# WebSphere Enterprise Service Bus 7.0 Augmentation, aggregation, and retry tutorial series Lab Two – Message splitting and aggregating

1
1
2
2
3
4
5
8
43
53
55
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 <u>Augmentation, aggregation and retry labs</u>. 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></wid_home>	C:\Program Files\IBM\WID70	/opt/IBM/WID70
<esb_profile_home></esb_profile_home>	<wid_home>_WTE\runtimes\bi_v7\profiles\qesb</wid_home>	<wid_home>&gt;_WTE\runtimes\bi_v7\profiles\qesb</wid_home>
<lab_files></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 staring 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 staring Lab Two: <LAB\_FILES>/workspaces/ws2

🚯 Workspace Launcher	×
Select a workspace	
IBM WebSphere Integration Developer 7.0 stores your projects in a folder called a workspace. Choose a workspace folder to use for this session.	
	Decision 1
Workspace: C:\Labfiles70\WESB\AugAggRetry\workspaces\ws2	Browse
Use this as the default and do not ask again	
ок	Cancel

- 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

🐮 Business Ir	ntegr 🛛 🤻 Patterns Expl	or		E
Integration	Solutions	Ne	<u>w</u> E	3
<u>New</u> integrati	on solution			
	New	•		
	🔁 Import			
	🛃 Expôst			_
Projects	🔊 Refresh		<u>w</u> E	3
<u>New</u> business	Add to Asset Repository			
	Properties			

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

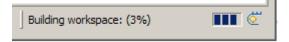
🥵 Import	
Select Import a project and its dependent projects from a Zip file.	Ľ
Select an import source:	
type filter text	
Web Web services XML Cher HTTP Project Interchange	
O < Back Next >	Finish Cancel

\_\_\_ 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

🌐 Import Project I	nterchange Contents	<u> </u>
Import Projects Import Projects from	a zip file.	<u></u>
From zip file: Project location root:	C:\Labfiles70\WESB\AugAggRetry\PI2-AugmentSolution-AggregateStart.zip	Browse
<ul> <li>✓ InventorySe</li> <li>✓ InventorySe</li> <li>✓ InventorySe</li> <li>✓ InventorySe</li> <li>✓ StoreLib</li> <li>✓ StoreMediation</li> </ul>		
Select All Desele	ct All Select Referenced	
0	< Back Next > Finish	Cancel

- \_\_\_\_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 <u>Augmentation, aggregation and retry tutorials</u> 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

1 Û U	<b>X</b>		
		□ □ ShipItem ⇒ <click filter="" to=""></click>	
		e itemID	string
		e orderQuantity	int
		e inventoryQuantit	y int
		e inventoryStatus	string
	Ŷ ₽		<ul> <li>ShipItem</li> <li>Click to filter&gt;</li> <li>itemID</li> <li>orderQuantity</li> <li>inventoryQuantit</li> </ul>

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

Integration Solutions	<u>New</u>
New integration solution	
Projects	<u>New</u>
InventoryService     InventoryService     StoreLib     StoreMediation     Service     Service     Service     Service     Dependencies     Integration Logic     Dete Types	
	Project 🕨
IIII Deen	🚔 Business Object
Сору	💕 Busines; Object From External Data
Paste	<mark>≓</mark> ∱ Other
Delete	

\_\_\_\_b. In the New Business Object dialog, provide the name TransientCtxBO2

🚯 New Business	Object 📃 🕹 🗾					
Create a Busi	Create a Business Object					
Business objects a such as a custome	are containers for application data that represent business functions or elements, er or an invoice.					
Module or library:	StoreMediation  Browse New					
Namespace:	http://StoreMediation					
Folder:	Browse					
Name:	TransientCtxBO2					
Inherit from:	<pre>cnone&gt; Browse New Clear</pre>					
?	< Back Next > Finish Cancel					

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

•Definition	Ŷ	Ŷ	*
☐ TransientCtxBO2 券 <click filter="" to=""></click>			
L			

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

▼Definition	P	Ŷ	Ŷ	×	
TransientCtxBO2 Click to filter>					
e customer string					

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

▼Definition		<u></u>	K 🛛 📰
*Demición		• · · •	
📋 TransientCtxB	02		
🔆 <click filter<="" td="" to=""><td>&gt;</td><td></td><td></td></click>	>		
e customer str	ing	_	
e item str	ing		
	string	1	•
	time	,	
		ntoryFault	http://StoreLi
	📋 📋 Inver	ntoryItem	http://StoreLi
	📋 Ordei	-	http://StoreLi
	📋 💭 Ordei	rItem	http://StoreLi
	📋 Ship		http://StoreLi
	📋 ShipI	tem	http://StoreLi
	🔲 Trans	ienkatxBO1	http://StoreM
		ientČtxBO2	http://StoreM

- \_\_\_\_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.

<b>▼D</b>	efinition	) fr 🕂 💥		
	☐ TransientCtxBO2		 □ ☐ ShipItem ⇒ <click filter="" to=""></click>	
	e customer string		e itemID	string
	e item ShipItem -		e orderQuantity	int
			e inventoryQuantity	int
			e inventoryStatus	string

\_\_\_\_i. Close the TransientCtxBO2, business object editor

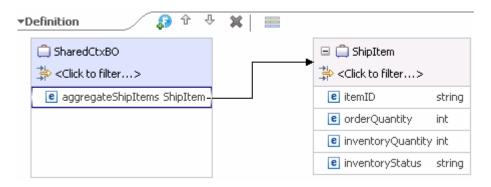
- 2. Create a new business object that is used as the shared context for the flow. The shared context defines the shared memory where the information for all of the items is aggregated during the iterative part of the flow.
  - Module : StoreMediation
  - Name : SharedCtxBO
  - Attributes: aggregateShipItems

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

ShipItem[]

SharedCtxBO	Г	🖃 📋 ShipItem	
→ → → → > <click filter="" to=""></click>		⇒io <click filter="" to=""></click>	
e aggregateShipItems ShipItem []_		e itemID	string
		e orderQuantity	int
		e inventoryQuantity	int
		e inventoryStatus	string

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

📅 Task Flows 🙆 B	uild Activities 🔲 Prop	erties 🛛	🛛 🔁 Problems	🛅 Server Logs	해 Servers		
Element - aggregateShipItems							
Description	Na <u>m</u> e:	aggregate	ShipItems		_		
Documentation	<u>T</u> ype:	ShipItem	{ http://StoreLib ]	Browse			
Application Info	Minimum occurrences:	0		M <u>a</u> ximum oco	currences:	inbounded 🔄	-

- \_\_\_\_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).

