

IBM WEBSHERE 6.1 – LAB EXERCISE

# WebSphere Enterprise Service Bus 6.1 mediation programming model

## Lab Three – Fault recovery and service call retry

What this exercise is about .....	1
Lab requirements .....	2
What you should be able to do .....	2
Introduction .....	2
Exercise Instructions .....	3
Understanding how to read the instructions .....	4
Part 1: Setting up the environment for the lab .....	5
Part 2: Authoring the mediation flow to recover from a modeled fault.....	9
Part 3: Test the recover from modeled fault mediation.....	23
Part 4: Authoring the mediation flow to use service invoke retry.....	30
Part 5: Test the service call retry mediation.....	32
Part 6: Clean up the environment if you will not proceed to the next lab.....	36
What you did in this exercise .....	38
Solution instructions .....	39
Task: Adding remote server to the WebSphere Integration Developer test environment .....	40

### What this exercise is about

The objective of this lab is to provide you with an understanding of how to recover from service call failures. This is done in the context of a message splitting and aggregating mediation flow where you might not want the failure for the processing of a single element to terminate processing for all the other elements. The first thing examined is how to develop a flow that will continue to process repeating elements in a message regardless of a failure for one of the elements. Then the flow is enhanced to make use of service call retry capabilities which automatically attempt to recover from a failing call in an effort to get a successful result for every element.

This lab is provided **AS-IS**, with no formal IBM support.

## Lab requirements

- WebSphere Integration Developer V6.1 installed on Windows or Linux
- WebSphere Enterprise Service Bus V6.1 or WebSphere Process Server V6.1. The server can either be the test server installed by WebSphere Integration Developer or a remote server from a separate WebSphere Enterprise Service Bus V6.1 or WebSphere Process Server V6.1 installation.

## What you should be able to do

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

- Configure a mediation flow to recover from a modeled fault being returned from a service invoke primitive for a single element in an iterating flow.
- Configure a service invoke mediation primitive to use retry capabilities to automatically attempt to recover from a modeled fault and obtain a successful result.

---

## Introduction

This lab exercise is the third of a series of four tutorials intended to illustrate the new programming model for mediations introduced by WebSphere Enterprise Service Bus V6.1. The new programming model includes message augmentation using service invoke, splitting and aggregating to handle repeating elements within a message and service call retry capabilities including the use of alternate service endpoints.

The four tutorials are described in the presentation entitled [Augmentation, aggregation and retry tutorials](#). You should familiarize yourself with the tutorials as described in the presentation before attempting this lab.

## 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	Example Windows location	Example Linux location
<WID_HOME>	C:\Program Files\IBM\WID61	/opt/IBM/WID61
<LAB_FILES>	C:\Labfiles61\WESB\61ProgModel	/tmp/Labfiles61/WESB/61ProgModel

## Instructions if using a remote server for testing

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 the remote test environment:

Reference variable	Example: Remote Windows test server location	Example: Remote z/OS <sup>®</sup> 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<sup>®</sup> or Solaris, can be found at the end of this document, in the section [“Task: Adding remote server to the WebSphere Integration Developer test environment”](#).

---

## Understanding how to read the instructions

In this lab, the instructions are written to allow an experienced user to complete the steps easily while at the same time providing very explicit instructions needed by the new user. The format of the instructions follows this pattern:

- \_\_\_ 1. This is a sentence or short paragraph that describes a particular task to be completed. In some cases this is sufficient for an experienced user, but in other cases the experienced user might require some additional specific information to complete the task. In that case, there is a bulleted list that helps the experience user know specifics.
- **Additional information for experienced user**
  - **This information, along with the above paragraph, should allow the experienced user to complete the task**
  - -----
- \_\_\_ a. First step needed by the new user
- \_\_\_ b. Second step needed by the new user
- 1) Additional details for completing this step
  - 2) More details for completing this step
- \_\_\_ c. Third step needed by new user
- \_\_\_ 2. Next task to be completed
- **Info for experience user**
  - -----
- \_\_\_ a. First step needed by the new user
- \_\_\_ b. Second step needed by the new user.

## Part 1: Setting up the environment for the lab

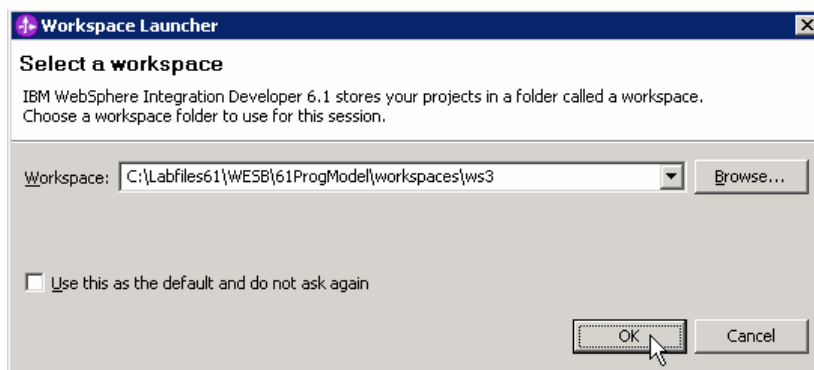
**What you will do in this part:** In this part you are getting the environment set up to do the lab. There are three different ways you might be approaching this which will dictate what you need to do.

**(1) Proceeding from Lab Two** – You are directly continuing from Lab Two and you did not shut down the server and development environment. In this case, there is nothing that needs to be done in this part.

**(2) Restarting from Lab Two** – You are continuing from Lab Two which you did previously and therefore you did shut down the server and development environment. In this case, you will need to restart the development environment and server but will not need to import a project interchange to initialize the workspace.

**(3) Directly starting Lab Three** – You are starting this lab from scratch, regardless of whether you had previously completed Lab 2.

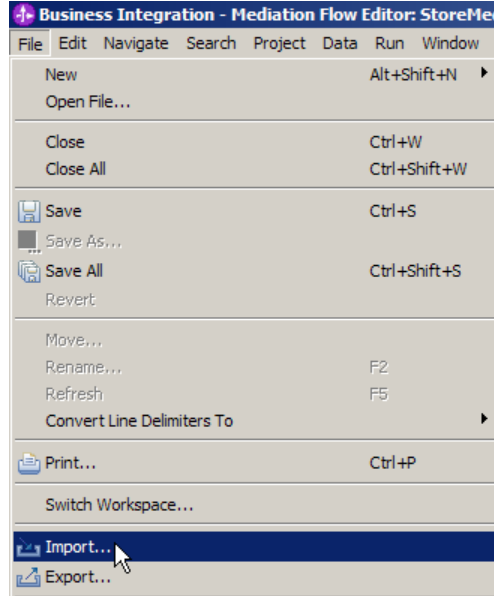
- \_\_\_ 1. If you are **proceeding from Lab Two** there is nothing to do, skip directly to **Part 2 Authoring the mediation flow to recover from a modeled fault**
  - -----
- \_\_\_ 2. Start WebSphere Integration Developer.
  - **Restarting from Lab 2:** Use the same workspace used in Lab 2
  - **Directly starting Lab 3:** Suggested location:  
 <LAB\_FILES>/workspaces/ws3
  - -----
- \_\_\_ a. Select **Start → All Programs → IBM WebSphere Integration Developer → IBM WebSphere Integration Developer V6.1 → WebSphere Integration Developer V6.1**
- \_\_\_ b. From the Workspace Launcher window, enter the name of the workspace in the Workspace field
  - 1) Restarting from Lab 2: Use the same workspace used in Lab 2
  - 2) Directly starting Lab 3: <LAB\_FILES>/workspaces/ws3



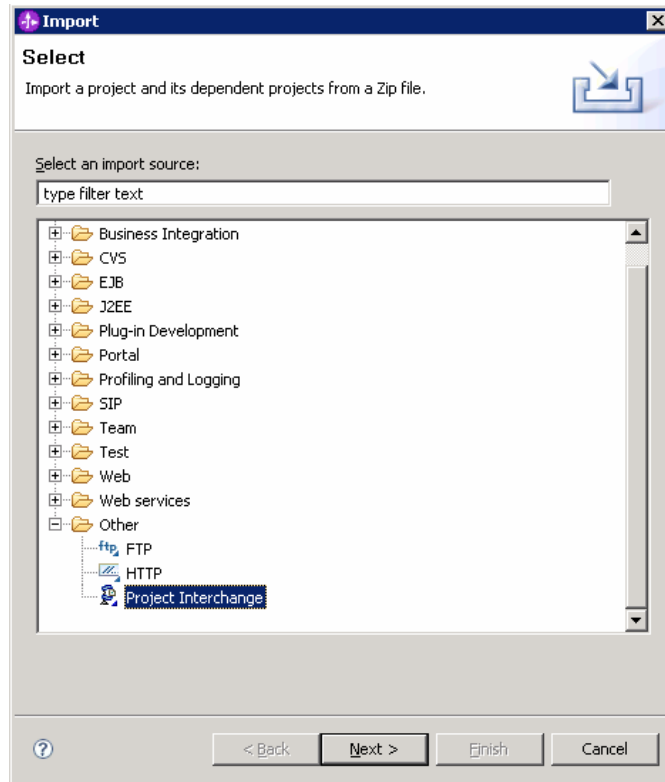
- \_\_\_ c. If the Welcome panel is displayed, click on **Go to the Business Integration perspective** or the arrow next to it in the upper right corner of the panel.



- \_\_\_ 3. If you are **restarting from Lab Two** there is nothing additional to do, skip directly to **Part 2 Authoring the mediation flow to recover from a modeled fault**
- -----
- \_\_\_ 4. If you are **directly starting Lab Three** import the project interchange file containing the starting point for the lab
- <LAB\_FILES>/PI3-AggregateSolution-RetryStart.zip
  - -----
- \_\_\_ a. From the menu, select **File → Import...**

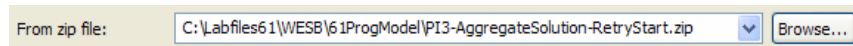


\_\_ b. In the **Import** dialog, select **Other** → **Project Interchange**

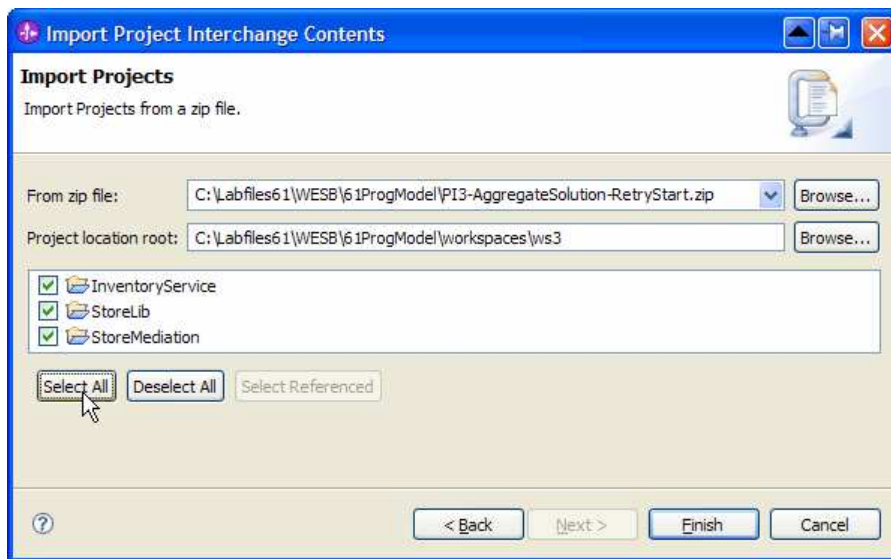


\_\_ c. Click **Next**

\_\_ d. In the **Import Project Interchange Contents** dialog set the **From zip file:** value to **<LAB\_FILES>/PI3-AggregateSolution-RetryStart.zip**

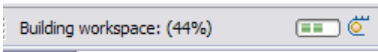


\_\_ e. Click **Select All** to selected all of the projects listed



\_\_\_ f. Click **Finish**

\_\_\_ g. Wait for the build process to complete for the imported projects, which is seen at the lower right corner of WebSphere Integration Developer.



