterprise's data is stored and persisted, typically in a relational database.

Java EE applications are comprised of components, containers, and services. Web components, such as servlets and JSPs, provide dynamic responses to requests from a Web page. EJB components contain server-side business logic for enterprise applications. Web and EJB component containers host services that support Web and EJB modules.

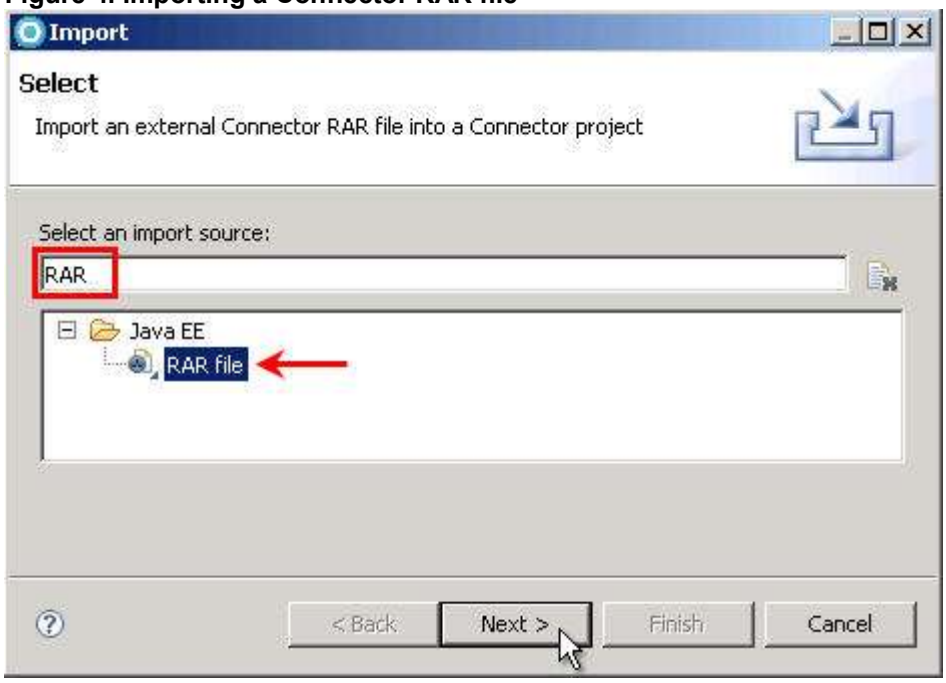
3. Press **OK** to switch to the J2EE Perspective.

Import the IMS TM Resource Adapter

1. Click **File** > **Import** to open the Import dialog box.

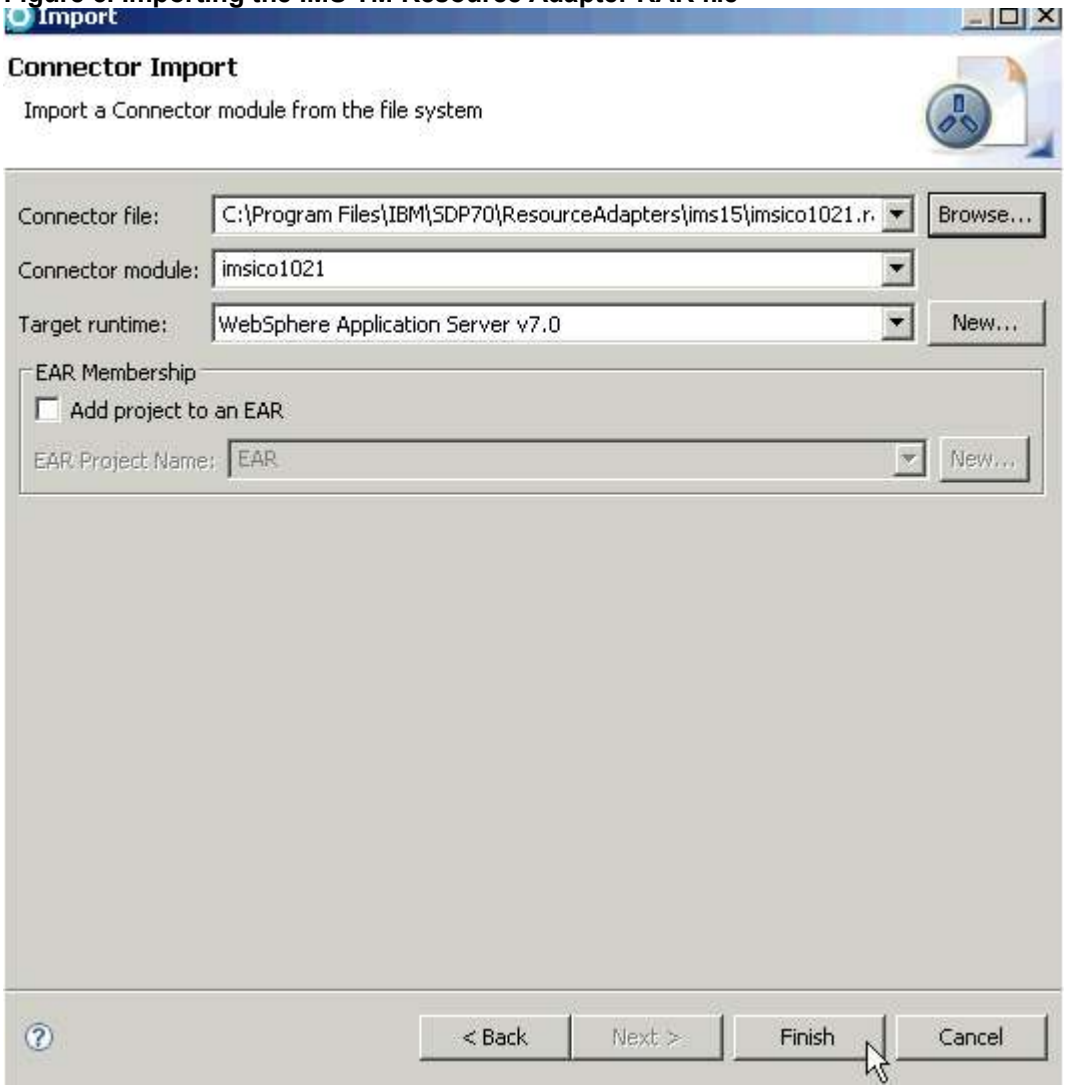
2. Enter `RAR`, select **RAR file**, and then click **Next** to continue, as shown in the following Figure.

Figure 4. Importing a Connector RAR file



3. From the Import dialog, click **Browse** to locate and import the IMS TM Resource Adapter file.

Figure 5. Importing the IMS TM Resource Adapter RAR file

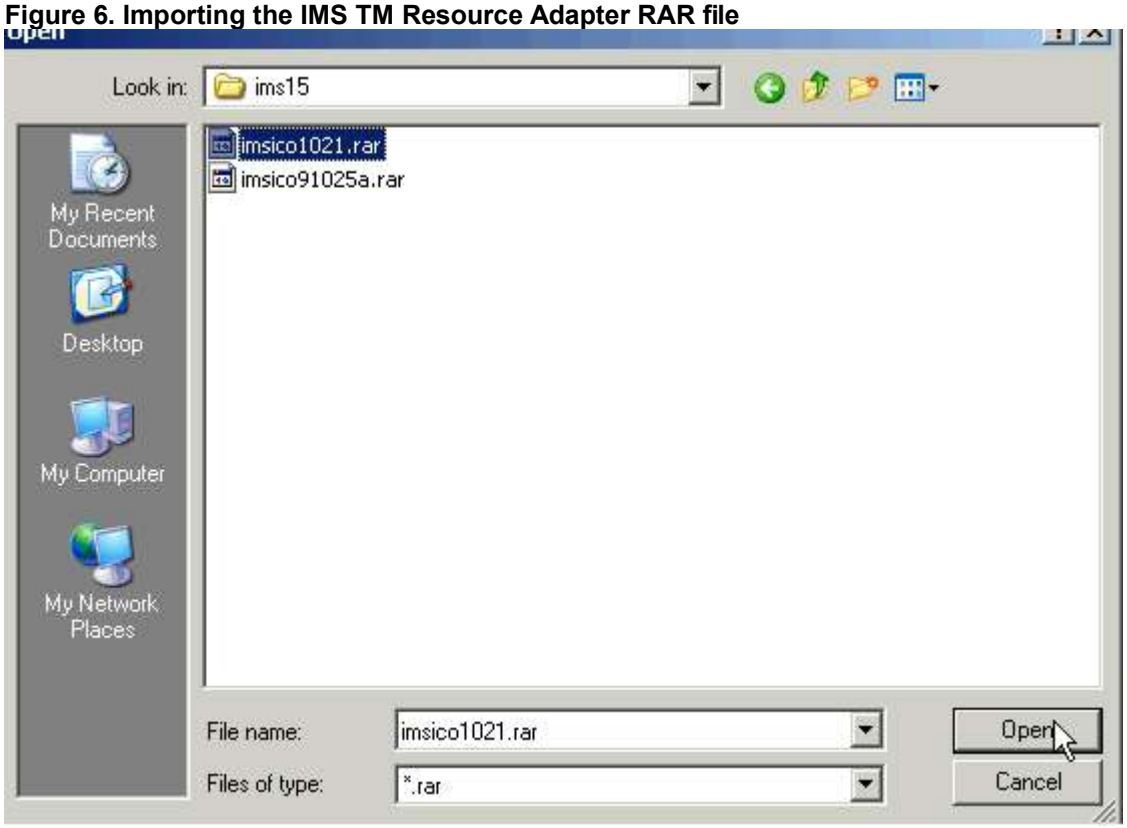


What is a resource adapter?

Resource adapters allow your application to communicate with the enterprise information system (EIS). A resource adapter is a system-level software driver that is used by a Java application to connect to an EIS. The resource adapters reside on the application server and provide connectivity between the EIS, the application server, and the enterprise application. Applications deployed on the application server communicate with the resource adapter using the Common Client Interface (CCI). The RAR contains all the information necessary for installing, configuring, and running a resource adapter. Resource adapters comply with the J2EE Connector Architecture specification. In this lab, you are using IMS TM Resource Adapter to connect to IMS.

4. Navigate to the file C:\Program Files\IBM\SDP70\ResourceAdapters\ims15\imsico1021.rar and click **Open**, as shown in the following Figure. .

Figure 6. Importing the IMS TM Resource Adapter RAR file



5. For **Target runtime, choose** WebSphere Application Server 7.0**, as shown in the following Figure.**

6. Click **Finish**.

The IMS TM Resource Adapter module should now be visible in your J2EE perspective, as shown in the following Figure.

Figure 7. IMS TM Resource Adapter Version 10.2.1 module in J2EE project explorer

Task 2. Create a bean that communicates with IMS

In this section, you will create a bean that communicates with IMS using J2C. You will define whether the bean is a managed or non-managed resource, along with defining the TCP/IP address, port, and IMS datastore name.

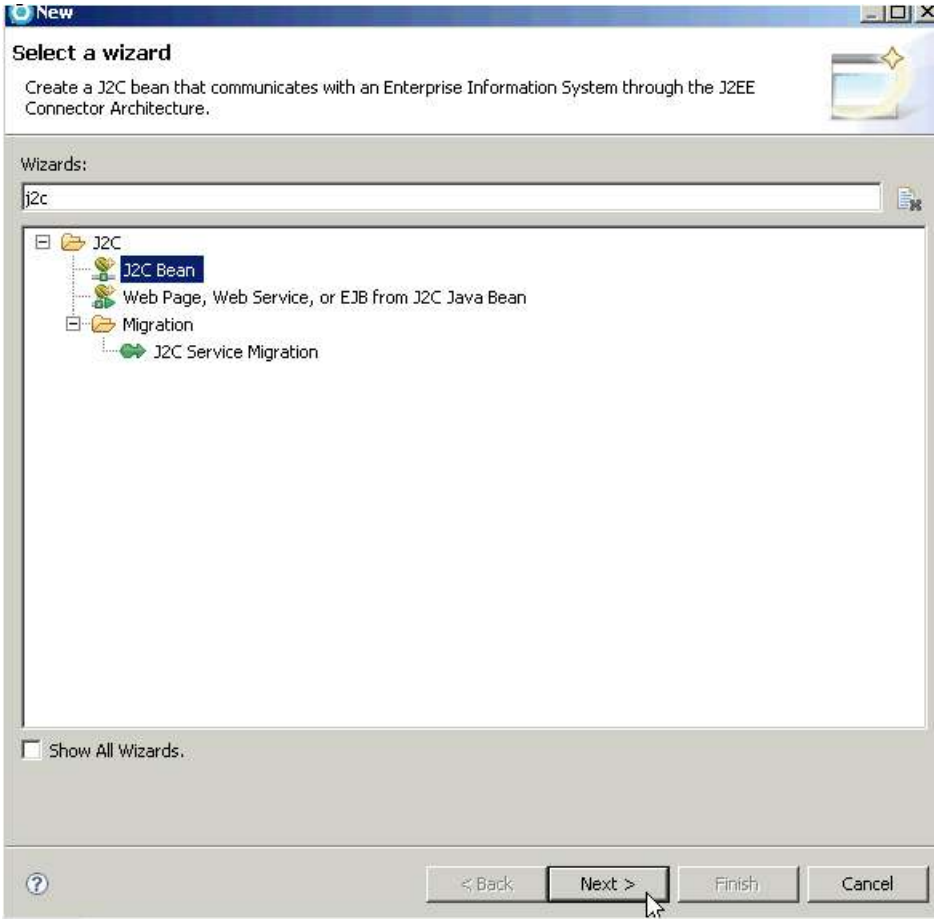
Using the J2C Java Bean wizard

1. Start the J2C Java Bean wizard by clicking **File > New > Other** to open the Select a Wizard dialog, as shown in the following Figure.

Figure 8. Starting the J2C Java Bean wizard

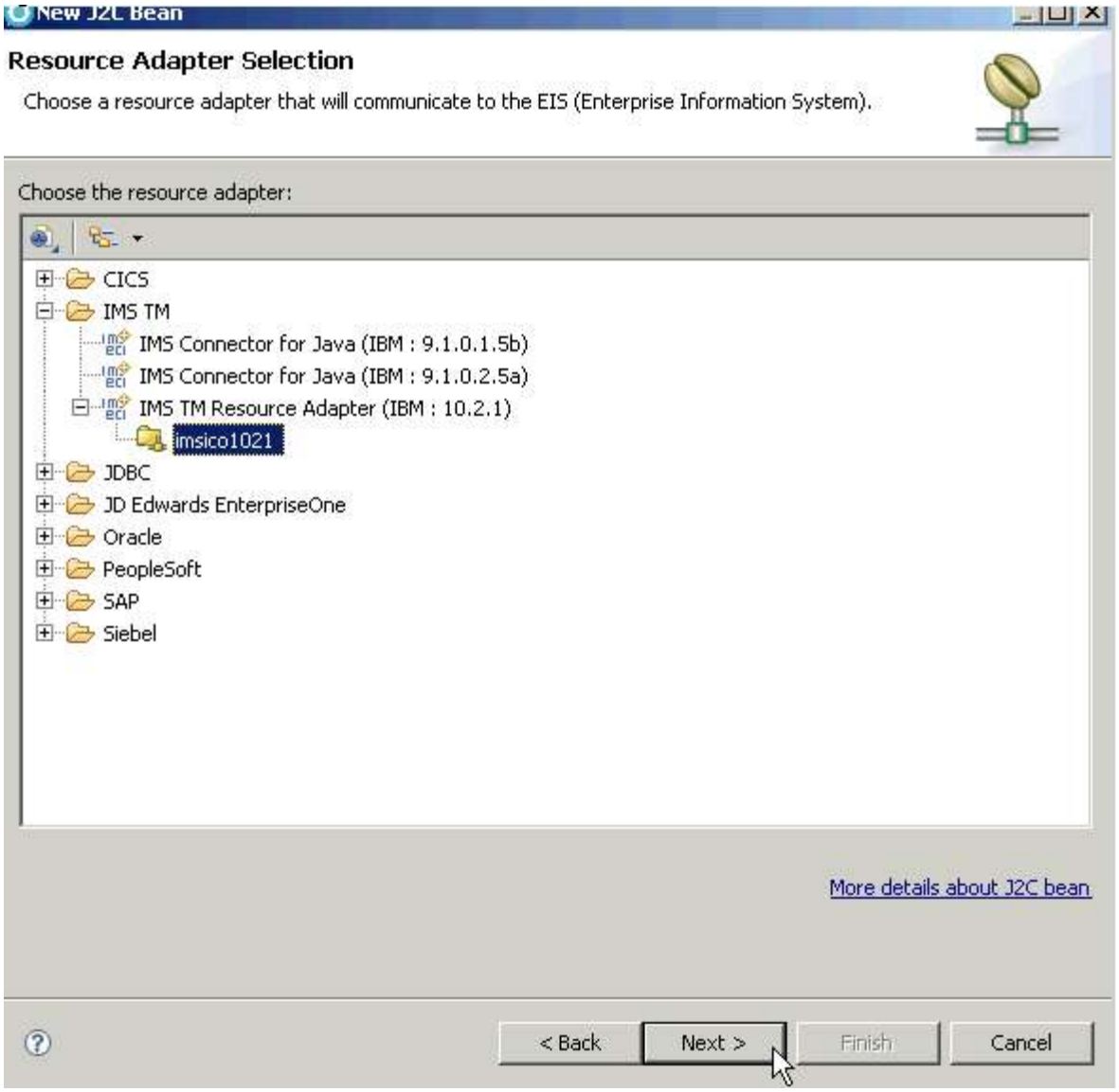
2. Enter `j2c`, select **J2C Java Bean**, as shown in the following Figure, and click **Next:**

Figure 9. Selecting the J2C Java Bean wizard



3. Select the appropriate resource adapter for the J2C Java Bean. Expand **1.5**, expand **IMS TM Resource Adapter (IBM: 10.2.1**), and select **imsico1021**, as shown in the following Figure.

