

IBM WEBSHERE 6.1 – LAB EXERCISE

# WebSphere Enterprise Service Bus 6.1 mediation programming model

## Lab Two – Message splitting and aggregating

What this exercise is about .....	1
Lab requirements .....	1
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 for splitting and aggregating .....	8
Part 3: Test the splitting/aggregating mediation .....	40
Part 4: Clean up the environment if you will not proceed to the next lab.....	50
What you did in this exercise .....	52
Solution instructions .....	53
Task: Adding remote server to the WebSphere Integration Developer test environment .....	54

### What this exercise is about

The objective of this lab is to provide you with an understanding of how to use the fan out and fan in mediation primitives to do message splitting and aggregating within a mediation flow. This allows a message with repeating elements to perform individual processing for each element, such as augmenting the information about each 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 use the fan out and fan in mediation primitives in iterate mode to allow element by element processing of a message with repeating elements.
  - Use the returned values from a service invoke mediation primitive to augment information about each element.
  - Utilize the shared context to build up (aggregate) information about all elements during the iterative processing of each element.
  - Construct the final message containing the augmented information for all of the repeating elements.
- 

## Introduction

This lab exercise is the second 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 can be 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

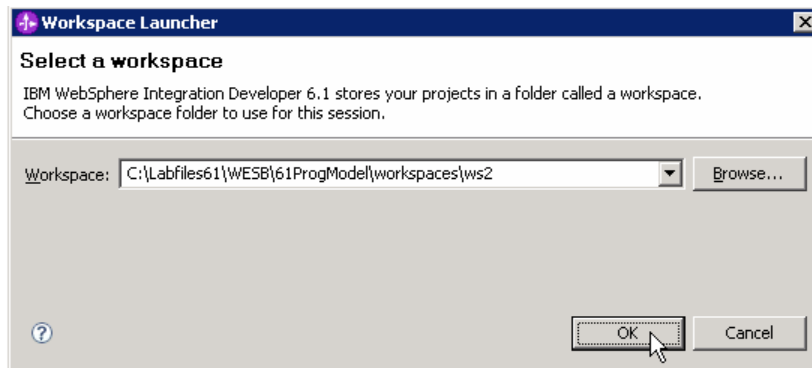
**What you will do in this part:** In this part you are getting the environment set up to do the lab. There are three different ways you might be approaching this which will dictate what you need to do.

**(1) Proceeding from Lab One** – You are directly continuing from Lab One 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 One** – You are continuing from Lab One 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 Two** - You are starting this lab from scratch, regardless of whether you had previously completed Lab 1.

- \_\_\_ 1. If you are **proceeding from Lab 1** there is nothing to do, skip directly to **Part 2 Authoring the mediation flow for splitting and aggregating**
- \_\_\_ 2. Start WebSphere Integration Developer.
  - **Restarting from Lab 1:** Use the same workspace used in Lab 1
  - **Directly starting Lab 2:** Suggested location:  
 <LAB\_FILES>/workspaces/ws2
  - -----
  - \_\_\_ 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 1: Use the same workspace used in Lab 1
    - 2) Directly starting Lab 2: <LAB\_FILES>/workspaces/ws2



- \_\_\_ 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.

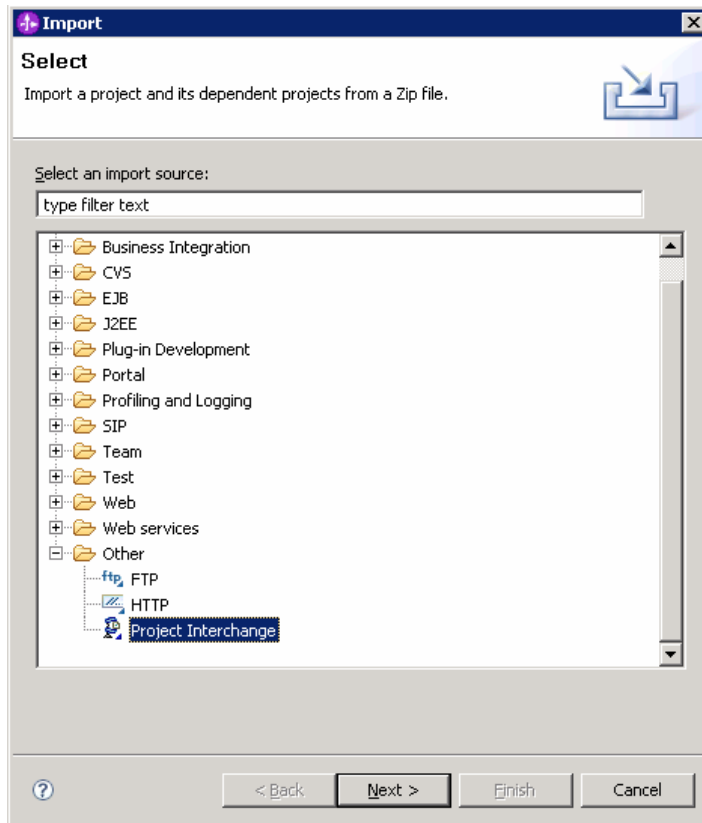


- \_\_\_ 3. If you are **restarting from Lab 1** there is nothing additional to do, skip directly to **Part 2 Authoring the mediation flow for splitting and aggregating**

- \_\_\_ 4. If you are **directly starting Lab Two** import the project interchange file containing the starting point for the lab
- <LAB\_FILES>/PI2-AugmentSolution-AggregateStart.zip
  - -----

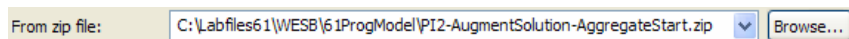
\_\_\_ a. From the menu, select **File → Import...**

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

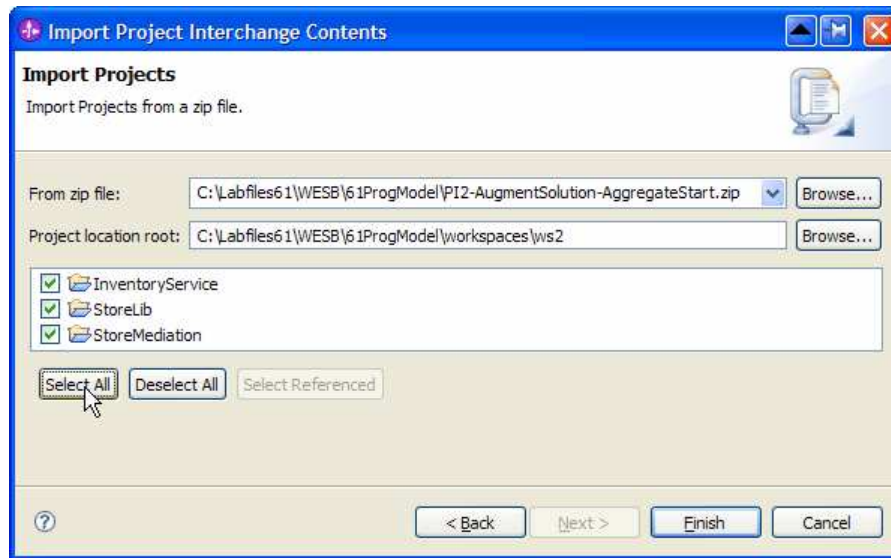


\_\_\_ c. Hit **Next >**

\_\_\_ d. In the **Import Project Interchange Contents** dialog set the **From zip file:** value to <LAB\_FILES>/PI2-AugmentSolution-AggregateStart.zip

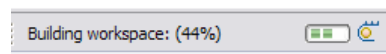


\_\_ e. Hit **Select All** to selected all of the projects



\_\_ f. Hit **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.



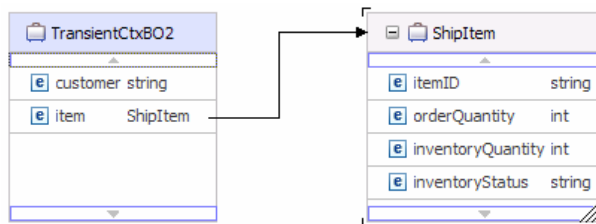
## Part 2: Authoring the mediation flow for splitting and aggregating

**What you will do in this part:** You will start with the completed flow for Lab One which performs augmentation of a single element message. You will modify that flow so that it will perform splitting and aggregating of the message so that augmentation of multiple elements in the message can occur. By starting with the result of the previous lab, you are better able to compare and contrast the flow requirements for these two scenarios.

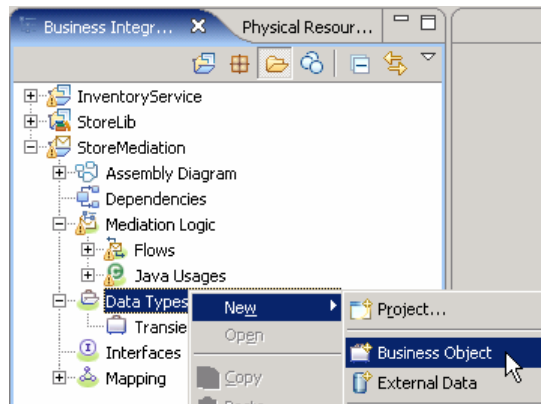
See the presentation entitled [Augmentation, aggregation and retry tutorials](#) to better understand what this part is doing

1. Create a new business object that is used as the transient context for the flow. Similar to the previous flow, information that might otherwise be lost during the flow is placed here. In addition, it is the place where all the item information for a single item is built up during the iterative part of the flow (that is item information from both Order and Inventory).

- **Module** : StoreMediation
- **Name** : TransientCtxBO2
- **Attributes:** customer string
- item ShipItem



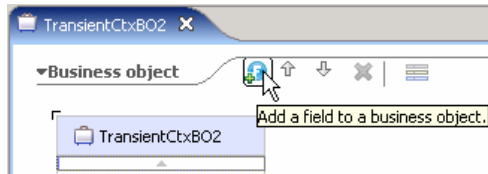
- a. In the Business Integration view, expand **StoreMediation**, right click **Data Types** and then select **New → Business Object** from the pop-up menu



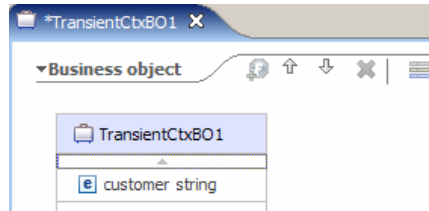
- b. In the **New Business Object** dialog, provide the name **TransientCtxBO2**
- c. Click **Finish**. The business object editor opens



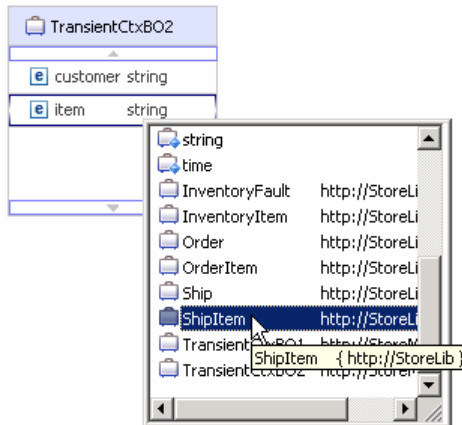
\_\_\_ d. In the editor, click the **Add a field to a business object** icon



\_\_\_ e. Edit the new field to have a name of **customer** and a type of **string**

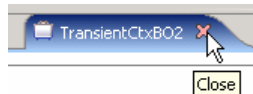


\_\_\_ f. Also add the field **item** of type **ShipItem**, using the drop down box to select the type



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

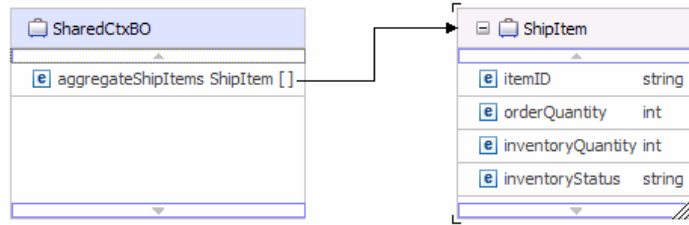
\_\_\_ h. Click on **X** to close TransientCtxBO2, business object editor



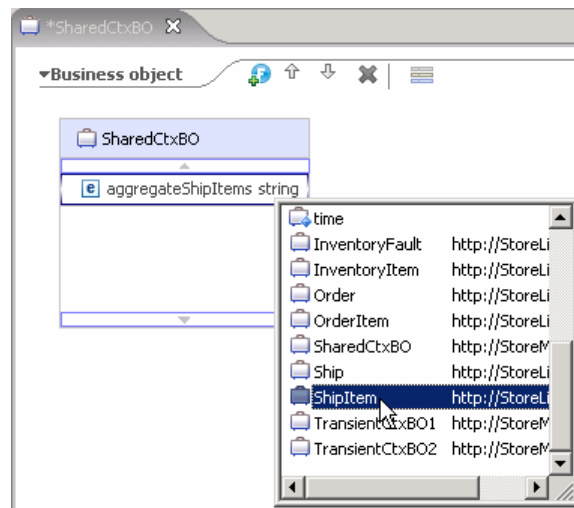
\_\_\_ 2. Create a new business object that is used as the shared context for the flow. The shared context defines the shared memory where the information for all of the items is aggregated during the iterative part of the flow.

- **Module** : StoreMediation
- **Name** : SharedCtxBO
- **Attributes:** aggregateShipItems ShipItem[ ]

**NOTE:** Ensure that the array check box in the properties for aggregateShipItems has been checked to make this attribute an array.

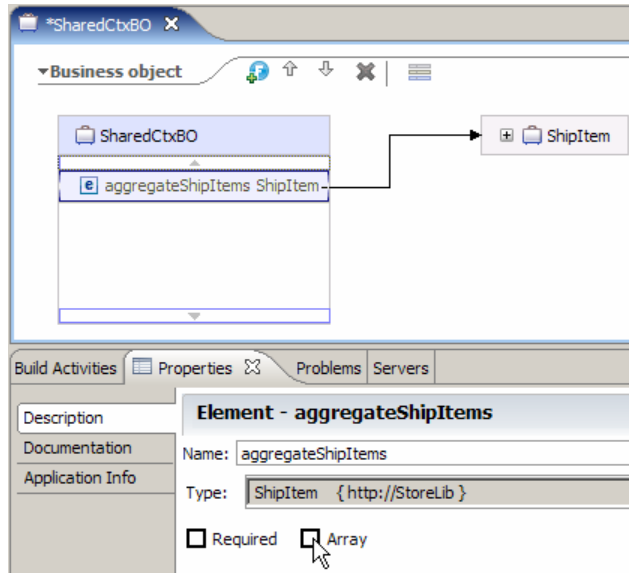


- \_\_\_ a. In the Business Integration view, expand **StoreMediation**, right click **Data Types** and then select **New → Business Object** from the pop-up menu
- \_\_\_ b. In the **New Business Object** dialog, provide the name **SharedCtxBO**
- \_\_\_ c. Click **Finish**
- \_\_\_ d. In the editor, click the **Add a field to a business object** icon
- \_\_\_ e. Edit the new field to have a name of **aggregateShipItems** and a type of **ShipItem** using the drop down box to select the type



- \_\_\_ f. Ensure that the aggregateShipItems attribute is highlighted, and then select **Properties → Description**

\_\_\_ g. Check the box for **Array**



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

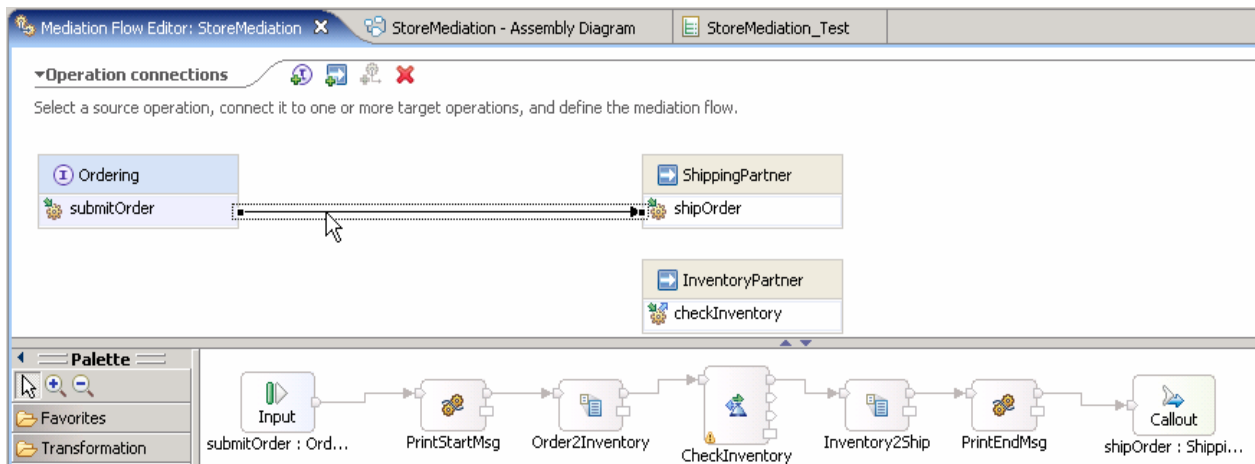
\_\_\_ i. Click on **X** to close SharedCtxBO, business object editor

\_\_\_ 3. Open the StoreMediation flow found in the StoreMediation module (this is the flow from Lab One which augments a single element in the message).



