IBM WebSphere® Enterprise Service Bus V6.0.1 – Lab Exercise

WebSphere Enterprise Service Bus Lab 2 – Add Custom Mediation, Database Lookup, Message Filter and XSL Transformation

What this exercise is about

The objective of this lab is to create another backend, CustomerServiceExtended, and use mediations to help decide which backend to use. Read the introduction below for a bigger picture of what this lab is about.

Lab Requirements

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

- WebSphere Integration Developer V6.0.1 with the WebSphere Enterprise Service Bus test server option installed.

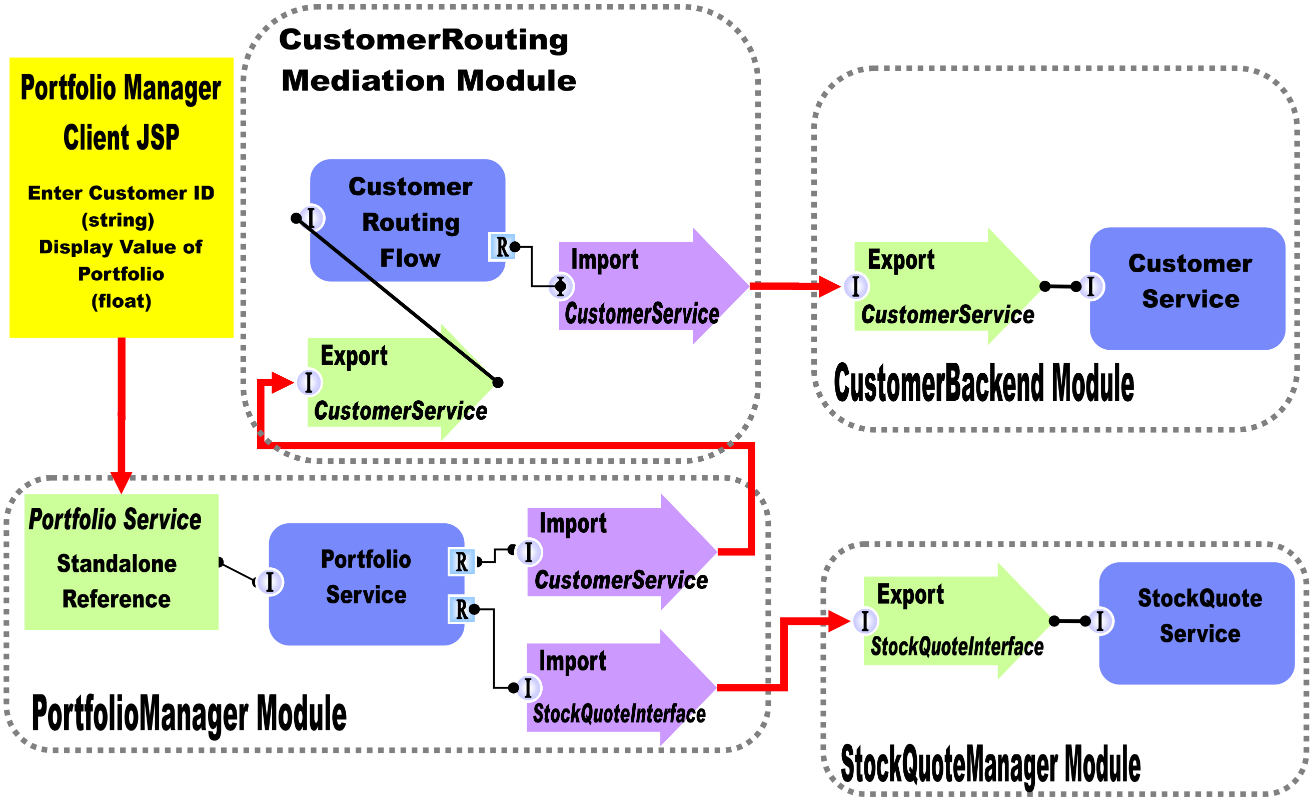
What you should be able to do

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

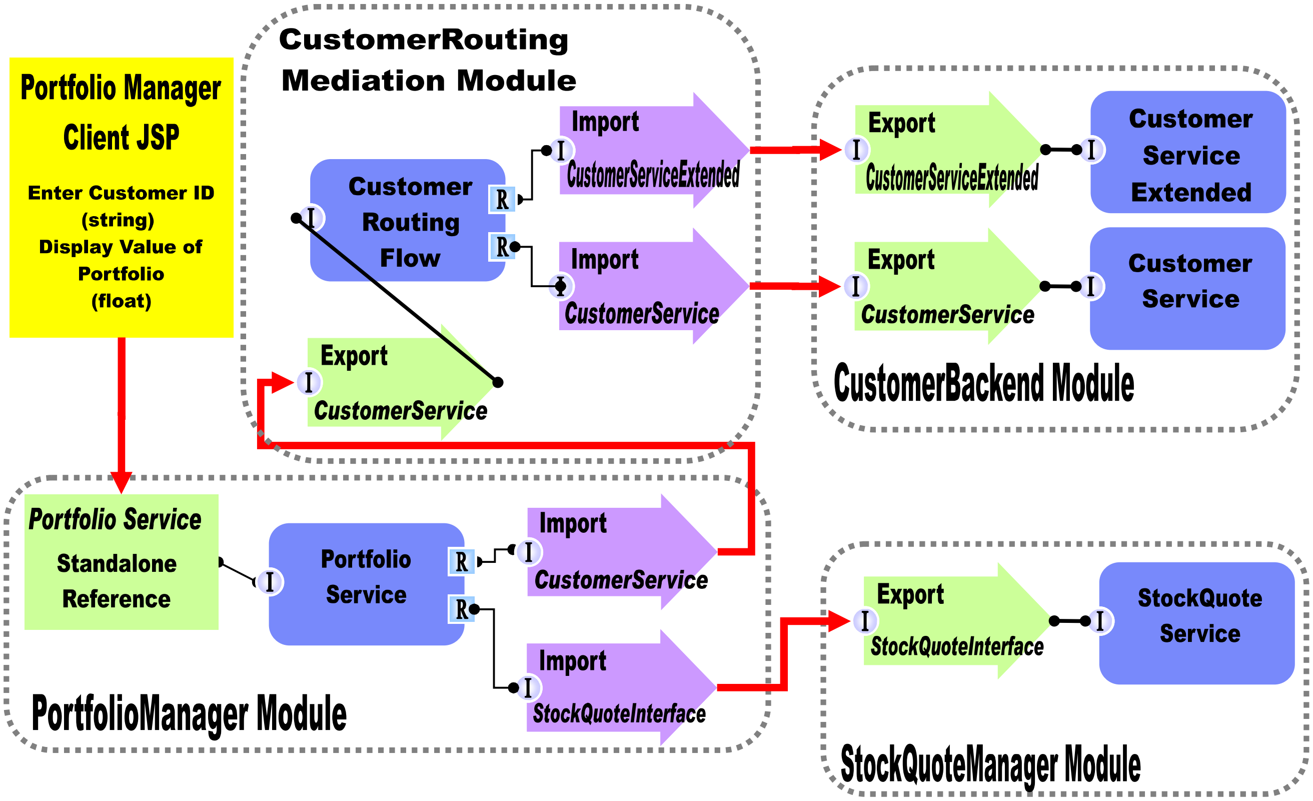
- Import project interchange files into the WebSphere Integration Developer V6.0.1 development environment.
- Know how to create Custom Mediation primitives and understand what they do.
- Know how to create Database Lookup primitives and understand what they do.
- Know how to create Message Filter primitives and understand what they do.
- Know how to create XSL Transformation primitives and understand what they do.
- Navigate the Properties View for mediation information.

Introduction

In Lab 1 of this lab series, you changed an SCA application to one that uses a mediation module as described in the diagram below. You added the CustomerRoutingMediationModule between the PortfolioManager service and the CustomerBackend service to allow you access to message information being passed between them. You then created a mediation flow called CustomerRoutingFlow that uses a Message Logger mediation to store message information in a cloudscape database.



In Lab2, you will add to the Message Logger mediation primitive with a couple others available in WebSphere Integration Developer V6.0.1. The main goal for Lab2 will be 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 a 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, 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 an 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 would end in 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.

In Lab 4, you are going to learn how to create EAR files for mediation modules, install EAR files to the WebSphere ESB Server profile, and edit/connect SCA Binding information without using WebSphere Integration Developer. You will be using the Administration Console for the WebSphere ESB Server.

Throughout the lab you will learn what WebSphere Integration Developer does behind the scenes by manually exporting EAR files, installing EAR files, starting/stopping the ESB server, and using the Administrative Console to edit SCA Binding information. This is helpful to know how to take these actions outside of the development environment and to understand what WebSphere Integration Developer is doing for you when inside the development environment.

Exercise Instructions

Some instructions in this lab may be Windows® operating-system specific. If you plan on running the lab on an operating-system other than Windows, you will need to run the appropriate commands, and use appropriate files (.sh vs. .bat) for your operating system. 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\ID	
<LAB_FILES>	C:\Labfiles601	/tmp/Labfiles601
<TEMP>	C:\temp	/tmp

Windows users 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:\LabFiles601\ would be replaced by C:/LabFiles601/

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	

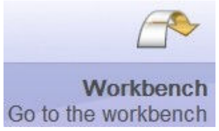
Instructions for using a remote testing environment, such as z/OS, AIX or Solaris, can be found at the end of this document, in the section "**Task: Adding Remote Server to WebSphere Integration Developer Test Environment**".

Part 1: Prepare Environment for Lab 2

In this section of the lab, you will be importing all projects inside the WESBlab_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 V6.0.1 with a workspace location of **C:\LabFiles601\WESB\Lab2\workspace**.

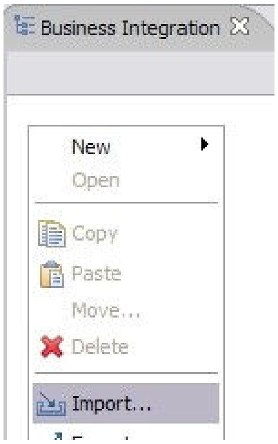
____ 2. On the Welcome window, click the curved arrow at top right to **go to workbench**.



Import Project Interchange file, **WPIv601_ESB_StartLab2_PI.zip**, into the development environment if starting fresh, or **WPIv601_ESB_FinishedLab1_PI.zip** if continuing from Lab1.

