WebSphere Enterprise Service Bus lab 1: Add mediation, mediation flow and message logger

What this exercise is about

The objective of this lab is to provide you with an initial understanding of how to create a mediation module and mediation flow in WebSphere Integration Developer V6.1. Then you will run a mediation module with Message Logger mediation on the WebSphere Enterprise Service Bus V6.1 server. You will also learn how to check message logger results from an Embedded Derby database using the 'Database Explorer' from the Data perspective.

Lab requirements

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

- WebSphere Integration Developer V6.1 with the WebSphere Enterprise Service Bus test server option installed

What you should be able to do

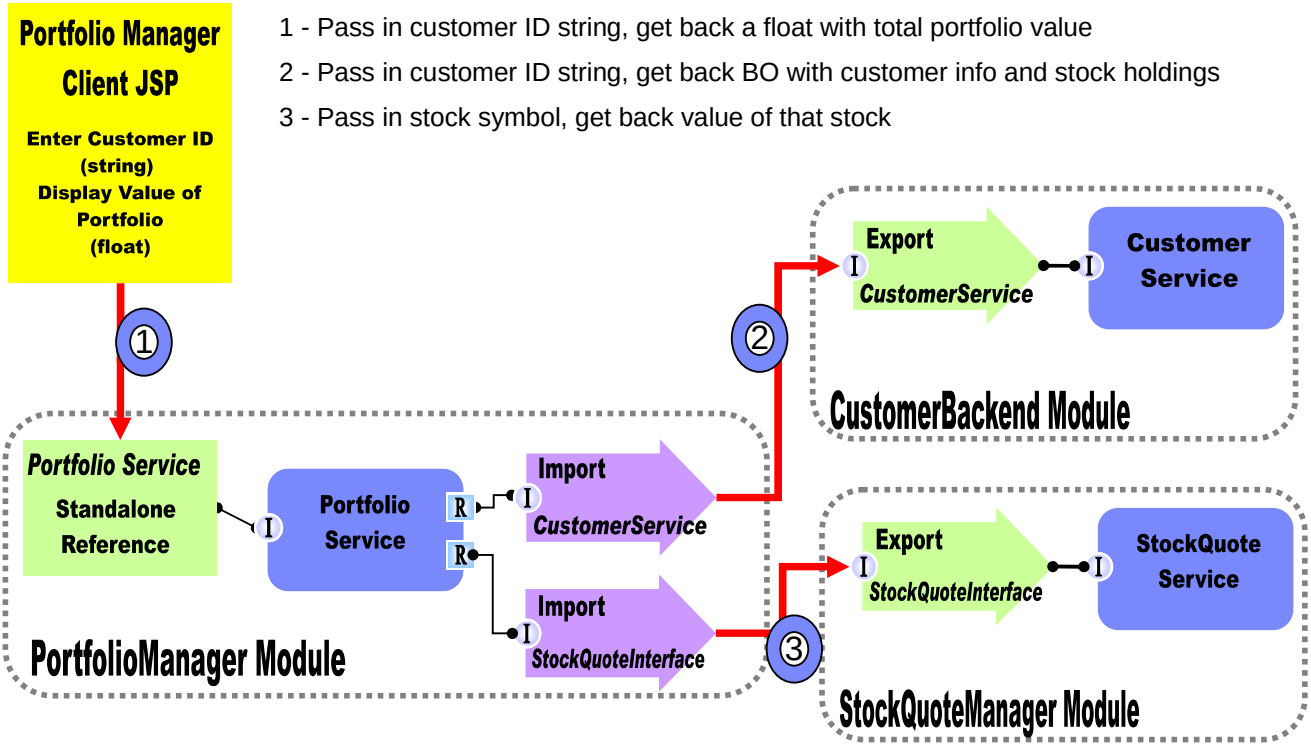
At the end of this lab you should be able to:

- Import project interchange files into the WebSphere Integration Developer V6.1 development environment
- Create and edit mediation modules and mediation flows
- Generate implementation and binding from the development environment
- Navigate the Properties View for mediation information
- Create and learn about a message logger primitive and test by running a JSP on the WebSphere Enterprise Service Bus V6.1 server
- View message logger results from the derby database using the database explorer

Introduction

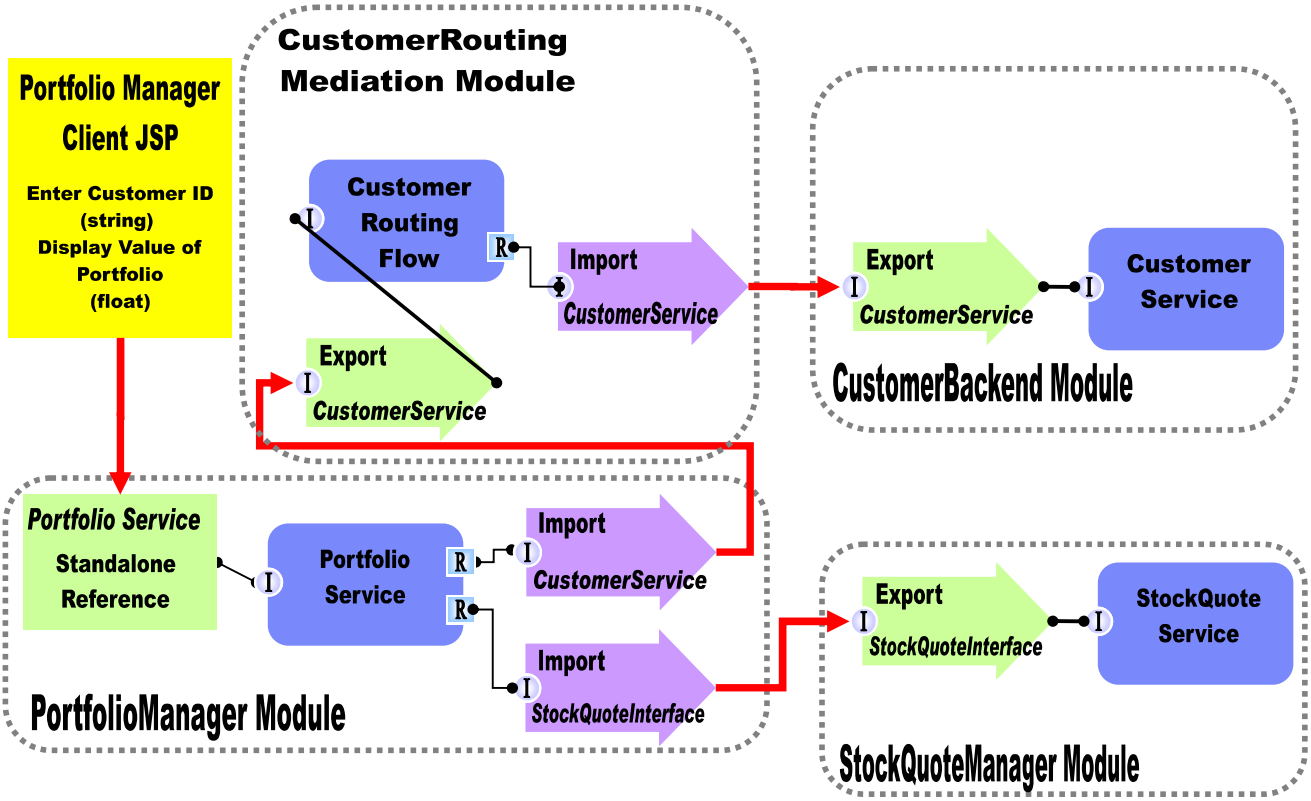
In this lab, you will start by importing a project interchange file that has sample SCA application which will provide the mediation module you will create a starting point. You will create a brand new mediation module and learning about what makes the mediation module different from any of the other business integration modules. You will also create a simple mediation flow with only one type of mediation, message logger mediation, to start off with. The job of the message logger mediation is to send data, whether the /body, /context, or /header, to a derby database where you will see what data was sent to the database. The rest of the mediations are added in Lab-2 of this lab series, and are debugged in Lab-3 to show you how the mediations work.