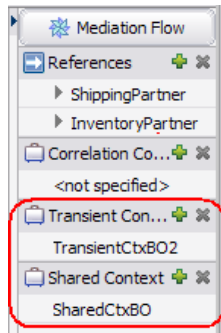
\_\_\_ a. In the Business Integration view, expand **StoreMediation → Mediation Logic → Flows** and then double-click on **StoreMediation** to open it in the mediation flow editor

\_\_\_ b. The line connecting **Ordering → submitOrder** and **ShippingPartner → shipOrder** should be selected for you to view the flow:



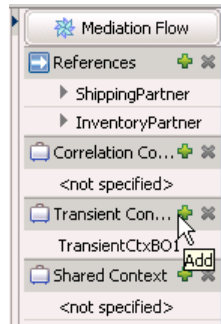
\_\_\_ 4. Configure the transient and shared contexts that are used in the flow.

- **Transient context: TransientCtxBO2**
- **Shared context : SharedCtxBO**

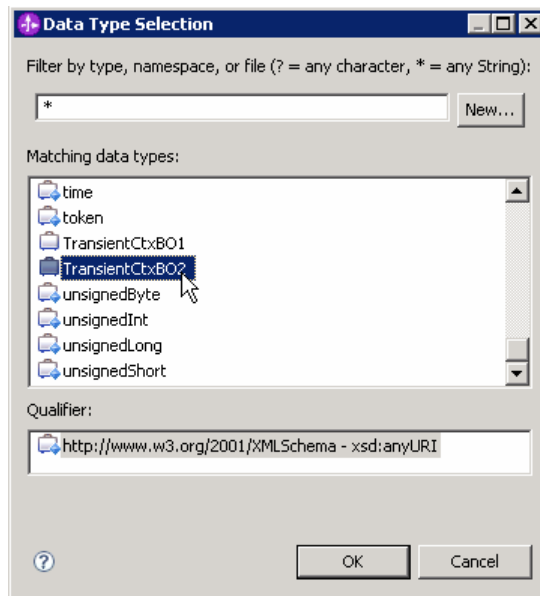


\_\_\_ a. Configure transient context

- 1) In the Mediation Flow on the right side, hit **Add (+)** icon next to **Transient Context**



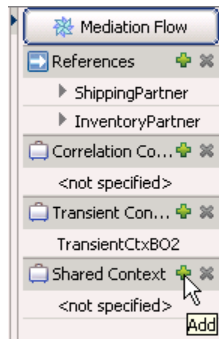
- 2) In the Data Type Selection window, scroll down to select **TransientCtxBO2** under **Matching data types:**



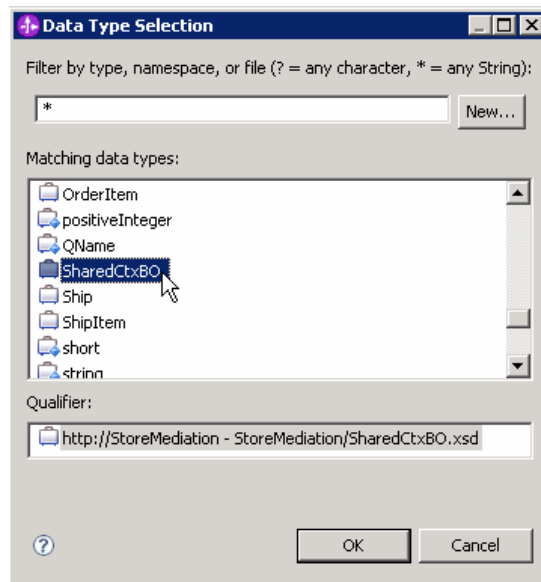
- 3) Click **OK**

\_\_\_ b. Now, configure shared context

1) In the Mediation Flow on the right side, hit **Add** (+) icon next to **Shared Context**



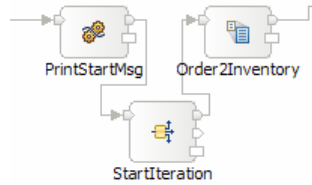
2) In the Data Type Selection window, scroll down to select **SharedCtxBO** under **Matching data types**:



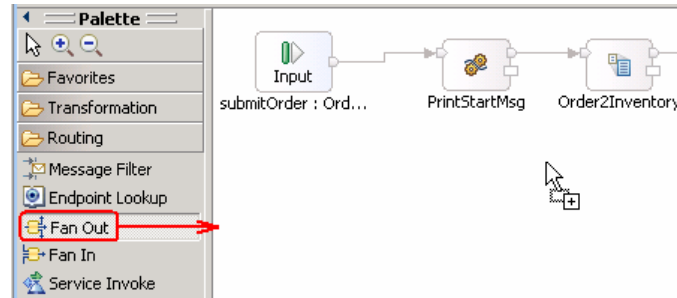
3) Click **OK**

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

- \_\_\_ 5. Add a fan out primitive to the canvas and rewire the flow so that it is between the PrintStartMsg and Order2Inventory primitives. This primitive defines where the iterative part of the flow begins.
- **Display Name:** StartIteration
  - **Wire** : PrintStartMsg to StartIteration
  - **Wire** : StartIteration(out terminal) to Order2Inventory

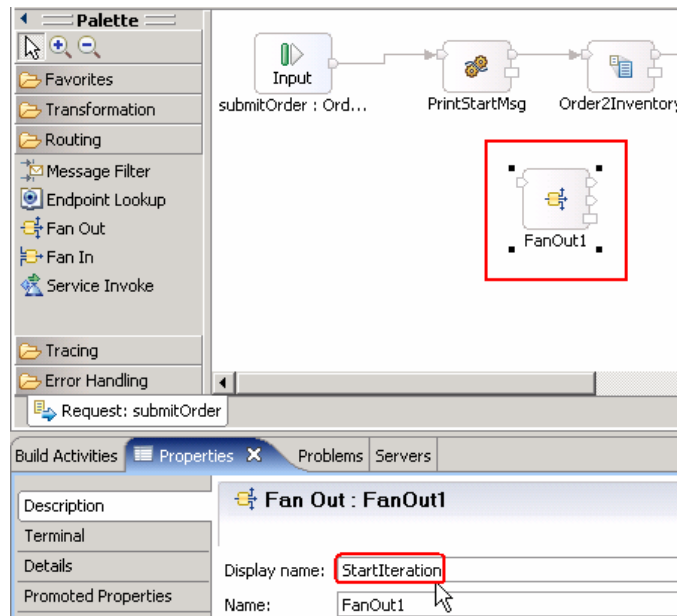


- -----
- \_\_\_ a. From the **Palette**, select **Routing** → **Fan Out** and then click on the canvas as shown below (between the PrintStartMsg and Order2Inventory primitives)



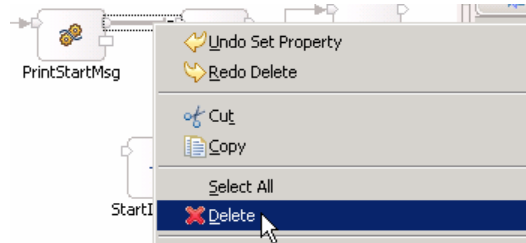
- \_\_\_ b. You will now see a new Fan Out primitive added to the flow. Ensure that this primitive is highlighted and select **Properties** → **Description**

- \_\_\_ c. Change the **Display name** to **StartIteration**



\_\_\_ d. Remove connection between **PrintStartMsg** and **Order2Inventory** primitives

- 1) Right-click on the line connecting **PrintStartMsg** and **Order2Inventory** primitives and select **Delete** from the pop-up menu



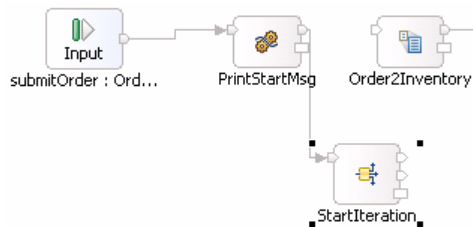
\_\_\_ e. Add a connection from **PrintStartMsg** to **StartIteration** primitive

- 1) Hover your mouse over **out** terminal of **PrintStartMsg** to get the 'Add a connection to an input terminal' (the orange bubble shown when hovering over the terminal)



- 2) Click on the "Add a connection to an input terminal" orange bubble and then click on **StartIteration** primitive

- 3) You should now see a connection from **PrintStartMsg** to **StartIteration** primitive



\_\_\_ f. Similarly, add a connection from **StartIteration** primitive to **Order2Inventory** primitive

- 1) Hover your mouse over **out** terminal of **StartIteration** to get the 'Add a connection to an input terminal'

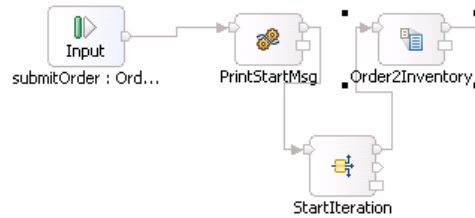
---

**NOTE:** There are two output terminals for a fan out primitive, so make sure you are wiring from the **out** terminal and not from the **noOccurrences** terminal.

---

- 2) Click on the orange bubble and then click on **Order2Inventory** primitive

\_\_ g. You should now see a connection from StartIteration to Order2Inventory primitive:



\_\_\_ 6. Configure the StartIteration fan out primitive to iterate through the items array in the Order business object.

- **For each element in XPath expression:**  
`/body/submitOrder/order/items`

Fire output terminal with original input message

once

for each element in XPath expression

---

**NOTE:** To set the value of the XPath expression you open the XPath Expression Builder using the **Edit...** button. Note that it works differently than it did in V6.0.2. Using the Data Types Viewer, you must either double click on an element or drag the element and drop it to get the XPath Expression field initialized with the path.

---

- -----

\_\_ a. Ensure that the StartIteration fan out primitive is highlighted and then select **Properties → Details**

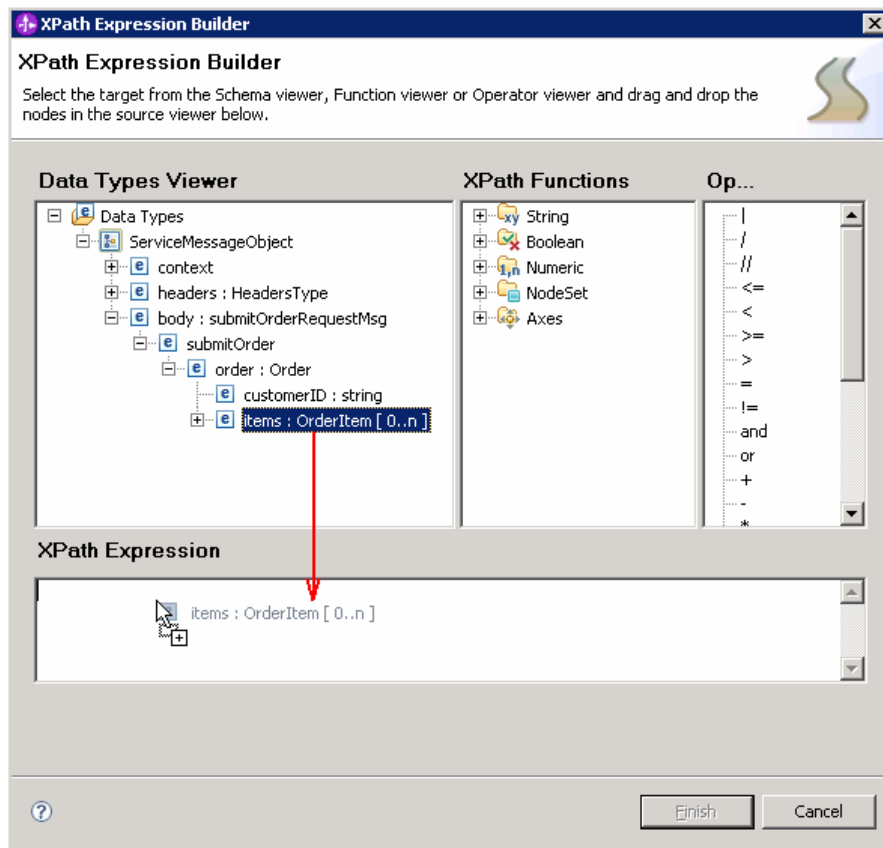
\_\_ b. Select the radio button next to 'for each element in XPath expression'

\_\_ c. Click the **Edit...** button. The XPath Expression Builder wizard is opened

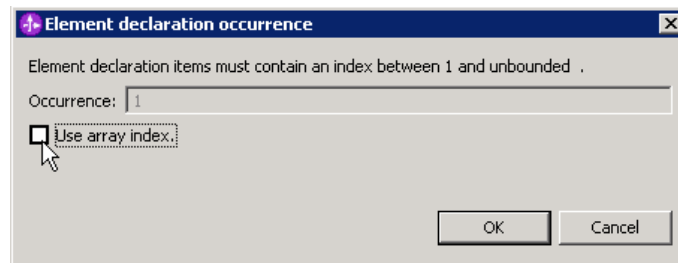
\_\_ d. Expand **ServiceMessageObject → body → submitOrder → order**



\_\_\_ e. Drag **items** and drop it under **XPath Expression** (or double-click on items).



\_\_\_ f. Element declaration occurrence window will pop-up. Clear the check box for 'Use array index'



\_\_\_ g. Click **OK**

\_\_\_ h. You should now see this expression under **XPath Expression**



\_\_\_ i. Click **Finish**

\_\_\_ 7. In the Order2Inventory XSL transformation primitive create a new map named Order2Inventory2 (this will replace the existing Order2Inventory1 map currently configured for the primitive).

