IBM WebSphere® 6.0 – Lab Exercise

Business Objects and Service Component Architecture

What this exercise is about

The objective of this lab is to provide you with an understanding how to use WebSphere Integration Developer V6 tools to build an SCA based application that uses Business Objects.

Lab Requirements

List of system and software required for the student to complete the lab.

- WebSphere Integration Developer V6 installed
- WebSphere Process Server V6 test environment installed

What you should be able to do

At the end of this lab you should be able to:

- Create a business integration library
- Create and build a business integration module

- Create a business object using WebSphere Integration Developer V6
- Define a component interface that uses a business object
- Define a service export component
- Define a service import component

Introduction

One of the primary objectives of the business object framework is to provide a data abstraction for the service component architecture (SCA). In this exercise you will define 2 business objects for a simple SCA-based financial application. The primary service in this application is a Portfolio service that includes an operation for determining the portfolio value for a particular customer. The top-level business object in this application is the Customer business object. Each customer has a firstName, lastName, customerID, and an array of stocks. All properties defined on the Customer business object are strings, with the exception of the array of stocks which is a collection of Stock business objects. The Stock business object consists of 2 properties, numberOfShares and symbol.

In addition to providing an introduction to basic business object usage with the SCA architecture, this exercise will also demonstrate how to define and use import and export SCA components. The following is a diagram of the application in this exercise:

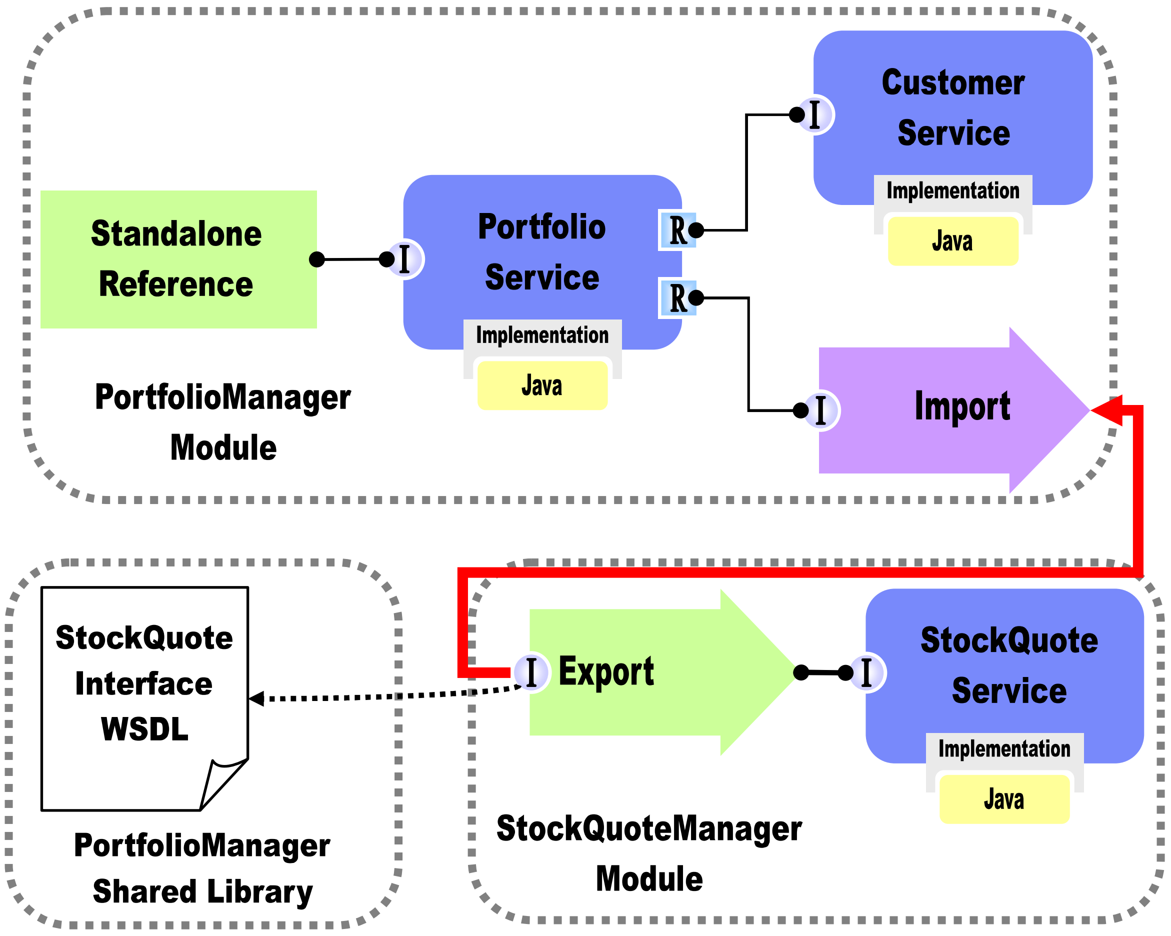


Figure 1

The PortfolioManager Module includes the primary service used in this application. This service is called PortfolioService, and allows the caller to query the total worth of a customer's portfolio given the customer identification number. When invoked, the PortfolioService makes a call to the service CustomerService to query the customer information based upon the customer ID provided. Included in this customer information is a list of stocks that are owned by the customer. The next step in the process is to invoke the StockQuote Service to get the current price on each stock held by the customer. In order to illustrate the use of import and export components in SCA, the StockQuote service is included in a module outside of the PortfolioManager module. In addition to this, both the PortfolioManager and StockQuoteManager modules need access to the StockQuoteInterface WSDL interface, so this artifact has been placed in a library.

Exercise Instructions

Some instructions in this lab might be specific for Windows® platforms. If you run the lab on a platform other than Windows, you will need to run the appropriate commands, and use appropriate files (for example .sh in place of .bat) for your operating system. The directory locations are specified in the lab instructions using symbolic references as follows:

Reference Variable	Windows Location	Linux® Location
<LAB_NAME>	Brokerage	Brokerage
<WID_HOME>	C:\Program Files\IBM\WebSphere\ID\6.0	/opt/IBM/WebSphere/ID/6.0
<WPS_HOME>	<WID_HOME >\runtimes\bi_v6	<WID_HOME>/runtimes/bi_v6
<LAB_FILES>	C:\Labfiles60	tmp/Labfiles60/
<WORKSPACE>	C:\Labfiles60\Brokerage\workspace	/tmp/Labfiles60/Brokerage/workspace
<TEMP>	C:\temp	/tmp
<SOLUTION>	C:\Labfiles60\Brokerage\solution	/tmp/Labfiles60/Brokerage/solution

Windows users' note: When directory locations are passed as parameters to a Java™ program such as EJBdeploy or wsadmin, you must replace the backslashes with forward slashes to follow the Java convention. For example, C:\LabFiles60\ would be replaced by C:/LabFiles60/

Note that the previous table is relative to where you are running WebSphere Integration Developer. The following table is related to where you are running remote test environment:

Reference Variable	Example: Remote Windows test server location	Example: Remote z/OS test server location	Input your values for the remote location of the test server
<SERVER_NAME>	server1	cl1sr01	
<WAS_HOME>	C:\Program Files\IBM\WebSphere\AppServer	/etc/cl1cell/AppServerNode1	
<HOSTNAME>	localhost	mvsxxx.rtp.raleigh.ibm.com	
<BOOTSTRAP_PORT>	2809	2809	
<TELNET_PORT>	N/A	1023	
<PROFILE_NAME>	AppSrv01	default	
<USERID>	N/A	cl1admin	
<PASSWORD>	N/A	fr1day	

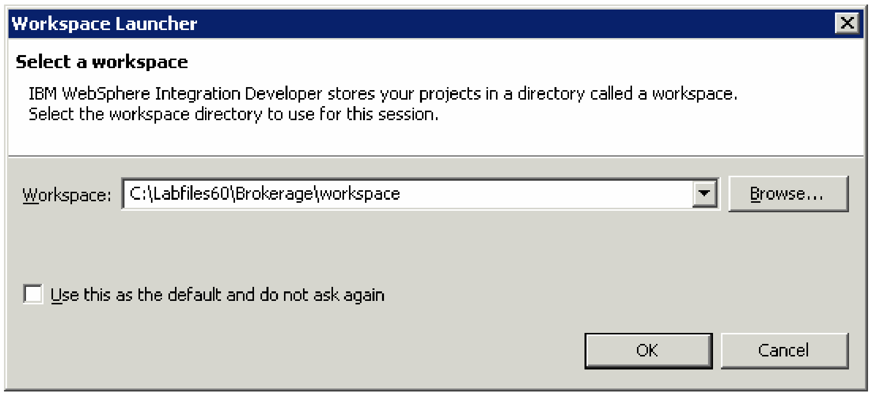
Instructions for using a remote testing environment, such as z/OS®, AIX® or Solaris, can be found at the end of this document, in the section "**Task: Adding Remote Server to WebSphere Integration Developer Test Environment**".

