

IBM WEBSPHERE 7.0 – LAB EXERCISE

# WebSphere Enterprise Service Bus 7.0

## Augmentation, aggregation, and retry tutorial series

### 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 .....	43
Part 4: Clean up the environment if you will not proceed to the next lab.....	53
What you did in this exercise .....	55
Solution instructions .....	56

### 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 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 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 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 above 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

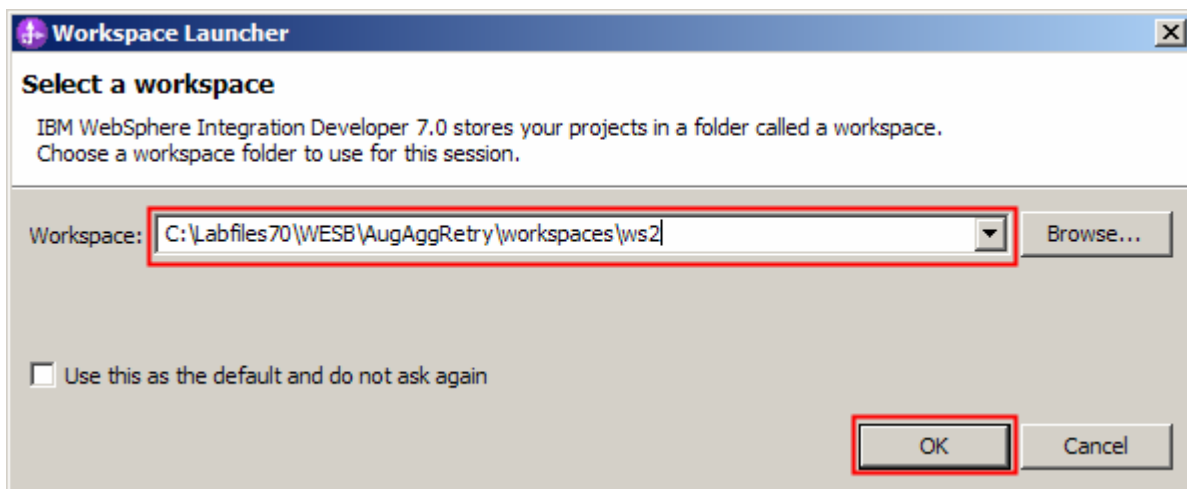
**What you will do in this part:** In this part you are getting the environment set up to do the lab. There are three different ways you might be approaching this which will dictate what you need to do.

**(1) Proceeding from Lab 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.

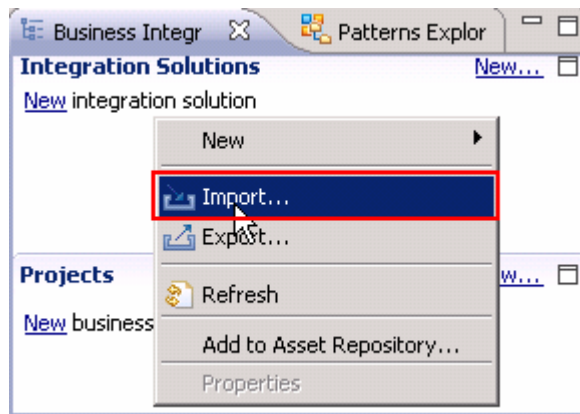
**(2) Restarting from Lab One** – You are continuing from Lab One which you did previously and therefore you did shut down the server and development environment. In this case, you will need to restart the development environment and server but will not need to import a project interchange to initialize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

**(3) Directly starting Lab Two** - You are starting this lab from scratch, regardless of whether you had previously completed Lab One.

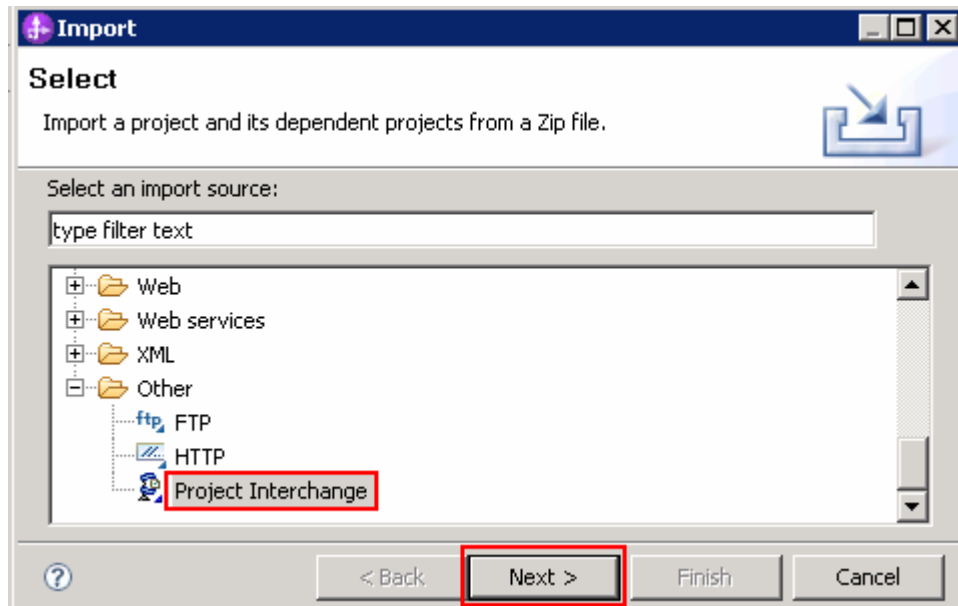
- \_\_\_ 1. If you are **proceeding from Lab One** there is nothing to do, skip directly to **Part 2 Authoring the mediation flow for splitting and aggregating**
  - -----
  
- \_\_\_ 2. Start WebSphere Integration Developer.
  - **Restarting from Lab One:** Use the same workspace used in Lab One
  -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



- \_\_\_ 3. If you are **restarting from Lab One** 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. Right-click inside **Business Integration View** (top left view in the Business Integration Perspective) and select **Import** from the pop-up menu

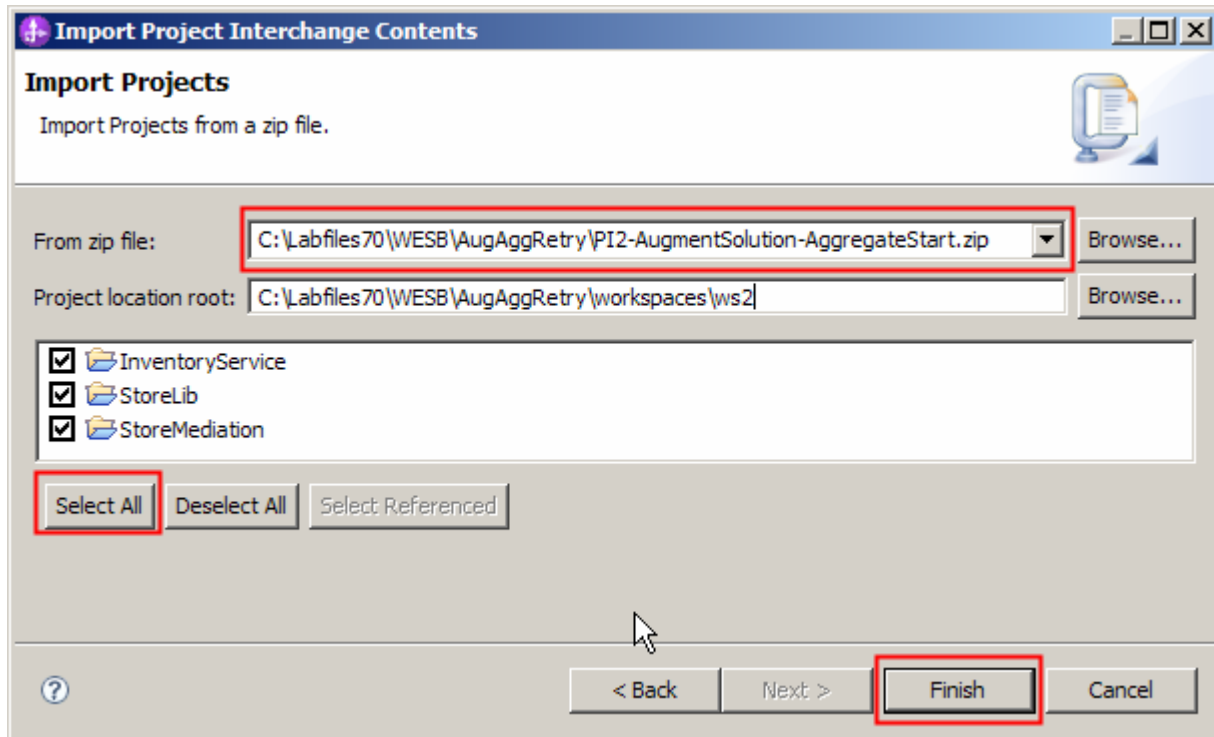


- \_\_\_ b. In the **Import** dialog, expand **Other** and select **Project Interchange** from the list

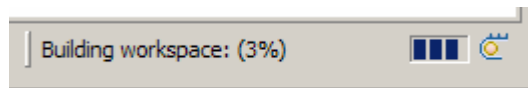


- \_\_\_ c. Click **Next**

- \_\_\_ d. In the next panel, click the **Browse** button for **From zip file** and navigate to <LAB\_FILES>/PI2-AugmentSolution-AggregateStart.zip and click **Open**



- \_\_\_ e. Click **Select All** to select all of 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 the WebSphere Integration Developer work bench



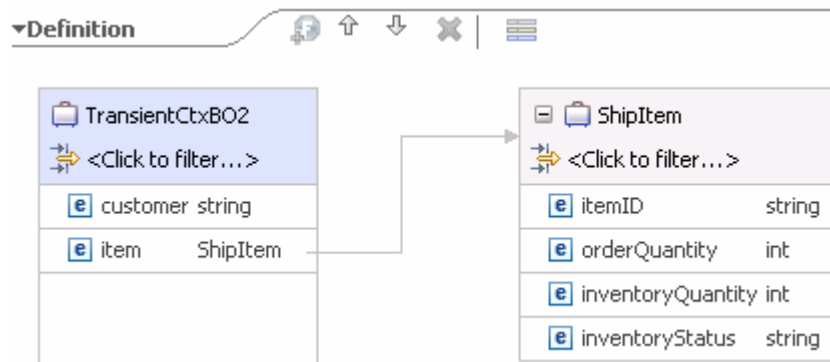
## Part 2: Authoring the mediation flow for splitting and aggregating

**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).

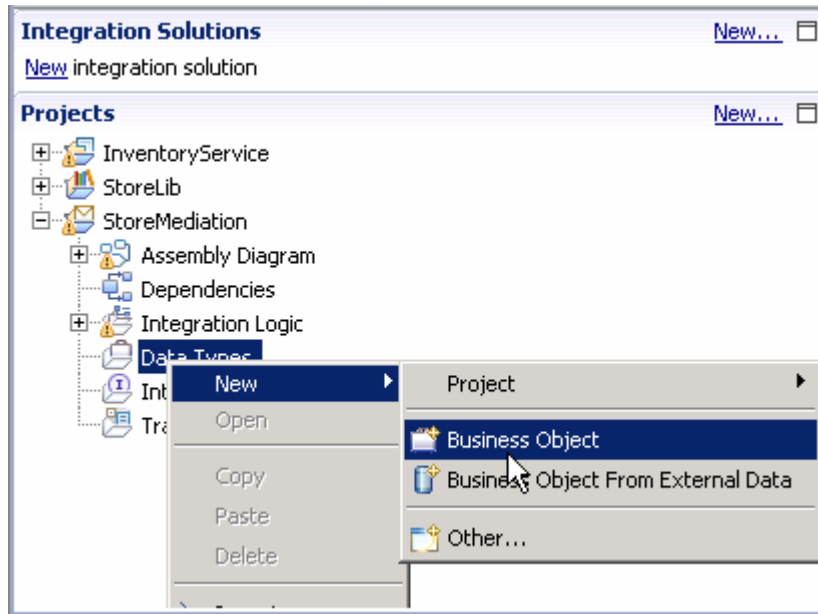
- **Module** : StoreMediation
- **Name** : TransientCtxBO2
- **Attributes:** customer string
- item ShipItem



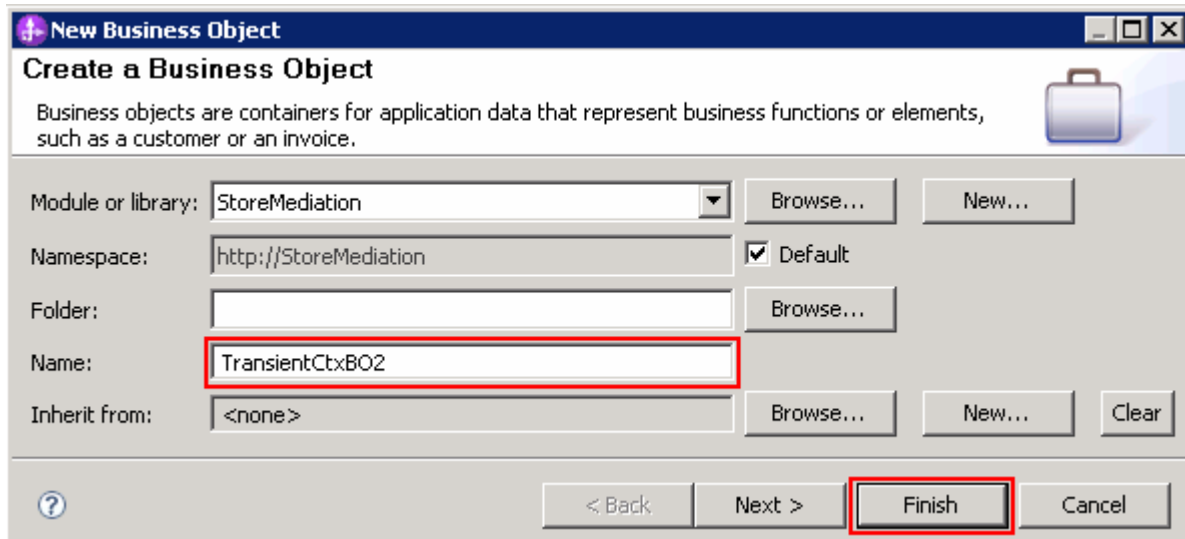
•



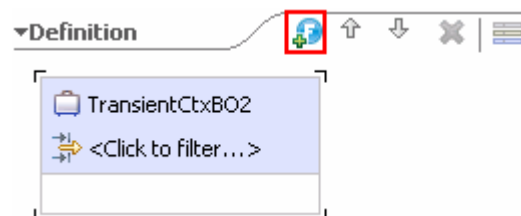
- \_\_\_ a. In the Business Integration view, expand **StoreMediation**, right click **Data Types** and then select **New → Business Object** from the pop-up menu



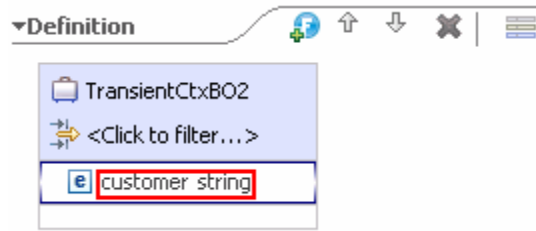
- \_\_\_ b. In the **New Business Object** dialog, provide the name **TransientCtxBO2**



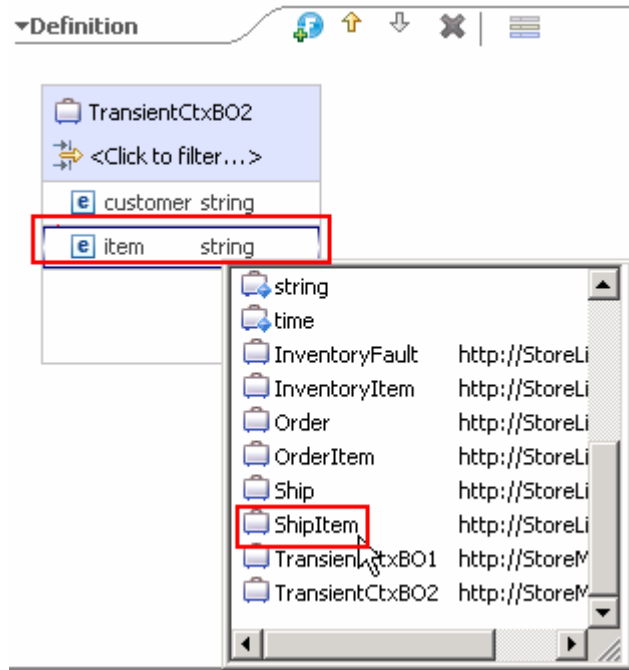
- \_\_\_ c. Click **Finish**. The business object editor opens
- \_\_\_ d. In the editor, click the **Add a field to a business object** icon



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

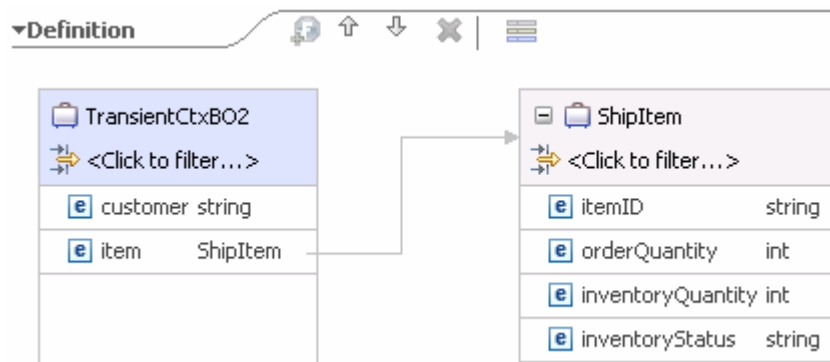


\_\_\_ f. Also add the field **item** of type **ShipItem**, using the drop down box to select the type



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

\_\_\_ h. Check that your business object looks like the following screen capture.



\_\_\_ i. Close the **TransientCtxBO2**, business object editor

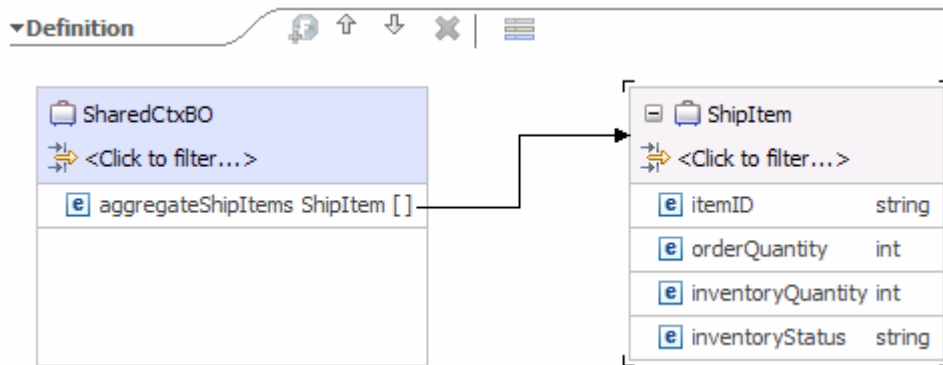
\_\_\_ 2. Create a new business object that is used as the shared context for the flow. The shared context defines the shared memory where the information for all of the items is aggregated during the iterative part of the flow.