The sample application is a stock checking SCA application. The application has a JSP that allows a user to enter a customer ID which is fed into a call to the portfolio service by way of a stand-alone reference. From here the portfolio service will first call the customer service to get a customer's account information like what stocks they own and how many shares of that stock. Portfolio service will then call the StockQuote service to see the value of those stocks. Finally, the portfolio service returns a sum to the JSP. Here is a diagram of the project you are importing in this lab:



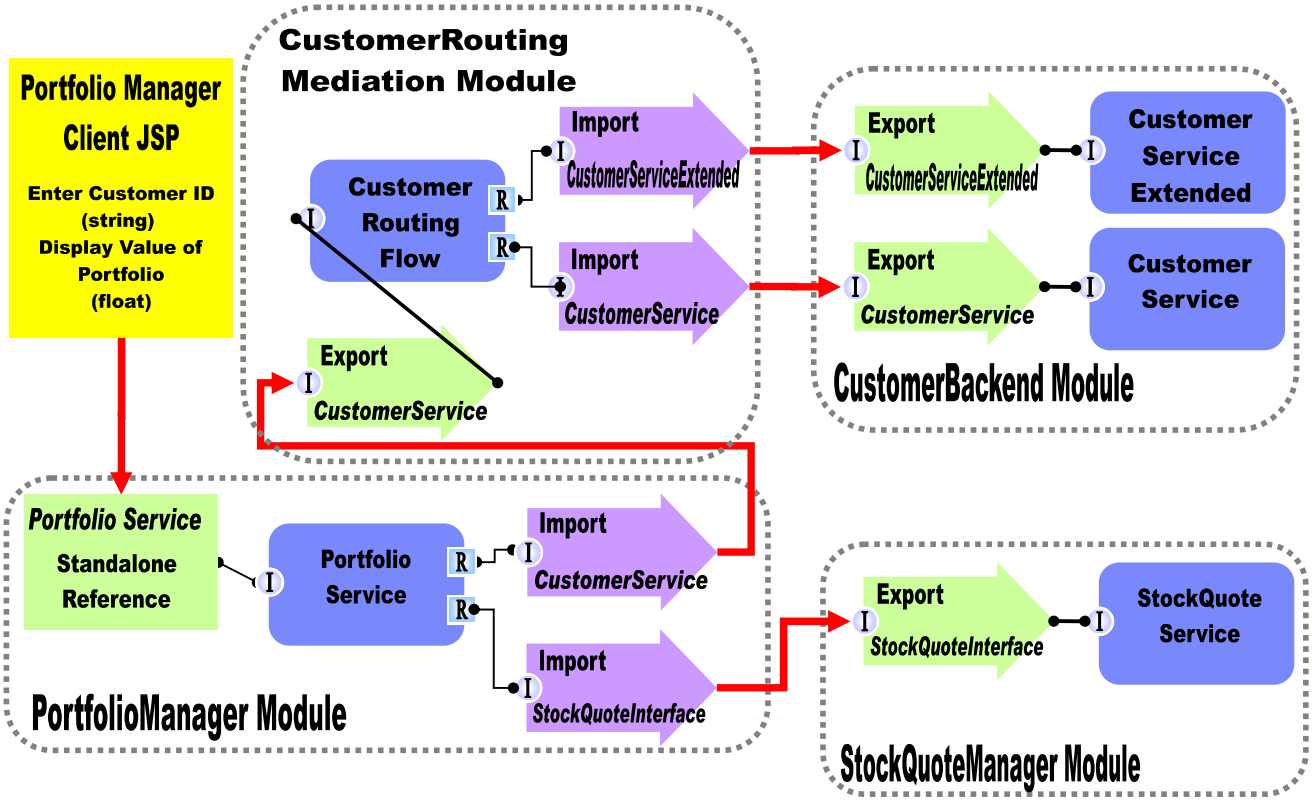
This original application is great, but the company that is running this application is going to need to distribute customer data across new multiple "CustomerBackend Modules and easily enhance end-user functionality.

In Lab-1 of this lab series, you will change the application from the above diagram to the one below. You will add a **CustomerRoutingMediation** Module between the **PortfolioManager** service and the **CustomerBackend** service to allow you access to message information being passed between them. You will create a mediation flow called **CustomerRoutingFlow** that will use a Message Logger mediation to store message information in a derby database.



In Lab2, you will add to the Message Logger mediation primitive with a couple others available in WebSphere Integration Developer V6.1. The main goal for Lab2 is to create another backend, **CustomerServiceExtended**, and use mediations to help decide which backend to use. You will first add a Custom Mediation. The Custom Mediation is a mediation primitive that is used when you want some custom functionality from mediation. For this custom mediation, it will use a Java snippet that will extract a two digit prefix from the customer ID and place 2 digit it in the transient context (a business object). Think of transient context as a scratchpad, a place for information to be stored in a message as it passes through a flow. You will then add a Database Lookup primitive that uses that two digit prefix from transient context as key to lookup a backend identifier to place into transient context. The Customer ID prefixes with 11, 22, 33, and 44 will go to the CustomerService backend. Customer ID prefixes with 55, 66, 77, 88, and 99 will go to a **CustomerServiceExtended** backend.

To determine the message routing based on backend identifier, you will add a Message Filter primitive. The old backend will go directly to the callout for CustomerService and the new backend will go to the XSL Transformation. The XSL Transformation will transform the CustomerService business object (BO) into the **CustomerServiceExtended** business object, and then pass the newly formed message to the callout for **CustomerServiceExtendedPartner** instead of the **CustomerServicePartner**. Notice that this is just the request. There is also a flow for the response. On the response side, the **CustomerServicePartner** callout response will go directly to the CustomerService input response. However, the **CustomerServiceExtended** callout response will need to go back through XSL Transformation mediation in order to transform the response body from the **CustomerServiceExtended** business object back to the CustomerService business object. Feeding the **CustomerServiceExtended** business object to application ends in an error because the application only knows how to work with the CustomerService business object. This is the reason why you need an XSL Transformation mediation primitive; to map one business object to another. The diagram will now look like the one below.



In Lab-3 of this lab series, you will use the Visual Debugger to step through the application in order to see what is happening behind the scenes.

Exercise instructions

Some instructions in this lab are Windows operating system specific. If you plan on running WebSphere Integration Developer on a Linux operating system you will need to run the appropriate commands and use appropriate files for Linux. The directory locations are specified in the lab instructions using symbolic references, as follows:

Reference Variable	Windows Location	AIX®/UNIX® Location
<WID_HOME>	C:\Program Files\IBM\WID61	/opt/IBM/WID61
<ESB_PROFILE_HOME>	<WID_HOME>\pf\esb	<WID_HOME>/pf/esb
<LAB_FILES>	C:\Labfiles61\WESB\Lab1	/tmp/Labfiles61/WESB/Lab1
<WORKSPACE>	C:\Labfiles61\WESB\Lab1\workspace	/tmp/Labfiles61/WESB/Lab1/workspace
<SOLUTION>	C:\Labfiles61\WESB\Lab1\solution	/tmp/Labfiles61/WESB/Lab1/solution

Windows user note: When directory locations are passed as parameters to a Java™ program such as EJBdeploy or wsadmin, it is necessary to replace the backslashes with forward slashes to follow the Java convention. For example, C:\LabFiles61\ is replaced by C:/LabFiles61/

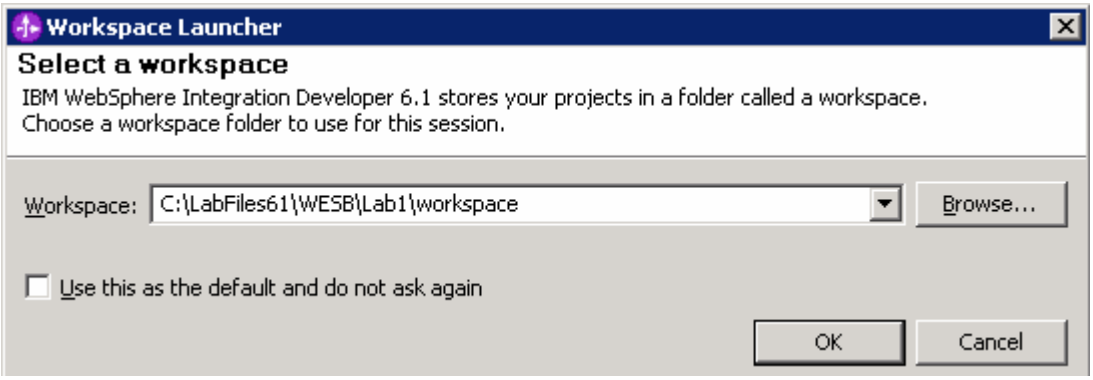
Reference variable	Example: Remote Windows test server location	Example: Remote z/OS® test server location	Input your values for the remote location of the test server
<SERVER_NAME>	server1	sssr011	
<WAS_HOME>	C:\Program Files\IBM\WebSphere\AppServer	/etc/sscell/AppServer	
<HOSTNAME>	localhost	mvsxxx.rtp.raleigh.ibm.com	
<SOAP_PORT>	8880	8880	
<TELNET_PORT>	N/A	1023	
<PROFILE_NAME>	AppSrv01	default	
<USERID>	N/A	ssadmin	
<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: Prepare environment for lab

In this section of the lab, you will import all the projects inside the **WPIv61_ESB_StartLab1_PI.zip** project interchange file into your workspace. Remember this is the sample SCA application that you are going to add ESB specific mediations to.