__ a. Right-click inside **Business Integration View** (top left view in the Business Integration Perspective)

__ b. Select **Import** from list.



__ c. Select **Project Interchange** from list. Click **Next**.

__ d. Select the top **Browse** button for **From zip file**:

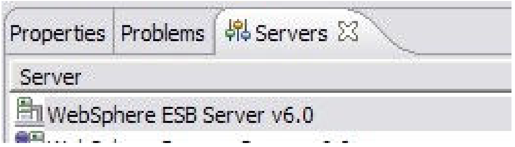
__ e. Navigate to **C:/LabFiles601/WESB/Lab2/ WPIv601_ESB__StartLab2_PI.zip**.

__ f. Click the **Select All** button to check all checkboxes for projects listed.

__ g. Verify you have **CustomerBackend**, **CustomerRoutingMediationModule**, **PortfolioLibrary**, **Portfolio Manager**, and **StockQuoteManager** modules listed in the Business Integration view.

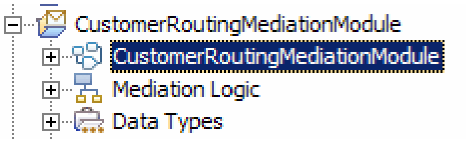
__ h. Click **Finish** button (projects will be imported and auto-build will run).

__ i. Verify you have WebSphere ESB Server v6.0 listed in your Servers view.

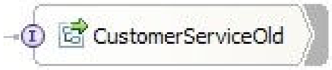


Prepare Assembly Diagram for Lab 2

__ j. Open Assembly Diagram for **CustomerRoutingMediationModule** by double-clicking the entry in Business Integration View.


CustomerRoutingMediationModule
CustomerRoutingMediationModule
Mediation Logic
Data Types

__ k. Change the name of the import **CustomerServiceOut** to **CustomerServiceOld**.



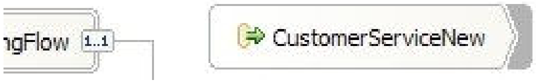
Add Import **CustomerServiceNew** in Assembly Diagram of CustomerRoutingMediationModule.

Drag and drop an import on the right-hand side of CustomerRoutingFlow.

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

Click or drag import to the right side of CustomerRoutingFlow.

Change default import name from Import1 to **CustomerServiceNew**.

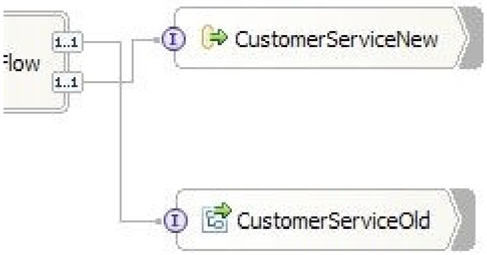


Hover over import or click on import to make the add interface icon appear.

Click “I” icon ( ) to add an interface.

1) Select **CustomerServiceExtended** from the list of matching interfaces. Click **OK**.

2) Wire CustomerRoutingFlow and CustomerServiceNew together by clicking on **CustomerRoutingFlow** and drag a wire to **CustomerServiceNew**.
Click **OK** for any pop-up windows.



3) Update SCA binding for **CustomerServiceNew**.

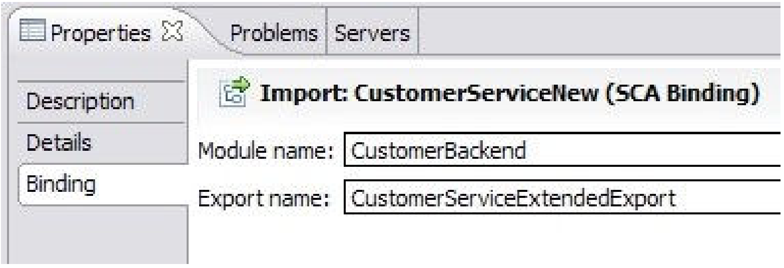
a) Right-click on CustomerServiceNew and select **Generate Binding > SCA Binding**.

b) Select the Import **CustomerServiceNew** on the Assembly Diagram.

c) Open Properties view to the **Binding tab**.

d) Click **Browse…** button and select **CustomerServiceExtendedExport**. Click **OK**.

e) Here is what your bindings tab should look like now.



Import: CustomerServiceNew (SCA Binding)

Module name: CustomerBackend

Export name: CustomerServiceExtendedExport

f) Save all work by **File > Save All** or **Crtl + Shift + S**.

g) After saving the Assembly Diagram, there should be a **red X** on the CustomerRoutingFlow. You need to merge the implementation between the Assembly Diagram and the Mediation Flow to resolve this conflict.

4) Merge Implementation to synchronize Assembly Diagram and Mediation Flow from adding the new input.

a) Right click on the **CustomerRoutingFlow** in the Assembly Diagram.

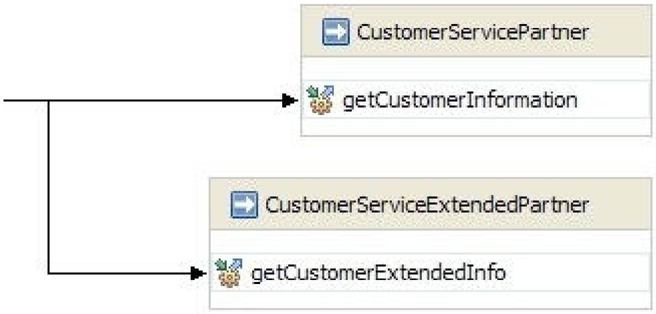
b) Select **Merge Implementation**.

c) Click **OK** on warning message that appears.

d) Click **OK** to add the **CustomerServiceExtendedPartner** to the Mediation Flow.

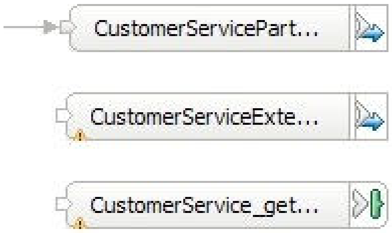
e) The Mediation Flow Editor will appear and **CustomerServiceExtendedPartner** will be added to the Operation connections view (top-most view of Mediation Flow Editor).

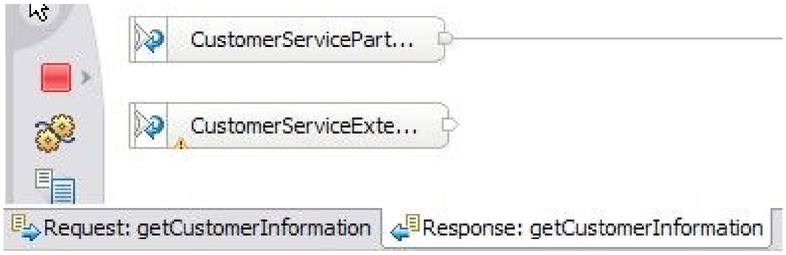
f) Drag a wire from the CustomerService operation on left-hand side of the Operation connections view to the CustomerServiceExtendedPartner.



g) Click on the wire in the Operation connections view in the Mediation Flow Editor to see the Mediation Flow view (middle view).

h) Notice also in the Mediation Flow view that a **callout** (for Request tab - 1) and an **input** (for Response tab - 2) has been added for CustomerServiceExtendedPartner.



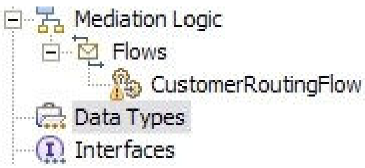

CustomerServicePart...
CustomerServiceExte...
Request: getCustomerInformation
Response: getCustomerInformation

i) Save editor by selecting **File > Save** or **Ctrl + S**.

5) Create a transient context business object.

In this lab, the custom mediation primitive you will be creating in the next section uses java code that will extract a two digit prefix from the customer ID and places those 2 digits in the transient context. Think of the transient context as a scratchpad, somewhere to store information in a message. It is defined as a business object (or DataType in the Business Integration view) and part of the SMO (Service Message Object). These following steps show you how to create a transient context business object.

a) Go to the **Business Integration view** on your workspace. Directly under the CustomerRoutingFlow mediation flow is an entry called **Data Types**.



- Mediation Logic
  - Flows
    - CustomerRoutingFlow
  - Data Types
  - Interfaces

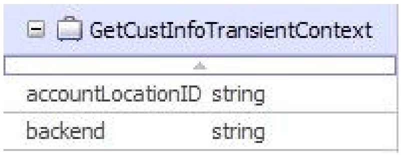
b) Right-click on **Data Types** and select **New > Business Object**.

c) Type in **GetCustInfoTransientContext** in the Name field. Click **Finish**.

d) The Business object editor opens. Click the add attribute icon () twice to add two attributes to the business object.

e) Change the text attribute1 to **accountLocationID**.

f) Change the text attribute2 to **backend**. The result should look like below picture.



GetCustInfoTransientContext	
accountLocationID	string
backend	string

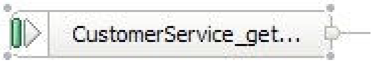
g) **Save GetCustInfoTransientContext** and close the business object editor.

6) Add **GetCustInfoTransientContext** to mediation flow input.

a) Open the **CustomerRoutingFlow** Mediation from the Business Integration view by double-clicking the entry **CustomerRoutingMediationModule > Mediation Logic > Flows > CustomerRoutingFlow**.

b) Click on wire connecting CustomerService to CustomerServicePartner and CustomerServiceExtendedPartner.

c) Select/Highlight the only input for request tab of the Mediation Flow view, **CustomerService_getCustomerInformation_Input**.



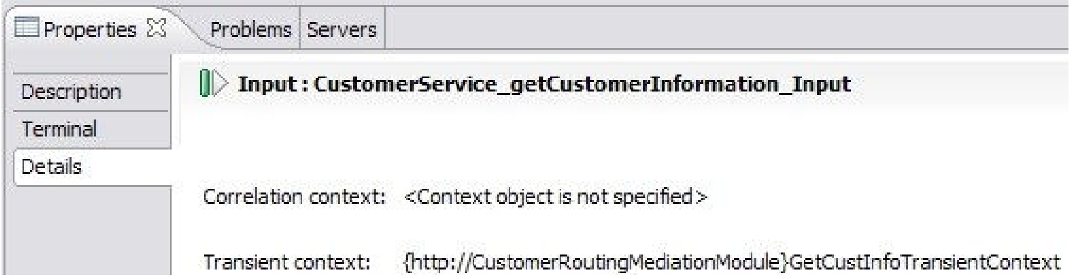
CustomerService_get...

d) Go to the **Details tab** of the **Properties view** once the input is selected.

e) Click the **Browse…** button in the same row as **Transient Context**. You will see that Transient Context, nor the Correlation context, is specified

f) Select **GetCustInfoTransientContext**. Click **OK**.

g) Your **Details tab** will now look like the following:



Input : CustomerService_getCustomerInformation_Input

Correlation context: <Context object is not specified>

Transient context: {http://CustomerRoutingMediationModule}GetCustInfoTransientContext

h) Save changes by selecting **File > Save** or **Ctrl+S**.