Part 1: Set up Development Environment

____ 1. Start WebSphere Integration Developer V6 with a new workspace.

__ a. Select **Start** > **Programs** > **IBM WebSphere** > **Integration Developer V6.0.1** > **WebSphere Integration Developer V6.0.1** from the start menu.

__ b. Enter **<LAB_FILES>\<LAB_NAME>\workspace** for your workspace and click **OK** when prompted.


Workspace Launcher

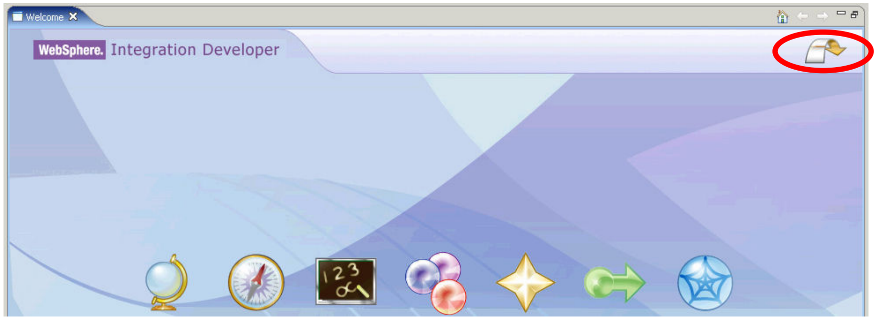
Select a workspace

IBM WebSphere Integration Developer stores your projects in a directory called a workspace.
Select the workspace directory to use for this session.

Workspace: C:\Labfiles60\Brokerage\workspace

☐ Use this as the default and do not ask again

__ c. Close the **Welcome page** after WebSphere Integration Developer V6.0 opens.



Part 2: Setup the PortfolioLibrary Library Project

A library project is a business integration project type in WebSphere Integration Developer V6 that is used to store artifacts that are shared between multiple modules. In this part of the exercise you will create a library project that contains the **StockQuoteInterface** WSDL definition. This interface is used by both the **PortfolioManager** and the **StockQuoteManager** modules. In this exercise you will not be creating the **StockQuoteInterface**, instead you will be importing it into the library.

____ 1. Create a new Library called **PortfolioLibrary**.

__ a. Select **File** > **New** > **Other**.

__ b. Expand **Business Integration** and select **Library**.

__ c. Click **Next**.

__ d. Enter **PortfolioLibrary** for the library name and click **Finish**.

____ 2. Import the **StockQuoteInterface** WSDL file.

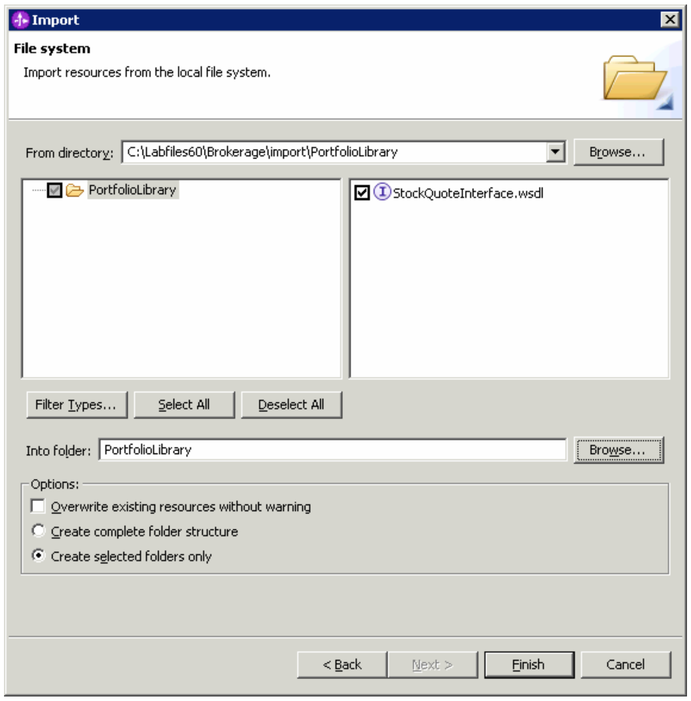
__ a. Select **File > Import** from the menu.

__ b. Select **File System** for the import source and click **Next**.

__ c. Use the Browse button to select **<LAB_FILES>\<LAB_NAME>\import\PortfolioLibrary** for the from directory.

__ d. Check the box next to **StockQuoteInterface.wsdl**.

__ e. Use the **Browse** button to select **PortfolioLibrary** for the **Into folder**.



File system

Import resources from the local file system.

From directory: C:\Labfiles60\Brokerage\import\PortfolioLibrary

[✓] PortfolioLibrary

[✓] StockQuoteInterface.wsdl

Into folder: PortfolioLibrary

Options:

[ ] Overwrite existing resources without warning

○ Create complete folder structure

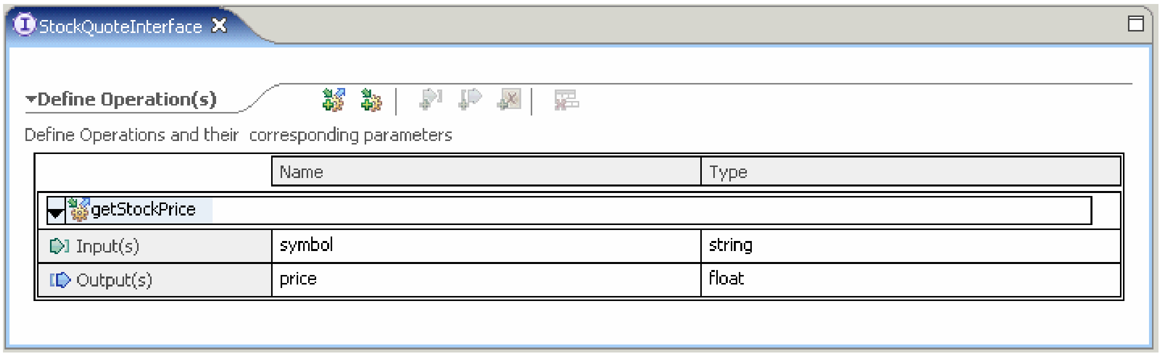
● Create selected folders only

__ f. Click **Finish**.

____ 3. Open the **StockQuoteInterface** WSDL file in the WSDL editor and examine the **getStockPrice** operation.

__ a. Expand **PortfolioLibrary > Interfaces** and double click on **StockQuoteInterface** from the business integration perspective.