Figure 10. Selecting IMS TM Resource Adapter Version 10.2.1

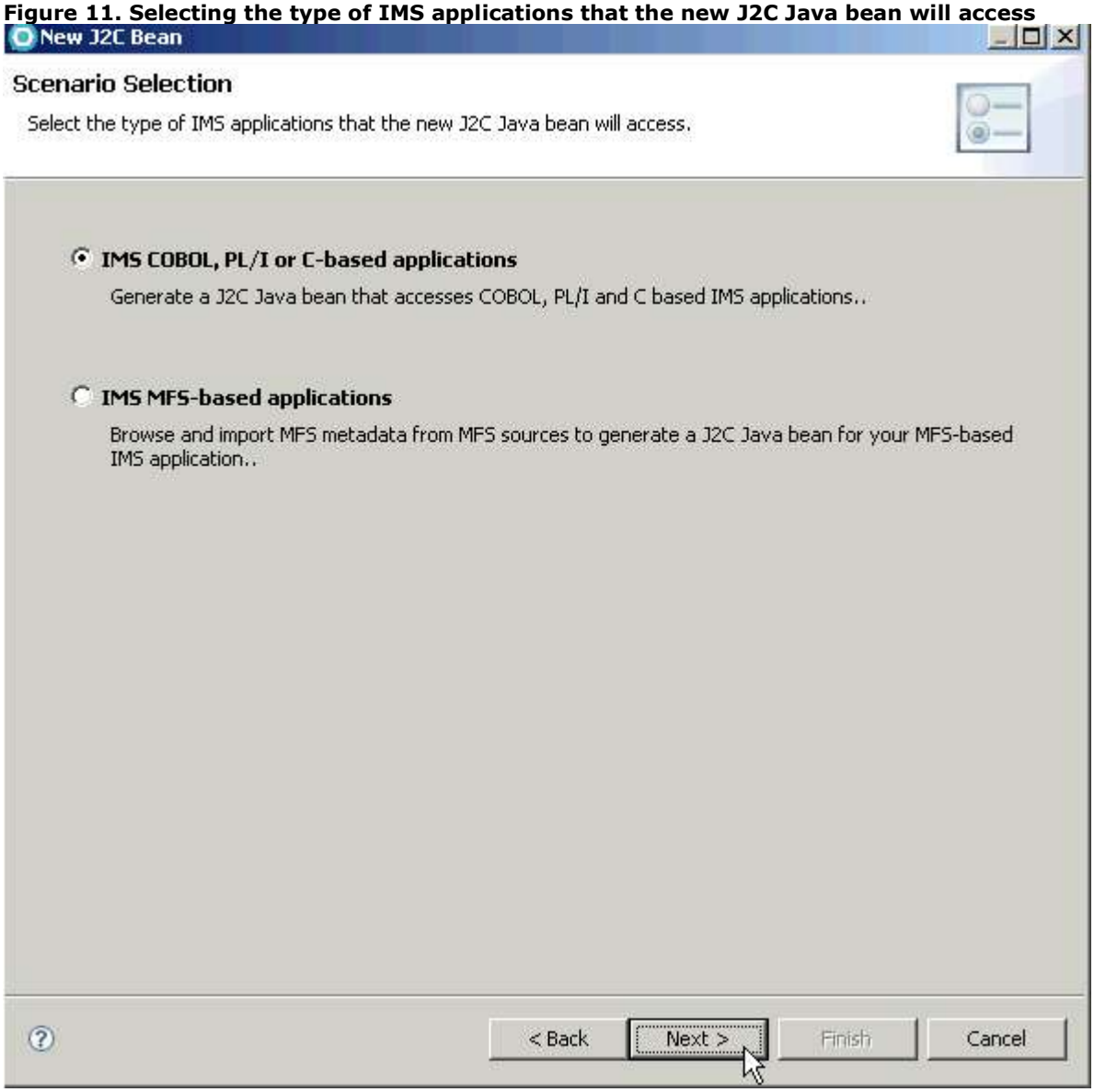


4. Click **Next** to continue. The Scenario Selection page opens.

Rational Developer for System z supports J2EE Connector Architecture Version 1.0 and Version 1.5. IMS TM Resource Adapter Version 10 is based on the newer JCA 1.5 standards, and is therefore located under the 1.5 section.

5. In the Scenario Selection page, select **IMS COBOL, PL/I or C-based applications** and click **Next**.

Figure 11. Selecting the type of IMS applications that the new J2C Java bean will access



Managed and non-managed connections
A **managed connection** runs inside a Web application server. With a managed connection, the application server provides transaction management and connection pool management, and it can send security information. In addition, managed connections allow connection information to be maintained by the system administrator. As connection information changes (the type of communications, the port, and so on), the system administrator can adjust the connection characteristics, and no Java objects need to be regenerated.
A **non-managed connection** is designed to run where connections management supplied by an application server is not available. The characteristics of the connection must be specified, and are hard-coded into the generated object. You can change the connection characteristics from your program, but you will need to generate your J2C code appropriately. Because non-managed connections are not always convenient to change, and they do not take advantage of the connection pooling, transaction management, and security management that are provided by an application server, it is easy to see why managed connections are recommended.

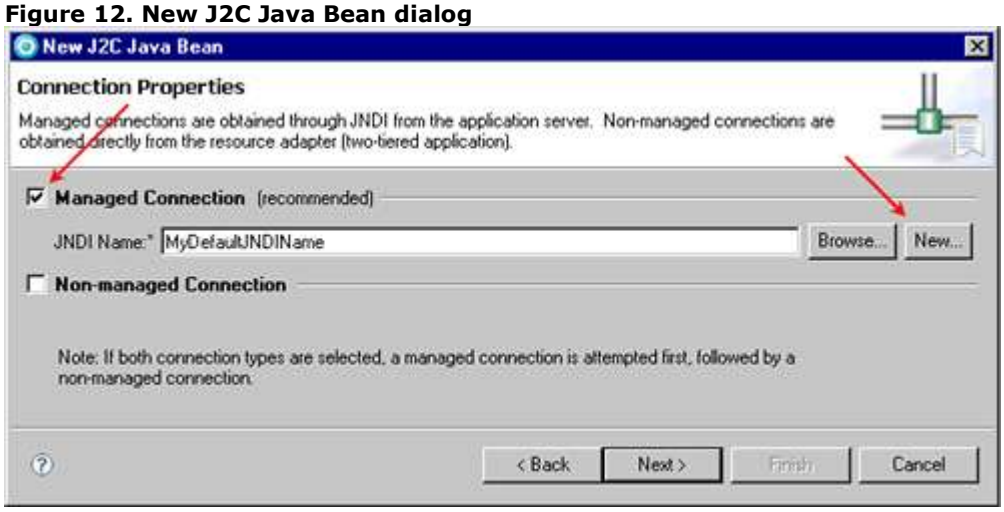
The example in this tutorial uses a managed connection to IMS.

After you have selected the scenario, you must provide the connection properties information. These properties will be stored in a connection factory.

This application will interact with the IMS TM Resource Adapter through an object called the **connection factory**. IMS connection factories are used to create pre-configured connections to the **IMS transaction manager** (IMS TM). When an application uses the IMS TM Resource Adapter, it interacts with IMS using connections between the IMS TM Resource Adapter and IMS Connect that are created by the IMS TM Resource Adapter. These connections can be **managed** or **non-managed**.

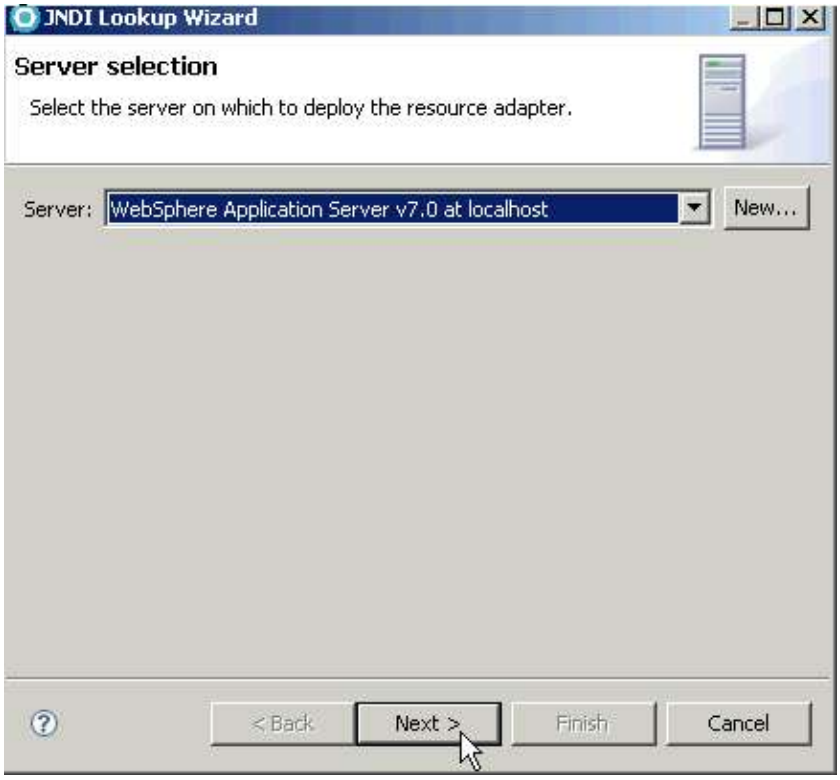
6. Select **Managed Connection** from the Connection Properties page and click **New**, as shown in the following Figure.

Figure 12. New J2C Java Bean dialog



7. Make sure that **WebSphere Application Server v7.0** is selected and click **Next**, as shown in the following Figure.

Figure 13. Selecting the server on which to deploy the IMS TM Resource Adapter



8. The New J2C Connection Factory dialog displays. Enter a new **JNDI Name** (for example, `PhoneBookJNDIName`).

> **What is JNDI?**
>
> The Java Naming and Directory Interface (JNDI) is an API for directory service that allows clients to discover and look up data and objects via a name. In this tutorial you will assign a unique JNDI name to your managed connection. Our J2C bean will then use this JNDI name to look up the connection on the WebSphere Application Server.

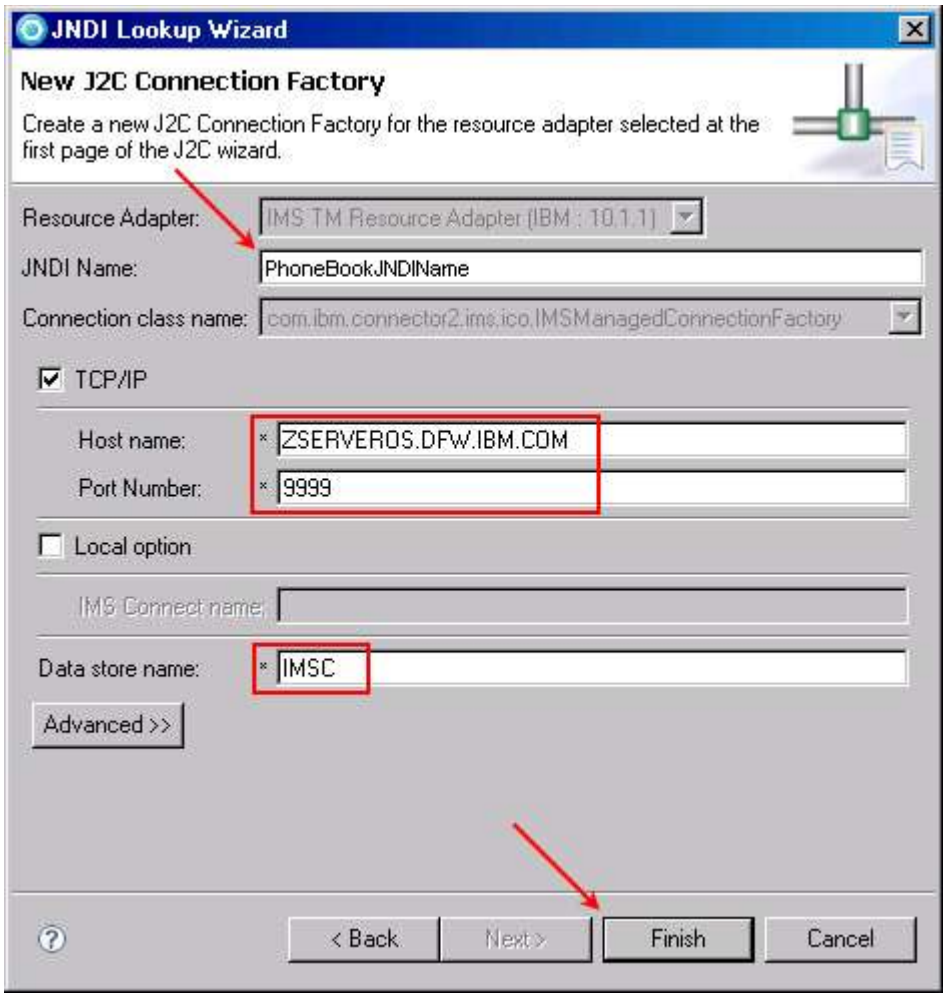
9. Make sure that **TCP/IP** is selected (the default) and enter the required connection information (indicated by the asterisk [*]), as shown in the following Figure:

   a. **Host name:** `ZSERVEROS.DFW.IBM.COM`

   b. **Port:** `9999`

   c. **Data store name:** `IMSC`

Figure 14. Specifying connection information

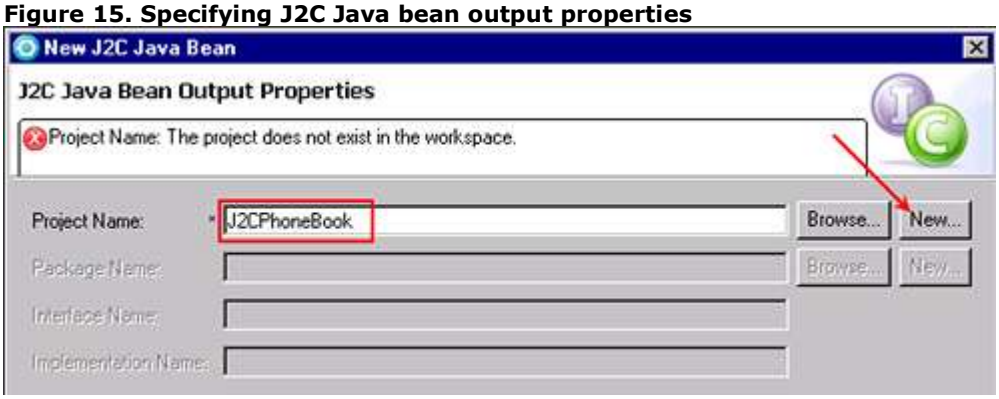


10. Click **Finish**.

11. On the Connection Properties screen, click **Next** to continue.

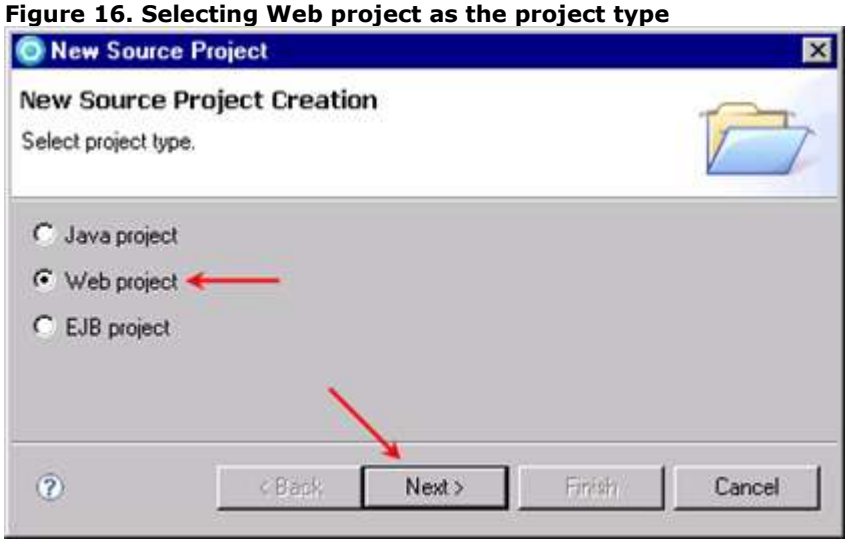
12. On the J2C Java Bean Output Properties page, enter `J2CPhoneBook` as the **Project Name** and click **New** to define the project properties, as shown in the following Figure.

Figure 15. Specifying J2C Java bean output properties



13 On the New Source Project Creation page, select **Web project** and click **Next** to continue, as shown in the following Figure.

Figure 16. Selecting Web project as the project type



Why Web project?
The J2C wizard gives you a choice between creating a Java, Web, or EJB project. Choose Web project because you will be creating a Web interface for your J2C bean in the form of a simple JSP component and as a Web Service (see optional Task 4).

Web projects hold all of the Web resources that you create and use to develop your Web applications.

14. On the New Dynamic Web Project page shown in the following Figure.

    a. In **Project contents**, leave **Use default** checked.

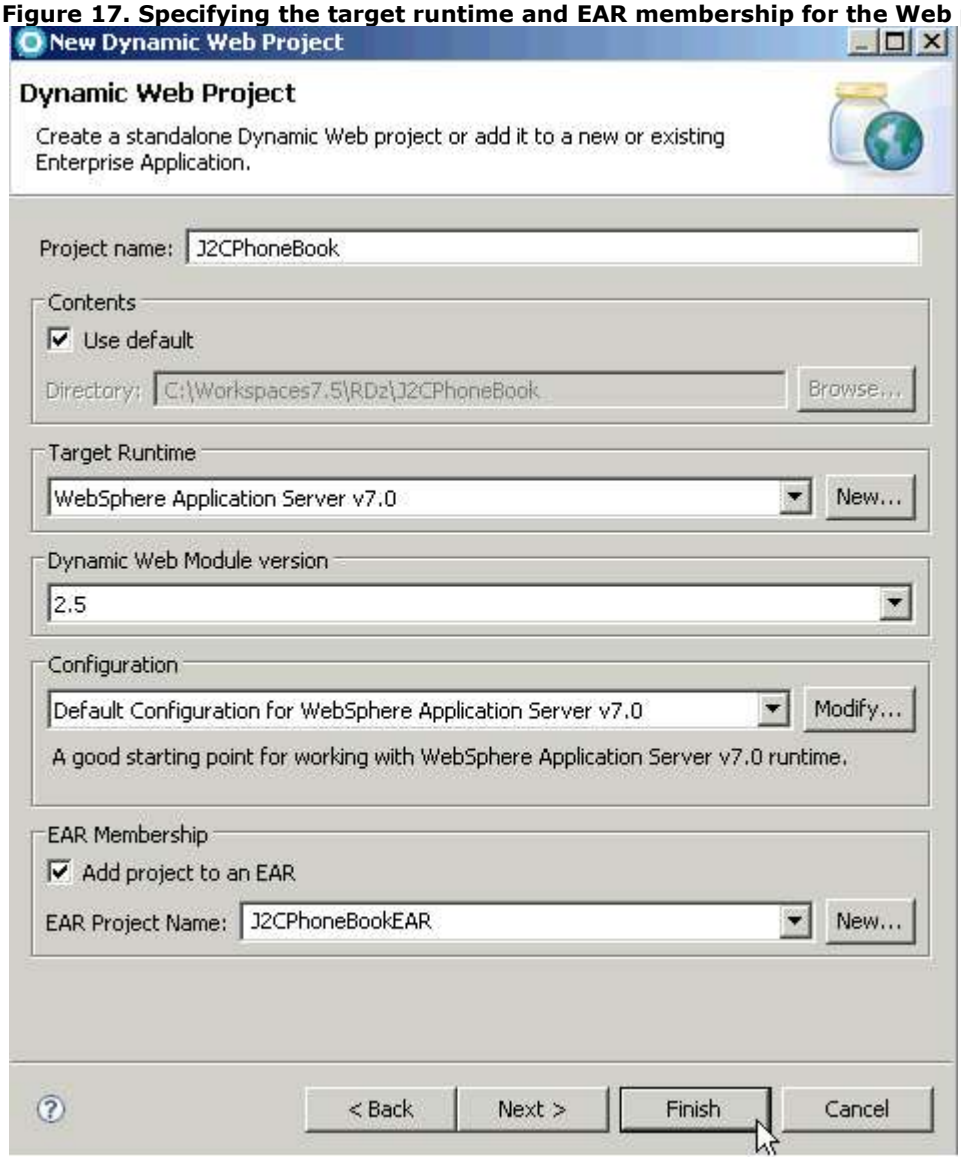
    b. For the **Target Runtime**, make sure **WebSphere Application Server v7.0 is selected** as the server.

    c. Leave the **Dynamic Web Module version** and **Configurations** settings as is.

    d. Select the **Add Project to an EAR** check box. Allow the wizard to add "EAR" to your EAR project name. Web projects and EAR projects must have different names.

    e. Click **Finish** to create the Dynamic Web Project.