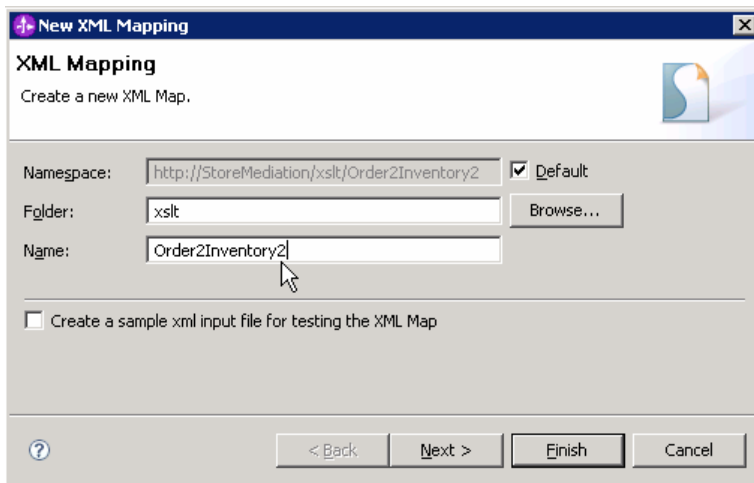
- **Map Name** : Order2Inventory2
- **Message Root** : /
- **Input Message Body** : submitOrderRequestMsg

- **Output Message Body:**     **checkInventoryRequestMsg**

\_\_\_ a. Select **Order2Inventory** primitive from the canvas and then select **Properties** → **Details**

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

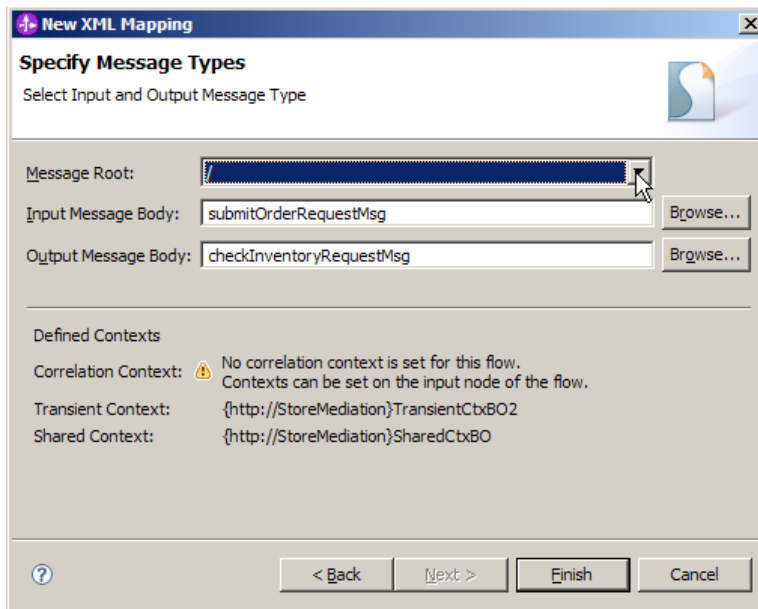
\_\_\_ c. For **Name** enter **Order2Inventory2**



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

\_\_\_ e. From Specify Message Types screen

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

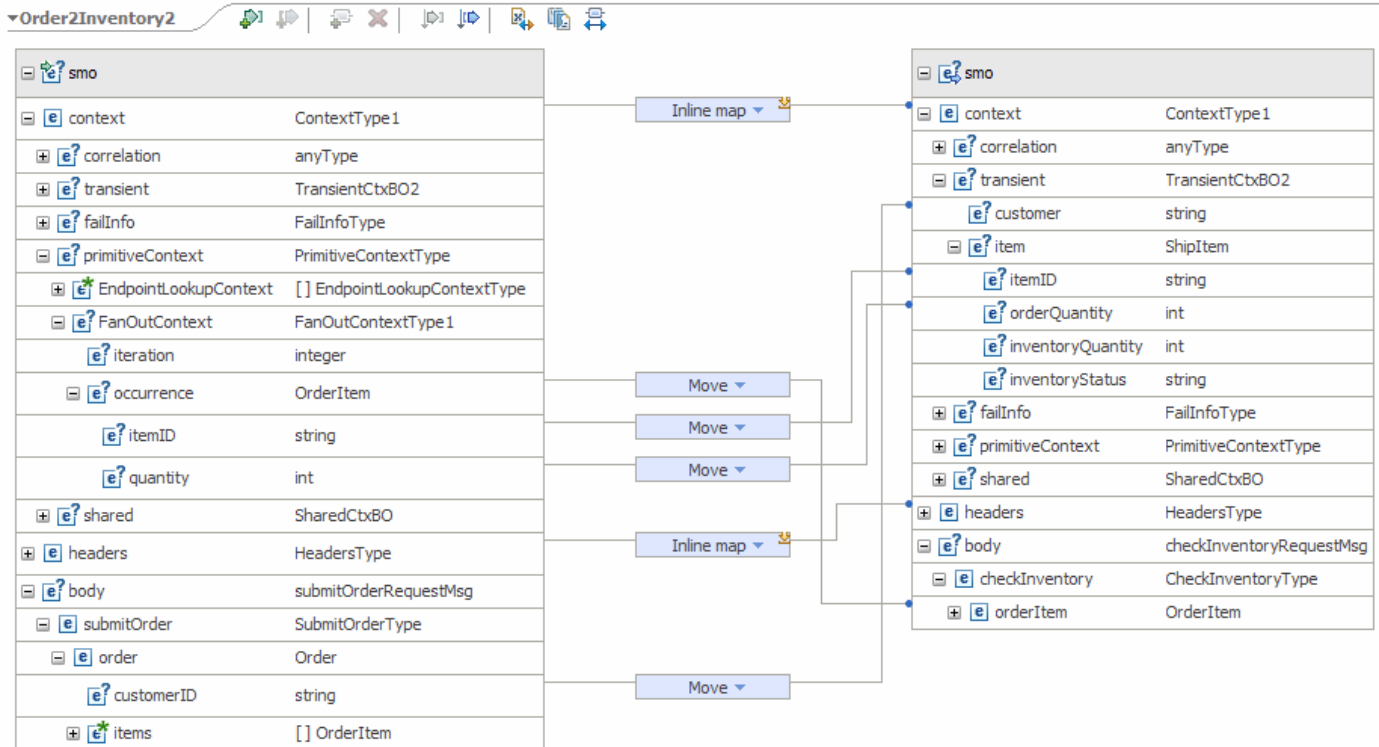


\_\_\_ f. Click **Finish**. An Order2Inventory2.map file is opened in the XML Mapping editor

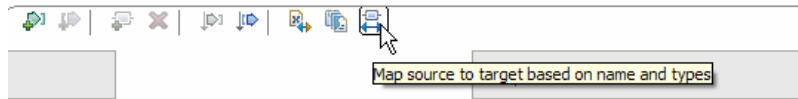
\_\_\_ 8. Define the mapping for the Order2Inventory2 map. The mapping must address: (1) Moving fields between the source and target SMOs that will not change. (2) Saving information in the transient context from the source message body that is needed later in the flow. (3) Saving information in the transient context from the FanOutContext that is needed later in the flow. (4) Setting up the target message body needed to call the Inventory service using information from the FanOutContext

- Use toolbar icon (↔) 'Map source to target based on name and types'
- Navigate down into the context/transient to context/transient inline map
- Remove the customer to customer move transformation
- Remove the item to item inline transformation
- At the SMO level (top level map) add these move transforms

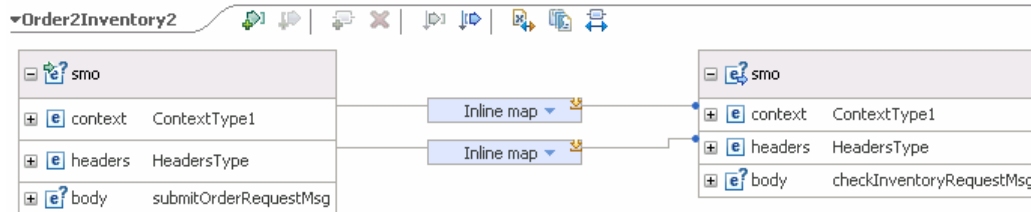
Source (left side)	Target (right side)
body/submitOrder/order/customerID	context/transient/customer
context/primitiveContext/FanOutContext/occurrence/itemID	context/transient/item/itemID
context/primitiveContext/FanOutContext/occurrence/quantity	context/transient/item/orderQuantity
context/primitiveContext/FanOutContext/occurrence	body/checkInventory/orderItem



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

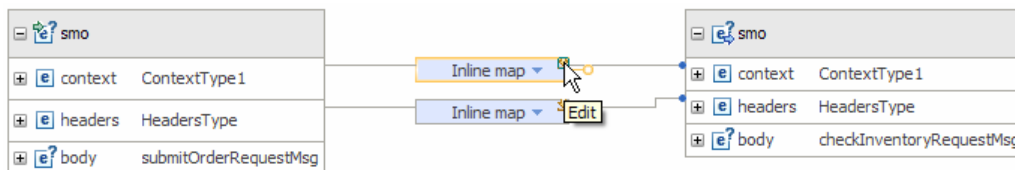


\_\_ b. This will map context and headers using inline maps as shown below:

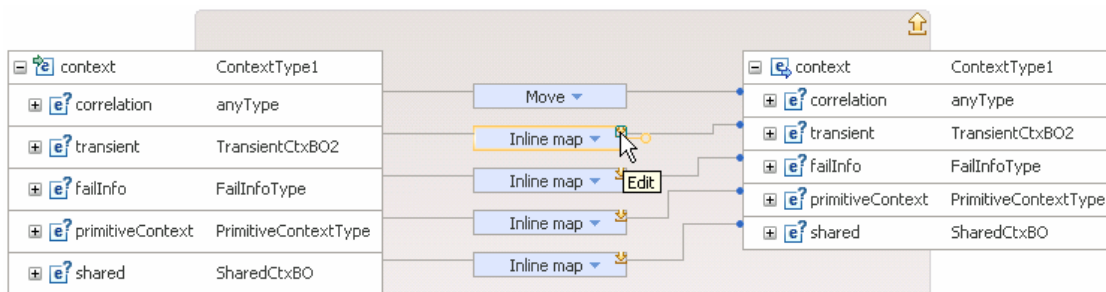


\_\_ c. Navigate down into the context/transient to context/transient inline map and remove all of the transformations

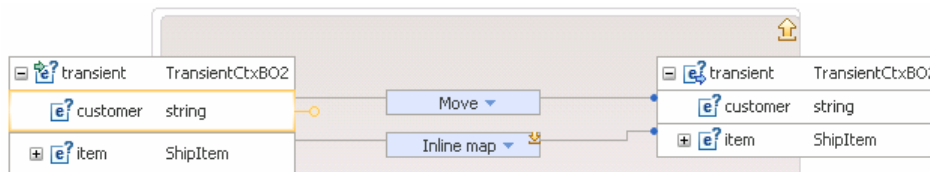
1) Click on **Edit** (🔧) icon on the Inline map connecting **context** attributes



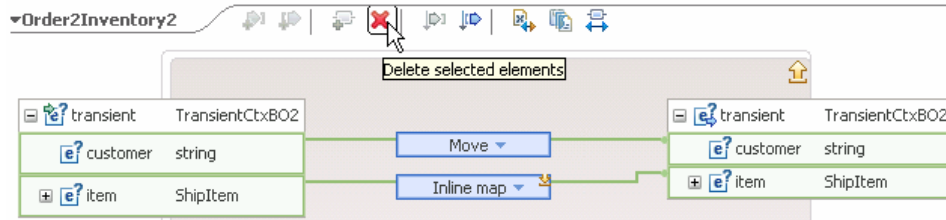
2) This will open all the transformations defined under context. Click on **Edit** (🔧) icon on the Inline map connecting **transient** attributes:



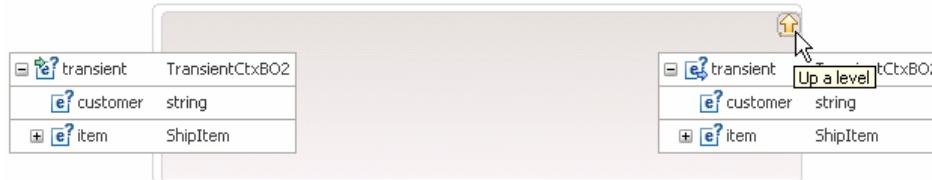
3) This will open the transformations of transient:



4) Select both **Move** and **Inline map** transformations (hold down Ctrl button on keyboard and click on each transformation) and hit '**Delete selected elements**' (✖) icon from the top:



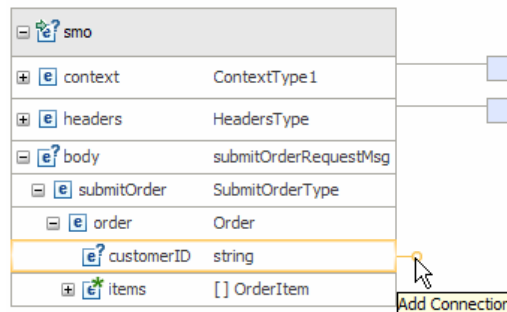
5) Click on 'Up a level' (↑) icon which will bring you one level up to the context mapping



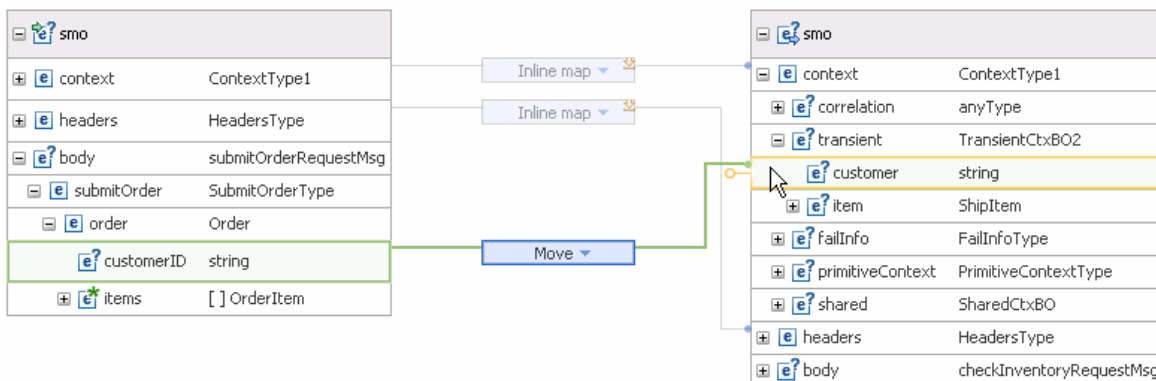
6) Click on 'Up a level' (↑) icon one more time which will now bring you back the smo mapping level

\_\_\_ d. Move body/submitOrder/order/customerID to context/transient/customer

- 1) In the left smo, expand **body** → **submitOrder** → **order**
- 2) In the right smo, expand **context** → **transient**
- 3) In the left smo, hover your mouse over **customerID** and click on **Add Connection**:

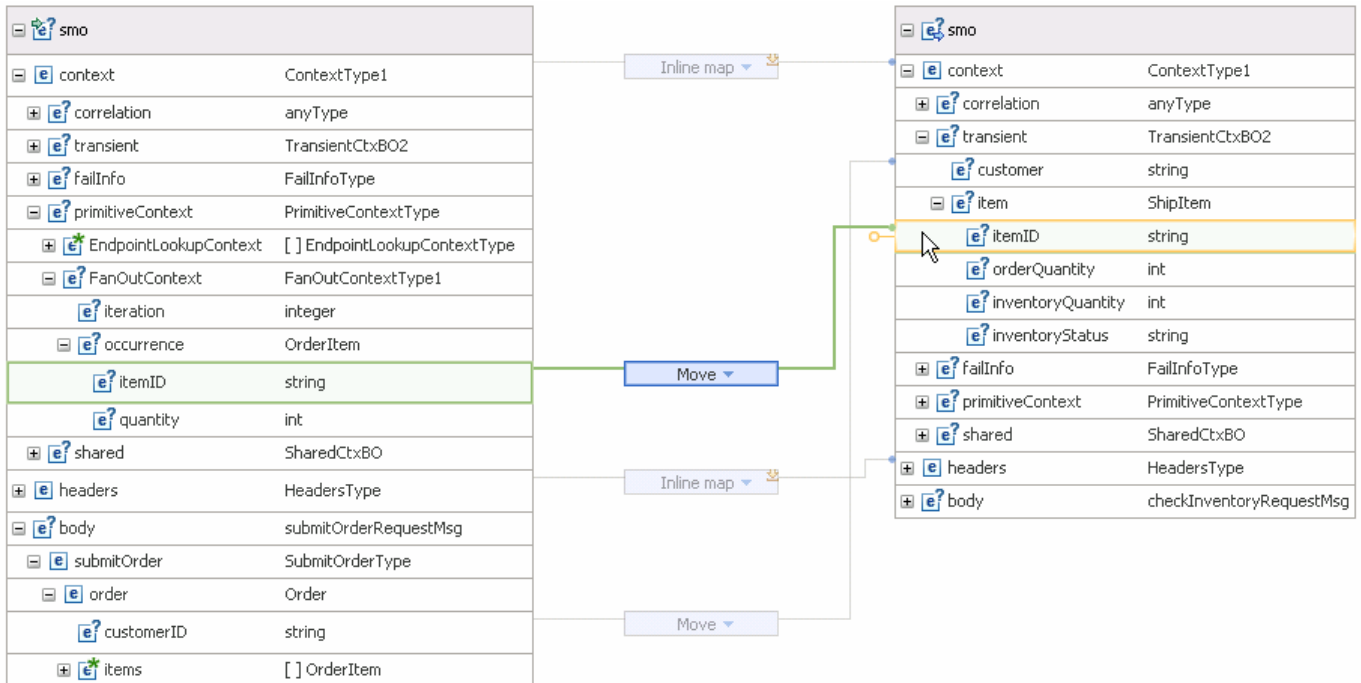


4) Now click on **customer** in the right smo and you should see a new connection from customerID to customer:

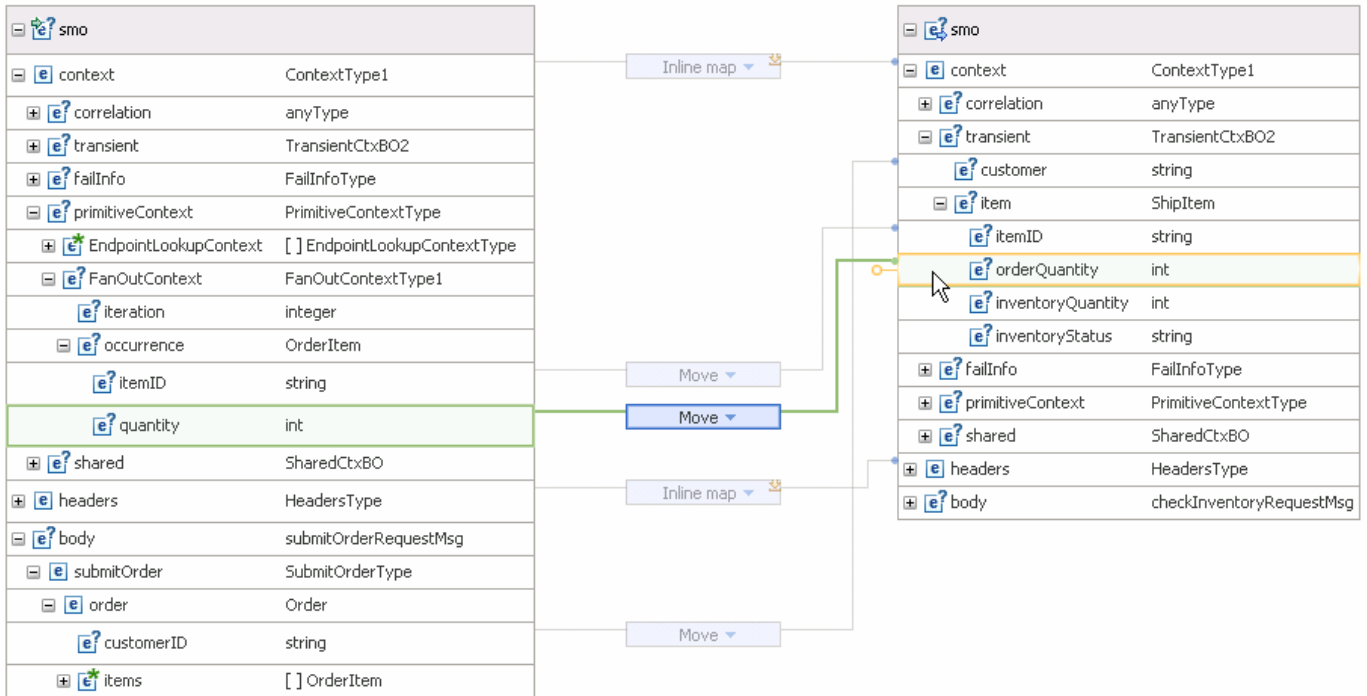


\_\_\_ e. Move context/primitiveContext/FanOutContext/occurrence/itemID to context/transient/item/itemID

- 1) In the left smo, expand **context** → **primitiveContext** → **FanOutContext** → **occurrence**
- 2) In the right smo, expand **context** → **transient** → **item**
- 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:

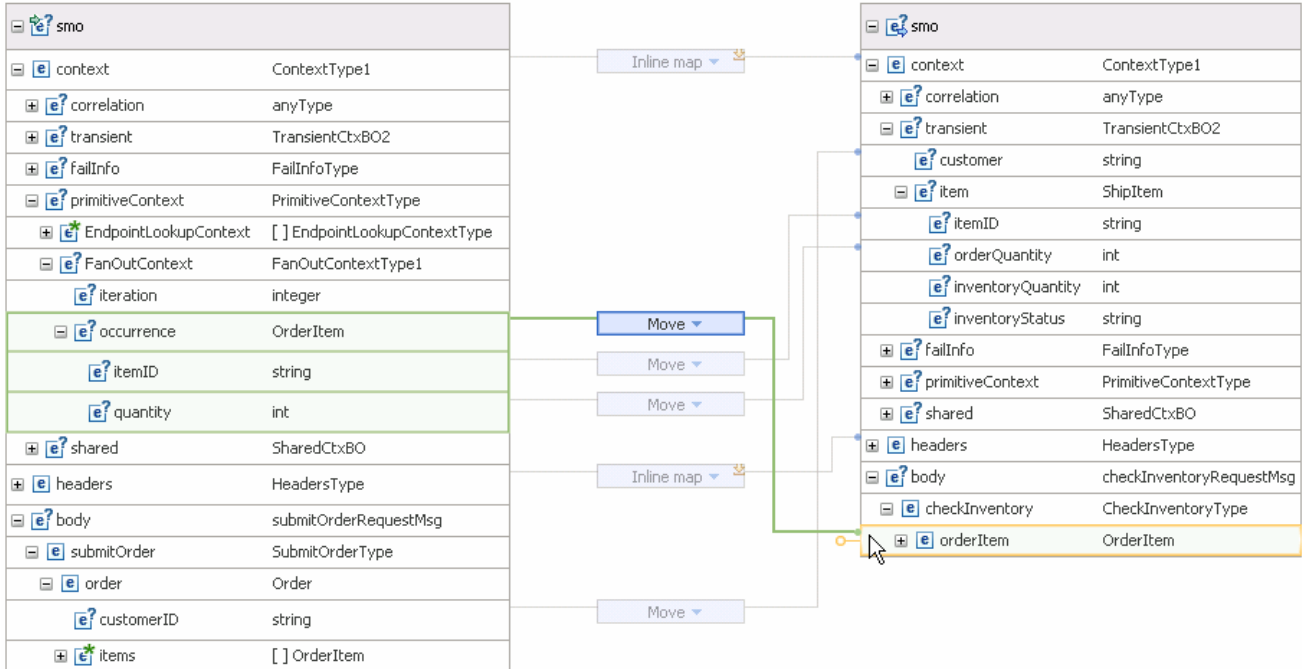


\_\_\_ f. Similarly, move context/primitiveContext/FanOutContext/occurrence/**quantity** to context/transient/item/**orderQuantity**:



\_\_\_ g. Now, move context/primitiveContext/FanOutContext/occurrence to body/checkInventory/orderItem

- 1) In the left smo, expand **context** → **primitiveContext** → **FanOutContext** → **occurrence**
- 2) In the right smo, expand **body** → **checkInventory** → **orderItem**
- 3) In the left smo, hover your mouse over **occurrence** and click on **Add Connection**
- 4) Now click on **orderItem** in the right smo and you should see a new connection from occurrence to orderItem:



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

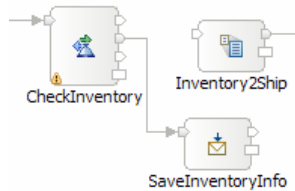
\_\_\_ i. Click on **X** to close Order2Inventory2.map file

\_\_\_ 9. The CheckInventory, service invoke primitive does not need to be changed.

• -----

\_\_\_ 10. Add a message element setter primitive and rewire the flow so that it follows the CheckInventory primitive.

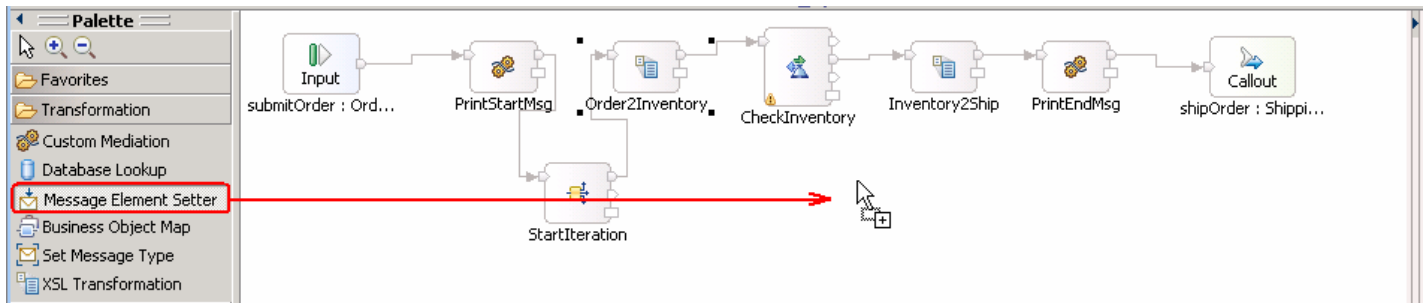
- **Display Name:** SaveInventoryInfo
- **Wire:** CheckInventory(out terminal) to SaveInventoryInfo



• -----

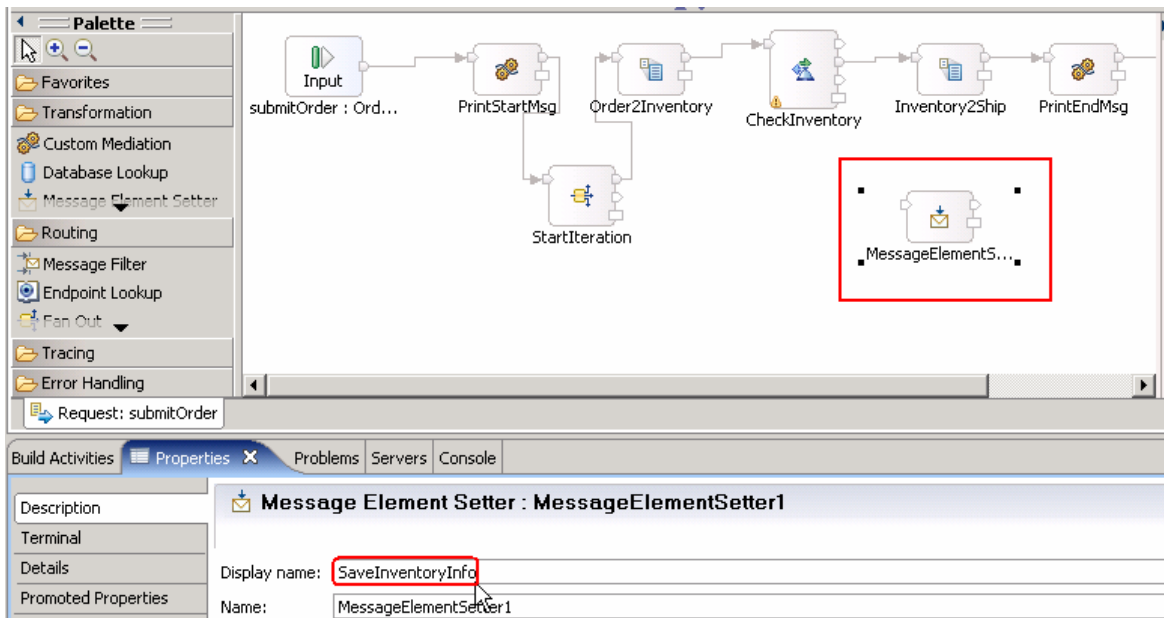
\_\_\_ a. From the **Palette**, select **Transformation** → **Message Element Setter** and then click on the canvas as shown below (between the CheckInventory and Inventory2Ship primitives)





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

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



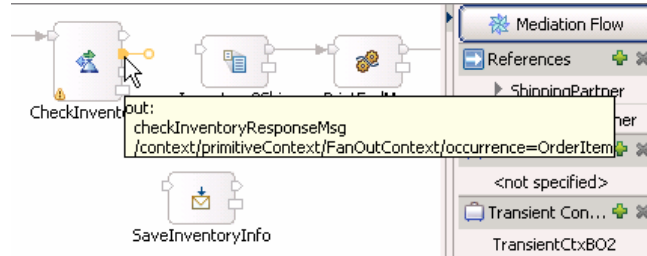
\_\_ d. Remove connection between CheckInventory and Inventory2Ship primitives

- 1) Right-click on the line connecting **CheckInventory** and **Inventory2Ship** primitives and select **Delete** from the pop-up menu

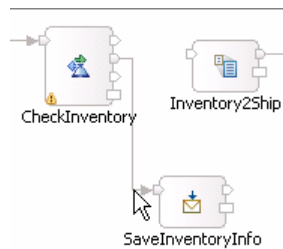
\_\_\_ e. Add a connection from **CheckInventory** to **SaveInventoryInfo** primitive

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

**NOTE:** The top two terminals, **out** and **timeout**, do not always appear in the same order on the primitive. Ensure you are adding the connection from the **out** terminal.



- 2) Click on the orange bubble and then click on **SaveInventoryInfo** primitive
- 3) You should now see a connection from **CheckInventory** to **SaveInventoryInfo** primitive



\_\_\_ 11. Configure the **SaveInventoryInfo** primitive to copy the inventory information from the body of the response message to the transient context.