Ordering     Input	Callout ShippingPartner
PrintStartMsg	PrintEndMsg
Order2Inventory CheckInventory	Inventory2Ship

, \_\_\_\_\_\_,

\_\_\_\_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

StoreMediation >	Б
P Overview	
Operation connections 🔊 🔊	-
Select a source operation to create or edit its mediation flow implementation	
Ordering     ShippingPartner	
SubmitOrder	
InventoryPartner	
👹 checkInventory	

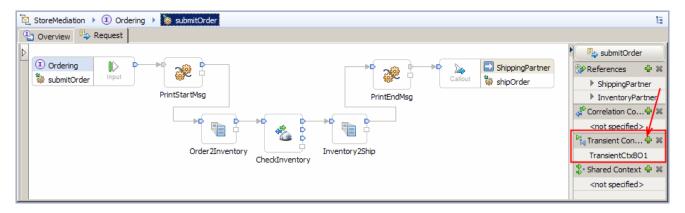
\_\_\_\_b. In the mediation flow editor, click the source operation, **submitOrder**, as shown in the picture above. This action opens the Request flow as shown here

🔯 StoreMediation 🕨 🛈 Ordering 🕨 🐞 submitOrder	E
Proverview 🖳 Request	
	submitOrder
Calout	💱 References 🛛 🖶 🕷
	ShippingPartner
PrintStartMsg PrintEndMsg	InventoryPartner
	🖉 Correlation Co 🖶 🕷
	<not specified=""></not>
	👫 Transient Con 🖶 🕷
Order2Inventory Inventory2Ship CheckInventory	TransientCtxBO1
	🐉 Shared Context 🖶 🕷
	<not specified=""></not>

- \_\_\_\_4. Configure the transient and shared contexts that are used in the flow.
  - Transient context: TransientCtxBO2
  - Shared context : SharedCtxBO

🖳 🖳 submitOrder	
🕪 References	∲ %
ShippingPartner	
InventoryPartner	
are Correlation Context	♦ ≈
<not specified=""></not>	
<sup>▶</sup> id 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).



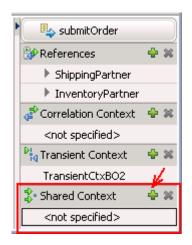
1) In the Mediation Flow on the right side, hit Add (🔤) icon next to Transient Context

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

🖶 Data Type Selection 🛛 🔹	
Filter by type, namespace, or file (? = any character, $*$ = any	String):
* N	ew
Matching data types:	
<ul> <li>InventoryFault</li> <li>InventoryItem</li> <li>Order</li> <li>OrderItem</li> <li>SharedCtxBO</li> <li>Ship</li> <li>ShipItem</li> <li>TransientCtxBO1</li> <li>TransientCtxBO2</li> </ul>	
Qualifier:	×sd
] ОК Сап	icel

3) Click OK

- \_\_\_\_b. Now, configure the shared context
  - 1) In the Mediation Flow on the right side, hit **Add** (<sup>1</sup>/<sub>2</sub>) icon next to **Shared Context**



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

🚯 Data Type Selection		_ 🗆 ×
Filter by type, namespace, or file	e (? = any character, * =	any String):
*		New
Matching data types:		
📮 InventoryFault		
🛱 InventoryItem		
🛱 Order		
🛱 OrderItem		
SharedCtxBO		
📮 Ship		
💭 ShipItem		
TransientCtxBO1		
🔲 TransientCtxBO2		
Qualifier:		
http://StoreMediation - Stor	reMediation/SharedCtxB0	D.xsd
0	ок	Cancel

#### 3) Click OK

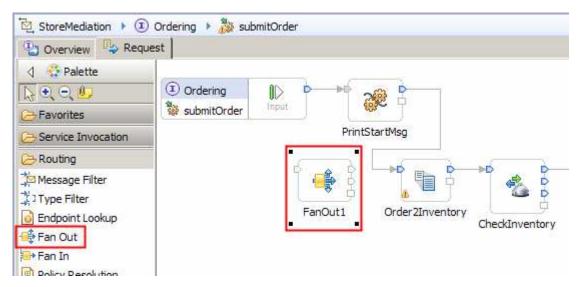
\_\_\_\_ c. From the menu, select File → Save (or Ctrl + S from your keyboard) to save your changes

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

<ul> <li>① Ordering</li> <li>② submitOrder</li> </ul>	Input	<b>2</b>
	PrintSt	tartMsg
•	StartIteration	Order2Inventory

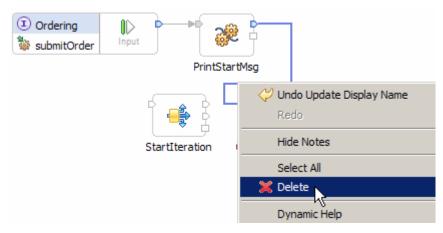
\_\_\_\_\_a. From the **Palette**, select **Routing → Fan Out** and then click the canvas as shown below (between the PrintStartMsg and Order2Inventory primitives as shown below).



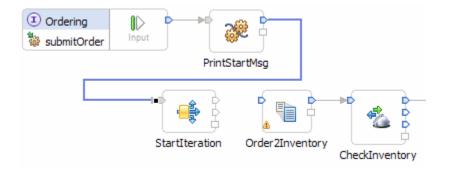
b. Ensure that the fan out primitive is selected and navigate to the **Description** panel of the **Properties** view and change the **Display name** to **StartIteration** 

🗞 Build Activities 🔲 Prope	rties 🛛 [	Problems 👸 Server Logs 👯 Servers 🛛 🗸 🖓 🗖			
📑 Fan Out : StartIteration					
Description	Display name:	StartIteration			
Terminal					
Details Description:		StartIteration			
		Create different messages from a repeating element in the input			
Promotable Properties		message. The Fan Out primitive can also be used with the Fan In 👘			
		primitive, to aggregate (combine) service responses.			