__ b. Verify and examine the contents of the WSDL file.



Define Operations and their corresponding parameters

	Name	Type
getStockPrice		
Input(s)	symbol	string
Output(s)	price	float

__ c. **Close** the **StockQuoteInterface** WSDL file.

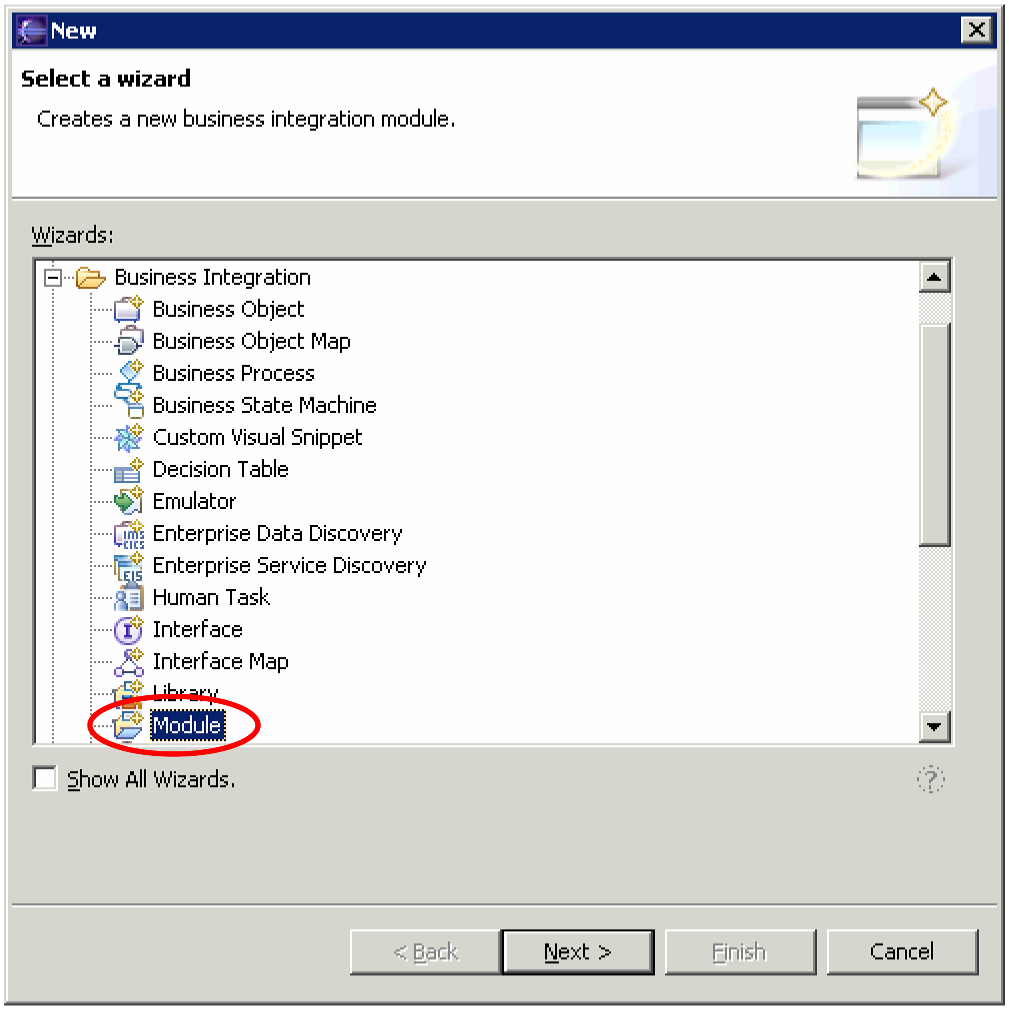
Part 3: Build the StockQuoteManager Module

The **StockQuoteManager** module is an SCA module that provides the implementation for the **StockQuote** service component. This service component is exported to make this functionality available to other modules, such as the **PortfolioManager** module you will build in the next part of this lab. The **StockQuote** service is a simple service that provides the ability to obtain the current stock price for a particular stock, given the stock symbol. This part of the exercise will guide you through creating the **StockQuoteManager** module and the service component and export component in the SCA assembly diagram.

____ 1. Create a new Module called **StockQuoteManager**.

__ a. Select **File** > **New** > **Other**.

__ b. Select **Module** and click **Next**.



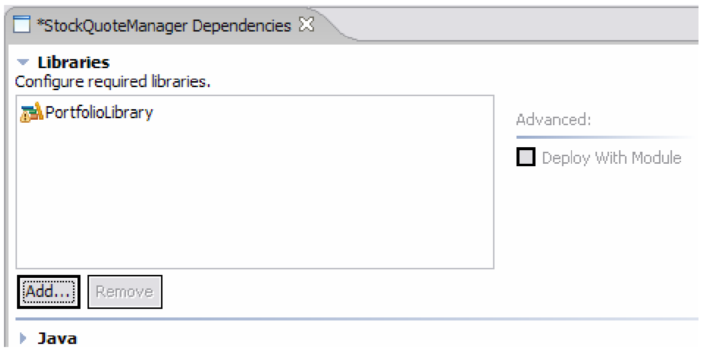
__ c. Enter **StockQuoteManager** for the Module name and click **Finish**.

____ 2. Because the **StockQuoteService** component you defined in this part of the exercise implements the **StockQuoteInterface**; you will need to specify that the **StockQuoteManager** module is dependent on the **PortfolioLibrary** project.

__ a. Right click on StockQuoteManager in the Business Integration view and select **Open dependency Editor**.

__ b. Click the **Add** button under the Library section for the dependency editor.

__ c. Select **PortfolioLibrary** and click **OK**.



NOTE: If you click on the PortfolioLibary in the libraries list you will notice that the '**Deploy With Module'** box is checked. By default dependent projects that are added in the dependency editor are deployed with the module.

__ d. **Save** and **close** the dependency editor.

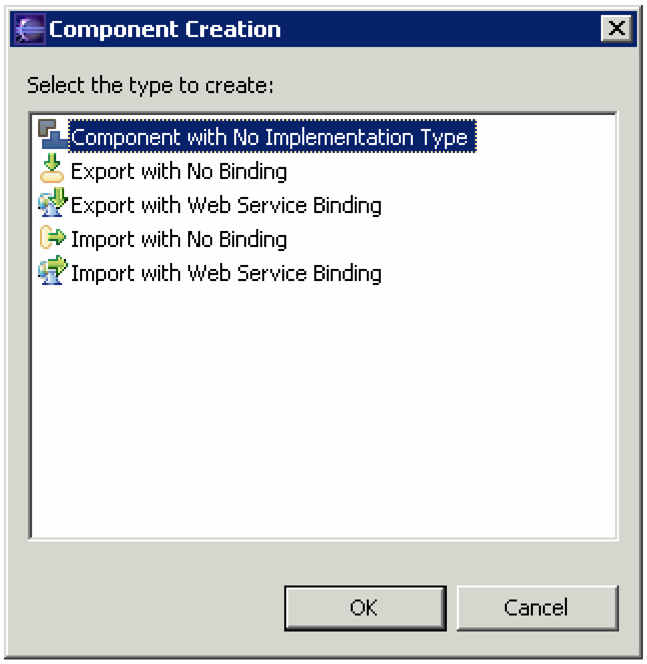
____ 3. Open the assembly editor for the **StockQuoteManager** module.

__ a. Expand **StockQuoteManager** in the Business Integration view and double click on **StockQuoteManager**.

____ 4. Add the **StockQuoteInterface** to the assembly diagram.

__ a. Expand **PortfolioLibrary** > **Interfaces** in the Business Integration view, drag and drop the **StockQuoteInterface** onto the assembly diagram.

__ b. Select **Component with No Implementation Type** and click **OK**.


Component Creation
Select the type to create:
Component with No Implementation Type
Export with No Binding
Export with Web Service Binding
Import with No Binding
Import with Web Service Binding
OK
Cancel

__ c. Click on **Component1** in the assembly editor and change the **Display name** and **Name** to **StockQuoteService** in the **properties** view.

____ 5. Provide an implementation for the **StockQuoteService** component.

__ a. From the assembly editor, right click on **StockQuoteService** and select **Generate Implementation > Java** from the context menu.

__ b. Select **default** package as package where the Java Implementation will be generated and click **OK**.

__ c. The Java editor will open in your work area with a Java file called **StockQuoteServiceImpl.java**.

__ d. Locate the **getStockPrice** method definition in the class file and enter the following code in the body of the method.

```
float result = 0.00f;
if (symbol.equals("IBM")) {
      result = 90.00f;
} else {
      result = 50.00f;
}
return new Float(result);
```

NOTE: For your convenience, this code can be found in **<LAB_FILES>\Brokerage \snippets\snippet1.txt.**

__ e. **Save** and **close** the file.

____ 6. Add an Export component for the **StockQuoteInterface**.

__ a. Locate and click on the **export** icon (📥).

__ b. Click anywhere on the canvas of the assembly diagram.

____ 7. Add the **StockQuoteInterface** to the **Export1** component.

__ a. Click on the **Export1** component in the assembly diagram.

__ b. Hover your mouse over the **Export1** component until you see the Add interface icon ( ) and click this icon.

NOTE: If you do not see the hover over icon, you can also add an interface from the Interfaces tab of the properties view while the Export component is selected in the assembly diagram.