- Set properties of message element setter to copy values as follows

Target location in SMO	Value (source) location in SMO
/context/transient/item/inventoryQuantity	/body/checkInventoryResponse/inventoryItem/inStockQuantity
/context/transient/item/inventoryStatus	/body/checkInventoryResponse/inventoryItem/status

Message Elements:

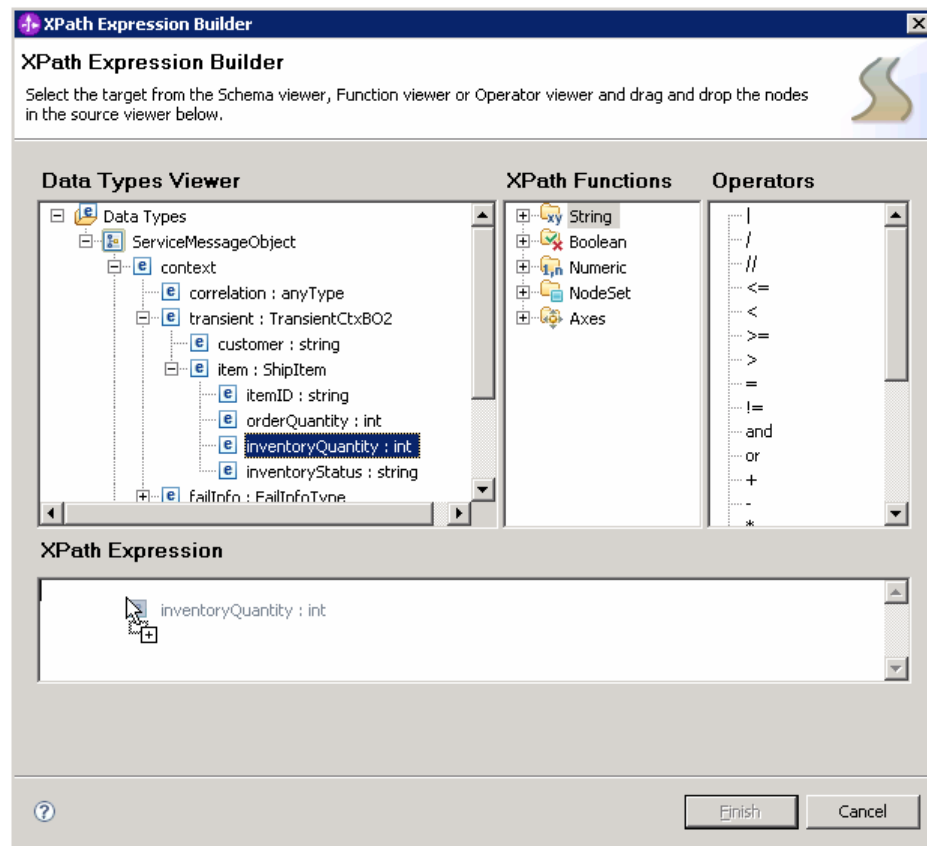
Target	Type	Value
/context/transient/item/inventoryQuantity	copy	/body/checkInventoryResponse/inventoryItem/inStockQuantity
/context/transient/item/inventoryStatus	copy	/body/checkInventoryResponse/inventoryItem/status

\_\_\_ a. Copy quantity to target /context/transient/item/inventoryQuantity from source /body/checkInventoryResponse/inventoryItem/inStockQuantity

- 1) Ensure that the **SaveInventoryInfo** primitive is highlighted and then select **Properties → Details**
- 2) Under Message Elements table, click the **Add...** button. The Add/Edit window is opened
- 3) Define Target:
  - a) Click **Edit...** next to **Target**. The XPath Expression Builder wizard is opened

b) Expand **ServiceMessageObject** → **context** → **transient** → **item**

c) Drag **inventoryQuantity** and drop it under **XPath Expression** (or double-click on **inventoryQuantity**)



d) You should now see this expression under **XPath Expression**



e) Click **Finish** button. You are now back to Add/Edit window

4) For **Type**, select **copy** from the drop down list

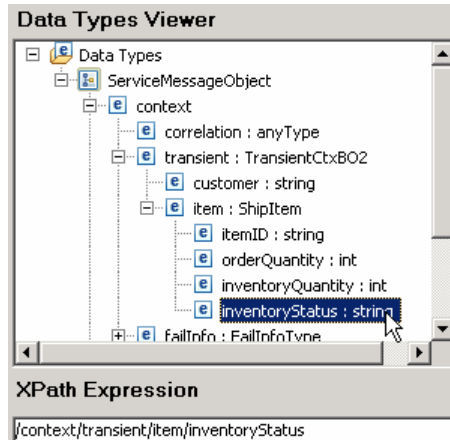
5) Define Value:

a) Click **Edit...** next to **Value**

b) Expand **ServiceMessageObject** → **body** → **checkInventoryResponse** → **inventoryItem**



d) You should now see this expression under **XPath Expression**



e) Click **Finish** button. You are now back to Add/Edit window

3) For **Type**, select **copy** from the drop down list

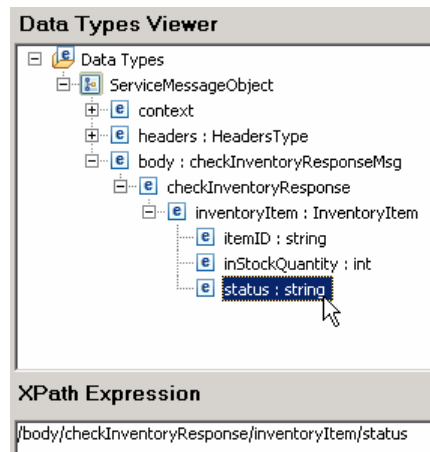
4) Define Value:

a) Hit **Edit...** next to **Value**

b) Expand **ServiceMessageObject** → **body** → **checkInventoryResponse** → **inventoryItem**

c) Drag **status** and drop it under **XPath Expression** (or double-click on status)

d) You should now see this expression under **XPath Expression**



e) Click **Finish**

5) Your Add/Edit window should look like this:



6) Click **Finish**

7) You will see one more entry in the Message Elements table:

Message Elements:

Target	Type	Value	
/context/transient/item/inventoryQuantity	copy	/body/checkInventoryResponse/inventoryItem/inStockQuantity	Add...
/context/transient/item/inventoryStatus	copy	/body/checkInventoryResponse/inventoryItem/status	Edit...

\_\_\_ 12. Add a message element setter primitive and wire it into the flow following the SaveInventoryInfo primitive.

- **Display Name:** **AggregateItemInfo**
- **Wire :** **SaveInventoryInfo to AggregateItemInfo**



\_\_\_ a. From the **Palette**, select **Transformation → Message Element Setter** and then click on the canvas following the SaveInventoryInfo primitive

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

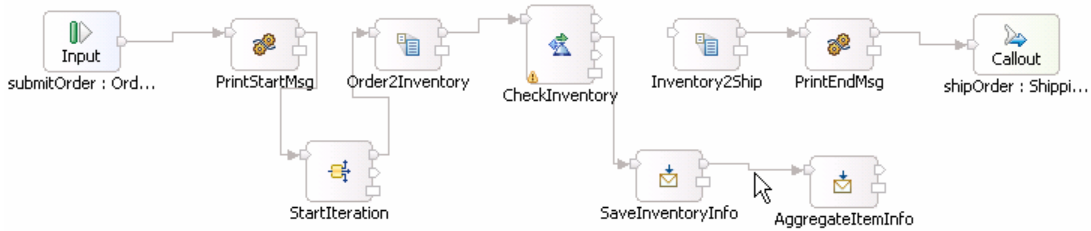
\_\_\_ c. Change the **Display name** to **AggregateItemInfo**

The screenshot shows the IBM WebSphere ESB Designer interface. On the left is the 'Palette' with various tool categories. The main canvas displays a message flow with several primitives: 'Ord...', 'PrintStartMsg', 'Order2Inventory', 'CheckInventory', 'Inventory2Ship', 'PrintEndMsg', 'shipOrder : Shippi..', 'StartIteration', 'SaveInventoryInfo', and 'MessageElementS...'. The 'MessageElementS...' primitive is highlighted with a red box. Below the canvas, the 'Properties' tab is active, showing the 'Description' section. The 'Display name' field is set to 'AggregateItemInfo' and is highlighted with a red box. The 'Name' field is set to 'MessageElementSetter2'.

\_\_\_ d. Add a connection from **SaveInventoryInfo** primitive to **AggregateItemInfo** primitive

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

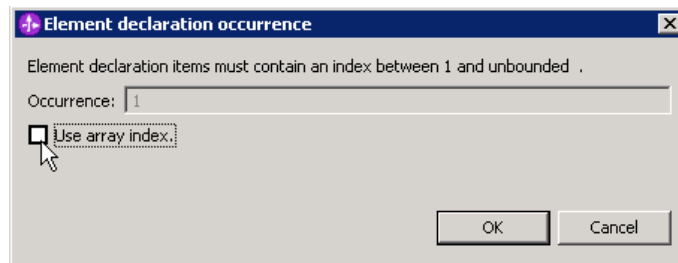
\_\_\_ e. You should now see a connection from **SaveInventoryInfo** to **AggregateItemInfo** primitive;



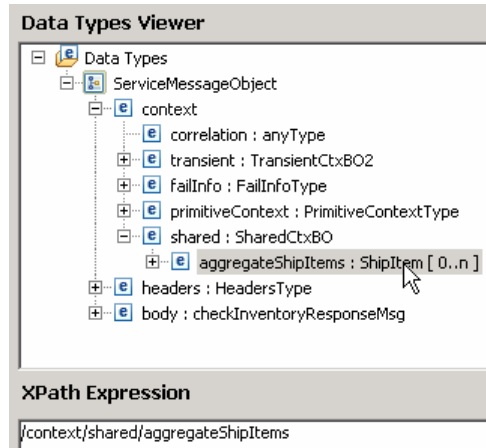
- \_\_\_ 13. Configure the **AggregateItemInfo** primitive to take the item information in the transient context and append it to the end of the array of items in the shared context
- Set properties of message element setter to append to the end of the array (make sure you set the **Type = append**)

Target array location in SMO	Value (source) location in SMO of value to append to array
/context/shared/aggregateShipItems	/context/transient/item

- -----
- \_\_\_ a. Ensure that the **AggregateItemInfo** primitive is highlighted and then select **Properties → Details**
- \_\_\_ b. Under Message Elements table, click the **Add...** button. The Add/Edit window is opened
- \_\_\_ c. Define Target:
- 1) Click **Edit...** next to **Target**. The XPath Expression Builder wizard is opened
  - 2) Expand **ServiceMessageObject → context → shared**
  - 3) Drag **aggregateShipItems** and drop it under **XPath Expression** (or double-click on aggregateShipItems)
  - 4) Element declaration occurrence window will pop-up. Uncheck the box for '**Use array index**' and then click **OK**



5) You should now see this expression under **XPath Expression**

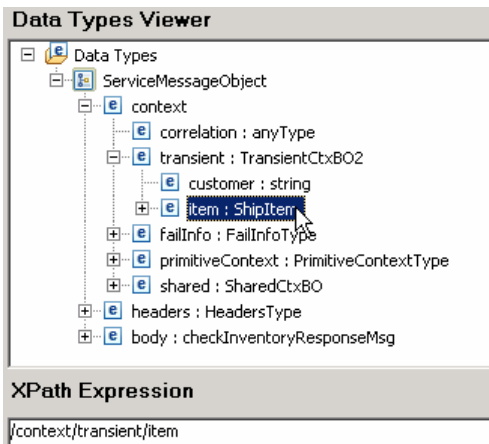


6) Click **Finish** button. You are now back to Add/Edit window

\_\_\_ d. For **Type**, select **append** from the drop down list

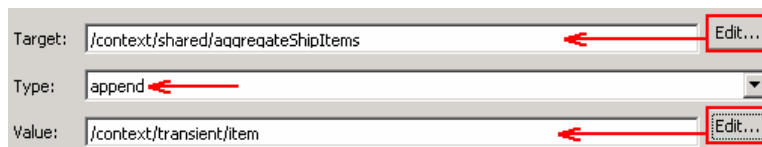
\_\_\_ e. Define Value:

- 1) Click **Edit...** next to **Value**
- 2) Expand **ServiceMessageObject** → **context** → **transient**
- 3) Drag **item** and drop it under **XPath Expression** (or double-click on item)
- 4) You should now see this expression under **XPath Expression**



a) Click **Finish**

5) Your Add/Edit window should look like this:



6) Click **Finish**



7) You will see this entry in the Message Elements table:

Message Elements:

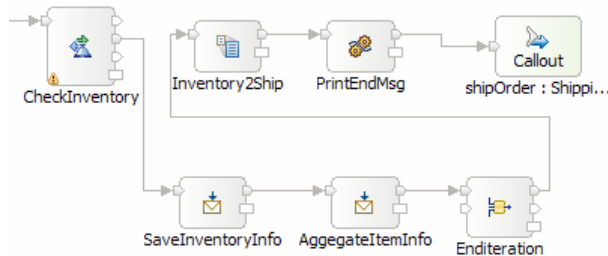
Target	Type	Value	Add...
/context/shared/aggregateShipItems	append	/context/transient/item	Edit...

14. Add a fan in primitive to the flow and wire it between the AggregateItemInfo and Inventory2Ship primitives.

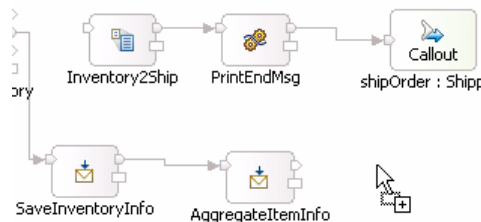
- **Display Name** : EndIteration
- **Associated Fan Out**: StartIteration

**NOTE:** When dropping the fan in primitive on the canvas a Fan Out Primitive Selection dialog is presented. This is where you select StartIteration to be the associated fan out. Do not create a new fan out.

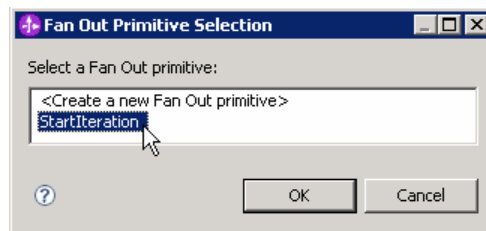
- **Wire:** AggregateItemInfo to EndIteration(in terminal)
- **Wire:** EndIteration(out terminal) to Inventory2Ship



a. From the **Palette**, select **Routing** → **Fan In** and then click on the canvas as shown below (to the right of AggregateItemInfo primitive)



1) A 'Fan Out Primitive Selection' pop-up window is opened. Select **StartIteration** under 'Select a Fan Out primitive':



2) Click **OK**

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

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

\_\_ d. Add a connection from **AggregateItemInfo** to **EndIteration**

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

---

**NOTE:** There are two input terminals for a fan in primitive, so make sure you are wiring to the **in** terminal and not from the **stop** terminal.

---

3) You should now see a connection from **AggregateItemInfo** to **EndIteration** primitive



\_\_ e. Similarly, add a connection from **EndIteration** primitive to **Inventory2Ship** primitive

- 1) Hover your mouse over **out** terminal of **EndIteration** to get the 'Add a connection to an input terminal'

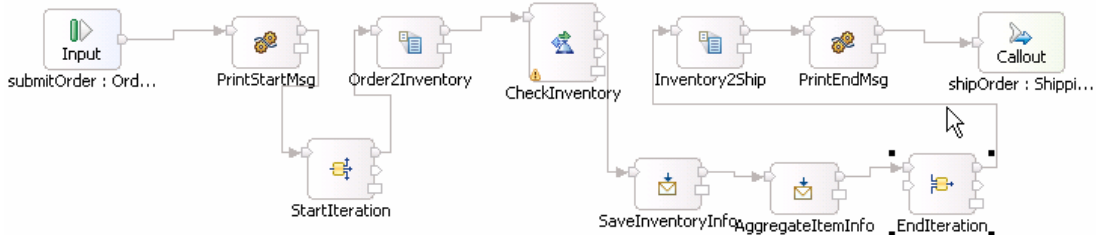
---

**NOTE:** There are two output terminals for a fan in primitive, so make sure you are wiring from the **out** terminal and not from the **incomplete** terminal.