Figure 17. Specifying the target runtime and EAR membership for the Web project



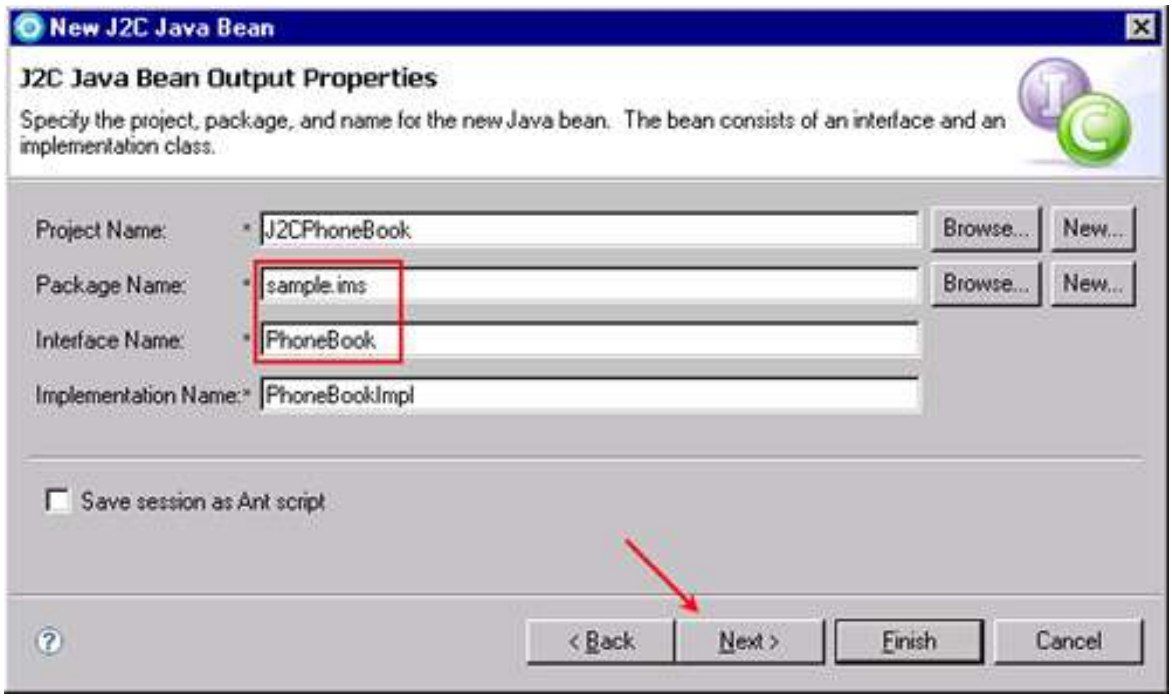
Rational Developer for System z will now complete the creation of the J2EE components that support the J2C bean. Notice that both a dynamic Web project and an EJB project have been added to your work space.
Also, now that the supporting projects are created, the J2C Java Bean wizard returns to define the J2C bean Output Properties.

15. On the J2C Java Bean Output Properties page, leave the project name as **J2CPhoneBook** . Enter the following required fields (respect the upper/lower case) as shown in the following Figure.

a. **Package Name:** `sample.ims`

b. **Interface Name:** `PhoneBook`
Notice that `PhoneBookImpl` in the **Implementation Name** field will be created for you once the **Interface Name** is supplied.

Figure 18. Specifying the package name and interface name

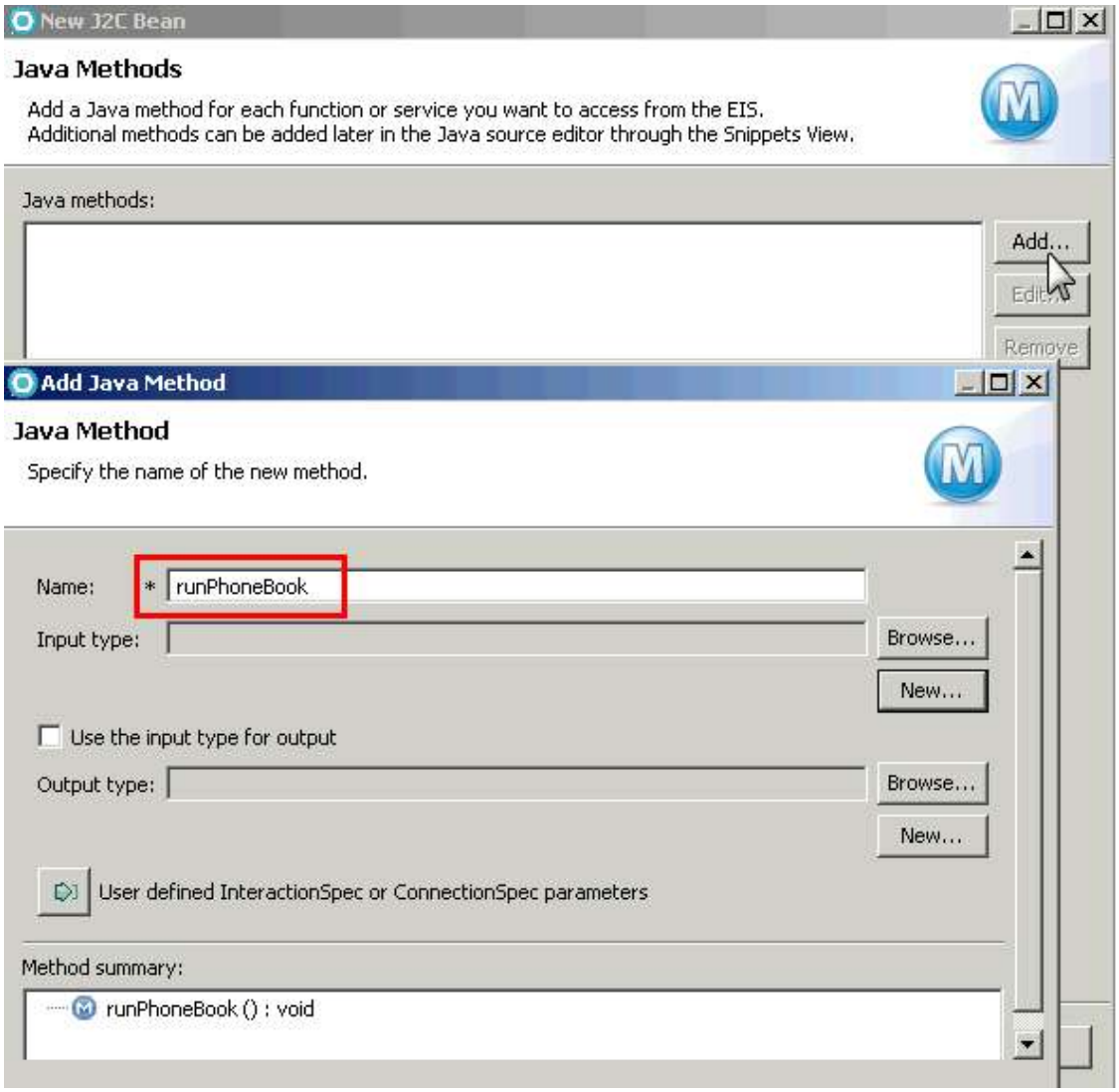


16. Click **Next** to continue.

17. From the Java Methods page, click the **Add** button to add a Java method to the sample.ims package defined in the previous page.

18. From the Add Java Methods page, enter **runPhoneBook** as the Java Method **Name**:

Figure 19. Adding a Java method

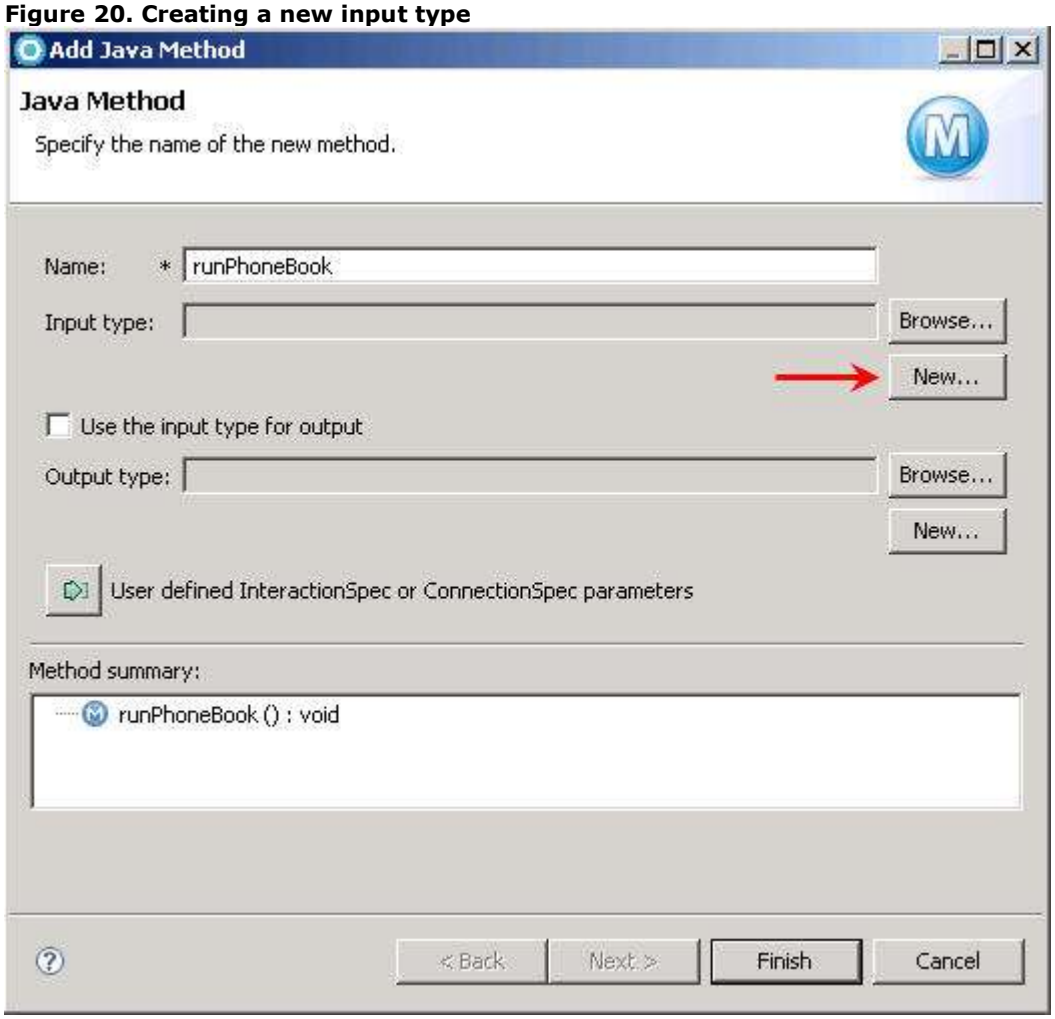


What's Next

You have just started creating a Java method that will provide translation between COBOL data that IMS can understand and Java data types suitable for Java EE applications. Next, you will import a COBOL copybook for the IVTNO transaction. The wizard will parse the copybook and identify input and output fields. It will then generate translation code that will provide the mapping between COBOL and Java. As elsewhere in this tutorial, all the work is done by the Rational Developer for System z tooling, and no manual coding is required.
Now you will create the input and output data mappings between COBOL and Java. In this step you will import the data definitions from the Ex01.cbl (COBOL) copybook.

20. Click **New** next to the **Input type** to create the input mappings for the `runPhoneBook` Java method as shown in the following Figure.

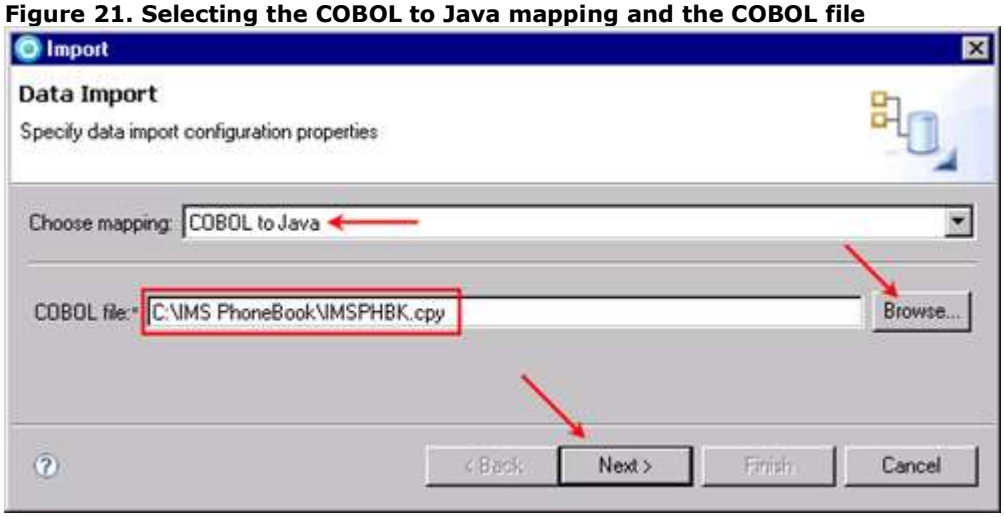
Figure 20. Creating a new input type



21. From the Data Import page, accept the default **COBOL to Java** mapping.

22 Click the **Browse** button and find the file **C:\IMS PhoneBook\IMSPHBK.cpy** as shown in the following Figure. Click **Open** to accept it:

Figure 21. Selecting the COBOL to Java mapping and the COBOL file



23. Click **Next** to continue.

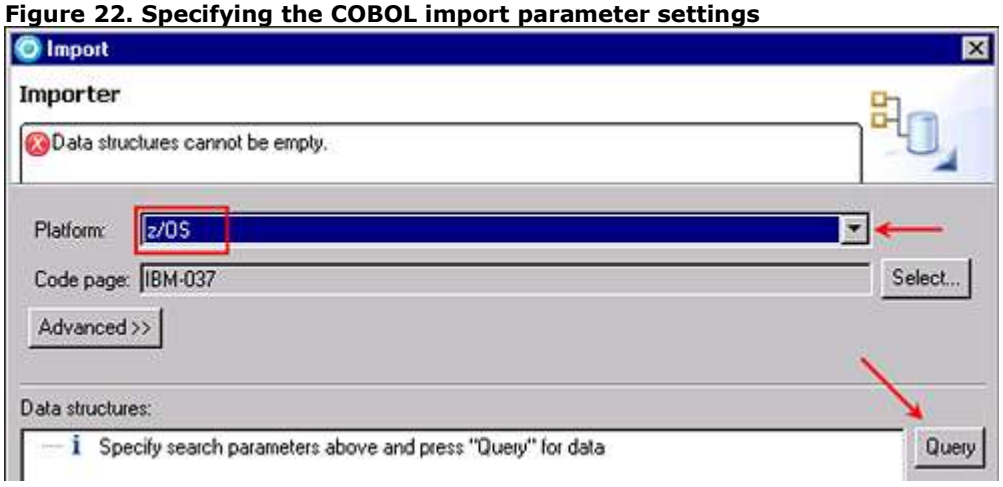
24. The Importer page opens, allowing you to define the appropriate COBOL Import Parameter settings. Default settings are for the Win32 platform.

    a. From the **Platform** list, select **z/OS**. The code page and other parameter settings will correctly change for the z/OS platform.

    b. You can press the **Advanced>>** button to observe the other parameter settings for COBOL. However, do not change any of these advanced settings for this tutorial.

    c. Click **Query** to select the appropriate input data from the IMSPHBK.cpy COBOL file as shown in the following Figure.

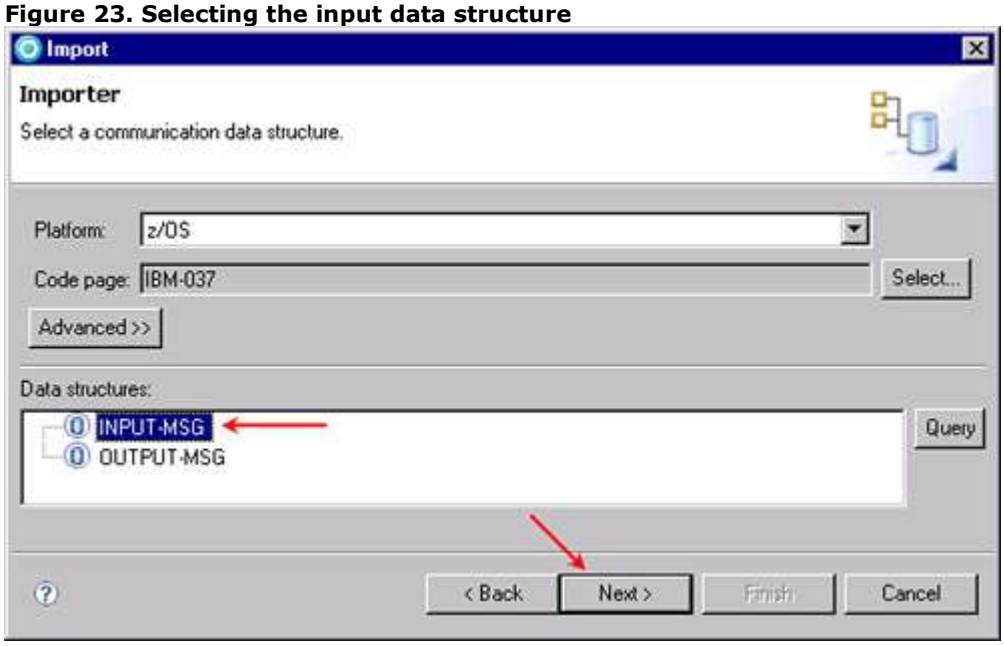
Figure 22. Specifying the COBOL import parameter settings



The COBOL program IMSPHBK.cpy contains the following **Data structures** in the Linkage Section: **INPUT-MSG** and **OUTPUT-MSG**. Notice that Rational Developer for System z inspected the IMSPHBK.cpy COBOL program and provided the data areas to map to the input record.

d. Select the **INPUT-MSG** data structure to map this data area to the input record, as shown in the following Figure, and then click **Next** to continue:

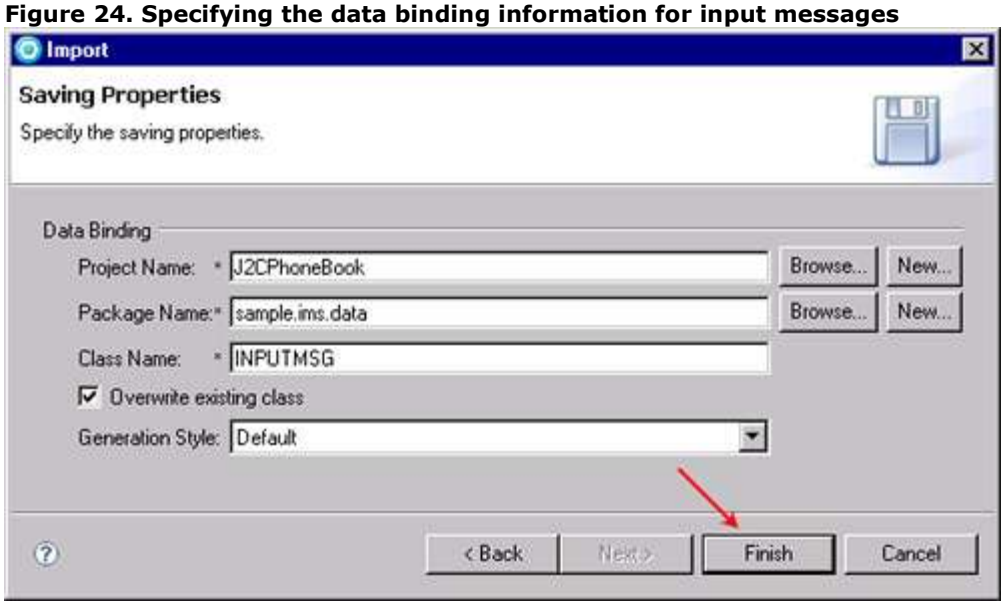
Figure 23. Selecting the input data structure



25. The Import page displays. Notice that you can change the generation style and data bindings. Accept the default generation style and data bindings.