__ c. From the Add Interface dialog, click on **StockQuoteInterface** from the Matching interfaces list and click **OK**.

____ 8. Set the SCA export binding.

__ **a.** Right click on the **Export1** component in the assembly diagram and select **Generate Binding > SCA Binding**.

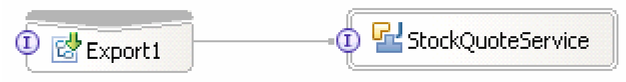
____ 9. Create a wire between the **Export1** component and the **StockQuoteService** component.

__ a. Select the wire tool ( ) and then click on **Export1**.

__ b. Click on the **StockQuoteService** component.

__ c. Click on the selection tool ( ) from the palette.

Right click on the canvas and select **Arrange Contents Automatically** and verify that your diagram looks like the following.



____ 10. **Save** and **close** the assembly diagram.

Part 4: Build the PortfolioManager Module

The **PortfolioManager** module is the primary SCA module in this exercise. The key service component in the module is the called **PortfolioService**. This service provides the ability to return the total value of a customer's portfolio given the customer ID. The **PortfolioService** component references another service component in the same module called **CustomerService**. The **CustomerService** component provides information about the customer, including what stocks the customer owns. The data exchanged between the **PortfolioService** and the **CustomerService** is a top level business object called Customer and you will build in this business object in this part of the exercise. Once the customer information is obtained the **PortfolioService** utilizes a reference to an import component that provides service access to **StockQuoteService** defined in the previous section.

____ 1. Create the **PortfolioManager** Module.

__ a. Select **File > New > Other**.

__ b. Select **Module** and click **Next**.

__ c. Enter **PortfolioManager** for the Module name and click **Finish**.

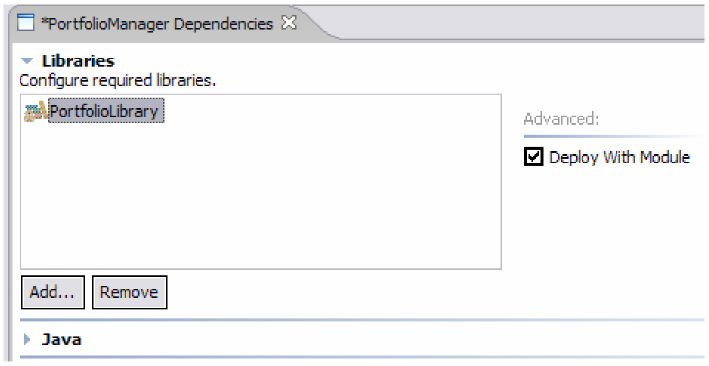
____ 2. Because the import component you will define in this part of the exercise utilizes the **StockQuoteInterface**, you will need to specify that the **PortfolioManager** module is dependent on the **PortfolioLibrary** project.

__ a. Right click on **PortfolioManager** in the Business Integration view and select **Open Dependency Editor**.

__ b. Click the **Add** button under the Library section for the dependency editor.

__ c. Select **PortfolioLibrary** and click **OK**.

__ d. Click on the PortfolioLibrary in the libraries list, and verify that the box **Deploy With Module** is checked



__ e. **Save** and **close** the dependency editor.

____ 3. Create the **Stock** business object.

__ a. Expand **PortfolioManager** from the Business Integration view.

__ b. Right click on **Data Types** and select **New > Business Object**.

__ c. Enter **Stock** for the name of the business object and click **Finish**.

__ d. The business object editor will open in your workspace. To begin defining the **Stock** business object, select the **Stock** object in the diagram.

NOTE: You can edit the properties of the business object from the properties view or within the business object editor. Within the business object editor hover over icons are available that allow you to add and remove attributes associated with the business object.

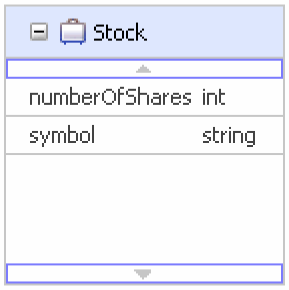
__ e. Right click on the **Stock** object and select **Add attribute**.

__ f. Select the newly added attribute. The attribute name and type can be edited from either the properties view or directly within the business object editor. Change the attribute name to **numberOfShares** and the type to **int**.

NOTE: The names of the business object attributes are case sensitive.

__ g. Create another attribute on the Stock business object. Enter **symbol** for the attribute name and leave the type as **string**.

__ h. Verify that the **Stock** business object looks like the following.



__ i. **Save** and **close** the file.

____ 4. Create the **Customer** business object.

__ a. Expand **PortfolioManager** from the business integration view.

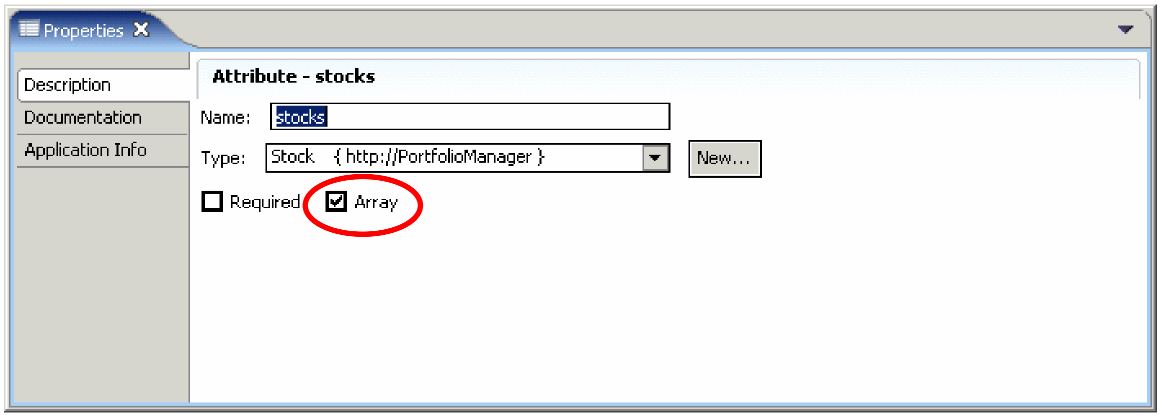
__ b. Right click on **Data Types** and select **New > Business Object**.

__ c. Enter **Customer** for the name of the business object and click **Finish**.

__ d. Create the following attributes with the types specified in the table below.

Attribute Name	Type
customerID	string
firstName	string
lastName	string
stocks	Stock http://PortfolioManager

__ e. Note that the **stocks** attribute type is the **Stock** business object you created in the previous step. Also note that this should be specified as an array. Use the properties view to set this attribute type to array.