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

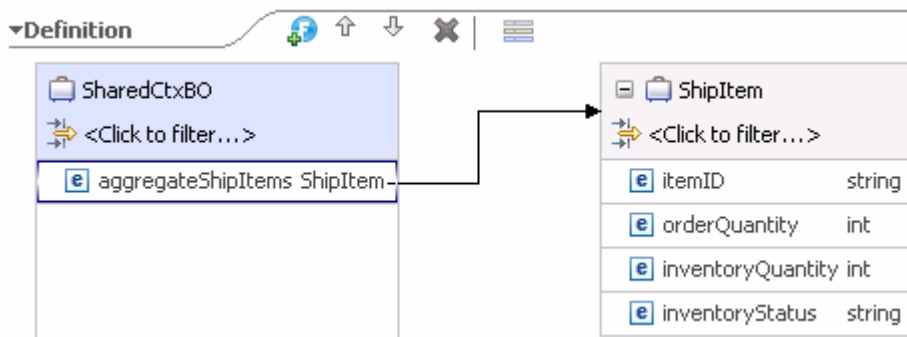
---

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

---

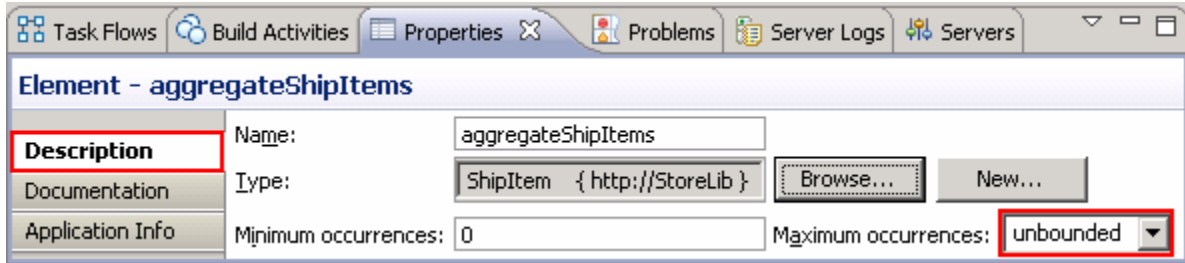


- \_\_\_ a. In the Business Integration view, expand **StoreMediation**, right click **Data Types** and then select **New → Business Object** from the pop-up menu
- \_\_\_ b. In the **New Business Object** dialog, provide the name **SharedCtxBO**
- \_\_\_ c. Click **Finish**
- \_\_\_ d. In the editor, 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. Make the **aggregateShipItems** attribute an array

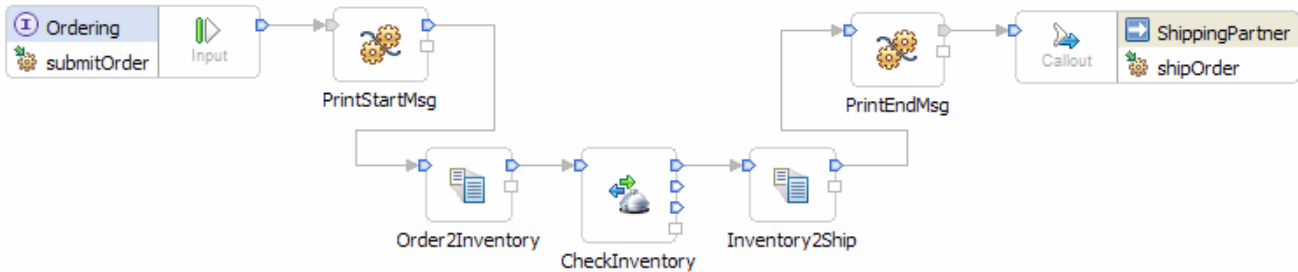
- 1) Ensure the **aggregateShipItems** attribute is selected, and then navigate to the **Description** panel of the **Properties** view.
- 2) Select **unbounded** for **Maximum occurrences**



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

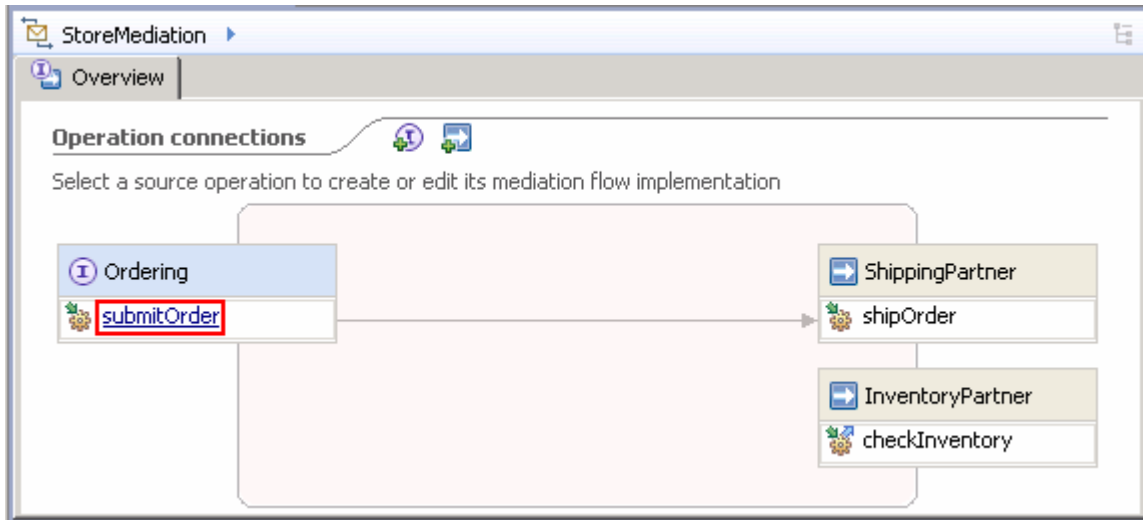
\_\_\_ h. Close the **SharedCtxBO**, business object editor

\_\_\_ 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).

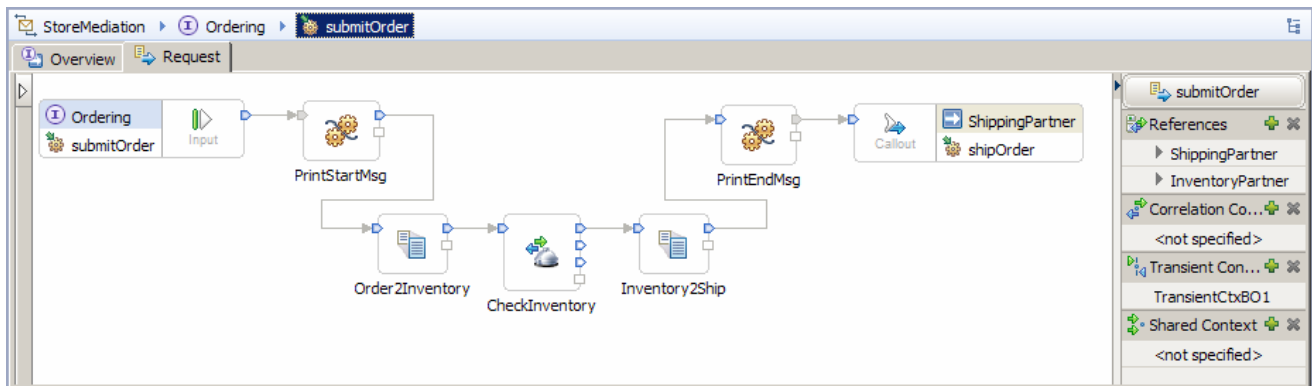


• -----

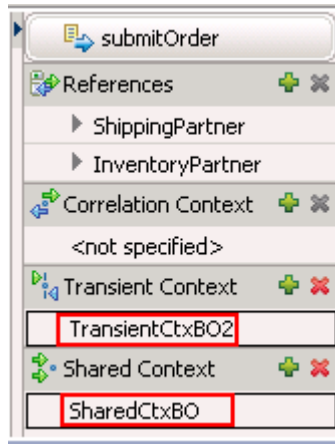
- \_\_\_ a. In the Business Integration view, expand **StoreMediation** → **Integration Logic** → **Mediation Flows** and then double-click **StoreMediation** to open it in the mediation flow editor



- \_\_\_ b. 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

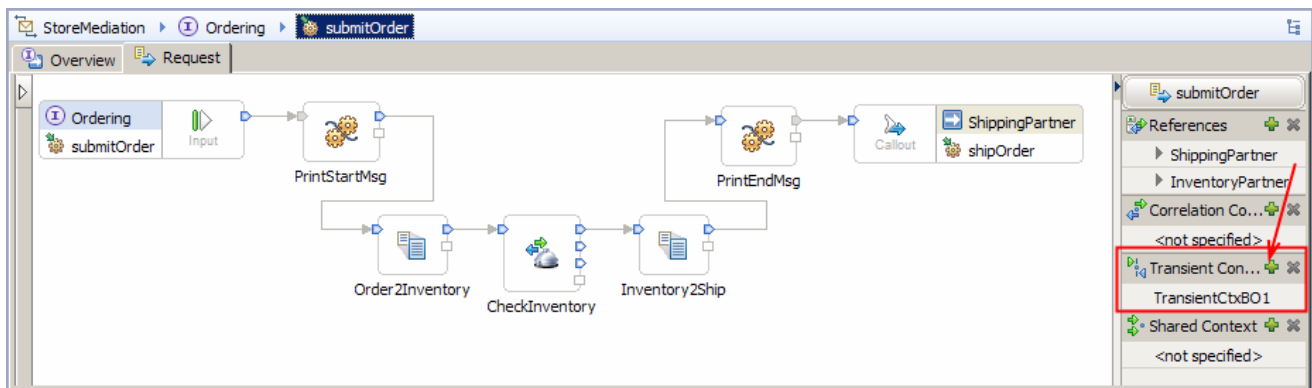


- \_\_\_ 4. Configure the transient and shared contexts that are used in the flow.
- **Transient context:** **TransientCtxBO2**
  - **Shared context** : **SharedCtxBO**

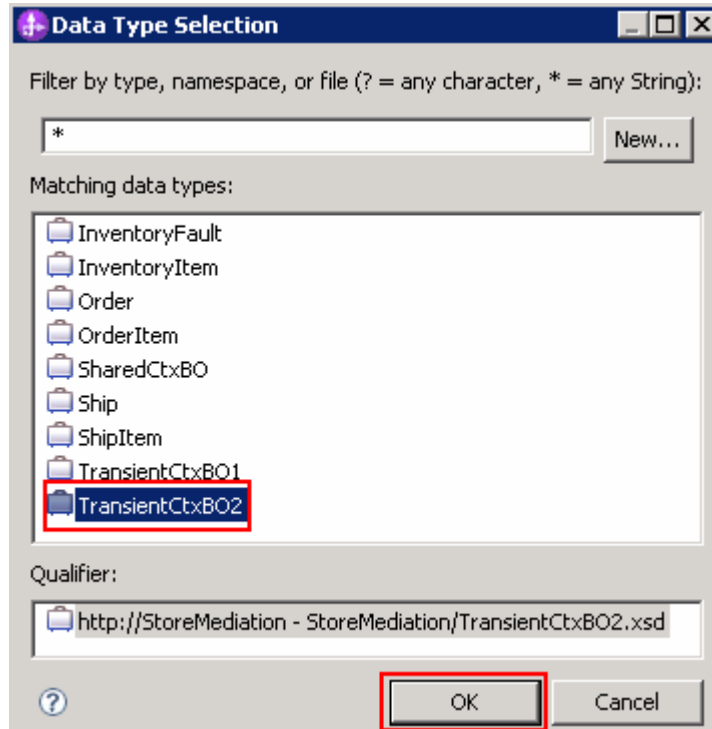


\_\_\_ a. In the right side of the mediation flow editor, configure the transient context for the flow to use **TransientCtxBO2** (replacing TransientCtxBO1 used in the previous lab).

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



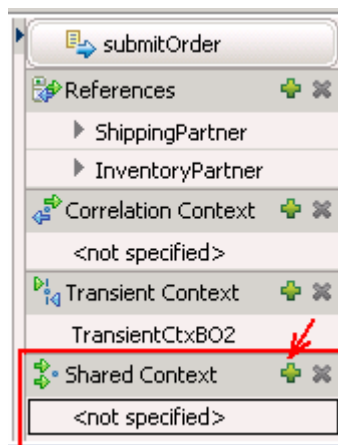
- 2) In the Data Type Selection window, scroll down to select **TransientCtxBO2** under **Matching data types:**



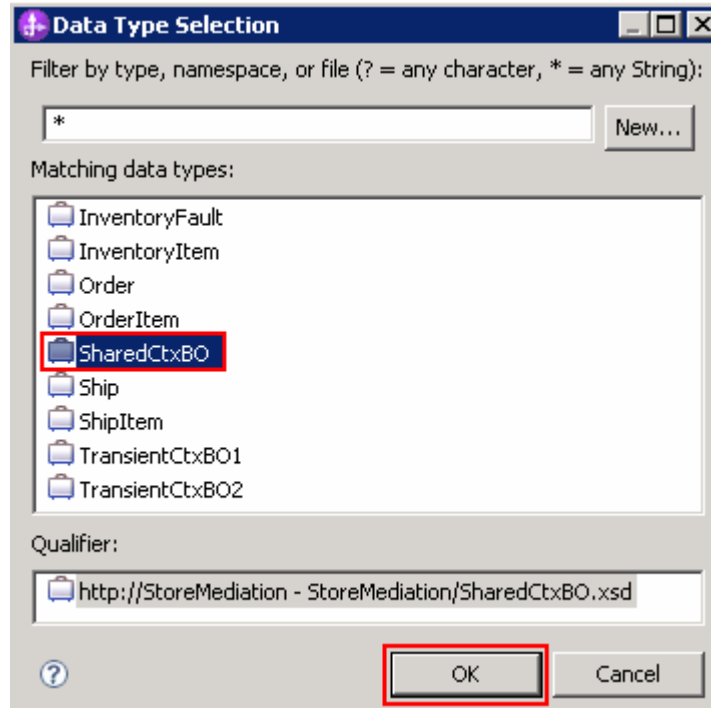
- 3) Click **OK**

\_\_\_ b. Now, configure the 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**:

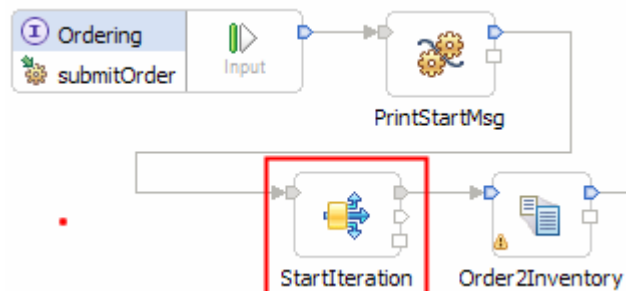


3) Click **OK**

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

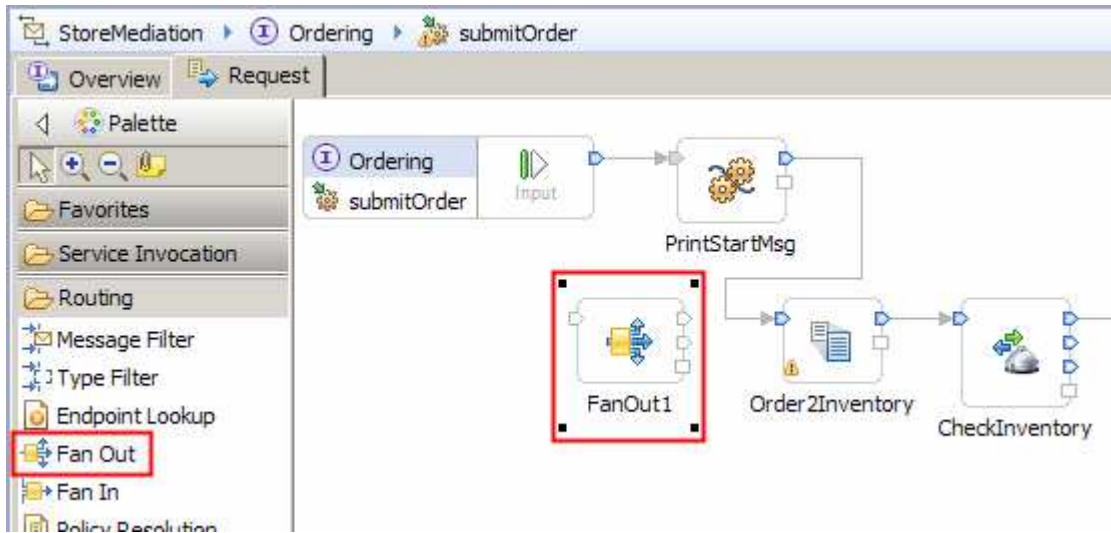
\_\_\_ 5. Add a fan out primitive to the canvas and rewire the flow so that it is between the PrintStartMsg and Order2Inventory primitives. This primitive defines where the iterative part of the flow begins.