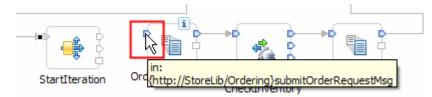
\_\_\_\_ c. Remove connection between the PrintStartMsg and Order2Inventory primitives. Right-click the line connecting the primitives and select Delete from the pop-up menu



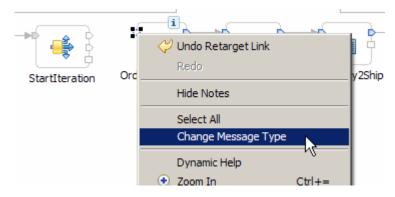
\_\_\_\_\_d. Add a connection from PrintStartMsg to StartIteration primitive. Click the out terminal of PrintStartMsg and drag to the input terminal of StartIteration



- \_\_\_\_\_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.

• Specific message type
Eilter: *
Interfaces:
Inventory
- Inventory Ordering - SubmitOrder - - - - - - - - - - - - -

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

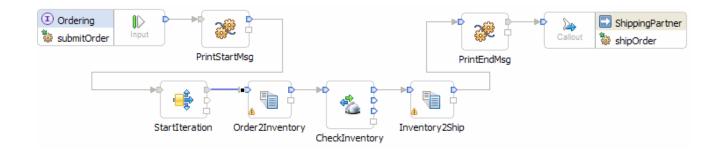
🚯 Change Message Type	×
Automatically determined message type	
This terminal type will be determined by type propagation. More	
O <u>A</u> ny message type	
This terminal will set the type of the message to be any type. You can wire terminals with different types to this terminal without needing a transformation primitive. <u>More</u>	
O Specific message type	_
<u>F</u> ilter: *	
Interfaces:	
Show all available interfaces	
Namespace: http://StoreLib/Ordering	

#### 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 **noOccurances** 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

뿝 Task Flows ⓒ Build Acti	vities 🔲 Properties 🕺  🛃 Problems 👸 Server Logs 🖗 Servers 📮 Console	~
📑 Fan Out : StartIte	eration	
Description Terminal Details	Fire output terminal with original input message O o <u>n</u> ce	
Promotable Properties	• for each element in XPath expression //body/submitOrder/order/items	Edit
	<ul> <li>Check for asynchronous responses after all messages have been fired</li> <li>Check for asynchronous responses after messages have been fired</li> </ul>	

- ------
- \_\_\_\_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:

🌐 Simple XPath Expression Bui	lder				×
Build a simple XPath Exp	ression				1
Select a field and optionally add fill	ters and co	onditions.			$\sum$
Expression:					
/body/submitOrder/order/ite	ms				
Select a field:					
<type filter="" to=""></type>					•
	rRequestM ler lerID : strir OrderIten	ng 1 [ 0n ]			•
Array Fields Where		Operator	Value	and/or (optional)	Delete
e items e itemI	D	=	10.00		
Example					
<ul> <li>Add an optional condition</li> </ul>					
Operator	Value				Delete
=					
Example					
0				ОК	Cancel

\_\_\_f. Click OK

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

Task Flows 🙆 Build Activ	vities 🔲 Properties 😫 🔀 Problems 💱 Server Logs 👯 Servers 🚍 Console	~ - 0
🖶 Fan Out : StartIte	eration	
Description Terminal	Fire output terminal with original input message O o <u>n</u> ce	
Details Promotable Properties		Edit
	<ul> <li>Check for asynchronous responses after all messages have been fired</li> <li>Check for asynchronous responses after messages have been fired</li> </ul>	

- 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

🚯 New XML M	ap		<u>- 🗆 ×</u>
Create an X XML maps are	ML Map used to transform data between source and targe	et business objects.	X
Namespace: Folder:	http://StoreMediation/xslt/Order2Inventory2	Default Browse	
Name:	Order2Inventory2		
Create a sa	ample xml input file for testing the XML Map	Ŀ\$	
(?)	< Back	lext > Finish	Cancel

- \_\_\_ 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

🖶 New XML Map	_ 🗆 🗡
Specify Message Types	
Select Input and Output Message Type	
Message Root:	
Input Message Body: submitOrderRequestMsg	Browse
Output Message Body: checkInventoryRequestMsg	Browse
Defined Contexts	
Correlation Context: 💧 No correlation context is set for this flow. Contexts can be set on the input node of the flow.	
Transient Context: {http://StoreMediation}TransientCtxBO2	
Shared Context: {http://StoreMediation}SharedCtxBO	
C Back Next > Finish	Cancel

\_\_\_\_\_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 (  $\stackrel{\longrightarrow}{\leftrightarrow}$  ) '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

Target (right side)
context/transient/customer
context/transient/item/itemID
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/FanOutContex	t/occurrence body/checkInventory/orderItem
•	ween the source and target SMOs that will not change
	ween the source and target SMOs that will not change
1) Click 'Map source to target base	d on name and types' icon (++)
•Order2Inventory2	

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:

•Order2Inventory2	2 🔊 🍋 🛱 🗶	j⊳ı j¢	🖓 🖳		<b>i</b> iii 🗈	
🖃 🔁 smo					🖃 💽 smo	
⇒ <click filter<="" td="" to=""><td>&gt;</td><td></td><td></td><td></td><td>♣ <click filter<="" td="" to=""><td>.&gt;</td></click></td></click>	>				♣ <click filter<="" td="" to=""><td>.&gt;</td></click>	.>
🛨 🖻 context	[11] <anonymous></anonymous>	Loc	cal map 🔻 🎽	2	🛨 🖻 context	[11] <anonymous></anonymous>
	[11] HeadersType	Loc	cal map 👻 🖄	<u> </u>	🛨 🖻 headers	[11] HeadersType
					표 🖻 body	[01] checkInventoryRequestMsg
🕀 🖻 body	[01] submitOrderRequestMsg				🗄 🖻 attachments	[0*] AttachmentType
🕀 🖻 attachments	[0*] AttachmentType	Fo	<u>ir each</u> 🔻 🎽	-		L J

- \_\_\_\_\_ 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.
  - 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