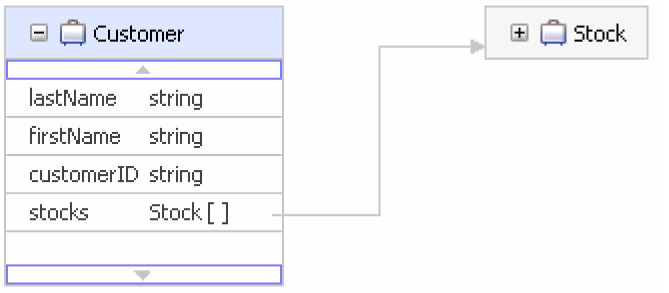
Attribute - stocks

Name: stocks

Type: Stock { http://PortfolioManager }

☐ Required ☑ Array

__ f. Verify that the Customer business object looks like the following.



lastName	string
firstName	string
customerID	string
stocks	Stock [ ]

__ g. **Save** and **close** the file.

_____ 5. Import the CustomerService and PortfolioService Java interfaces and implementations.

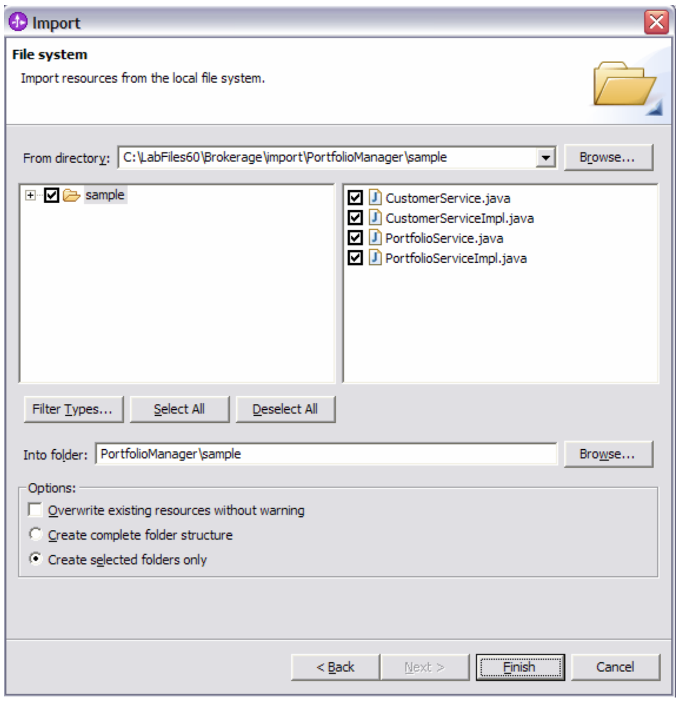
__ a. Select **File > Import** from the menu.

__ b. Select File System for the import source and click **Next**.

__ c. Use the Browse button and select **<LAB_FILES>\<LAB_NAME>\import\PortfolioManager\sample** for the from directory.

__ d. Check the box next to the sample folder.

__ e. Specify the '**Into folder**' as **PortfolioManager\sample**.



From directory: C:\LabFiles60\Brokerage\import\PortfolioManager\sample

sample

CustomerService.java
CustomerServiceImpl.java
PortfolioService.java
PortfolioServiceImpl.java

Into folder: PortfolioManager\sample

Options:
Overwrite existing resources without warning
Create complete folder structure
Create selected folders only

__ f. Click **Finish**.

NOTE: For both the **PortfolioService** and **CustomerService** components the interfaces have been specified as Java Interfaces and were imported into the project in the previous step along with Java implementations for these interfaces. In the following steps you will build the assembly diagram for the **PortfolioManager** Module. An easy way to build this diagram is to drag and drop the Java implementation files onto the assembly editor.

____ 6. Open the assembly editor for the **PortfolioManager** module.

__ a. Expand **PortfolioManager** in the Business Integration view and double click on **PortfolioManager**.

____ 7. Add the **PortfolioService** component to the assembly diagram.

__ a. Add a java component to the canvas of the assembly diagram. To do this, locate and click on the ( ) icon on the palette of the assembly diagram and then click anywhere on the canvas of the assembly diagram.

NOTE: Click the gray > symbol next to () on the palette to locate the Java component icon ().

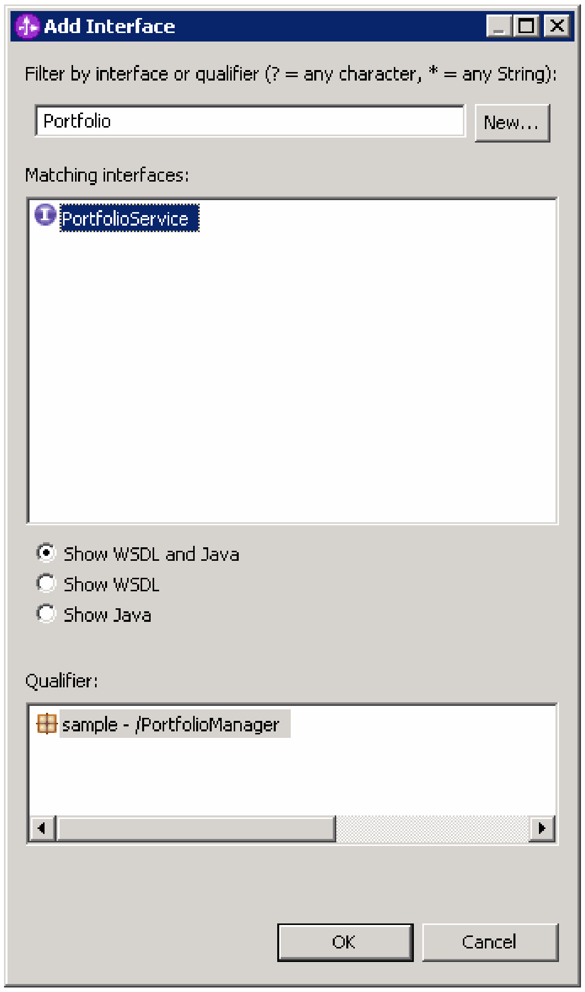
__ b. In the assembly diagram, select the component added in the previous step.

__ c. Change the **Display Name** and **Name** of the component from **Component1** to **PortfolioService** from the properties view.

__ d. Hover your mouse over the **PortfolioService** component until you see the **Add interface** icon () and click this icon.

NOTE: If you do not see the hover over icon, you can also add an interface from the Interfaces tab of the properties view while the **PortfolioService** component is selected in the assembly diagram.

__ e. From the Add Interface dialog, begin typing **PorfolioService** into the "**Filter by interface or qualifiers…**" (Defaulted to *) box. Select the **PortfolioService** interface from the "**Matching types**" lists and when it appears and click **OK**.


Add Interface

Filter by interface or qualifier (? = any character, * = any String):

Portfolio

Matching interfaces:

PortfolioService

(●) Show WSDL and Java
( ) Show WSDL
( ) Show Java

Qualifier:

sample - /PortfolioManager

__ f. Specify the implementation for the **PortfolioService** component. Right click on the **PortfolioService** component and select **Select Implementation** from the context menu. When the **Pick Implementation** dialog pops up, type in **PortfolioServiceImpl** into the "**Select entries**". Select the **PortfolioServiceImpl** from the "**Matching types:**" list when it appears and click **OK**.

____ 8. Add the **CustomerService** component to the assembly diagram

__ a. Repeat the previous steps outlined in Step 7 to add the interface and implementation for the **CustomerService** as specified in the table below

Name	Type
CustomerService	Interface
CustomerServiceImpl	Implementation class

__ b. Change the **Display name** and **Name** of the component from **Component1** to **CustomerService** from the properties view.

____ 9. Add the **Import** component for the StockQuoteService.

__ a. Locate and click on the **Import** icon ( ).

__ b. Click anywhere on the canvas of the assembly diagram.

____ 10. Add the **StockQuoteInterface** to the **Import1** component.

__ a. Click on the **Import1** component in the assembly diagram.

__ b. Hover your mouse over the **Import1** component until you see the **Add interface** icon ( ) and click this icon.

NOTE: If you do not see the hover over icon, you can also add an interface from the Interfaces tab of the properties view while the Export component is selected in the assembly diagram.

__ c. From the **Add Interface** dialog, click on **StockQuoteInterface** from the Matching interfaces list and click **OK**.

____ 11. Set the import binding type for **Import1**.

__ a. Right click on the **Import1** component in the assembly diagram and select **Generate Binding > SCA Binding**.

__ b. With the **Import1** component still selected, go to the **Binding** tab of the Properties view.

__ c. Enter **StockQuoteManager** for the module name.

__ d. Click the **Browse** button and select **Export1** from the list of matches for the Export name field.

____ 12. Create a wire between the **PorfolioService** component and the **CustomerService** component.

__ a. Select the wire tool ( ) and then click on the **PorfolioService**.

__ b. Click on the **CustomerService** component.

__ c. Click **OK** when you see the message appear indicating that a matching reference will be created on the source node.