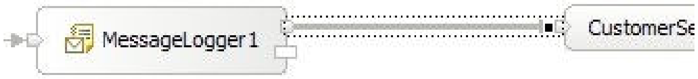
i) You are now ready to start creating mediation primitives.

Part 2: Add a Custom Mediation primitive to Mediation Flow

In this section of the lab, you will be adding a custom mediation primitive to the mediation flow. The custom mediation uses java code which is going to be provided for you in a snippet that will extract a two digit prefix from the customer ID and place it in the transient context.

____ 1. Delete wire from MessageLogger1 to CustomerServicePartner callout in order to connect message logger primitive to a new custom mediation primitive.

__ a. In the Mediation Flow view of the Mediation Flow Editor, click on the **wire** connecting **MessageLogger1** to **CustomerServicePartner** callout.



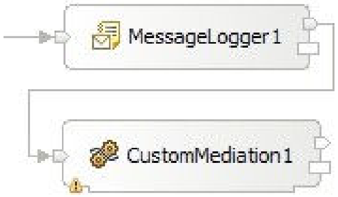
__ b. Press the Delete button on the keyboard or right click on wire and select **Delete**.

Drag and drop a custom mediation primitive on the Mediation Flow to create a new custom mediation primitive.

__ c. Click the custom mediation icon ( ) in the mediation flow palette (left-hand side) and then click under the MessageLogger1. The CustomMediation1 will appear.

Wire the output terminal of MessageLogger1 to the input terminal of CustomMediation1.

__ d. Click on output terminal of MessageLogger1 and drag a wire to the input terminal of CustomMediation1.



Define a Service Operation for CustomMediation1.

__ e. Click on CustomMediation1 and navigate to the **Details tab** of **Properties View**.

__ f. Click on the **Define…** button.

__ g. The Define Custom Mediation wizard will appear. Take the default radio button "**Create a new interface with implementation**". Click **Next**.

__ h. For Message Root drop down box, change the value from "/body" to "/".

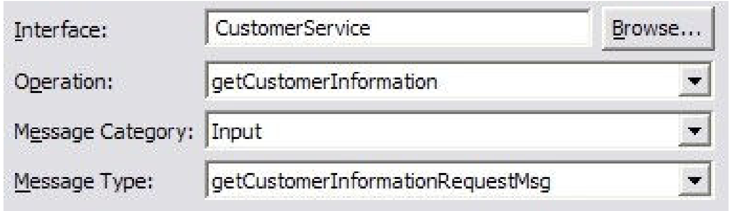
__ i. Since you wired the MessageLogger1 output to the input of CustomMediation1, the input message body has been automatically defined. However, since you have not wired the output to any other object, the output message has not been defined. Click on **Browse…** next to Output Message Body to run the Change Message Type wizard.

__ j. Click the **Browse…** to open a list of interfaces. Select **CustomerService** and click the **OK** button.

__ k. For operation, click the drop-down button to populate the box with the **getCustomerInformation** operation.

__ l. For Message Category, click the box and select **Input**. You will notice later input is for the request side of the flow and output is for the response side of the flow.

__ m. The Message Type then gets populated when choosing input or output (request or response). The wizard should look like this:



Interface: CustomerService

Operation: getCustomerInformation

Message Category: Input

Message Type: getCustomerInformationRequestMsg

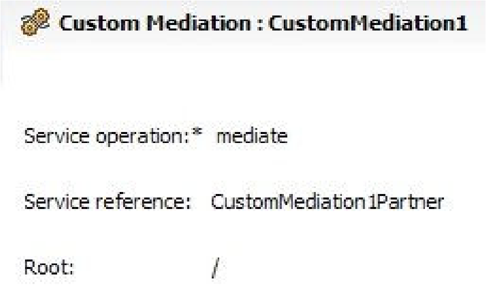
__ n. Click **OK**.

__ o. Notice that the Transient Context is defined here when defining a new service operation. Remember this is what you are going to use to store the 2 digits you will retrieve from the CustomerID. Click **Next**.

__ p. On the "create a new interface" section of the Define Custom Mediation wizard, notice the operation will be named "mediate". Click **Next**.

__ q. Select the second radio button. **Specify the implementation manually as Java Component or Import in the Assembly Editor, default Java implementation will not be generated**. After you save the mediation flow, you will have the Assembly Diagram merge implementation to generate the java component. Click **Finish**.

__ r. Here is what you should have for the service operation definition:



Custom Mediation : CustomMediation1

Service operation:* mediate

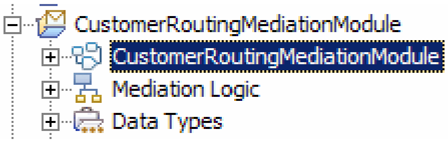
Service reference: CustomMediation1Partner

Root: /

__ s. Save all work by **File > Save All** or **Crtl + Shift + S**.

Merge Implementation in Assembly Diagram to synchronize the Mediation Flow with Assembly Diagram.

__ t. Open the **Assembly Diagram** for CustomerRoutingMediationModule by double-clicking the entry in Business Integration View.



CustomerRoutingMediationModule
CustomerRoutingMediationModule
Mediation Logic
Data Types

__ u. Right click on the **CustomerRoutingFlow** in the Assembly Diagram.

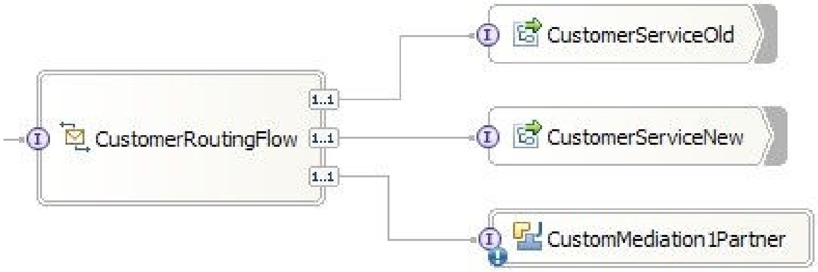
__ v. Select **Merge Implementation**.

__ w. Click **OK** on warning message that appears.

__ x. Click **OK** to add the CustomMediation1Partner to the Mediation Flow.

__ y. The Mediation Flow Editor will appear. Go back to Assembly Diagram.

__ z. Notice that the CustomMediation1Partner java component has been created on the Assembly Diagram. Save the update (File > Save or key in Ctrl + S) and the red x should disappear after project has built.



Generate implementation code for the java component and copy in snippet code.

__ aa. The blue exclamation icon on CustomMediation1Partner states it does not have any implementation code. Therefore, right-click on CustomMediation1Partner and select **Generate Implementation**.

__ bb. Click on **New Package…** button. Type in **sca.java.impl** for the package, and select it from the generated list. Click **OK**.

__ cc. CustomMediationPartner1_1Impl.java will be created and displayed in Java editor. Scroll down the page till you see the line… **public DataObject mediate(DataObject input1) {**

__ dd. Add snippet to mediate method and organize imports for code.

1) Outside of WebSphere Integration Developer, navigate to **C:/Labfiles601/WESB/Lab2/snippet1.txt** and open in a text editor.

2) Copy contents and paste inside mediate method between the { } symbols where currently there is…

```
//TODO Needs to be implemented.
return null;
```

The code to replace this with is…

```
ServiceMessageObject smo = (ServiceMessageObject)input1;
ContextType context = smo.getContext();
DataObject transientContext = (DataObject)context.getTransient();
DataObject body = (DataObject)smo.getBody();
DataObject customerInfo = (DataObject)body.get("getCustomerInformation");
String customerID = (String)customerInfo.get("customerID");
if (customerID.length() > 1)
```

```
        transientContext.setString("accountLocationID", customerID.substring(0,2));
else
        transientContext.setString("accountLocationID", "00");
return (DataObject)smo;
```

3) Notice there will be errors. You will need to organize imports in order for this code to work. Right-click in the java editor and select **Source -> Organize Imports** or **Ctrl + Shift + O**.

4) Select the first interface that displays, **com.ibm.websphere.sibx.smobo.ContextType**. Click **Finish**.

5) Errors should go away. Save work by navigating to **File** > **Save** or **Ctrl + S**.

6) Close Java Editor.

__ ee. Save Assembly Editor navigating to **File** > **Save** or **Ctrl + S.**

__ ff. You have now fully created a custom mediation primitive with working java code to support it.

Part 3: Add a Database Lookup mediation primitive to Mediation Flow

This section is two fold. In the first part of this section, you will learn how to set up a datasource that the database lookup mediation primitive will use. A simple database is supplied for this lab, however walking through the steps to set up the datasource allows you to view the Administration Console and what it takes to use a database lookup mediation primitive. In the second part of this section, you will add a Database Lookup primitive that uses that two digit prefix from transient context as key to lookup a backend identifier to also 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.

____ 1. Add a datasource in the administration console for the database lookup primitive to use.

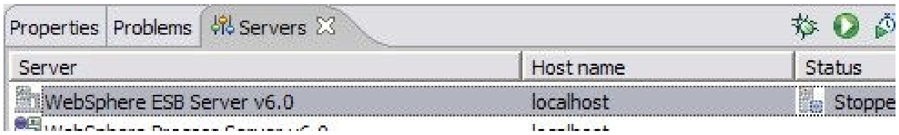
Start **WebSphere ESB Server v6.0**.

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, to start the remote server.

If using a local testing environment:

1) Open **Servers View**.

2) Highlight WebSphere ESB Server v6.0 and click **Start button** (▶).



Server	Host name	Status
WebSphere ESB Server v6.0	localhost	Stoppe

3) This will take some time. Wait for the server status to say **Started**.