- **Display Name:** StartIteration
- Change the message type assigned to the input terminal of Order2Inventory from "Specific message type" to "Automatically determined message type"
- Wire : PrintStartMsg to StartIteration
- Wire : StartIteration(out terminal) to Order2Inventory

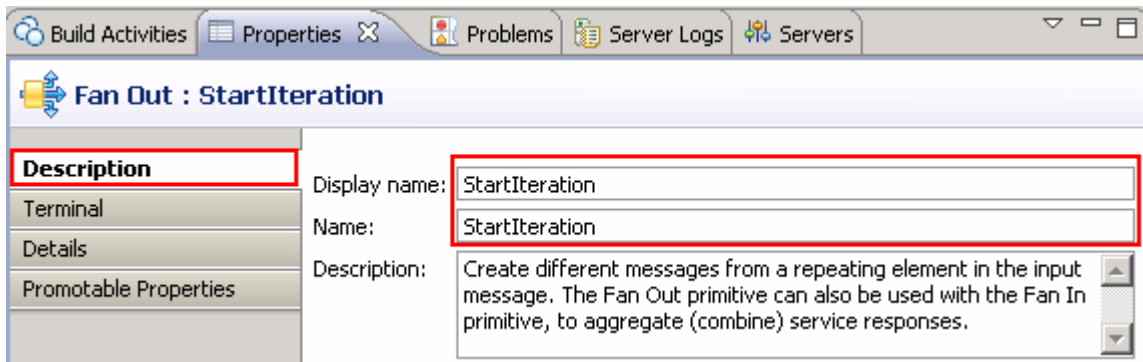




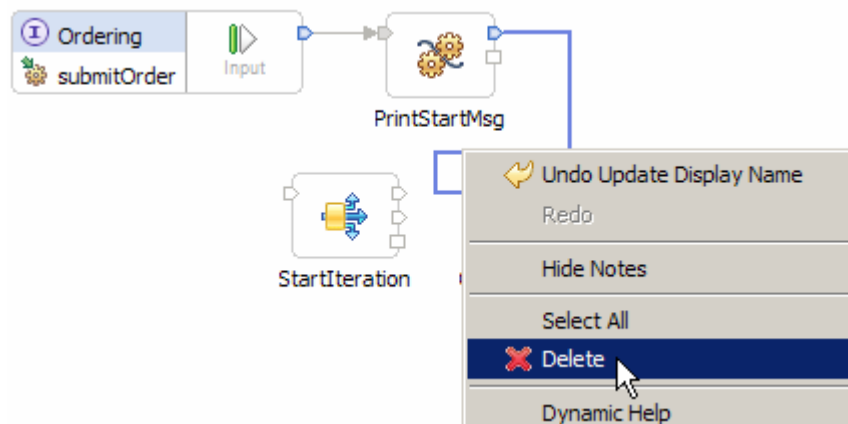
- \_\_\_ a. From the **Palette**, select **Routing** → **Fan Out** and then click the canvas as shown below (between the PrintStartMsg and Order2Inventory primitives as shown below).



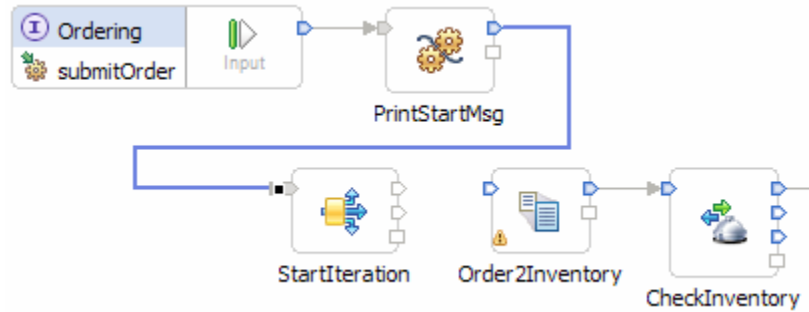
- \_\_\_ b. Ensure that the fan out primitive is selected and navigate to the **Description** panel of the **Properties** view and change the **Display name** to **StartIteration**



- \_\_\_ c. Remove connection between the **PrintStartMsg** and **Order2Inventory** primitives. Right-click the line connecting the primitives and select **Delete** from the pop-up menu

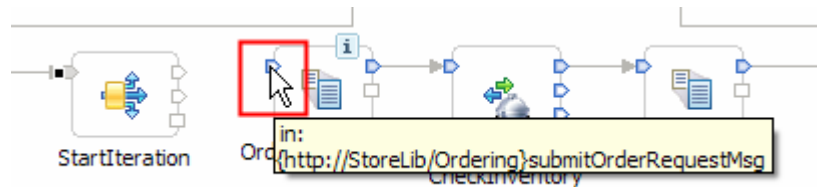


- \_\_\_ d. Add a connection from **PrintStartMsg** to **StartIteration** primitive. Click the **out terminal** of **PrintStartMsg** and drag to the **input terminal** of **StartIteration**

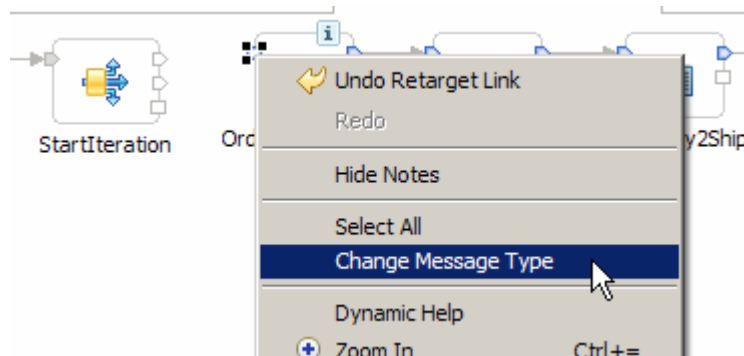


\_\_\_ e. The message type of the input terminal of Order2Inventory needs to pick up propagated type information from the fan out primitive. To enable this to happen, you need to change the message type setting for the terminal.

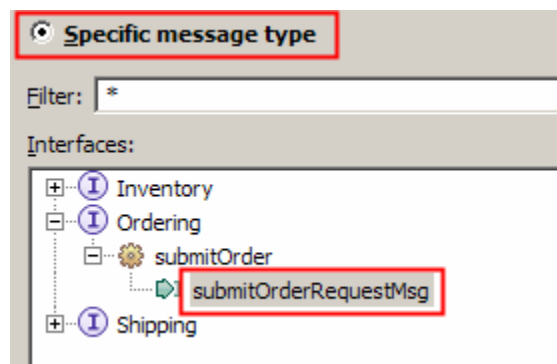
1) Place the mouse pointer over the input terminal of Order2Inventory



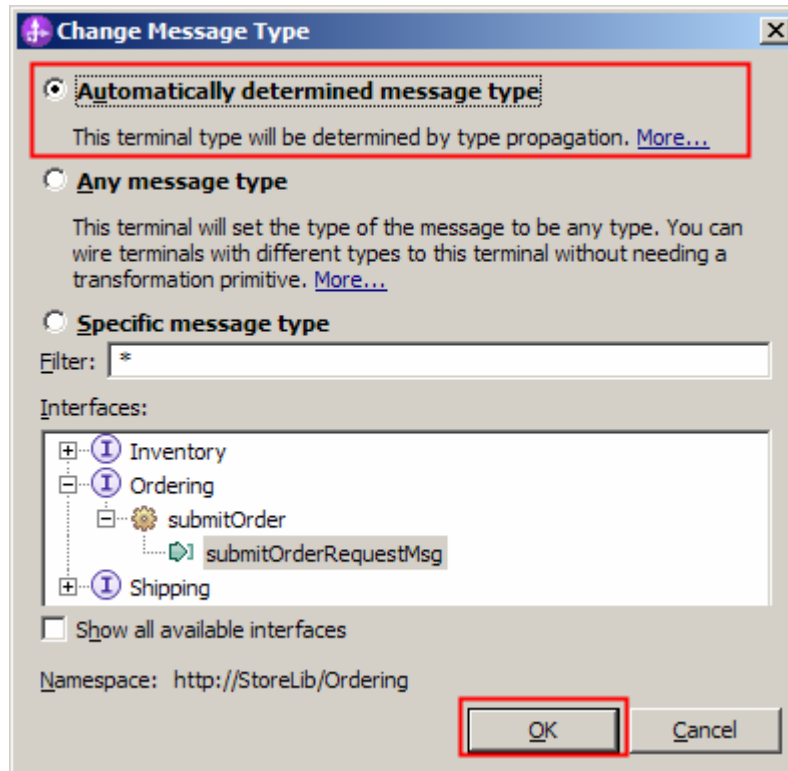
2) Right click to get a pop-up menu for the terminal and select **Change Message Type**



3) The Change Message Type dialog is displayed. Note that input terminal has a specific message type set which will prevent type information propagation from the fan out.



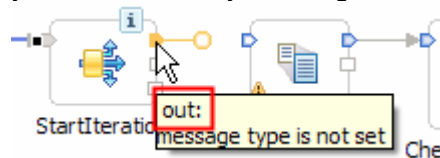
- 4) Change the message type setting by selecting the radio button **Automatically determined message type**

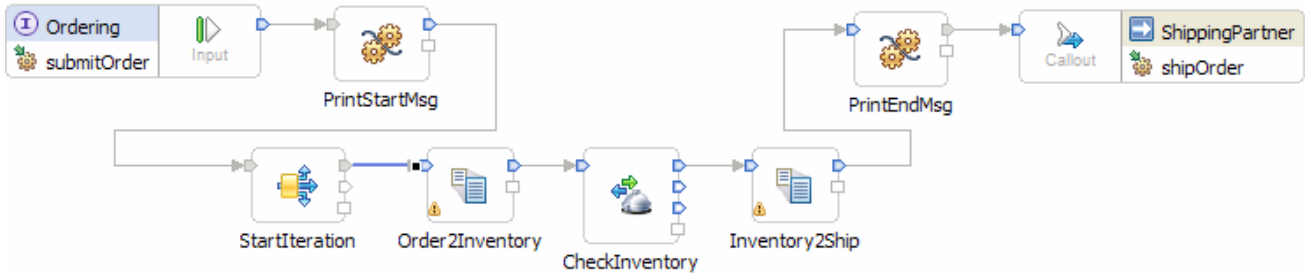


- 5) Hit **OK**

- \_\_\_ f. Now add a connection from the **StartIteration** primitive to the **Order2Inventory** primitive. Click the **out terminal** of **StartIteration** and drag to the **input terminal** of **Order2Inventory**

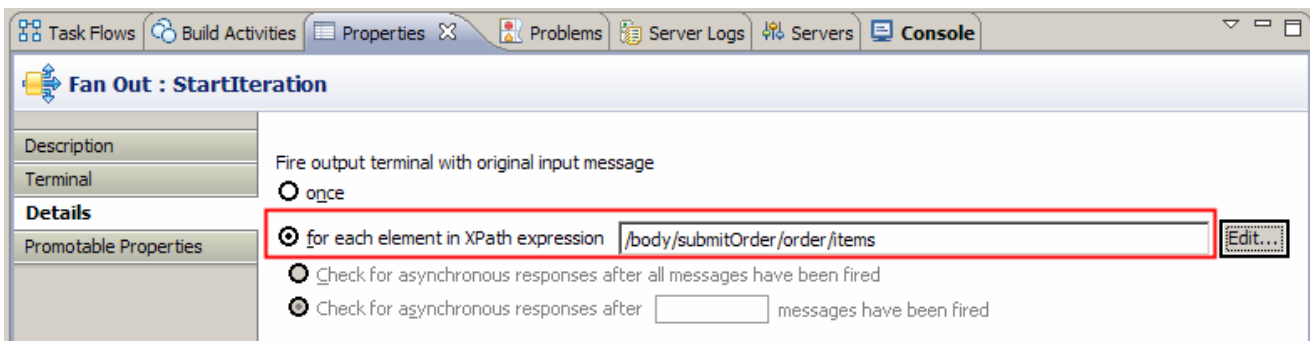
**NOTE:** There are two output terminals for a fan out primitive, so make sure you are wiring from the **out** terminal and not from the **noOccurrences** terminal. The out terminal is the top one, and you can see this by hovering the mouse pointer over the terminal.





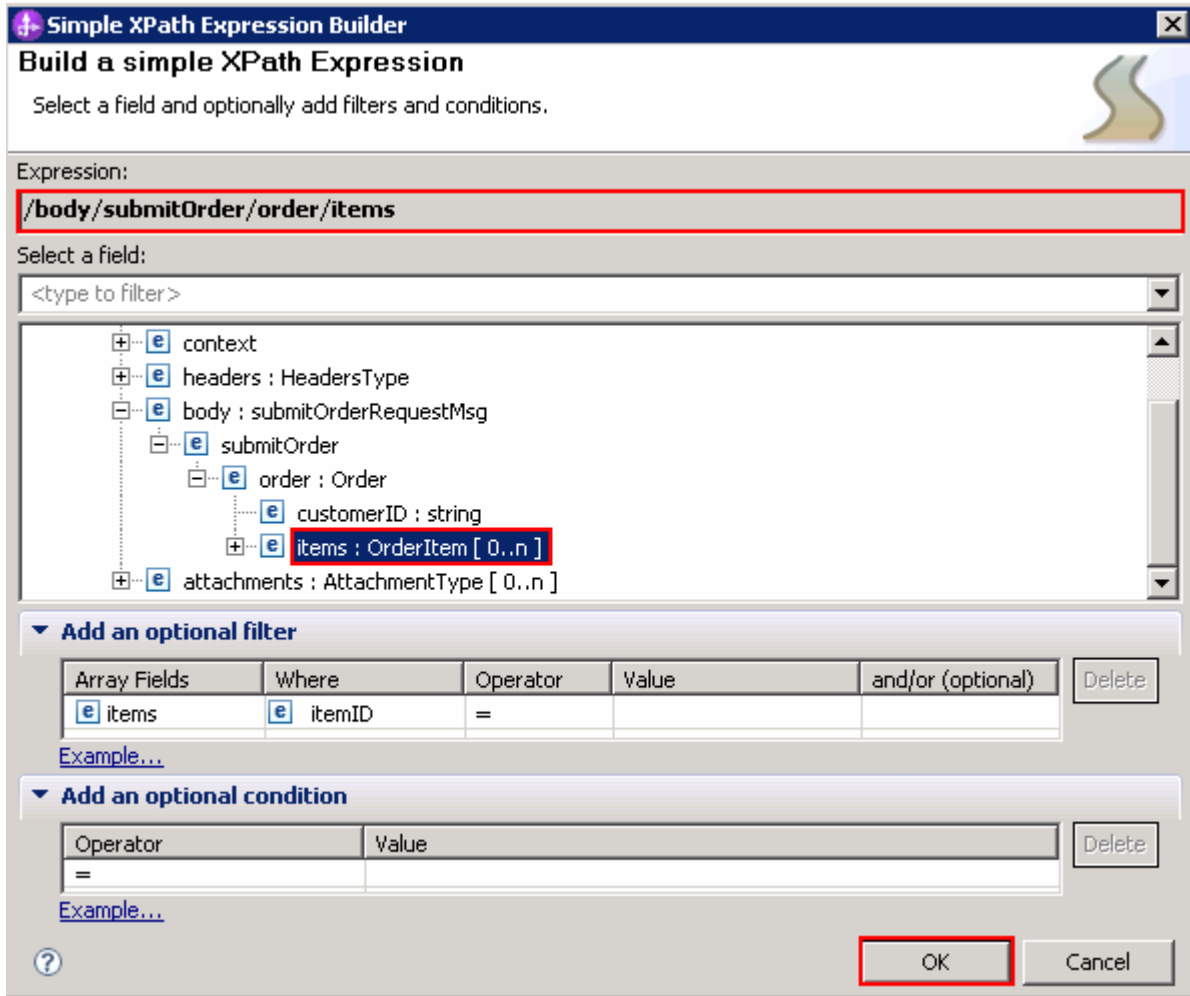
\_\_\_ 6. Configure the StartIteration fan out primitive to iterate through the items array in the Order business object.

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



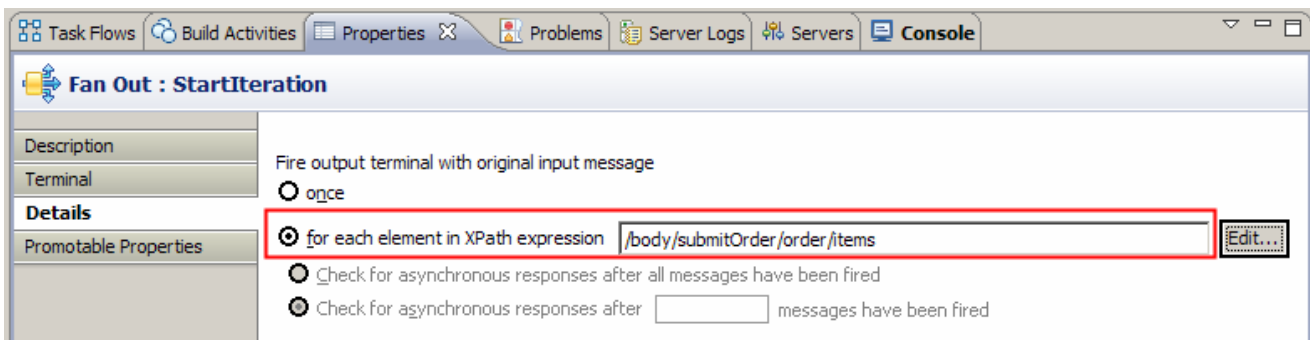
- -----
  - \_\_\_ a. Ensure that the **StartIteration** fan out primitive is selected and navigate to the **Details** panel of the **Properties** view.
  - \_\_\_ b. Select the radio button next to **for each element in XPath expression**
  - \_\_\_ c. Click the **Edit...** button. The Simple XPath Expression Builder wizard is opened
  - \_\_\_ d. Expand **ServiceMessageObject** → **body** → **submitOrder** → **order**

\_\_\_ e. Select **items**. You should see the XPath expression populated in the **Expression** text area as shown below:



\_\_\_ f. Click **OK**

\_\_\_ g. The resulting properties should look like this:



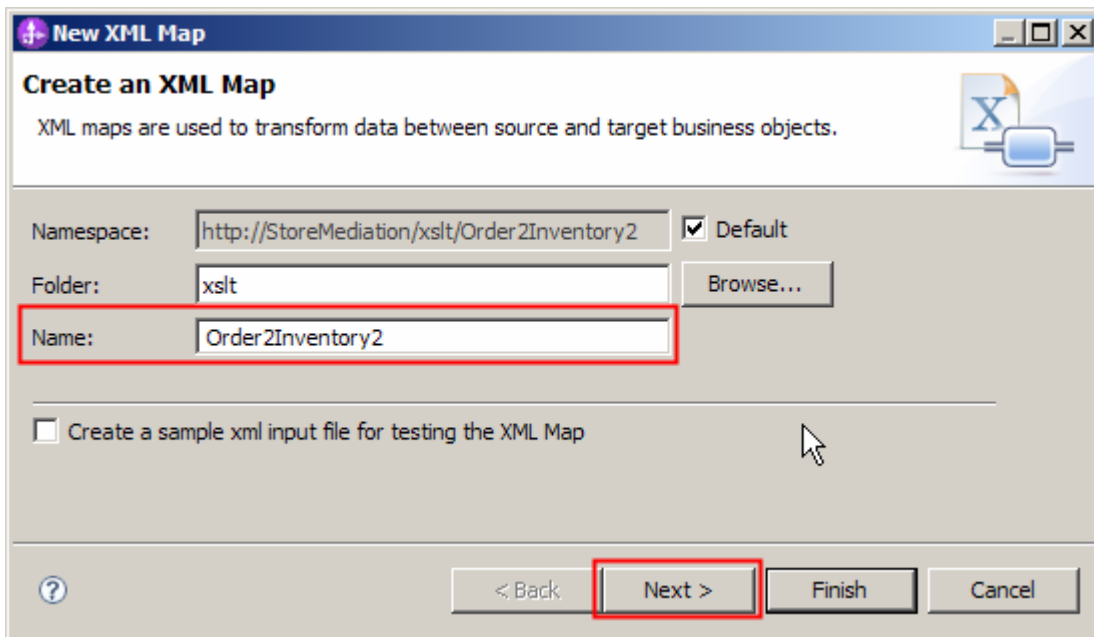
\_\_\_ 7. In the Order2Inventory XSL transformation primitive create a new map named Order2Inventory2 (this will replace the existing Order2Inventory1 map currently configured for the primitive).

