

IBM WEBSHERE 7.0 – LAB EXERCISE

WebSphere Enterprise Service Bus 7.0

Augmentation, aggregation, and retry tutorial series

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.....	8
Part 3: Test the recover from modeled fault mediation.....	18
Part 4: Authoring the mediation flow to use service invoke retry.....	27
Part 5: Test the service call retry mediation.....	29
Part 6: Clean up the environment if you will not proceed to the next lab.....	34
What you did in this exercise	36
Solution instructions.....	37

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 V7.0 installed on Windows or Linux
- WebSphere Enterprise Service Bus V7.0 or WebSphere Process Server V7.0. The server can either be the test server installed by WebSphere Integration Developer or a remote server from a separate WebSphere Enterprise Service Bus V7.0 or WebSphere Process Server V7.0 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 in a series of four labs intended to illustrate elements of the mediation programming model, addressing these capabilities:

- Using a service invoke primitive in a flow
- Augmenting message content with the results of a service invoke
- Using message splitting and aggregation for message augmentation of repeating elements
- Service call retry of failing service invocations
- Using alternate endpoints for service call retry

The four labs are described in the presentation entitled [Augmentation, aggregation and retry labs](#). You should familiarize yourself with the labs 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\WID70	/opt/IBM/WID70
<ESB_PROFILE_HOME>	<WID_HOME>_WTE\runtimes\bi_v7\profiles\qesb	<WID_HOME>_WTE\runtimes\bi_v7\profiles\qesb
<LAB_FILES>	C:\Labfiles70\WESB\AugAggRetry	/tmp/Labfiles70\WESB\AugAggRetry

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 previous paragraph, should allow the experienced user to complete the task
 - The following line of dashes begins the step by step instructions needed by the new user to complete this task. The experience user can skip to the next 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 Two.

___ 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.

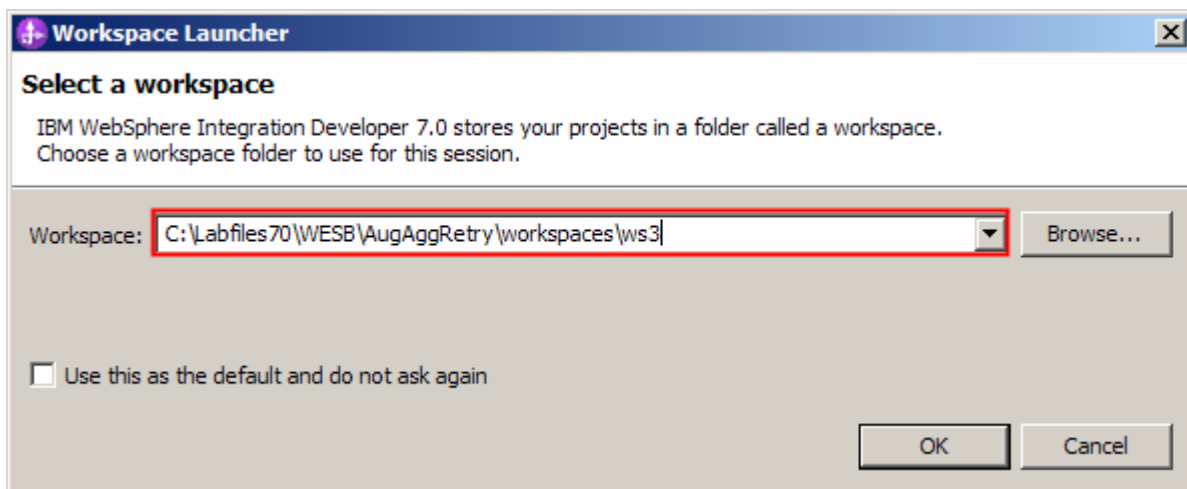
- **Restarting from Lab Two:** Use the same workspace used in Lab Two
- **Directly starting Lab Three: Suggested location:**
<LAB_FILES>/workspaces/ws3
- -----

___ a. Select **Start** → **All Programs** → **IBM WebSphere Integration Developer** → **IBM WebSphere Integration Developer V7.0** → **WebSphere Integration Developer V7.0**

___ b. From the Workspace Launcher window, enter the name of the workspace in the Workspace field

1) Restarting from Lab Two: Use the same workspace used in Lab Two

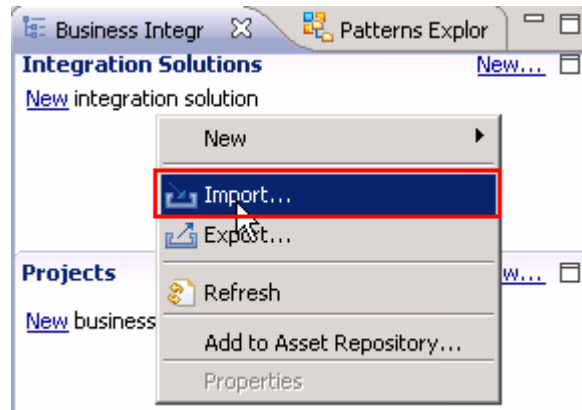
2) Directly starting Lab Three: <LAB_FILES>/workspaces/ws3



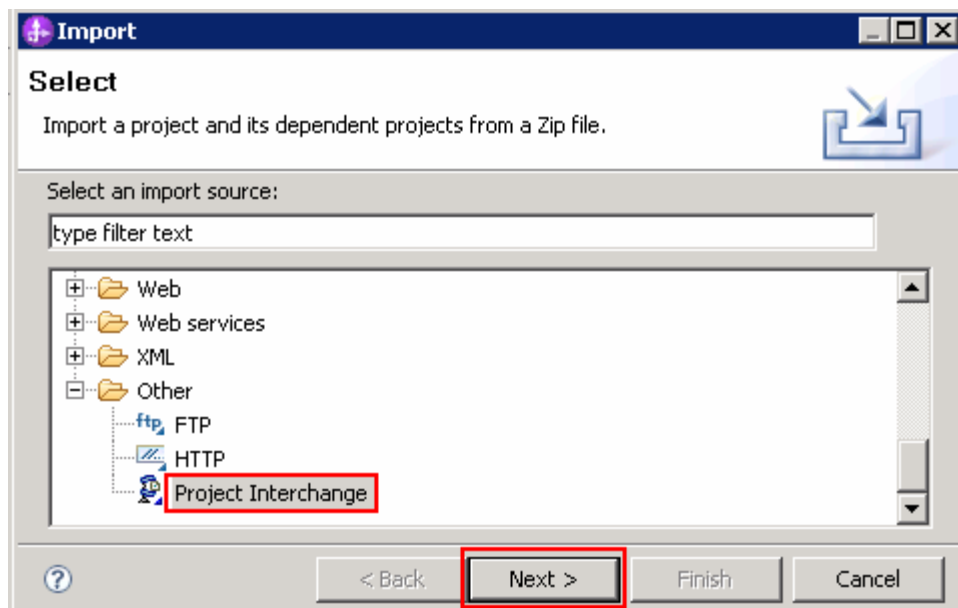
- ___ 3. If you are **restarting from Lab Two** there is nothing additional to do, skip directly to **Part 2 Authoring the mediation flow to recover from a modeled fault**
 - -----

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

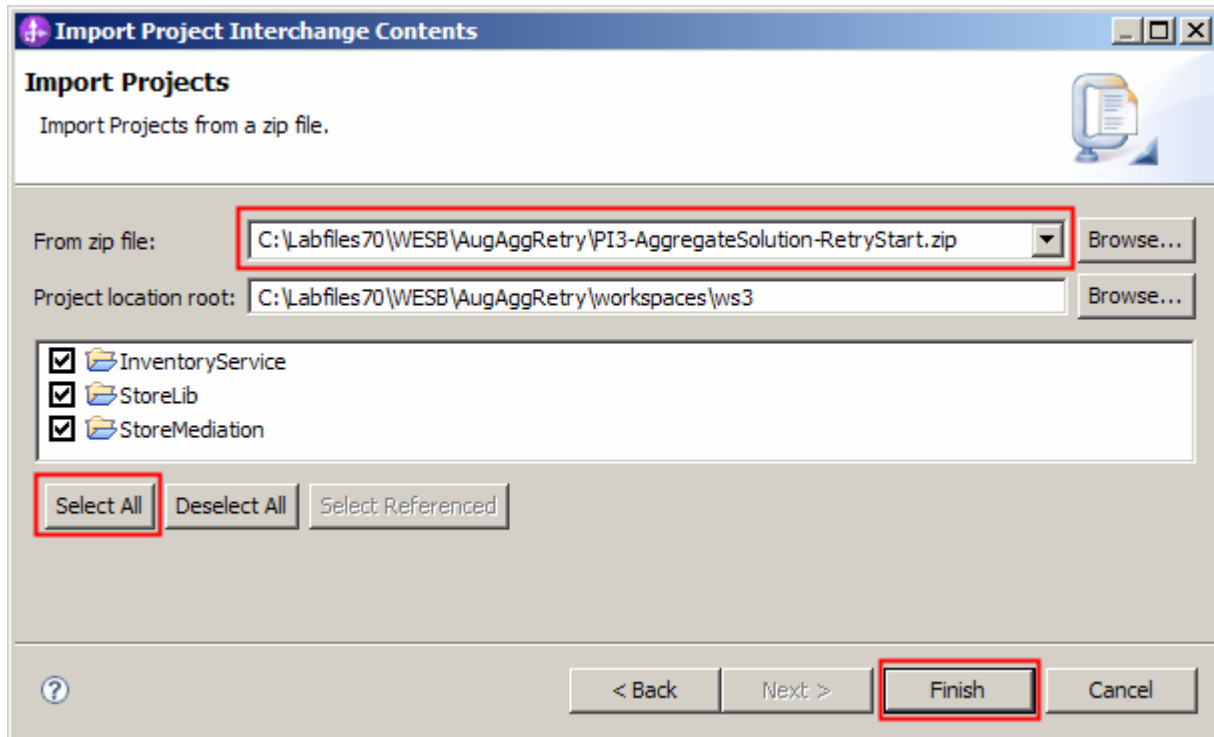


- ___ b. In the **Import** dialog, expand **Other** and select **Project Interchange** from the list

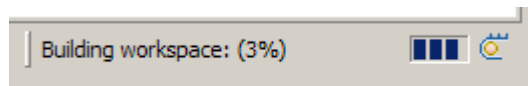


- ___ c. Click **Next**

- ___ d. In the next panel, click the **Browse** button for **From zip file** and navigate to **<LAB_FILES>/PI3-AggregateSolution-RetryStart.zip** and click **Open**



- ___ e. Click **Select All** to select all the projects listed and then click **Finish**
- ___ f. Wait for the build process to complete for the imported projects, which is seen at the lower right corner of WebSphere Integration Developer.