NOTE: You may notice a relationship error in the console view after the server has started. Please disregard and continue with lab. It is a problem that is currently being fixed.

__ b. Right-click on the WebSphere ESB Server v6.0 in the Servers view and select **Run administration console**.

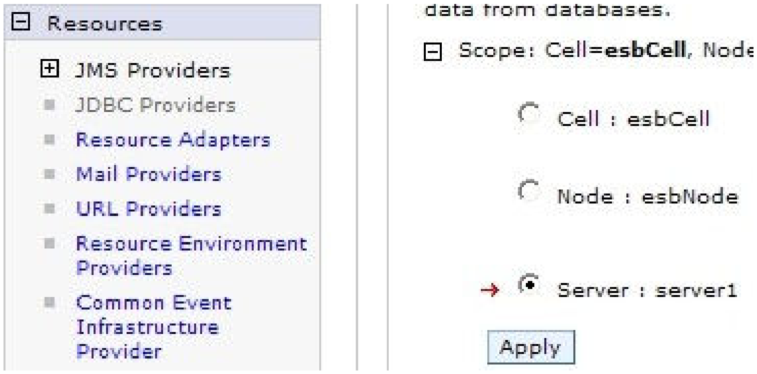
__ c. Log-in to the administration console with any user ID you like.

__ d. Under the Task Filtering Selector section on the first page of the administration console, click on **Server and Bus**, then the **Apply** Button.

__ e. In the left hand menu options, find **Resources** and click to open.

__ f. Click on **JDBC Providers**.

__ g. There are 3 radio button choices when you first come to the JDBC Providers page. Click the radio button next to **Server: <SERVER_NAME>** and click the **Apply button**.



data from databases.

⊟ Scope: Cell=**esbCell**, Node

○ Cell : esbCell

○ Node : esbNode

● Server : server1

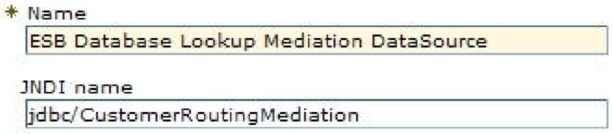
__ h. Below the Apply button is the list of JDBC providers that the server knows of. Click on **Cloudscape JDBC Provider (XA)**.

__ i. Under **Additional Properties** on the right hand side of the page, click on **Data sources** link.

__ j. At the datasources table, click on the **New** button at top-right of the data sources table.

__ k. For name, delete **Cloudscape JDBC Driver XA Data Source** and replace with **ESB Database Lookup Mediation Data Source**.

__ l. For JNDI name, replace **jdbc/Cloudscape JDBC Driver XA Data Source** with **jdbc/CustomerRoutingMediation**.

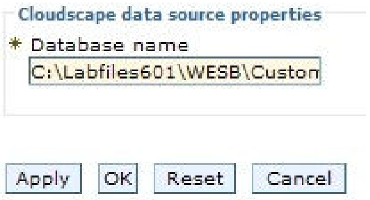


* Name
ESB Database Lookup Mediation DataSource

JNDI name
jdbc/CustomerRoutingMediation

__ m. Scroll down to the bottom of the page and enter **C:\Labfiles601\WESB\CustomerRoutingMediation** for **Database Name**.

If using a remote system for testing, you will need to put the database on your remote system by uploading it in binary. This entails uploading the entire database folder. Once on your remote system, upload just the service.properties file in ASCII. You will then need to substitute the directory where you placed the database, such as /**u**/**wsuser**/**CustomerRoutingMediation**. Make sure the permissions are set up to allow the server to access the database.

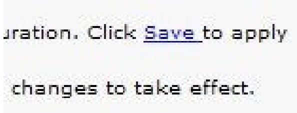


Cloudscape data source properties

* Database name
C:\Labfiles601\WESB\Custon

__ n. Click the **Apply Button**.

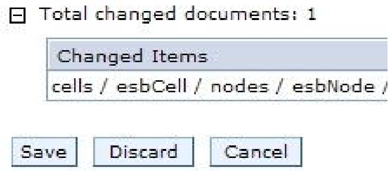
__ o. After the page refreshes, click on the **save** link at the top of the page.



ıration. Click Save to apply

changes to take effect.

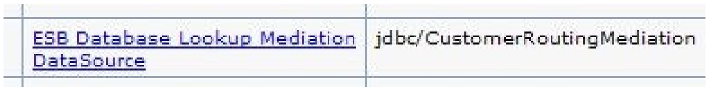
__ p. You will be taken to the Save page. Click the **Save button**. Only one file, resources.xml, was updated.



⊟ Total changed documents: 1

Changed Items
cells / esbCell / nodes / esbNode /

__ q. After the file has saved on the running application, you will see the JDBC Providers table is updated with your new datasource.



ESB Database Lookup Mediation DataSource	jdbc/CustomerRoutingMediation

__ r. At the top-left of the Administration Console page, click on **Logout**. Close **Administrative Console**.

You have now built the datasource and are ready to build the Database Lookup mediation primitive in your workspace.

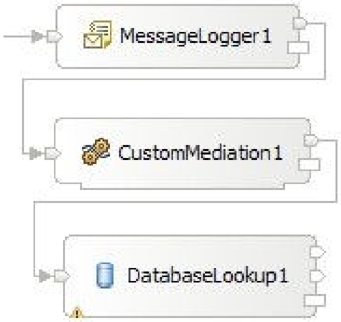
____ 3. Open the Mediation Flow diagram if not already up.

____ 4. Drag and drop a database lookup primitive on the Mediation Flow to create a new database lookup primitive.

__ a. Click the database lookup icon ( ) in the mediation flow palette (left-hand side) and then click under CustomMediation1. The DatabaseLookup1 primitive will appear.

Wire the output terminal of CustomMediation1 to the input terminal of DatabaseLookup1.

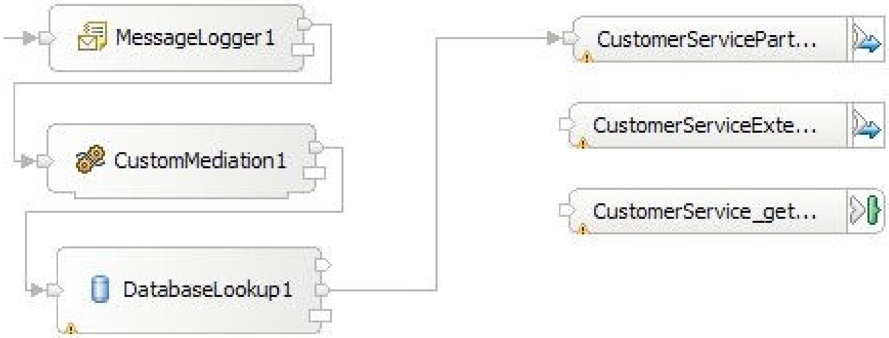
__ b. Click on output terminal of CustomMediation1 and drag a wire to the input terminal of DatabaseLookup1.



Connect the keyNotFound output terminal with the CustomerServicePartner_getCustomerInformation_Callout.

__ c. The middle output on DatabaseLookup1 is called the **keyNotFound** output. As you would think, if a key is not found when a lookup is performed on the database, the message will be sent to this output terminal. If a key is not found, you would like to send that message to the **CustomerServiceOld** backend.

Click the keyNotFound output terminal of DatabaseLookup1 and drag to CustomerServicePartner callout.



__ **d.** In the above image, notice that the bottom node on the right-hand side, **CustomerService_getCustomerInformation_InputResponse**, will not be wired in this example. InputResponse stops the message and sends it as-is back as the response (in this case not going to the backend module and back through the response side of the mediation flow). Since you do want your message to go to the backend module, you will send the message through the two callout nodes.

Define the datasource and database lookup information in the details tab of Properties view.

__ e. Click on DatabaseLookup1 and open **Properties View**.

__ f. Click on **Details tab**.

__ g. Enter the **datasource** information.
We have setup a database for you already and have defined this database with a datasource in

the Administrative Console of the WebSphere Enterprise Service Bus server. In this sub-step, you are filling out the form to allow this application to use this datasource and database.

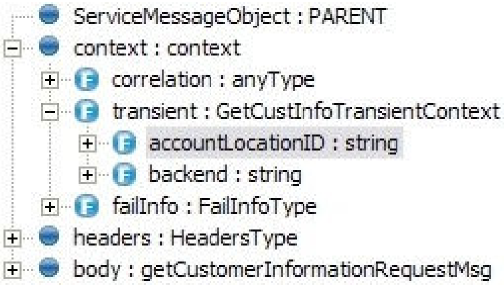
1) Data source name: **jdbc/CustomerRoutingMediation**

2) Table Name: **BACKEND_LOCATIONS**

3) Key column name: **ACCT_NO_PREFIX**

4) Key path: Click on **Custom Xpath…** button to define which element to use.

a) Navigate the menu and click on **context > transient > accountLocationID**.


ServiceMessageObject : PARENT
context : context
correlation : anyType
transient : GetCustInfoTransientContext
accountLocationID : string
backend : string
failInfo : FailInfoType
headers : HeadersType
body : getCustomerInformationRequestMsg

b) Click **OK**.

5) In the Data Elements table, enter the backend element from the transient context as your message element you want to lookup.

a) Click on the **Add** button at the bottom of the Details tab page.

b) In the Add/Edit properties wizard, enter the following values

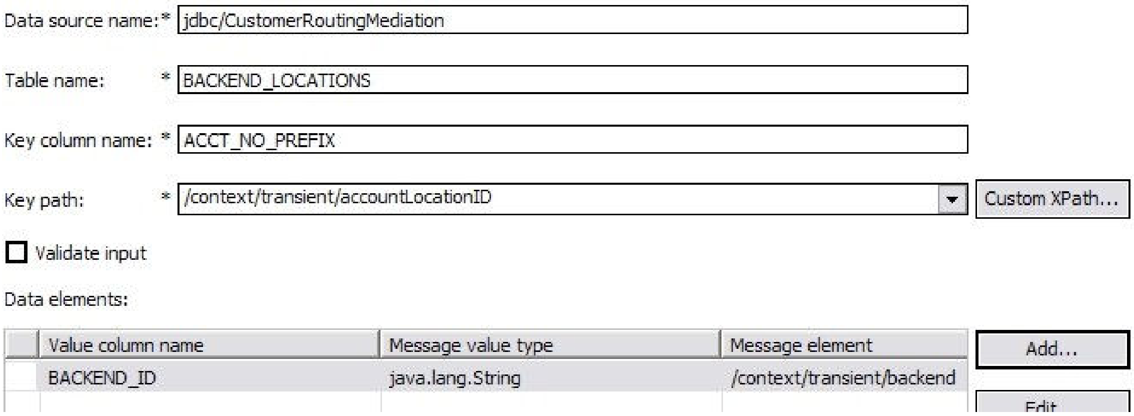
(1) Value column name: **BACKEND_ID**

(2) Message value type: **java.lang.String** (leave as is)

(3) Message element: Click on **Custom Xpath…** and navigate to **context > transient > backend**.