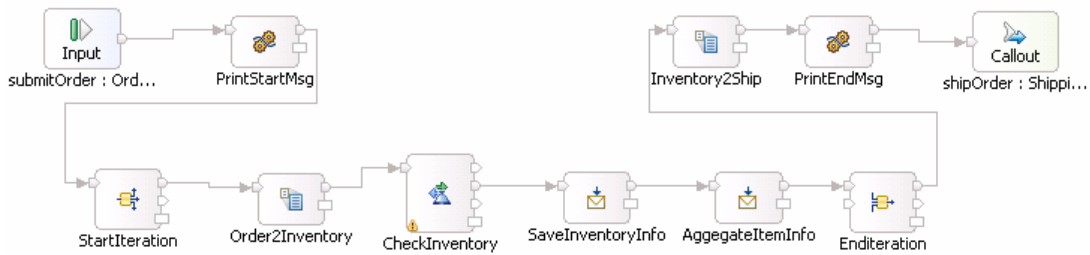


## Part 2: Authoring the mediation flow to recover from a modeled fault

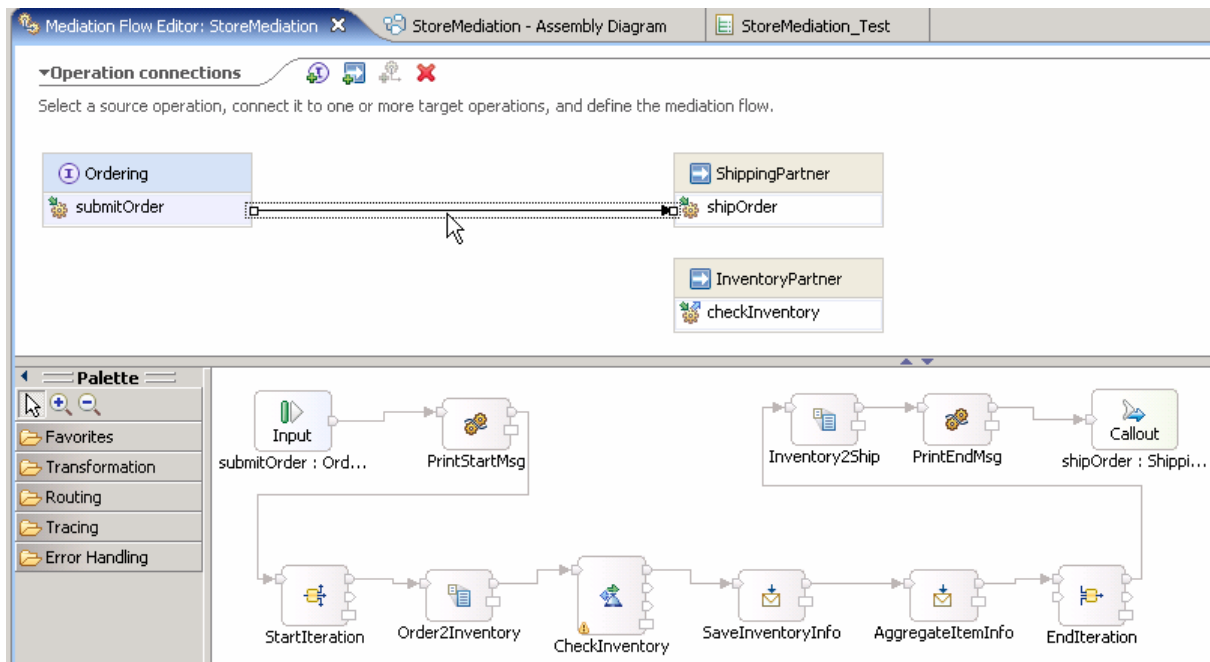
**What you will do in this part:** In this part, you will modify the flow to handle the situation when a modeled fault is returned by the inventory service. The requirement is for the modeled fault is, not to cause the entire flow to fail. The item which resulted in a modeled fault during the call to the inventory service is included in the aggregation of all items. However, because of the fault, the item will not contain the augmented information from the inventory service.

See the presentation entitled [Augmentation, aggregation and retry tutorials](#) to better understand what this part is doing.

1. Open the StoreMediation flow found in the StoreMediation module.



- a. In the Business Integration view, expand **StoreMediation** → **Mediation Logic** → **Flows** and then double-click on **StoreMediation** to open it in the mediation flow editor
- b. The line connecting **Ordering** → **submitOrder** and **ShippingPartner** → **shipOrder** should be selected for you to view the flow



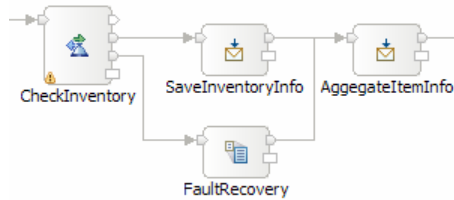
\_\_\_ 2. Add an XSL transformation primitive to the canvas and wire it into the flow. This primitive provides the required transformations to convert a fault returned by the inventory service so that the item can still be processed rather than allowing the flow to fail. However, because of the fault, the item will only have the order information and will not have the inventory information included.

- **Display Name:** FaultRecovery
- **Wire** : CheckInventory(checkInventory\_InventoryFaultMsg terminal) to FaultRecovery
- **Wire** : FaultRecovery to AggregateItemInfo

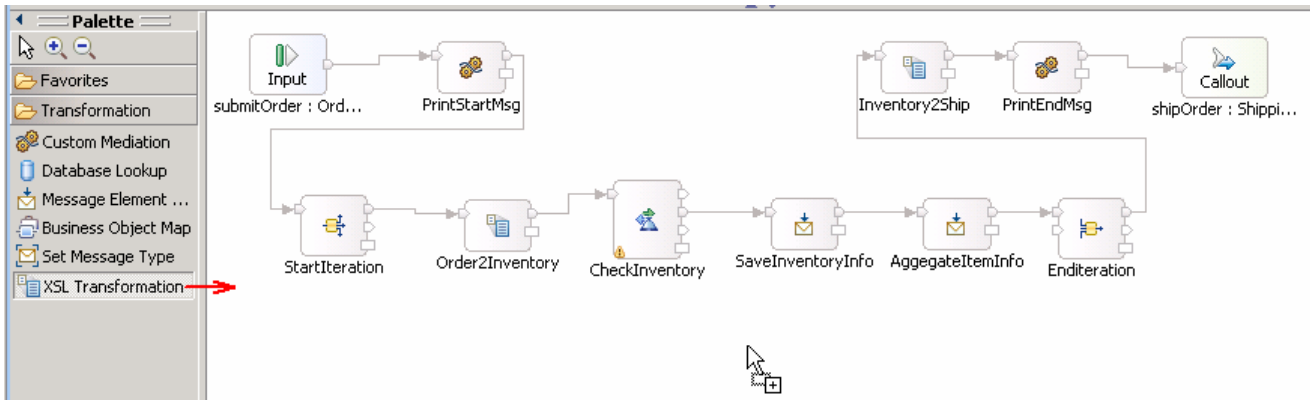
---

**NOTE:** If you are unable to create the wire from the FaultRecovery to the AggregateItemInfo, get a pop up menu from the **out** terminal of FaultRecovery and select '**Reset Message Type**'.

---

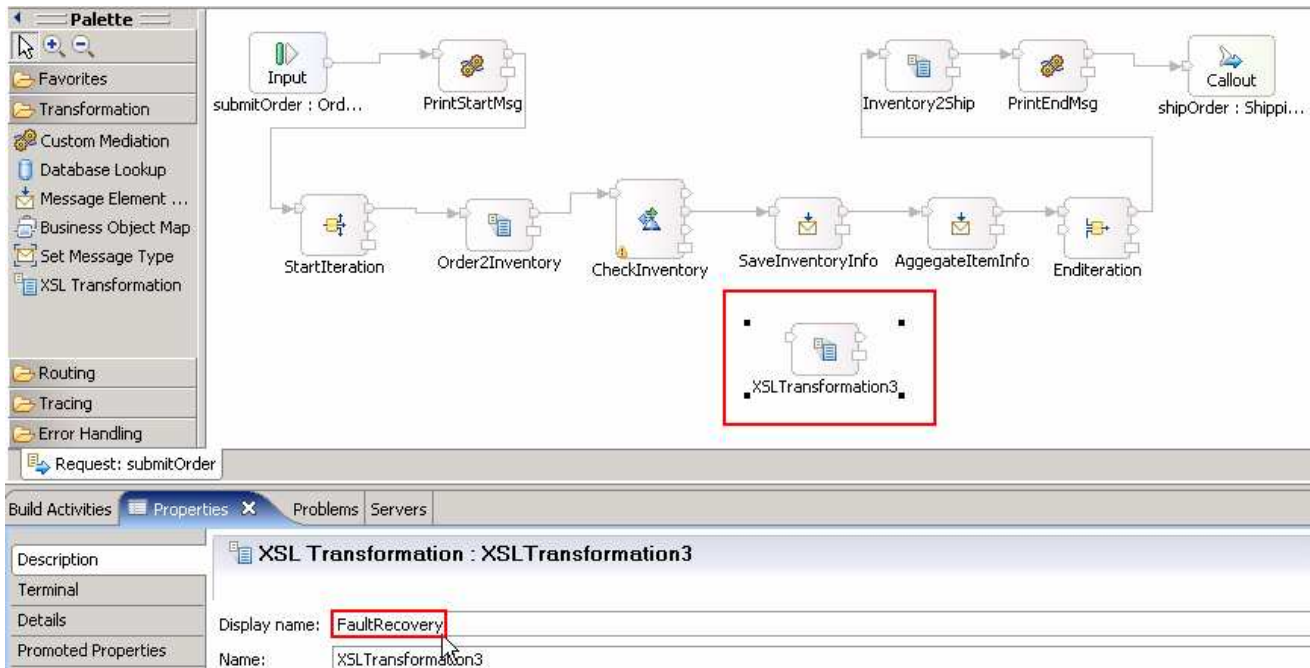


\_\_\_ a. From the **Palette**, select **Transformation → XSL Transformation** and then click on the canvas below the SaveInventoryInfo primitive



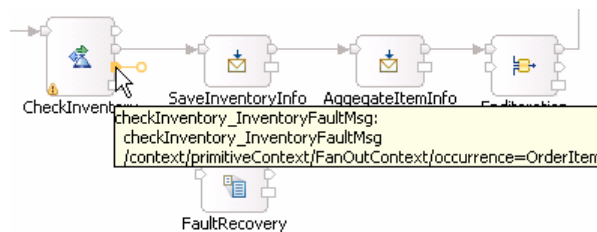
\_\_\_ b. You will now see a new XSL Transformation primitive added to the flow. Ensure that this primitive is highlighted and select **Properties → Description**

\_\_\_ c. Change the **Display name** to **FaultRecovery**

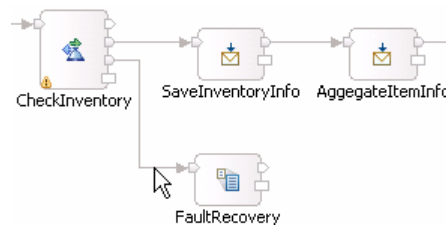


\_\_\_ d. Add a connection from CheckInventory(checkInventory\_InventoryFaultMsg terminal) to FaultRecovery

- 1) Hover your mouse over **checkInventory\_InventoryFaultMsg** terminal of **CheckInventory** primitive to get the 'Add a connection to an input terminal'



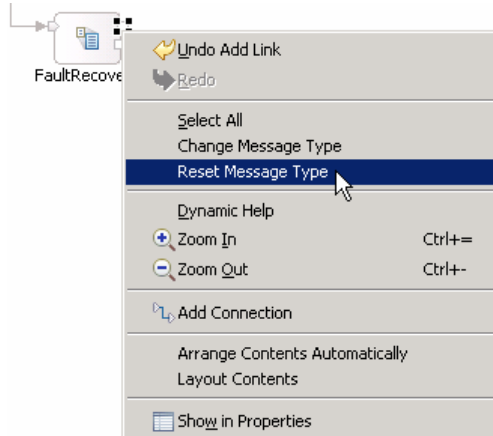
- 2) Click on the orange bubble and then click on **FaultRecovery** primitive
- 3) You should now see a connection from CheckInventory(checkInventory\_InventoryFaultMsg terminal) to FaultRecovery



\_\_\_ e. Similarly, add a connection from **FaultRecovery** to **AggregateItemInfo** primitive

- 1) Hover your mouse over **out** terminal of **FaultRecovery** to get the 'Add a connection to an input terminal'
- 2) Click on the orange bubble and then click on **AggregateItemInfo** primitive.

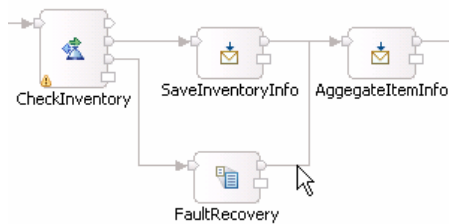
**NOTE:** If you are unable to create the wire from the FaultRecovery to the AggregateItemInfo, right click over the **out** terminal of FaultRecovery primitive and then select '**Reset Message Type**' from the pop-up menu as shown below:



→ Click **OK** over the '**Reset Message Type**' pop-up dialog and continue to reset the message type.