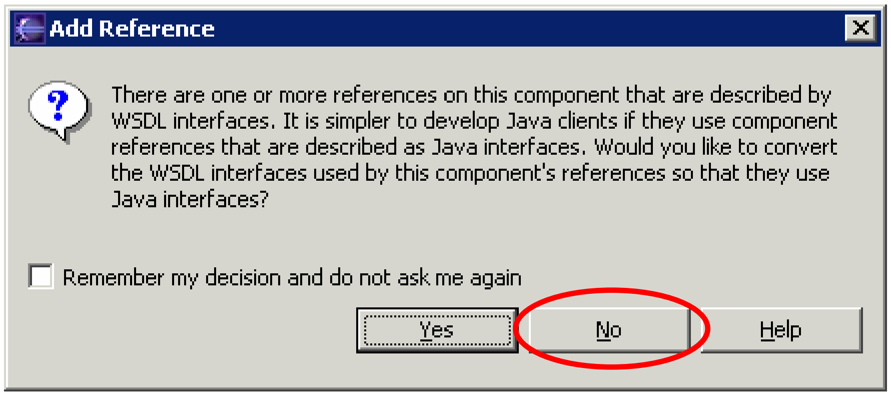
____ 13. Create a wire between the **PorfolioService** component and the **Import1** component.

__ a. Select the wire tool ( ) and then click on the **PorfolioService**.

__ b. Click on the **Import1** component.

__ c. Click **OK** when you see the message appear indicating that a matching reference will be created on the source node.

__ d. Click **No** when prompted if you would like to convert the WSDL interface to a Java interfaces.


Add Reference

There are one or more references on this component that are described by WSDL interfaces. It is simpler to develop Java clients if they use component references that are described as Java interfaces. Would you like to convert the WSDL interfaces used by this component's references so that they use Java interfaces?

____ 14. Add a stand-alone reference to the diagram.

__ a. Locate and click on the stand-alone reference icon ( ).

__ b. Click anywhere on the canvas of the assembly diagram.

____ 15. Add a reference to the stand-alone reference component from the previous step.

__ a. In the assembly diagram select the stand-alone reference and click the Add Reference icon ( ) from the hover over menu.

__ b. Select the radio button next to **Show Java**.

__ c. Begin typing **PortfolioService** in the '**Filter by interface …**' (Defaulted to *) box and select the **PortfolioService** from the '**Matching Interfaces**' list when it appears.

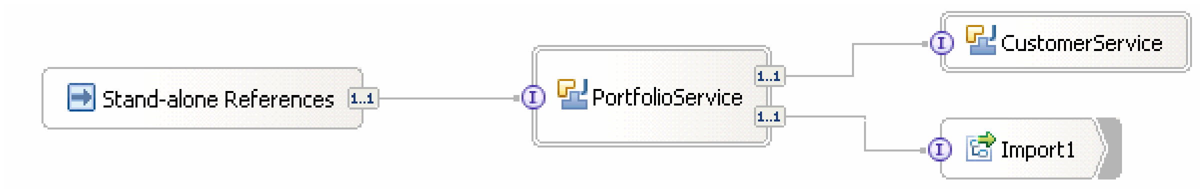
__ d. Enter **PortfolioServicePartner** for the name of the reference and click **OK**.

____ 16. Create a wire between the stand-alone reference and the PortfolioService component.

__ a. Select the wire tool ( ) and then click on the reference box on the stand-alone reference ( ).

__ b. Next, click on the PortfolioService component.

__ c. Verify that your diagram looks like the following.



____ 17. **Save** and **Close** the assembly editor file.

____ 18. Create a dynamic web project to hold the JSP that provides the user interface for the PortfolioManager application. (Step 18 is required for 6.0.1 and beyond but is not required for version 6.0)

__ a. Select the **J2EE Perspective** and expand **Enterprise Applications.**

__ b. Right click on HelloWorldApp and select **New > Dynamic Web Project** from the list.

__ c. Enter **HelloWorldClient** for the name of the Dynamic Web project.

__ d. Click **Finish**.

__ e. Click **Yes** to confirm a switch to the Web Perspective.

NOTE: This will enable the Web Development capability within WebSphere Integration Developer for this workspace.

____ 19. Import the client JSP file.

__ a. Select **File > Import** from the menu.

__ b. Select 'File System' and click **Next**.

__ c. From the File system import dialog click the **Browse** button and choose the following From directory: **<LAB_FILES>\<LAB_NAME>\import\PortfolioManager**.

__ d. Check the box next to the index.jsp file and click the Browse button to select **PortfolioManagerClient/WebContent** as the Into folder.

__ e. Click **Finish**.

Part 5: Test the Application

____ 1. Start the server.

__ a. If using a remote testing environment, follow the directions provided in **TASK: Adding Remote Server to WebSphere Integration Developer Test Environment** at the end of this document to add a server to the WebSphere Integration Developer test environment and start it. This is especially true for z/OS, AIX, Solaris remote test environment, where the WebSphere Integration Developer will be remote to the test environment.

If using a local testing environment, right click on WebSphere Process Server V6.0 from the Server view and select **Start** from the context menu.

__ b. Wait for the server to start.

____ 2. Add the StockManager and PortfolioManager project to the configured projects on the server.

__ a. Right click on the WebSphere Process Server V6.0 from the Server view and select **Add and remove projects…** from the context menu.

__ b. Select **StockQuoteManagerApp** from the available projects list and click **Add**.

__ c. Select **PortfolioManagerApp** from the available projects list and click **Add**.

__ d. Click **Finish**.

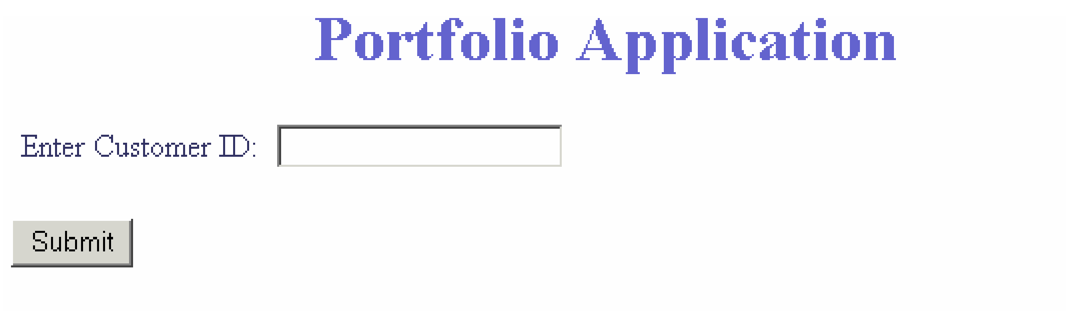
____ 3. Test the PortfolioService service component.

__ a. Open a web browser and enter the following URL.

`http://<HOSTNAME>:9080/PortfolioManagerClient/index.jsp`

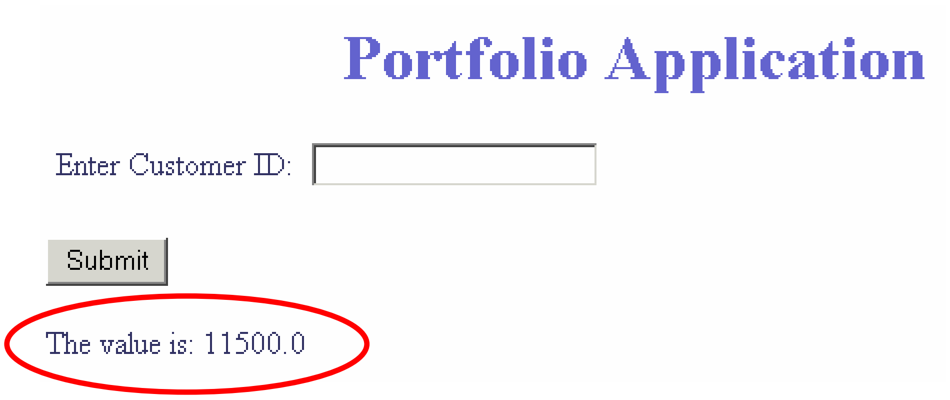
NOTE: To open the embedded browser within WebSphere Integration Developer V6 select **Window > Customize Perspective** from the menu. In the Customize Perspective dialog click on the Commands tab and scroll down to the bottom of the available command groups list until you find Web Browser. Check the box next to Web Browser and click **OK**. Click the Web browser icon (🌐) now in your toolbar to open the embedded Web browser.