---

2) Click on the orange bubble and then click on **Inventory2Ship** primitive

\_\_ f. You should now see a connection from **EndIteration** primitive to **Inventory2Ship** primitive:



\_\_\_\_\_ 15. Configure the **EndIteration** primitive to complete when all the items in the associated **StartIteration** fan out have been iterated through

- **Select 'the associated Fan Out primitive has iterated through all messages'**

**Fan In**

Fire output terminal when

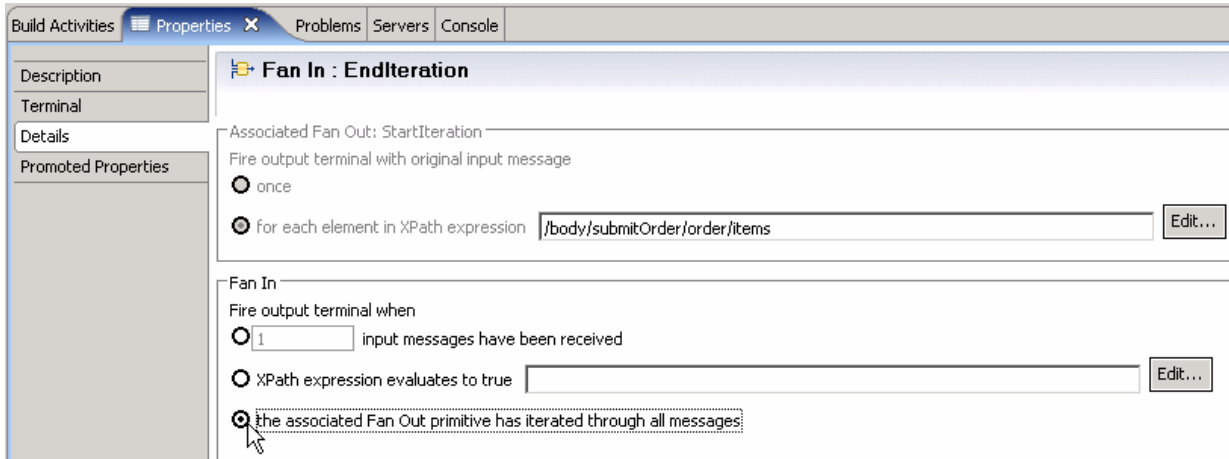
1 input messages have been received

XPath expression evaluates to true

the associated Fan Out primitive has iterated through all messages

\_\_ a. Ensure that **EndIteration** primitive is highlighted, and select **Properties →Details**

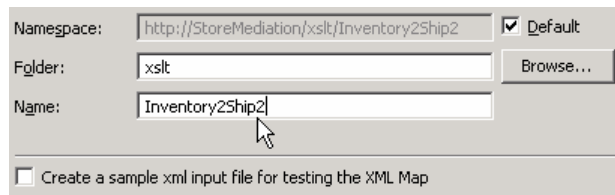
- \_\_\_ b. Select the radio button for **'the associated Fan Out primitive has iterated through all messages'**



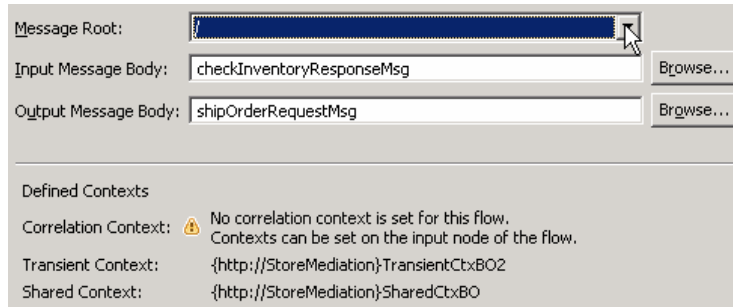
- \_\_\_ 16. In the Inventory2Ship XSL transformation primitive create a new map named Inventory2Ship2 (this will replace the existing Inventory2Ship1 map currently configured for the primitive).

- **Map Name** : **Inventory2Ship2**
- **Message Root** : **/**
- **Input Message Body** : **checkInventoryResponseMsg**
- **Output Message Body**: **shipOrderRequestMsg**
- -----

- \_\_\_ a. Select **Inventory2Ship** 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 **Inventory2Ship2**



- \_\_\_ d. Click **Next**
- \_\_\_ e. From Specify Message Types screen
  - 1) For **Message Root**, select **'/'** from the drop down list
  - 2) For **Input Message Body**, accept the default selection: **checkInventoryResponseMsg**
  - 3) For **Output Message Body**, accept the default selection: **shipOrderRequestMsg**



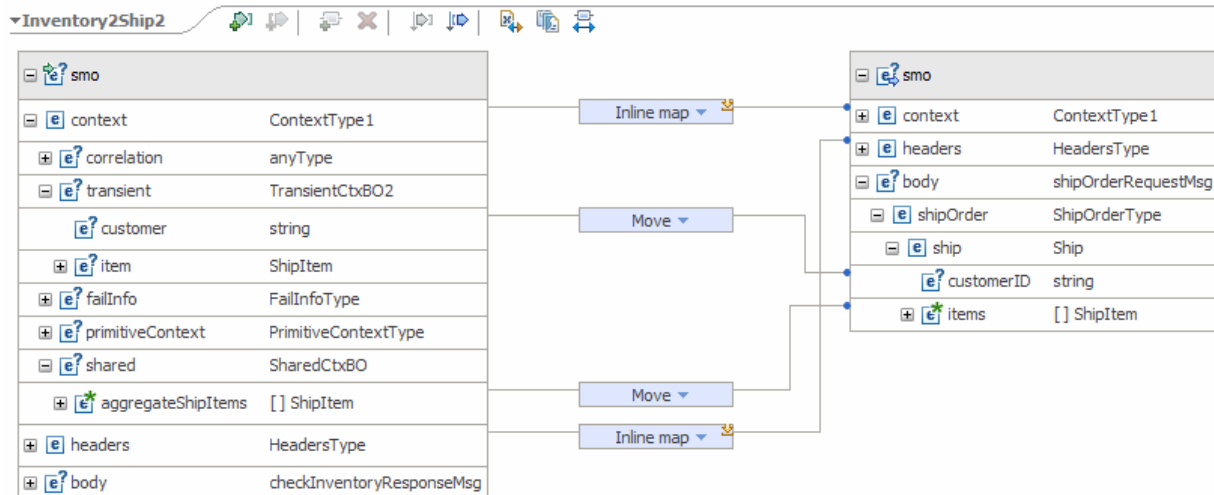
\_\_ f. Click **Finish**. An Inventory2Ship2.map file is opened in the XML Mapping editor

\_\_\_\_ 17. Define the mapping for the Inventory2Ship2 map. The mapping must address (1) moving fields between the source and target SMOs that will not change, (2) setting up the target message body needed to call the shipping service. Because the shared context had been designed with the same Shipltem array definition as is needed by target message body, the entire array can be handled with a simple move transformation

- Use toolbar icon (↔) "Map source to target based on name and types"
- At the SMO level (top level map) add these move transforms

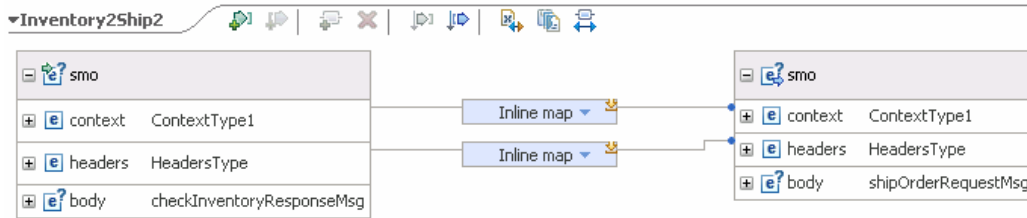
**NOTE:** Unlike previous cases of arrays, where you moved a specifically indexed array element, in this case the aggregateShipltems move to shipltems is for the entire array. Therefore, there is no need to specify the cardinality in the properties of the move transform.

Source (left side)	Target (right side)
context/transient/customer	body/shipOrder/ship/customerID
context/shared/aggregateShipltems [ ]	body/shipOrder/ship/items [ ]



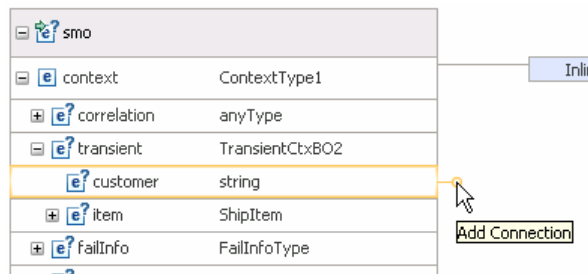
\_\_\_\_ a. Click on 'Map source to target based on name and types' icon (↔)

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

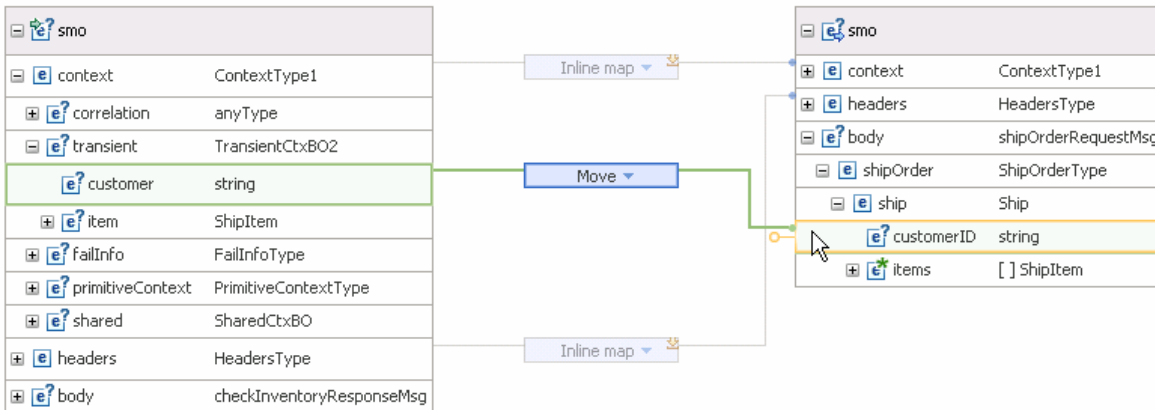


\_\_ c. Move context/transient/customer to body/shipOrder/ship/customerID

- 1) In the left smo, expand **context** → **transient**
- 2) In the right smo, expand **body** → **shipOrder** → **ship**
- 3) In the left smo, hover your mouse over **customer** and click on **Add Connection**:

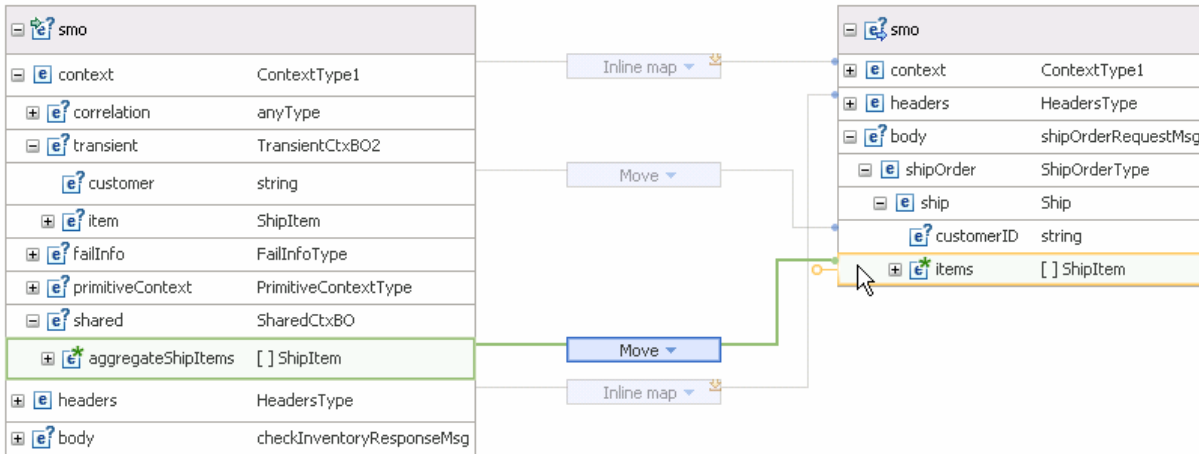


- 4) Now click on **customerID** in the right smo and you should see a new connection from customer to customerID:



\_\_ d. Next, move context/shared/aggregateShipItems[] to body/shipOrder/ship/items[]

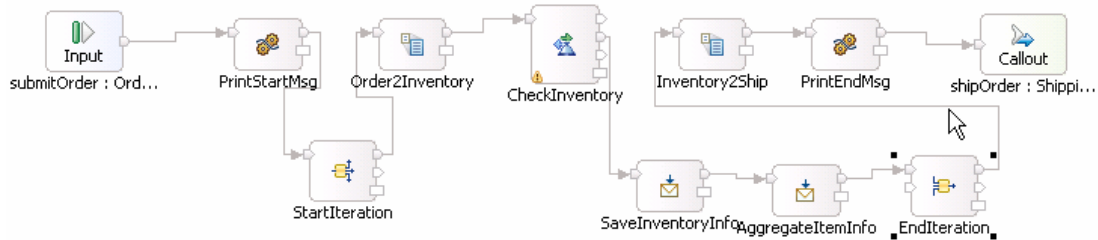
- 1) In the left smo, expand **context** → **shared**
- 2) In the right smo, expand **body** → **shipOrder** → **ship**
- 3) In the left smo, hover your mouse over **aggregateShipItems** and click on **Add Connection**
- 4) Now click on **items** in the right smo and you should see a new connection from aggregateShipItems to items:



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

\_\_\_ f. Click on **X** to close Inventory2Ship2.map file

\_\_\_ 18. The flow is now completed and should look something similar to this.

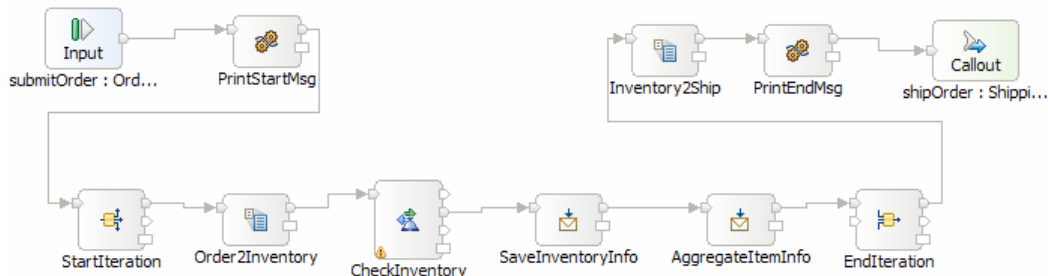


\_\_\_ 19. Optionally, if you want to rearrange the layout of the flow in the editor you have two options. It can be done by the editor using the **Layout Contents** menu option or you can move individual primitives to an arrangement that makes sense when considering the function of the flow.