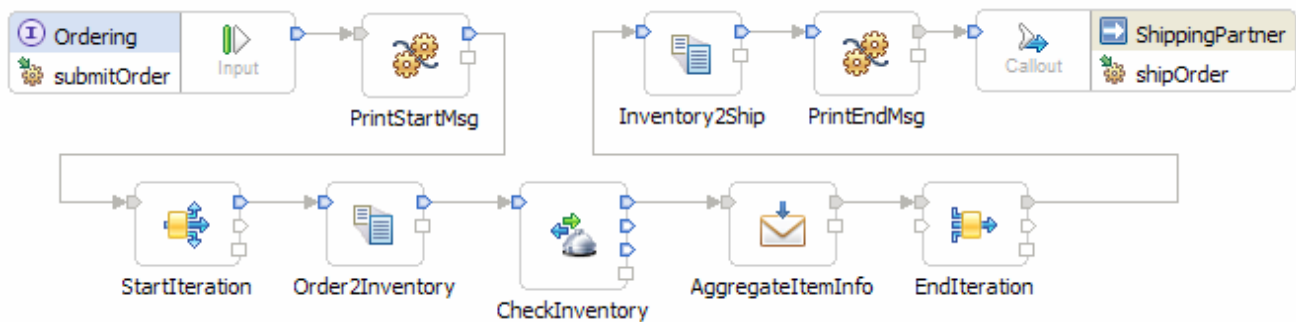


Part 2: Authoring the mediation flow to recover from a modeled fault

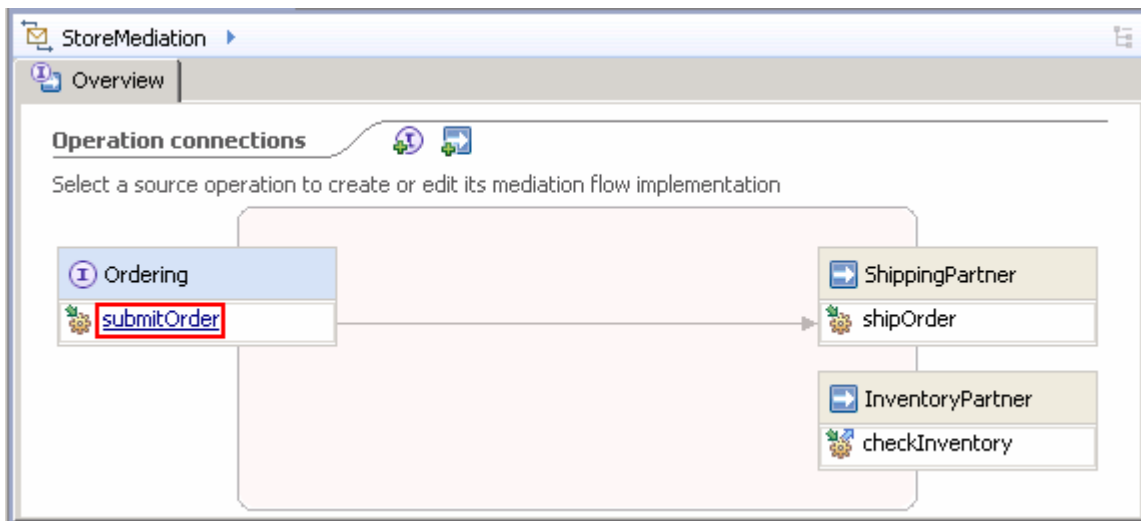
What you will do in this part: In this part, you will modify the flow to handle the situation when a modeled fault is returned by the inventory service. The requirement is for the modeled fault not to cause the entire flow to fail. The item which resulted in a modeled fault during the call to the inventory 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.

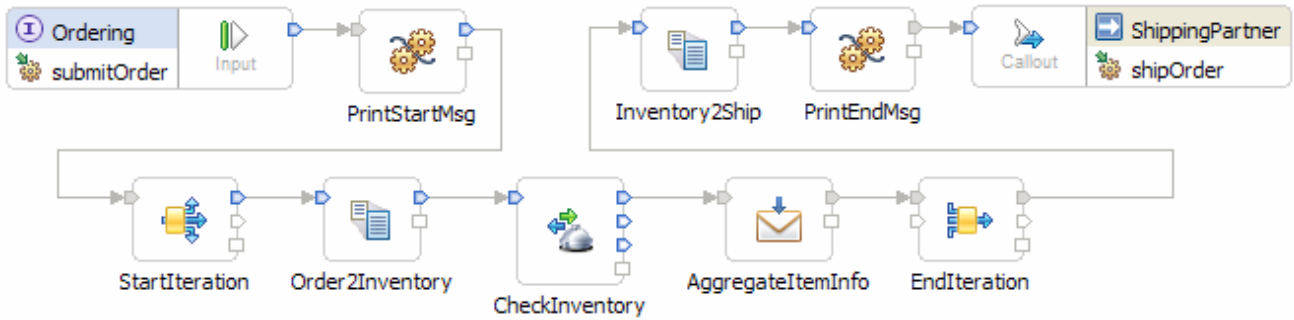
1. Open the StoreMediation flow found in the StoreMediation module.



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

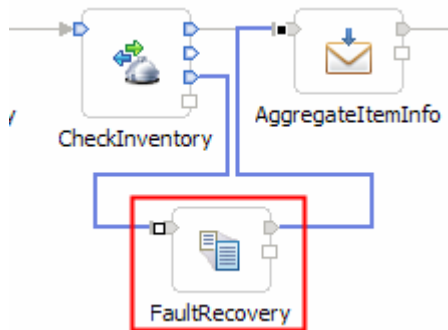


2. In the mediation flow editor, click the source operation, **submitOrder**, as shown in the picture above. This action opens the Request flow as shown here:



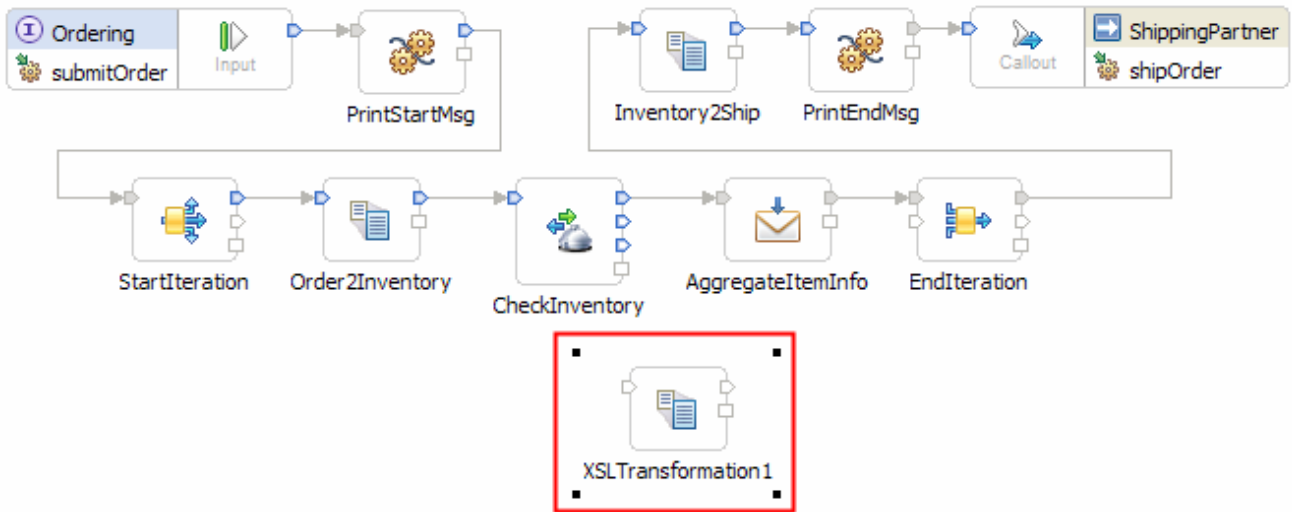
3. Add an XSL transformation primitive to the canvas and wire it into the flow. This primitive provides the required transformations to convert a fault returned by the inventory service so that the item can still be processed rather than allowing the flow to fail. To do that, the message body for the fault must be converted to look like the message body returned by a normal inventory service call. However, because of the fault, the inventory quantity is set to zero and the inventory status carries information describing the fault.

- **Display Name: FaultRecovery**
- **Wire : CheckInventory(checkInventory_InventoryFaultMsg**
- **Z terminal) to FaultRecovery**
- **Wire : FaultRecovery to AggregateItemInfo**
-

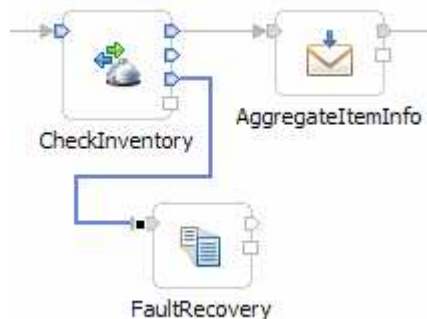


-
- -----

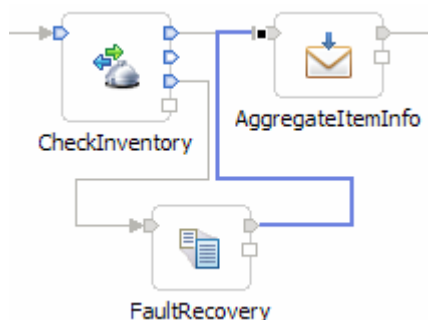
- ___ a. From the **Palette**, select **Transformation → XSL Transformation** and then click the canvas to add the primitive (between **CheckInventory** and **AggregateItemInfo** primitives as shown below).