__ b. Verify that you see the following page displayed in the browser.



__ c. Enter **123-45-6789** for the Customer ID and click Submit.

__ d. Verify that you see the following output.



The value is: 11500.0

Part 6: Restore Server Configuration

____ 1. Remove the StockManager and PortfolioManager project from the configured projects on the server.

From the Servers view right click on the WebSphere Process Server V6.0 and select **Add and remove projects…** from the context menu.

Select PortfolioManager from the configured projects list and click **Remove**.

Select StockManager from the configured projects list and click **Remove**.

Click **Finish**.

Stop the server. Right click on WebSphere Process Server V6.0 server from the Servers view and select **Stop** from the context menu.

What you did in this exercise

In this exercise you saw how to use WebSphere Integration Developer V6 tools to do the following tasks:

- Create a business integration library
- Create and build a business integration module
- Create a business object
- Define a component interface that uses a business object
- Define a service export component
- Define a service import component

.

Solution Instructions

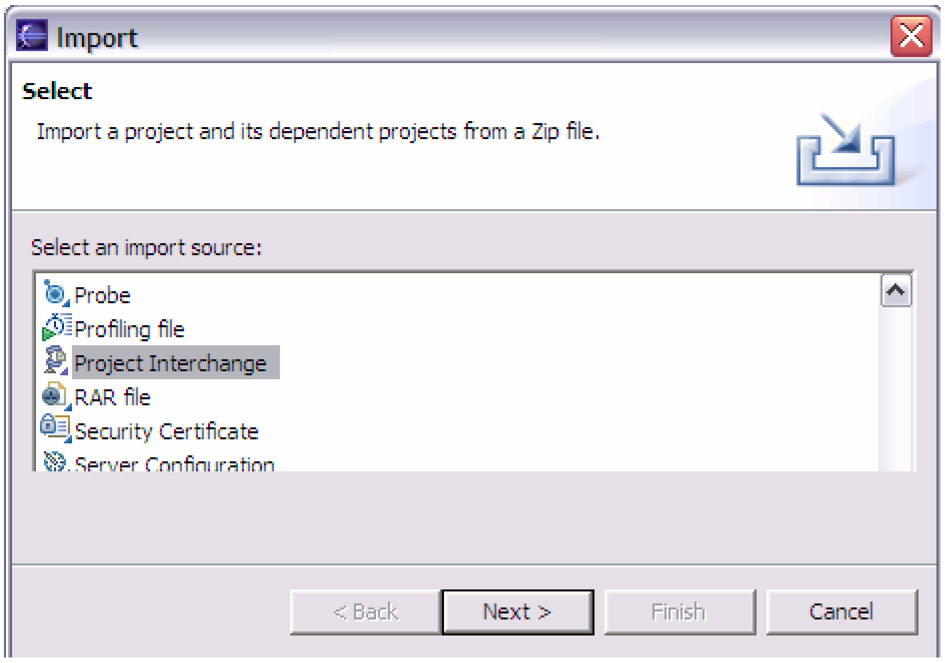
____ 1. Start WebSphere Integration Developer V6 with a new workspace.

__ a. Follow the instructions outlined in Part 1 of this exercise.

____ 2. Import the project interchange file **Brokerage_PI.zip** from **<LAB_FILES>\<LAB_NAME>\solution** directory.

__ a. Select **File > Import** from the menu.

__ b. Select **Project Interchange** in the **Import** dialog and click **Next**.



__ c. For the **From zip file**, click on the **Browse** button and select the **Brokerage_PI.zip** in the **<LAB_FILES>\<LAB_NAME>\solution** director.

__ d. Enter **<LAB_FILES>\<LAB_NAME>\workspace** for the **Project location root**.

__ e. Click the Select All button and click **Finish**.

____ 3. Continue with Part 5 of this exercise.

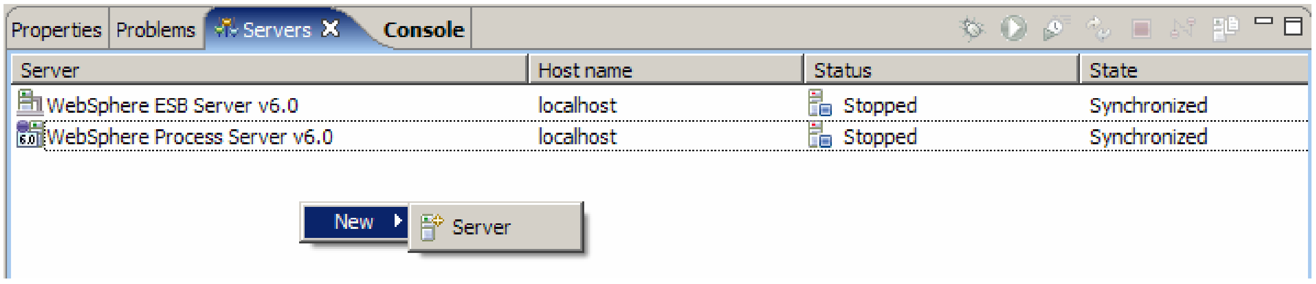
Task: Adding Remote Server to WebSphere Integration Developer Test Environment

This task describes how to add a remote server to the WebSphere Integration Developer Test environment. The sample will use a z/OS machine.

____ 4. Create a new remote server.

__ a. Right click on the background of the Servers view to access the pop-up menu.

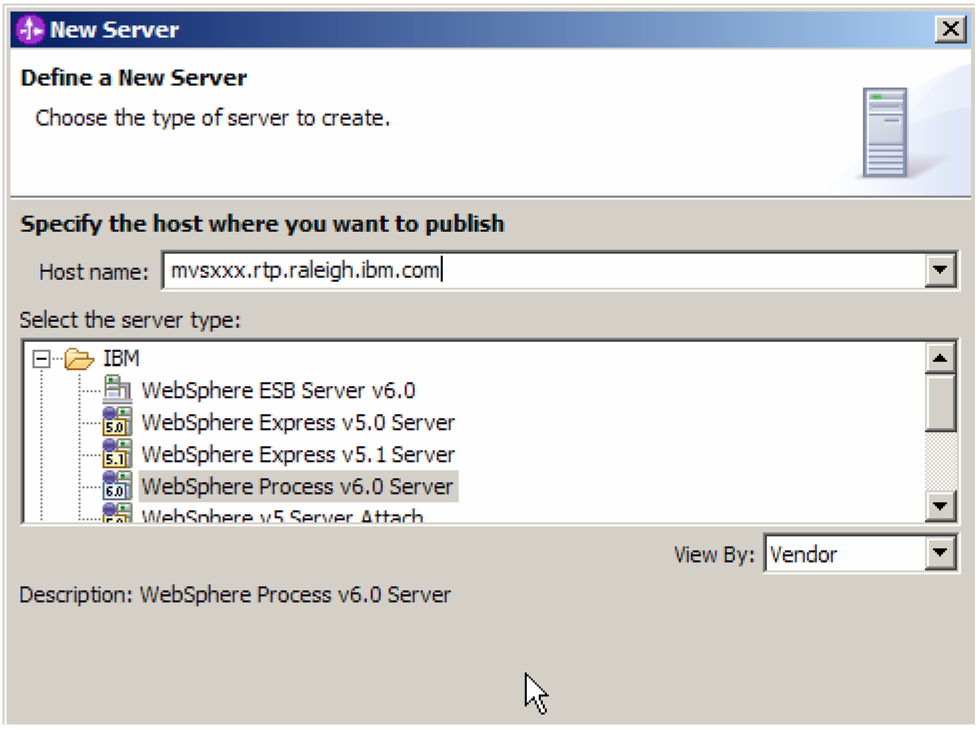
__ b. Select **New > Server.**



Server	Host name	Status	State
WebSphere ESB Server v6.0	localhost	Stopped	Synchronized
WebSphere Process Server v6.0	localhost	Stopped	Synchronized

__ c. Specify hostname to the remote server, **<HOSTNAME>**.

__ d. Ensure that '**WebSphere Process v6.0 Server**' is highlighted in the server type list.



Define a New Server

Choose the type of server to create.