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

\_\_\_ a. Select the **Order2Inventory** 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 window is opened

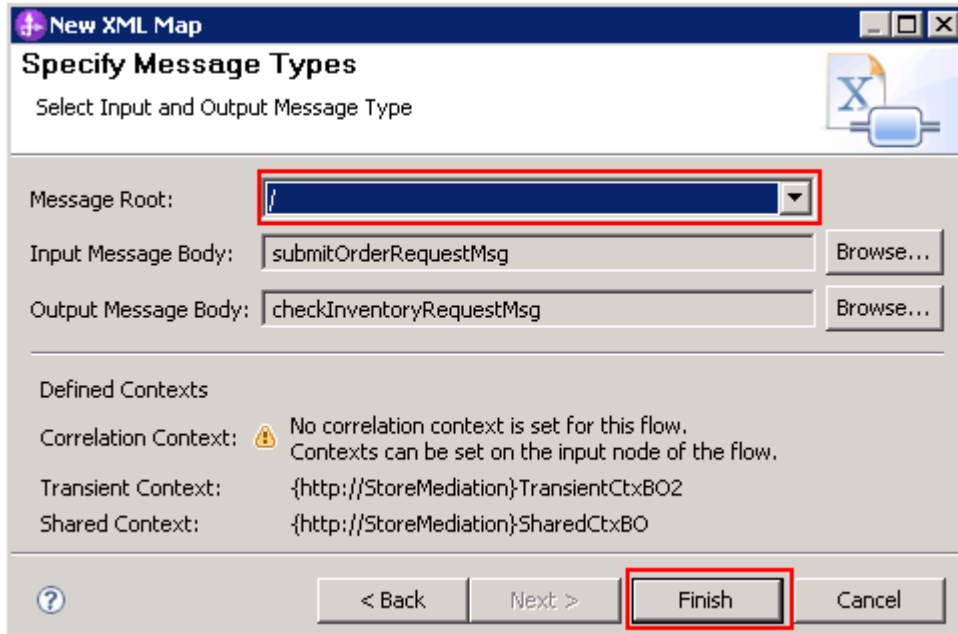
\_\_\_ c. For **Name** enter **Order2Inventory2**



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


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

- 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.

- (1) Moving fields between the source and target SMOs that will not change.
- Use toolbar icon (  ) 'Map source to target based on name and types'
- (2 and 3) Saving information in the transient context that is needed later in the flow, from (2)the source message body and from (3)the FanOutContext

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

- 
- (4) Setting up the target message body needed to call the Inventory service using information from the FanOutContext.

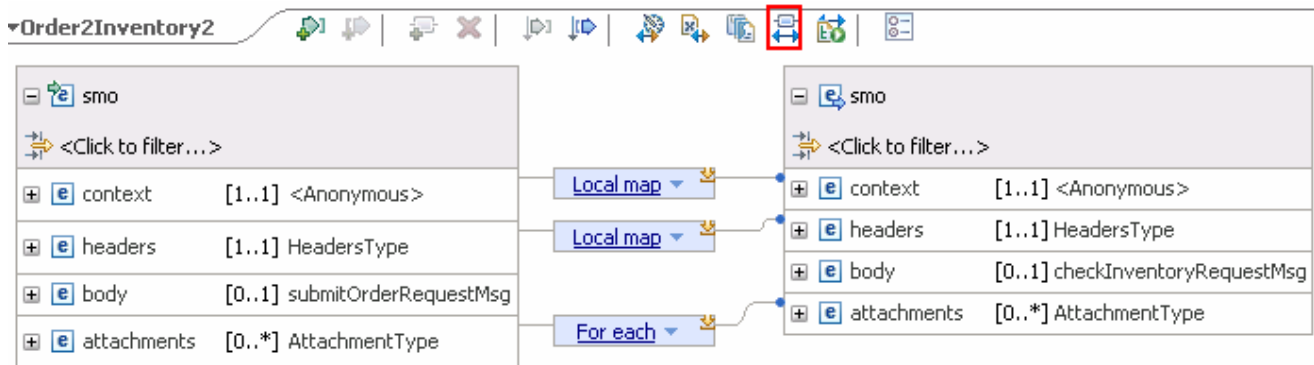
Source (left side)	Target (right side)
context/primitiveContext/FanOutContext/occurrence	body/checkInventory/orderItem

• -----  
 \_\_\_ a. These steps address moving fields between the source and target SMOs that will not change

1) Click 'Map source to target based on name and types' icon (↔)



2) This maps the context and headers using local maps. The body is not mapped because the source and target bodies are different. The attachments are mapped with a For each transform. The result is shown below:

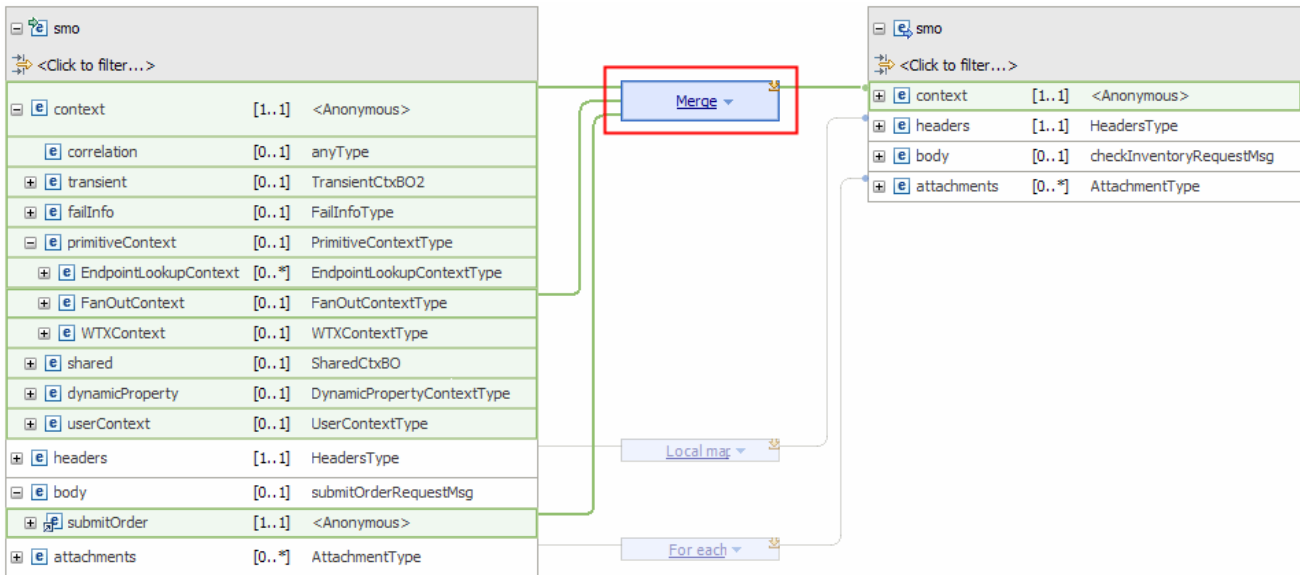


\_\_\_ b. These steps address saving information in the transient context. Some of that information comes from the source message body and the item information for the current iteration comes from the FanOutContext. The following steps are needed to map those elements by using merge rather than local maps.

- 1) In the source (left side) smo, expand **context** → **primitiveContext**, revealing **FanOutContext**
- 2) Click **FanOutContext** and drag it to the **Local map** from context to context, which converts it to a **Merge** transform
- 3) Again in the source smo, expand body, revealing **submitOrder**
- 4) Click **submitOrder** and drag it to the **Merge** transform



5) The resulting map should look like this:



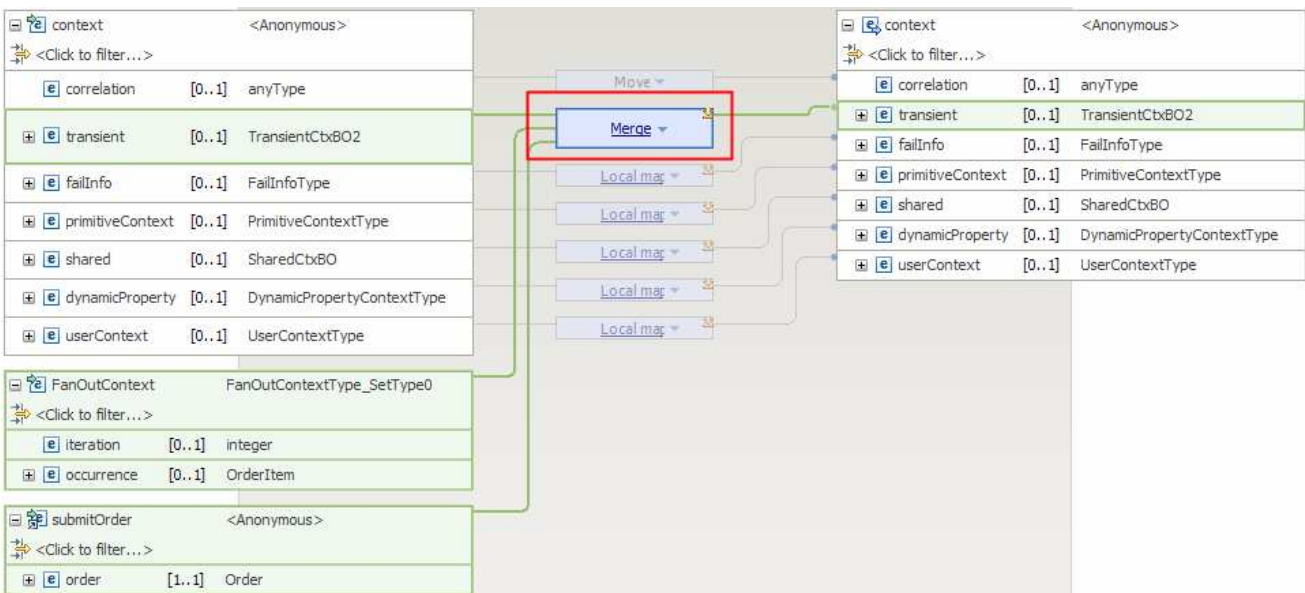
6) On the **Merge** transform, click the **Edit** (🔧) icon to navigate into the merge.



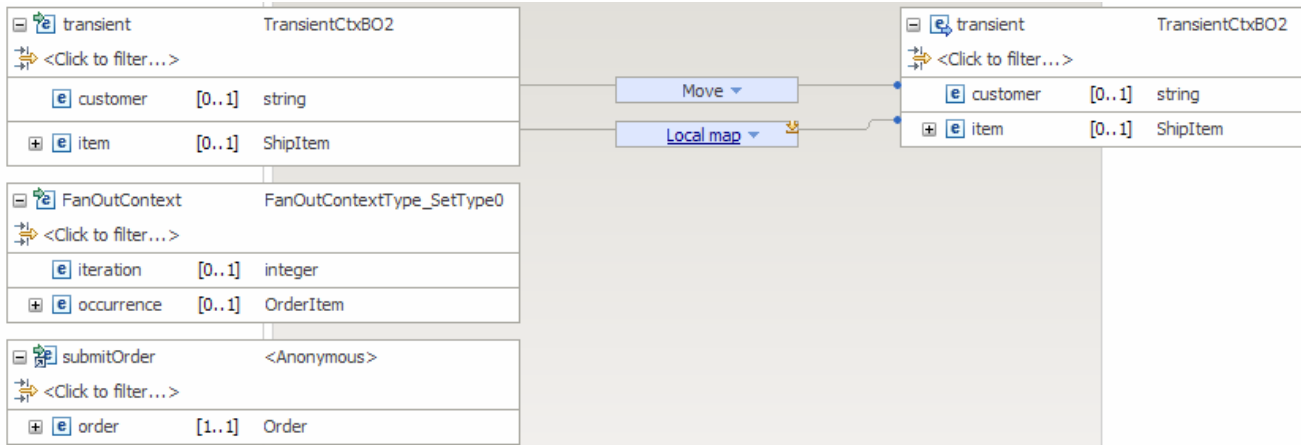
7) In the merge, click **FanOutContext** and drag it to the **Local map** for transient to transient, which converts it into a **Merge** transform.

8) Likewise, click **submitOrder** and drag it to the **Merge** transform.

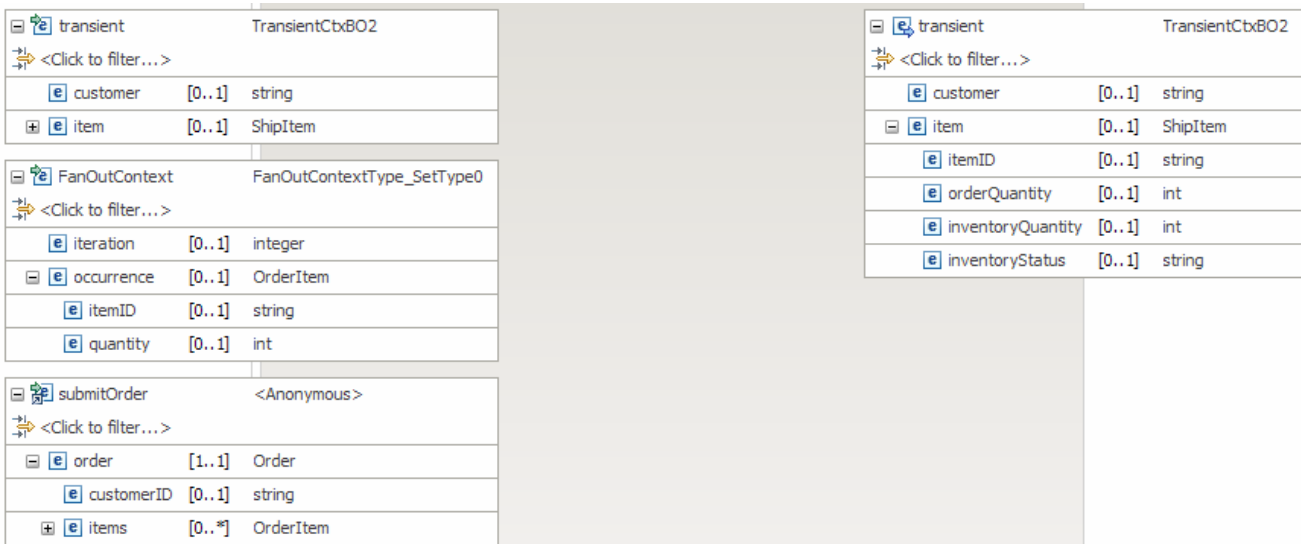
9) The mapping for the merge should look like this:



- 10) On the **Merge** transform, click the **Edit** (🔧) icon to navigate into the merge. It will look like this:

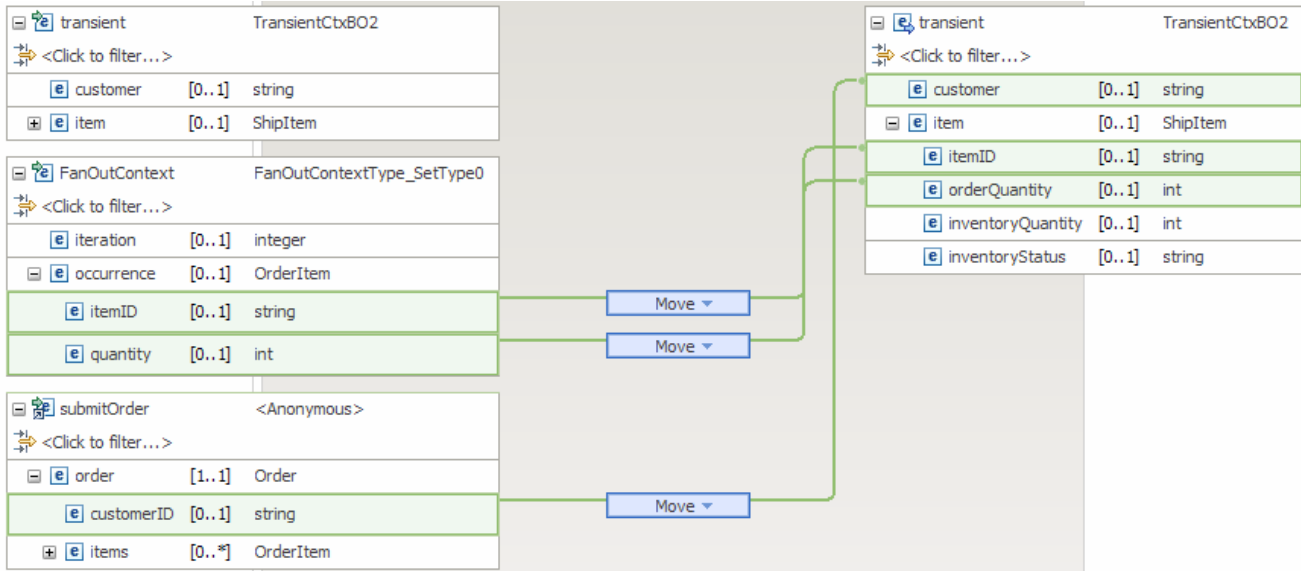


- 11) Select both the **Move** and **Local map** transformations (hold down Ctrl button on keyboard and click each transformation) and hit **Delete selected elements** (✖) icon from the top
- 12) On the source side, under **submitOrder**, expand **order**, revealing **customerID**
- 13) Also on the source side, under **FanOutContext**, expand **occurrence**, revealing **itemID** and **quantity**
- 14) On the target side, expand **item**
- 15) The merge mapping now looks like this:

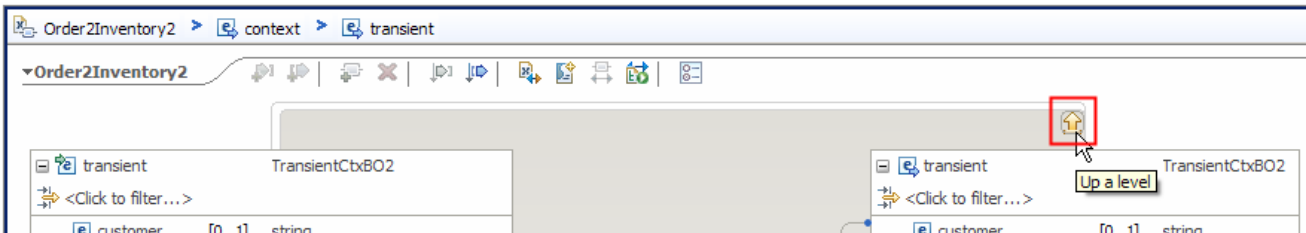


- 16) From under **submitOrder**, click **customerID** and drag it to **customer** to create a Move transform.
- 17) Under **FanOutContext**, click **itemID** and drag to **itemID**.
- 18) Also under **FanOutContext**, click **quantity** and drag to **orderQuantity**.