- ___ b. Ensure that the XSL transformation primitive is selected and navigate to the **Description** panel of the **Properties** view and change the **Display Name** to **FaultRecovery**.
- ___ c. Add a connection from **CheckInventory** (the **checkInventory_InventoryFaultMsg** terminal) to the **FaultRecovery** primitive. Click the **checkInventory_InventoryFaultMsg** terminal and drag it to the **in** terminal of **FaultRecovery**.



- ___ d. Similarly, add a connection from the **FaultRecovery** primitive to the **AggregateItemInfo** primitive. Click the **out** terminal of **FaultRecovery** and drag to the **in** terminal of **AggregateItemInfo**:



___ e. From the menu select **File → Save** (or **Ctrl + S** from your keyboard) to save your changes to the mediation flow.

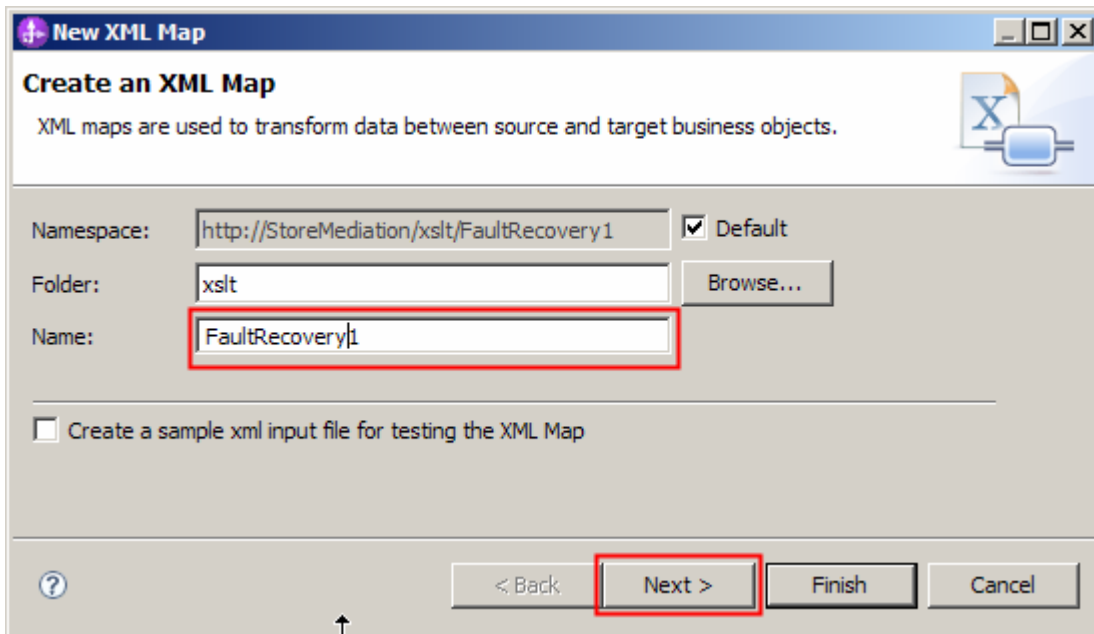
___ 4. In the FaultRecovery XML transformation primitive, create a new XML map file used to define the XSL transformation.

- **Map Name** : **FaultRecovery1**
- **Message Root** : **/body**
- **Input Message Body** : **checkInventory_InventoryFaultMsg**
- **Output Message Body**: **checkInventoryResponseMsg**
- -----
-

___ a. Select the **FaultRecovery** primitive on the canvas and then navigate to the **Details** panel of the **Properties** view.

___ b. Click **New...** next to '**Mapping file**'. The New XML Map wizard is opened.

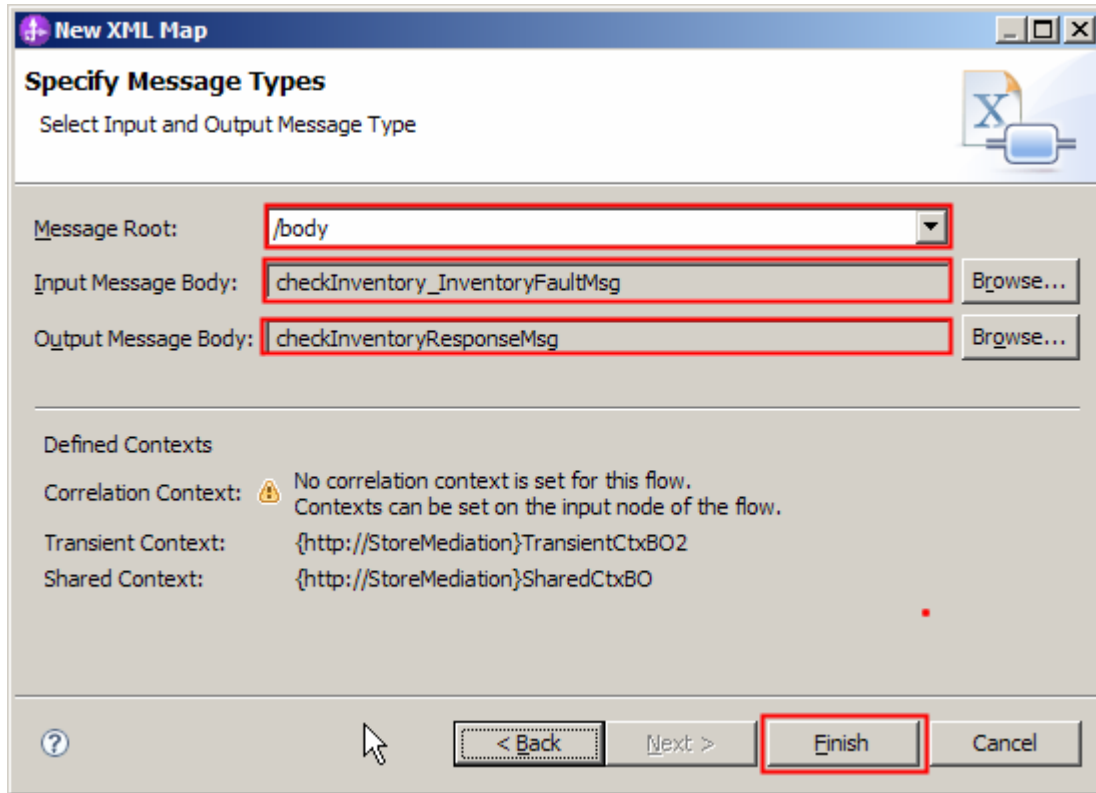
___ c. For **Name** enter **FaultRecovery1**



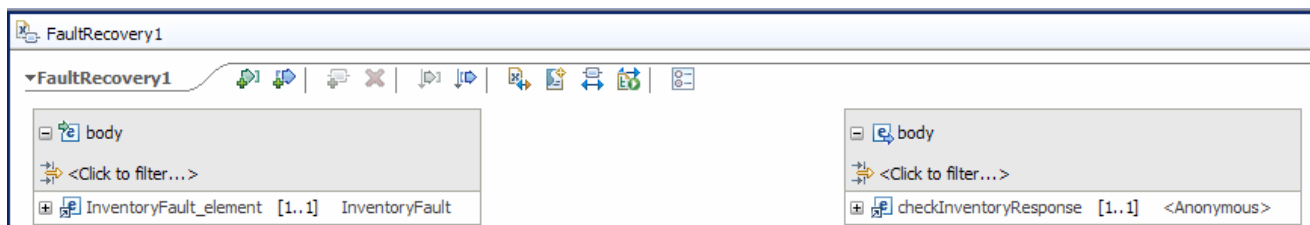
___ d. Click **Next**

__ e. From Specify Message Types panel, enter:

- 1) For **Message Root**, accept the default selection of **/body**
- 2) For **Input Message Body**, accept the default selection: **checkInventory_InventoryFaultMsg**
- 3) For **Output Message Body**, accept the default selection: **checkInventoryResponseMsg**

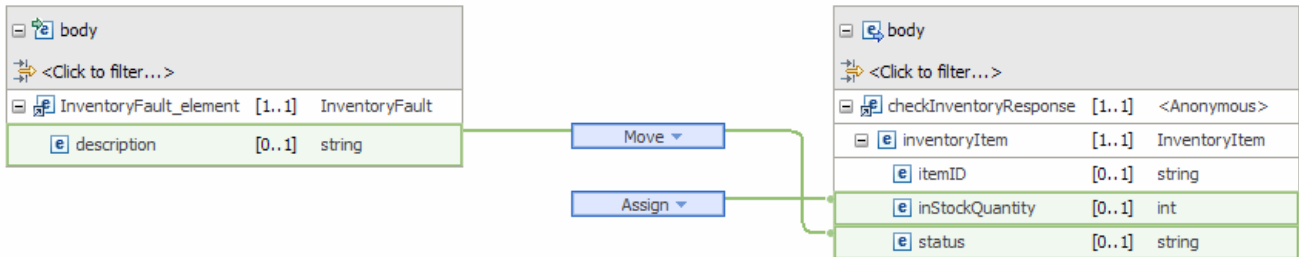


__ f. Click **Finish**. The FaultRecovery1.map file is opened in the XML Mapping editor.



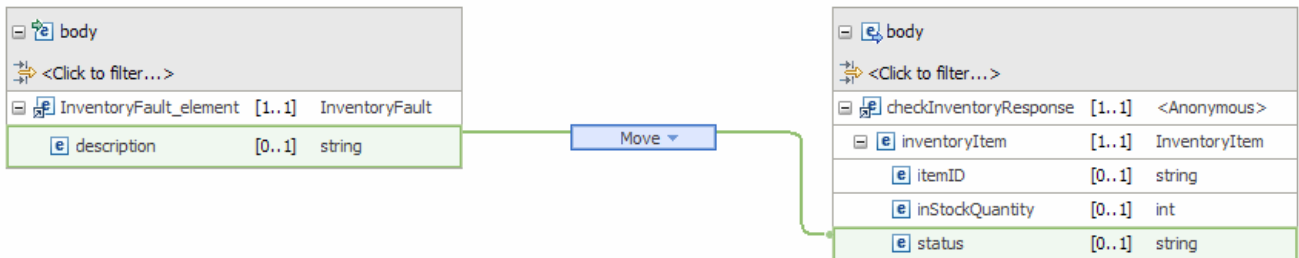
5. Define the mapping for the FaultRecovery1 map. The mapping must address making the body of the message appear as if it was returned by a normal response of a call to the inventory service. There are three fields in a normal response which will be initialized as (1) itemID – not initialized because it is not used downstream in the flow, (2) inStockQuantity – set to zero because this value is unknown due to the failed call and (3) status – set to the error description returned by the fault