(4) Click **OK**. Click **Finish**.

__ h. Your final detail tab page should now look like this…



Data source name:* jdbc/CustomerRoutingMediation

Table name: * BACKEND_LOCATIONS

Key column name: * ACCT_NO_PREFIX

Key path: * /context/transient/accountLocationID

□ Validate input

Data elements:

Value column name	Message value type	Message element
BACKEND_ID	java.lang.String	/context/transient/backend

Save your work by navigating to **File > Save** or hitting **Ctrl + S** on your keyboard. You are finished defining a Database lookup mediation primitive that looks up the backend identifier.

Part 4: Add a Message Filter mediation primitive to Mediation Flow

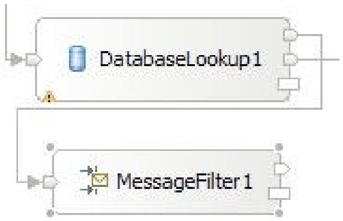
In this section you will determine the message routing based on backend identifier by adding a Message Filter primitive to the Mediation Flow. The old backend will go directly to the callout for CustomerService and the new backend will go to an XSL Transformation mediation primitive (XSL Transformation will be explained in next section).

____ 1. Drag and drop a Message Filter mediation primitive on the Mediation Flow to create a new message filter.

__ a. Click the message filter icon ( ) in the mediation flow palette (left-hand side) and then click under DatabaseLookup1. The MessageFilter1 primitive will appear.

Wire the output terminal of DatabaseLookup1 to the input terminal of MessageFilter1.

__ b. Click on **output terminal** of DatabaseLookup1 and drag a wire to the **input terminal** of MessageFilter1.



Add OldBackend1 and NewBackend1 output terminals to MessageFilter1

__ c. Click on MessageFilter1 and open the **Terminal tab** of the **Properties View**.

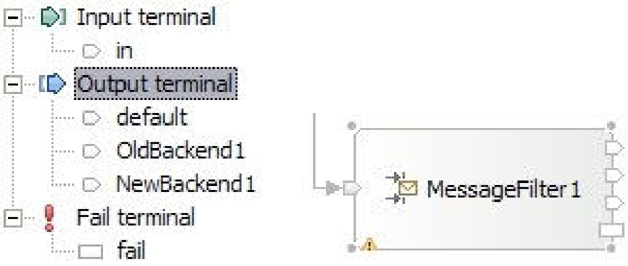
__ d. Right-click on **Output terminal** and select **Add Output Terminal**.

__ e. Leave terminal type as “match”, but change the **Terminal Name** to **OldBackend1**. Click **OK**.

__ f. Right-click on **Output terminal** and select **Add Output Terminal** again.

__ g. Leave terminal type as “match”, but change the **Terminal Name** to **NewBackend1**. Click **OK**.

__ h. You should now have 3 output terminals on MessageFilter1.



Add two filter options on Details tab to determine the message routing based on the backend identifier/pattern.

__ i. Click on the **Details tab** of the **Properties View**. Here is where you will provide the filter options for the message filter to make decisions on which output to send the message to, NewBackend1 or OldBackend1.

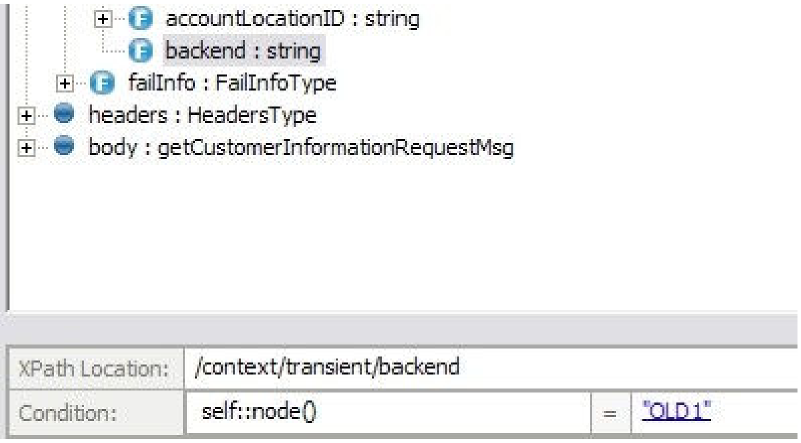
__ j. Click on the **Add button** on right-hand side of the Details tab page.

__ k. In the Add/Edit wizard, click on the **Custom Xpath…** button.

__ l. Navigate to **context > transient > backend** and click on **backend**.

__ m. Below, next to **Condition:** click on the cell that currently has "Location" in it. Click the **self:: () node : string** option.

__ n. For the value, click on "Value" in blue text, and then click on String. Enter the text **OLD1** and hit enter button on keyboard.


accountLocationID : string
backend : string
failInfo : FailInfoType
headers : HeadersType
body : getCustomerInformationRequestMsg

XPath Location: /context/transient/backend

Condition: self::node() = "OLD1"

__ o. Click **OK**. Keep the Terminal Name of **OldBackend1**.

__ p. Click **Finish**.

__ q. Now do this again to add a filter option for NEW1. Click on the **Add button** on right-hand side of the Details tab page.

__ r. In the Add/Edit wizard, click on the **Custom Xpath button**.

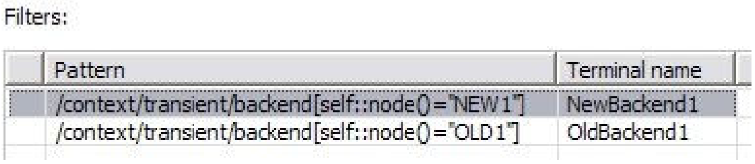
__ s. Navigate to **context -> transient -> backend** and click on **backend**.

__ t. Below, next to **Condition:** click on the cell that currently has "Location" in it. Click the **self:: () node : string** option.

__ u. For the value, click on "Value" in blue text, and then click on String. Now this time, enter the text **NEW1** and hit enter button on keyboard. Click **OK**.

__ v. Click the drop-down box for Terminal Name and change it to **NewBackend1**.

__ w. Click **Finish**. Here is what you should have in your filters table.



Filters:

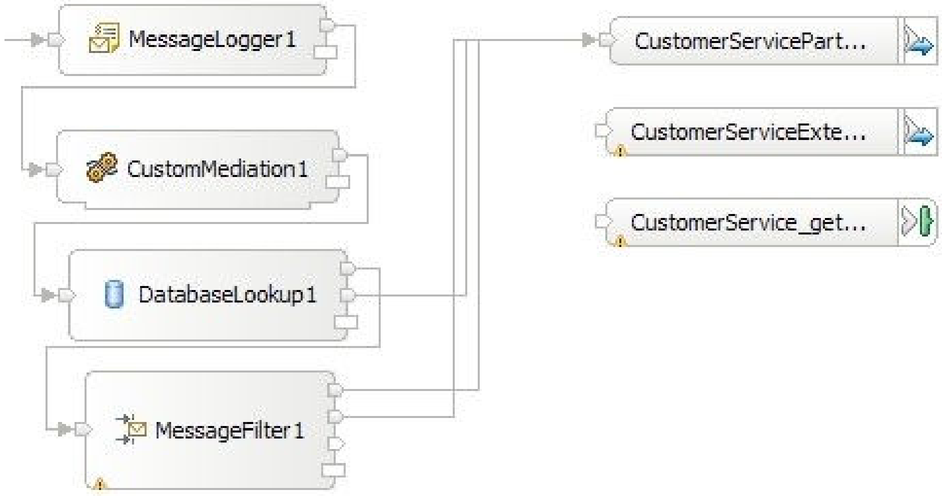
Pattern	Terminal name
/context/transient/backend[self::node()="NEW1"]	NewBackend1
/context/transient/backend[self::node()="OLD1"]	OldBackend1

__ x. Save your work by navigating to **File > Save** or hitting **Ctrl + S** on your keyboard.

Wire default and OldBackend1 output terminals to CustomerServicePartner.

__ y. In the mediation flow diagram, click on the **default output terminal** of MessageFilter1 and drag it to the **CustomerServicePartner callout**.

__ z. Click on the **OldBackend1 output terminal** of MessageFilter1 and drag it to the **CustomerServicePartner callout**.



__ aa. The NewBackend1 is now reserved to go to an XSL Transformation since you need to change the CustomerService business object to a CustomerServiceExtended business object.

__ bb. Save all changes.

Part 5: Add XSL Transformation mediation primitives to Mediation Flow

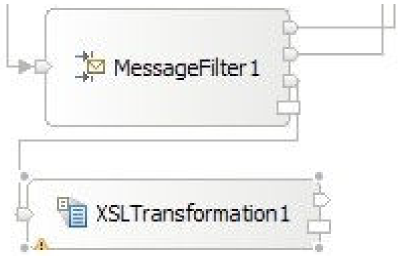
In this section you will create two XSL Transformations that will transform the CustomerService business object (BO) into the CustomerServiceExtended business object, then pass the newly formed message to the callout for CustomerServiceExtendedPartner instead of the CustomerServicePartner. Notice that 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 an 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 would end in 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.

____ 1. Drag and drop an XSL Transformation mediation primitive on the Mediation Flow to create a new XSL Transformation.

__ a. Click the XSL Transformation icon ( ) in the mediation flow palette (left-hand side) and then click under MessageFilter1. The XSLTransformation1 primitive will appear.

Wire the output terminal of MessageFilter1 to the input terminal of XSLTransformation1.

__ b. Click on **output terminal (NewBackend1)** of MessageFilter1 and drag a wire to the **input terminal** of XSLTransformation1.



Create a new mapping for this XSL Transformation.