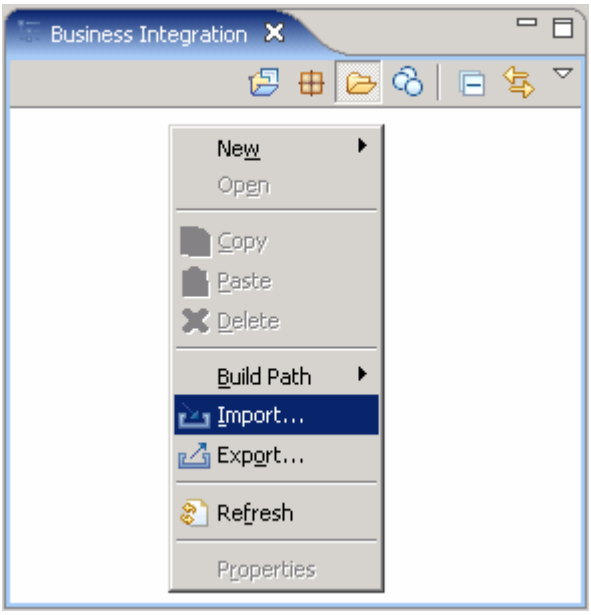
____ 1. Start WebSphere Integration Developer V6.1 with a workspace location of **<WORKSPACE>**



____ 2. Click the curved arrow at top right to **Go to the Business Integration perspective**

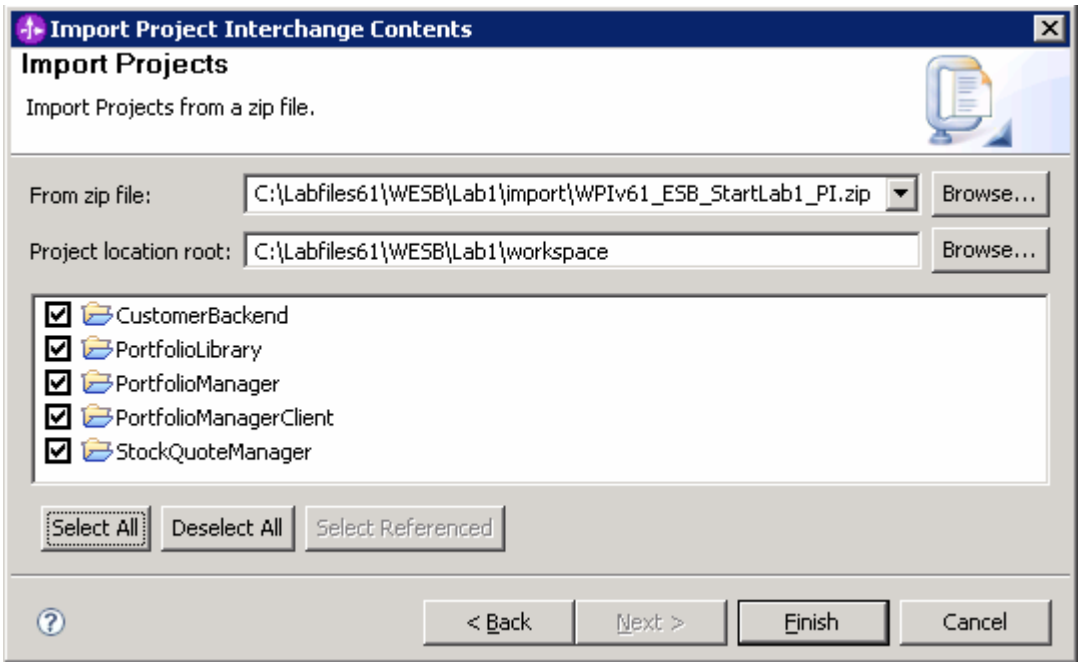
____ 3. Import Project Interchange file, **WPIv61_ESB_StartLab1_PI.zip**, into the development environment

__ a. Right-click inside **Business Integration View** (top left view in the Business Integration Perspective) and select **Import** from the pop-up menu



__ b. Select **Project Interchange** from the list. Click **Next**

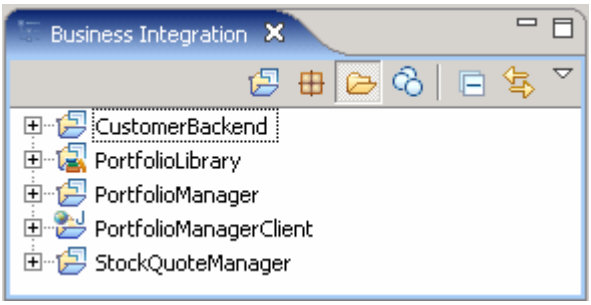
__ c. Click the **Browse** button for "**From zip file**" and navigate to **<LAB_FILES>/import/WPIv61_ESB_StartLab1_PI.zip** and hit **Open**



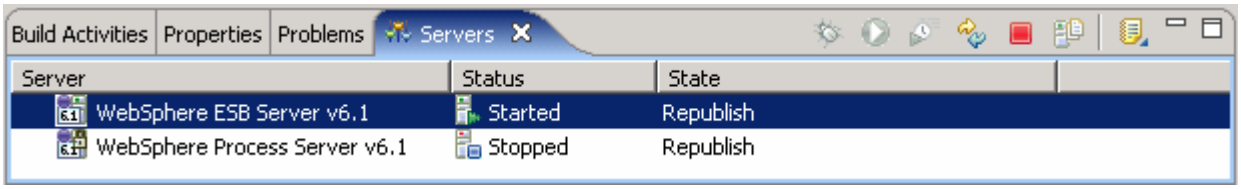
__ d. Click the **Select All** button to select all the check boxes for the projects listed.

__ e. Click **Finish** button (projects are imported and auto-build will run).

__ f. Verify you have **CustomerBackend**, **PortfolioLibrary**, **PortfolioManager**, **PortfolioManagerClient** and **StockQuoteManager** modules listed in the Business Integration view.



__ g. Verify the '**WebSphere ESB Server V6.1**' listed in your Servers view

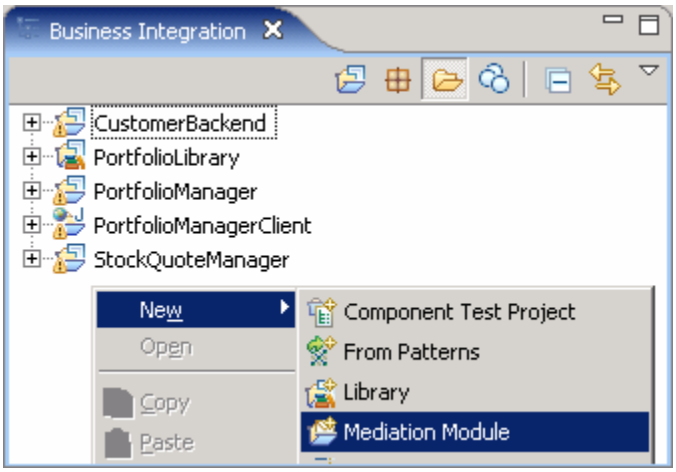


Part 2: Create mediation module and mediation flow

In this section of the lab, you will creat a new mediation module and a mediation flow. You can only have one mediation module for each deployable project. Every mediation module will have a mediation flow inside of it to connect and control services and mediations.

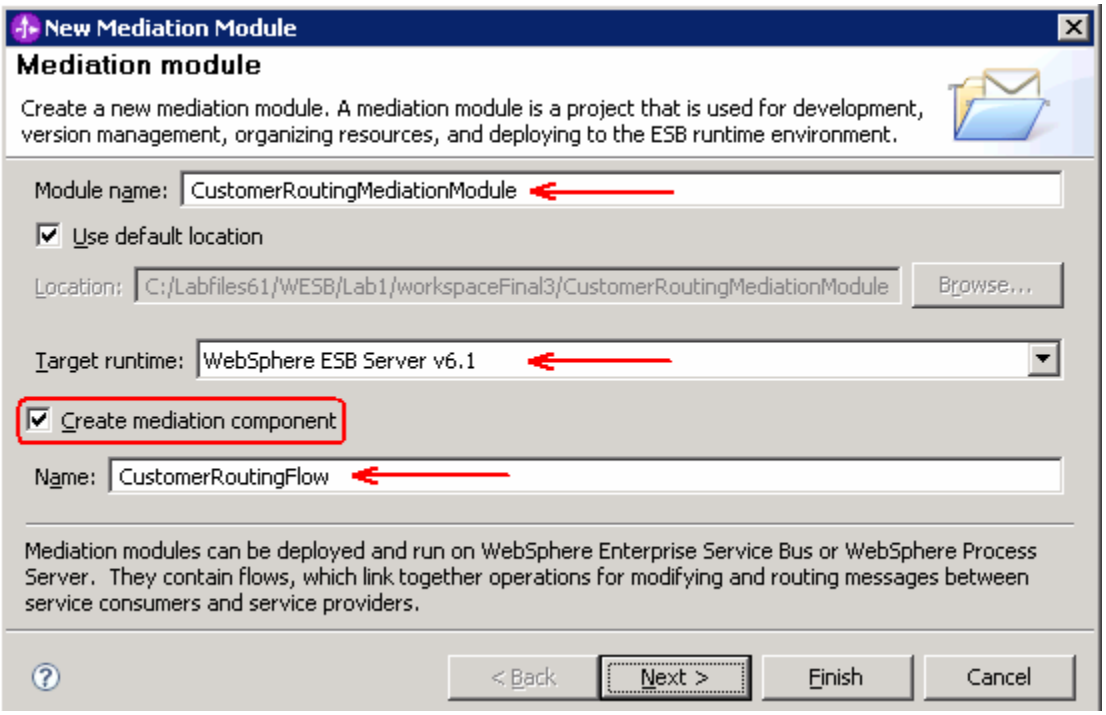
____ 1. Create a new mediation module.