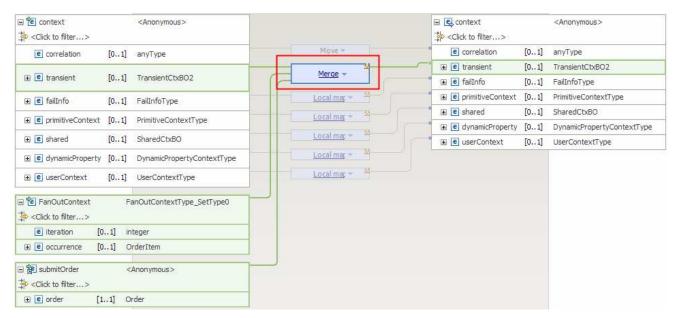
🖃 🔁 smo								🖃 💽 smo		
$\Rightarrow_{i}$ <click filter="" to=""></click>				г				$\Rightarrow$ <click filter="" to=""></click>		
e context	[11]	<anonymous></anonymous>		7	Merge 👻		-	🛨 🖻 context	[11]	<anonymous></anonymous>
	[1012]	( anony mode)		T,		ſ	•		[11]	HeadersType
e correlation	[01]	anyType						€ body	[01]	checkInventoryRequestMsg
🗉 🖻 transient	[01]	TransientCtxBO2					_		[0*]	AttachmentType
🗉 🖻 failInfo	[01]	FailInfoType								
🖃 🖻 primitiveContext	[01]	PrimitiveContextType								
🗄 🖻 EndpointLookupContext	[0*]	EndpointLookupContextType								
🗄 🖻 FanOutContext	[01]	FanOutContextType	$\vdash$							
	[01]	WTXContextType								
🗄 🖻 shared	[01]	SharedCtxBO	1							
🗄 🖻 dynamicProperty	[01]	DynamicPropertyContextType								
	[01]	UserContextType								
🗄 🖻 headers	[11]	HeadersType			Local mar 👻 💆					
🖃 🖻 body	[01]	submitOrderRequestMsg	1							
🗄 📌 submitOrder	[11]	<anonymous></anonymous>	<u> </u>							
∎ e attachments	[0*]	AttachmentType	]		<u>For each</u> ▼ <sup>28</sup>					

5) The resulting map should look like this:

6) On the **Merge** transform, click the **Edit** ( $\stackrel{1}{\cong}$ ) 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** (<sup>14</sup>) icon to navigate into the merge. It will look like this:

transient		TransientCtxBO2				transient		TransientCtxB
Click to filter> e customer	[01]	string		Move 🔻		Click to filter>	[01]	string
. € item		ShipItem		Local map 🔻	<u>¥</u>	🗈 🖻 item	[01]	ShipItem
□ PanOutContext		FanOutContextType_SetType0						
e iteration	[01]	integer						
∎ e occurrence	[01]	OrderItem	]					
🖃 🄁 submitOrder		<anonymous></anonymous>						
♣ <click filter="" to=""></click>								
主 🖻 order	[11]	Order						

- 11) Select both the **Move** and **Local map** transformations (hold down Ctrl button on keyboard and click each transformation) and hit **Delete selected elements** (X) 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:

🔁 transient		TransientCtxBO2	🖃 💽 transient		
<click filter="" to=""></click>			Click to filter>		
e customer	[01]	string	e customer	[01]	
🗉 🖻 item	[01]	ShipItem	🖃 🖻 item	[01]	
FanOutContext		FanOutContextType_SetType0	e itemID	[01]	
Click to filter>		Panoutcontextrype_setrypeo	e orderQuantity	[01]	i
	F0 41		e inventoryQuantity	[01]	i
e iteration		integer	e inventoryStatus	[01]	s
e occurrence	[01]	OrderItem			
e itemID	[01]	string			
e quantity	[01]	int			
<b>診</b> submitOrder		<anonymous></anonymous>			
Click to filter>					
🖃 🖻 order	[11]	Order			
e customerID	[01]	string			
🗉 🖻 items	[0*]	OrderItem			

- 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.

- A transient		Transie 405 800					Transisation 000
🖃 🔁 transient		TransientCtxBO2			🖃 💽 transient		TransientCtxBO2
Click to filter>					⇒ <click filter="" to=""></click>		
e customer	[01]	string		C	e customer	[01]	string
\pm 🖻 item	[01]	ShipItem			🖃 🖻 item	[01]	ShipItem
				t d	e itemID	[01]	string
FanOutContext		FanOutContextType_SetType0			e orderQuantity	[01]	int
⇒ <click filter="" to=""></click>					e inventoryQuantity	[01]	int
e iteration	[01]	integer			e inventoryStatus	[01]	string
e occurrence	[01]	OrderItem	Move 🔻			[0114]	sung
e itemID	[01]	string					
e quantity	[01]	int					
🖃 🏂 submitOrder		<anonymous></anonymous>					
$\stackrel{\text{\tiny $1$}}{\Rightarrow} \stackrel{\text{\scriptsize $1$}}{<} Click to filter>$							
🖃 🖻 order	[11]	Order					
customerID	[01]	string	Move 🔻				
	[0*]	OrderItem					

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

20) Click '**Up a level**' (<sup>1</sup>) icon which will bring you one level up to the context merge mapping.

🖳 Order2Inventory2 👌 😫 context 🎽 😫 transient	
▼Order2Inventory2	
🖃 🔁 transient TransientCtxBO2	
→i →i <	Click to filter>
e customer [0 1] string	📑 🖪 austomer 🛛 10 11 string

- 21) Click '**Up a level**' (<sup>1</sup>) 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.

🖃 🔁 smo						6	🗉 🖳 smo		
→ <click filter="" to=""></click>							*↓ ↓↓ <click filter="" to=""></click>		
e context	[11]	<anonymous></anonymous>	1	Merge 👻		•	e context	[11]	<anonymous></anonymous>
	[1.1.1]	Shiriyindasz				•	e headers	[11]	HeadersType
e correlation	[01]	anyType				6	e body	[01]	checkInventoryRequ
	[01]	TransientCtxBO2					🖃 🛃 checkInventory	[11]	<anonymous></anonymous>
⊕ e failInfo	[01]	FailInfoType			Ċ	-1	표 🖻 orderItem	[11]	OrderItem
e primitiveContext	[01]	PrimitiveContextType				<b>–</b>	e attachments	[0*]	AttachmentType
	[0*]	EndpointLookupContextType							
🖃 🖻 FanOutContext	[01]	FanOutContextType_SetType0							
e iteration	[01]	integer							
e occurrence     e	[01]	OrderItem		Move 🔻					
	[01]	WTXContextType	1						
	[01]	SharedCtxBO	1						
🗄 🖻 dynamicProperty	[01]	DynamicPropertyContextType	1						
∎ e userContext	[01]	UserContextType	1						
• e headers	[11]	HeadersType		Local mar 🔻 💆					
= e body	[01]	submitOrderRequestMsg	1						
🗉 📌 submitOrder	[11]	<anonymous></anonymous>	1						
± e attachments	[0*]	AttachmentType	1	<u>For each</u> ▼ <sup>2</sup>					

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.