__ c. Select XSLTransformation1 and open the **Details tab** of the **Properties View**.

__ d. Leave the Root: as /**body**.

__ e. Click on the **New…** button on the Mapping File row.

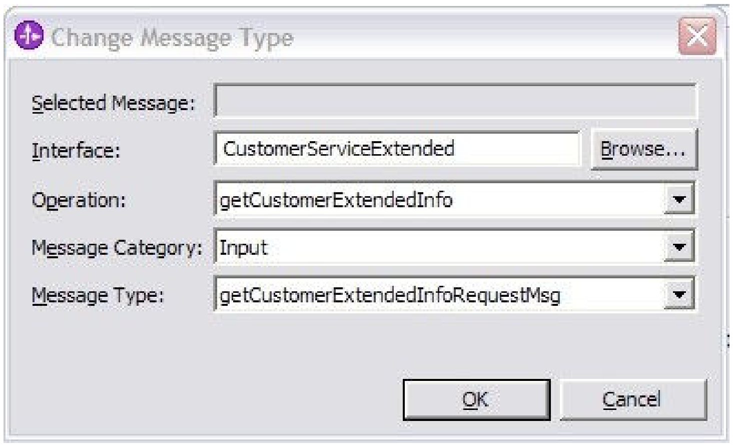
__ f. Since you wired the MessageFilter1 to XSLTransformation1, the input message body is already assigned. Click the **Browse…** button next to Output Message Body.

__ g. Click the **Browse…** button to open a list of interfaces. Select **CustomerServiceExtended** and click the **OK** button.

__ h. For operation, click the drop-down button to populate the box with the **getCustomerExtendedInfo** operation.

__ i. For Message Category, click the box and select **Input** since you are still working on the request side. You will notice input is for the request side of the flow and output is for the response side of the flow.

__ j. The Message Type then gets populated when choosing input or output (request or response). The wizard should look like this:



Change Message Type

Selected Message:

Interface: CustomerServiceExtended

Operation: getCustomerExtendedInfo

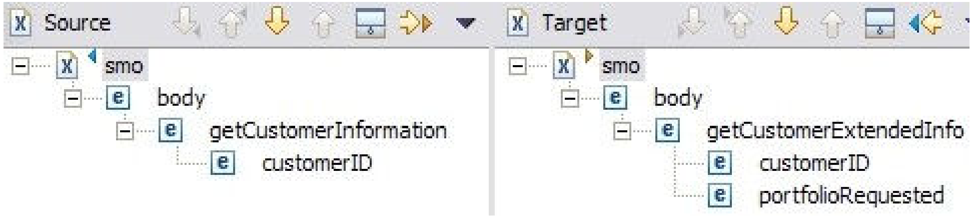
Message Category: Input

Message Type: getCustomerExtendedInfoRequestMsg

__ k. Click **OK**.

__ l. Now you should have **getCustomerInformationRequestMsg** for the Input Message Body and **getCustomerExtendedInfoRequestMsg** for the Output Message Body. Click **Finish**.

__ m. The mapping file XSLTransformation1_req_1.xmx will be created and opened. Fully expand both the **Source and Target objects**.



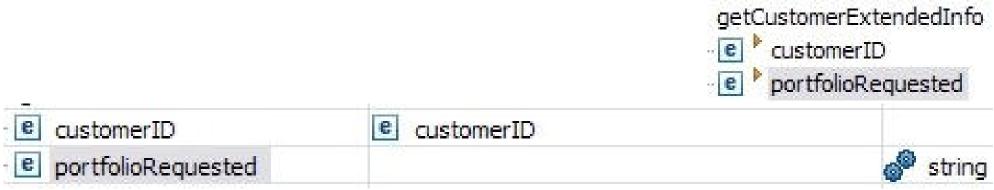
__ n. Click on **customerID** from the **Source** side and drag to the **customerID** on the **Target** side. Notice triangle icons appear next to the mapped elements and that the Overview view shows the mapping.

__ o. Since **portfolioRequested** is a new element for the target, you are going to assign a default value to fill **portfolioRequested** with something. You are going to put the string "default" in for this value. Right-click on **portfolioRequested** and select **Define XSLT Function**.

__ p. Leave **string** as default function. Click **Next**.

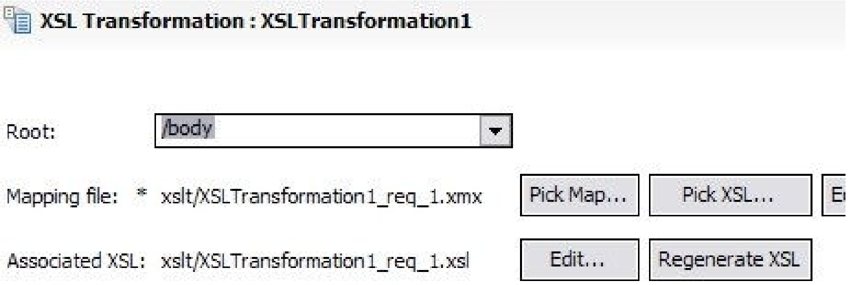
__ q. Change **Function Name** to **String** in the drop-down list.

__ r. Click the **Add button** for input parameters and type **'default'**. Click **OK**. Click **Finish**.



__ s. Save the mapping by navigating to **File -> Save** or hit **Ctrl + S** on your keyboard. **Close** mapping file.

__ t. Back in the Details tab of Properties View for the XSL Transformation, click on the **Regenerate XSL** button. Click **OK** on the information message. You will see that XSLTransformation1_req_1.xsl has been generated.



XSL Transformation : XSLTransformation1

Root: /body

Mapping file: * xslt/XSLTransformation1_req_1.xmx

Associated XSL: xslt/XSLTransformation1_req_1.xsl

__ u. Click on Mediation Flow and navigate to **File -> Save** or hit **Ctrl + S** on your keyboard to save your **Mediation Flow**.

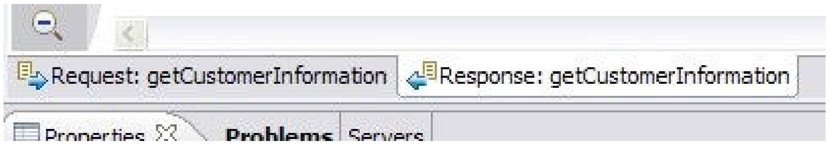
Wire the output terminal of XSLTransformation1 to the CustomerServiceExtendedPartner_getcustomerExtendedInfo_Callout.

__ v. Click on output terminal of XSLTransformation1 and drag to the CustomerServiceExtendedPartner_getcustomerExtendedInfo_Callout.

__ w. Click on Mediation Flow and navigate to **File -> Save** or hit **Ctrl + S** on your keyboard to save your **Mediation Flow**.

Add an XSL Transformation to the response side of Mediation flow.
This action turns the CustomerServiceExtended business object back into a CustomerService business object that the application knows how to work with.

__ x. Click on the **Response tab** of mediation flow (found at bottom of mediation flow, two tabs are available. Request and Response.



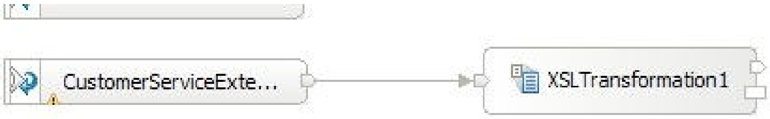
Request: getCustomerInformation
Response: getCustomerInformation
Properties
Problems
Servers

__ y. Drag and drop an XSL Transformation mediation primitive on the Mediation Flow to create a new XSL Transformation.

1) Click the XSL Transformation icon ( ) in the mediation flow palette (left-hand side) and then click on the flow editor canvas. The XSLTransformation1 primitive will appear.

__ z. Wire the callout response for CustomerServiceExtendedPartner to the input terminal of XSLTransformation1.

1) Click on **output terminal** of CustomerServiceExtendedPartner and drag a wire to the **input terminal** of XSLTransformation1.



__ aa. Create a new mapping for this XSL Transformation.

1) Select XSLTransformation1 and open the **Details tab** of the **Properties View**.

2) Leave the Root: as /**body**.

3) Click on the **New…** button on the Mapping File row.

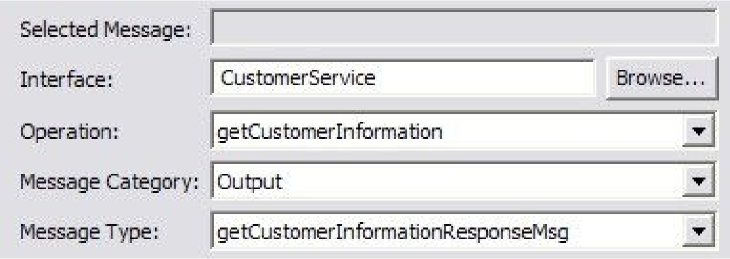
4) Since you wired the MessageFilter1 to XSLTransformation1, the input message body is already assigned. Click the **Browse…** button next to Output Message Body.

5) Click the **Browse…** button to open a list of interfaces. Select **CustomerService** and click the **OK** button.

6) For operation, click the drop-down button to populate the box with the **getCustomerInformation** operation.

7) For Message Category, click the box and select **Output** since you are now working on the response side. You will notice input is for the request side of the flow and output is for the response side of the flow.

8) The Message Type then gets populated when choosing input or output (request or response). The wizard should look like this:



Selected Message:

Interface: CustomerService

Operation: getCustomerInformation

Message Category: Output

Message Type: getCustomerInformationResponseMsg

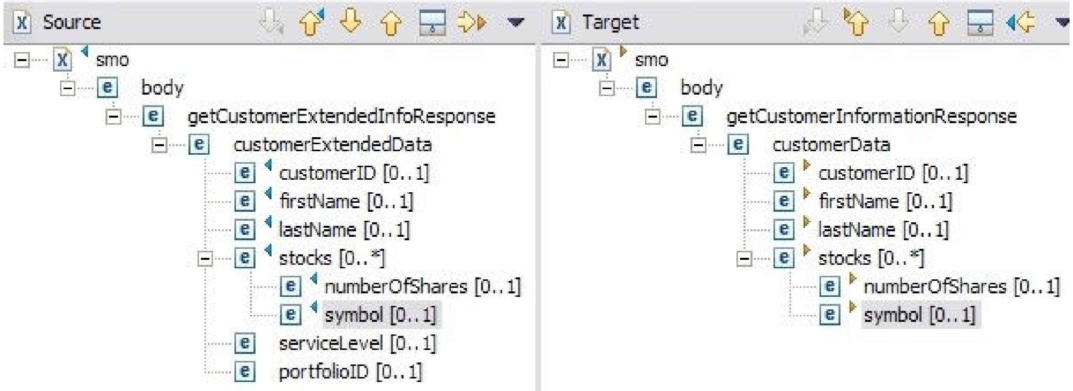
9) Click **OK**.