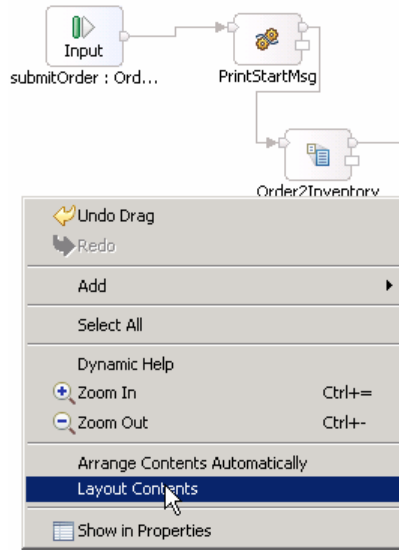
• **Layout done using the Layout Contents menu option**



• **Possible layout done by moving individual primitives. This is showing the iterative loop in the lower row of the flow.**



\_\_\_ a. Right-click on the canvas of mediation flow and select **Layout Contents** from the pop-up menu



\_\_\_ 20. Check that all the artifacts have been saved.

- -----

\_\_\_ a. Look at the tabs for the various artifact editors. Any tab with an asterisk ( \* ) before the name needs to be saved:

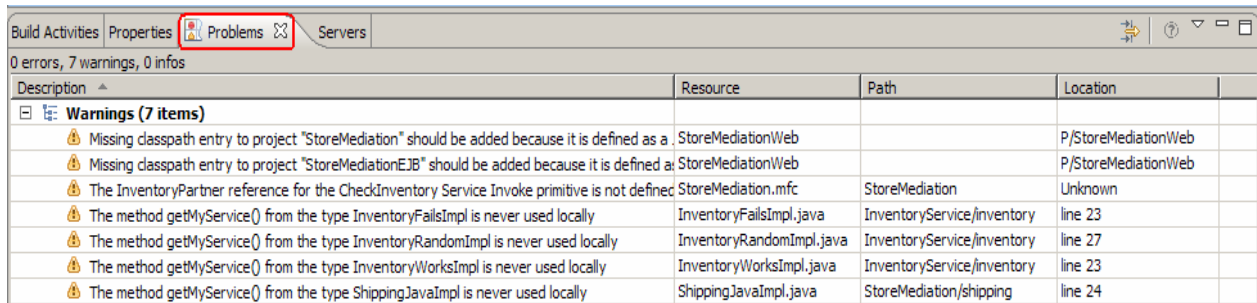


\_\_\_ b. For each tab with an asterisk ( \* ), click on the tab to give it focus and from the menu select **File** → **Save** (or **Ctrl + S** from your keyboard) to save your changes.

\_\_\_ 21. 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:



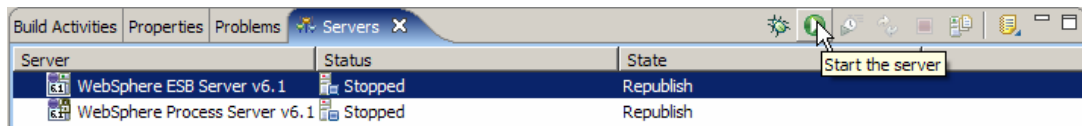
## Part 3: Test the splitting/aggregating mediation

**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 click 'Start the server' icon (🟢) from the toolbar



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

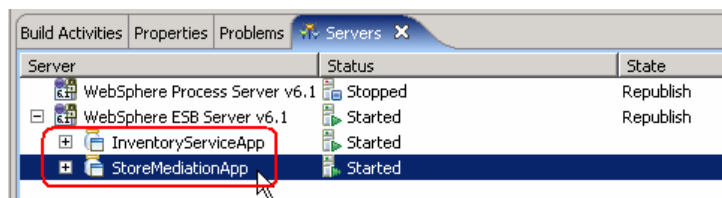


**NOTE:** Depending upon the preferences you have specified for the **Console** view, the **Console** view might grab the focus from the **Servers** view. If this is the case, the status in the lower right should indicate when the server startup is complete, at which time you can switch back to the **Servers**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

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

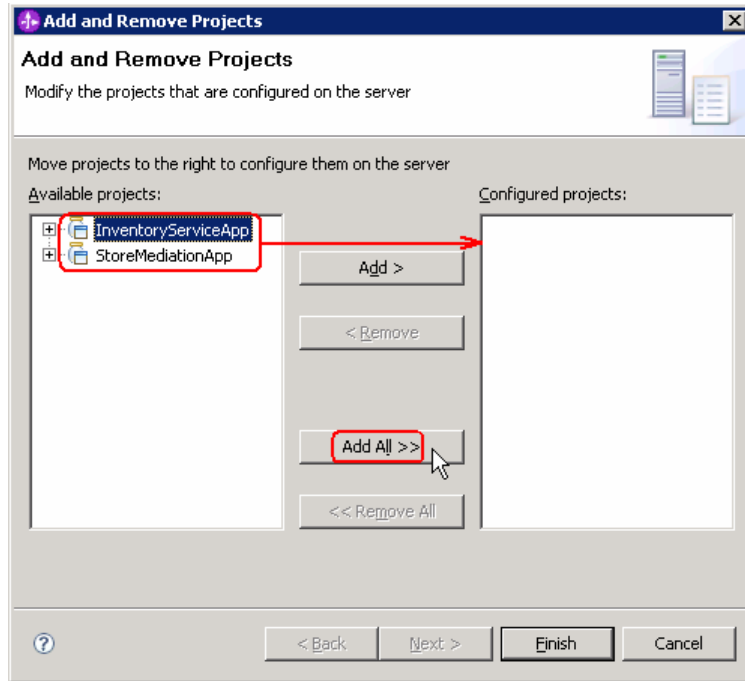


- \_\_\_ b. If the applications are **not deployed yet**, add both projects to the server following these steps:

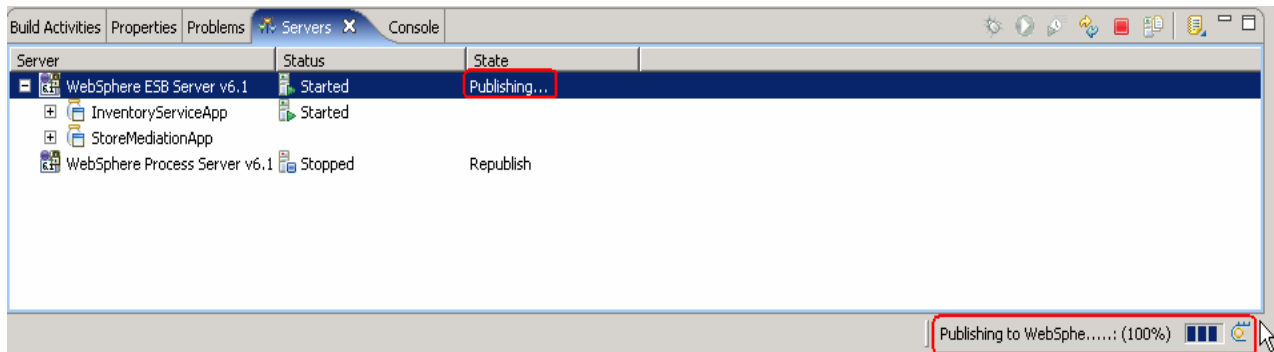
- 1) Right-click on **WebSphere ESB Server v6.1** under the Servers view and select **Add and remove projects...** from the context menu



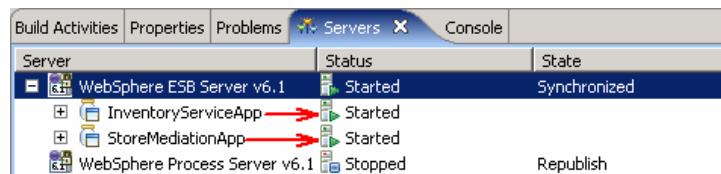
- 2) In the Add and Remove Projects window, click **Add All >>** to add InventoryServiceApp and StoreMediationApp to the Configured projects panel



- 3) The projects will now be moved to Configured projects. Click **Finish**
- 4) Wait while the projects are being published to the server

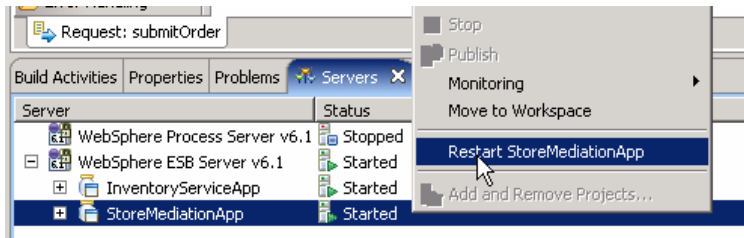


- 5) Once the publishing is done, from the Servers view, expand **WebSphere ESB Server v6.1** and you should see the 2 applications started as following:



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

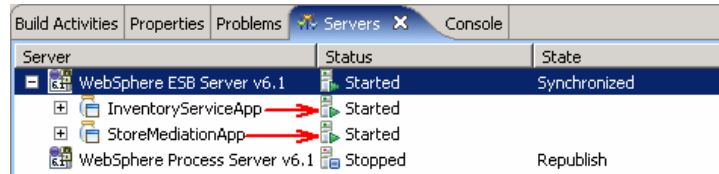
- 1) Right click on StoreMediationApp to get the pop-up menu and select **Restart StoreMediationApp**




- 2) Wait while the application stops and restarts.



- 3) Once it has restarted, you should see the two applications started:



\_\_\_ 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
- -----

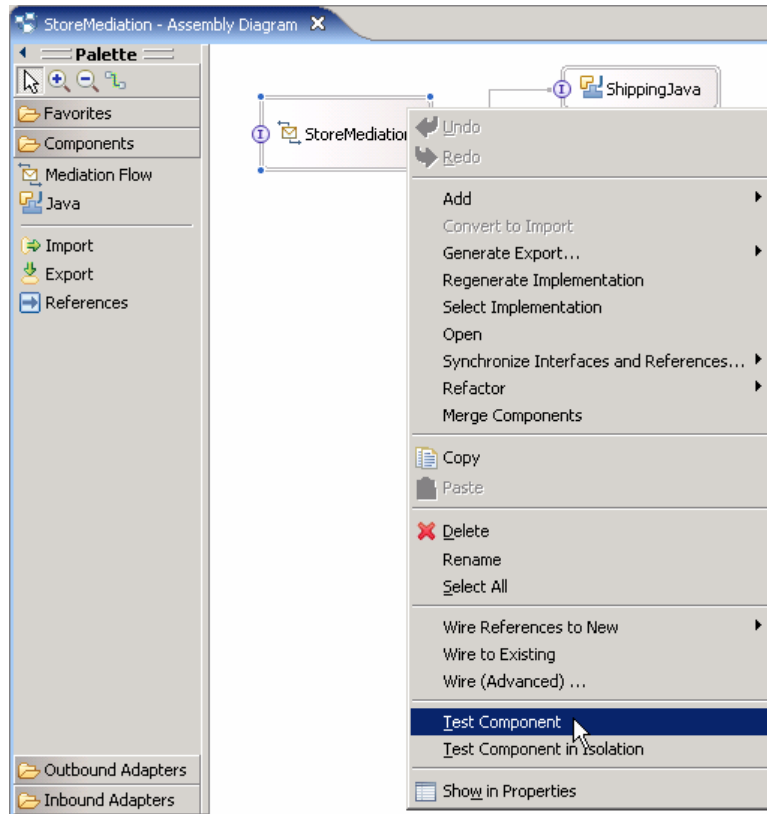
\_\_\_ a. Look at the tabs to see if StoreMediation\_Test is still open as shown in this screen capture.  
**Follow the appropriate instructions in step b or step c**



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

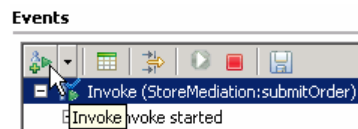
- 1) In the Business Integration window, expand **StoreMediation** and double-click on **Assembly Diagram** to open it in Assembly editor

- 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.



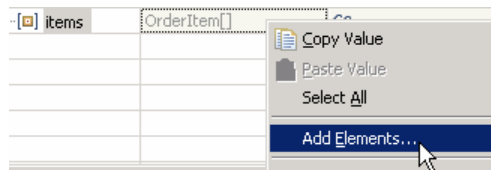
- \_\_\_ 4. Initialize the test data

- 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`

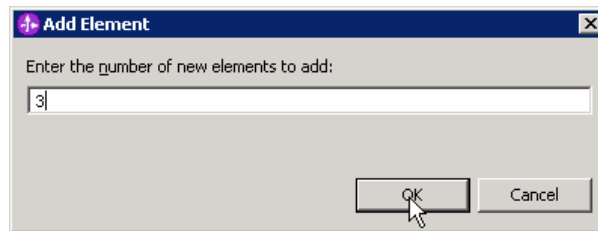
Name	Type	Value
order	Order	✓
customerID	string	✓ cust123
items	OrderItem[]	6
items[0]	OrderItem	✓
itemID	string	✓ item001
quantity	int	✓ 3
items[1]	OrderItem	✓
itemID	string	✓ item009
quantity	int	✓ 5
items[2]	OrderItem	✓
itemID	string	✓ item002
quantity	int	✓ 15

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

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



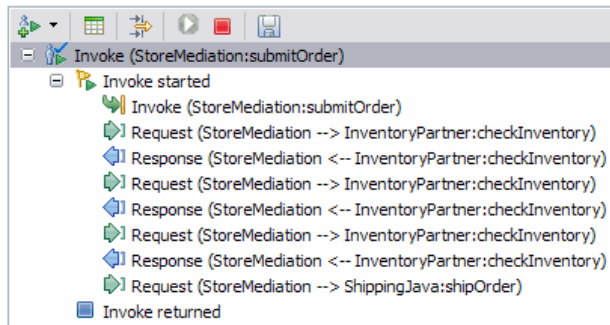
3) Enter '3' in the Add Element window:



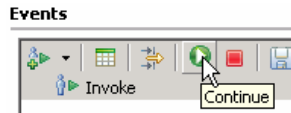
- 4) Click **OK**
- 5) Enter values for items[0]:
  - a) For **itemID**, click under Value and enter **item001**
  - b) For **quantity**, click under Value and enter **3**
- 6) Enter values for items[1]:
  - a) For **itemID**, click under Value and enter **item009**
  - b) For **quantity**, click under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

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

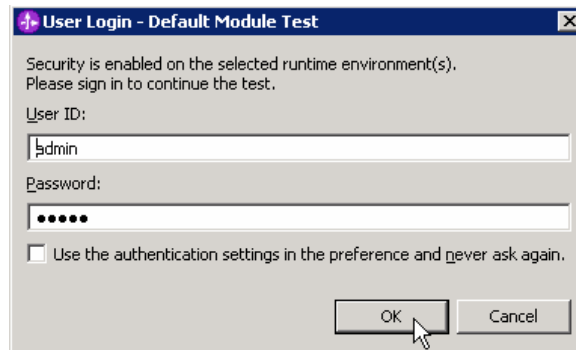
5. Run the test by hitting the Continue icon (🟢). Results should show three calls to the Inventory Service, one for each item found on input, as shown here:



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

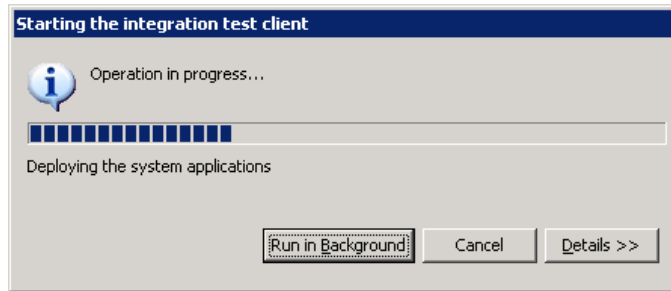


b. If presented with the **User Login** dialog, enter the **User ID** and **Password**, which by default is normally set to **admin** and **admin**. Optionally, you can check the 'Use the authentication settings in the preference and never ask again' check box to prevent this dialog from being displayed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