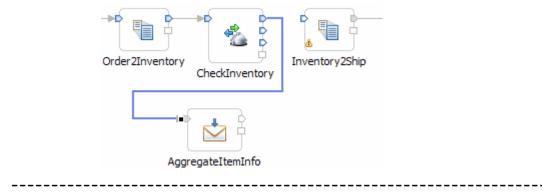
AggregateItemInfo

Display Name:

:

• Wire

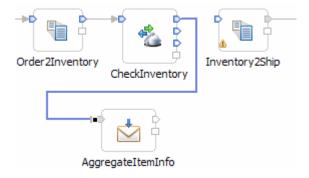
CheckInventory(out terminal) to AggregateItemInfo



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

🕒 Overview 🖳 Reque	st
Overview     Verview     Acque     Palette     Pavorites     Service Invocation     Routing     Transformation     XSL Transformation	Image: Solution of the second seco
Business Object Map Message Element Se Set Message Type Database Lookup	StartIteration Order2Inventory Inventory2Ship
Custom Mediation	MessageElementSetter 1

- \_\_\_\_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 rightclicking 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/inventoryQuantit
	2	Copy element /body/checkInventoryResponse/inventoryItem/status to element /context/transient/item/inventoryStatus
	3	Append /context/transient/item to element /context/shared/aggregateShipItems
- 1		

• ------

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

🚯 Simple XPath Expre	ssion Buil	der				×		
Build a simple XPath Expression 🛛 🔧 🖉								
Select a field and optionally add filters and conditions.								
Expression:								
/context/transient/it	em/inver	nto <mark>ry</mark> Qua	antity					
Select a field:								
<type filter="" to=""></type>						•		
<pre></pre>								
Array Fields	Where		Operator	Value	and/or (optional)	Delete		
Example								
✓ Add an optional condition								
Operator Value Delet								
Example								
0					ОК	Cancel		

- 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:

🚯 Add/Edit			
Add/Edit Configure th	P <b>roperties</b> e properties for the Message Element Setter		
<u>A</u> ction:	Сору		-
Ta <u>r</u> get:	/context/transient/item/inventoryQuantity		Br <u>o</u> wse
<u>T</u> ype:	{http://www.w3.org/2001/XMLSchema}int		Bro <u>w</u> se
<u>S</u> ource:	/body/checkInventoryResponse/inventoryItem/inStockQuanti	ty	Browse
?		Einish	Cancel

- 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:

	Properties he properties for the Message Element Setter	
<u>A</u> ction:	Сору	•
Ta <u>r</u> get:	/context/transient/item/inventoryStatus	Br <u>o</u> wse
<u>T</u> ype:	{http://www.w3.org/2001/XMLSchema}string	Bro <u>w</u> se
<u>S</u> ource:	/body/checkInventoryResponse/inventoryItem/status	Browse
0	<u> </u>	Cancel

#### 6) Click Finish

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

#### Message Elements:

2 Copy element /body/checkInventoryResponse/inventoryItem/status to element /context/transient/item/inventoryStatus	oryQuantity
2 Copy element /body/crieckinventorykesponse/inventory/tem/status to element /context/d ansient/item/inventorystatus	;

- \_\_\_\_ 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:

<u>A</u> ction:	Append	•
Target:	/context/shared/aggregateShipItems	Br <u>o</u> wse
<u>T</u> ype:	{http://StoreLib}ShipItem	Bro <u>w</u> se
<u>S</u> ource:	/context/transient/item	Brows <u>e</u>

#### 6) Click Finish

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

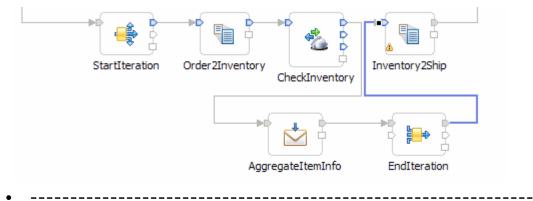
#### Message Elements:

	Copy element /body/checkInventoryResponse/inventoryItem/inStockQuantity to element /context/transient/item/inventoryQuantity
2 Co	copy element /body/checkInventoryResponse/inventoryItem/status to element /context/transient/item/inventoryStatus
3 Ap	uppend /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

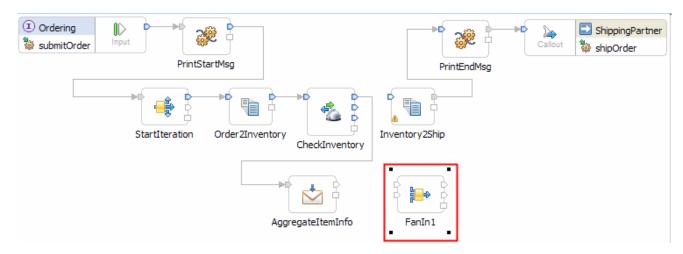


- \_\_\_\_a. Add a new Fan In to the canvas:
  - 1) From the **Palette**, select **Routing** → **Fan In** and then click the canvas to the right of the AggregateitemInfo primitive
  - 2) A Fan Out Primitive Selection dialog is opened. Select **StartIteration** as the associated Fan Out.

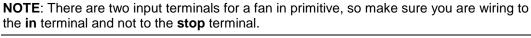
🔒 Fan Out Primitive Selection		_ 🗆 🗵
Select a Fan Out primi	tive:	
<create a="" fan<br="" new="">StartIteration</create>	Out primitive>	
0	ок	Cancel

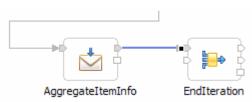
#### 3) Click OK

4) You will now see a new Fan In primitive added on the canvas.