→ Continue with **step e** to add a connection from **FaultRecovery** to **AggregateItemInfo** primitive.

\_\_ f. Your flow now should have these connections:



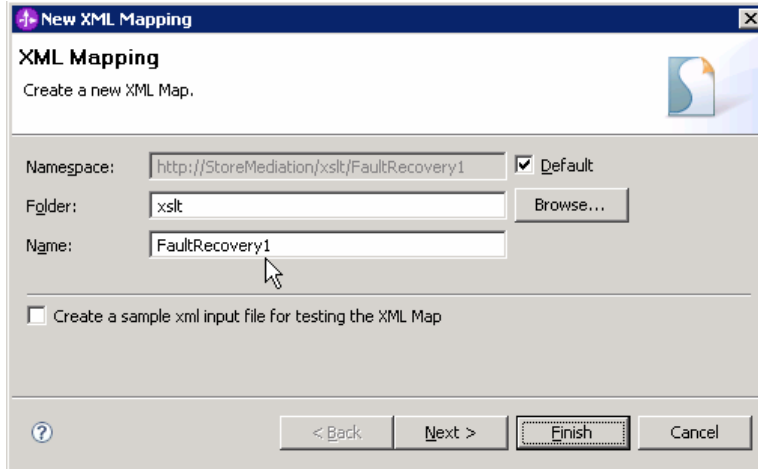
\_\_\_ 3. In the FaultRecovery XML transformation primitive, create a new XML mapping file used to define the XSL transformation.

- **Map Name** : **FaultRecovery1**
- **Message Root** : **/**
- **Input Message Body** : **checkInventory\_InventoryFaultMsg**
- **Output Message Body**: **checkInventoryResponseMsg**
- -----

\_\_ a. Select **FaultRecovery** primitive from the canvas and then select **Properties → Details**

\_\_ b. Click on **New...** next to '**Mapping file**'. The New XML Mapping wizard is opened.

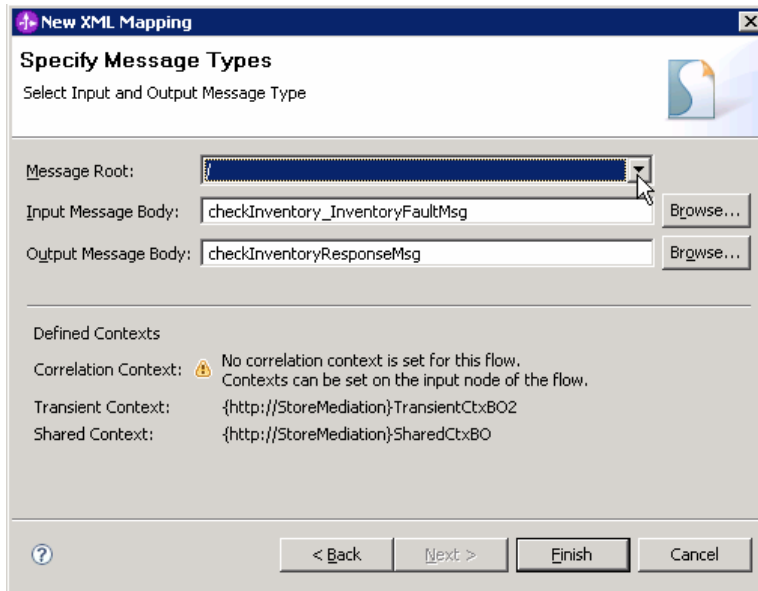
\_\_ c. For **Name** enter **FaultRecovery1**



\_\_\_ d. Click **Next**


\_\_\_ e. From Specify Message Types panel, enter:

- 1) For **Message Root**, select **'/'** from the drop down list
- 2) For **Input Message Body**, accept the default selection: **checkInventory\_InventoryFaultMsg**
- 3) For **Output Message Body**, accept the default selection: **checkInventoryResponseMsg**



\_\_\_ f. Click **Finish**. The FaultRecovery1.map file is opened in the XML Mapping editor.

\_\_\_ 4. Define the mapping for the FaultRecovery1 map. The mapping must address (1) moving fields between the source and target SMOs that will not change, (2) setting the inventory information in the transient context (indicating a failing call) and (3) setting the message body (setting the message body is not really required as it is not used again in the flow, but is done for completeness of the transformation).

- Use toolbar icon (  ) **"Map source to target based on name and types"**

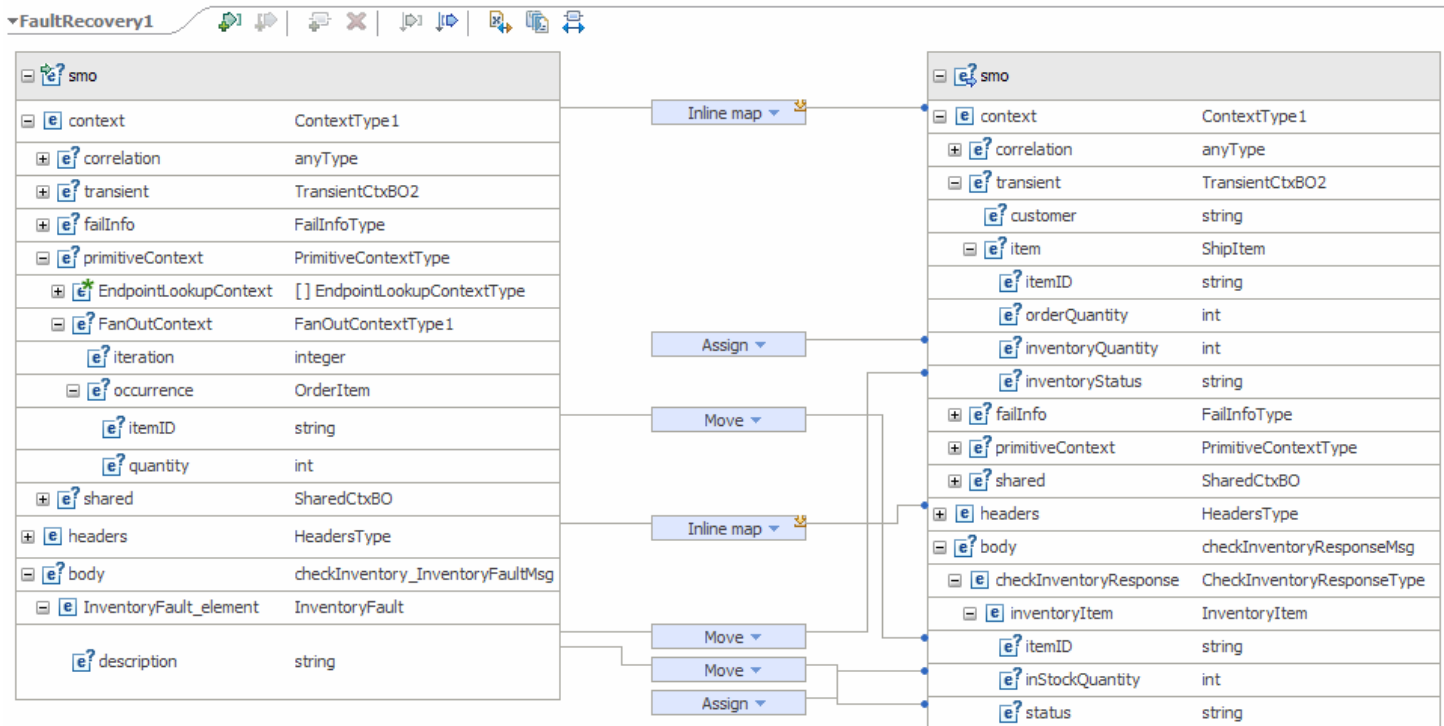
- Navigate down into the context/transient/item to context/transient/item inline map
- Remove the inventoryQuantity to inventoryQuantity move transformation
- Remove the inventoryStatus to inventoryStatus move transformation
- At the SMO level (top level map) add these assign transforms

**NOTE:** To create an assign transform, right click on the target element and select **Create Transform** from the pop-up menu. Set the value in **Properties** → **General** panel.

Value	Target (right side)
0	context/transient/item/inventoryQuantity
0	body/checkInventoryResponse/inventoryItem/inStockQuantity

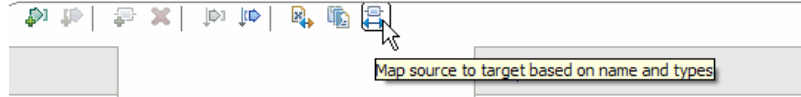
- At the SMO level (top level map) add these move transforms

Source (left side)	Target (right side)
body/InventoryFault_element/description	context/transient/item/inventoryStatus
context/primitiveContext/FanOutContext/occurrence/itemID	body/checkInventoryResponse/inventoryItem/itemID
body/InventoryFault_element/description	body/checkInventoryResponse/inventoryItem/status

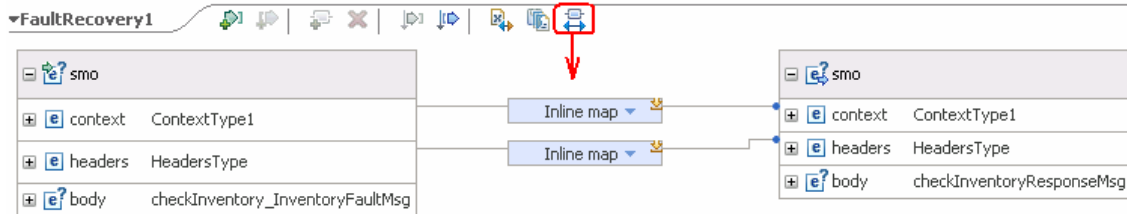


-----

\_\_\_ a. Click on **'Map source to target based on name and types'** icon (↔)

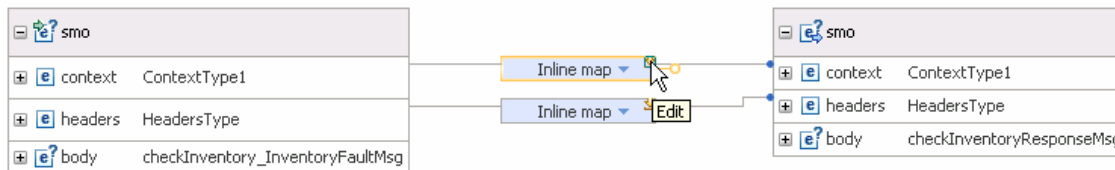


\_\_\_ b. This will map context and headers using inline maps as shown below:

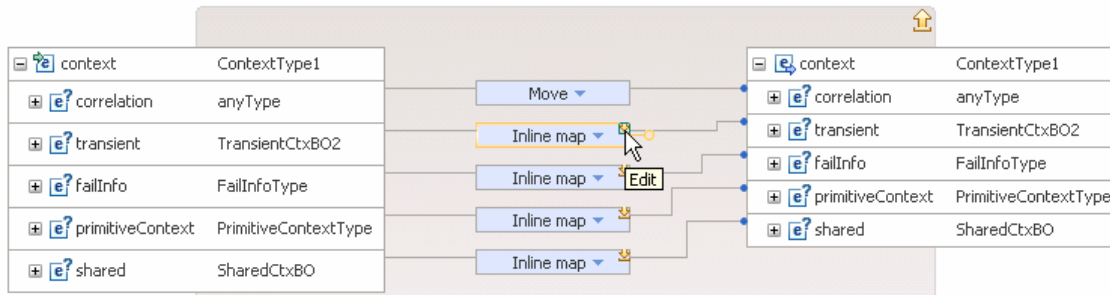


\_\_\_ c. Navigate down into the context/transient/item to context/transient/item inline map and delete the two move transformations

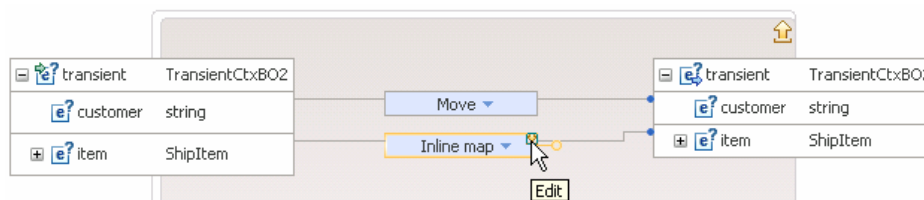
1) Click on **Edit** (🔧) icon on the Inline map connecting **context** attributes