26. Click **Finish** to generate the INPUTMSG Java class for the input mapping:

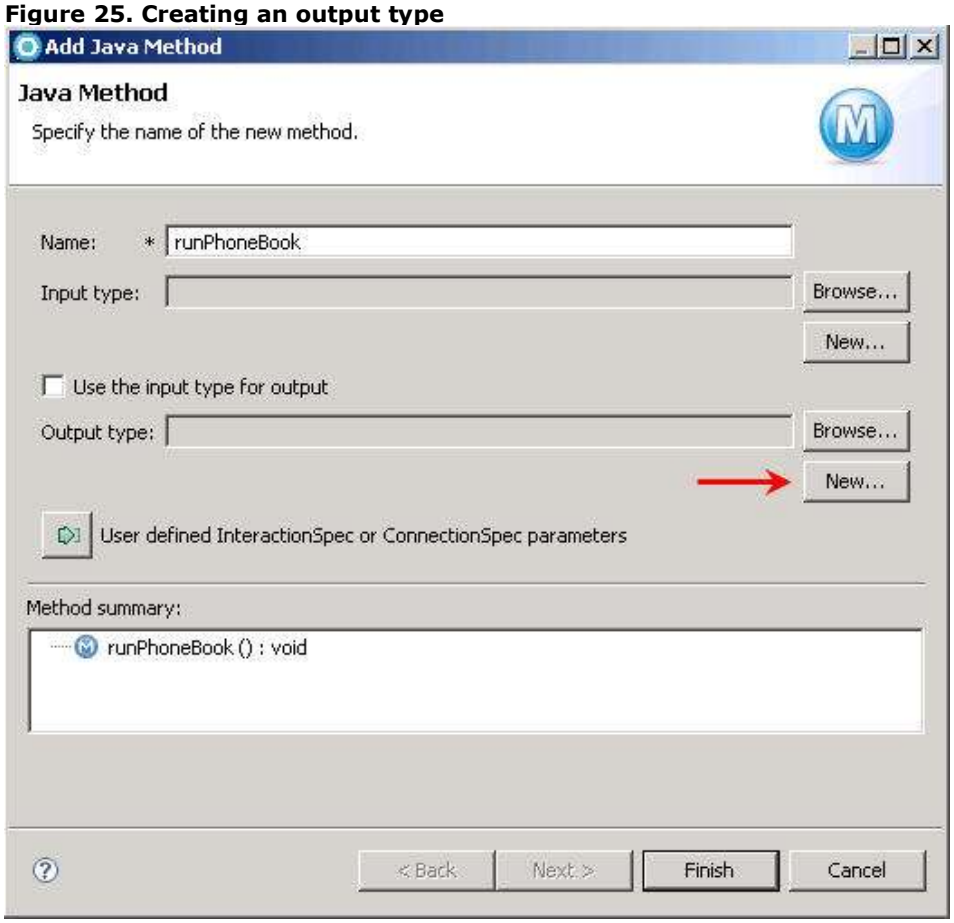
Figure 24. Specifying the data binding information for input messages



The Add Java Method page redisplays. Notice that the **Input type** has been defined with the INPUTMSG Java class.

27 Click **New** to define the output type, as shown in the following Figure.

Figure 25. Creating an output type



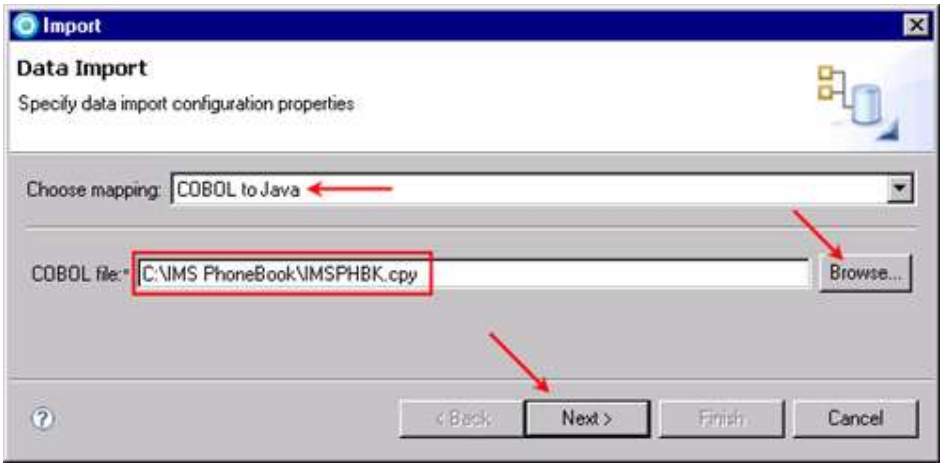
You must now create the **Output type**, repeating similar actions that you already performed for the Input type.

28. From the Data Import page, accept the default COBOL to Java mapping.

29. Click **Browse** and select the C:\IMS PhoneBook\IMSPHBK.cpy file again, as shown in the following Figure.

30. Click **Open** to accept it.

Figure 26. Selecting the COBOL to Java mapping and the COBOL file



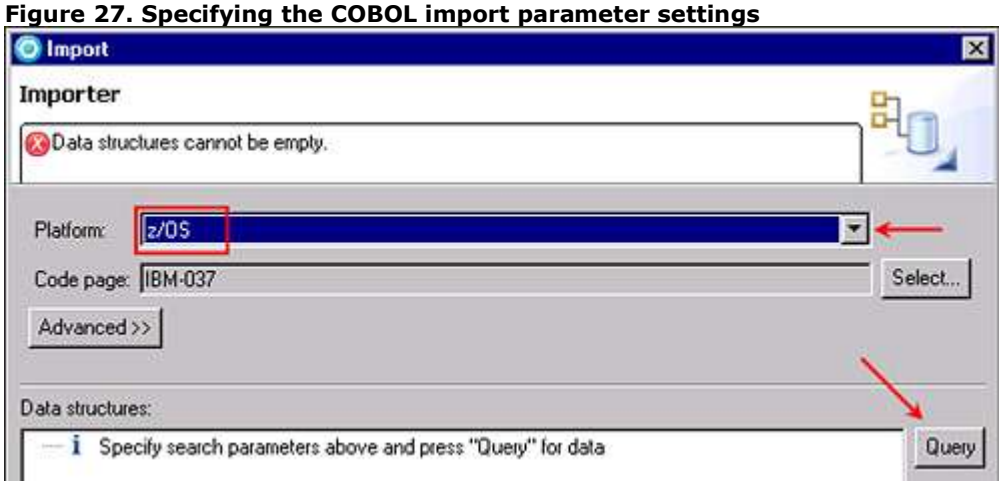
31. Click **Next** to continue.

32. The Importer page opens, in which you define the appropriate COBOL Import Parameter settings, as shown in the following Figure. . Default settings are for the Win32 platform.

a. Select **z/OS** as the platform. The code page and other parameter settings will correctly change for the z/OS platform.

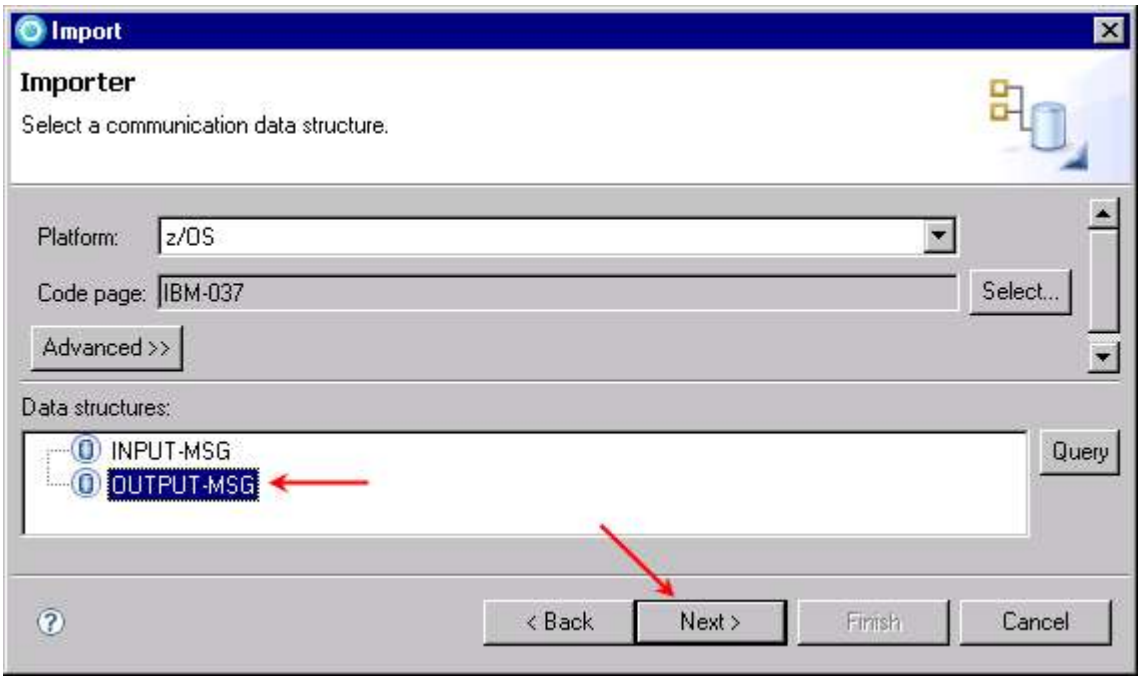
b. Click **Query** to select the appropriate input data from the IMSPHBK.cpy COBOL file:

Figure 27. Specifying the COBOL import parameter settings



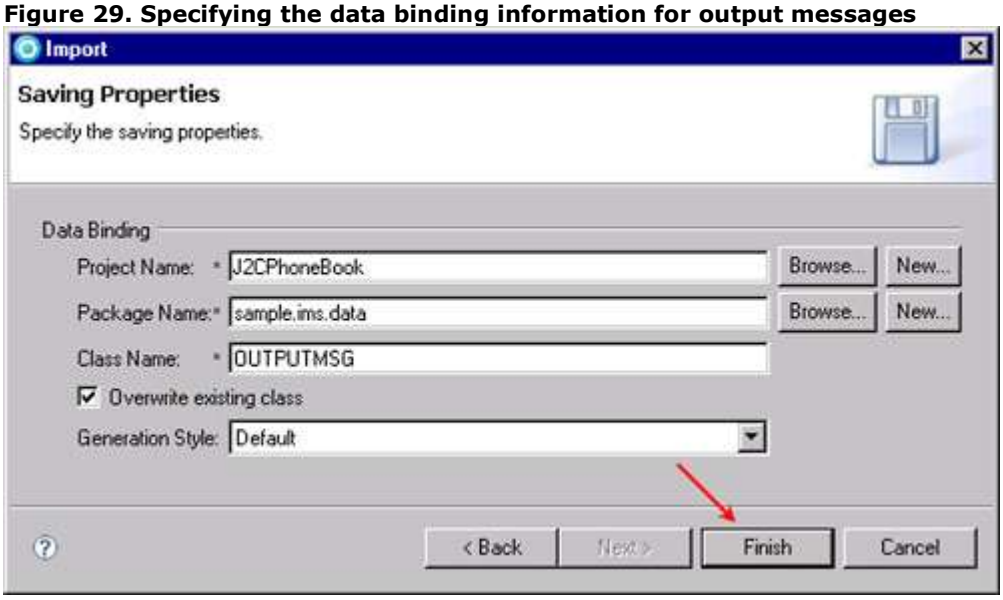
33. Select the **OUTPUT-MSG** as the data structure for the output type, as shown in the following Figure, and then click **Next** to continue:

Figure 28. Selecting the output data structure



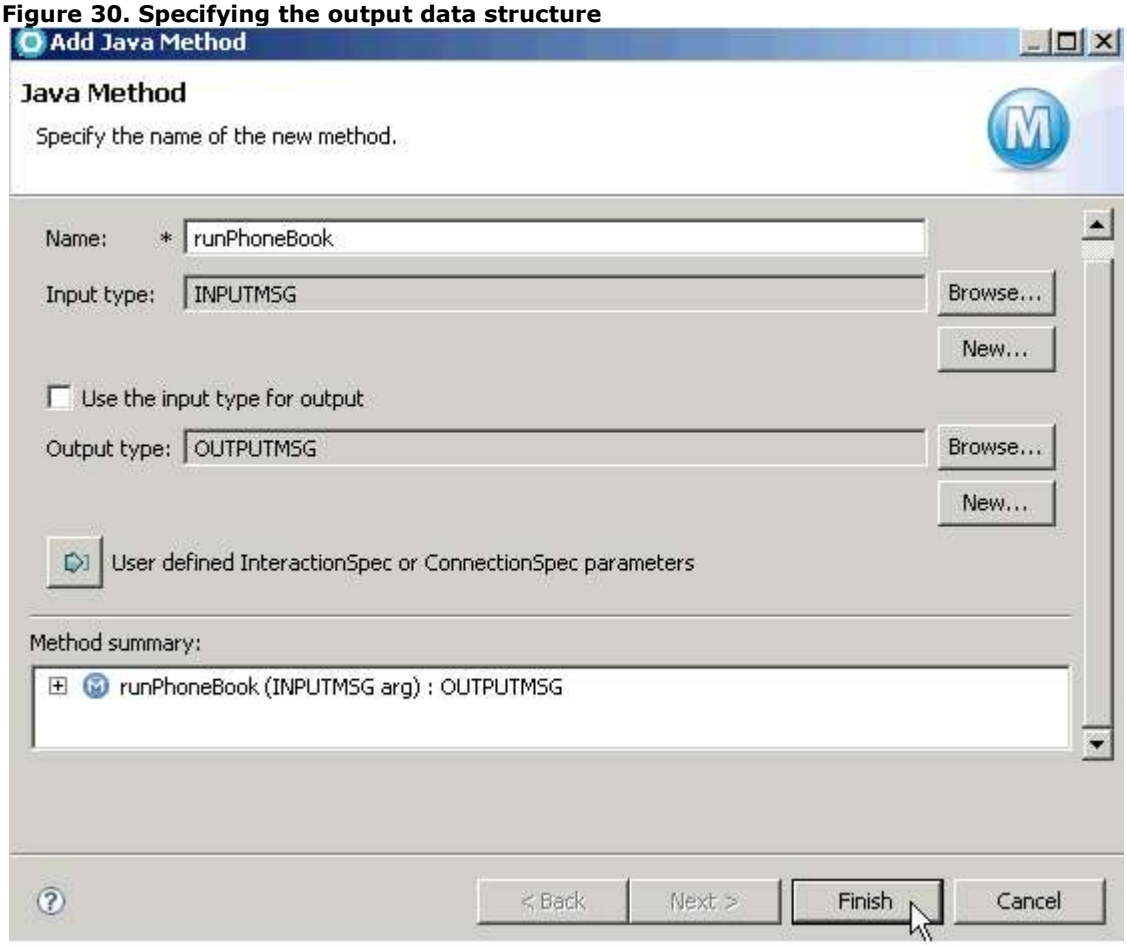
34. Accept the default generation style and data bindings. Click **Finish** to generate the OUTPUTMSG Java class, as shown in the following Figure.

Figure 29. Specifying the data binding information for output messages



35. The Add Java Method page should now be defined with an Input type: INPUTMSG and an Output type: OUTPUTMSG, as shown in the following Figure.

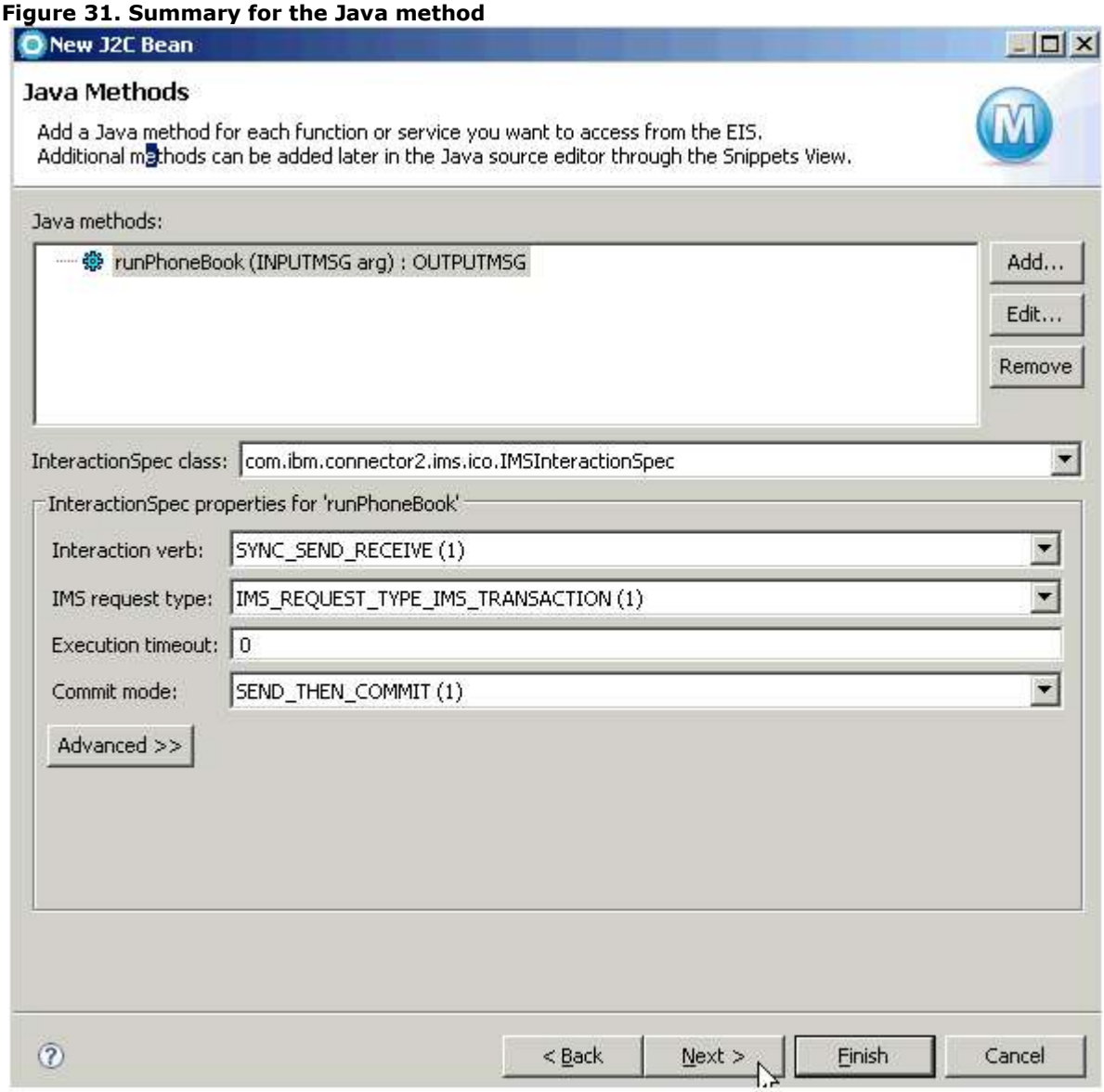
Figure 30. Specifying the output data structure