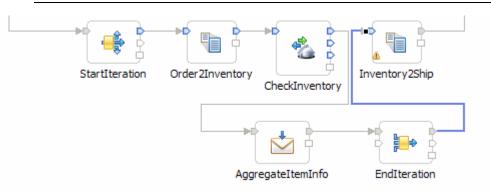
- b. Ensure that this primitive is selected and then navigate to the **Description** panel of the **Properties** view
- \_\_\_ c. Change the **Display name** to **EndIteration**
- \_\_\_\_\_d. Add a connection from the **AggregateItemInfo** primitive to the **in** terminal of the **EndIteration** primitive. Click the **out** terminal and drag it to the **in** (uppermost) terminal.





\_\_\_\_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		
O 1 input messages have been received		
O XPath expression evaluates to true Edit		
the associated Fan Out primitive has iterated through all messages		
Enable timeout		
Fire incomplete terminal if the output terminal has not fired after 10 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

Build Activities 🔲 Proper	rties 🛛 🔀 Problems) 🗑 Server Logs 👯 Servers 📄 💎 🗖 🖡
⊨ Fan In : EndIterat	ion
Description Terminal Details Promotable Properties	Associated Fan Out: StartIteration Fire output terminal with original input message once for each element in XPath expression /body/submitOrder/order/items Edit Check for asynchronous responses after all messages have been fired Check for asynchronous responses after 1 messages have been fired Fan In Fire output terminal when 1 input messages have been received XPath expression evaluates to true Edit Edit the associated Fan Out primitive has iterated through all messages Fire incomplete terminal if the output terminal has not fired after 10 seconds

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

🚯 New XML Ma	ар	
Create an XI XML maps are u	s.	
Namespace:	http://StoreMediation/xslt/Inventory2Ship2	
Folder:	xslt Browse	
Name:	Inventory2Ship2	
Create a sa	mple xml input file for testing the XML Map	•
0	< Back Next > F	inish Cancel

- \_\_\_ 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

🚯 New XML Map							
Specify Message	Types						
Select Input and Outpu	t Message Type						
Message Root:	1						
Input Message Body:	checkInventoryResponseMsg	Browse					
O <u>u</u> tput Message Body:	utput Message Body: shipOrderRequestMsg						
Defined Contexts							
Correlation Context:	No correlation context is set for this flow. Contexts can be set on the input node of the flow.						
Transient Context:	ansient Context: {http://StoreMediation}TransientCtxBO2						
Shared Context:	{http://StoreMediation}SharedCtxBO						
0	< <u>B</u> ack Next > Einish	Cancel					

\_\_\_\_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 (  $\stackrel{\longrightarrow}{\leftrightarrow}$  ) "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 shipItems 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[ ]

In 🗄

#### ▼Inventory2Ship2 🖉 💭 💭 🐺 🖉 🎼

□ 陰? smo		⊡ <b>⊑</b> \$ smo	
🖃 🤨 context	ContextType1	Inline map 👻 🕈 🗈 🖻 context	ContextType1
	anyType	te headers	HeadersType
🖃 🖻 transient	TransientCtxBO2	🖃 🖻 e? body	shipOrderRequestMsg
e? customer	string	Move 🔻 🖃 🖻 shipOrder	ShipOrderType
	-	🖃 🖻 ship	Ship
	ShipItem	e? customerID	string
🗄 🖻 failInfo	FailInfoType	∎ et items	[] ShipItem
	PrimitiveContextType		[] Shiptteni
🖃 🖻 e? shared	SharedCtxBO		
	[] ShipItem	Move 🔻	
	HeadersType	Inline map 🔻 🖄	
<b>⊞ e?</b> body	checkInventoryResponseMsg		

\_\_\_\_\_

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

▼Inventory25hip2	🖗 🎼 🛱 🚺 👘 🗍	🗠 🛛 🖓 🖳 🌇 🛱	8	
🖃 🔁 smo			🖃 💽 smo	
⇒ <click filter<="" td="" to=""><td>&gt;</td><td></td><td>🔆 <click filter<="" td="" to=""><td>&gt;</td></click></td></click>	>		🔆 <click filter<="" td="" to=""><td>&gt;</td></click>	>
🗉 🖻 context	[11] <anonymous></anonymous>	Local map 👻	🗉 🖻 context	[11] <anonymous></anonymous>
	[11] HeadersType	Local map 👻 💆	🗄 🖻 headers	[11] HeadersType
			🛨 🖻 body	[01] shipOrderRequestMsg
🕀 🖻 body	[01] checkInventoryResponseMsg		🗄 🖻 attachments	[0*] AttachmentType
🗄 🖻 attachments	[0*] AttachmentType	For each 🔻 💆		four Tracaciment type

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

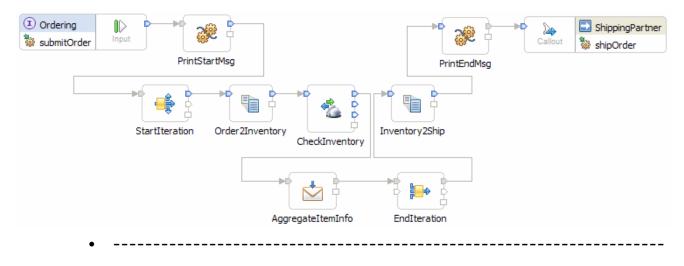
- 1) In the source smo, expand **context**  $\rightarrow$  **transient**, revealing **customer**
- 2) In the target smo, expand **body**  $\rightarrow$  **shipOrder**  $\rightarrow$  **ship**, revealing **customerID** and **items**
- 3) Click **customer** in the source smo and drag it to **customerID**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