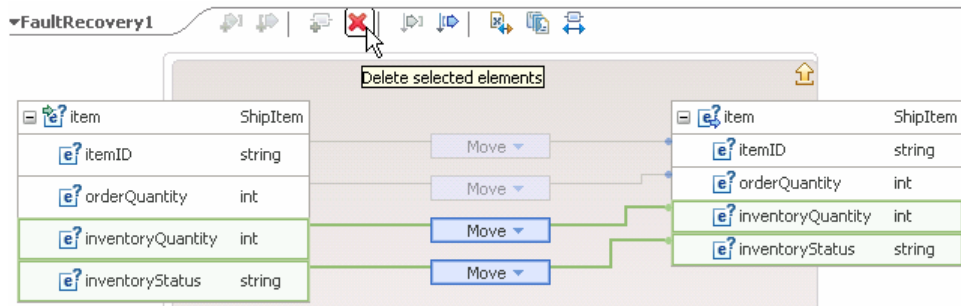
2) This will open all the transformations defined under context. Click on **Edit** (🔧) icon on the Inline map connecting **transient** attributes:



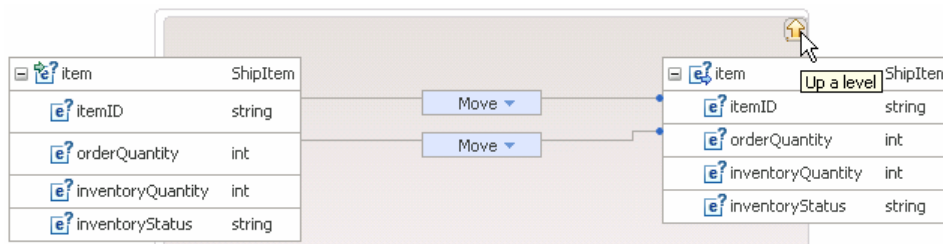
3) This will open the transformations of transient. Click on **Edit** (🔧) icon on the Inline map connecting **item** attributes:



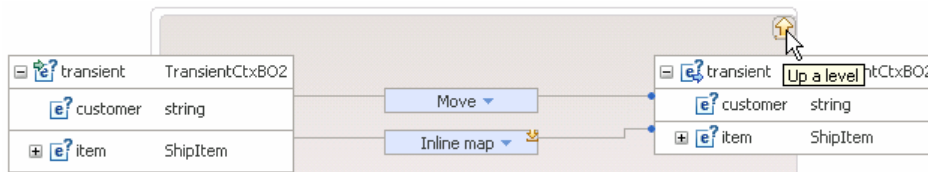
- 4) Select the two **Move** transformations (hold down Ctrl button on keyboard and click on each Move transform) for inventoryQuantity and inventoryStatus and then hit **'Delete selected elements'** (✖) icon from the top:



- 5) Click on **'Up a level'** (⬆) icon which will bring you one level up to the **transient** mapping



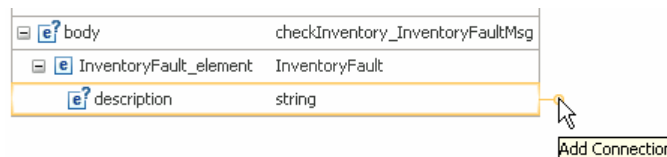
- 6) Click on **'Up a level'** (⬆) icon again which will bring you one level up to the **context** mapping



- 7) Click on **'Up a level'** (⬆) icon one more time which will now bring you back the **smo** mapping level

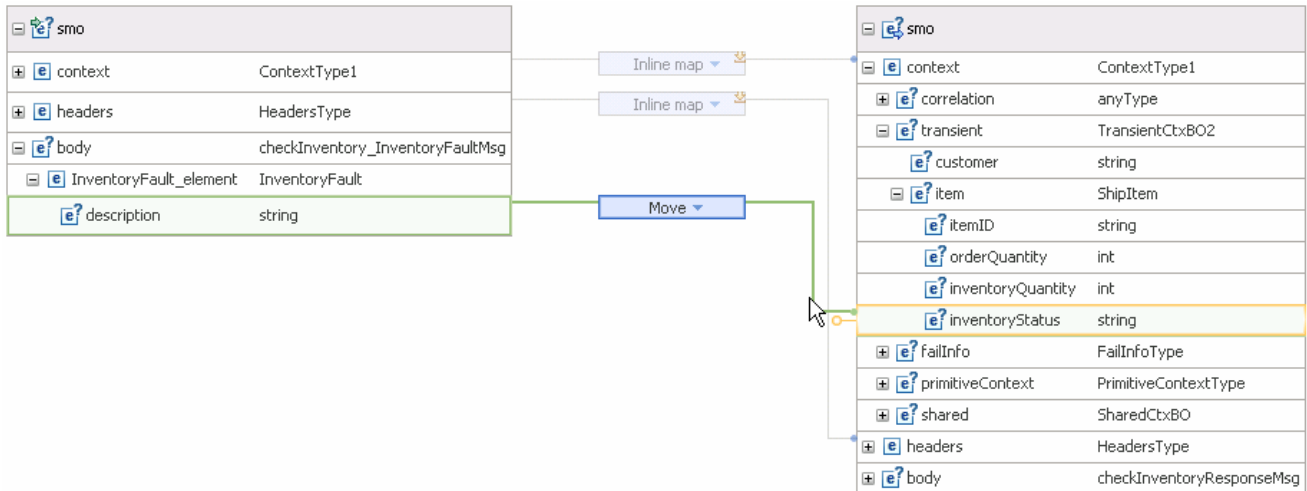
\_\_\_ d. Move body/InventoryFault\_element/description to context/transient/item/inventoryStatus

- 1) In the left smo, expand **body** → **InventoryFault\_element**
- 2) In the right smo, expand **context** → **transient** → **item**
- 3) In the left smo, hover your mouse over **description** and click on **Add Connection**:



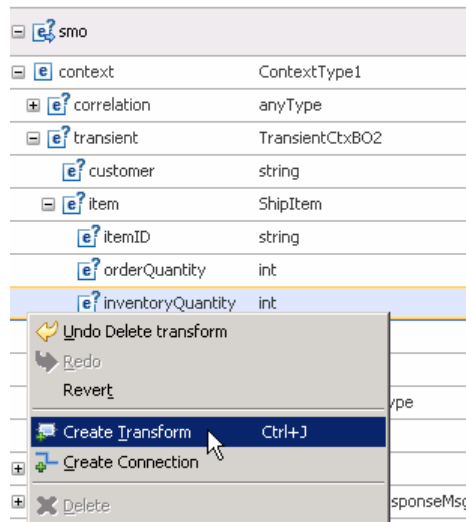


4) Now click on **inventoryStatus** in the right smo and you should see a new connection from description to inventoryStatus:



\_\_\_ e. Assign the value 0 to context/transient/item/inventoryQuantity

- 1) In the right smo, expand **context** → **transient** → **item**
- 2) Right-click on **inventoryQuantity** and select **Create Transform** from the pop-up menu



3) You will now see a new Transform 'Assign' for inventoryQuantity. Ensure that the above defined Assign is highlighted and then select **Properties** → **General**

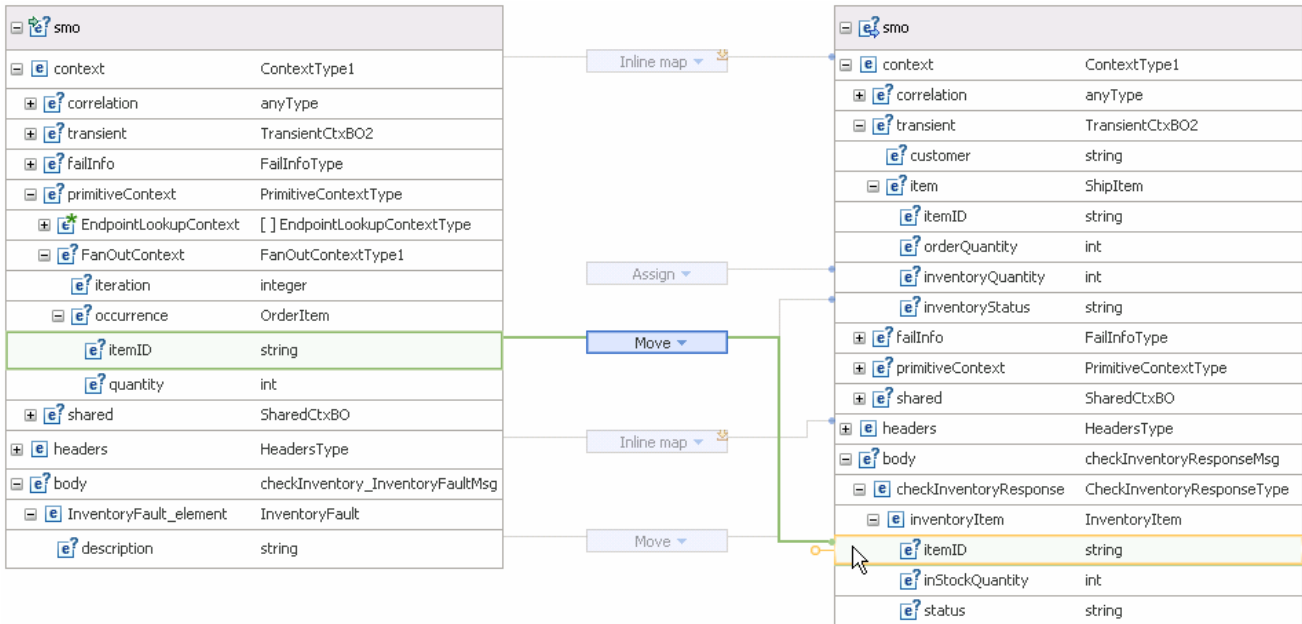
#### 4) Enter '0' for Value

The screenshot displays the configuration of a Transform - Assign activity. On the left, the source SMO (Service Model Object) is expanded to show 'InventoryFault\_element' with a 'description' property of type 'string'. On the right, the target SMO is expanded to show 'item' with an 'inventoryQuantity' property of type 'int'. A red box highlights the 'Assign' activity, and a green arrow indicates the mapping from the source 'description' to the target 'inventoryQuantity'. Below the main workspace, the 'Transform - Assign' activity properties are visible, with the 'Value' field set to '0'.

\_\_\_ f. Now, move context/primitiveContext/FanOutContext/occurrence/itemID to /body/checkInventoryResponse/inventoryItem/itemID

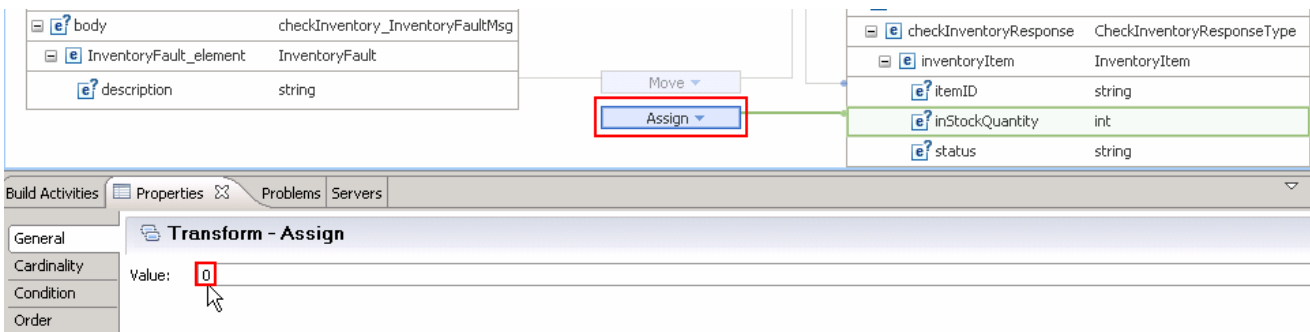
- 1) In the left smo, expand **context** → **primitiveContext** → **FanOutContext** → **occurrence**
- 2) In the right smo, expand **body** → **checkInventoryResponse** → **inventoryItem**
- 3) In the left smo, hover your mouse over **itemID** and click on **Add Connection**

4) Now click on **itemID** in the right smo and you should see a new connection from itemID to itemID:



\_\_\_ g. Assign the value 0 to body/checkInventoryResponse/inventoryItem/inStockQuantity