19) All the transforms required for the transient context are now defined. The merge map will look like this:



20) Click 'Up a level' (🏠) icon which will bring you one level up to the context merge mapping.

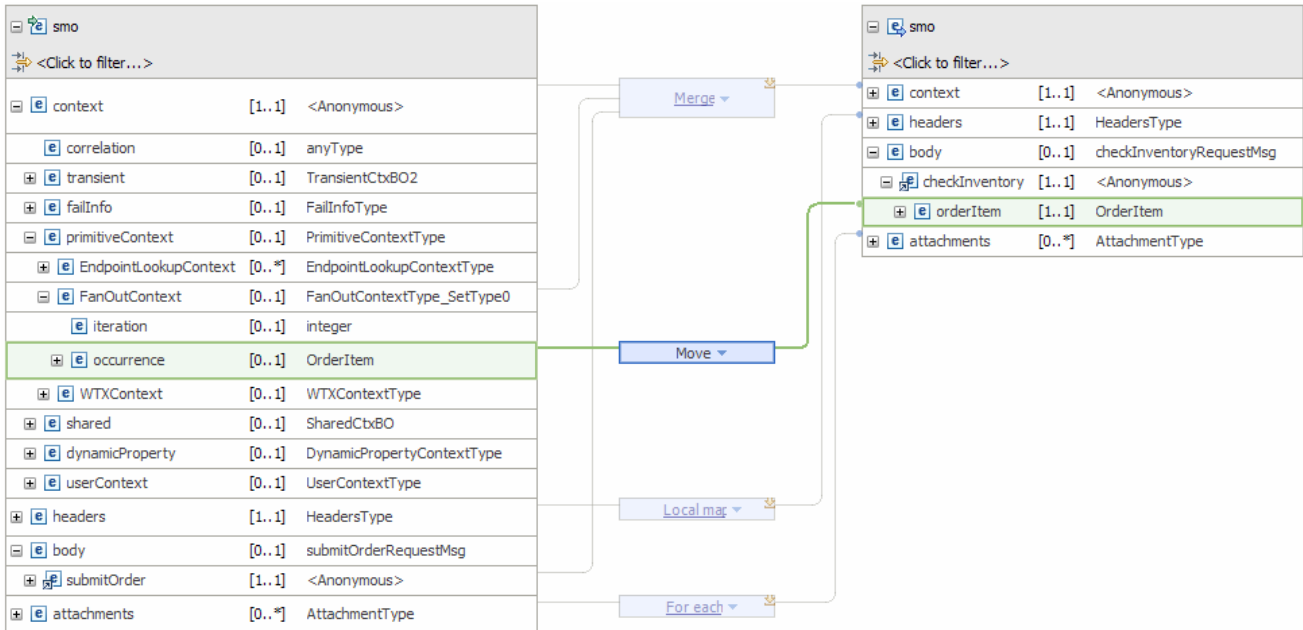


21) Click 'Up a level' (🏠) icon one more time which will now bring you back the smo mapping level

\_\_\_ c. These steps are needed to set up the target message body needed to call the inventory service. The item information for this iteration is found in /context/primitiveContext/FanOutContext/occurrence and needs to be moved to the target message body at body/checkInventory/orderItem

- 1) On the source smo, expand **FanOutContext** which reveals **occurrence**
- 2) On the target smo, expand **body** → **checkInventory** which reveals **orderItem**
- 3) Click **occurrence** and drag to **orderItem** to create a **Move** transformation.

4) The resulting map looks like this:



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

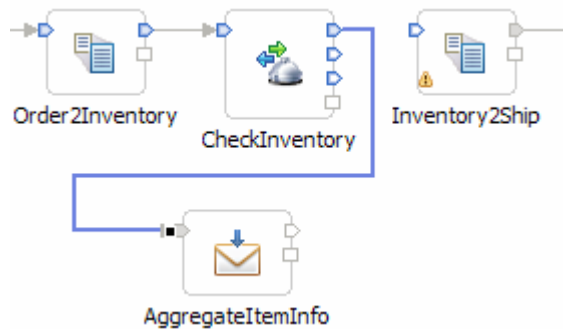
\_\_\_ e. Close the Order2Inventory2.map file

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

- -----

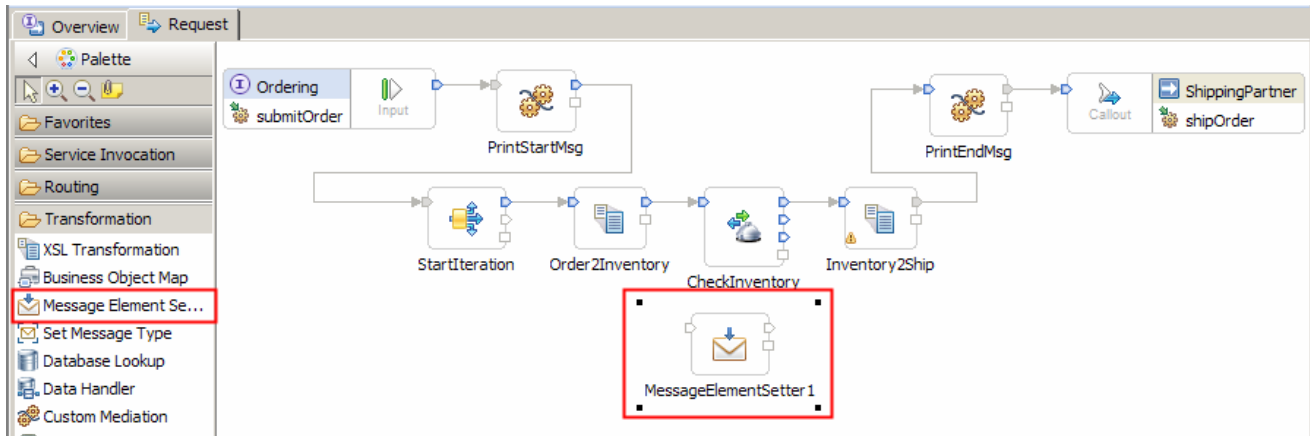
\_\_\_ 10. Add a message element setter primitive used to add the complete order and inventory information for the current item to an aggregation of items. Rewire the flow so that it follows the CheckInventory primitive.

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

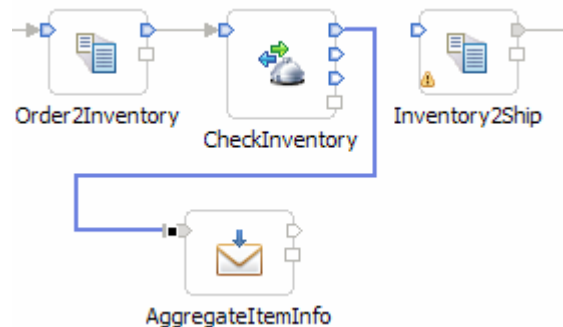


- -----

- \_\_\_ a. From the **Palette**, select **Transformation** → **Message Element Setter** and then click the canvas to add the primitive as shown below.



- \_\_\_ b. Ensure that this primitive is selected and navigate to the **Description** panel of the **Properties** view.
- \_\_\_ c. Change the **Display name** to **AggregateItemInfo**
- \_\_\_ d. Remove the **connection** between the **CheckInventory** and **Inventory2Ship** primitives by right-clicking on the line connecting them and selecting **Delete** from the pop-up menu
- \_\_\_ e. Add a connection from the **CheckInventory's out** terminal to the **AggregateItemInfo** primitive. Click the **out** (uppermost) terminal and drag it to the input terminal.



- \_\_\_ 11. Configure the **AggregateItemInfo** primitive to copy the inventory information from the body of the response message to the transient context. Then move the complete item information for both the order and inventory from the transient context to the shared context where all the items are being aggregated.
- 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

- Then set properties to append to the end of the shared context 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

- 
- The resulting message elements should look like this. The append must be the last element

Message Elements:

1	Copy element /body/checkInventoryResponse/inventoryItem/inStockQuantity to element /context/transient/item/inventoryQuantity
2	Copy element /body/checkInventoryResponse/inventoryItem/status to element /context/transient/item/inventoryStatus
3	Append /context/transient/item to element /context/shared/aggregateShipItems

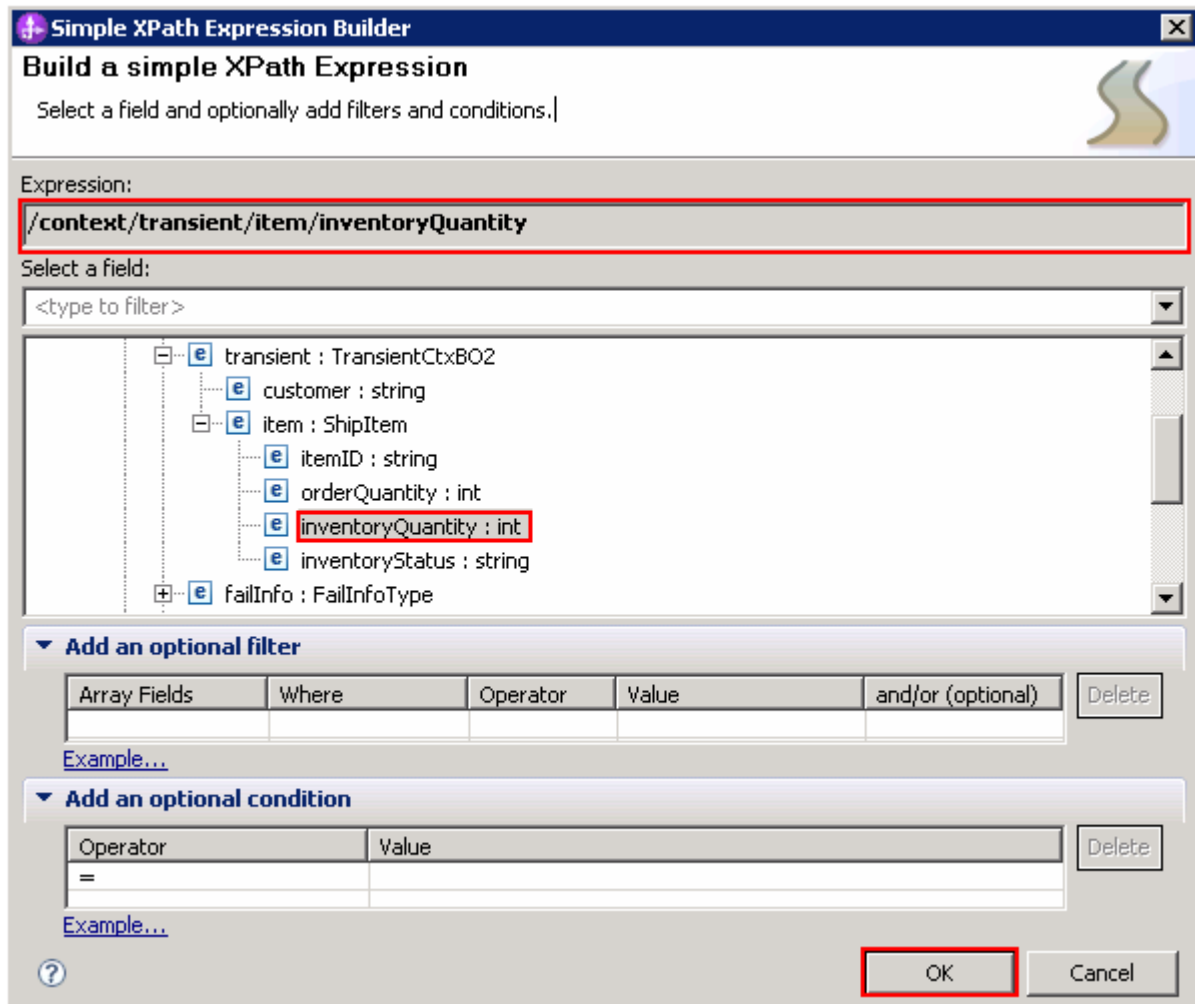
- -----

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

- 1) Ensure that the **AggregateItemInfo** primitive is selected and navigate to the **Details** panel of the **Properties** view
- 2) Under Message Elements table, click the **Add...** button. The Add/Edit window is opened
- 3) Select **Copy** for **Action**

## 4) Define the Target:

- a) Click **Browse...** for **Target**. The Simple XPath Expression Builder wizard is opened
- b) Expand **ServiceMessageObject** → **context** → **transient** → **item**
- c) Select **inventoryQuantity**. You should see the Xpath Expression populated in the **Expression** text area

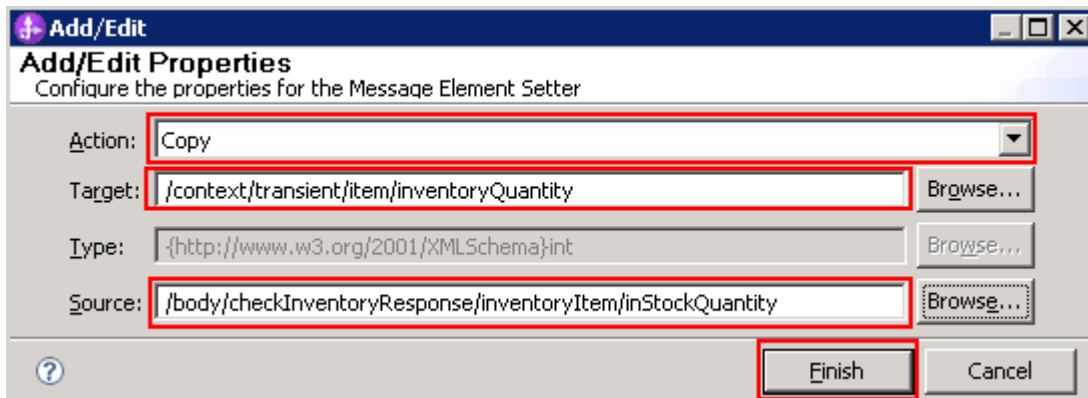


- d) Click **OK**. You are now back to Add/Edit window

## 5) Similarly, define the Source:

- a) Click **Browse** for **Source** to open the Simple XPath Expression Builder
- b) Expand **ServiceMessageObject** → **body** → **checkInventoryResponse** → **inventoryItem**
- c) Select **inStockQuantity**. You should see the Xpath Expression populated in the **Expression** text area
- d) Click **OK**

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



7) Click **Finish**

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

Message Elements:

1	Copy element /body/checkInventoryResponse/inventoryItem/inStockQuantity to element /context/transient/item/inventoryQuantity
---	--

\_\_\_ b. Similarly, copy status to target /context/transient/item/inventoryStatus from source /body/checkInventoryResponse/inventoryItem/status

1) Under Message Elements table, click the **Add...** button. The Add/Edit window is opened

2) Select **Copy** for **Action**

3) Define Target:

a) Click **Browse...** for **Target**. The Simple XPath Expression Builder wizard is opened

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

c) Select **inventoryStatus**. You should see the Xpath Expression populated in the **Expression** text area

d) Click the **OK** button. You are now back to Add/Edit window

4) Define Source:

a) Click **Browse** for **Source** to open the Simple XPath Expression Builder

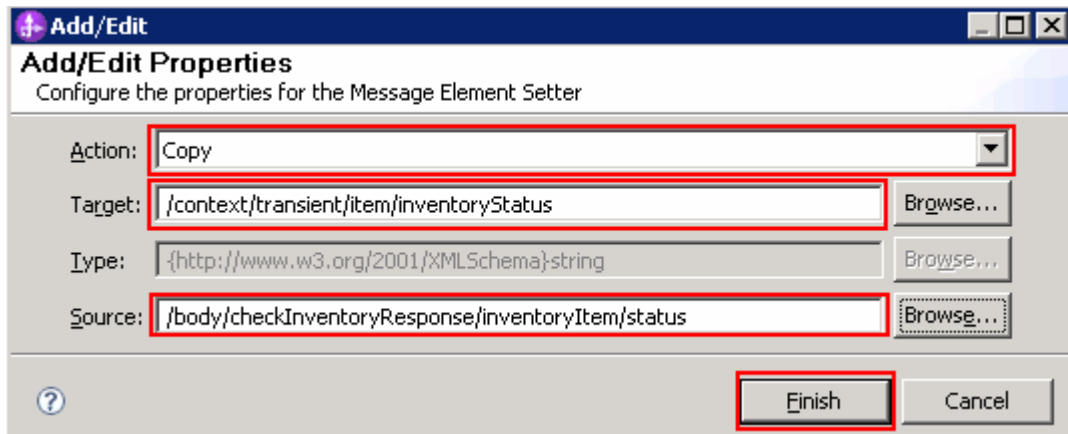
b) Expand **ServiceMessageObject** → **body** → **checkInventoryResponse** → **inventoryItem**

c) Select **status**. You should see the Xpath Expression populated in the **Expression** text area

d) Click **OK**



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:

1	Copy element <code>/body/checkInventoryResponse/inventoryItem/inStockQuantity</code> to element <code>/context/transient/item/inventoryQuantity</code>
2	Copy element <code>/body/checkInventoryResponse/inventoryItem/status</code> to element <code>/context/transient/item/inventoryStatus</code>

\_\_\_ c. Now take the item information in the transient context and append it to the end of the array of items in the shared context

1) Under Message Elements table, click the **Add...** button. The Add/Edit window is opened

2) Select **Append** for **Action**

3) Define the Target:

a) Click **Browse** for **Target**. The Simple XPath Expression Builder wizard is opened

b) Expand **ServiceMessageObject** → **context** → **shared**

c) Select **aggregateShipItems**. You should see the Xpath Expression populated in the **Expression** text area

d) Click **OK** button. You are now back to Add/Edit window

4) Define the Source:

a) Click **Browse** for **Source**

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

c) Select **item**. You should see the Xpath Expression populated in the **Expression** text area

d) Click **OK**

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

Action:	Append	
Target:	/context/shared/aggregateShipItems	Browse...
Type:	{http://StoreLib}ShipItem	Browse...
Source:	/context/transient/item	Browse...

6) Click **Finish**

7) The Message Elements table should now look like this.