36. Click **Finish** to continue to the Java methods summary page.

37. The Java method summary page should look like that shown in the following Figure.

Figure 31. Summary for the Java method



38. Click **Next** to complete the Java method creation.

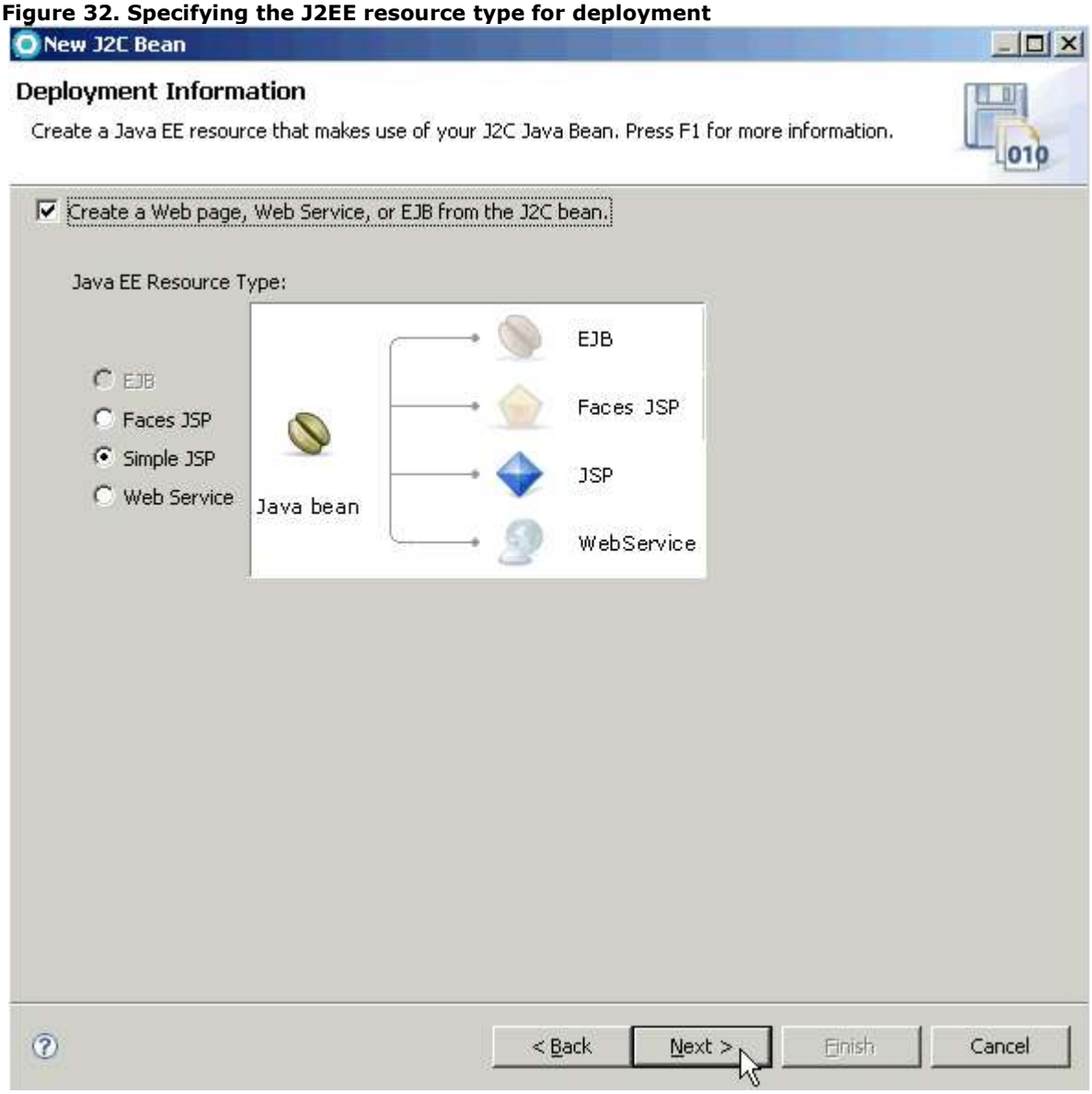
39. The Deployment Information page displays, as shown in the following Figure. From here you could complete the J2C Java bean generation. However, Rational Developer for System z provides additional generation capabilities (Web page, Web service, or EJB) to consume this J2C Java bean.

a. Select the **Create a Web Page, Web Service, or EJB from the J2C bean** check box. More options will become available, as shown in the following Figure.

b. Select **Simple JSP**.

c. Click **Next** to continue:

Figure 32. Specifying the J2EE resource type for deployment



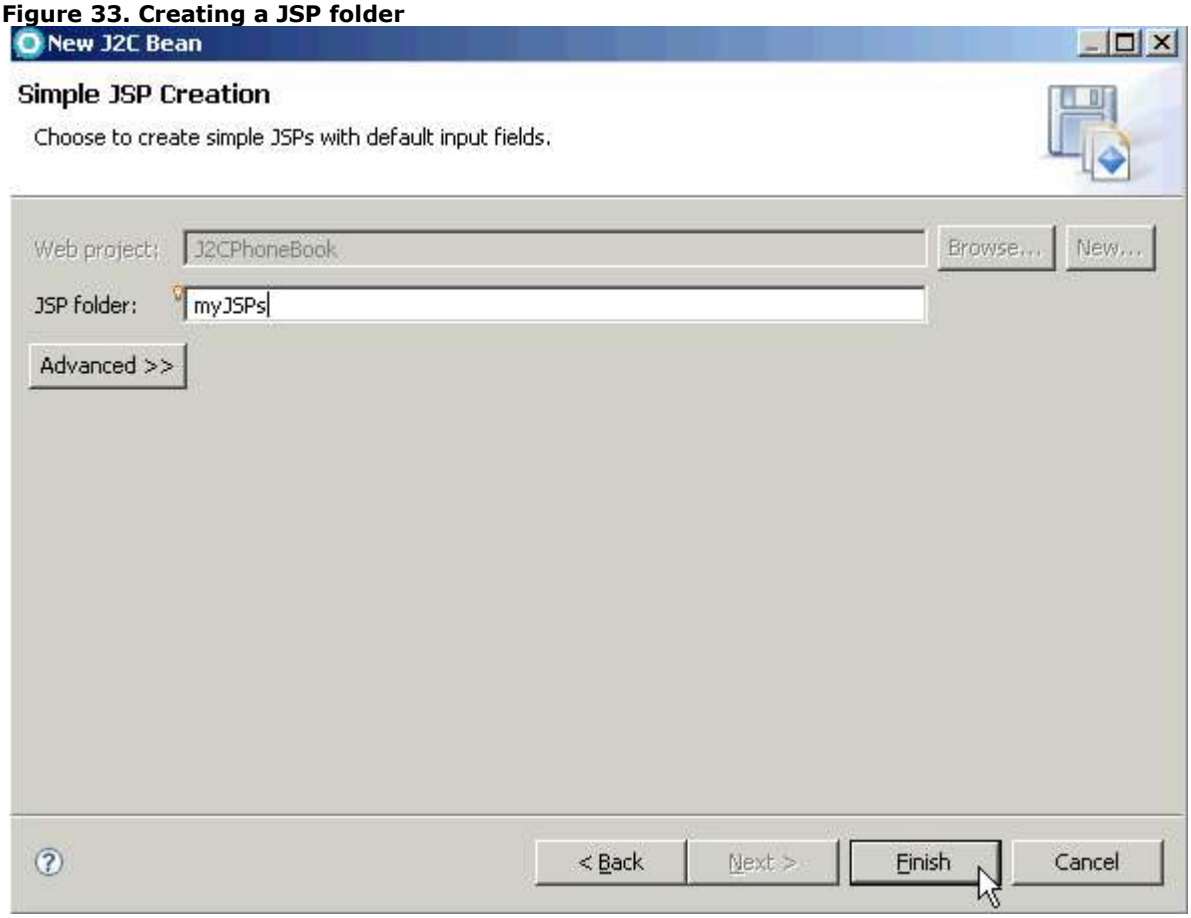
The **JavaServer Pages (JSP)** technology enables you to generate dynamic web content (such as HTML, DHTML, XHTML, and XML files) to include in a Web application. JSP files are one way that the product implements server-side

dynamic page content. JSP files allow a Web server, such as WebSphere Application Server, to add content dynamically to your HTML pages before they are sent to a requesting browser.

EJB (Enterprise JavaBeans™) and **Web Services** are other powerful architectures that can interface with your J2C bean. Web services are covered in the optional Task 4 of this tutorial.

40. From the Simple JSP Creation dialog, enter `myJSPs` for the **JSP folder** name and click **Finish** to complete the simple JSP, as shown in the following Figure.

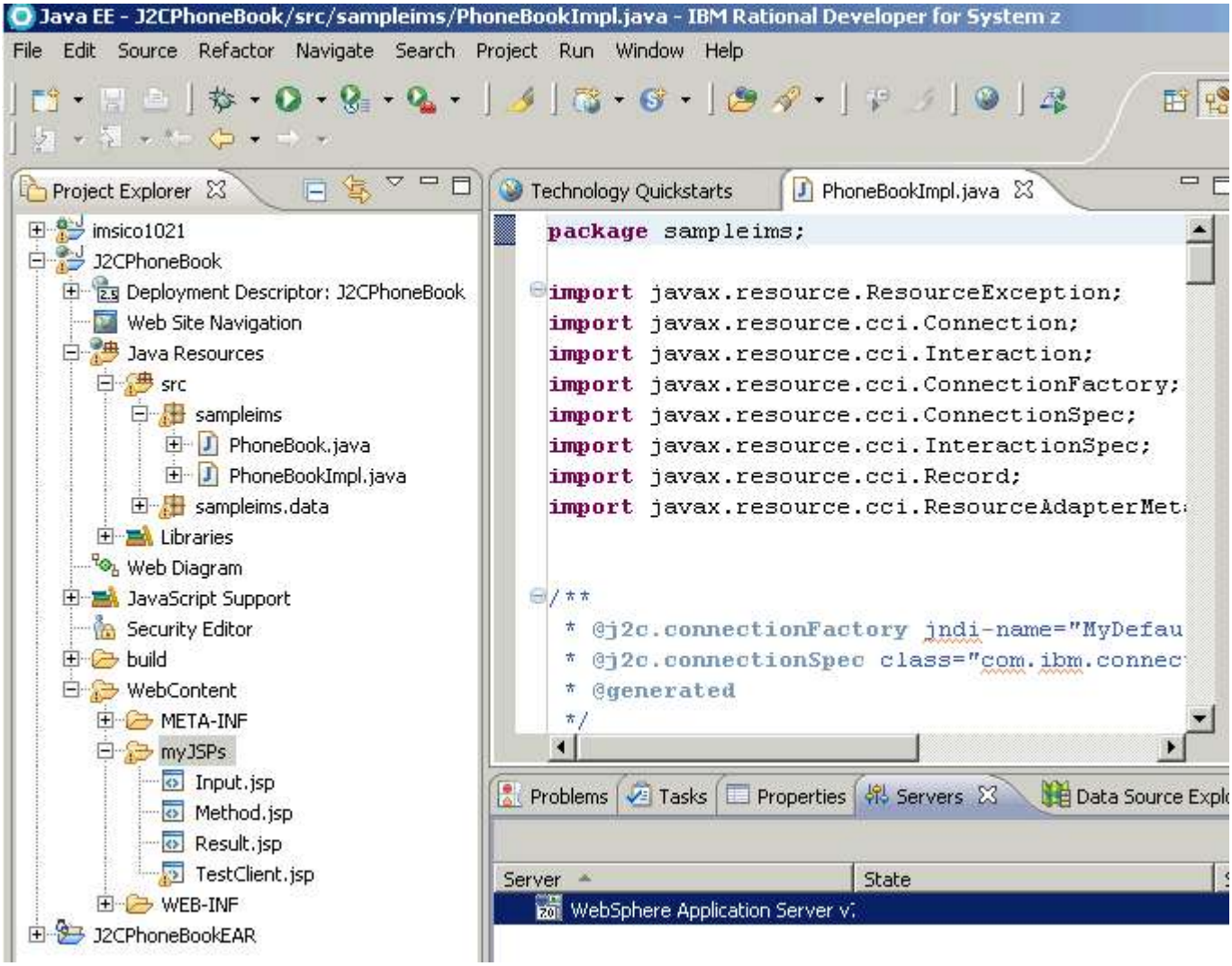
Figure 33. Creating a JSP folder



Notice that after clicking **Finish**, the Rational Developer for System z workspace opens up the **PhoneBookImpl.java** file. This is the implementation of the mapping between COBOL and Java. It contains the `runPhoneBook()` method, as well as other generated methods. It also contains the reference to the JNDI name `PhoneBookJNDIName`, which is used to look up the Managed Connection factory that we created. The connection information (along with the input data) is passed to the IMS TM Resource Adapter, which in turn calls IMS.

41. Expand **Java Resources: src > sample.ims** under the J2CPhoneBook project, as shown in the following Figure. Take a moment and look at the generated components in the Project Explorer. The **myJSPs** folder contains the newly generated JSPs that will be used to test our J2C bean implementation.

Figure 34. J2CPhoneBook project in the Project Explorer and the PhoneBookImp.java file



Task 3: Deploy and test your application

In this section, you will start the WebSphere Application Server, add your project to the application server runtime environment, and test your application using the simple JSP client that was created as part of the J2C Java Bean wizard.

Deploying your application

1. Select the **Servers** view within the J2EE perspective.

2. Using the **Servers** view, select the **WebSphere Application Server v7.0** and click the green arrow icon to start the server, as shown in the following Figure. It will take a few moments for the application server to start:

Figure 35. Starting the WebSphere Application Server

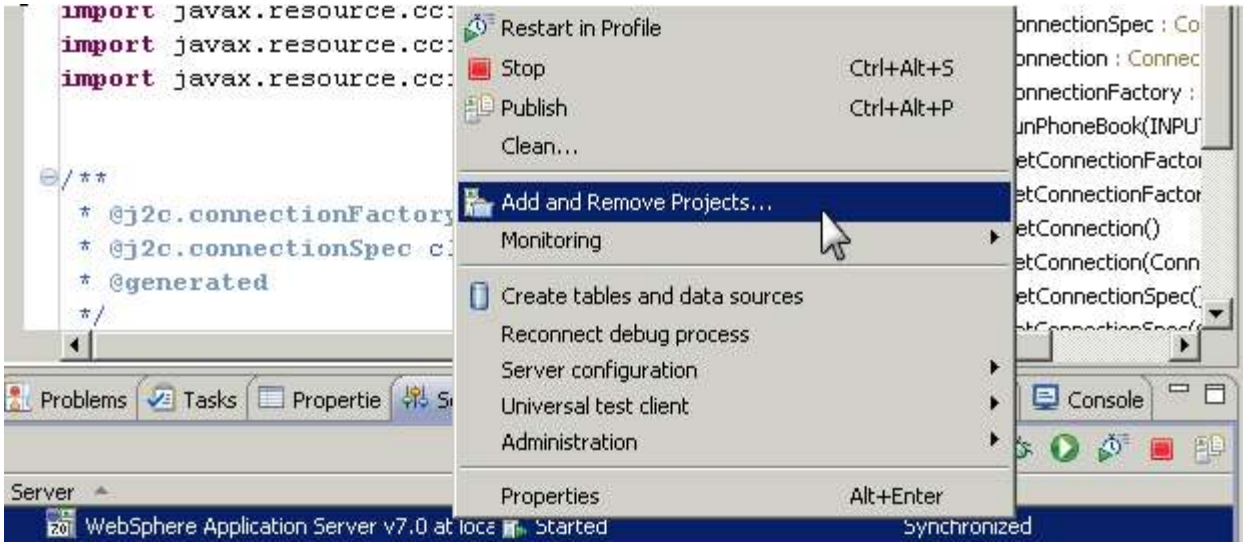
3. After the Console displays the "Server server1 open for e-business" message, click the **Servers** view and check the Status indicator.

4. When the WebSphere Application Server is started, the **Servers** view will display **Started** in the **Status** field, and **Synchronized** in the **State** field, as shown in the following Figure.

Figure 36. Status and State of the WebSphere Application Server

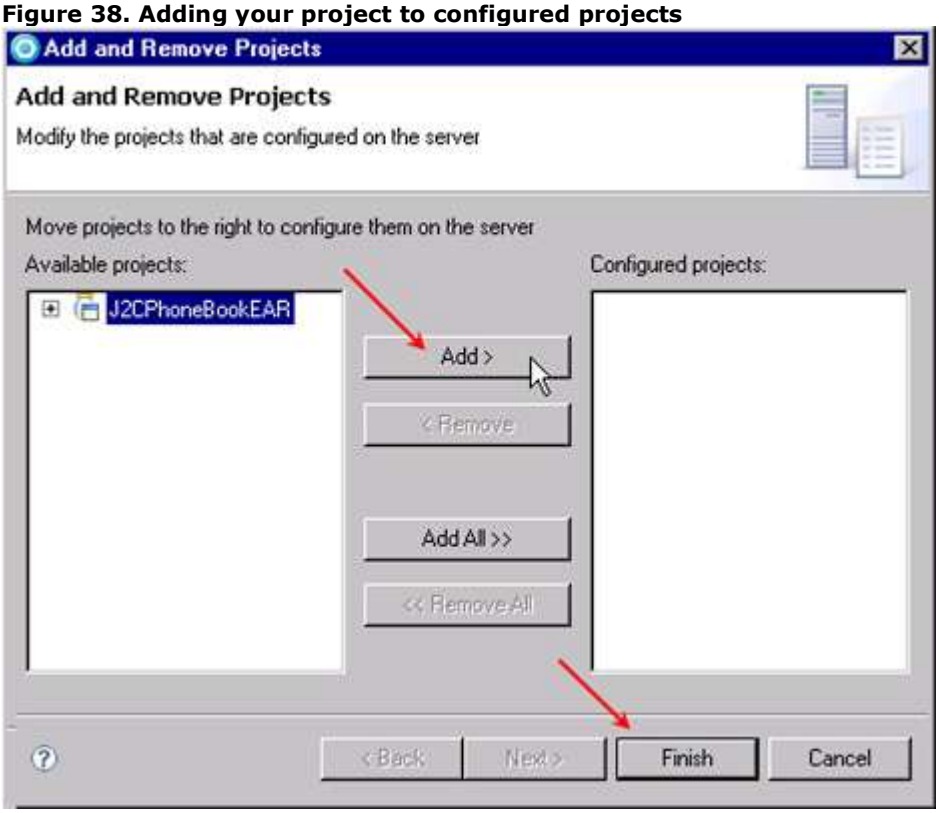
5. You now must add your project to the started application server. Using the **Servers** view, right-click **WebSphere Application Server v7.0** and select **Add and Remove Projects**, as shown in the following Figure.

Figure 37. Adding your project to the started application server



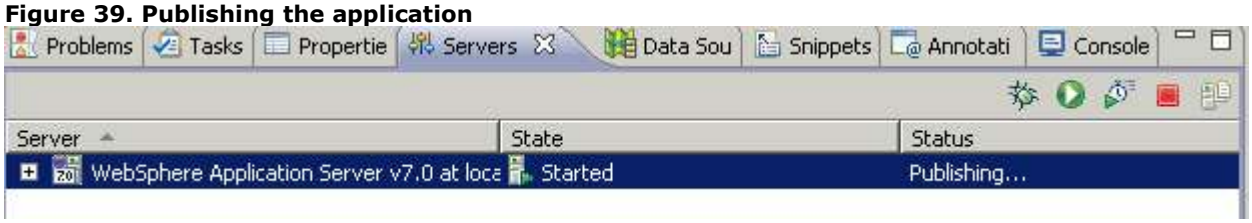
6. From the Add and Remove Projects page, select **J2CPhoneBookEAR** and click **Add >** to add your project to the Configured Projects, as shown in the following Figure.

Figure 38. Adding your project to configured projects



7. Click **Finish** to add your project to application server. The WebSphere Application Server will publish the application, as shown in the following Figure.

Figure 39. Publishing the application

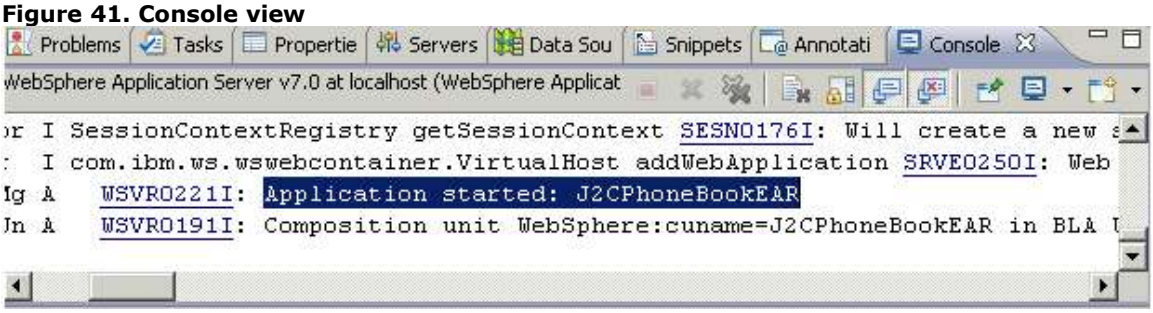


8. WebSphere Application Server then returns to a **Synchronized** state, as shown in the following Figure.

Figure 40. Returning to synchronized state after publishing the application

9. If not switched automatically, switch to the **Console** view and verify that the application has started successfully, as shown in the following Figure.