__ a. Right-click in Business Integration view and select **New → Mediation Module** from the pop-up menu



__ b. In the '**New Mediation Module**' panel, enter the parameters listed below:

- Module Name: **CustomerRoutingMediationModule**
- Ensure the target runtime is '**WebSphere ESB Server V6.1**'
- Select the check box for '**Create mediation component**'
- Change the name of the module from '**CustomerRoutingMediationModule**' to '**CustomerRoutingFlow**'



__ c. Click **Next**

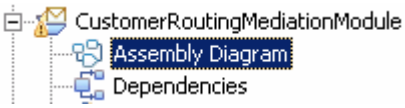
__ d. In the next '**Select required libraries**' panel, select the check box for '**PortfolioLibrary**' to add it as a library project

__ e. Click **Finish**

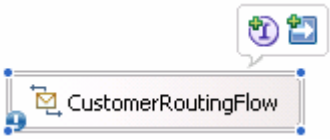
____ 2. Edit the Assembly Diagram to prepare Mediation Module

__ a. Open Assembly Diagram and update Mediation Flow

1) Expand **CustomerRoutingMediationModule** in the Business Integration view and double click on **Assembly Diagram** (Assembly Diagram) to open it in an Assembly Diagram editor



2) Notice that a mediation flow is created with the name '**CustomerRoutingFlow**'



3) Hover over **CustomerRoutingFlow** and click the "I" icon ( ) that appears to **add an interface**

4) Select **CustomerService** from the list of matching interfaces. Click **OK**

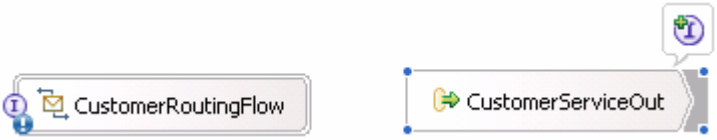
__ b. Add an Import to **CustomerRoutingFlow** in Assembly Diagram

1) Drag and drop an import on the right side of **CustomerRoutingFlow**

a) Click on **Import** icon from Assembly Diagram tray (Import)

b) Click or drag import to the right side of **CustomerRoutingFlow**

2) Change default import name from Import1 to **CustomerServiceOut**



3) Hover over import or click on import to make the add interface icon appear

Click "I" icon ( ) to add an interface

4) Select **CustomerService** from the list of matching interfaces. Click **OK**

__ c. Add an Export to **CustomerRoutingFlow** in Assembly Diagram

1) Drag and drop an **Export** on the left side of the **CustomerRoutingFlow**

a) Click on **Export** icon from Assembly Diagram tray (Export)

b) Click or drag icon to left side of **CustomerRoutingFlow**

2) Change name from 'Export1' to **CustomerServiceIn**

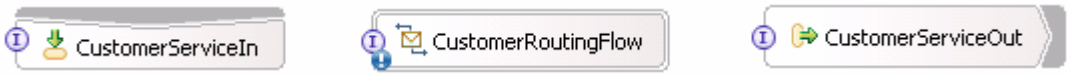
3) Hover over export or click on Export to make the add interface icon appear

Click "I" icon ( ) to add an interface

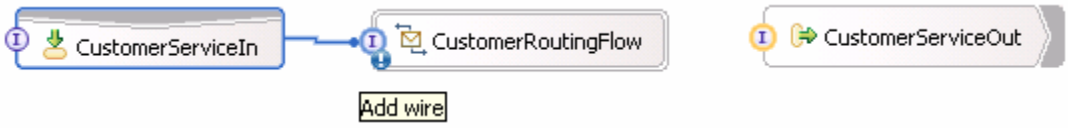
4) Select **CustomerService** from the list of matching interfaces. Click **OK**

__ d. Wire components together

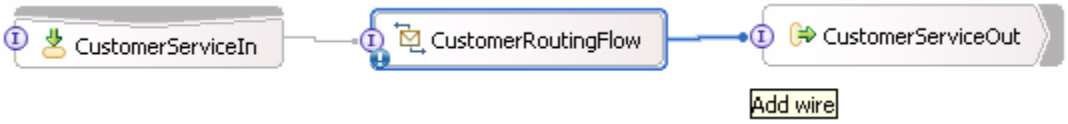
1) Your Assembly Diagram should look like the one below



2) Click on **CustomerServiceIn** and drag a wire to **CustomerRoutingFlow**



3) Click on CustomerRoutingFlow and drag a wire to CustomerServiceOut



4) Click **OK** to any pop-up windows

__ e. Generate SCA bindings for Import and Exports.

1) Right-click on **CustomerServiceOut** and select **Generate Binding → SCA Binding**

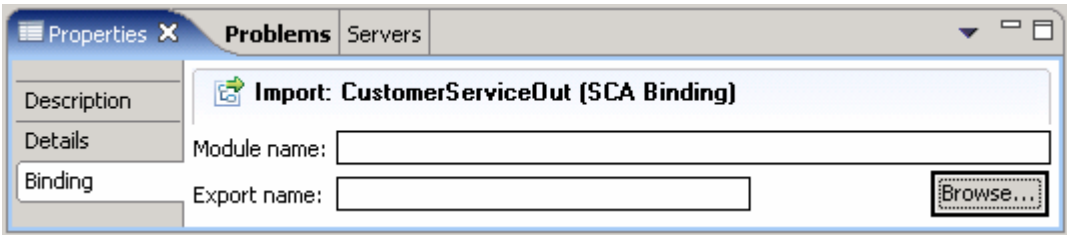
2) Right-click on CustomerServiceIn and select **Generate Binding → SCA Binding**

3) This is what your Assembly Diagram should now look like

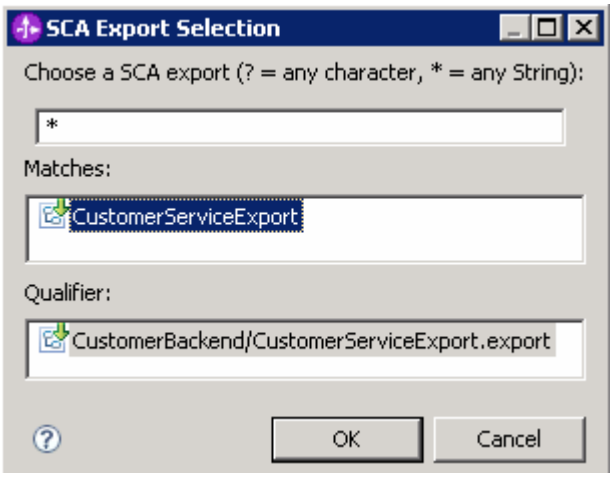
__ f. Update binding for CustomerServiceOut

1) Select the **Import** 'CustomerServiceOut' from the Assembly Diagram

2) Open Properties view to **Binding tab**

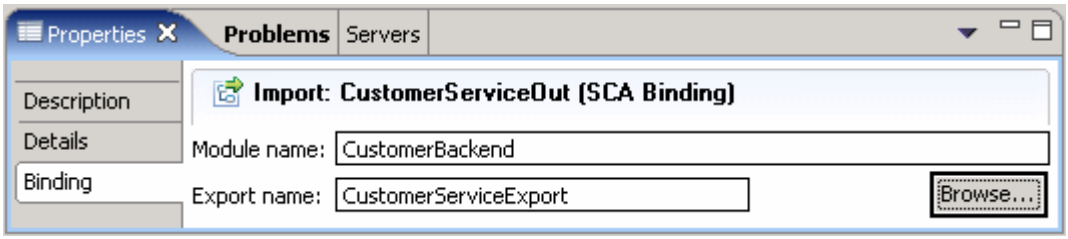


3) Click the **Browse** button and select **CustomerServiceExport** from the SCA Export selection panel



4) Click **OK**

5) This is how the binding information for the CustomerServiceOut import looks like



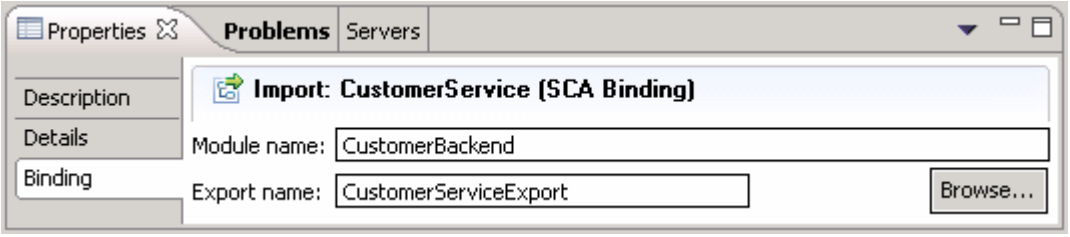
6) Save all work by **File → Save All** or **Crtl + Shift + S**

__ g. Update import binding '**CustomerService**' for **PortfolioManager** module

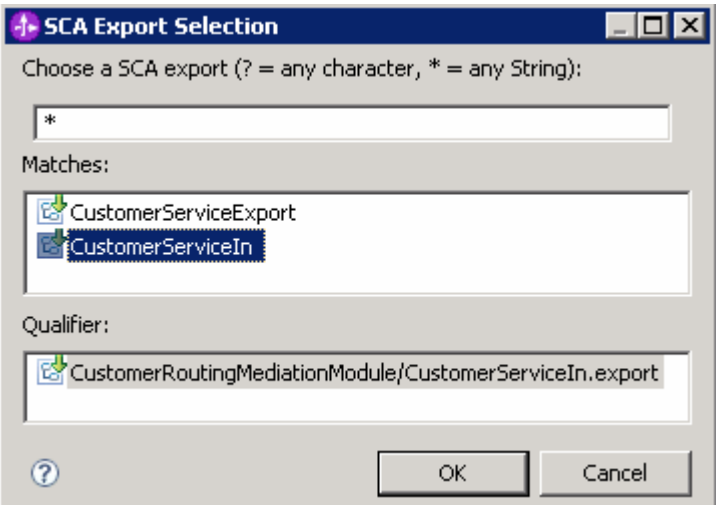
1) Expand **PortfolioManager** in the Business Integration view and double click on **Assembly Diagram** (Assembly Diagram) to open it in an Assembly Diagram editor