Specify the host where you want to publish

Host name: mvsxxx.rtp.raleigh.ibm.com

Select the server type:

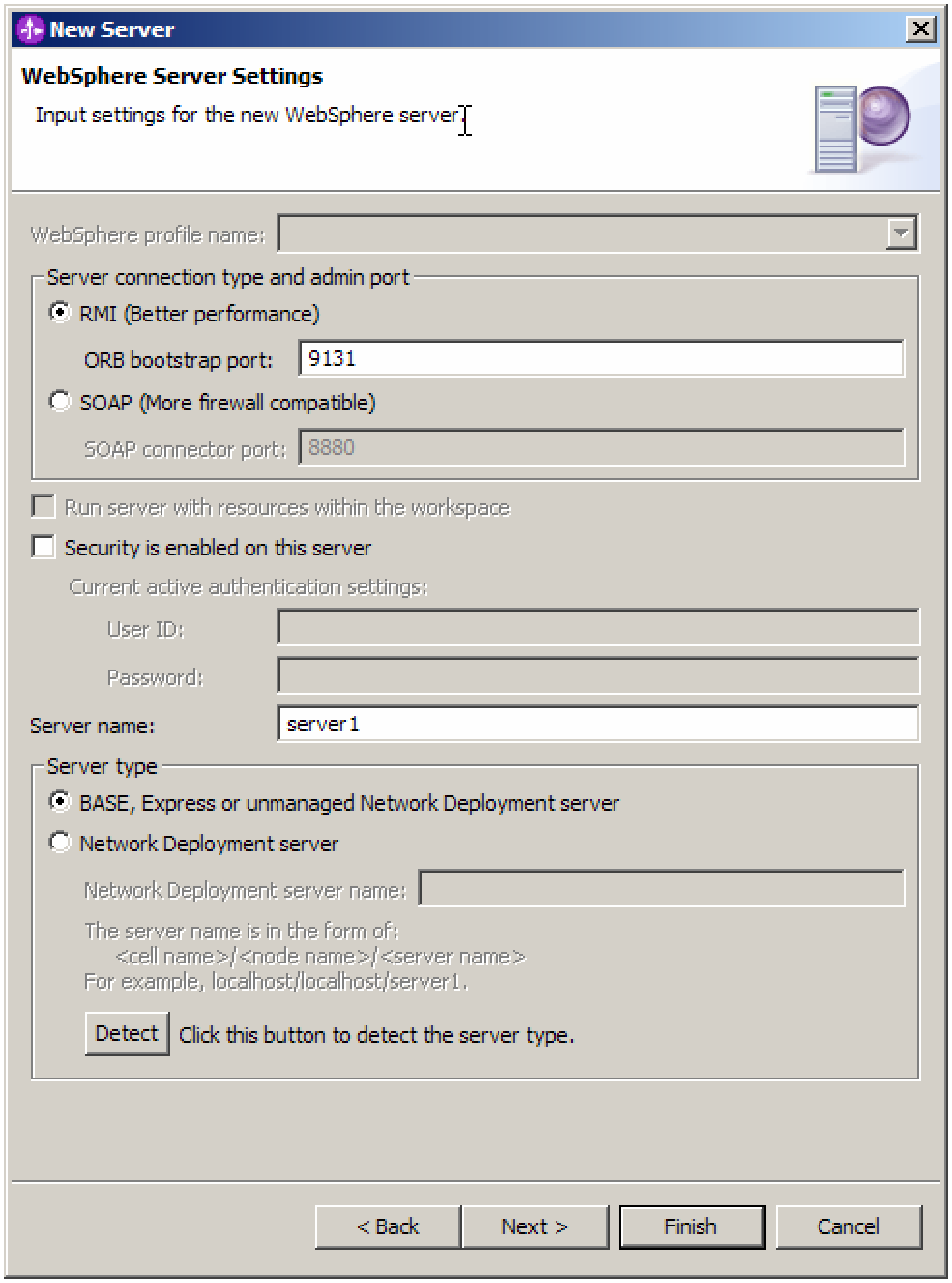
- IBM
  - WebSphere ESB Server v6.0
  - WebSphere Express v5.0 Server
  - WebSphere Express v5.1 Server
  - WebSphere Process v6.0 Server
  - WebSphere v5 Server Attach

View By: Vendor

Description: WebSphere Process v6.0 Server

__ e. Click **Next.**

__ f. On the WebSphere Server Settings page, select the radio button for **RMI** and change the ORB bootstrap port to the correct setting (**<BOOTSTRAP_PORT>**).



New Server

WebSphere Server Settings

Input settings for the new WebSphere server

WebSphere profile name:

Server connection type and admin port

(●) RMI (Better performance)

ORB bootstrap port: 9131

( ) SOAP (More firewall compatible)

SOAP connector port: 8880

☐ Run server with resources within the workspace

☐ Security is enabled on this server

Current active authentication settings:

User ID:

Password:

Server name: server1

Server type

(●) BASE, Express or unmanaged Network Deployment server

( ) Network Deployment server

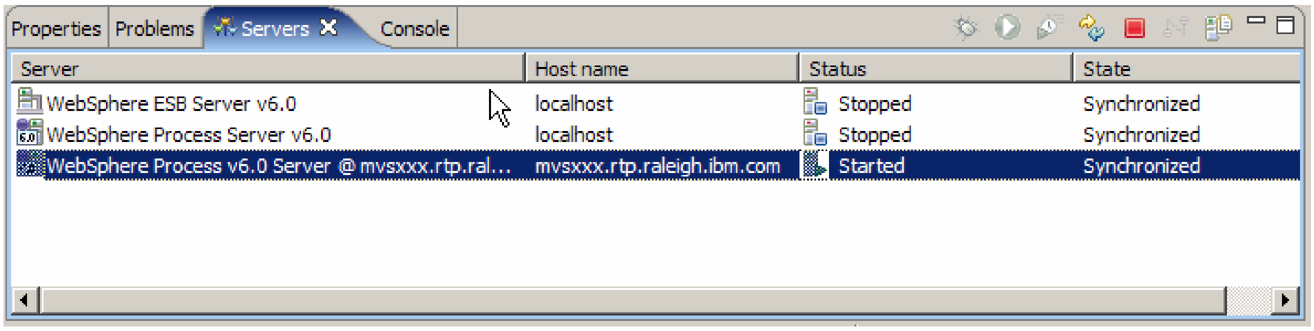
Network Deployment server name:

The server name is in the form of:
<cell name>/<node name>/<server name>
For example, localhost/localhost/server1.

Detect Click this button to detect the server type.

__ g. Click **Finish**.

__ h. The new server should be seen in the Server view.



Server	Host name	Status	State
WebSphere ESB Server v6.0	localhost	Stopped	Synchronized
WebSphere Process Server v6.0	localhost	Stopped	Synchronized
WebSphere Process v6.0 Server @ mvsxxx.rtp.ral...	mvsxxx.rtp.raleigh.ibm.com	Started	Synchronized

____ 5. Start the remote server if it is not already started. WebSphere Integration Developer does not support starting remote servers from the Server View.

__ a. From a command prompt, telnet to the remote system if needed:

‘**telnet <HOSTNAME> <TELNET_PORT>**’

userid : **<USERID>**

pw : **<PASSWORD>**

__ b. Navigate to the bin directory for the profile being used:

cd <WAS_HOME>/profiles/<PROFILE_NAME>/bin

__ c. Run the command file to start the server: **./startServer.sh <SERVER_NAME>**

__ d. Wait for status message indicating server has started:

```
ADMU3200I: Server launched. Waiting for initialization status.

ADMU3000I: Server cl1sr01 open for e-business; process id is 0000012000000002
```