- **Move InventoryFault_element/description to checkInventoryResponse/inventoryItem/status**
- **Use an Assign transform to set checkInventoryResponse/inventoryItem/inStockQuantity to 0**



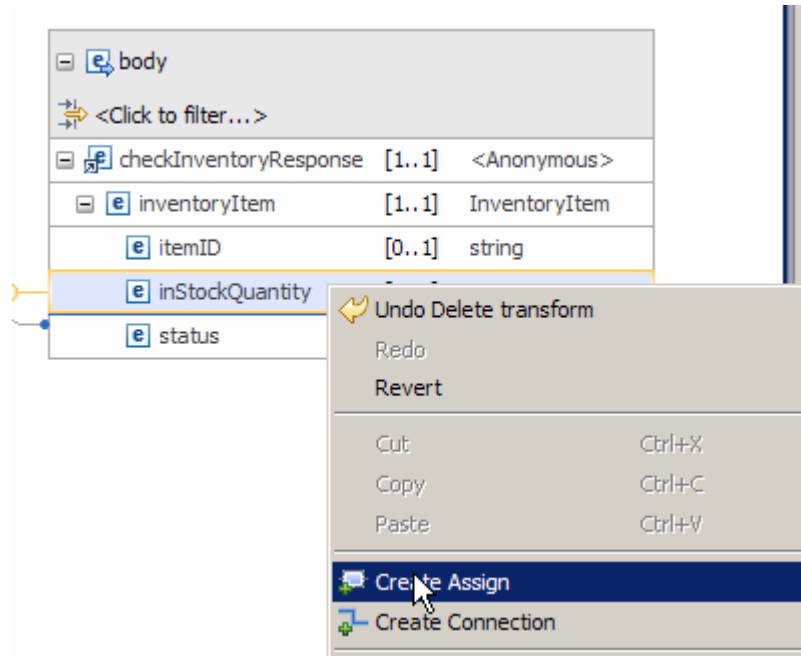
a. Move InventoryFault_element/description to checkInventoryResponse/inventoryItem/status

- 1) In the source body, expand **InventoryFault_element**
- 2) In the target body, expand **checkInventoryResponse** → **inventoryItem**
- 3) Create a **Move** transformation by clicking on **description** in the source body and dragging it to **status** in the target body

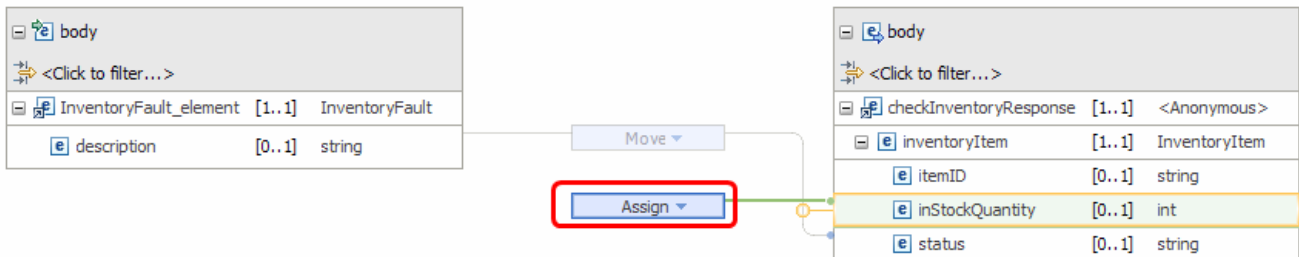


__ b. Assign the value 0 to checkInventoryResponse/inventoryItem/inStockQuantity

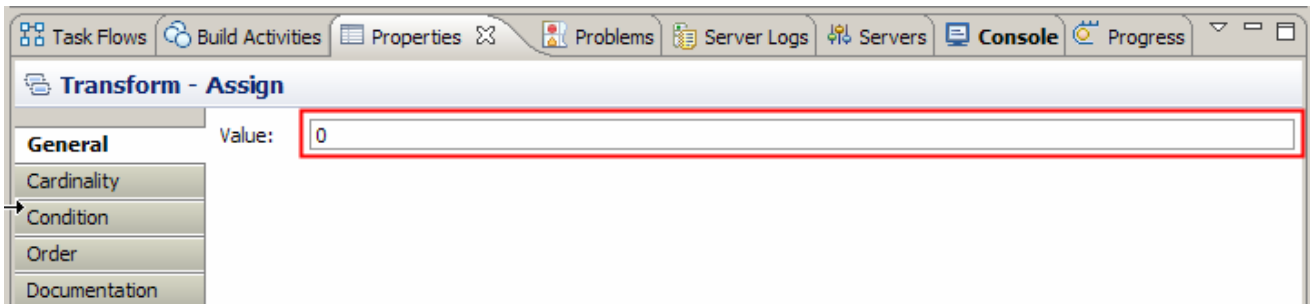
1) Right-click **inStockQuantity** and select **Create Assign** from the pop-up menu



The result is a new Assign transform being added to the map.



2) Ensure the new **Assign** transform is selected and navigate to the **General** panel of the **Properties** view. Set the **Value** property to **0** if not already set.



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

__ d. Close the FaultRecovery1.map file

6. Modify the InventoryPartner import in the StoreMediation assembly to call a different implementation of the InventoryService. This implementation sometimes works and sometimes return a modeled fault. Using this implementation allows the testing to illustrate the results when CheckInventory is successful and when it returns a fault.

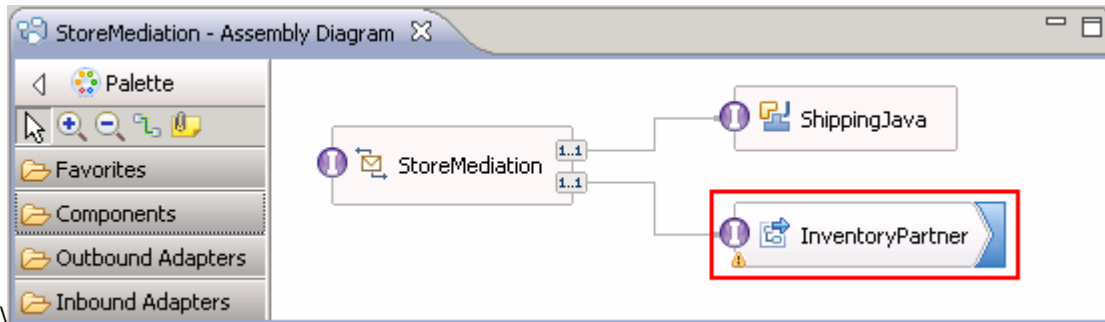
- **InventoryPartner import, Export name: InvRandom**

Module name:

Export name:

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

- b. Select the **InventoryPartner** import in the assembly diagram.



- c. Navigate to the **Binding** panel of the **Properties** view. Note that the InventoryService is using an Export names **InvWorks**.

Import: InventoryPartner (SCA Binding)

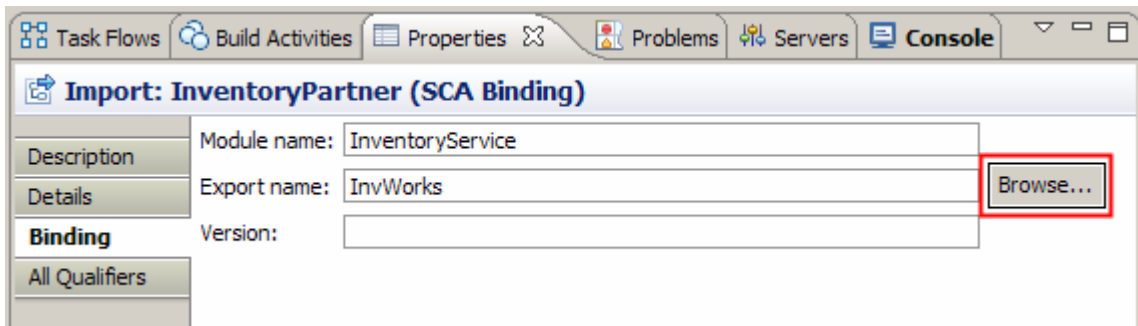
Description: Module name:

Details: Export name:

Binding: Version:

All Qualifiers

___ d. Click **Browse...** for **Export name**.

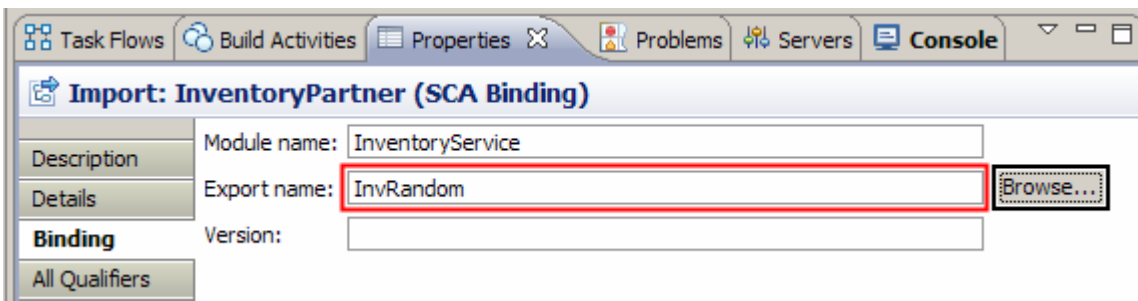


___ e. In the SCA Export Selection dialog, select **InvRandom** under Matches



___ f. Click **OK**

___ g. The Export name is now populated with **InvRandom**: This is the implementation of the InventoryService that randomly works or returns a fault.