2) Select the import **CustomerService**

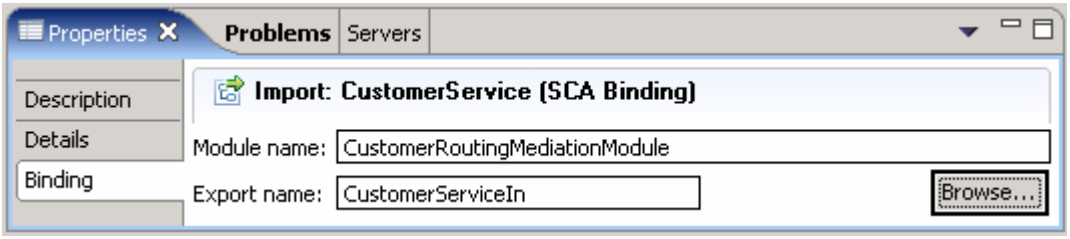
3) Open **Properties View** with the CustomerService import highlighted.
Select the **Binding tab** inside Properties View



4) Click **Browse** button and select **CustomerServiceIn** (instead of **CustomerServiceExport** that the binding is set to)



5) Click **OK**



6) Save the Assembly Diagram. **File → Save** or **Ctrl + S**

You have now successfully inserted the mediation flow between the PortfolioManager module and the CustomerBackend module. Redirecting PortfolioManager to **CustomerServiceIn** allows the message to be operated on by the Mediation Flow and then passed to the **CustomerBackend** through the import **CustomerServiceOut**.

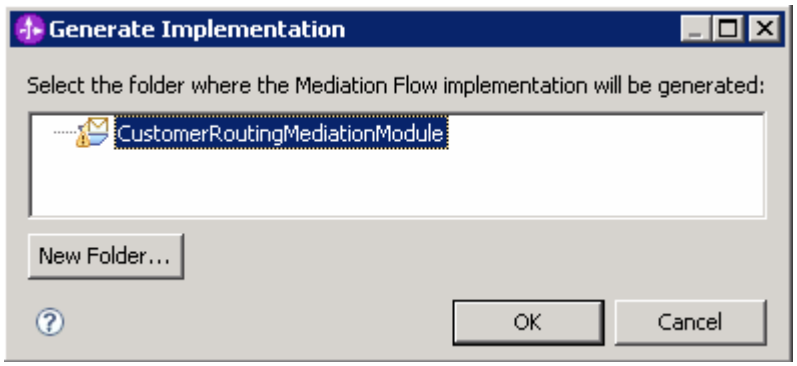
Part 3: Generate mediation flow implementation and create a mediation

In this section you will generate the implementation for the Mediation Flow. Afterwards, you will create only one type of mediation; that is, the message logger mediation. It is a simple mediation that sends data (whether the /body, /context, or /header part of a message) to a derby database. In this section, you will also learn how to open a derby database and view the data captured by the message logger.

____ 1. Generate implementation for CustomerRoutingFlow

__ a. Expand **CustomerRoutingMediationModule** in the Business Integration view and double click on **Assembly Diagram** (Assembly Diagram)

__ b. Right-click on **CustomerRoutingFlow** in the Assembly Diagram for CustomerRoutingMediationModule and select **Generate Implementation** from the pop-up menu. Select **CustomerRoutingMediationModule** as the folder from the **Generate Implementation** panel



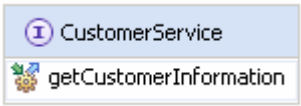
__ c. Click **OK**.

Note: You can create a mediation flow by right-clicking in the business integration view and select New → Mediation Flow. However, generating the implementation from the assembly diagram creates a new Mediation Flow with the name specified in the assembly diagram 'CustomerRoutingFlow'.

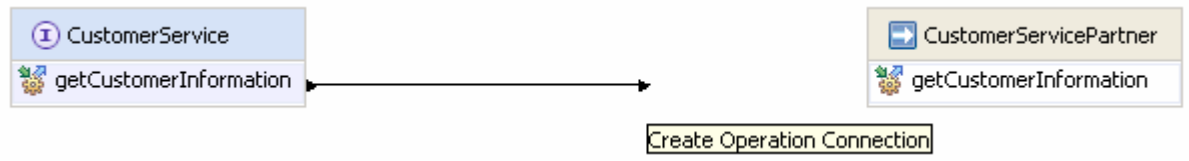
__ d. The **Mediation Flow Editor** will open after generating the mediation CustomerRoutingFlow implementation.

____ 2. Connect the **source operation** to **target operation** in Operation Connections view.

__ a. Click on getCustomerInformation section of the **CustomerService** Interface on the left side of the Operation Connections view.



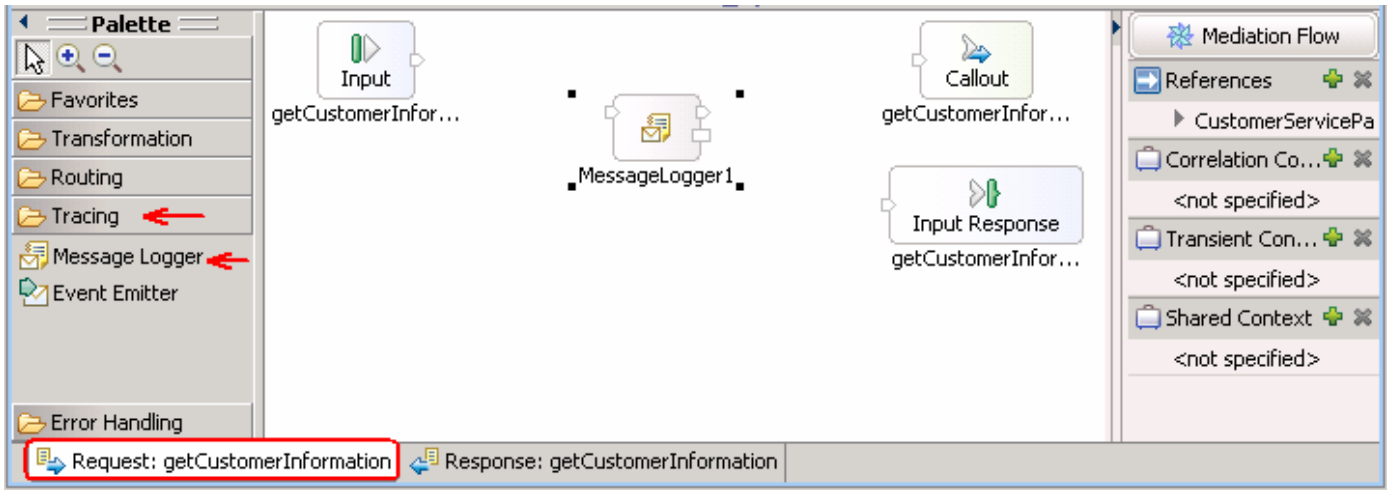
__ b. Drag to the getCustomerInformation section of the **CustomerServicePartner** on the right side of the Operation Connections view and unclick. There should be a **black line** from CustomerService to CustomerServicePartner



__ c. Click on the **black line** or the **getCustomerInformation** section of the CustomerService Interface (left-hand side)

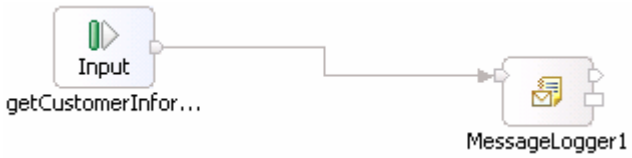
____ 3. Add **Message Logger** mediation to Mediation Flow diagram

__ a. In **Mediation Flow View** (middle), click to expand '**Tracing**' from the palette tray on the left side, then click to select the '**Message Logger**' icon (Message Logger) and finally click to drop it in the middle of the diagram as shown below:

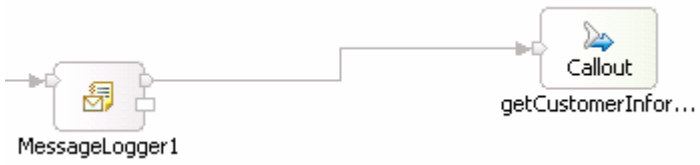


____ 4. Wire **input** and **output terminals** for the **Request**

__ a. Click on the output terminal **"out"** of the getCustomerInformation: CustomerService **Input** node (on top, left side of mediation flow diagram) and drag to the **input terminal** of the message logger