•Inventory2Ship2	💷 💷 🗮 🗣 🕼 🕼	🔉 🔍 🌇 🛱 🕯	6	0-	
🖃 🔁 smo				🖃 🖳 smo	
Click to filter>				♣ <click filter="" to=""></click>	
🖃 🖻 context	[11] <anonymous></anonymous>	Local map 🔻 💆	-	🗉 🖻 context	[11] <anonymous></anonymous>
e correlation	[01] anyType	ĺ	-	표 🖻 headers	[11] HeadersType
🖃 🖻 transient	[01] TransientCtxBO2			🖃 🖻 body	[01] shipOrderRequestMsg
e customer	[0.,1] string	Move 🔻 🗋		🖃 📌 shipOrder	[11] <anonymous></anonymous>
				🖃 🖻 ship	[11] Ship
E item	[01] ShipItem		-1	e customerII	D [01] string
🕀 🖻 failInfo	[01] FailInfoType			표 🖻 items	[0*] ShipItem
🗉 🖻 primitiveContex	t [01] PrimitiveContextType		1		[0*] AttachmentType
🗄 🖻 shared	[01] SharedCtxBO				[0] Attachmentrype
표 🖻 dynamicPropert	y [01] DynamicPropertyContextType				
표 🖻 userContext	[01] UserContextType				
🛨 🖻 headers	[11] HeadersType	Local map 🔻 💆			
🗄 🖻 body	[01] checkInventoryResponseMsg				
🗉 🖻 attachments	[0*] AttachmentType	For each 👻			

- \_\_\_\_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.

□ 1 smo ⇒ <click filter="" to=""></click>			□ Smo ⇒ <click filter="" to=""></click>	
🖃 🖻 context	[11] <anonymous></anonymous>	Local map 👻	🖅 🗈 context	[11] <anonymous></anonymous>
e correlation	[01] anyType	ſ	∎ 🖻 headers	[11] HeadersType
🖃 🖻 transient	[01] TransientCtxBO2		🖃 🖻 body	[01] shipOrderRequestMsg
e customer	[01] string	Move 🔻	🖃 📌 shipOrder	[11] <anonymous></anonymous>
			🖃 🖻 ship	[11] Ship
🕀 🖻 item	[01] ShipItem		e customer	ID[01] string
🕀 🖻 failInfo	[01] FailInfoType		🛨 💽 items	[0*] ShipItem
🗄 🖻 primitiveContext	[01] PrimitiveContextType		■ e attachments	
🖃 🖻 shared	[01] SharedCtxBO		E attachments	[0*] AttachmentType
🗄 🖻 aggregateShipIter	ns [0*] ShipItem	Move 🔻		
🗉 🖻 dynamicProperty	[01] DynamicPropertyContextType			
🗉 🖻 userContext	[01] UserContextType			
🗄 🖻 headers	[11] HeadersType	Local map 👻		
🗄 🖻 body	[01] checkInventoryResponseMsg			
🗄 🖻 attachments	[0*] AttachmentType	For each 🔻 💆		

- \_\_\_\_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)

① Ordering       Input       Imput
<ul> <li>Possible layout done by moving individual primitives. This is showing the iterative loop in the lower row of the flow.</li> </ul>
Imput       Imput <td< td=""></td<>
StartIteration Order2Inventory CheckInventory AggregateItemInfo EndIteration
•
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 <b>File</b> → <b>Save</b> (or <b>Ctrl + S</b> 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:

📅 Task Flows 🕝 Build Activities 🔲 Properties 🚼 Problems 🕱 👔 Server Logs 👯 Servers 🚍 Console								
0 errors, 6 warnings, 0 others								
Description A								
🗄 🍐 Warnings (6 items)								

### 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 <u>Augmentation, aggregation and retry tutorials</u> 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 (12) from the toolbar

꿈 Task Flows	Build Activities	Properties	🔁 Problems	🛅 Server Logs 🕀 Server	rs 🕅	於 🕥	S 🗉 🕑 🔚 🗖
Server -				State		Status	
📆 WebSp	🗱 WebSphere ESB Server v7.0 at localhost			a Stopped			
78 WebSp	here Process Serve	r v7.0 at localho	st	🛅 Stopped			

\_ 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

🔀 Task Flows 🔞 Build Activities 🔲 Properties 🔝 Pro	blems 🔞 Server L	ogs 👭 Servers 🕱 📮 Console 🕸 🜔 🖉 🔳 📳 🔚	
Server 🔺	State	Status	
WebSphere ESB Server v7.0 at localhost	Ъ Started	Republish	
🗉 🛅 InventoryServiceApp	🖡 Started	Synchronized	
🗉 🛅 StoreMediationApp	🖡 Started	Republish	
WebSphere Process Server v7.0 at localhost	E Stopped		

- \_\_\_\_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

🌐 Add and Remove Pro	ojects	
Add and Remove I Modify the projects that	Projects are configured on the server	
Move projects to the right	t to configure them on the serve	er
Available projects:		Configured projects:
⊞… 🖶 InventoryService		>
🔽 If server is started, p	ublish changes immediately	
0	< Back Next >	Finish Cancel

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

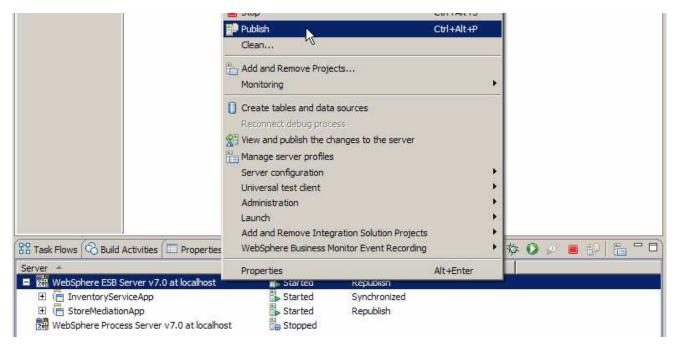
4) Wait while the projects are being published to the server

🔀 Task Flows 🔞 Build Activities 🔲 Properties 🔝 Pro	blems 🔞 Server	Logs 위상 Servers	🛛 📮 Console	参 🔘 🖗	S 🔳 🕑 📙 🗖 🗖
Server 🔺	State	Status			
WebSphere ESB Server v7.0 at localhost	🖡 Started	Publishing			
🕀 📑 InventoryServiceApp					
표 🛅 StoreMediationApp					
🔀 WebSphere Process Server v7.0 at localhost	🖥 Stopped				