Message Elements:

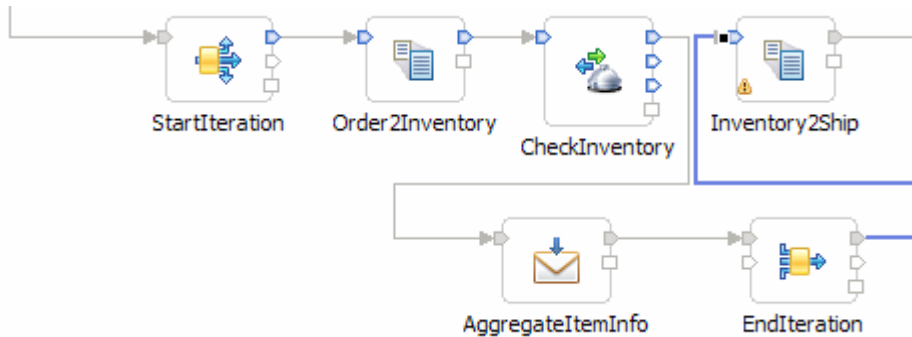
1	Copy element /body/checkInventoryResponse/inventoryItem/inStockQuantity to element /context/transient/item/inventoryQuantity
2	Copy element /body/checkInventoryResponse/inventoryItem/status to element /context/transient/item/inventoryStatus
3	Append /context/transient/item to element /context/shared/aggregateShipItems

12. 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.

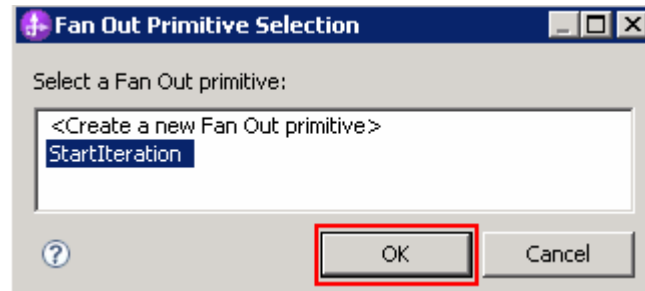
- **Wire:** AggregateItemInfo to EndIteration(in terminal)
- **Wire:** EndIteration(out terminal) to Inventory2Ship



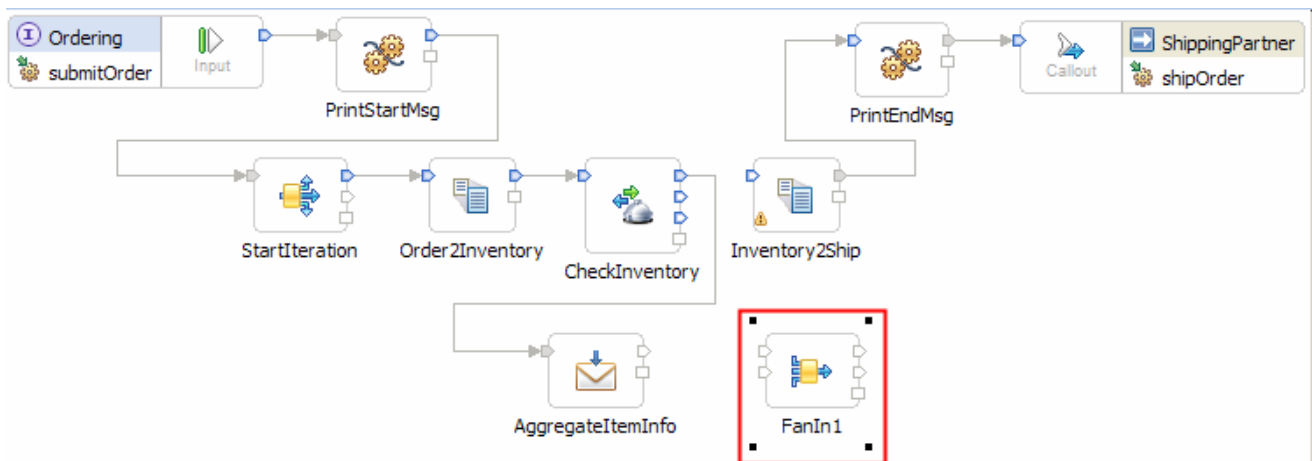
-----

\_\_\_ a. Add a new Fan In to the canvas:

- 1) From the **Palette**, select **Routing** → **Fan In** and then click the canvas to the right of the **AggregateItemInfo** primitive
- 2) A Fan Out Primitive Selection dialog is opened. Select **StartIteration** as the associated Fan Out.



- 3) Click **OK**
- 4) You will now see a new Fan In primitive added on the canvas.



\_\_\_ b. Ensure that this primitive is selected and then navigate to the **Description** panel of the **Properties** view

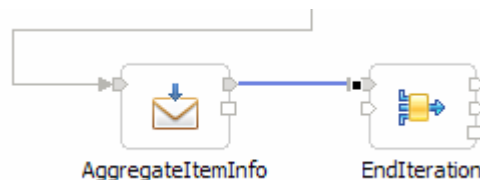
\_\_\_ c. Change the **Display name** to **EndIteration**

\_\_\_ d. Add a connection from the **AggregateItemInfo** primitive to the **in** terminal of the **EndIteration** primitive. Click the **out** terminal and drag it to the **in** (uppermost) terminal.

---

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

---

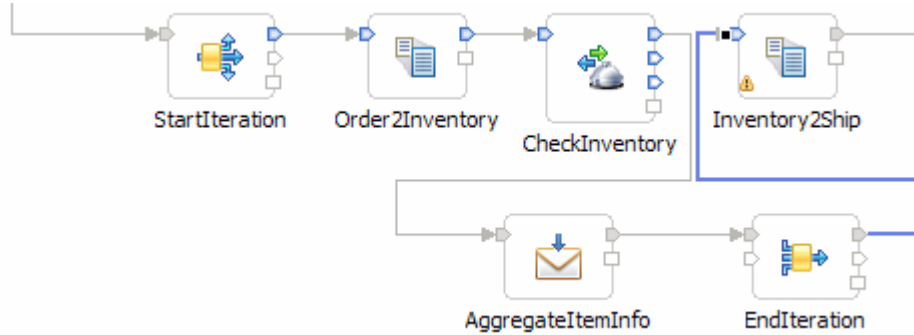


- \_\_\_ e. Similarly, add a connection from the **EndIteration** primitive's out terminal to the **Inventory2Ship** primitive. Click the **out** (uppermost) terminal and drag it to the input terminal.'

---

**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.

---



- \_\_\_ 13. 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

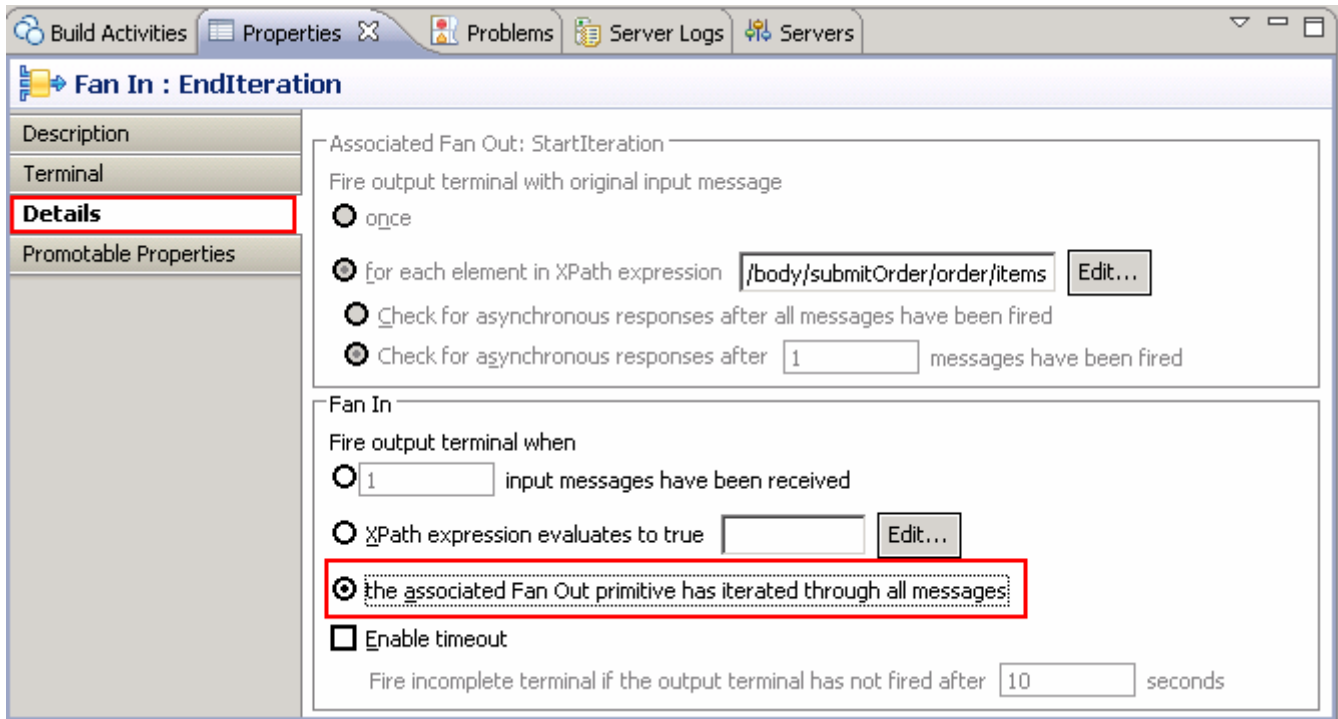
the associated Fan Out primitive has iterated through all messages

Enable timeout

Fire incomplete terminal if the output terminal has not fired after  seconds

- \_\_\_ a. Ensure that the **EndIteration** primitive is selected and and navigate to the **Details** panel of the **Properties** view

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



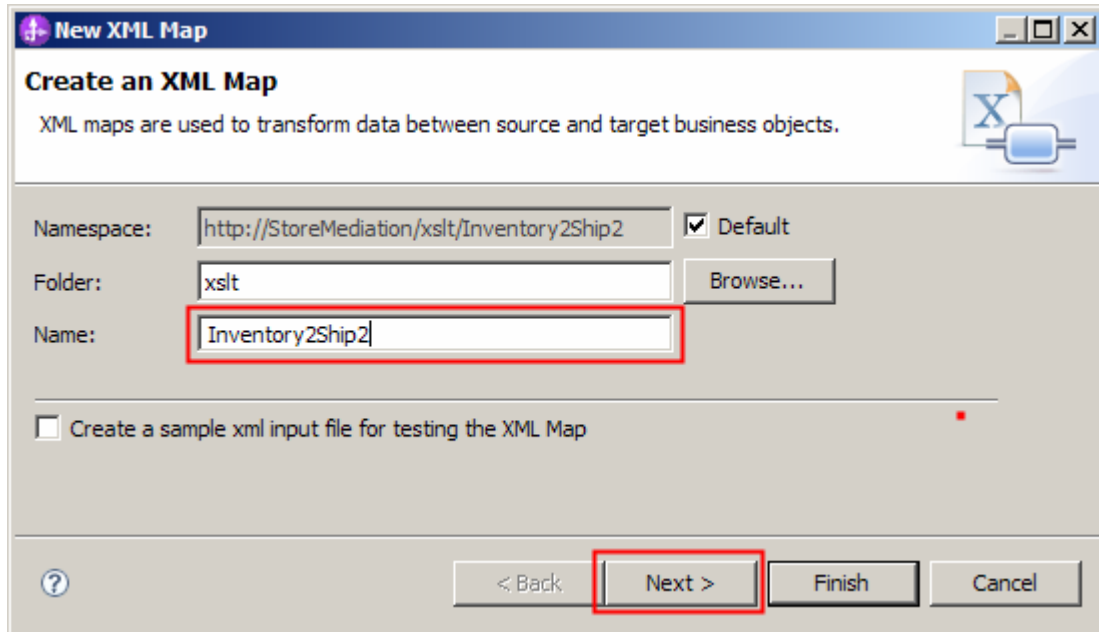
- \_\_\_ 14. In the Inventory2Ship XSL transformation primitive create a new map named Inventory2Ship2 (this will replace the existing Inventory2Ship1 map currently configured for the primitive).

- **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 **New...** next to **Mapping file**. The New XML Mapping wizard is opened.

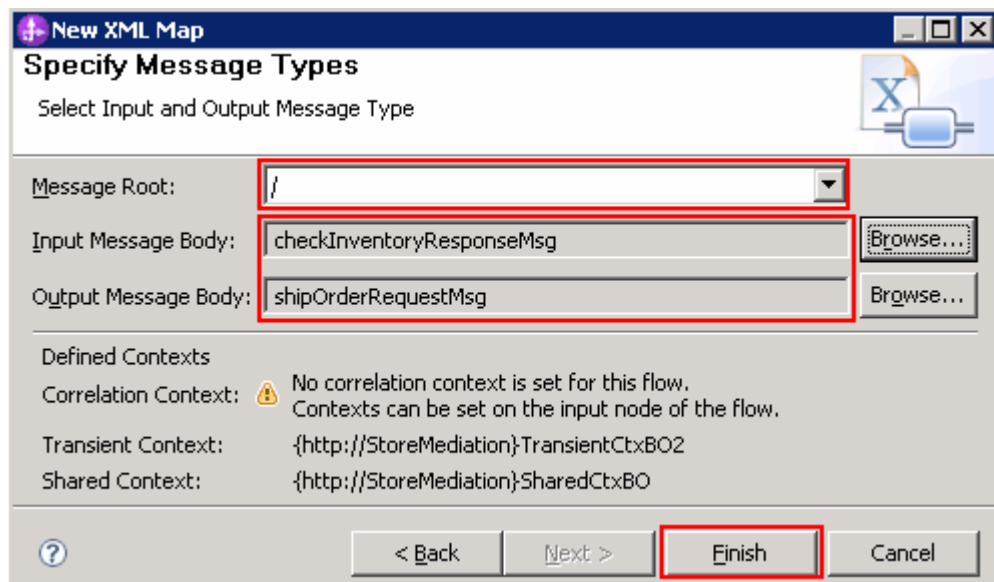
\_\_ c. For **Name** enter **Inventory2Ship2**



\_\_ 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**



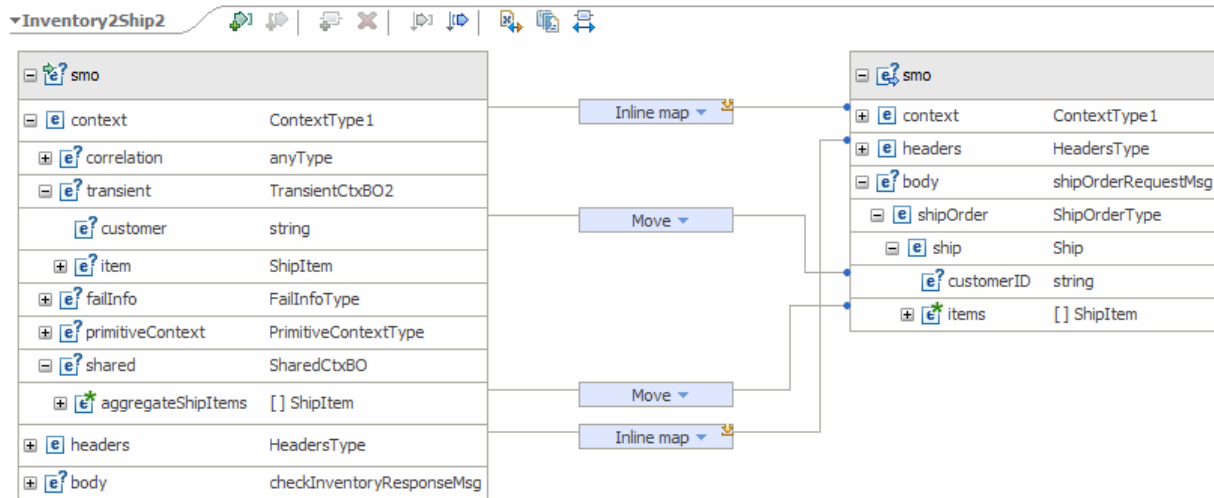
\_\_ f. Click **Finish**. An Inventory2Ship2.map file is opened in the XML Mapping editor

15. 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 ShipItem array definition as is needed by target message body, the entire array can be handled with a simple move transformation

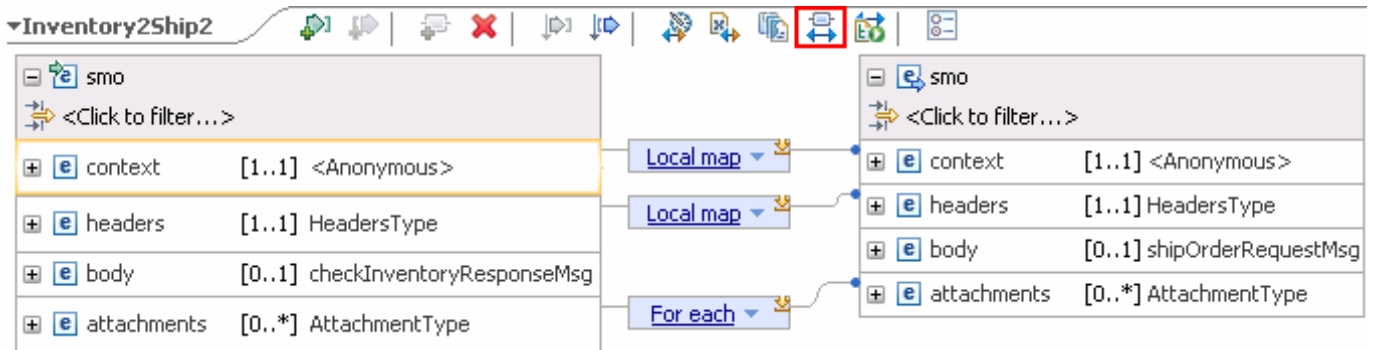
- Use toolbar icon (↔) "Map source to target based on name and types"
- At the SMO level (top level map) add these move transforms

**NOTE:** Unlike previous cases of arrays, where you moved a specifically indexed array element, in this case the aggregateShipItems 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/aggregateShipItems[ ]	body/shipOrder/ship/items[ ]