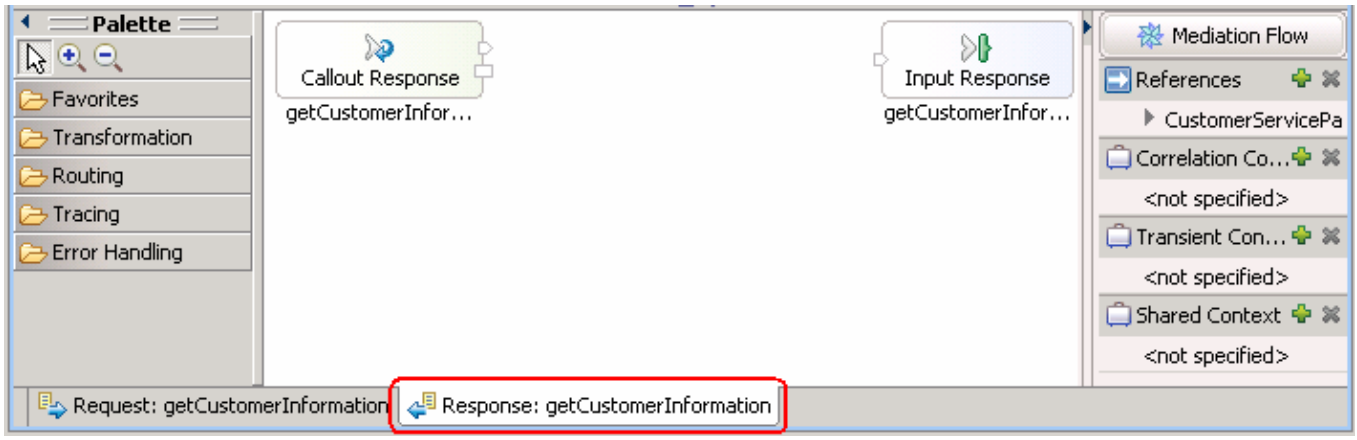
__ b. Click on the **output terminal "out"** of the Message Logger mediation (middle) and drag to the **input terminal** of the getCustomerInformation: CustomerServicePartner **Callout** node (top, right side)



____ 5. Wire **input** and **callout terminals** for the **Response**

At the bottom of the Mediation Flow view, there are two tabs. By default you will see the Request information when enacting the mediation flow view. However, now you need to set up the Response side of the mediation flow. You can set mediations on both sides of the flow

__ a. Select the **Response: getCustomerInformation** tab



__ b. Click on the output terminal '**out**' of the 'getCustomerInformation: CustomerServicePartner' **Callout Response** node (on the top, left side of mediation flow diagram) and drag to the input terminal of the 'getCustomerInformation: CustomerService' **Input Response** node

__ c. Save all with **Ctrl + Shift +S** or by navigating to **File → Save All**

____ 6. Investigate Message Logger

__ a. Click back on **Request tab** at bottom of Mediation Flow Editor

__ b. Click on **MessageLogger1** mediation

__ c. Then click on the **Properties View** (lower right quadrant) while 'MessageLogger1' is selected

__ d. Click on **Details tab** of Properties view and notice that WebSphere Enterprise Service Bus came with the **data source** to derby already set up (jdbc/mediation/messageLog)

__ e. The '**Root**' section sets what level of information you are sent to the database from the message (body, context, or headers). Leave as **/body**

__ f. Keep the transaction mode at '**Same**'. Notice here you have the option of creating a '**New**' transaction

Part 4: Test message logger mediation

In this section, you will run the mediation and use Data view to see what data was populated into the derby database.

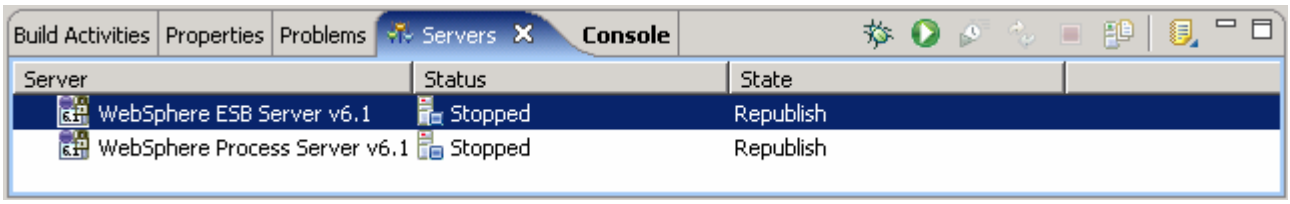
____ 1. Start **WebSphere ESB Server** and **add modules** to server

If using a remote testing environment, follow the instructions in '**Task: Adding remote server to WebSphere Integration Developer test environment**' at the end of this document.

If using a local testing environment:

__ a. Open **Servers View**

__ b. Select '**WebSphere ESB Server V6.1**' and click **Start button** ( )



__ c. This will take some time. Wait for the server to start

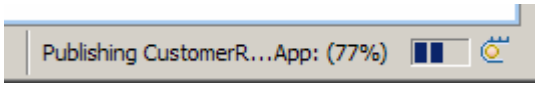
____ 3. Add **projects** to WebSphere Enterprise Service Bus Server (once the ESB Server is started, not before).

__ a. In Servers view, right-click on '**WebSphere ESB Server V6.1**' and select '**Add and Remove Projects...**'

Note: The ESB server you are using is configured with an ESB profile that is part of the installation and not part of the workspace. Therefore, if you have projects deployed to the server from a different workspace, there may be some naming conflicts or other problems. If this occurs, open the Administrative Console and uninstall the existing projects before adding new projects. That should avoid any potential errors.

__ b. Click **Add-All>>** button to move all projects to server and click **Finish** button

__ c. Wait for the deployment to finish. While the project is deploying you will see something like the following in the lower right corner of WebSphere Integration Developer.



__ d. Click the Web browser icon in the WebSphere Integration Developer tools panel to launch a browser

__ e. Enter http:// <HOSTNAME>:<PORT>/PortfolioManagerClient/index.jsp . Where hostname is the name of the system where the WebSphere Enterprise Service Bus server is located. Port is the **WC_defaulthost** port of the WebSphere Enterprise Service Bus profile.

Ex: http://localhost:9080/PortfolioManagerClient/index.jsp

Note: You can get the **WC_defaulthost** port by going to **serverindex.xml** file in <WID_HOME>\pf\esb\config\cells\esbCell\nodes\esbNode. Where WID_HOME is the location where WebSphere Integration Developer is installed

Ex: <ESB_PROFILE_HOME>\config\cells\esbCell\nodes\esbNode

__ f. Enter **2222222** (2 seven times) in text input box and click **Submit** to get a response displayed to the JSP and to the Console View of WebSphere Integration Developer. You should see the result as "The value is: 28500.0"

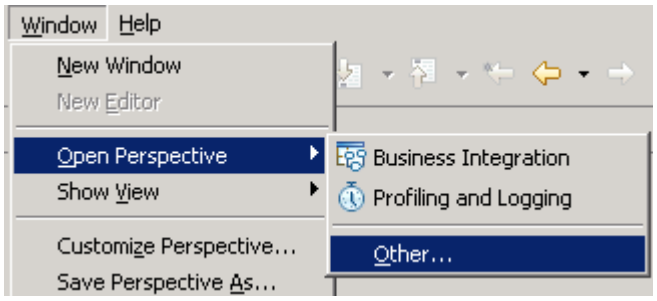
__ g. Close browser

__ h. Stop 'WebSphere ESB Server V6.1' by highlighting the WebSphere ESB Server v6.1 in the Servers View and click **Stop** button ( )
(For your information, you cannot view data in a derby database when server is running.)

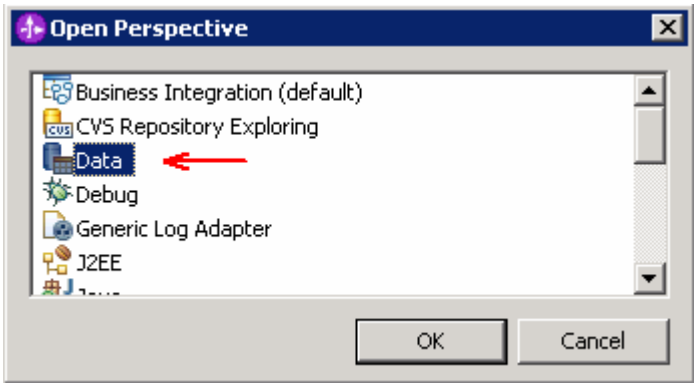
____ 4. Verify data was sent to database by MessageLogger1 using the '**Database Explorer**'

__ a. Open the '**Data Perspective**' to view the Database Explorer

__ b. From the '**Window**' menu, select '**Open Perspective → Other**'

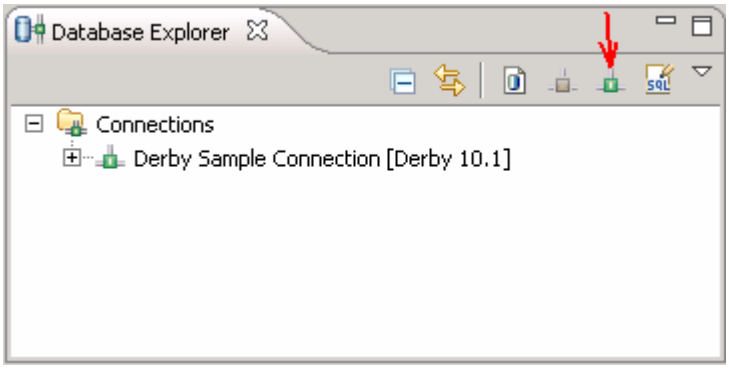


__ c. The '**Open Perspective**' panel opens



__ d. Select '**Data**' and click **OK**

__ e. The '**Database Explorer**' shows up at the bottom left corner frame of the WebSphere Integration Developer canvas



__ f. Select the '**New Connection**' icon ( ) to launch the '**New Connection**' wizard. Follow the inspections below to connect to an existing derby database:

- Expand '**Derby**' and the select '**10.1**' from the '**Select a database manager**' list
- Select '**Derby Embedded JDBC Driver**' from the '**JDBC Driver**' list
- Click the **Browse** button for '**Database location**', locate and select the appropriate database under the '**Connection URL details**' section

---

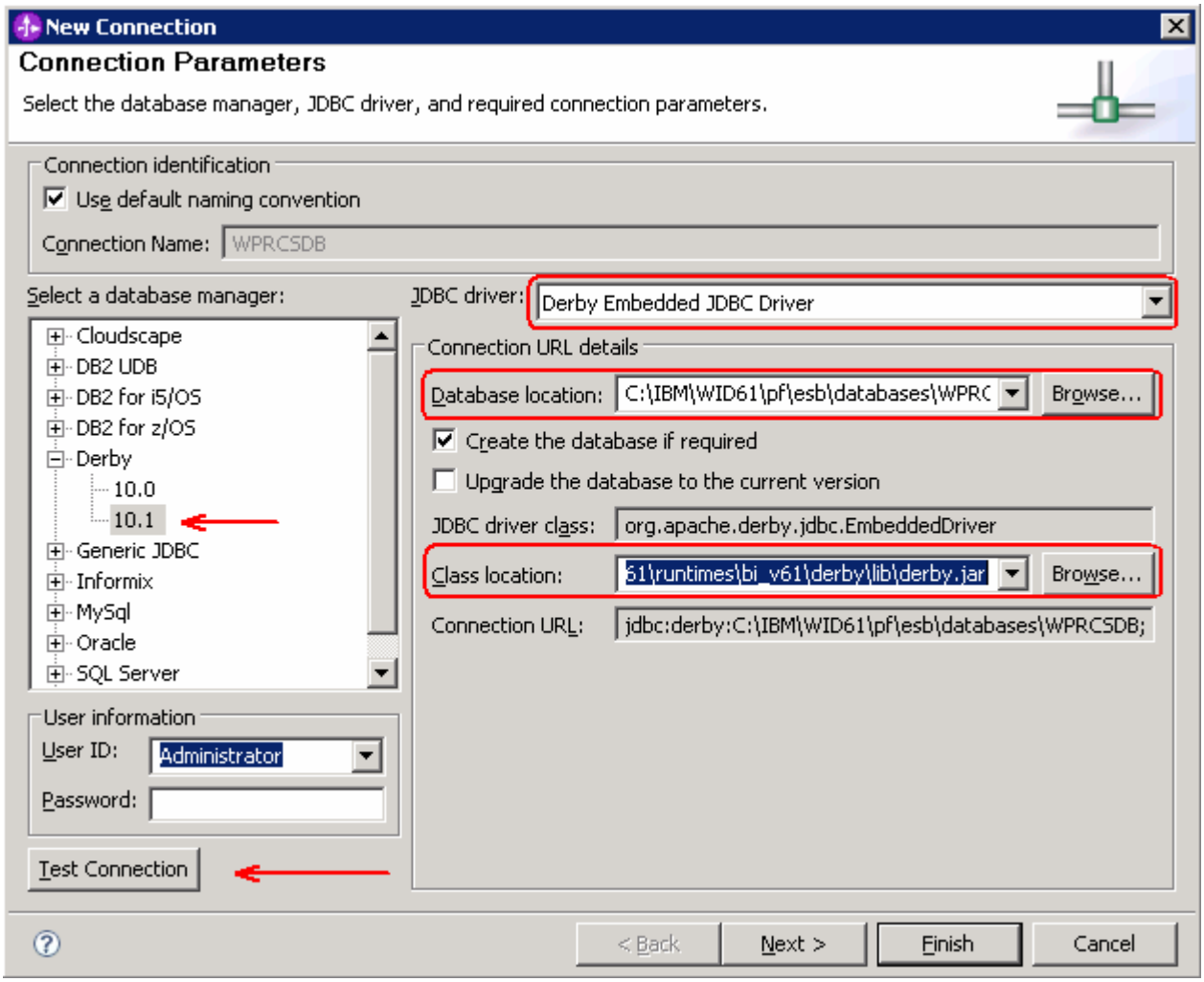
Note: Derby databases are located at **<ESB_PROFILE_HOME>\databases** for an Enterprise Service Bus profile. By default, the message logger table is created under **WPRCSDB** database.

- Click the **Browse** button for '**Class location**', locate and select the '**derby.jar**' library under the '**Connection URL details**' section

---

Note: Derby databases libraries are located at **<WID_HOME>\runtimes\bi_v61\derby\lib** directory.

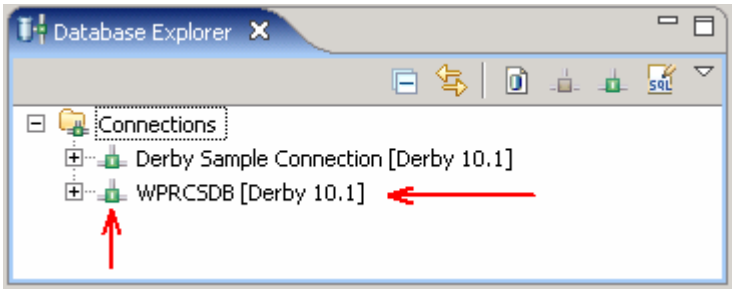
- Enter the '**User Information**' if necessary



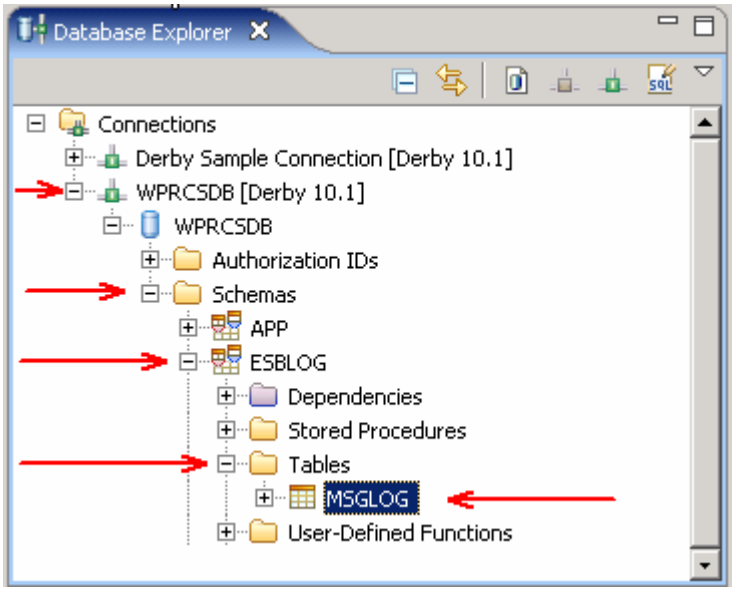
__ g. Click the '**Test Connection**' button. Ensure the connection is successful

__ h. Click **Finish**

__ i. You should see the '**WPRCSDB**' database listed in the 'Database Explorer' as shown below:

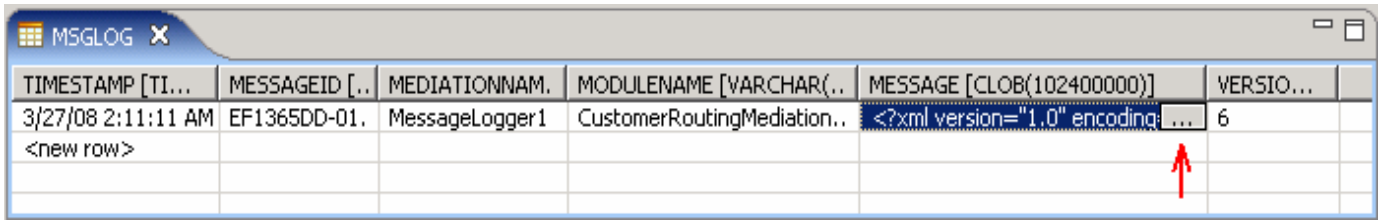


__ j. In the '**Database Explorer**', expand **WPRCSDB → Schemas → ESBLOG → Tables**' and select the '**MSGLOG**' table as shown below:



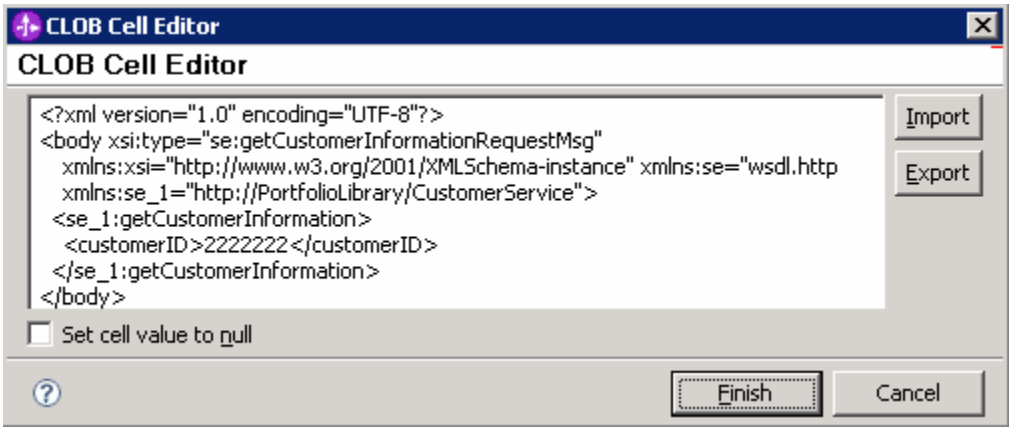
__ k. Now, right click the '**MSGLOG**' table and then select '**Data → Edit**' from the context menu