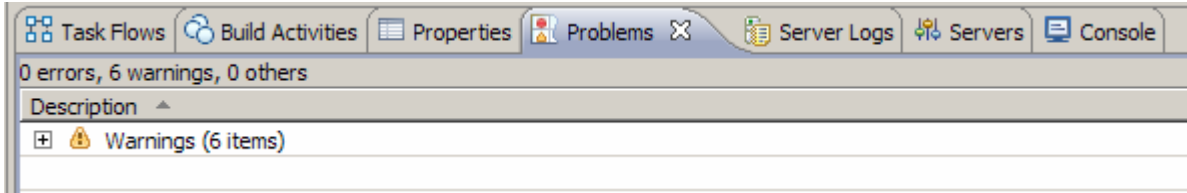
___ 7. Check that all the artifacts have been saved.

- -----
- ___ 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:




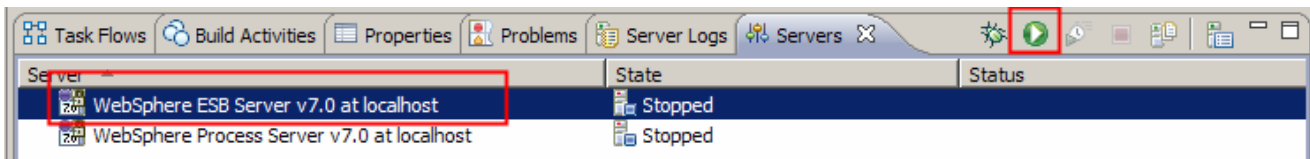
Part 3: Test the recover from modeled fault mediation

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 v7.0** and hit **Start the server** icon () from the toolbar



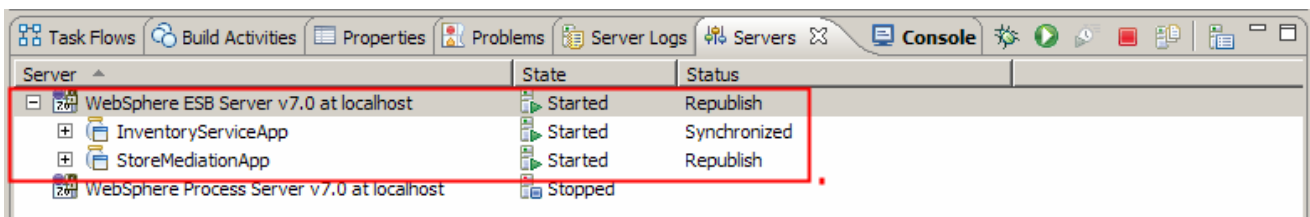
___ b. Wait until the server Status shows as **Started**

NOTE: Depending upon the preferences you have specified for the **Console** or **Server Logs** view, they might grab the focus from the **Servers** view. If this is the case, the status in the lower right should indicate when the server start 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:** Republish to ensure latest versions are loaded

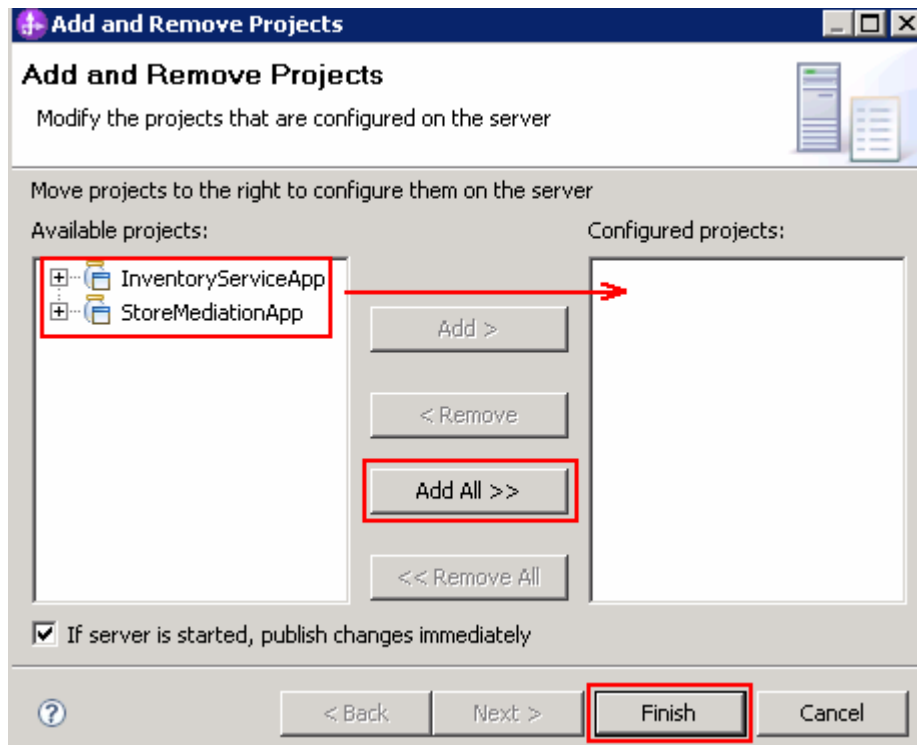
___ a. Check the Servers view to see if the InventoryServiceApp and StoreMediationApp are already deployed. If they are, they will appear below the server as shown here. **Follow the appropriate instructions in step b or step c**



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

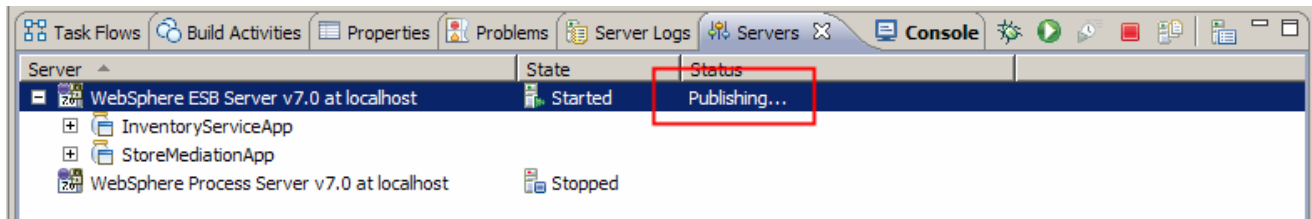
- 1) Right-click **WebSphere ESB Server v7.0** under the Servers view and select **Add and remove projects...** from the pop-up menu

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

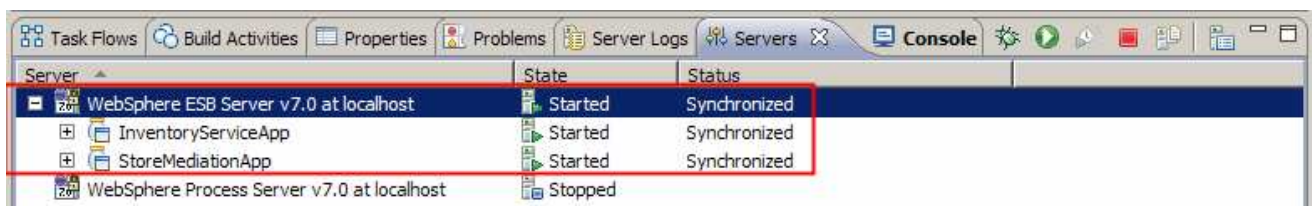


- 3) The projects will now be moved to Configured projects. Click **Finish**

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

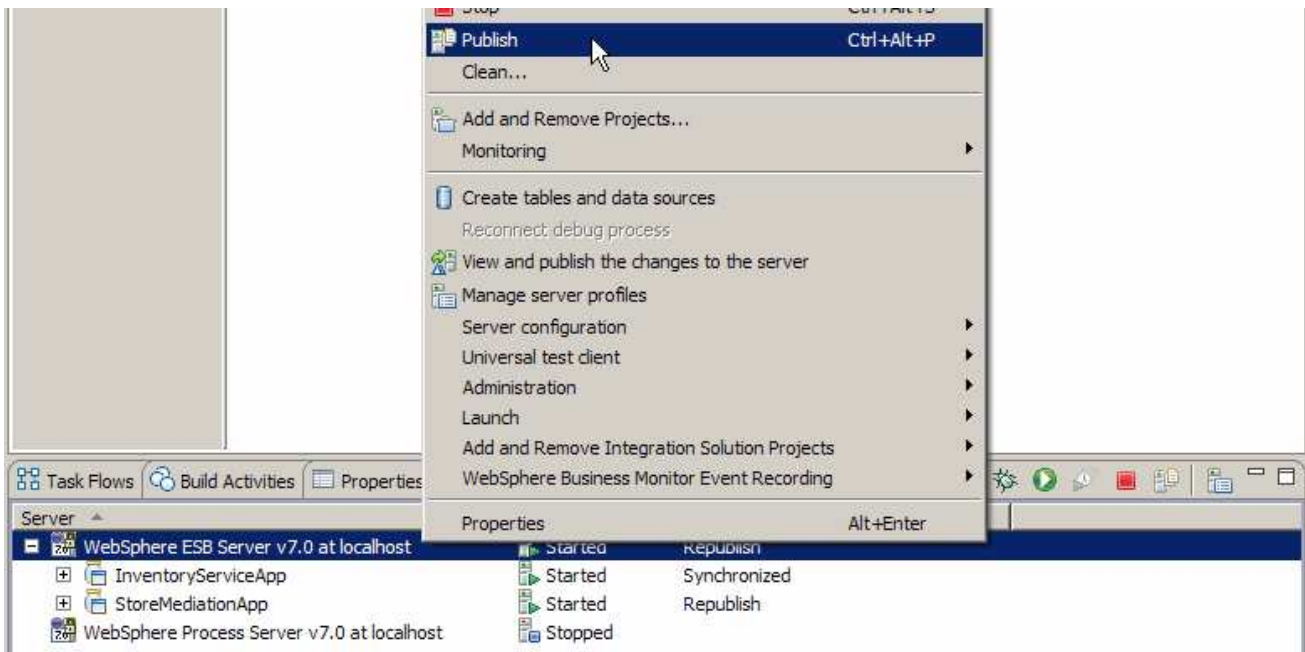


- 5) Once the publishing is done you should see the two applications started as shown here:

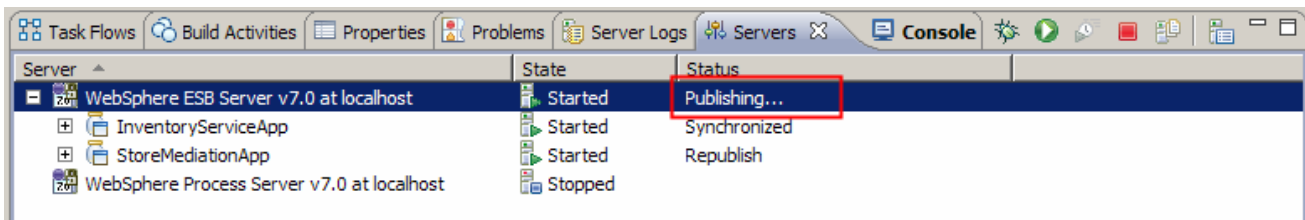


___ c. If the applications are **already deployed**, a new publish needs to be done to load them into the server with the latest changes.