10) Now you should have **getCustomerExtendedInfoResponseMsg** for the Input Message Body and **getCustomerInformationResponseMsg** for the Output Message Body. Click **Finish**.

11) The mapping file XSLTransformation1_res_1.xmx will be created and opened. Fully expand both the **Source and Target objects**.

12) Click on **customerID** from the **Source** side and drag to the **customerID** on the **Target** side. Notice triangle icons appear next to the mapped elements and that the Overview view shows the mapping.

13) Do this same dragging action (from source to target) for **firstName, lastName, stocks, numberOfShares, and symbol**. Since stocks is an array, you have to map the array elements so that each array and array element is transformed.



14) Save the mapping by navigating to **File** > **Save** or hit **Ctrl + S** on your keyboard. **Close** the mapping file.

15) Back in the Details tab of Properties View for the XSL Transformation, click on the **Regenerate XSL button**. Click **OK** on the information message. You will see that XSLTransformation1_res_1.xsl has been generated.

16) Click on Mediation Flow and navigate to **File** > **Save** or hit **Ctrl + S** on your keyboard to save your **Mediation Flow**.

__ bb. Wire the output terminal of XSLTransformation1 to the CustomerService_getCustomerInformation_InputResponse.

1) Click on output terminal of XSLTransformation1 and drag to the CustomerService_getCustomerInformation_InputResponse.

__ cc. Click on Mediation Flow and navigate to **File** > **Save** or hit **Ctrl + S** on your keyboard to save your **Mediation Flow**.

Part 6: Test the CustomerServiceExtended backend

In this section you will run a JSP that enters a customerID to start off the process. However, for lab 2 you are going to test the new CustomerServiceExtended backend instead of the old CustomerService backend.

____ 1. You have previously started the server as part of the lab. But since you configured a JDBC data source, you need to restart the server.

Start **WebSphere Enterprise Service Bus Server** and **add modules** to server

Open **Servers View**.

__ a. Highlight WebSphere ESB Server v6.0 and click **Stop button**

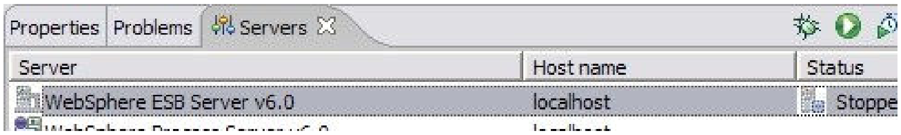
____ 2. Start **WebSphere Enterprise Service Bus Server** and **add modules** to server

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, to start the remote server.

If using a local testing environment:

__ a. Open **Servers View**.

__ b. Highlight WebSphere ESB Server v6.0 and click **Start button** (▶).

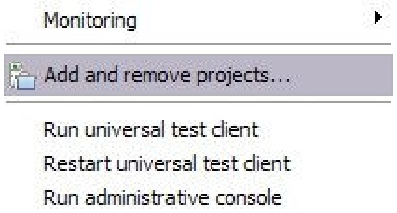


Server	Host name	Status
WebSphere ESB Server v6.0	localhost	Stoppe

NOTE: You may notice an error in the console view after the server has started. Please disregard and continue with lab. It is a problem that is currently being fixed.

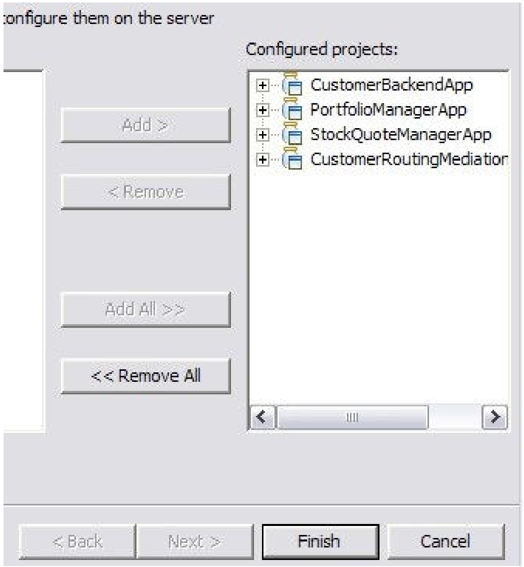
Add **projects** to WebSphere Enterprise Service Bus Server (once the ESB Server is started, not before).

__ c. In Servers view, right-click on WebSphere ESB Server v6.0 and select "**Add and Remove Projects..**."


Monitoring
Add and remove projects...
Run universal test client
Restart universal test client
Run administrative console

NOTE: Notice you are using an ESB profile and not a different server. Therefore, if you have projects with the same name deployed to the server, there may be some naming conflicts. Open Administrative Console and stop/uninstall those same-named projects before adding these projects to avoid errors.

__ d. Click **Add-All** button to move all projects to server and click **Finish** button. Wait for the deployment to finish



Configured projects:

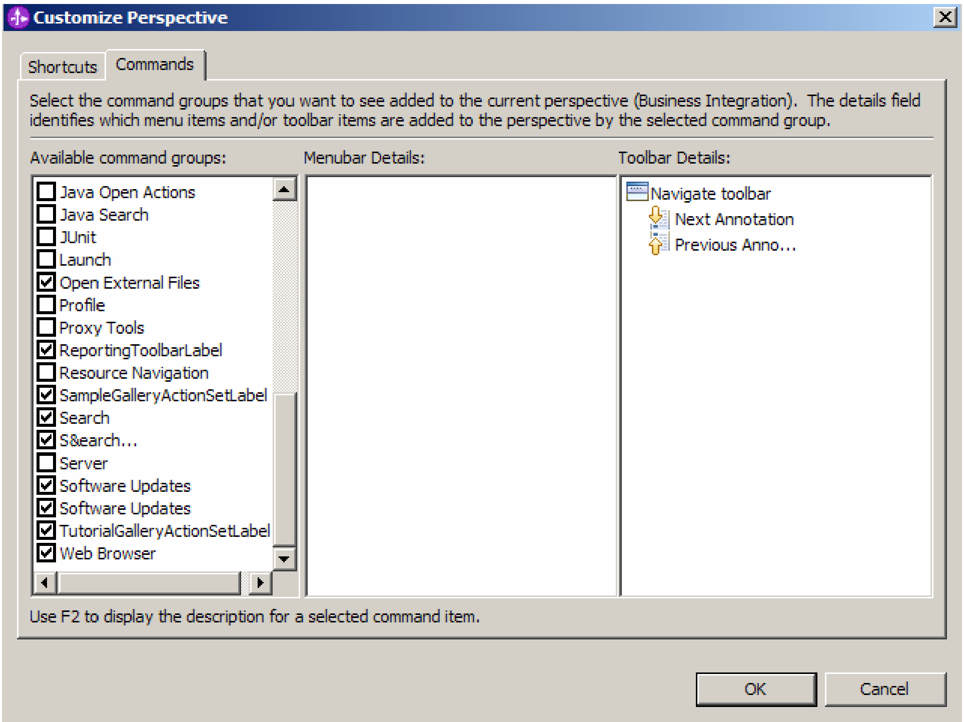
- CustomerBackendApp
- PortfolioManagerApp
- StockQuoteManagerApp
- CustomerRoutingMediatior

Enable web browser icon in WebSphere Integration Developer. You only need to do this once.

__ e. Go to **Window → Customize Perspective**

__ f. Click on the commands tab and scroll to the bottom

__ g. Click the check box next to Web browser



Select the command groups that you want to see added to the current perspective (Business Integration). The details field identifies which menu items and/or toolbar items are added to the perspective by the selected command group.

Available command groups:

☐ Java Open Actions
☐ Java Search
☐ JUnit
☐ Launch
☑ Open External Files
☐ Profile
☐ Proxy Tools
☑ ReportingToolbarLabel
☐ Resource Navigation
☑ SampleGalleryActionSetLabel
☑ Search
☑ S&earch...
☐ Server
☑ Software Updates
☑ Software Updates
☑ TutorialGalleryActionSetLabel
☑ Web Browser

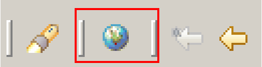
Menubar Details:

Toolbar Details:

Navigate toolbar
- Next Annotation
- Previous Anno...

Use F2 to display the description for a selected command item.

__ h. Click OK. This should put an icon for Web browser in the WebSphere Integration Developer tools panel.



__ i. Click on the web browser icon to launch a browser in WebSphere Integration Developer

__ j. 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: C\WID601\pf\esb\config\cells\esbCell\nodes\

__ k. Enter **7777777** (7 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.

__ l. Once you have the value of the customerID in the JSP, **Close** the browser.

__ m. Stop WebSphere ESB Server v6.0 by highlighting the WebSphere ESB Server v6.0 in the Servers View and click **Stop** button (■).
(FYI…You cannot view data in a Cloudscape database when server is running.)

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

Part 7: Save Work and Clean Up Server

____ 1. Export project as Project Interchange file

__ a. Navigate to **File -> Export**.

__ b. Select **Project Interchange**.

__ c. 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 into a workspace.

__ d. Save in C:/LabFiles601/WESB/Lab2/

__ e. Name the project interchange **WPIv601_ESB_FinishedLab2_PI.zip**.

If you are not continuing to Lab 3 right now, go ahead and clean up the **ESB Server**.

1. Start **WebSphere ESB Server v6.0** from the Servers view of the Business Integration perspective.

2. Right-click on WebSphere ESB Server v6.0 (once started) and select **Add and Remove projects...**

3. Select **Remove-All** and click **Finish**.

4. After remove is done, **stop** the WebSphere ESB Server v6.0.

What you did in this exercise

In this lab you added to the Message Logger mediation primitive with a couple others available in WebSphere Integration Developer V6.0.1. The main goal for Lab2 was to create another backend, CustomerServiceExtended, and use mediations to help decide which backend to use. You first added a Custom Mediation that was used to mine out a two digit prefix from the customer ID and place 2 digit it in the transient context (a business object) using a java snippet. You then added a Database Lookup primitive that uses that two digit prefix from transient context as key to lookup a backend identifier to also place into transient context. The Customer ID prefixes you used (77) went to a CustomerServiceExtended backend. To determine the message routing based on backend identifier, you added a Message Filter primitive. The old backend went directly to the callout for CustomerService and the new backend went to the XSL Transformation. The XSL Transformation transformed the CustomerService business object (BO) into the CustomerServiceExtended business object, then passed the newly formed message to the callout for CustomerServiceExtendedPartner instead of the CustomerServicePartner. On the response side, the CustomerServicePartner callout response went directly to the CustomerService input response. However, the CustomerServiceExtended callout response needed to go back through an XSL Transformation mediation in order to transform the response body from the CustomerServiceExtended business object back to the CustomerService business object.