Figure 41. Console view



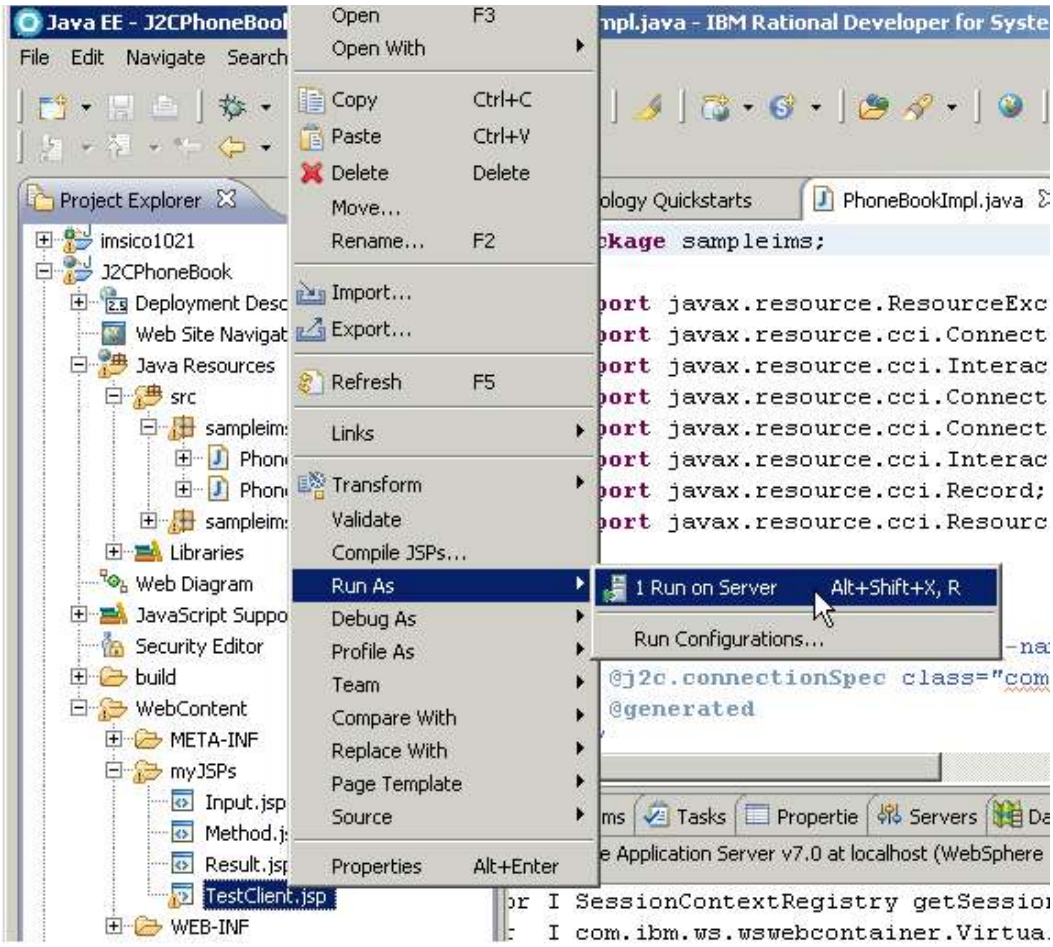
Testing your application

In this section you will test the simple JSP client that was created as part of the J2C Java Bean wizard.

1. Select the generated **TestClient.JSP** file that is located in the *myJSPs* folder. Recall that this is the folder you created to store the simple JSP test client. If the **TestClient.JSP** file is not visible, navigate to the **J2CPhoneBook > WebContent > myJSPs** folder in the Project Explorer.

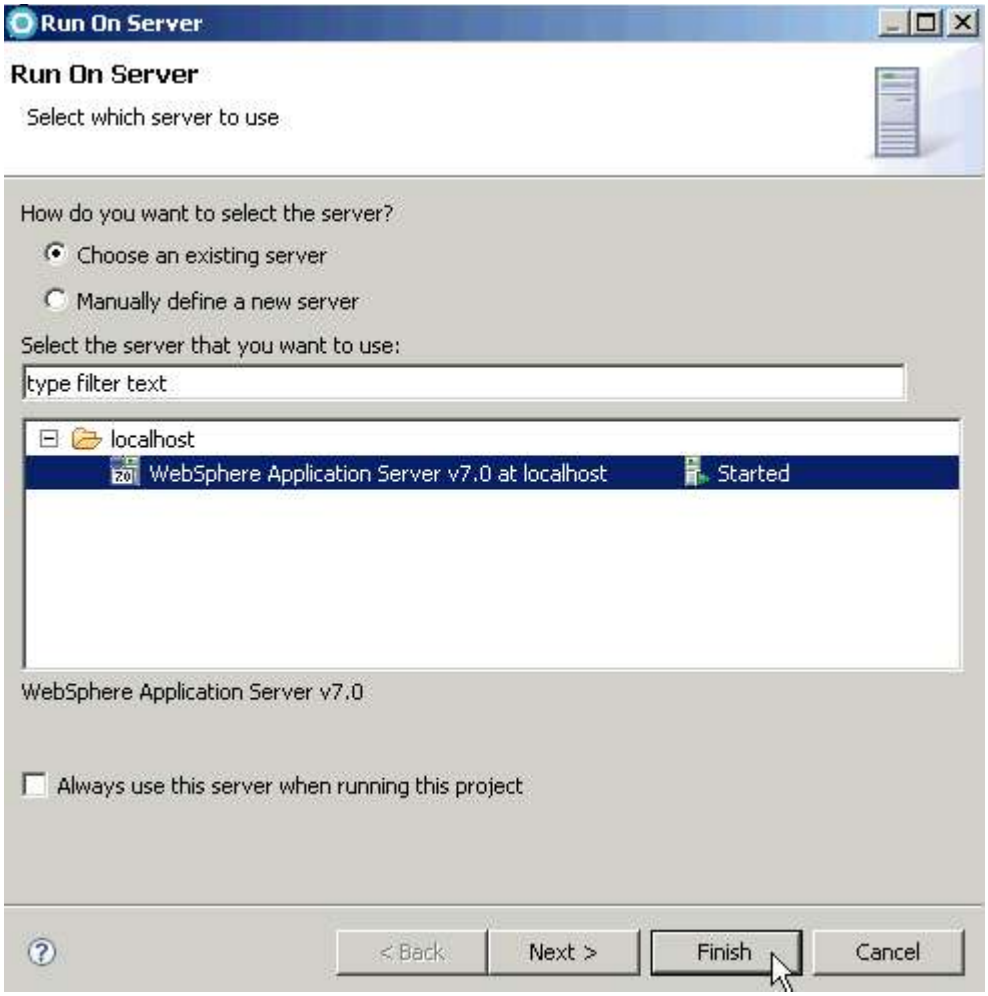
2. With the **TestClient.jsp** selected, Right-click and select **Run As > Run On Server**, as shown in the following Figure.

Figure 42. Running the TestClient.jsp



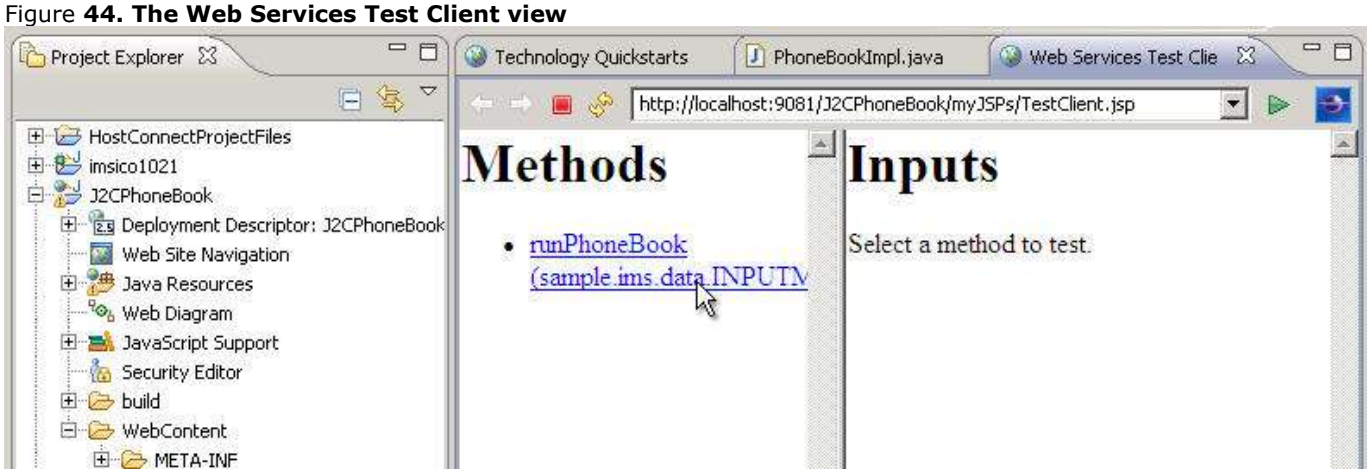
3. Choose the started WebSphere Application Server v7.0 and click **Finish** to run the TestClient.jsp, as shown in the following Figure.

Figure 43. Choosing the server to run the JSP



4. The Web Services Test Client will launch the TestClient.jsp. Notice that there are three separate panes (**Java Methods**, **Input parameters**, and **Results**) within the TestClient.jsp, as shown in the following Figure.

Figure **44. The Web Services Test Client view**



5. Double-click the **Web Services Test Client** title to maximize the window within the workbench, and then click the `runPhoneBook` method to test.

6. Enter the following values to test the J2C bean, as shown in the following Figure.

    a. **in__ll:** `59`

    b. **in__trcd:** `IVTNO`

    c. **in__zz:** `0`

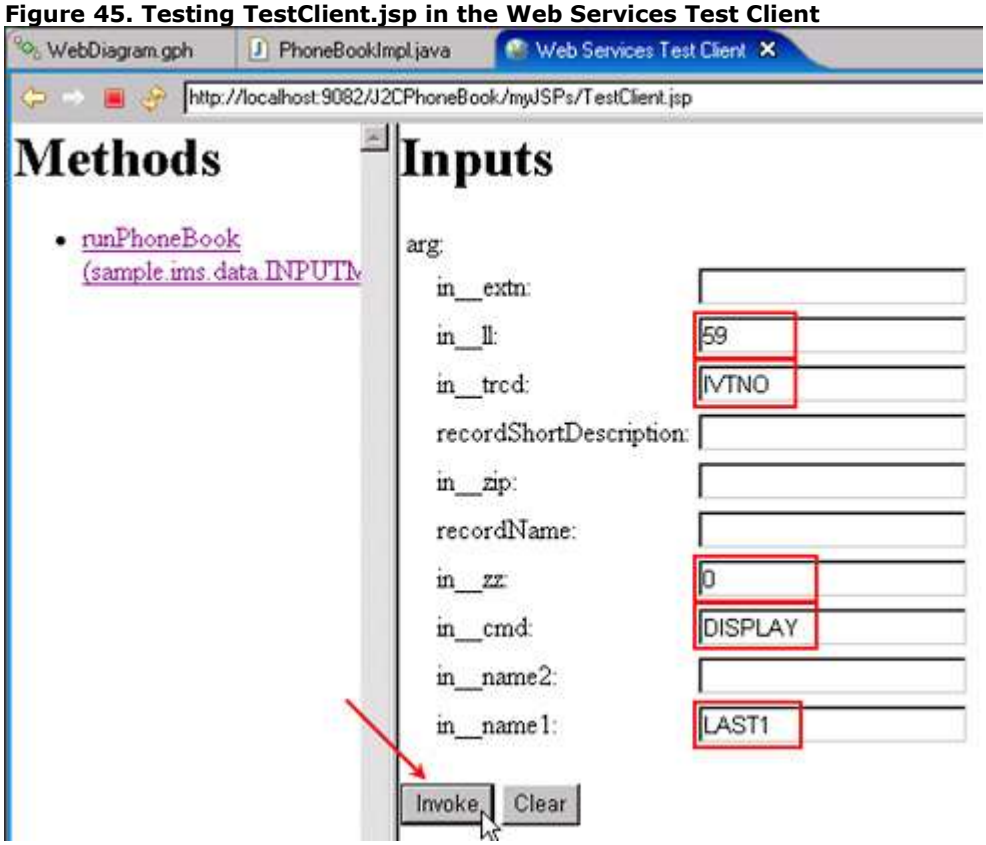
    d. **in__cmd:** `DISPLAY`

    e. **in__name1:** `LAST1`

7. Make sure not to enter any spaces before or after any of the input fields.

8. Click **Invoke** to run the application.

Figure 45. Testing TestClient.jsp in the Web Services Test Client

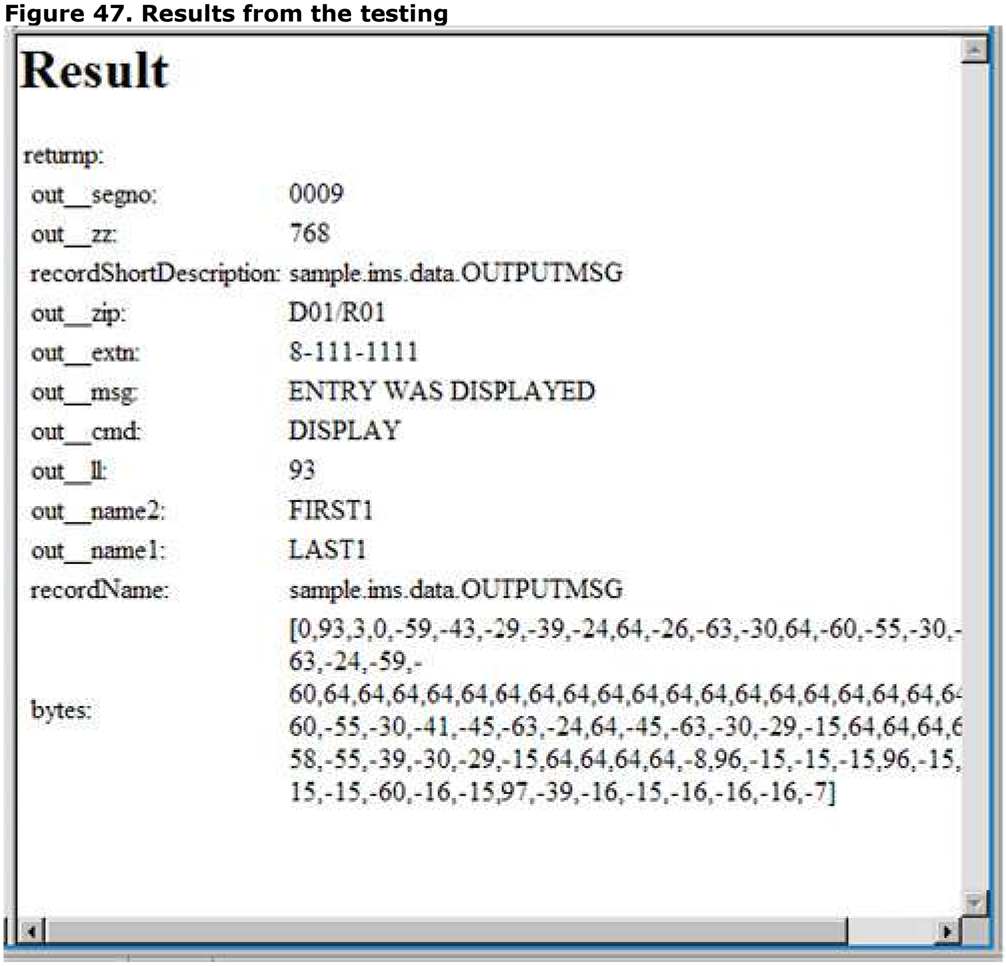


9. Resize the result area to check your results, as shown in the following Figure.

Figure 46. Resizing the result area

You should see something similar to the results shown in the following Figure.

Figure 47. Results from the testing



10. Select **Windows** > **Reset** perspective and **OK**.

11. Using the **Servers** tab, click the red square icon to stop the WebSphere Application Server, as shown in the following Figure.

Figure 48. Stopping the WebSphere Application Server

12. Close all open editor windows (pressing Ctrl+Shift+F4 should accomplish this).

Congratulations! You have completed the IMS J2C tutorial!

If you have extra time, you can do a bit more. How about taking the generated J2C bean and wrapping that as a Web service? This feature extends the usage of your J2C bean past simple Web clients and offers the transaction as a Web service.

Task 4 (Optional): Create a Web service to invoke the J2C bean

In the previous sections, you installed a J2C Resource Adapter, created a J2C bean that executed against the IMS sample transaction: IVTNO. You also created a simple JSP Test Client that tested your J2C bean. In this optional section, you will wrap the J2C bean as a Web service, and then test it by using the generated WSDL file. You will then generate a Web service client and test it as well.

Creating a Web service

1. Click **File > New > Other** to open the Select a Wizard dialog, as shown in the following Figure.

Figure 49. Starting the J2C Java Bean wizard

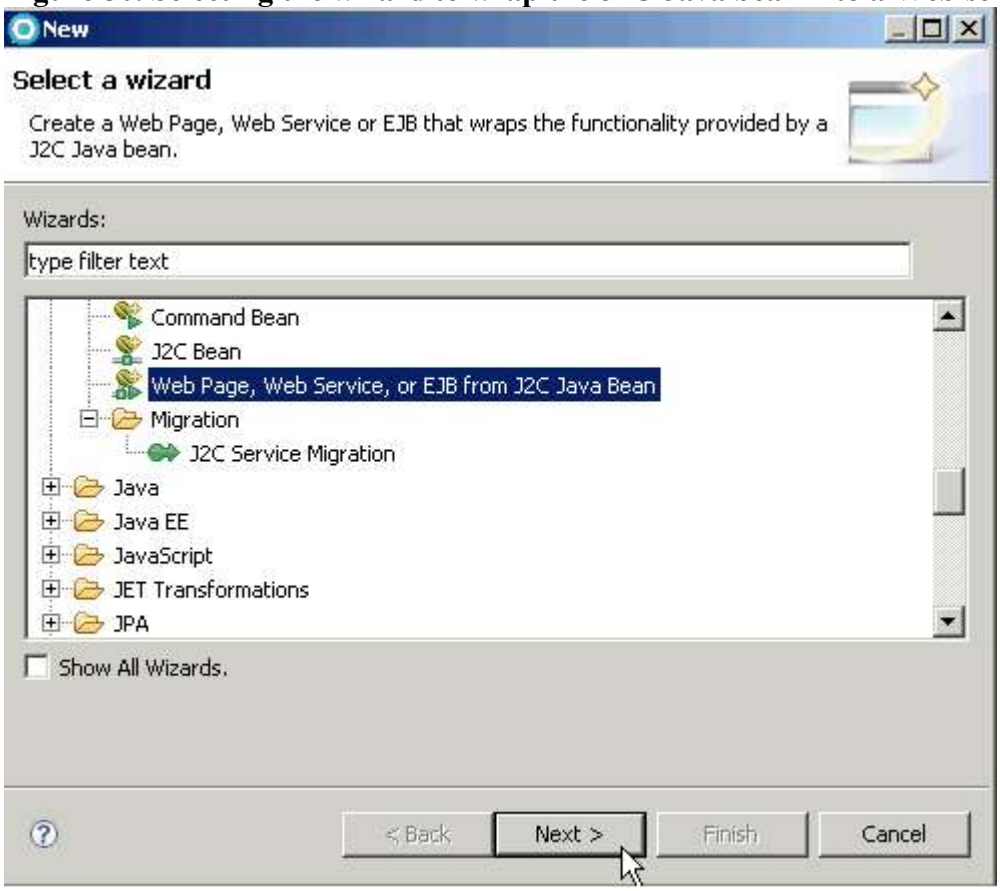
What is a Web service?