__ l. The '**MSGLOG**' table opens in an editor, where you can view the data populated as shown below:



__ m. Scroll to the right and click on the table cell below **MESSAGE** column. This action shows up a button (...) to the right of the cell

__ n. Click the button to view what the message logger saved from the /body of the message



__ o. You can either import or export messages in XML format. Depending on your action click '**Finish**' or '**Cancel**'

____ 5. You are done with this exercise. Go ahead and clean up the workspace (steps below) for future work

Part 5: Save work and clean up server

Export project as Project Interchange file. Switch to the '**Business Integration**' perspective (**Window → Open Perspective → Other** and then select '**Business Integration**')

____ 1. In WebSphere Integration Developer, Navigate to **File → Export**

____ 2. Select Project Interchange

____ 3. Out of all the projects listed, you only need to add a check next to **6 projects**
CustomerBackend
CustomerRoutingMediationModule
PortfolioLibrary
PortfolioManager
PortfolioManagerClient
StockQuoteManager

All other projects are generated upon import of the project interchange

____ 4. Save in C:/LabFiles61/WESB/Lab1/

____ 5. Name the project interchange WPIv61_ESB_FinishedLab1_PI.zip

What you did in this exercise

In this lab, you were provided with an initial understanding of how to create a mediation module and mediation flow in WebSphere Integration Developer V6.1. You then learned how to run the Message Logger mediation module on the WebSphere Enterprise Service Bus V6.1 server, and check for results in a derby database.

Solution instructions

____ 1. Import **Solution** Project Interchange file

__ a. With a blank workspace in WebSphere Integration Developer, Go to **File → Import → Project Interchange**

__ b. Click on top Browse button and navigate to **<LAB_FILES>/solution/WPIv61_ESB_Lab1_PI_Solution.zip**

__ c. Select all the 6 project listed

__ d. Click **Finish** button

____ 2. Start with '**Part 4: Test Message Logger Mediation**'

Task: Adding remote server to WebSphere Integration Developer test environment

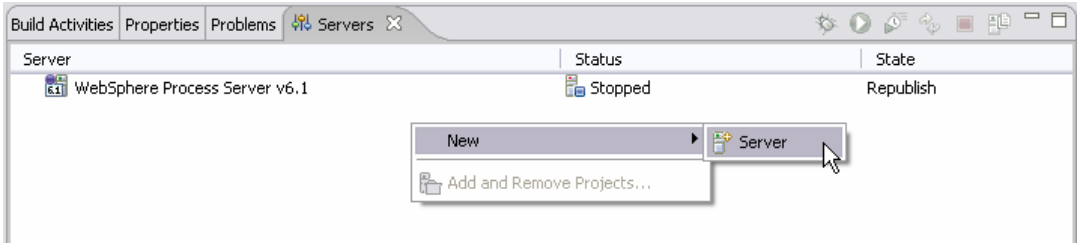
This task describes how to add a remote server to the WebSphere Integration Developer Test environment. This example uses a z/OS machine.

Create a new remote server.

____ 1. Define a new remote server to WebSphere Integration Developer.

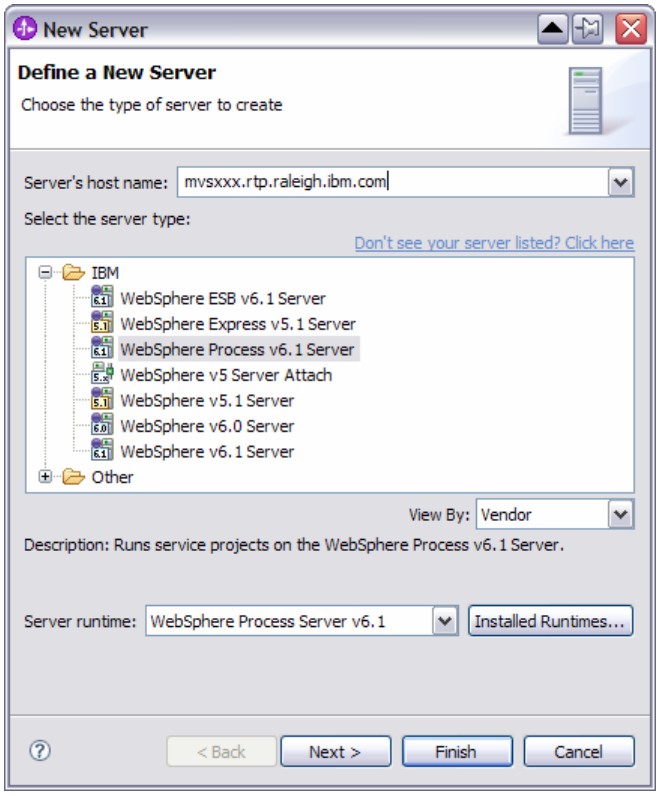
____ 2. Right click on the background of the **Servers** view to access the pop-up menu.

____ 3. Select **New → Server**.



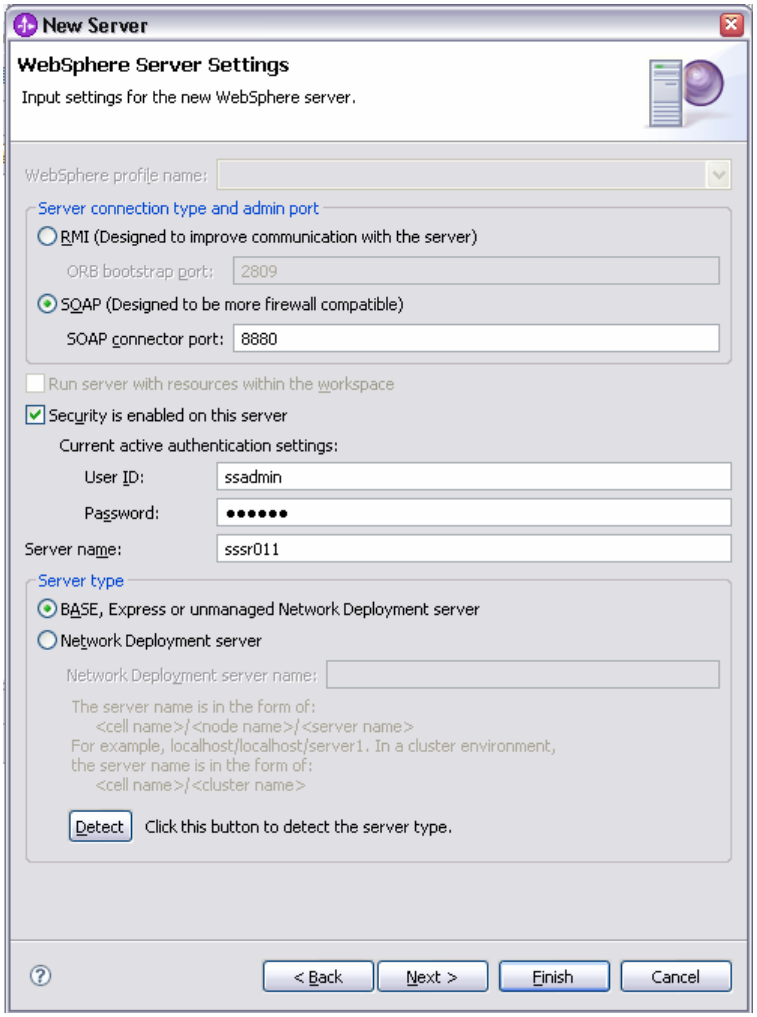
____ 4. In the New Server dialog, specify the remote server's host name, **<HOSTNAME>**.

____ 5. Ensure that the appropriate server type, '**WebSphere Process v6.1 Server**' or '**WebSphere ESB v6.1 Server**', is highlighted in the server type list



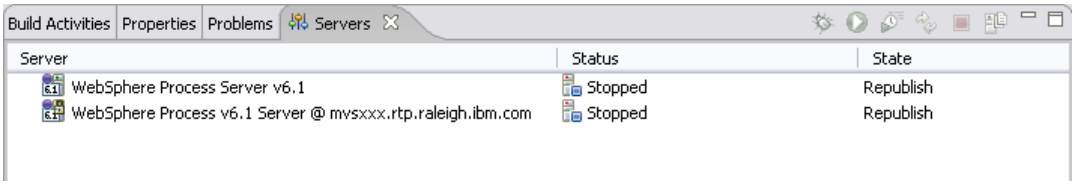
____ 6. Click **Next**

____ 7. On the WebSphere Server Settings page, leave the radio button for **SOAP** selected, changing the **SOAP connector port** to the correct setting (**<SOAP_PORT>**). If security is on in your server, check the box for **'Security is enabled on this server'** and input **<USERID>** for the user ID and **<PASSWORD>** for the password.



____ 8. Click **Finish**.

____ 9. The new server should be seen in the Server view.



____ 10. 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>**

password : **<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
```