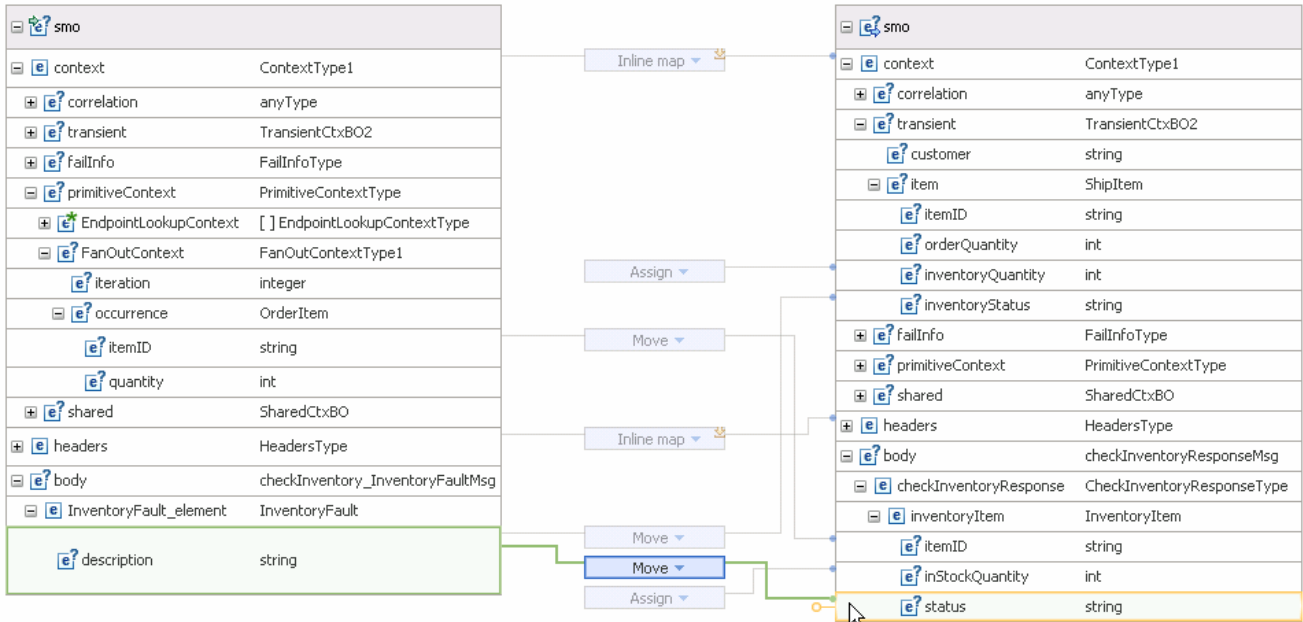
- 1) In the right smo, expand **body** → **checkInventoryResponse** → **inventoryItem**
- 2) Right-click on **inStockQuantity** and select **Create Transform** from the pop-up menu
- 3) You will now see a new Transform 'Assign' for inStockQuantity. Ensure that the above defined Assign is highlighted and then select **Properties** → **General**
- 4) Enter **'0'** for **Value**



\_\_\_ h. Finally, move body/InventoryFault\_element/description to body/checkInventoryResponse/inventoryItem/status

- 1) In the left smo, expand **body** → **InventoryFault\_element**
- 2) In the right smo, expand **body** → **checkInventoryResponse** → **inventoryItem**
- 3) In the left smo, hover your mouse over **description** and click on **Add Connection**:

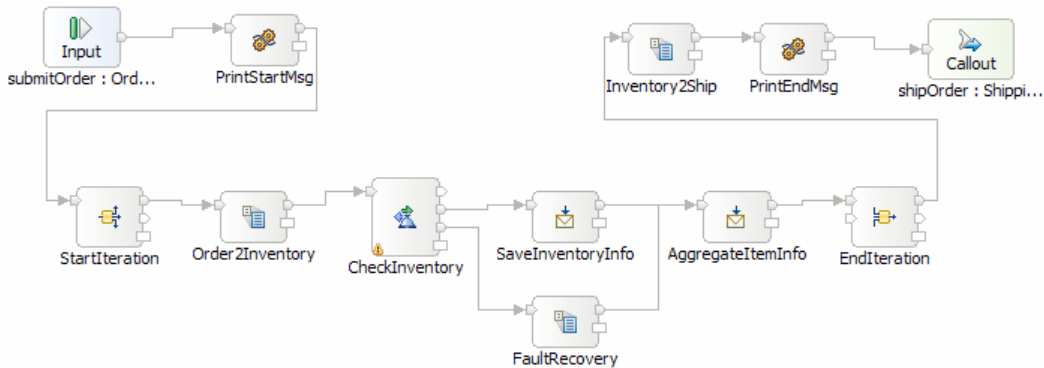
4) Now click on **status** in the right smo and you should see a new connection from description to status:



\_\_\_ i. From the menu select **File → Save** (or **Ctrl + S** from your keyboard) to save your changes to the map

\_\_\_ j. Click on **X** to close FaultRecovery1.map file

\_\_\_ 5. Ensure that the resulting flow looks like this.



\_\_\_ 6. Modify the InventoryPartner import in the StoreMediation assembly so that it calls the inventory service export implementation that will sometimes work and sometimes return a modeled fault. Using this implementation will allow the testing to illustrate the results when CheckInventory is successful and when it returns a fault.

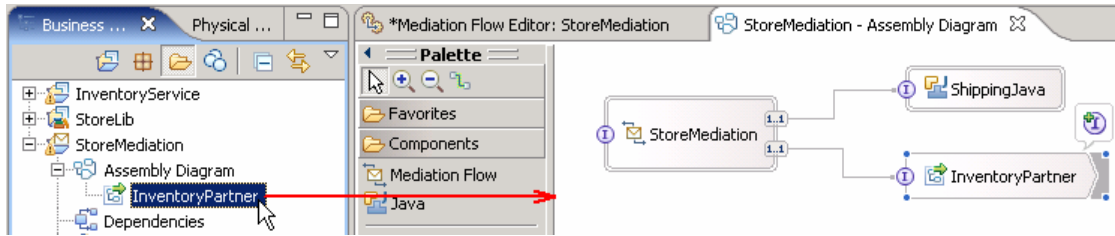
- **InventoryPartner import, Export name: InvRandom**

**Import: InventoryPartner (SCA Binding)**

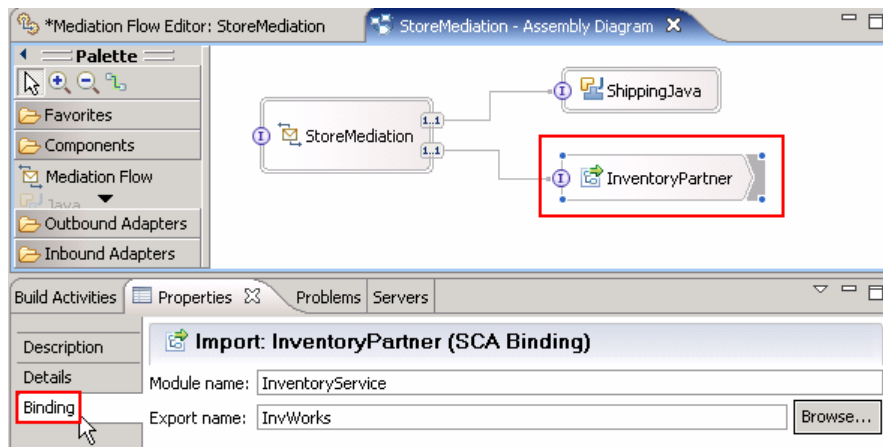
Module name:

Export name:

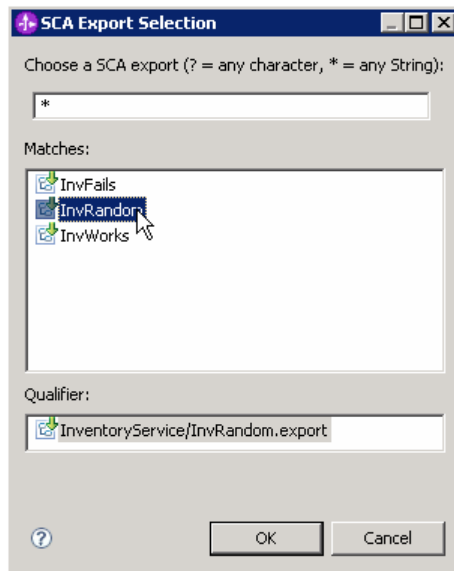
- \_\_\_ a. In the Business Integration window, expand **StoreMediation** and double-click on **Assembly Diagram** to open it in Assembly editor



- \_\_\_ b. Select **InventoryPartner** import component from Assembly diagram, and then select **Properties Binding**

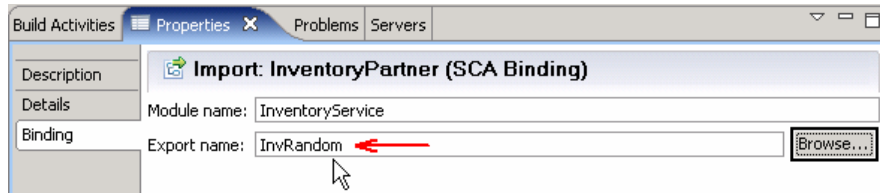


- \_\_\_ c. Click on **Browse...** for **Export name**. An SCA Export Selection window is opened
- \_\_\_ d. Select **InvRandom** under Matches

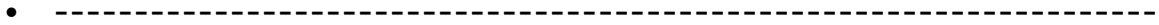


- \_\_\_ e. Click **OK**

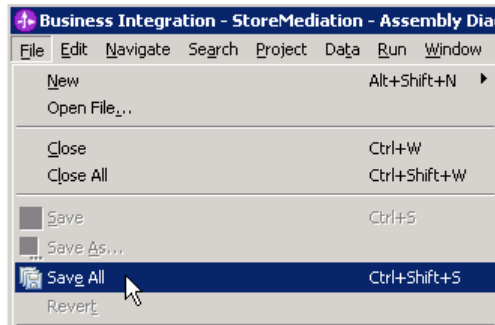
\_\_\_ f. Export name is now populated with InvRandom:



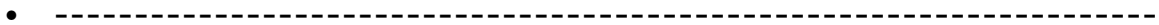
\_\_\_ 7. Check that all the artifacts have been saved.



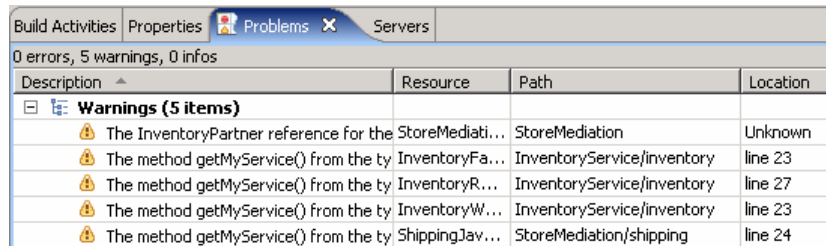
\_\_\_ a. From the menu select **File → Save All** to save your changes



\_\_\_ 8. Check that there are no errors reported in the Problems view.



\_\_\_ a. Select **Problems** view at the bottom. You can ignore warnings, but there should not be any errors at this time:




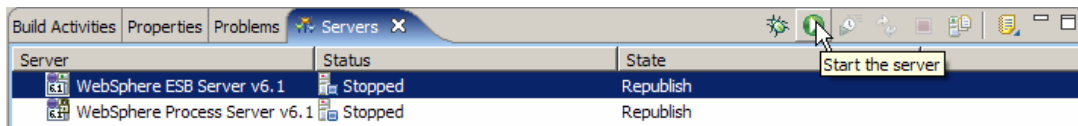
## Part 3: Test the recover from modeled fault mediation

**What you will do in this part:** In this part you use the component test facilities of WebSphere Integration Developer to test the mediation. The resulting output from the test is explained.

See the presentation entitled [Augmentation, aggregation and retry tutorials](#) to better understand what this part is doing.

- \_\_\_ 1. If not already running, start the WebSphere Enterprise Service Bus (or WebSphere Process Server) test server.

- \_\_\_ a. From the **Servers** view of WebSphere Integration Developer, select **WebSphere ESB Server v6.1** and hit 'Start the server' icon (  ) from the toolbar



- \_\_\_ b. Wait until the server Status shows as **Started**

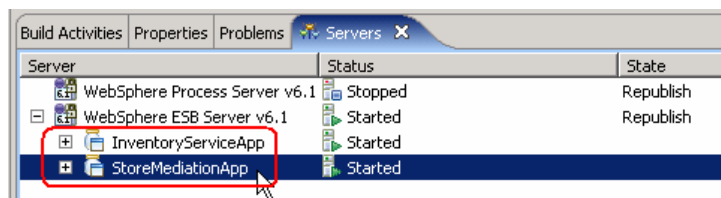


**NOTE:** Depending upon the preferences you have specified for the **Console** view, the **Console** view might grab the focus from the **Servers** view. If this is the case, the status in the lower right should indicate when the server startup is complete, at which time you can switch back to the **Servers** view

- \_\_\_ 2. Check to see if the InventoryServiceApp and StoreMediationApp are deployed on the server.