Web services are self-contained, self-describing modular applications that can be published, located, and invoked across the Web. Businesses can dynamically mix and match Web services to perform complex transactions with minimal programming. Web services allow buyers and sellers all over the world to discover each other, connect dynamically, and execute transactions in real time with minimal human interaction.

The following standards play key roles in Web services: **Web Services Description Language (WSDL)** and **Simple Object Access protocol (SOAP).**

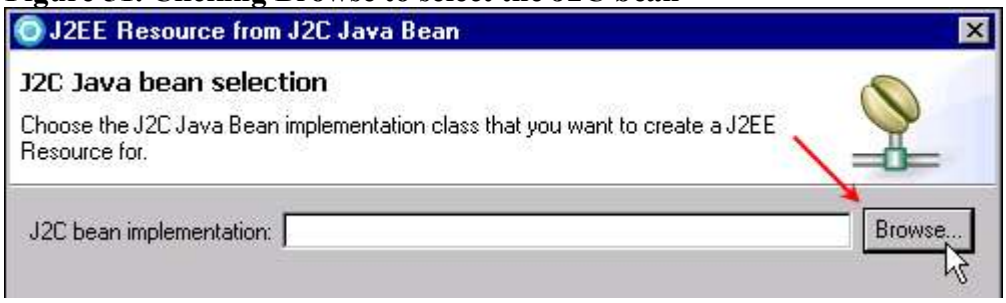
4. In the New wizard screen, select **Web Page, Web Service, or EJB from J2C Java Bean** and click **Next**, as shown in the following Figure.

Figure 50. Selecting the wizard to wrap the J2C Java bean into a Web service



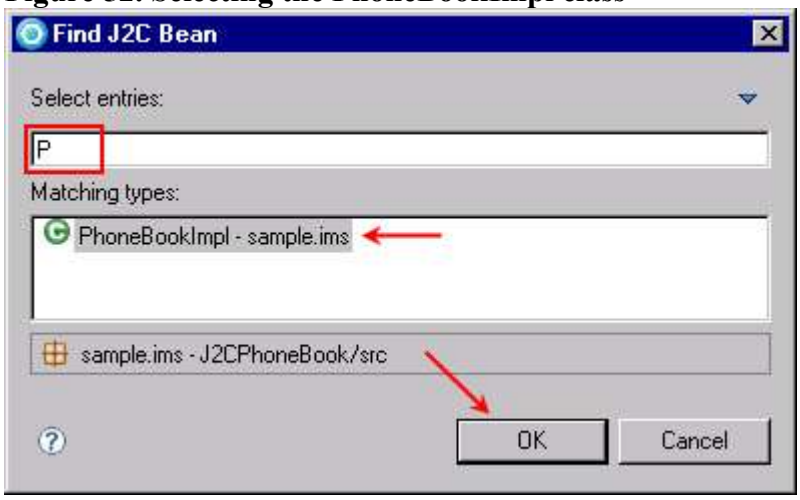
5. Press **Browse** to look up the J2C bean Implementation class, as shown in the following Figure.

Figure 51. Clicking Browse to select the J2C bean



6. Type `P`, select the **PhoneBookImpl** class in package **sample.ims** and click **OK**, as shown in the following Figure.

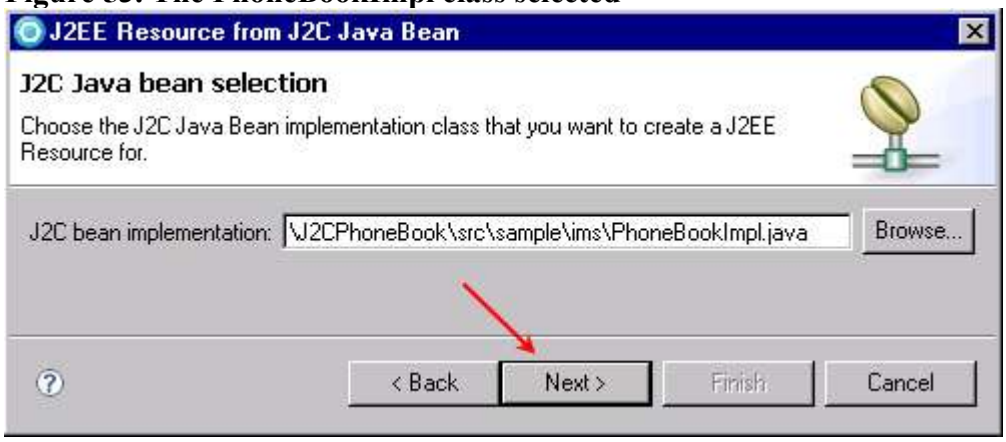
Figure 52. Selecting the PhoneBookImpl class



7. This returns the fully-qualified implementation class in the dialog, as shown in the following Figure.

8. Press **Next** to continue.

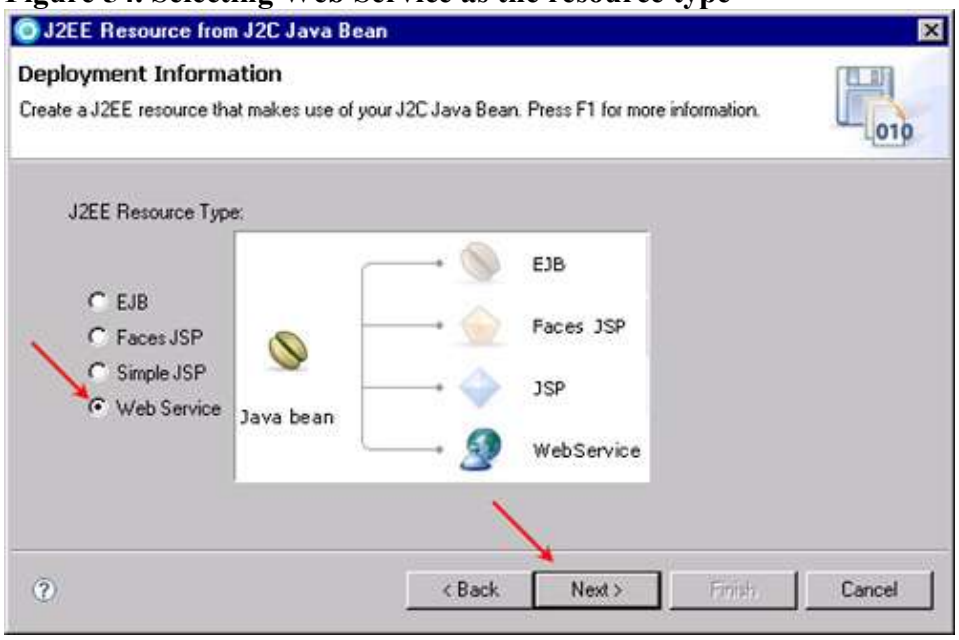
Figure 53. The PhoneBookImpl class selected



9. Select **Web Service** as the J2EE Resource Type, as shown in the following Figure.

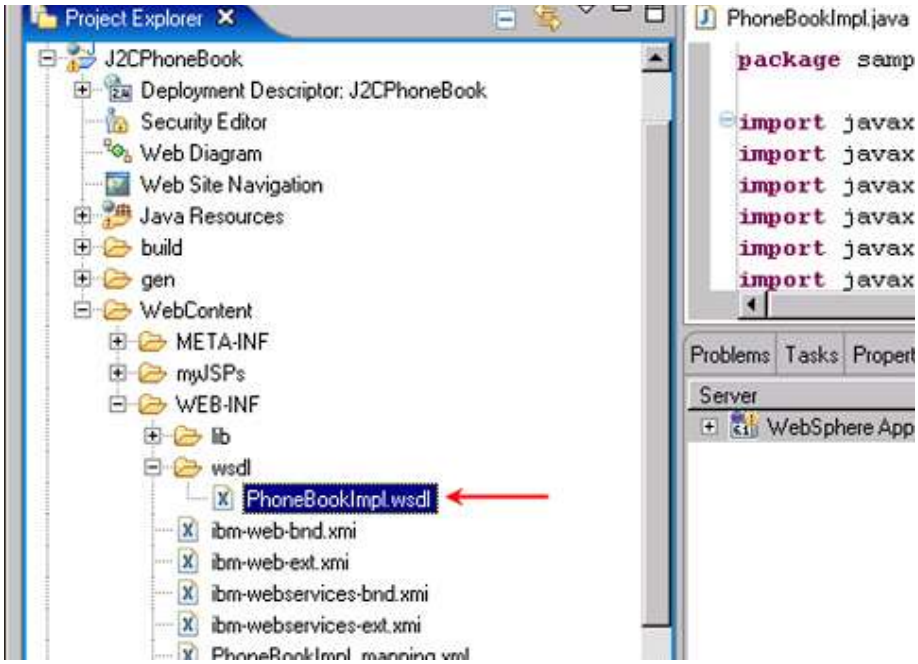
10. Click **Next,** and then **Finish**. The J2EE Resource from a J2C Java Bean wizard will now create the necessary J2EE resources for the Web service implementation. This might take a few moments.

Figure 54. Selecting Web Service as the resource type



11. Examine the generated J2EE resources for Web services. Expand **J2CPhoneBook > WebContent > WEB-INF > wsdl.** Note the generated **PhoneBookImpl.wsdl** file and associated xml mapping files, as shown in the following Figure.

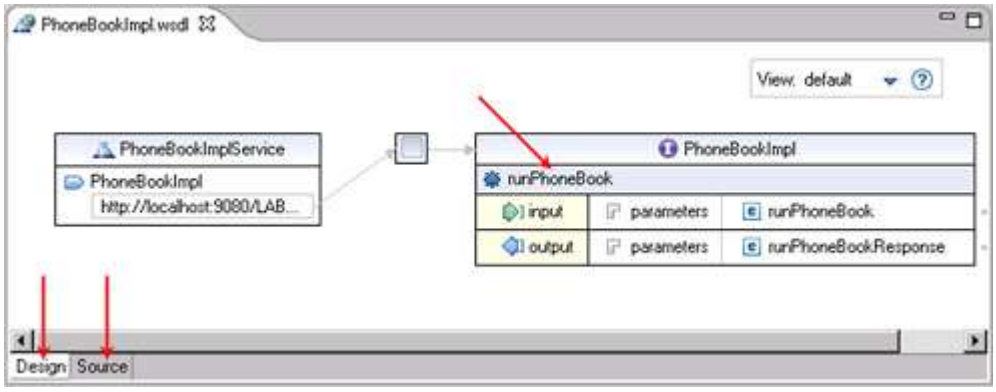
Figure 55. Generated J2EE resources for Web services based on the PhoneBookImpl J2C bean



13. Double-click the **PhoneBookImpl.wsdl** file to open up the WSDL editor. Observe the visual representation of the wsdl file in the **Design** view, as shown in the following Figure. There is also a source view, if you prefer.

Note that the Web service operation is **runPhoneBook** and that the operation has one input and one output, and that these map to the INPUTMSG and OUTPUTMSG defined previously.

Figure 56. PhoneBookImpl.wsdl in the Design view



What is WSDL?

Web Services Description Language (**WSDL**) is an XML-based open specification that describes the interfaces to and instances of Web services on the network. Businesses can make the WSDL documents for their Web services available though UDDI, WSIL, or by broadcasting the URLs to their WSDL via email or Web sites.

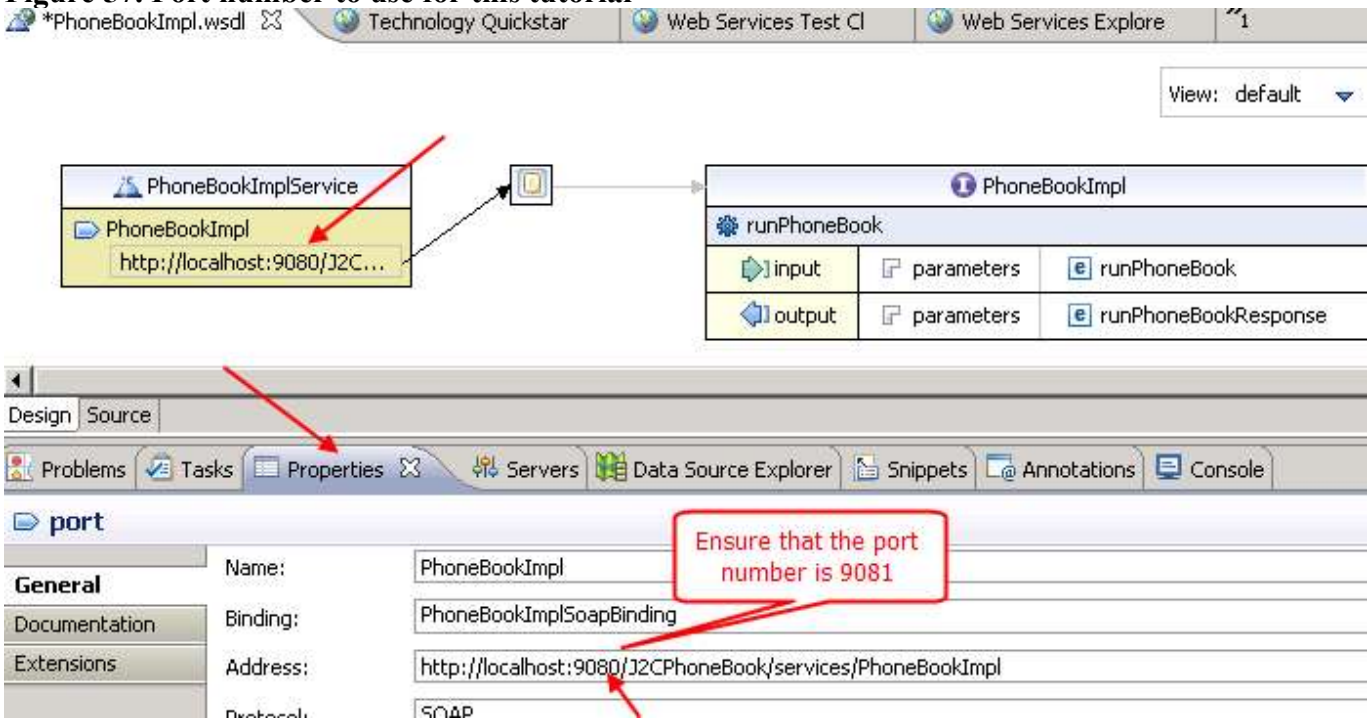
Note that the http://localhost:*portnumber*/... is generated. This tutorial uses port 9081.

If the port number is not 9081, in the **Design** view, click the http that is displayed in http:// localhost:*portnumber*/J2C....

Click the **Properties** tab and change the port number to 9081, as shown in the following Figure.

Save the changes (Ctrl+S).

Figure 57. Port number to use for this tutorial

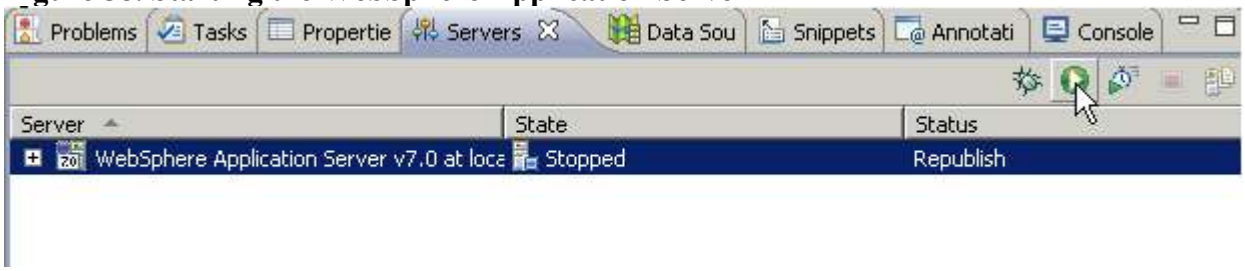


Testing the Web service using Web Service Explorer

In this part of the tutorial, you will use the Web Services Explorer to test your Web service.

1. Make sure that WebSphere Application Server is running. You will test your Web service using the WebSphere Application Server test environment. Using the Servers view, note the **Status**. If the WebSphere Application Server is not running, press the **green arrow** on the **Servers** tab to start it, as shown in the following Figure.. This will take a few moments.

Figure 58. Starting the WebSphere Application Server

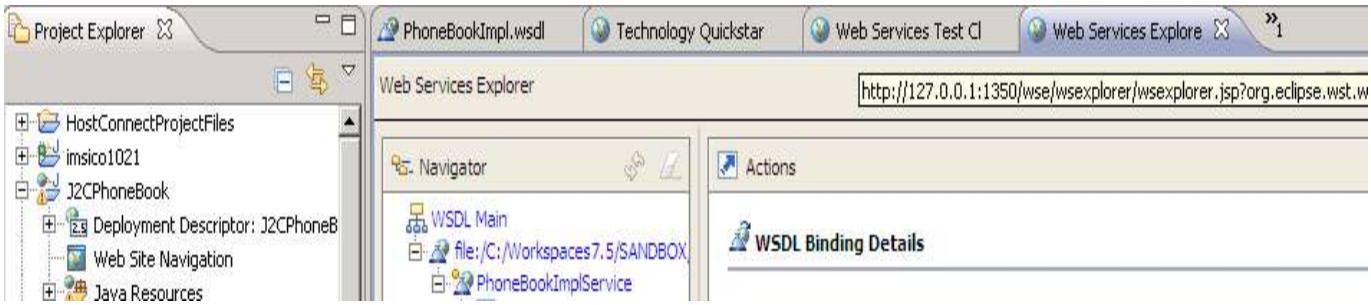


2. From the Project Explorer view, right-click the **PhoneBookImpl.wsdl** file and select **Web Services > Test with Web Services Explorer**, as shown in the following Figure.

Figure 59. Testing with Web Services Explorer

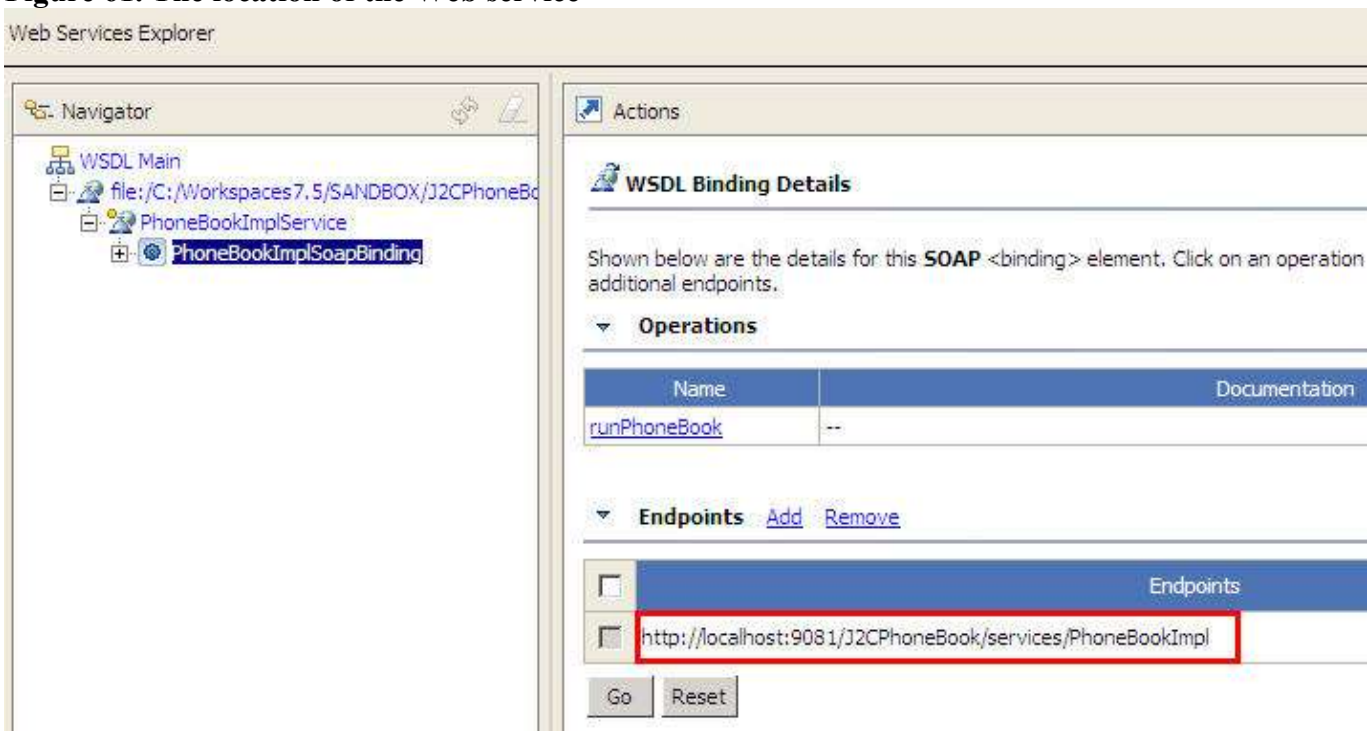
3. The **Web Services Explorer** window will launch. It contains two views: a **Navigator** view and an **Actions** view. To make more space, maximize the window by double-clicking the title, as shown in the following Figure.