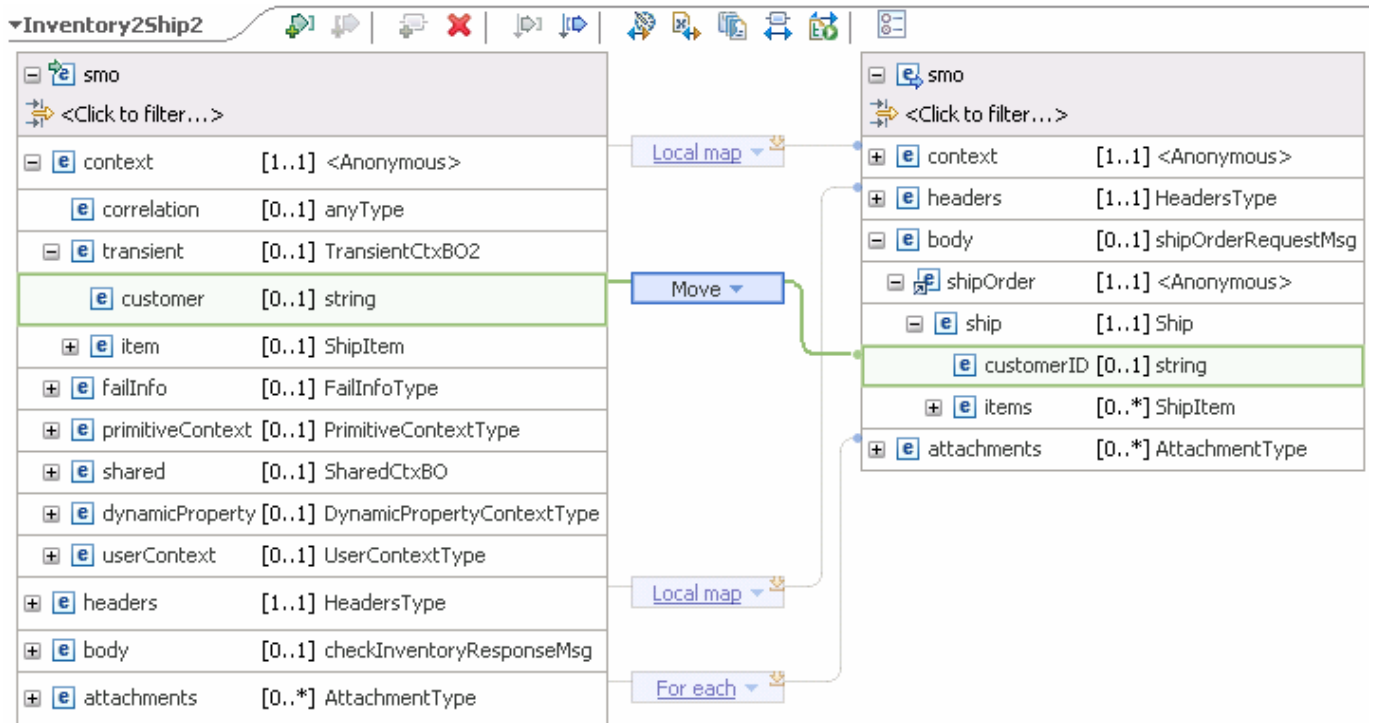


- a. Click 'Map source to target based on name and types' icon (↔). This maps the context and headers using local maps. The body is not mapped because the source and target bodies are different. The attachments are mapped with a For each transform. The result is shown below



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

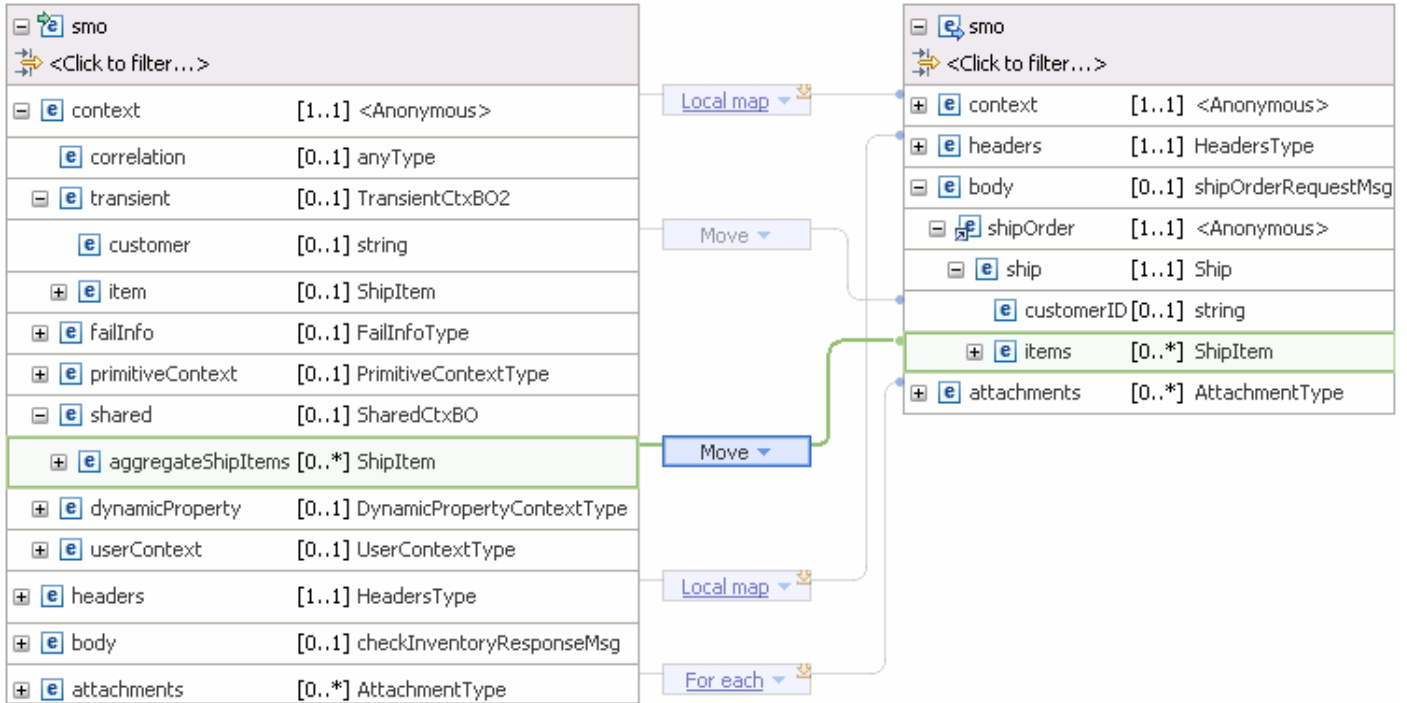
- 1) In the source smo, expand **context** → **transient**, revealing **customer**
- 2) In the target smo, expand **body** → **shipOrder** → **ship**, revealing **customerID** and **items**
- 3) Click **customer** in the source smo and drag it to **customerID** in the target, creating a **Move** transform.





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

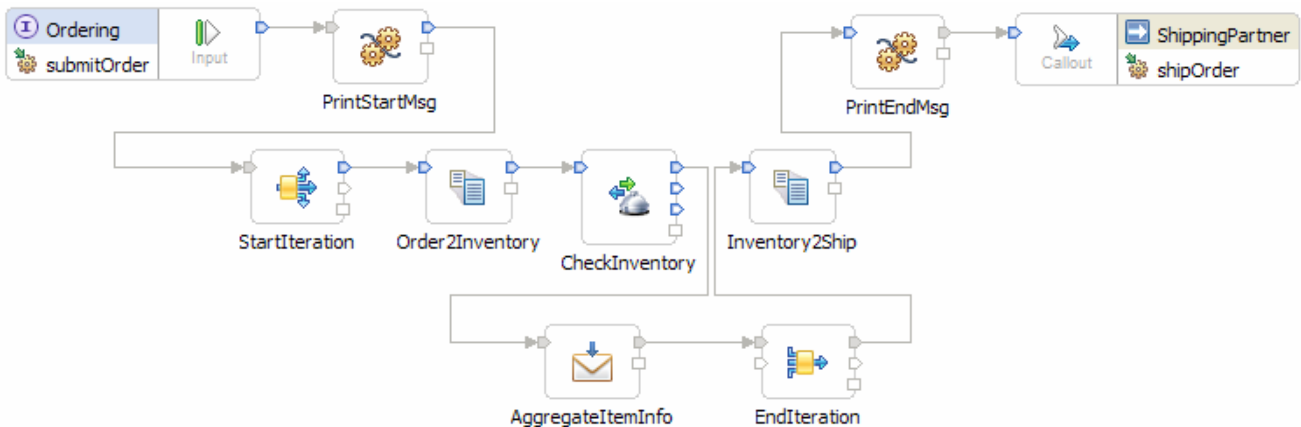
- 1) In the source smo, expand **context** → **shared**, revealing **aggregateShipItems**
- 2) Click **aggregateShipItems** in the source smo and drag it to **items** in the target, creating a **Move** transform.



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

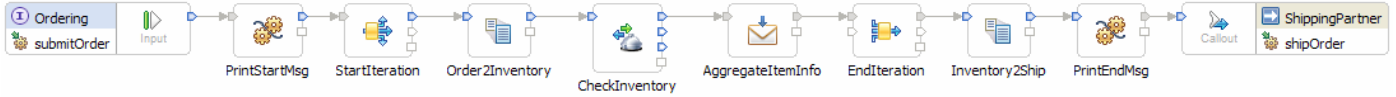
\_\_ e. Close the Inventory2Ship2.map file

\_\_\_ 16. The flow is now completed and should look something similar to this.

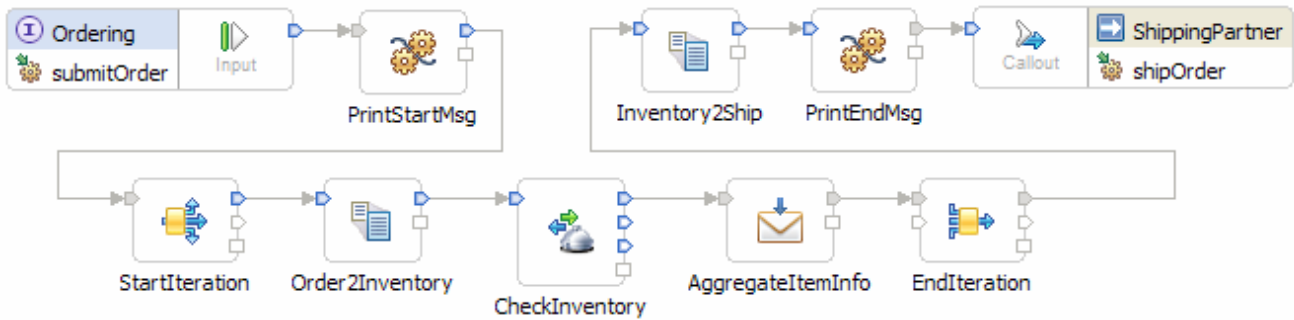


17. Optionally, if you want to rearrange the layout of the flow in the editor you have two options. It can be done by the editor using the **Layout Contents** menu option or you can move individual primitives to an arrangement that makes sense when considering the function of the flow.

- **Layout done using the Layout Contents pop-up menu option (right click mediation flow editor background)**



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



- -----

18. 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:

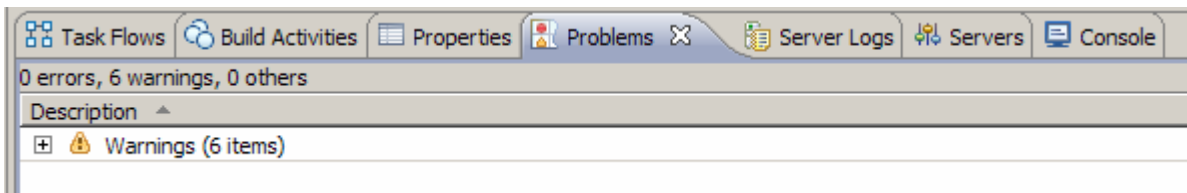


b. For each tab with an asterisk (\*), click the tab to give it focus and from the menu select **File → Save** (or **Ctrl + S** from your keyboard) to save your changes.

19. 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:

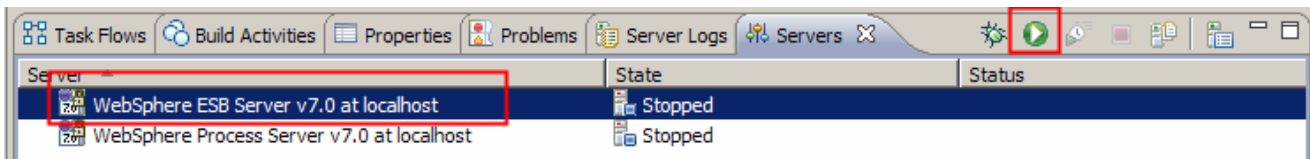


## Part 3: Test the splitting/aggregating mediation

**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

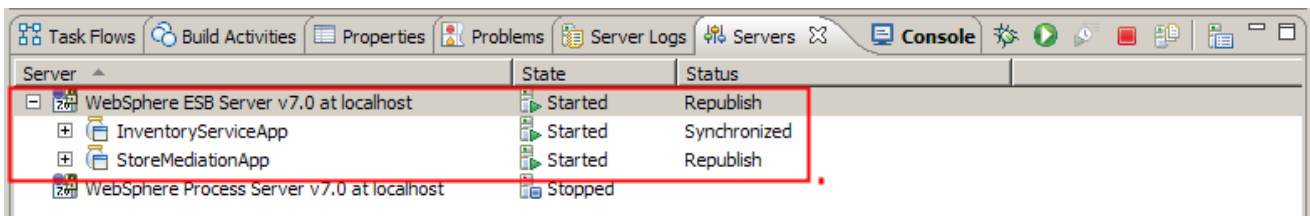
- \_\_\_ 1. If not already running, start the WebSphere Enterprise Service Bus (or WebSphere Process Server) test server.
  - -----
  - \_\_\_ a. From the **Servers** view of WebSphere Integration Developer, select **WebSphere ESB Server v7.0** and click **Start the server** icon (  ) from the toolbar



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

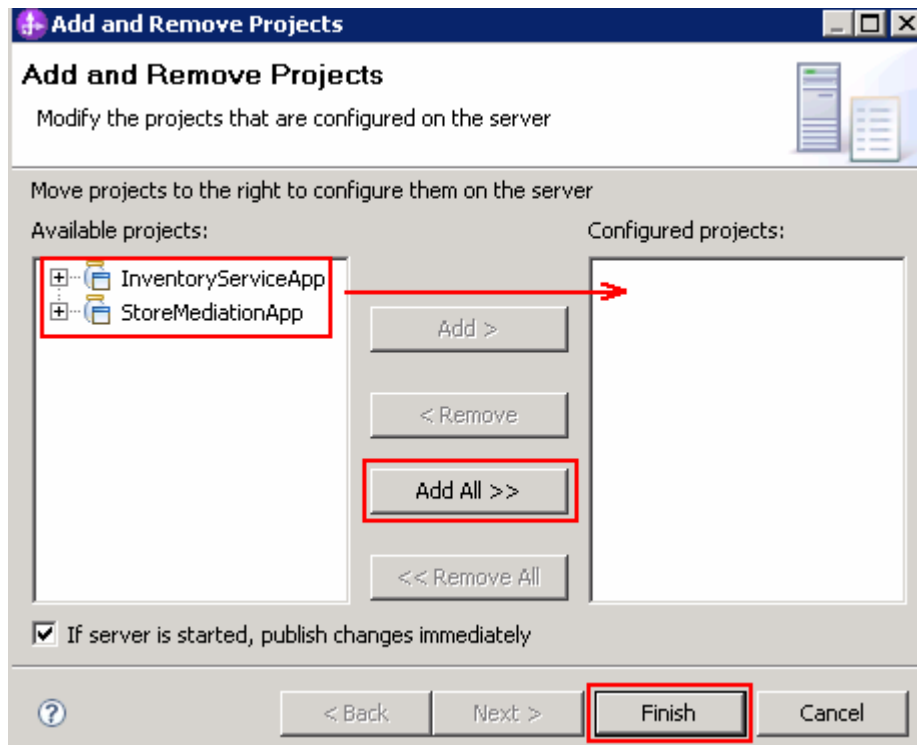
**NOTE:** Depending upon the preferences you have specified, the **Console** view or **Server Logs** view might grab the focus from the **Servers** view. If this is the case, the status in the lower right should indicate when the server start is complete, at which time you can switch back to the **Servers**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

- \_\_\_ 2. Check to see if the InventoryServiceApp and StoreMediationApp are deployed on the test 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 be displayed below the server as shown here. **Follow the appropriate instructions in either step b or step c**



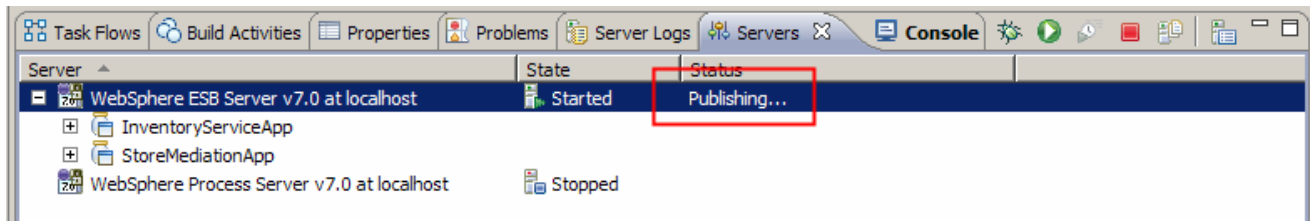
- \_\_\_ b. If the applications are **not deployed yet**, add both projects to the server following these steps:
  - 1) Right-click **WebSphere ESB Server v7.0** under the Servers view and select **Add and remove projects...** from the pop-up menu