- **No, not deployed yet:** Add both projects to the server
- **Yes, already deployed:** Restart the StoreMediationApp
- -----

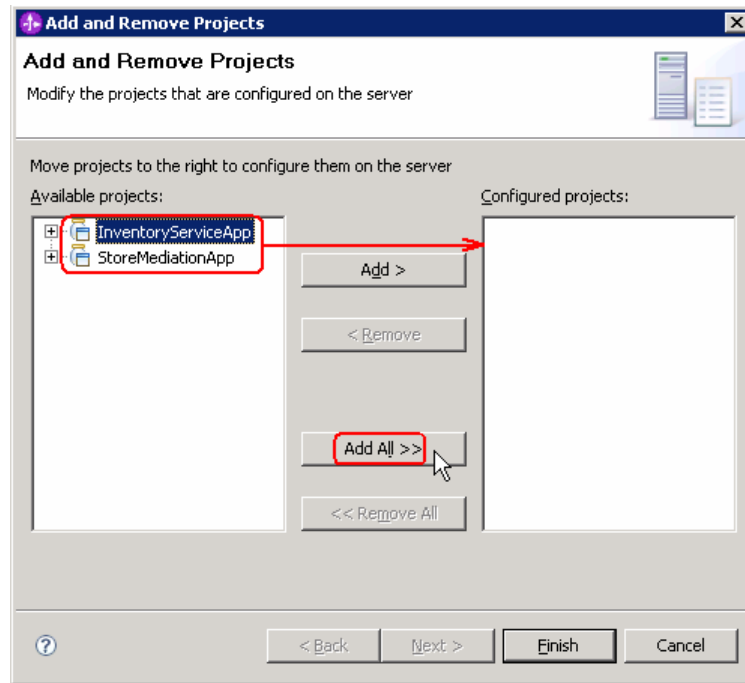
- \_\_\_ a. Check the Servers view to see if the InventoryServiceApp and StoreMediationApp are already deployed. If they are, they will appear below the server as shown here. **Follow the appropriate instructions in step b or step c**



- \_\_\_ b. If the applications are **not deployed yet**, add both projects to the server following these steps:

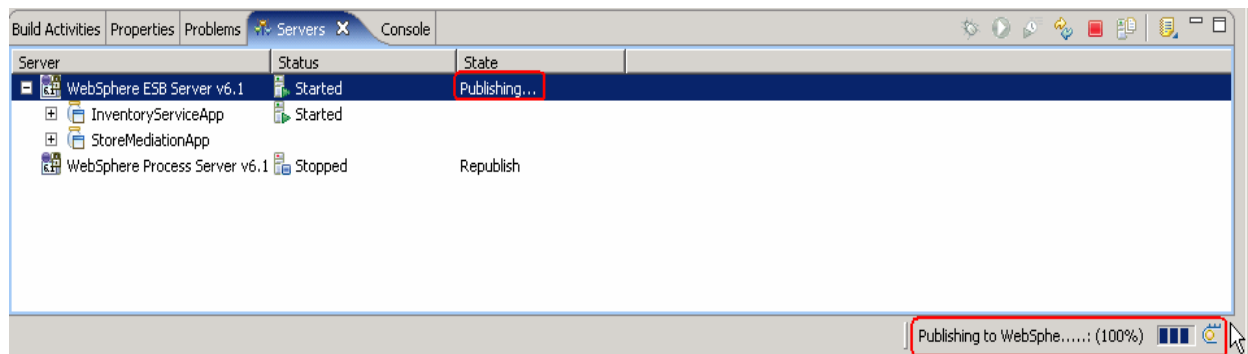
- 1) Right-click on **WebSphere ESB Server v6.1** under the Servers view and select **Add and remove projects...** from the context menu

- 2) In the Add and Remove Projects window, click **Add All >>** to add InventoryServiceApp and StoreMediationApp to the Configured projects panel

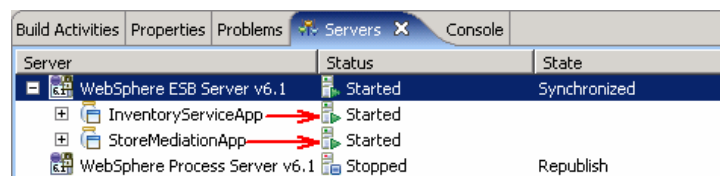


- 3) The projects will now be moved to Configured projects. Click **Finish**

- 4) Wait while the projects are being published to the server



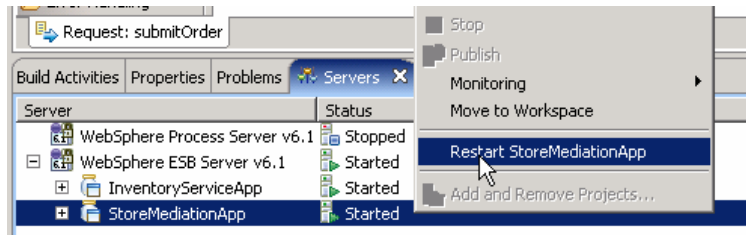
- 5) Once the publishing is done, from the Servers view, expand **WebSphere ESB Server v6.1** and you should see the 2 applications started as following:



\_\_\_ c. If the applications are **already deployed**, restart the StoreMediationApp following these steps:

- 1) Right click on StoreMediationApp to get the pop-up menu and select **Restart StoreMediationApp**

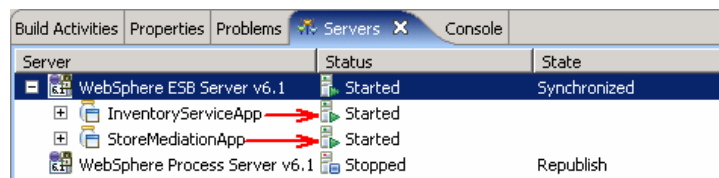




2) Wait while the application stops and restarts.



3) Once it has restarted, you should see the two applications started as shown here:



\_\_\_ 3. Check if the StoreMediation\_Test panel is still present from a previous lab.

- **No:** From the assembly diagram for StoreMediation, start Test Component for the StoreMediation component.
- **Yes:** Hit the Invoke icon (  ) in the Events panel
- -----

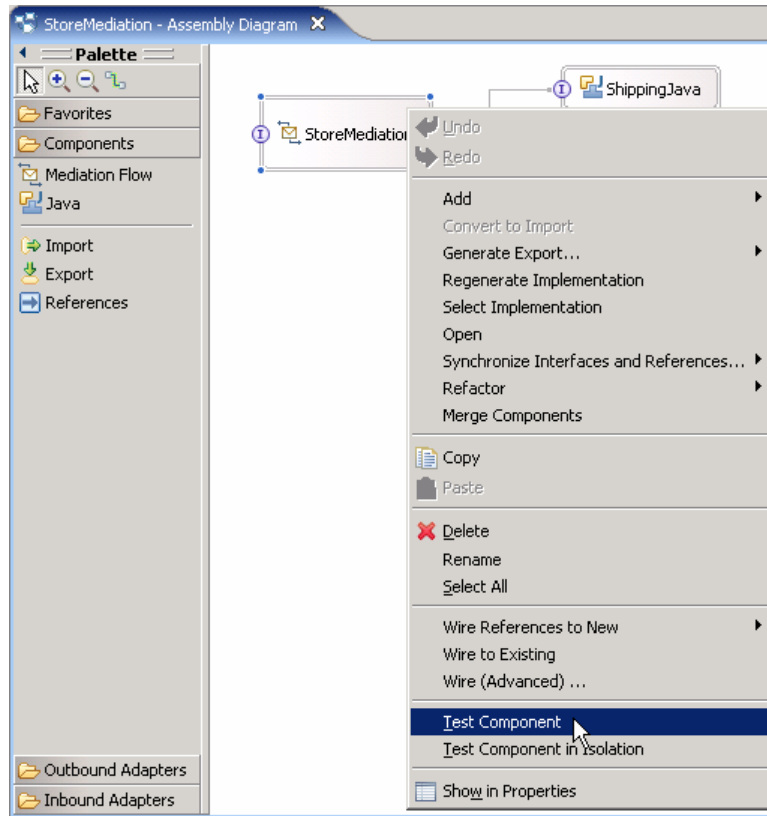
\_\_\_ a. Look at the tabs to see if StoreMediation\_Test is still open as shown in this screen capture.  
**Follow the appropriate instructions in step b or step c**




\_\_\_ b. **No, it is not open.** From the assembly diagram for StoreMediation, start Test Component for the StoreMediation component

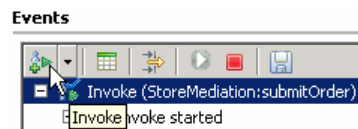
- 1) In the Business Integration window, expand **StoreMediation** and double-click on **Assembly Diagram** to open it in Assembly editor

- 2) From the StoreMediation-Assembly Diagram, right-click on **StoreMediation** component and select **Test Component** from the pop-up menu



- 3) The **StoreMediation\_Test** window is opened where you will enter your test data

\_\_\_ c. **Yes, it is open.** Click the Invoke icon (  ) in the Events panel.



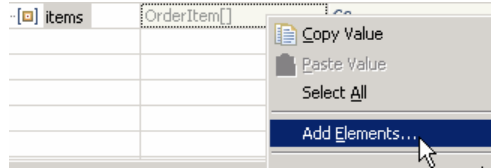
- \_\_\_ 4. Initialize the test data

- Set `customerID` to `cust123`
- Set `items/items[0]/itemID` to `item001`
- Set `items/items[0]/quantity` to `3`
- Set `items/items[1]/itemID` to `item009`
- Set `items/items[1]/quantity` to `5`
- Set `items/items[2]/itemID` to `item002`
- Set `items/items[2]/quantity` to `15`

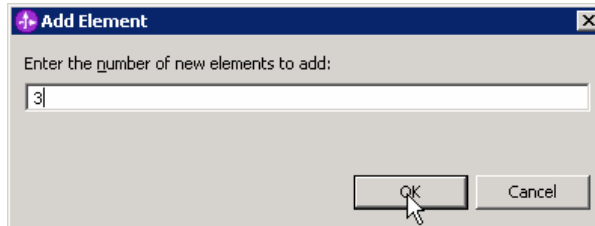
Name	Type	Value
order	Order	✓
customerID	string	✓ cust123
items	OrderItem[]	6
items[0]	OrderItem	✓
itemID	string	✓ item001
quantity	int	✓ 3
items[1]	OrderItem	✓
itemID	string	✓ item009
quantity	int	✓ 5
items[2]	OrderItem	✓
itemID	string	✓ item002
quantity	int	✓ 15

\_\_\_ a. Enter these values into the Initial request parameters table:

- 1) For **customerID**, click under Value and enter **cust123**
- 2) Right-click anywhere on the row containing **items** and select **Add Elements...** from the pop-up menu




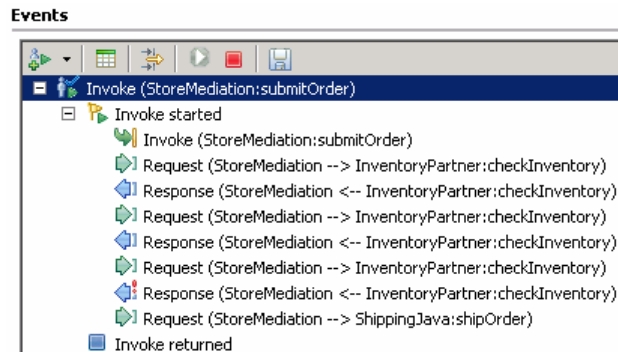
3) Enter '3' in the Add Element window:



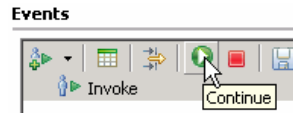
- 4) Click **OK**
- 5) Enter values for items[0]:
  - a) For **itemID**, click under Value and enter **item001**
  - b) For **quantity**, click under Value and enter **3**
- 6) Enter values for items[1]:
  - a) For **itemID**, click under Value and enter **item009**
  - b) For **quantity**, click under Value and enter **5**
- 7) Enter values for items[2]:
  - a) For **itemID**, click under Value and enter **item002**

b) For **quantity**, click under Value and enter **15**

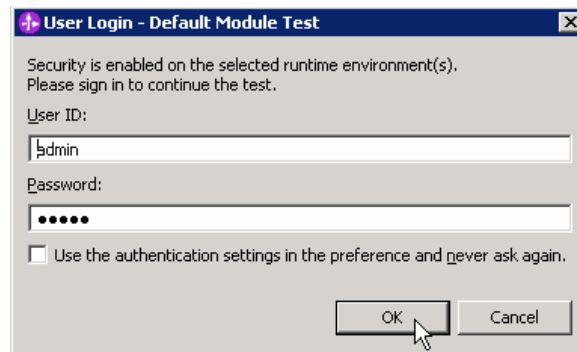
5. Run the test by hitting the Continue icon (  ). Results should show three calls to the inventory service, one for each item found on input.