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 **C:/LabFiles601/WESB/Lab2/ WPIv601_ESB_Lab2_solution_PI.zip**

__ c. Click **Finish** button.

Add the datasource from the first section of **Part 3**, the Database Lookup .

Continue past to **Part 6** to test the application.

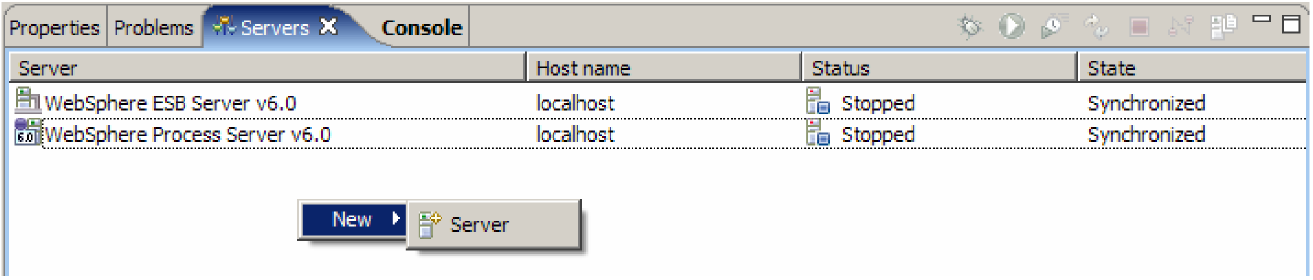
Task: Adding Remote Server to WebSphere Integration Developer Test Environment

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

____ 1. Create a new remote server.

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

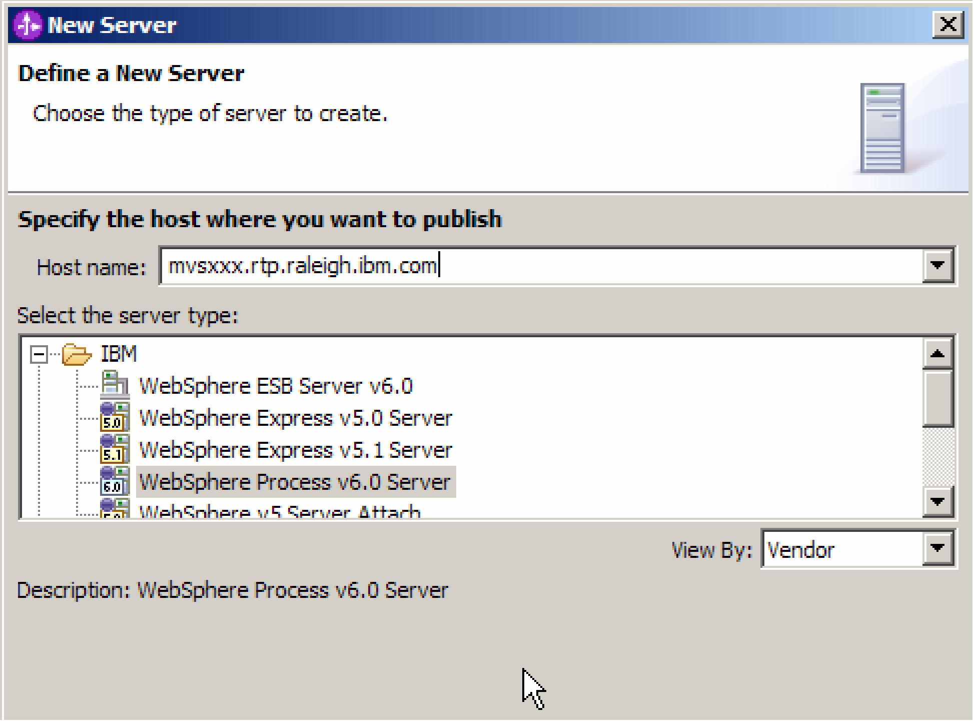
__ b. Select **New > Server.**



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:

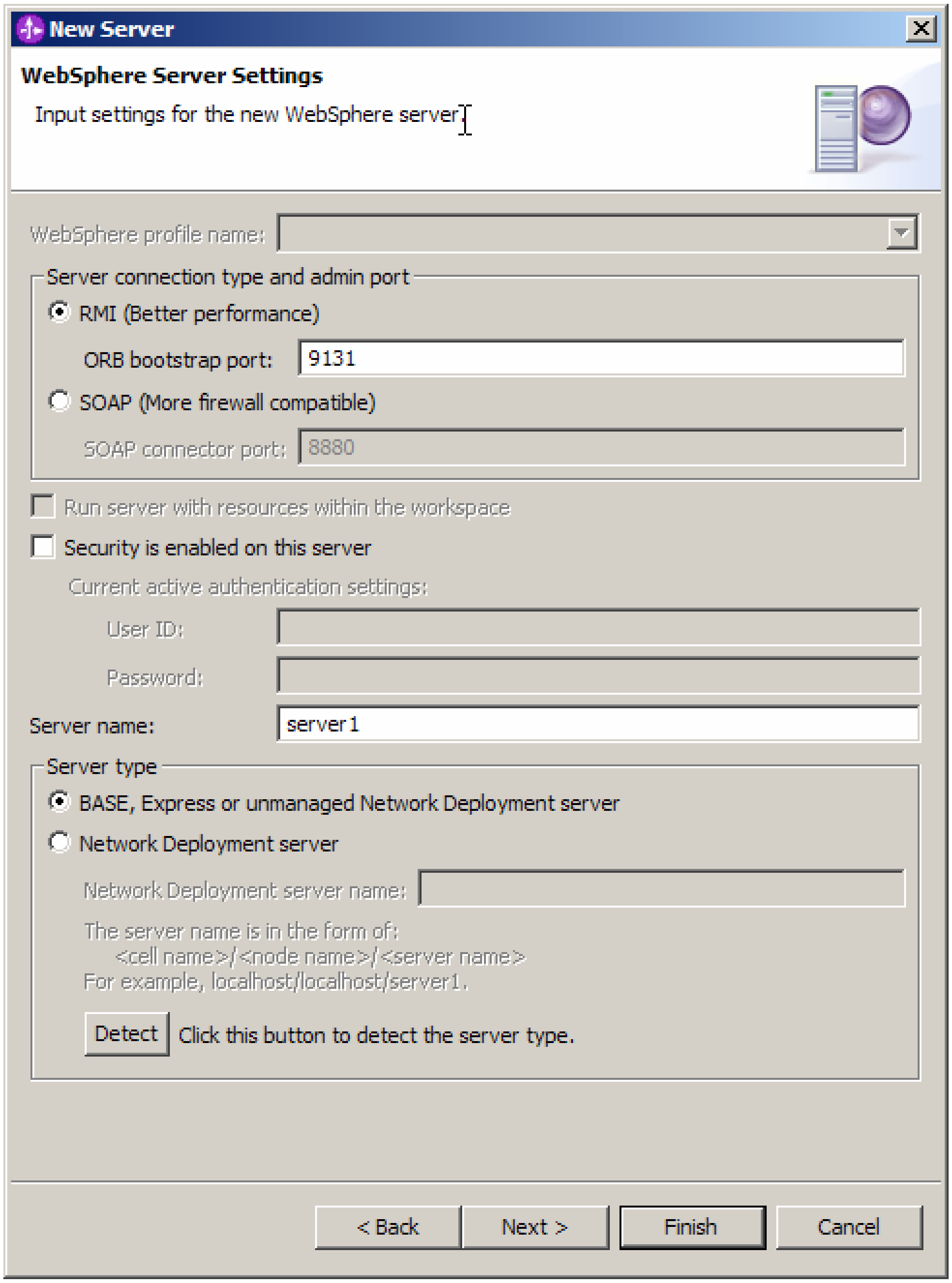
- IBM
  - WebSphere ESB Server v6.0
  - WebSphere Express v5.0 Server
  - WebSphere Express v5.1 Server
  - WebSphere Process v6.0 Server
  - WebSphere v5 Server Attach

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

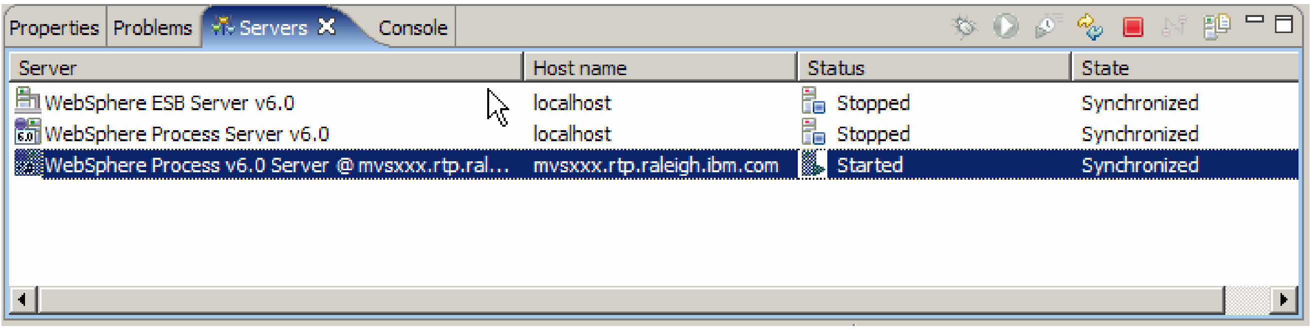
Network Deployment server name:

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

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

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

‘**telnet <HOSTNAME> <TELNET_PORT>**’

userid : **<USERID>**

pw : **<PASSWORD>**

If the remote server runs on a System i® server, sign on to the system and start a Qshell session.

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

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

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

__ l. 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
```

This page is left intentionally blank.