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

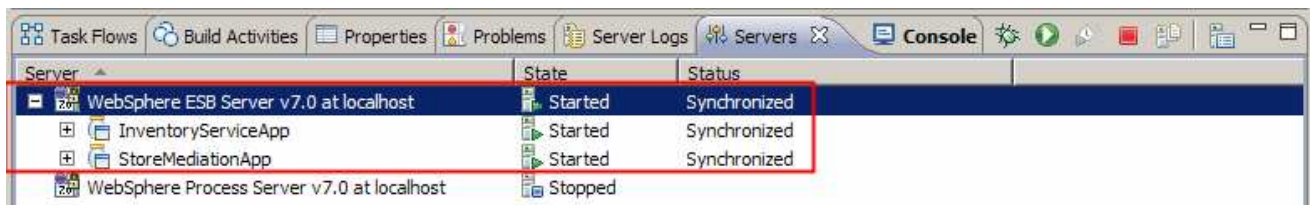


- 3) The projects will now be moved to Configured projects. Click **Finish**

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

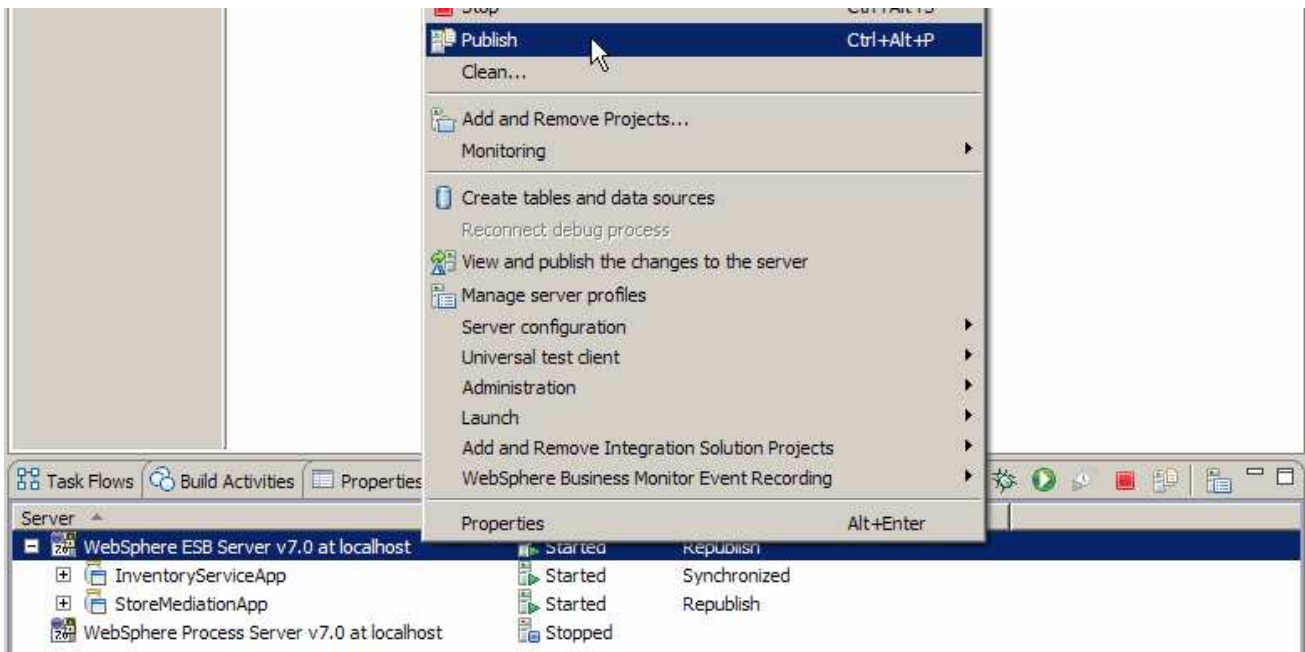


- 5) Once the publishing is done you should see the two applications started as shown here:

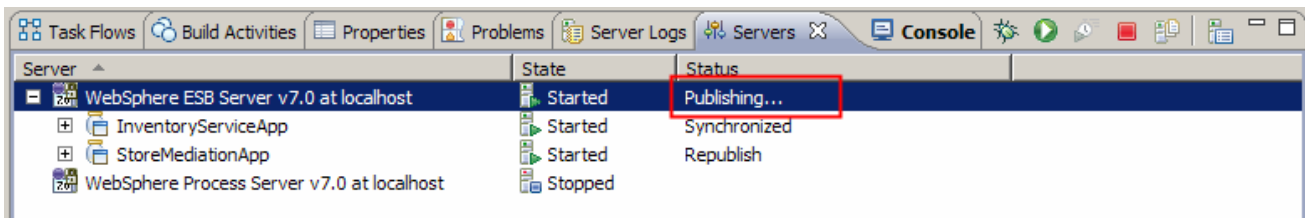


\_\_\_ c. If the applications are **already deployed**, a new publish needs to be done to load them into the server with the latest changes.

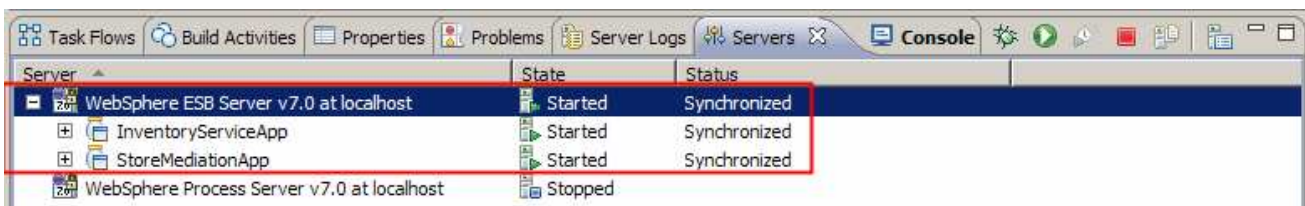
1) Right click the server and select **Publish** from the pop-up menu



2) Wait while the applications are publishing to the server



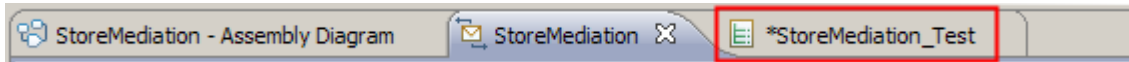
3) Once the publishing is done you should see the two applications started as shown here:




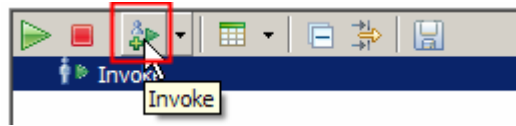
\_\_\_ 3. Check if the StoreMediation\_Test panel is still present from the previous lab.

- **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

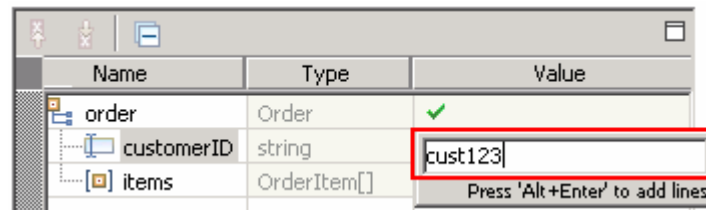
- Set customerID to cust123
- Set items/items[0]/itemID to item001
- Set items/items[0]/quantity to 3
- Set items/items[1]/itemID to item009
- Set items/items[1]/quantity to 5
- Set items/items[2]/itemID to item002
- Set items/items[2]/quantity to 15

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 Initial request parameters table:

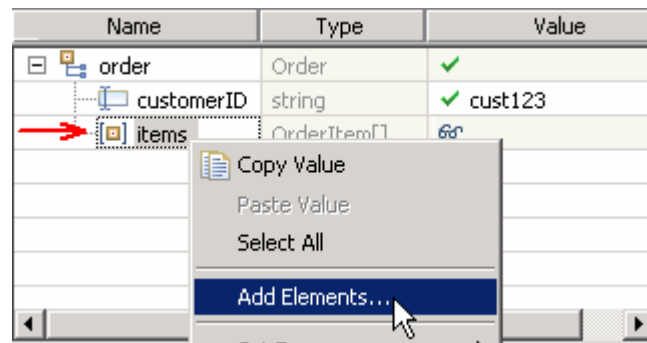
- 1) For **customerID**, click under Value and enter **cust123**



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

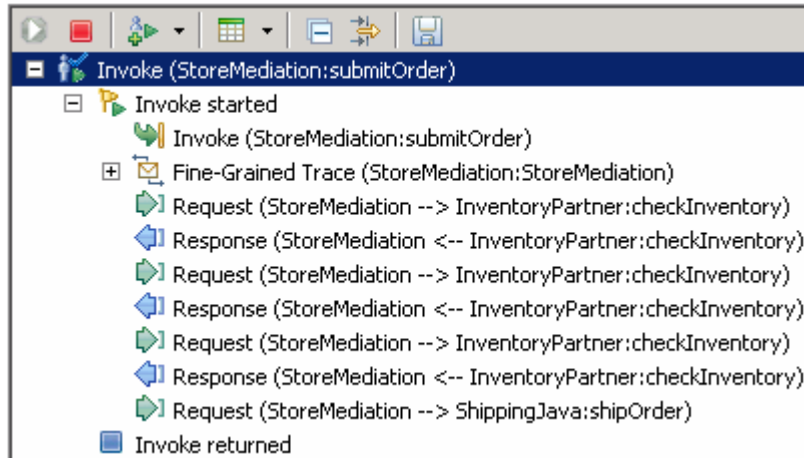


Name	Type	Value
order	Order	✓
customerID	string	✓ cust123
items	OrderItem[]	60

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

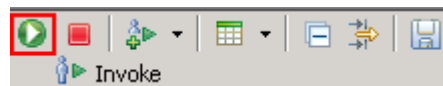
- 3) Enter **3** in the Add Element window:
- 4) Click **OK**
- 5) Enter values for items[0]:
  - a) For **itemID**, click under Value and enter **item001**
  - b) For **quantity**, click under Value and enter **3**
- 6) Enter values for items[1]:
  - a) For **itemID**, click under Value and enter **item009**
  - 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, as shown here:



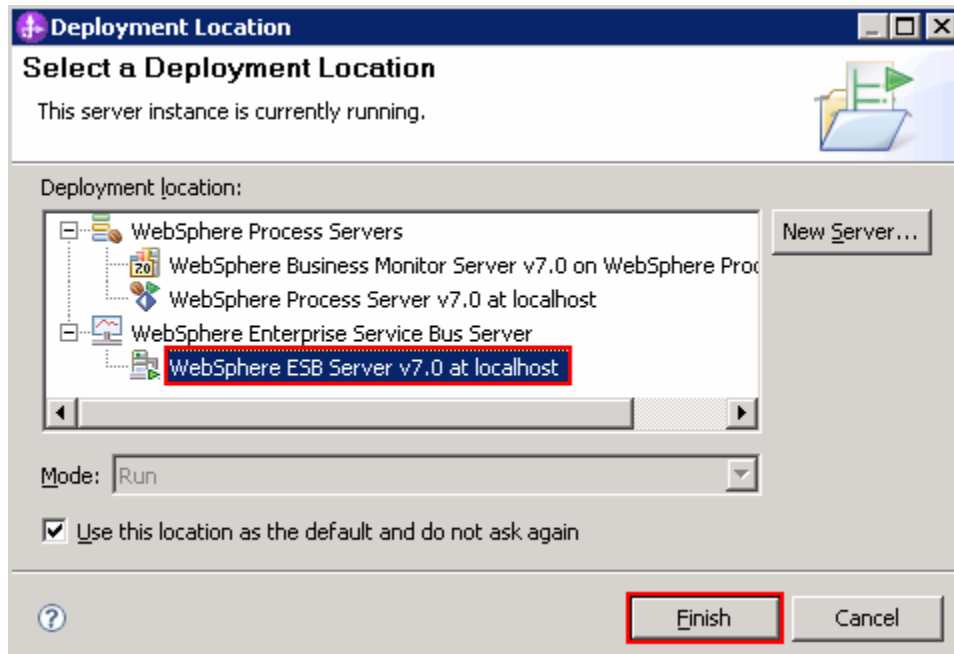
**NOTE:** The Fine-Grained Trace section has been collapsed. When expanded, it shows all of the nodes and primitives invoked by the flow. Although not discussed or explained in this lab, clicking on any of the entries within the fine grained trace will show the contents of the SMO at that point in the flow (in the 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

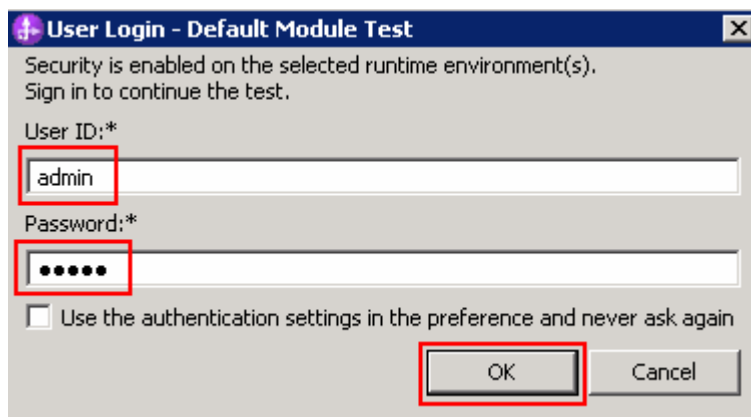




- \_\_\_ b. If test client is run for the first time, you should see a **Deployment Location** dialog prompting to select a run time sever where the applications are deployed. Select **WebSphere ESB Server v7.0** from the list



- \_\_\_ 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 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.



- \_\_\_ e. Click **OK**
- \_\_\_ f. Wait until the integration test client starts

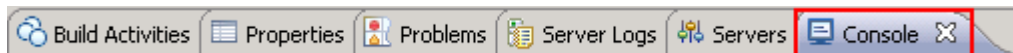
6. Switch to the Console view and examine the output, which should look similar to this screen capture. You can see the three invocations of the Inventory Service, one for each item, and the aggregated result sent to the Shipping Service containing all three items.

```

O *****
O ***** START mediation flow *****
O ***
I   processMessage
O ***** InvWorks - returning InventoryItem for itemID = item001
I   processMessage
I   processMessage
O ***** InvWorks - returning InventoryItem for itemID = item009
I   processMessage
I   processMessage
O ***** InvWorks - 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@5ede5ede
O Value:
O   customerID = cust123
O   items = ShipItem[3]
O     items[0] = <ShipItem@5efb5efb>
O       itemID = item001
O       orderQuantity = 3
O       inventoryQuantity = 5
O       inventoryStatus = OK - but stock is running low
O     items[1] = <ShipItem@5f085f08>
O       itemID = item009
O       orderQuantity = 5
O       inventoryQuantity = 45
O       inventoryStatus = OK - sufficient stock levels
O     items[2] = <ShipItem@5f155f15>
O       itemID = item002
O       orderQuantity = 15
O       inventoryQuantity = 10
O       inventoryStatus = Backorder - insufficient stock to fill o
O -- Ship object dump ends -----
O -----

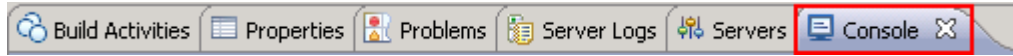
```

- 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).

\_\_\_ 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
    
```

---

**NOTE:** The message "I processMessage" may also appear, but this is produced by the system and not by the application.

---

- 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 EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@5ede5ede
O Value:
O   customerID = cust123
O   items = ShipItem[3]
O     items[0] = <ShipItem@5efb5efb>
O       itemID = item001
O       orderQuantity = 3
O       inventoryQuantity = 5
O       inventoryStatus = OK - but stock is running low
O     items[1] = <ShipItem@5f085f08>
O       itemID = item009
O       orderQuantity = 5
O       inventoryQuantity = 45
O       inventoryStatus = OK - sufficient stock levels
O     items[2] = <ShipItem@5f155f15>
O       itemID = item002
O       orderQuantity = 15
O       inventoryQuantity = 10
O       inventoryStatus = Backorder - insufficient stock to fill 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:

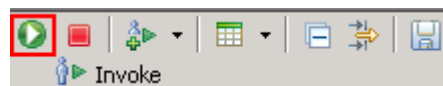
- customerID can be any string and should not have any particular affect on the results
- The items array can have any number of elements
- itemID values of "item001" through "item010" are recognized by the inventory service, all other values are not recognized but can be used.
- Inventory status will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

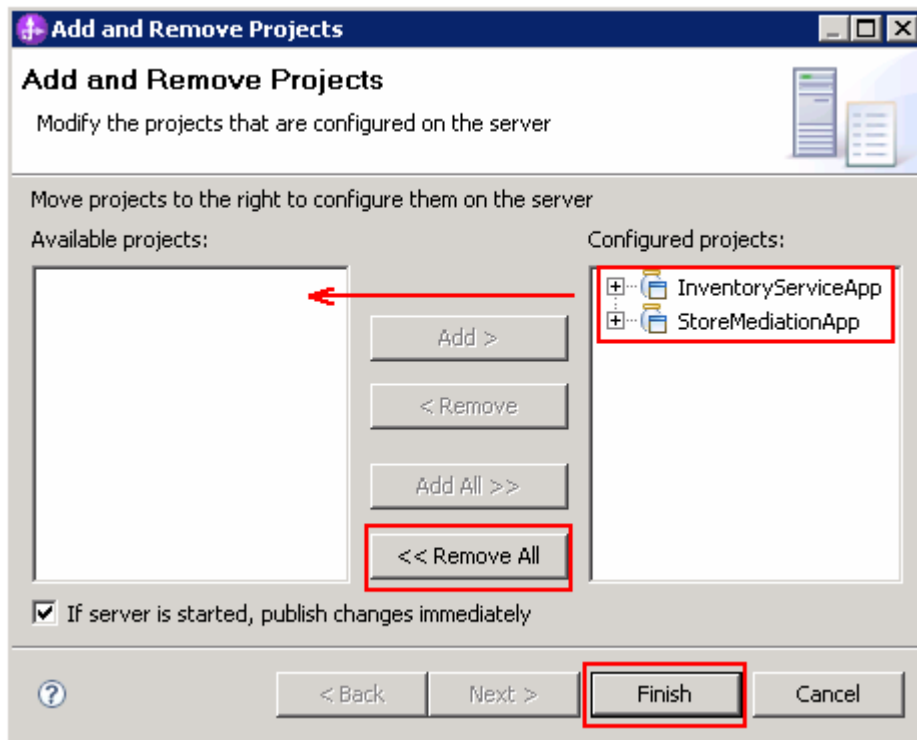
\_\_\_ c. Click **Continue** icon (  ) under Events panel



## 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 **WebSphere ESB Server v7.0** 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**

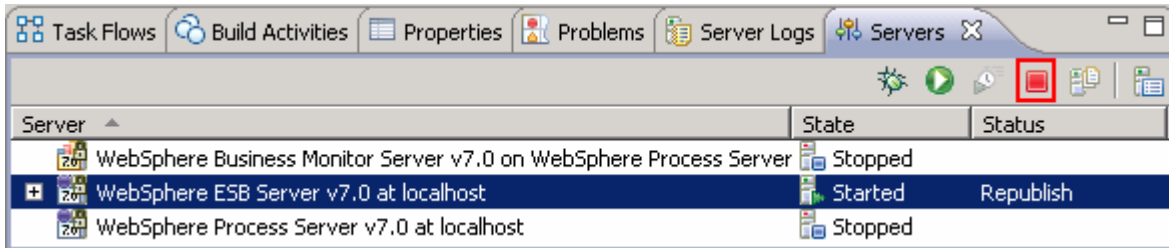


- \_\_\_ c. Click **Finish** after you see the applications moved to Available projects.
  - \_\_\_ d. Wait until the application is removed from the server
- \_\_\_ 2. Close the StoreMediation\_test panel without saving
    - -----
    - \_\_\_ a. Close the StoreMediation\_Test editor
    - \_\_\_ b. Click **No** from Save Test Trace window
    - \_\_\_ c. Close the 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 the main menu, select **File → Exit**

## 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, a message element setter primitive 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