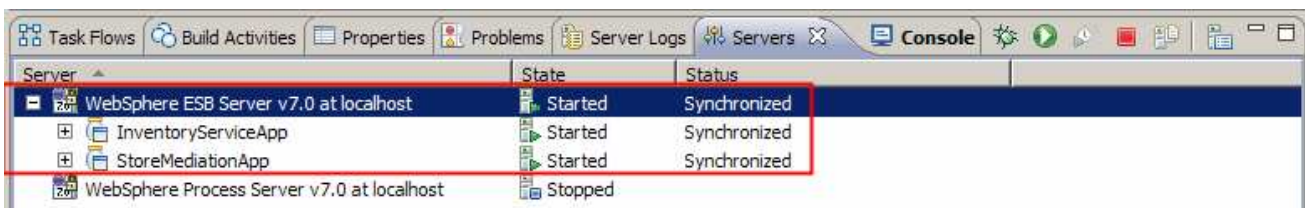
1) Right click the server and select **Publish** from the pop-up menu



2) Wait while the applications are publishing to the server



3) Once the publishing is done you should see the two applications started as shown here:



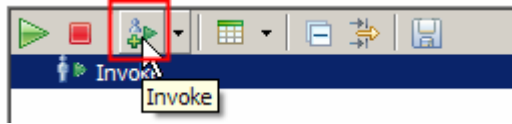
___ 3. Check if the StoreMediation_Test panel is still present from a previous lab.

- **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 either step b or step c



- ___ b. **Yes, it is open.** Click the Invoke icon () in the Events panel



- ___ c. **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 **Assembly Diagram** to open it in Assembly editor
- 2) From the StoreMediation-Assembly Diagram, right-click **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

___ 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[]	60
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**

Name	Type	Value
order	Order	✓
customerID	string	cust123
items	OrderItem[]	

Press 'Alt+Enter' to add lines

2) Right-click anywhere on the row containing **items** and select **Add Elements...** from the pop-up menu

Name	Type	Value
order	Order	✓
customerID	string	✓ cust123
items	OrderItem[]	60

Context menu options: Copy Value, Paste Value, Select All, Add Elements...

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**

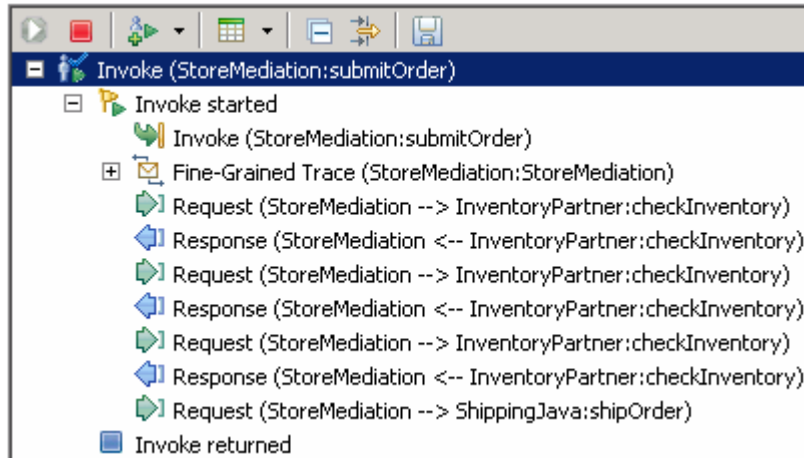
7) Enter values for items[2]:

a) For **itemID**, click under Value and enter **item002**

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

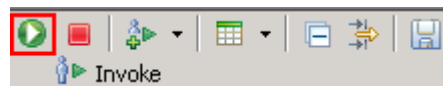
8) Your input data should now look like the table shown above.

- 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.

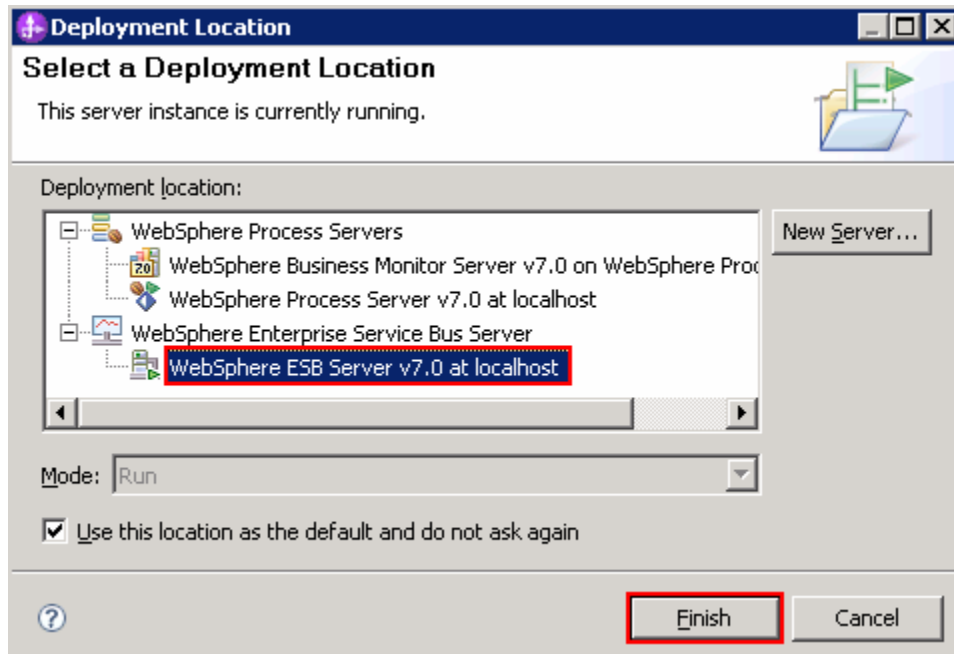


NOTE: The Fine-Grained Trace section has been collapsed. When expanded, it shows all of the nodes and primitives invoked by the flow. Although not discussed or explained in this lab, clicking on any of the entries within the fine grained trace will show the contents of the SMO at that point in the flow (in the Mediation Message panel to the right). This is a good way for you to see the changes in the SMO made by each of the primitives.

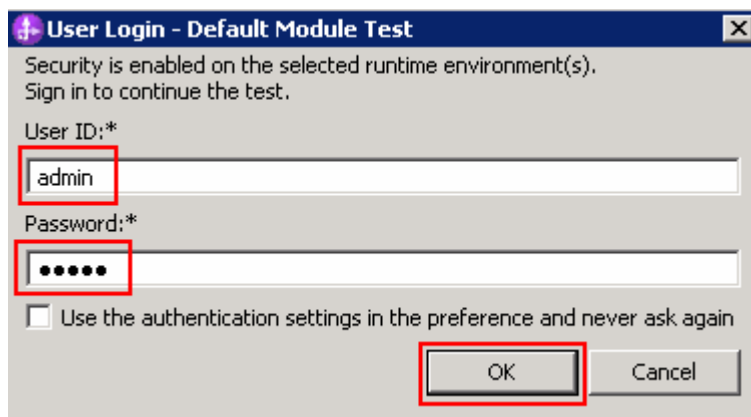
- -----
a. Click **Continue** icon (🟢) under Events panel



- ___ b. If test client is run for the first time, you should see a **Deployment Location** dialog prompting to select a run time sever where the applications are deployed. Select **WebSphere ESB Server v7.0** from the list



- ___ c. Click **Finish**
- ___ d. 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.



- ___ e. Click **OK**
- ___ f. 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 second item with itemID=item009, but your test run might show the fault occurring on different items.

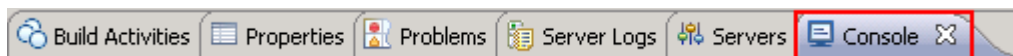
```

O *****
O ***** START mediation flow *****
O ***
I processMessage
O ***** InvRandom - returning InventoryItem for itemID = item001
I processMessage
O ***** InvRandom - EXCEPTION: InventoryFault for itemID = item009
I com.ibm.ws.ffdc.impl.FfdcProvider logIncident FFDC1003I: FFDC Incident emitted on C:\WID7_WTE\runtimes\bi_v7\profiles\
I com.ibm.ws.ffdc.impl.FfdcProvider logIncident FFDC1003I: FFDC Incident emitted on C:\WID7_WTE\runtimes\bi_v7\profiles\
I com.ibm.ws.ffdc.impl.FfdcProvider logIncident FFDC1003I: FFDC Incident emitted on C:\WID7_WTE\runtimes\bi_v7\profiles\
I processMessage
E ServiceMessageObject: ServiceMessageObject@18321832 (context=ContextType@18791879, headers=HeadersType@d560d56, body:
O ***** InvRandom - returning InventoryItem for itemID = item002
I processMessage
O ***
O ***** END mediation flow *****
O *****
O -----
O -- Ship object dump begins -----
O
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@6e7e6e7e
O Value:
O customerID = cust123
O items = ShipItem[3]
O items[0] = <ShipItem@6e9b6e9b>
O itemID = item001
O orderQuantity = 3
O inventoryQuantity = 5
O inventoryStatus = OK - but stock is running low
O items[1] = <ShipItem@6ea86ea8>
O itemID = item009
O orderQuantity = 5
O inventoryQuantity = 0
O inventoryStatus = ERROR during inventory check, failure accessing inventory information for item = item009
O items[2] = <ShipItem@6eb56eb5>
O itemID = item002
O orderQuantity = 15
O inventoryQuantity = 10
O inventoryStatus = Backorder - insufficient stock to fill order
O
O -- Ship object dump ends -----
O -----

```

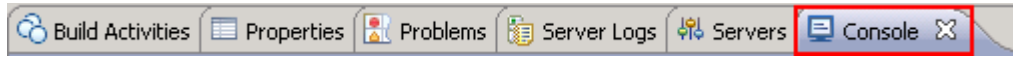
NOTE: Only the messages starting with “O” are produced by the application. The messages starting with “I” and “E” are produced by the system, and may or may not be present in your runs.