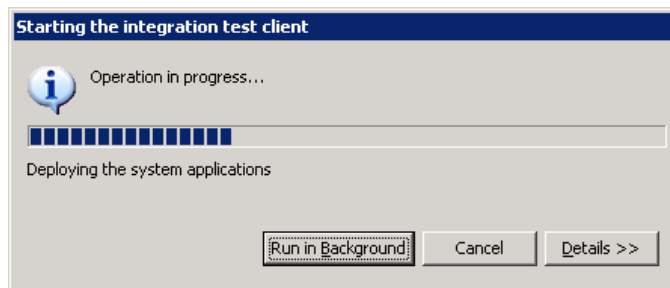
- a. Click **Continue** icon (  ) under Events panel



- b. If presented with the **User Login** dialog, enter the **User ID** and **Password**, which by default is normally set to **admin** and **admin**. Optionally, you can select the 'Use the authentication settings in the preference and never ask again' check box to prevent this dialog from being displayed in the future.



- c. Click **OK**
- d. Wait until the integration test client starts



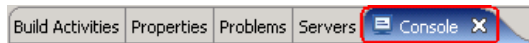
6. Switch to the Console view and examine the output, which should look similar to this screen capture. You can see the three invocations of the inventory service. There is one for each item, with one or two being successful and the other one or two resulting in a fault (which ones are successful and which ones fail will vary with each test run). Notice that the aggregated items sent to the shipping service indicate which items resulted in a fault and therefore do not have valid inventory information included. These are the items that flowed through the FaultRecovery XSL transformation. In this case it was the third item with itemID=item002, but your test run might show the fault occurring on different items.

```

O *****
O ***** START mediation flow *****
O ***
O ***** InvRandom - returning InventoryItem for itemID = item001
O ***** InvRandom - returning InventoryItem for itemID = item009
O ***** InvRandom - EXCEPTION: InventoryFault for itemID = item002
O ***
O ***** END mediation flow *****
O *****
O -----
O -- Ship object dump begins -----
O
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@d860d86
O Value:
O   customerID = cust123
O   items = ShipItem[3]
O     items[0] = <ShipItem@dbe0dbe>
O       itemID = item001
O       orderQuantity = 3
O       inventoryQuantity = 5
O       inventoryStatus = OK - but stock is running low
O     items[1] = <ShipItem@dd80dd8>
O       itemID = item009
O       orderQuantity = 5
O       inventoryQuantity = 45
O       inventoryStatus = OK - sufficient stock levels
O     items[2] = <ShipItem@df20df2>
O       itemID = item002
O       orderQuantity = 15
O       inventoryQuantity = 0
O       inventoryStatus = ERROR during inventory check, failure accessing inventory information for item = item002
O
O -- Ship object dump ends -----
O -----

```

- -----
- a. Double click on **Console** view next to Servers view to see the above message (by double clicking the Console view is maximized)



## Part 4: Authoring the mediation flow to use service invoke retry

**What you will do in this part:** In this part you will modify the service invoke mediation primitive to perform an automatic retry when the inventory service returns a modeled fault. By configuring the primitive to perform some number of retries, the fault flow used in the previous part of the lab exercise is not taken if there is a successful call to the inventory service on one of the retries.

See the presentation entitled [Augmentation, aggregation and retry tutorials](#) to better understand what this part is doing.

\_\_\_ 1. Modify the CheckInventory, service invoke primitive to make use of retry.

- **Retry on** : **Modeled fault**
- **Retry count**: **3**

**Service Invoke : CheckInventory**

Retry on: Modeled fault

Retry count: 3

Retry delay (seconds):

Try alternate endpoints

\_\_\_ a. Open the StoreMediation flow found in the StoreMediation module (if not opened already)

1) In the Business Integration view, expand **StoreMediation** → **Mediation Logic** → **Flows** and then double-click on **StoreMediation** to open it in the mediation flow editor

2) The line connecting **Ordering** → **submitOrder** and **ShippingPartner** → **shipOrder** should be selected for you to view the flow

\_\_\_ b. Select **CheckInventory** service invoke primitive and then select **Properties** → **Retry**

Build Activities Properties Problems Servers Console

Description

Terminal

Details

Retry

Promoted Properties

**Service Invoke : CheckInventory**

Retry on: Never

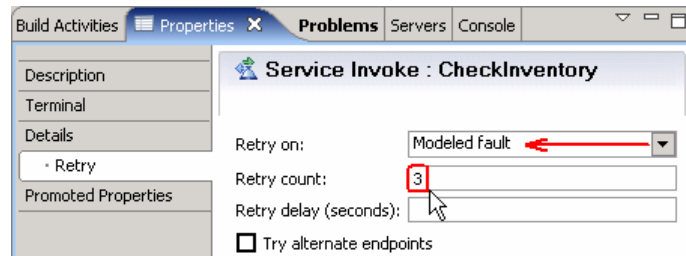
Retry count:

Retry delay (seconds):

Try alternate endpoints

\_\_\_ c. For 'Retry on', select **Modeled fault** from the drop down list

\_\_\_ d. For **Retry count**, enter **3**



\_\_\_ 2. Check that all the artifacts have been saved.

- -----

\_\_\_ a. From the menu select **File** → **Save All** to save your changes

\_\_\_ 3. Check that there are no errors reported in the Problems view.

- -----

\_\_\_ a. Select **Problems** view at the bottom. You can ignore warnings, but there should not be any errors at this time:

Description	Resource	Path	Location
<b>Warnings (14 items)</b>			
⚠ The InventoryPartner reference for the	StoreMediati...	StoreMediation	Unknown
⚠ The method getMyService() from the ty	InventoryFa...	InventoryService/inventory	line 23
⚠ The method getMyService() from the ty	InventoryR...	InventoryService/inventory	line 27
⚠ The method getMyService() from the ty	InventoryW...	InventoryService/inventory	line 23
⚠ The method getMyService() from the ty	ShippingJav...	StoreMediation/shipping	line 24
⚠ The serializable class _CSIServant_Stub	_CSIServan...	InventoryServiceEJB/ejbMo...	line 19
⚠ The serializable class _EJBHome_Stub d	_EJBHome_...	InventoryServiceEJB/ejbMo...	line 25
⚠ The serializable class _EJBObject_Stub	_EJBObject...	InventoryServiceEJB/ejbMo...	line 24
⚠ The serializable class _EJSWrapper_Stub	_EJSWrapp...	InventoryServiceEJB/ejbMo...	line 26

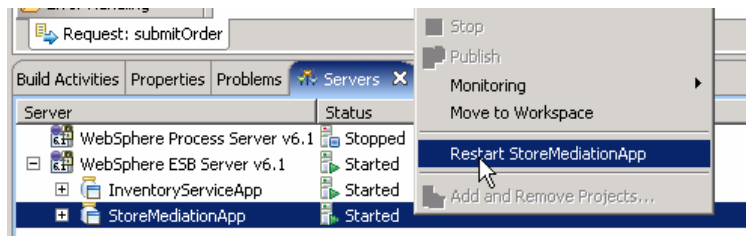
## Part 5: Test the service call retry mediation

**What you will do in this part:** In this part, you use the component test facilities of WebSphere Integration Developer to test the mediation. The resulting output is then explained. It should be noted that for this lab exercise the implementation of the inventory service is rigged to produce the required result for the lab. There will always be at least one use of service invoke to the inventory service that produces a modeled fault and the retry count is sufficient that all service invokes are eventually successful.

See the presentation entitled [Augmentation, aggregation and retry tutorials](#) to better understand what this part is doing.

### 1. Restart the StoreMediationApp in the test server

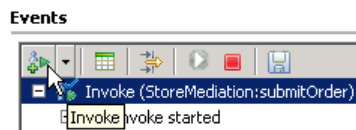
- -----
- \_\_\_ a. From the **Servers** view, expand **WebSphere ESB Server v6.1** to view the applications published onto this server
- \_\_\_ b. Right-click on **StoreMediationApp** and select **Restart StoreMediationApp** from the pop-up menu



- \_\_\_ c. Wait until the application Status shows as 'Started' again

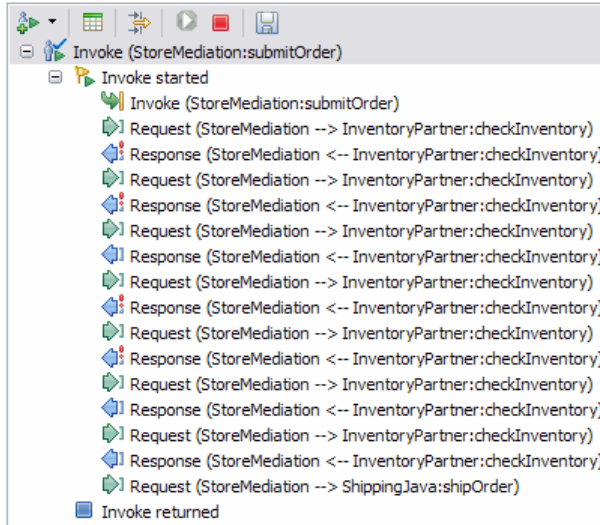
### 2. In StoreMediation\_Test, hit the Invoke icon (🚀) in the Events panel

- -----
- \_\_\_ a. Select tab for the StoreMediation\_Test panel
- \_\_\_ b. Click **Invoke** icon (🚀) under Events panel





- 3. Rerun the test using the same test data. Notice that there are more than three calls to the inventory service because failing calls were retried automatically within the CheckInventory primitive. In the case illustrated here there are seven calls. However, in your case there can be anywhere between four and nine calls (this is based on the algorithm used by the inventory service implementation to determine when to return successfully and when to create a fault)



- a. Repeat the instructions in **Part 3, Step 4** and provide the same values

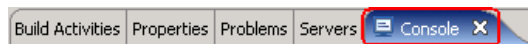
4. Switch to the Console view and examine the output which should be similar to this screen capture. In the specific case shown in this screen capture, the original calls for item001 and item009 each failed and each required two retries to succeed. The initial call for item002 was successful. Notice that the aggregated items sent to the shipping service all indicate success and the number corresponds to the number of input items, not to the number of calls to the inventory service. From this you can see that the fault flow, which you defined and tested in the previous sections, was not taken. It is only in the event that all retries fail that the fault flow is taken.

```

O -----
O *****
O ***** START mediation flow *****
O ***
O ***** InvRandom - EXCEPTION: InventoryFault for itemID = item001
O ***** InvRandom - EXCEPTION: InventoryFault for itemID = item001
O ***** InvRandom - returning InventoryItem for itemID = item001
O ***** InvRandom - EXCEPTION: InventoryFault for itemID = item009
O ***** InvRandom - EXCEPTION: InventoryFault for itemID = item009
O ***** InvRandom - returning InventoryItem for itemID = item009
O ***** InvRandom - returning InventoryItem for itemID = item002
O ***
O ***** END mediation flow *****
O -----
O -- Ship object dump begins -----
O
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@7cd27cd2
O Value:
O   customerID = cust123
O   items = ShipItem[3]
O     items[0] = <ShipItem@7d0a7d0a>
O       itemID = item001
O       orderQuantity = 3
O       inventoryQuantity = 5
O       inventoryStatus = OK - but stock is running low
O     items[1] = <ShipItem@7d247d24>
O       itemID = item009
O       orderQuantity = 5
O       inventoryQuantity = 45
O       inventoryStatus = OK - sufficient stock levels
O     items[2] = <ShipItem@7d3e7d3e>
O       itemID = item002
O       orderQuantity = 15
O       inventoryQuantity = 10
O       inventoryStatus = Backorder - insufficient stock to fill order
O
O -- Ship object dump ends -----
O -----

```

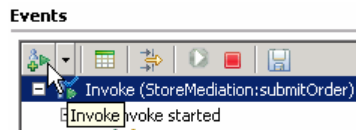
- a. Double click on **Console** view next to Servers view to see the above message (by double clicking the Console view is maximized)



\_\_\_ 5. If you want to, you can run additional tests using different input data to see how the output varies. Key things to note about the data you use:

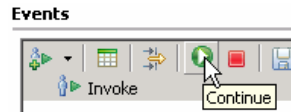
- **customerID** can be any string and should not have any particular affect on the results
- The **items** array can have any number of elements
- **itemID** values of "item001" through "item010" are recognized by the inventory service, all other values are not recognized
- Inventory status will change according to the relationship between the order and inventory quantities
- -----

\_\_\_ a. Click **Invoke** icon (  ) under Events panel



\_\_\_ b. Enter values for **customerID**, **itemID**, and **quantity** as per the above instructions

\_\_\_ c. Click **Continue** icon (  ) under Events panel



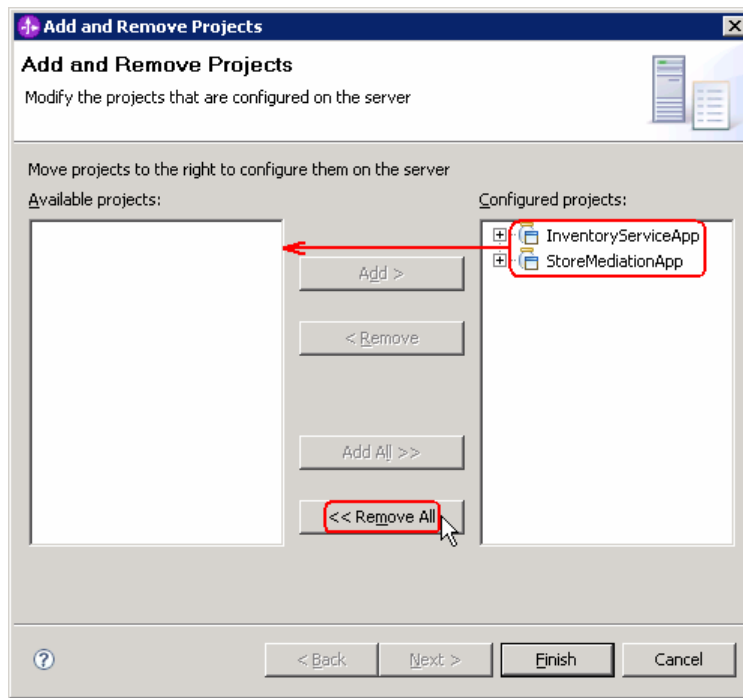
## Part 6: Clean up the environment if you will not proceed to the next lab

**Perform this part only if you are not continuing** to the retry alternate endpoints lab (the fourth lab in this series of labs).