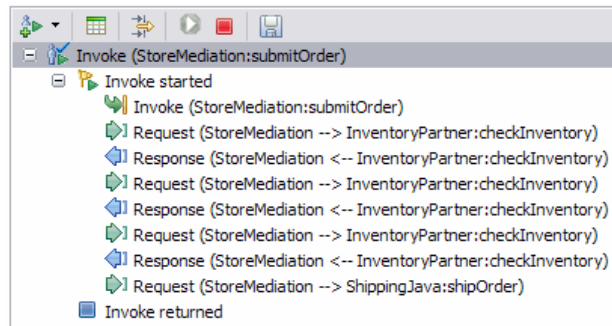


c. Click **OK**

\_\_\_ d. Wait until the integration test client starts



\_\_\_ e. You should see these results:



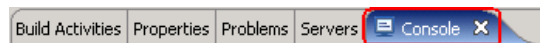
6. Switch to the Console view and examine the output, which should look similar to this screen capture. You can see the three invocations of the Inventory Service, one for each item, and the aggregated result sent to the Shipping Service containing all three items.

```

|O *****
O ***** START mediation flow *****
O ***
O ***** InvWorks - returning InventoryItem for itemID = item001
O ***** InvWorks - returning InventoryItem for itemID = item009
O ***** InvWorks - returning InventoryItem 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@10261026
O Value:
O   customerID = cust123
O   items = ShipItem[3]
O     items[0] = <ShipItem@10d010d0>
O       itemID = item001
O       orderQuantity = 3
O       inventoryQuantity = 5
O       inventoryStatus = OK - but stock is running low
O     items[1] = <ShipItem@10ea10ea>
O       itemID = item009
O       orderQuantity = 5
O       inventoryQuantity = 45
O       inventoryStatus = OK - sufficient stock levels
O     items[2] = <ShipItem@11041104>
O       itemID = item002
O       orderQuantity = 15
O       inventoryQuantity = 10
O       inventoryStatus = Backorder - insufficient stock to fill order
O
O -- Ship object dump ends -----
O -----

```

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



7. Being able to examine and understand the Console output is important throughout this series of lab exercises. To that end, the output above is explained here:

- Output produced by the PrintStartMsg primitive in the mediation flow:

```

O *****
O ***** START mediation flow *****
O ***

```

- Output produced by the inventory service. The "InvWorks" indicates which inventory service implementation was used (there are other implementations which are used in subsequent lab exercises examining service invoke retry capabilities). Notice that the inventory service has been called once for each item in the list. This illustrates the iterative processing that occurred in the flow.

```

O ***** InvWorks - returning InventoryItem for itemID = item001
O ***** InvWorks - returning InventoryItem for itemID = item009
O ***** InvWorks - returning InventoryItem for itemID = item002

```

- Output produced by the PrintEndMsg primitive in the mediation flow:

```

O ***
O ***** END mediation flow *****
O *****

```

- Output produced by the shipping service. This is a dump of the input business object, of type Ship. Notice that within Ship, each ShipItem is composed of information from the Order and additional data from the inventory service. This illustrates that message splitting (iterative) processing took place, thus enabling message augmentation for each item. Also notice that the augmented information for each item has been built up into an array, illustrating message aggregation.

```

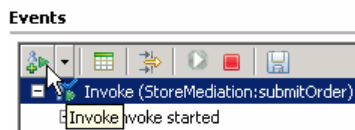
O -----
O -- Ship object dump begins -----
O
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@f780f78
O Value:
O   customerID = cust123
O   items = ShipItem[3]
O     items[0] = <ShipItem@fb00fb0>
O       itemID = item001
O       orderQuantity = 3
O       inventoryQuantity = 5
O       inventoryStatus = OK - but stock is running low
O     items[1] = <ShipItem@fca0fca>
O       itemID = item009
O       orderQuantity = 5
O       inventoryQuantity = 45
O       inventoryStatus = OK - sufficient stock levels
O     items[2] = <ShipItem@fe40fe4>
O       itemID = item002
O       orderQuantity = 15
O       inventoryQuantity = 10
O       inventoryStatus = Backorder - insufficient stock to fill order
O
O -- Ship object dump ends -----
O -----

```

8. If you want, 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

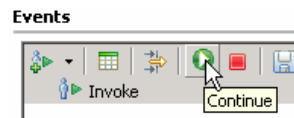
a. Click **Invoke** icon (  ) under Events panel





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

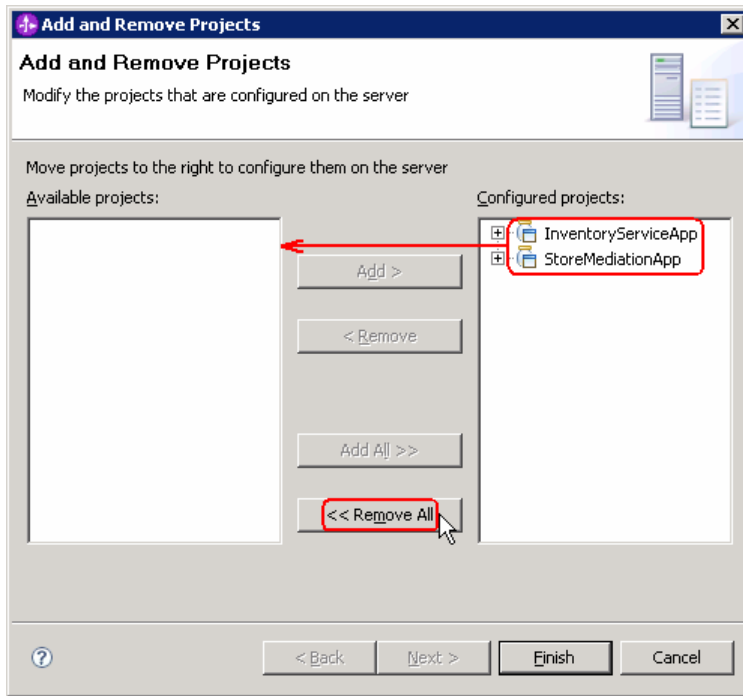
\_\_\_ c. Click **Continue** icon (  ) under Events panel



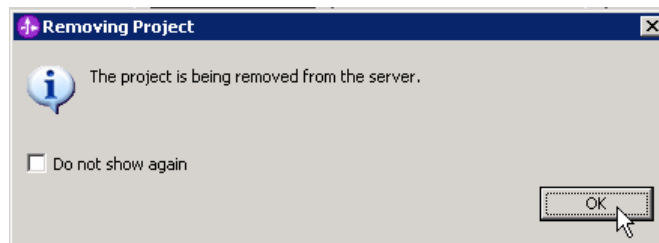
## Part 4: Clean up the environment if you will not proceed to the next lab

**Perform this part only if you are not continuing** to the service call retry lab (the third 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, hit **<< Remove All**

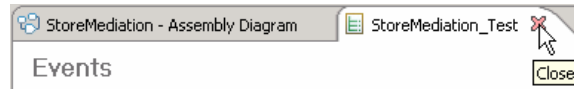


- \_\_\_ c. Click **Finish** after you see the applications moved to Available projects.
- \_\_\_ d. If displayed, click **OK** in 'Removing Project' window. Optionally, you can select the check box for 'Do not show again' not to be asked again when you remove projects later

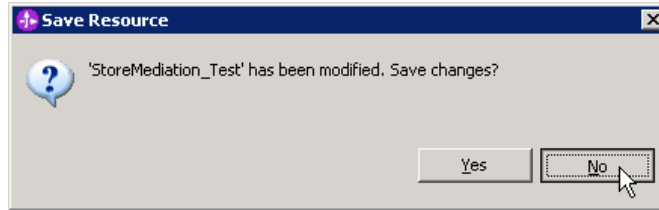


- \_\_\_ e. Wait until the application is removed from the server
- \_\_\_ 2. Close the StoreMediation\_test panel without saving
  - -----

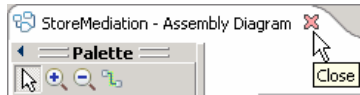
- o Click **X** on the StoreMediation\_Test tab



- o Click **No** from Save Resource window




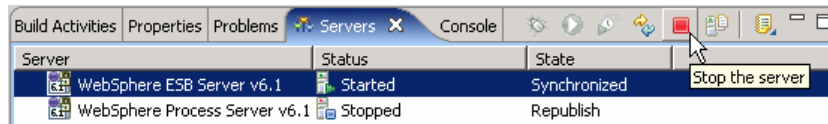
- o Click **X** on the StoreMediation – Assembly Diagram tab



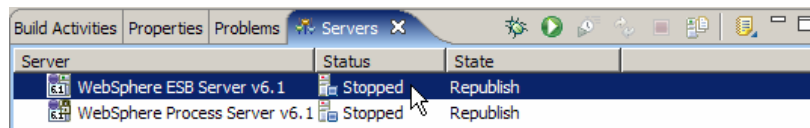
\_\_\_ 3. Stop the test server

- -----

- \_\_\_ 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 hit **'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 flow so that it performs splitting and aggregating, augmenting all elements of in an incoming array. This involved adding the fan out and fan in primitives, message element setter primitives and modifying the XSL transformation primitives.

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

## Solution instructions

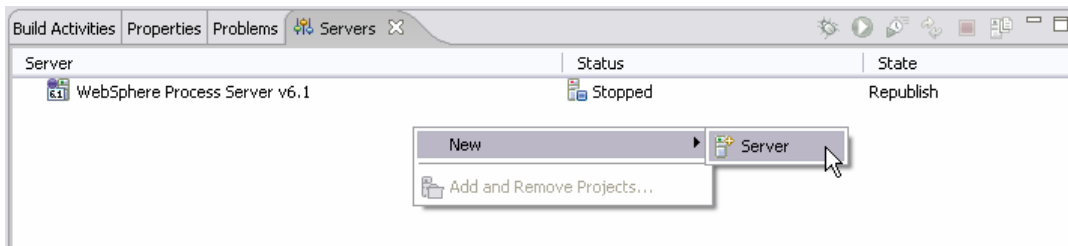
If you want to run this lab with a completed solution rather than authoring the flow yourself, follow these instructions:

- \_\_\_\_ 1. Follow **Part 1: Setting up the environment for the lab**, however use the project interchange file that contains the solution.
  - `<LAB_FILES>/PI3-AggregateSolution-RetryStart.zip`
- \_\_\_\_ 2. Skip to **Part 3: Test the splitting/aggregating mediation** and proceed through the rest of the lab

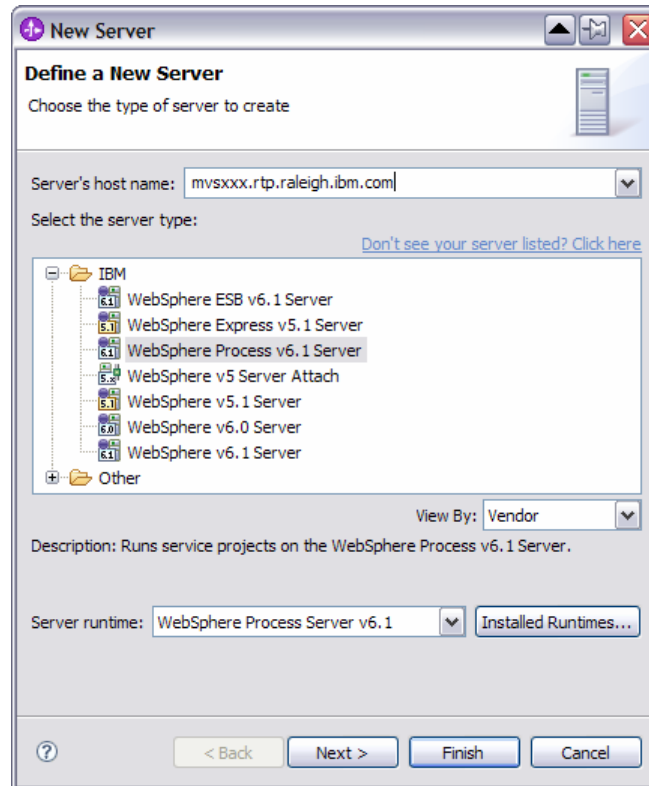
## Task: Adding remote server to the WebSphere Integration Developer test environment

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.

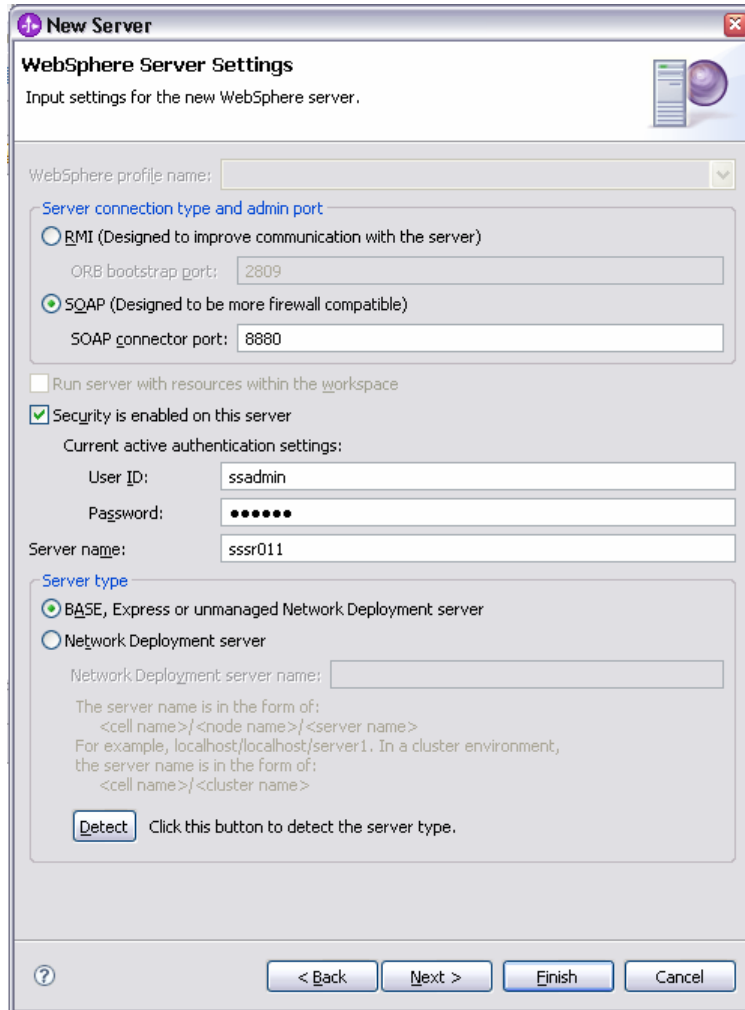


- \_\_\_ c. In the New Server dialog, specify the remote server's host name, <HOSTNAME>.
- \_\_\_ d. Ensure that the appropriate server type, 'WebSphere Process v6.1 Server' or 'WebSphere ESB v6.1 Server', is highlighted in the server type list

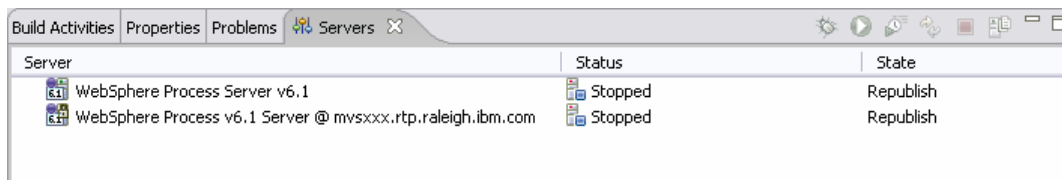


- \_\_\_ e. Click **Next**.

- \_\_\_ 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
```