- -----
- ___ a. Check to see if the tab for the **Console** view is showing (lower right quadrant of perspective)



___ b. If it is not showing, open it by selecting menu items **Window → Show View → Console**

___ c. Double click tab for the **Console** view to maximize it for a better view of the server trace.



___ d. Scroll to the bottom of the console to see the output from the test run (compare with output shown above).

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**
- **Try alternate endpoints:** **unchecked**

Retry on:

Retry count:

Retry delay (seconds):

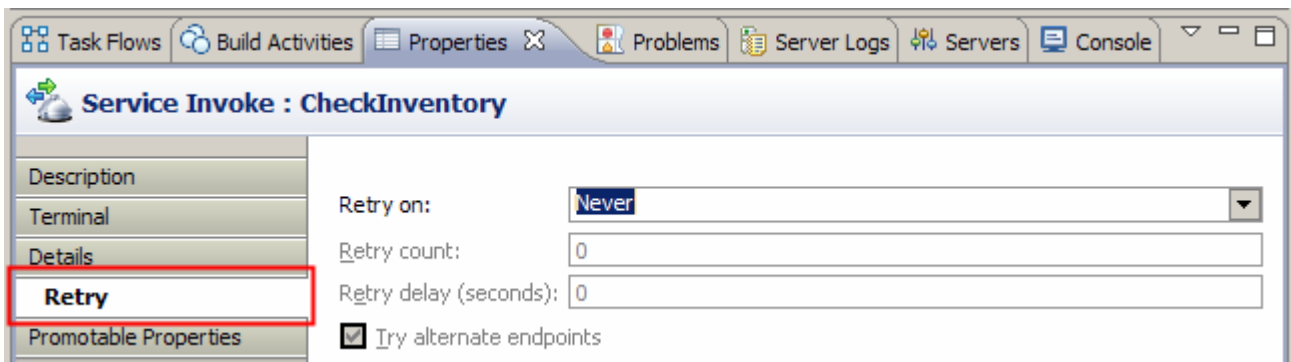
Try alternate endpoints

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

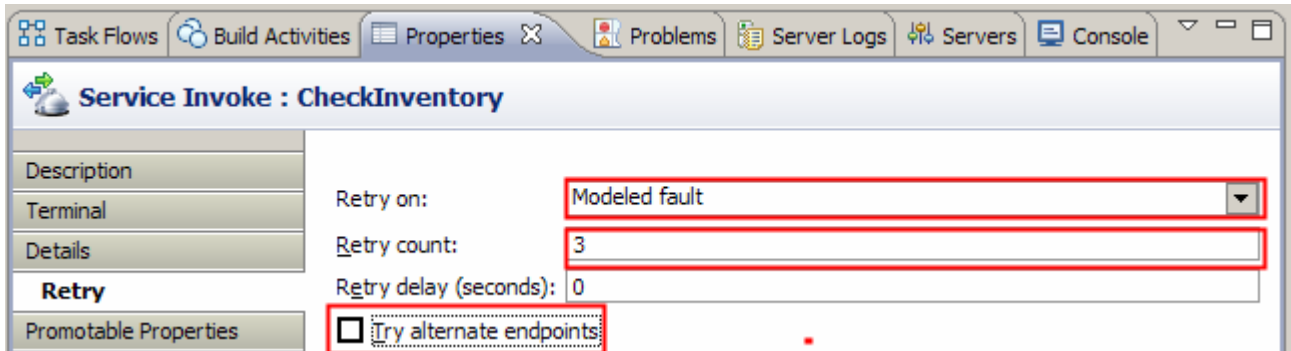
1) In the Business Integration view, expand **StoreMediation** → **Integration Logic** → **Mediation Flows** and then double-click **StoreMediation** to open it in the mediation flow editor

2) In the mediation flow editor double click the source operation. In this case, **submitOrder** as shown in the picture above. This action opens the Request flow

___ b. Select **CheckInventory** service invoke primitive and then select **Properties** → **Details** → **Retry**



- ___ c. For **Retry on**, select **Modeled fault** from the drop down list
- ___ d. For **Retry count**, enter **3**
- ___ e. **Remove** the check for **Try alternate endpoints**



- ___ 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:

Part 5: Test the service call retry mediation

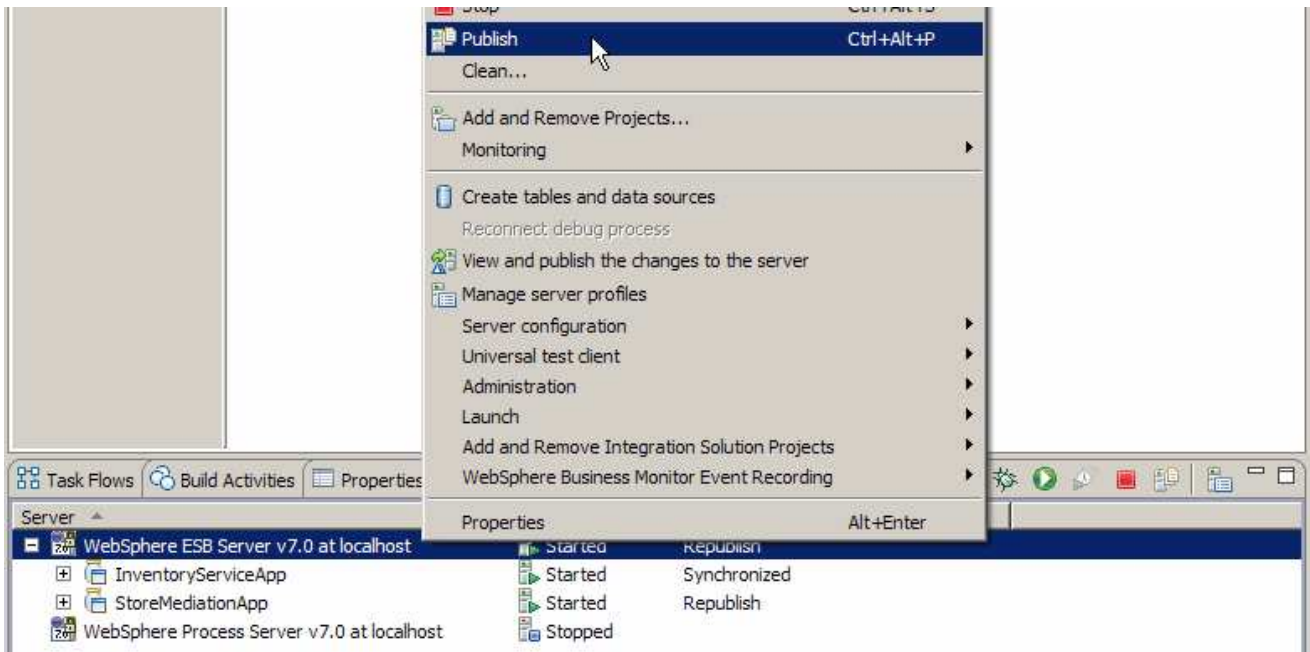
What you will do in this part: In this part, you use the component test facilities of WebSphere Integration Developer to test the mediation. The resulting output is then explained. It should be noted that for this lab exercise the implementation of the inventory service is rigged to produce the required 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.

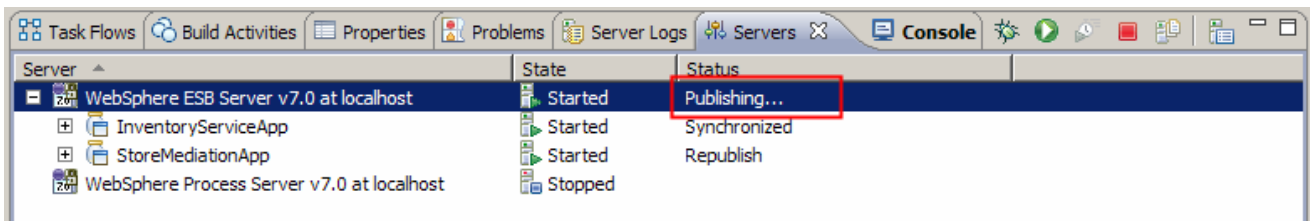
___ 1. Do a publish to synchronize and restart the applications in the test server

- -----

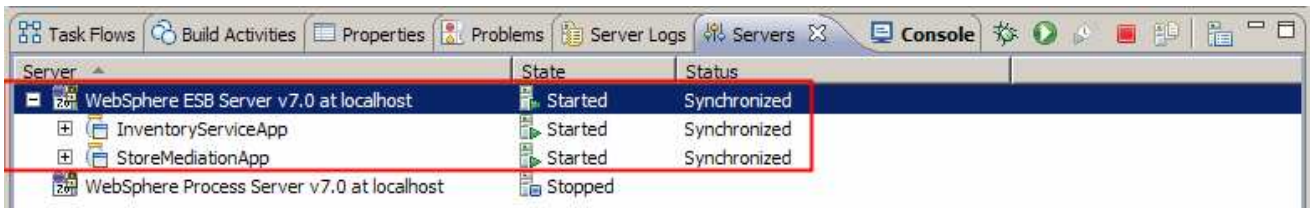
___ a. In the **Servers** view right click the server and select **Publish** from the pop-up menu