\_\_\_ 1. Remove the InventoryServiceApp and StoreMediationApp from the test server.

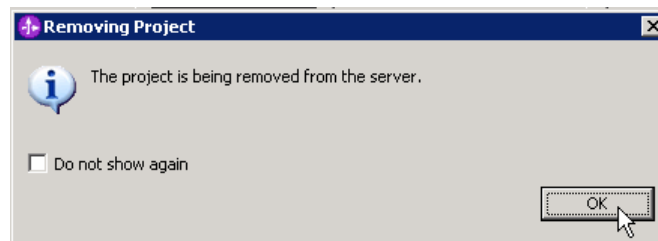
\_\_\_ a. Right-click on **WebSphere ESB Server v6.1** under the Servers view and select **Add and remove projects...** from the context menu

\_\_\_ b. From the Add and Remove Projects window, click **<< Remove All**



\_\_\_ c. Click **Finish** after you see the applications moved to Available projects.

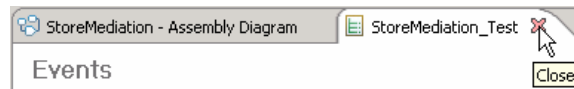
\_\_\_ d. If displayed, click **OK** on 'Removing Project' window. Optionally, you can select the check box for 'Do not show again' not to be asked again when you remove projects later



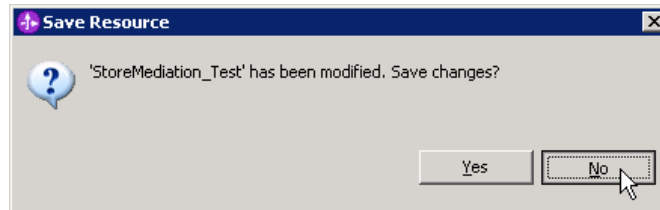
\_\_\_ e. Wait until the application is removed from the server

\_\_\_ 2. Close the StoreMediation\_test panel without saving

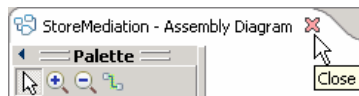
\_\_\_ a. Click **X** on the StoreMediation\_Test tab




\_\_\_ b. Click **No** from Save Resource window

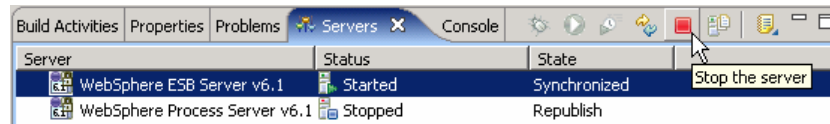


\_\_\_ c. Click **X** on the StoreMediation – Assembly Diagram tab

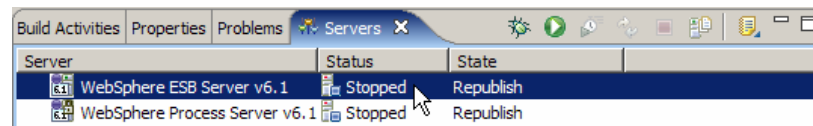


\_\_\_ 3. Stop the test server

\_\_\_ a. From the **Servers** view of WebSphere Integration Developer, select **WebSphere ESB Server v6.1** and hit '**Stop the server**' icon (  ) from the toolbar



\_\_\_ b. Wait until the server Status shows as **Stopped**



\_\_\_ 4. Exit from WebSphere Integration Developer.

\_\_\_ a. From menu, select **File** → **Exit** or Click '**X**' at the right top corner of your WebSphere Integration Developer window

## What you did in this exercise

In this exercise, you modified an augmentation and aggregation message flow to recover from faults that might be thrown by the inventory service. There were two approaches taken. The first was to recover within the flow by handling a fault returned through a terminal on the service invoke primitive. The second was to enable the service invoke primitive to perform automatic try.

Reviewing the presentation entitled [Augmentation, aggregation and retry tutorials](#) will help you better understand what was done in the lab.

## Solution instructions

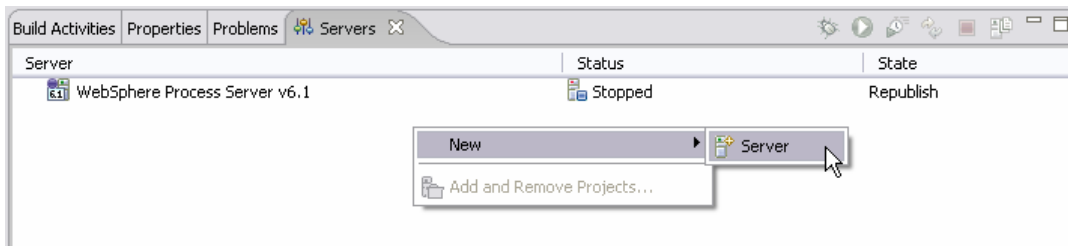
If you want to run this lab with a completed solution rather than authoring the flow yourself, follow these instructions:

- \_\_\_\_ 1. Follow **Part 1: Setting up the environment for the lab**, however use the project interchange file that contains the solution.
  - `<LAB_FILES>/PI4-RetrySolution-AlternateEndpointsStart.zip`
- \_\_\_\_ 2. Skip to **Part 5: Test the service call retry mediation** and proceed through the rest of the lab.

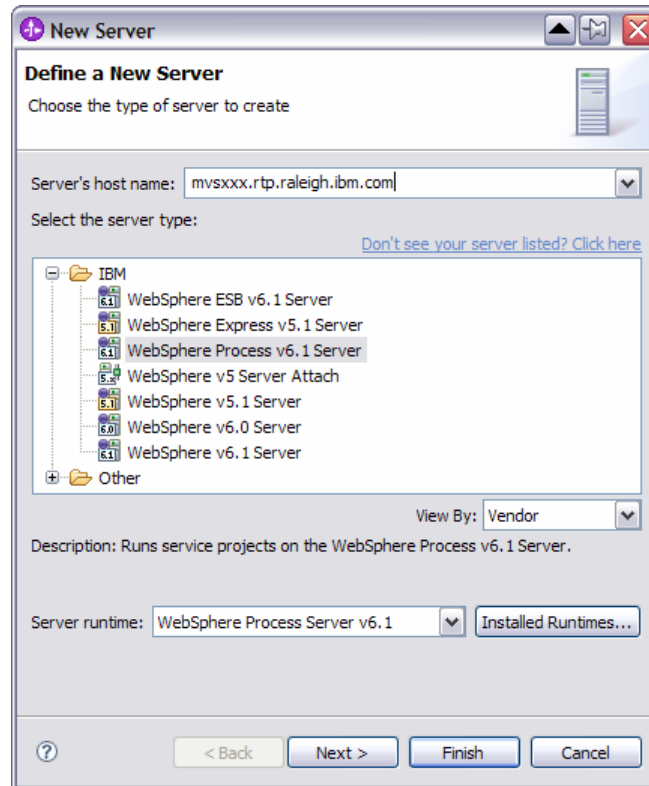
## Task: Adding remote server to the WebSphere Integration Developer test environment

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

- \_\_\_ 1. Define a new remote server to WebSphere Integration Developer.
  - \_\_\_ a. Right click on the background of the Servers view to access the pop-up menu.
  - \_\_\_ b. Select New → Server.



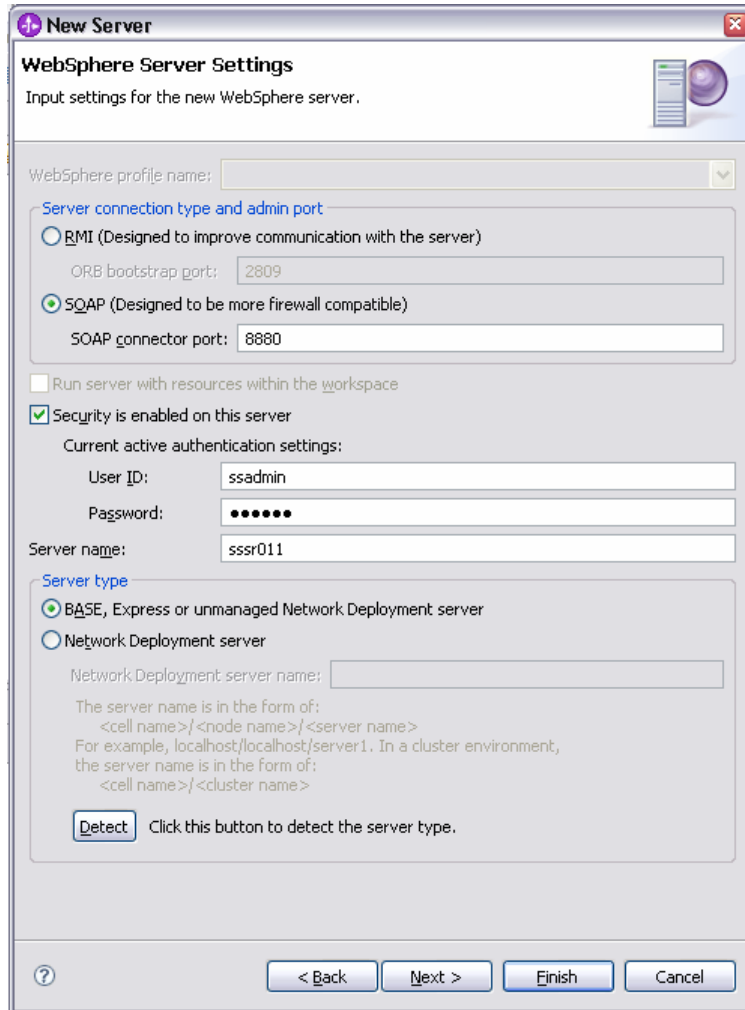
- \_\_\_ c. In the New Server dialog, specify the remote server's host name, <HOSTNAME>.
- \_\_\_ d. Ensure that the appropriate server type, 'WebSphere Process v6.1 Server' or 'WebSphere ESB v6.1 Server', is highlighted in the server type list



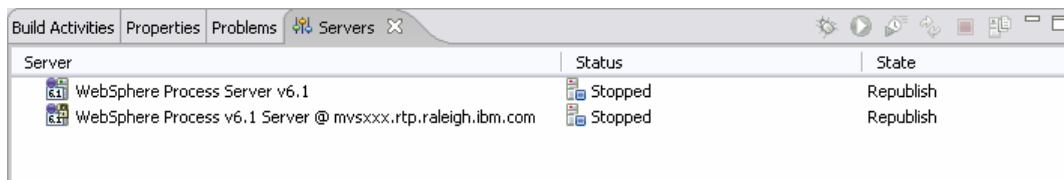
- \_\_\_ e. Click **Next**.



- \_\_\_ f. 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.



- \_\_\_ g. Click **Finish**.
- \_\_\_ h. The new server should be seen in the Server view.



- \_\_\_ 2. 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>'**

User ID : **<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 sssr01 open for e-business; process id is 0000012000000002
```