Figure 60. Double-clicking the Web Services Explorer tab to maximize the window



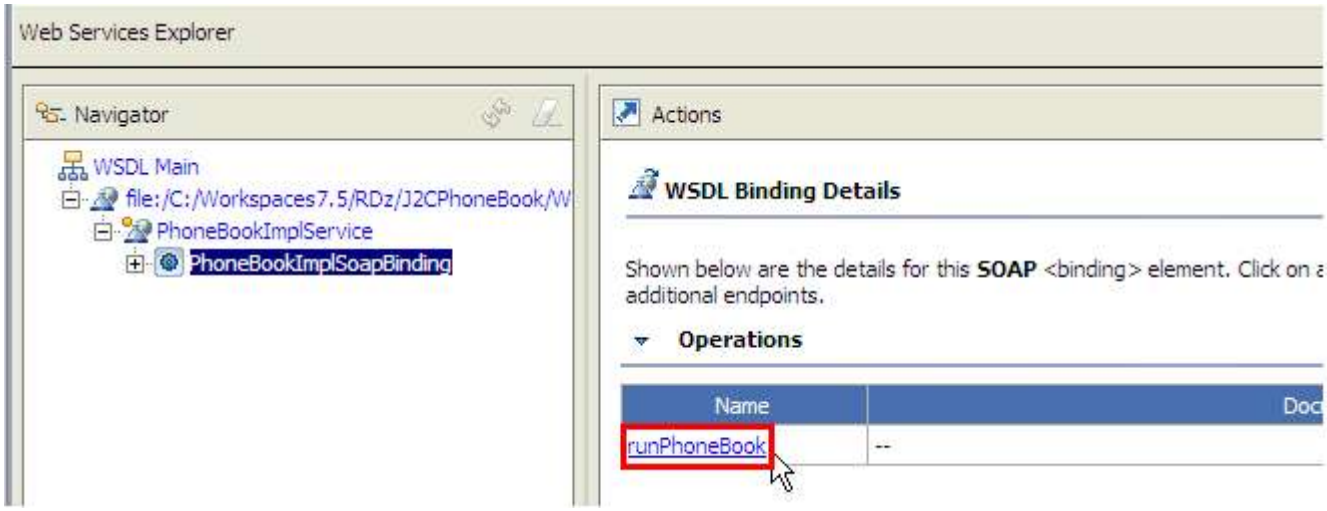
4. The Actions view specifies both the operations and endpoints for the Web service. The Endpoint is defined as http://localhost:9081/J2CPhoneBook/services/PhoneBookImpl. Note that this is the location of the Web service and that it points to the WebSphere Application Server running on the workstation and listening on port **9081**, as shown in the following Figure.

Figure 61. The location of the Web service



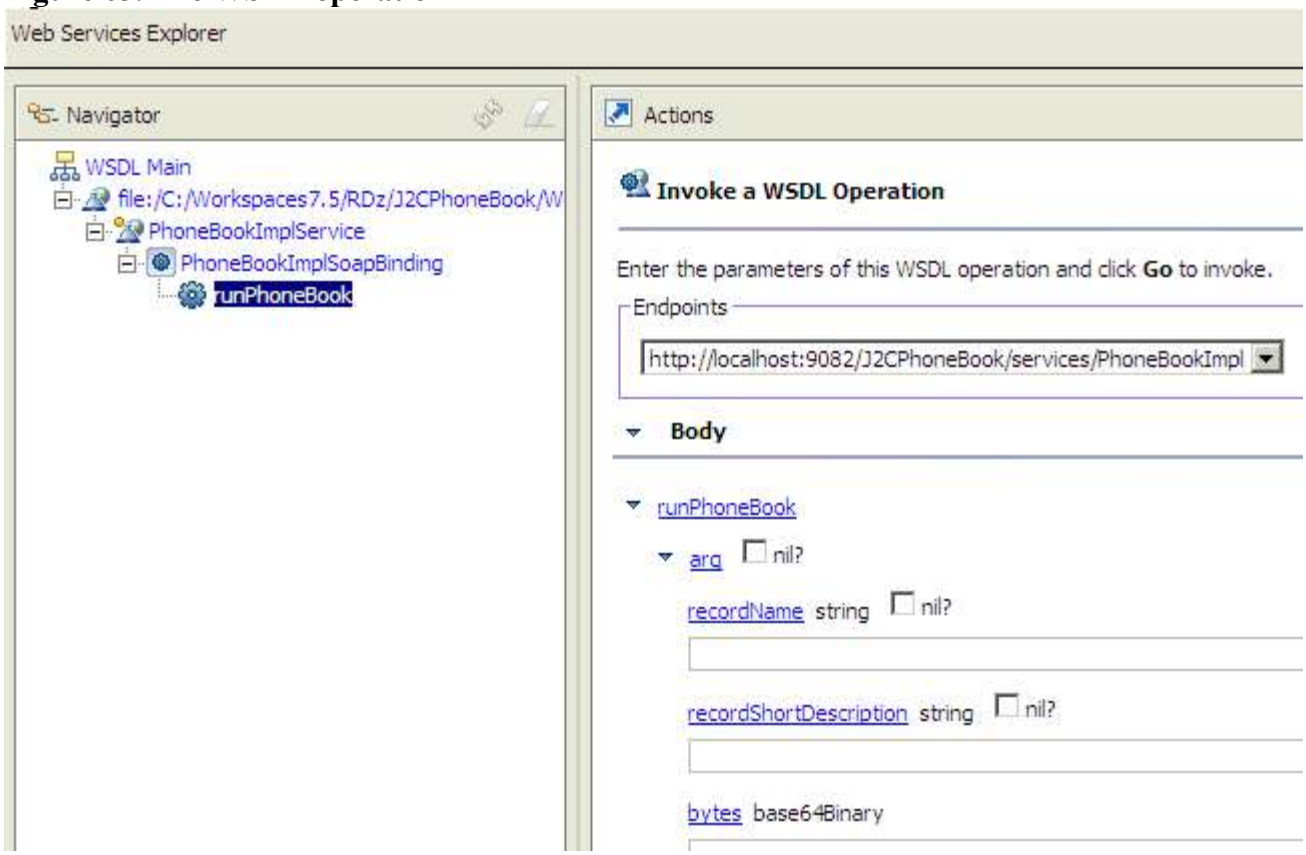
5. Click the **runPhoneBook** operation, as shown in the following Figure.

Figure 62. The runPhoneBook operation for the Web service in Web Services Explorer



6. This will invoke the WSDL operation, as shown in the following Figure.

Figure 63. The WSDL operation



7. Enter the necessary information to run the IVTNO transaction, as shown in the following Figure. 67. These will be the same parameters used for TestClient.jsp:

    a. **in__ll:** `59`

    b. **in__zz:** `0`

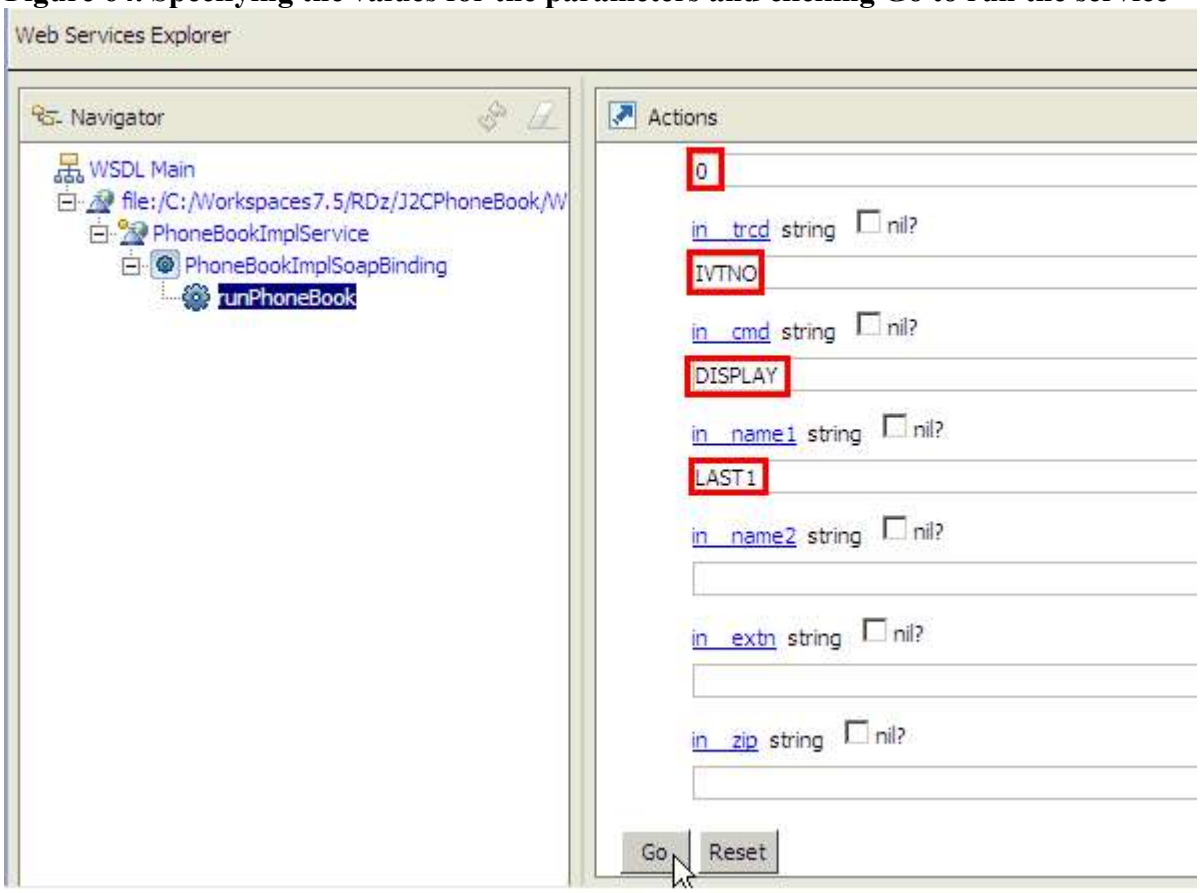
    c. **in__trcd:** `IVTNO`

    d. **in__cmd:** `DISPLAY`

    e. **in__name1:** `LAST1`

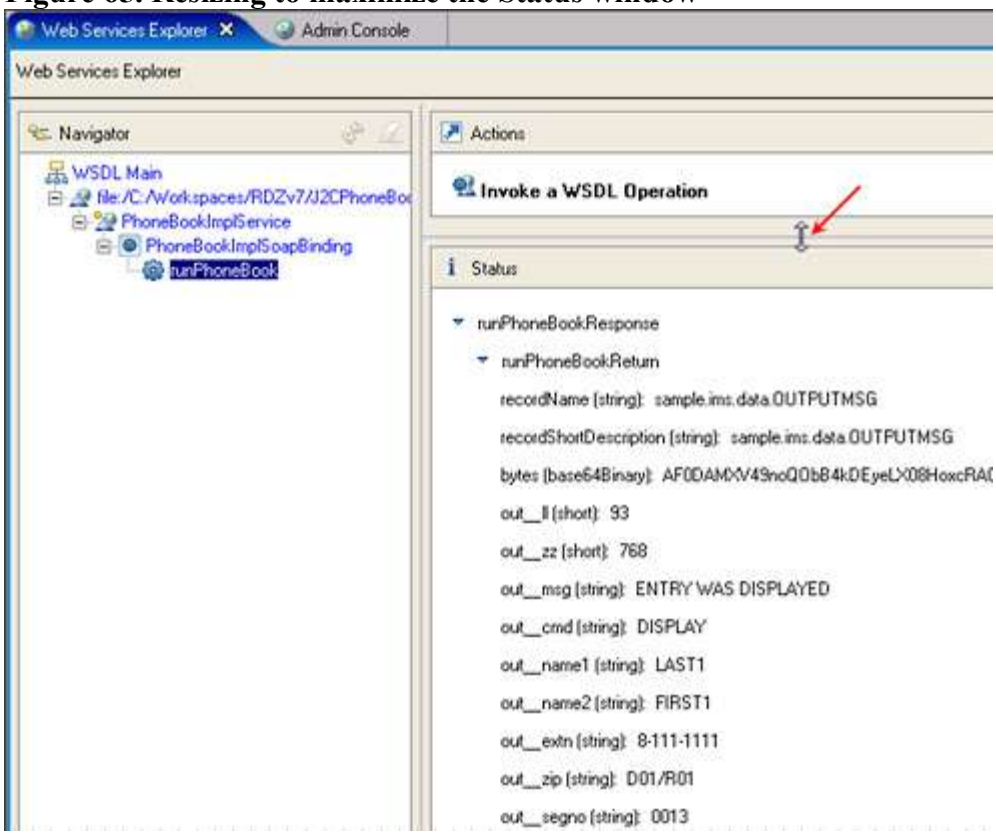
8. Click **Go** to run the Web service:

Figure 64. Specifying the values for the parameters and clicking Go to run the service



9. The Web service executes, and the results are displayed in the **runPhoneBookResponse** object, as shown in the following Figure. Scroll through the Status window or resize the window, as necessary, to see the results:

Figure 65. Resizing to maximize the Status window

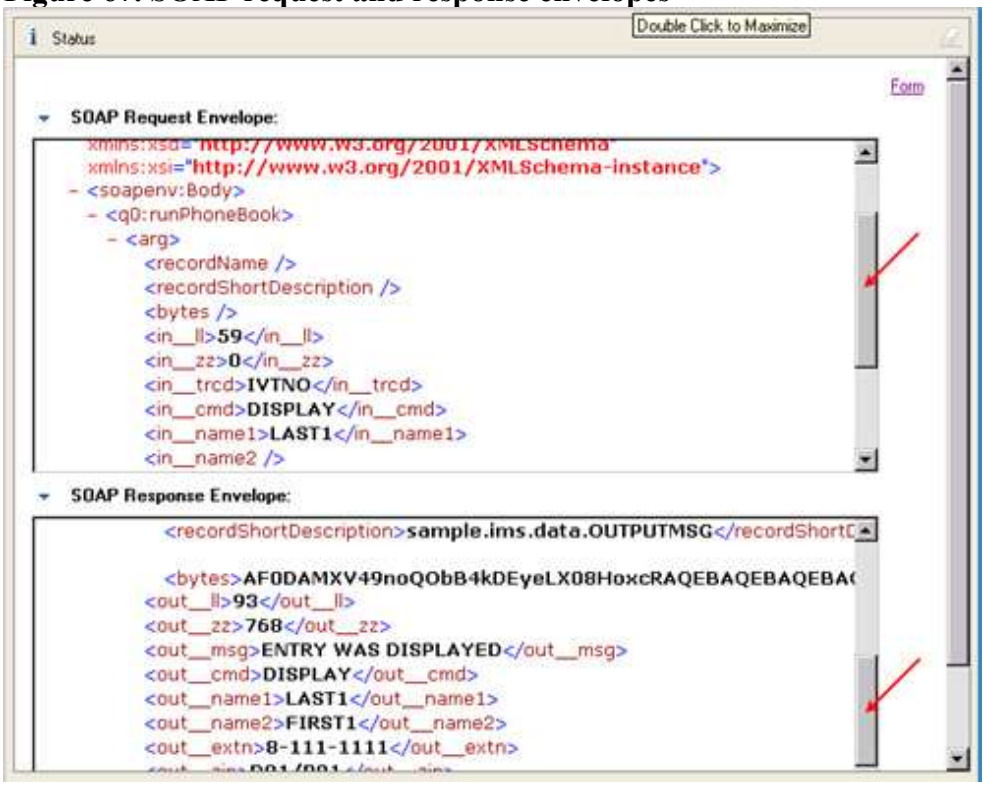


10. Click the **Source** link to display the Web service results in the Form view, as shown in the following Figure.

Figure 66. Clicking the Source link in the Status window

11. This will display the actual SOAP response and SOAP request envelope. Scroll down to see the data being transmitted to and from IMS, as shown in the following Figure.

Figure 67. SOAP request and response envelopes



What is SOAP?

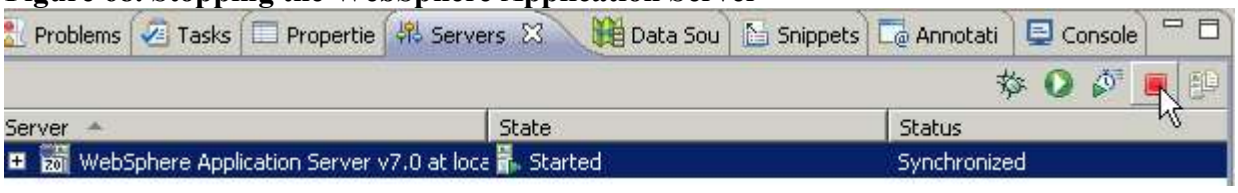
Simple Object Access Protocol (**SOAP**) is an XML-based standard for messaging over HTTP and other Internet protocols. It is a lightweight protocol for the exchange of information in a decentralized, distributed environment. It is based on XML, and consists of three parts:

- **An envelope** that defines a framework for describing what is in a message and how to process it.
- **A set of encoding rules** for expressing instances of application-defined data types.

A convention for representing remote procedure calls and responses.

11. When you are done testing, in the **Server** view, click the red square icon to stop the WebSphere Application Server, as shown in the following Figure.

Figure 68. Stopping the WebSphere Application Server



13. Close all open editor windows (pressing Ctrl+Shift+F4 to accomplish this).

You have completed the optional Web service generation part of the J2C bean tutorial!

Resources

Learn

- Visit the Rational Developer for System z zone on developerWorks® for technical resources and best practices for using this product.
- Visit the IMS TM Resource Adapter Web site.
- Visit the IMS zone on developerWorks for technical resources on Information Management System.
- Visit the Information Management System (IMS) page for product and purchasing information.
- Subscribe to the IBM developerWorks newsletter, a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, Webcasts and events.
- Subscribe to the Rational Edge newsletter for articles on the concepts behind effective software development.
- Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

- Download a trial version of Rational Developer for System z.
- Download IMS TM Resource Adapter.
- Download IBM product evaluation versions and get your hands on application development tools and middleware products from all IBM brands including DB2®, Lotus®, Tivoli®, and WebSphere.

About the authors

- Evgueni Liakhovitch is a Software Developer at IBM.
- Yee-Rong in an Information Developer at IBM.
- Shahin Mohammadi-Rashedi is an IMS SOA Demonstration Team Lead at IBM.

Trademark notice

information" at www.ibm.com/legal/copytrade.shtml.