___ b. Wait while the applications are publishing to the server



__ c. Once the publishing is done you should see the two applications started as shown here:



___ 2. In StoreMediation_Test, use the Rerun option to run the test with the same input data as the last run.

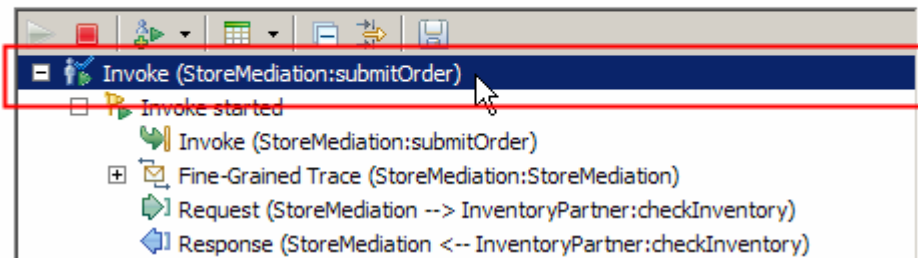
- -----

__ a. Select tab for the **StoreMediation_Test** panel

__ b. Click the last Invoke (**StoreMediation:submitOrder**)

Events

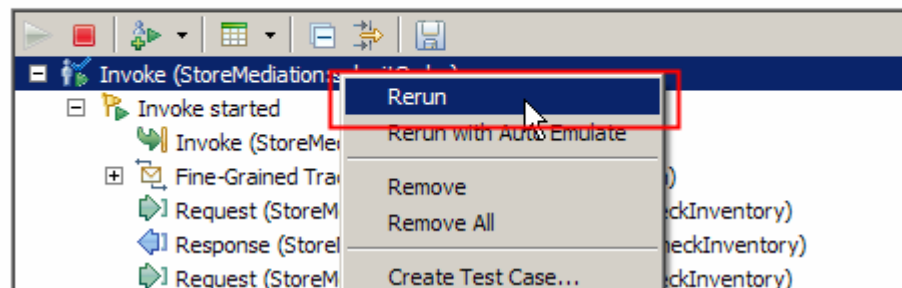
This area displays the events in a test trace. Select an event to display its properties in the General Properties and Detailed Properties sections. [More...](#)



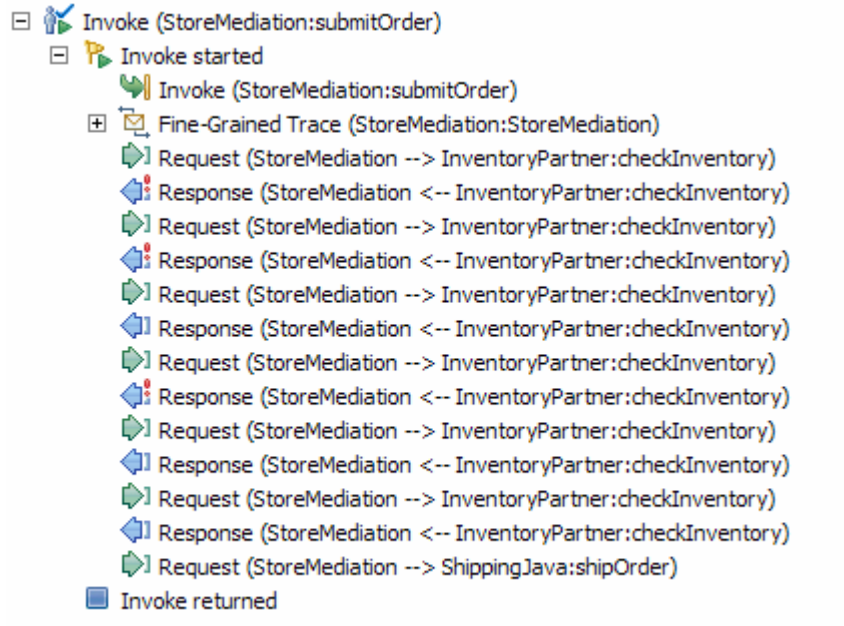
__ c. Right click to get a pop-up menu, and select **Rerun**

Events

This area displays the events in a test trace. Select an event to display its properties in the General Properties and Detailed Properties sections. [More...](#)



3. Examine the output in StoreMediation_Test. 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 six 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)



4. Switch to the Console view and examine the output which should be similar to the screen capture on the following page. In the specific case shown in this screen capture, the initial call for item001 failed and required two retries to succeed. Then the initial call for item009 also failed, but the first retry was successful in this case. Finally, 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.

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



```
O *****
O ***** START mediation flow *****
O ***
I processMessage item001
O ***** InvRandom - EXCEPTION: InventoryFault for itemID = item001
I processMessage
E ServiceMessageObject: ServiceMessageObject@322c322c (context=ContextType@327:
O ***** InvRandom - EXCEPTION: InventoryFault for itemID = item001
I processMessage
E ServiceMessageObject: ServiceMessageObject@27652765 (context=ContextType@27a:
O ***** InvRandom - returning InventoryItem for itemID = item001
I processMessage item009
O ***** InvRandom - EXCEPTION: InventoryFault for itemID = item009
I processMessage
E ServiceMessageObject: ServiceMessageObject@696c696c (context=ContextType@69b:
O ***** InvRandom - returning InventoryItem for itemID = item009
I processMessage item002
O ***** InvRandom - returning InventoryItem for itemID = item002
I processMessage
O ***
O ***** END mediation flow *****
O *****
O -----
O -- Ship object dump begins -----
O
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@bb50bb5
O Value:
O customerID = cust123
O items = ShipItem[3]
O items[0] = <ShipItem@bd20bd2>
O itemID = item001
O orderQuantity = 3
O inventoryQuantity = 5
O inventoryStatus = OK - but stock is running low
O items[1] = <ShipItem@bdf0bdf>
O itemID = item009
O orderQuantity = 5
O inventoryQuantity = 45
O inventoryStatus = OK - sufficient stock levels
O items[2] = <ShipItem@bec0bec>
O itemID = item002
O orderQuantity = 15
O inventoryQuantity = 10
O inventoryStatus = Backorder - insufficient stock to fill order
O
O -- Ship object dump ends -----
O -----
```


___ 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:

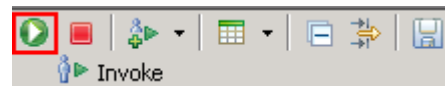
- **customerID** can be any string and should not have any particular affect on the results
- The **items** array can have any number of elements
- **itemID** values of "item001" through "item010" are recognized by the inventory service, all other values are not recognized
- Inventory status will change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

___ c. Click **Continue** icon () under Events panel

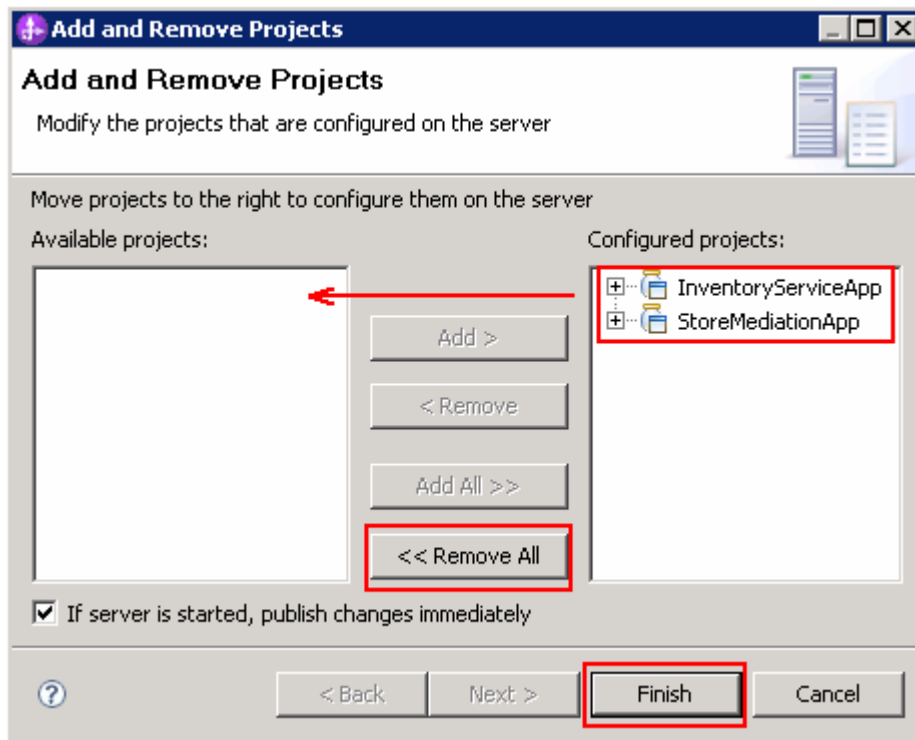


Part 6: Clean up the environment if you will not proceed to the next lab

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 **WebSphere ESB Server v7.0** under the Servers view and select **Add and remove projects...** from the pop-up 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. Wait until the application is removed from the server

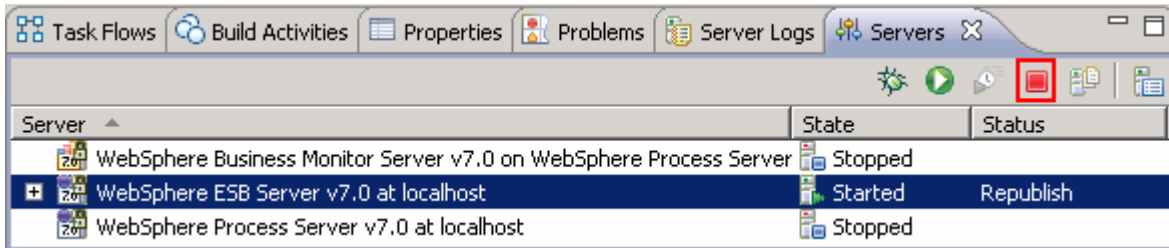
___ 2. Close the StoreMediation_test panel without saving

- -----
- ___ a. Close the StoreMediation_Test editor
- ___ b. Click **No** from Save Test Trace window
- ___ c. Close **StoreMediation** – Assembly Diagram

___ 3. Stop the test server

- -----

___ a. From the **Servers** view of WebSphere Integration Developer, select **WebSphere ESB Server v7.0** 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**

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 retry.

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.