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

😤 Task Flows 🕜 Build Activities 🔲 Properties 🔝 P	roblems 🔞 Server	.ogs 👭 Servers 🛛 📮 Console 🎋 🚺 🖉 🖺 🔛 🖡
Server -	State	Status
E 🔀 WebSphere ESB Server v7.0 at localhost	🖥 Started	Synchronized
🗄 🧮 InventoryServiceApp	🚡 Started	Synchronized
🗉 🛅 StoreMediationApp	🔓 Started	Synchronized
🔀 WebSphere Process Server v7.0 at localhost	E Stopped	

\_\_\_\_\_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

🔡 Task Flows 🙆 Build Activities 🔲 Properties 🔝 Pr	oblems 🔞 Server L	.ogs 👭 Servers 🛛 📮 Console) 🎋 💽 🖉 📕 😳 🖡 🖓 🕻
Server 🔺	State	Status
WebSphere ESB Server v7.0 at localhost	🔓 Started	Publishing
🗉 🛅 InventoryServiceApp	🖡 Started	Synchronized
🕀 🛅 StoreMediationApp	🖡 Started	Republish
🔀 WebSphere Process Server v7.0 at localhost	🛅 Stopped	

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

erver 🔺	State	Status
🛚 🧱 WebSphere ESB Server v7.0 at localhost	🖁 Started	Synchronized
🗉 🦷 InventoryServiceApp	🚡 Started	Synchronized
표 🛅 StoreMediationApp	Started	Synchronized

\_\_\_\_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

StoreMediation - Assembly Diagram	🔁 StoreMediation 🛛 目	*StoreMediation_Test			
b. <b>Yes, it is open</b> . Click the Invoke icon (					
	条 •   🖩 •   🖻 🆆 🔛				

Invoka

Invoke

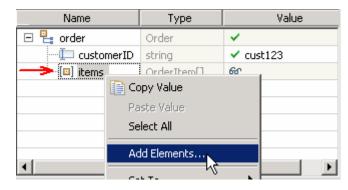
- \_\_\_\_ c. **No, it is not open**. From the assembly diagram for StoreMediation, start Test Component for the StoreMediation component
  - 1) In the Business Integration window, expand **StoreMediation** and double-click **Assembly Diagram** to open it in Assembly editor
  - From the StoreMediation-Assembly 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	Туре	Value
🖃 🔚 order	Order	<b>v</b>
🖳 🛄 customerID	string	🗸 cust123
É…[□] items	OrderItem[]	66
🚊 🖳 📴 items[0]	OrderItem	×
İlə İtem ID	string	🗸 item001
🛄 🛄 quantity	int	<b>√</b> 3
🚊 🖳 items[1]	OrderItem	¥
İtem ID	string	🗸 item009
🛄 🛄 quantity	int	✓ 5
🗄 🖳 items[2]	OrderItem	¥
İtem ID	string	🗸 item002
🛄 quantity	int	🖌 15
·		

- \_\_\_\_a. Enter these values into Initial request parameters table:
  - 1) For customerID, click under Value and enter cust123

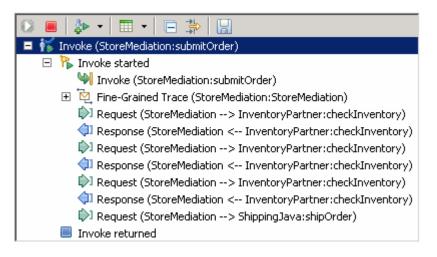
§ g □		
Name	Туре	Value
🋂 order	Order	✓
ustomerID	string	cust123
[III] items	OrderItem[]	Press 'Alt+Enter' to add lines

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



- 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 (<sup>()</sup>). 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 (<sup>1</sup>) 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

🚯 Deployment Location	
Select a Deployment Location	
This server instance is currently running.	
Deployment location:	
WebSphere Process Servers      WebSphere Business Monitor Server v7.0 on WebSphere Proc      WebSphere Process Server v7.0 at localhost      WebSphere Enterprise Service Bus Server      WebSphere ESB Server v7.0 at localhost	New Server
Mode: Run	
? <u>E</u> inish	Cancel

- \_\_\_ 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.

🥵 User Login - Default Module Test 🛛 🛛 🔀
Security is enabled on the selected runtime environment(s). Sign in to continue the test.
User ID:*
Password:*
Use the authentication settings in the preference and never ask again

\_\_\_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 ********* START mediation flow *********
0 ***
Т
   processMessage
0 ***** InvWorks - returning InventoryItem for itemID = item001
Т
   processMessage
Т
   processMessage
0 ***** InvWorks - returning InventoryItem for itemID = item009
Т
   processMessage
т
   processMessage
0 ***** InvWorks - returning InventoryItem for itemID = item002
Т
  processMessage
0 ***
O ********* END mediation flow ************
0 -----
0 -- Ship object dump begins ------
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@5ede5ede
O Value:
0
     customerID = cust123
Ο
     items = ShipItem[3]
0
        items[0] = <ShipItem@5efb5efb>
Ο
           itemID = item001
Ο
           orderQuantity = 3
0
           inventoryQuantity = 5
0
           inventoryStatus = OK - but stock is running low
Ο
        items[1] = <ShipItem@5f085f08>
0
           itemID = item009
0
           orderQuantity = 5
0
           inventoryQuantity = 45
Ο
           inventoryStatus = OK - sufficient stock levels
0
        items[2] = <ShipItem@5f155f15>
           itemID = item002
o
0
           orderQuantity = 15
0
           inventoryQuantity = 10
0
           inventoryStatus = Backorder - insufficient stock to fill o:
0 -- Ship object dump ends
                        _____
0 -----
      _____
```

\_\_\_\_a. Check to see if the tab for the **Console** view is showing (lower right quadrant of perspective)

🗟 Build Activities 🔲 Properties 🚼 Problems 🔚 Server Logs 🖓 Servers 📃 Console 🛛 🔪

\_ b. If it is not showing, open it by selecting menu items **Window**  $\rightarrow$  **Show View**  $\rightarrow$  **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:

• 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:

• 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.

```
0 -----
0 -- Ship object dump begins ------
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@5ede5ede
O Value:
     customerID = cust123
0
0
     items = ShipItem[3]
0
         items[0] = <ShipItem@5efb5efb>
0
            itemID = itemOO1
0
            orderQuantity = 3
0
            inventoryQuantity = 5
0
            inventoryStatus = OK - but stock is running low
         items[1] = <ShipItem@5f085f08>
0
0
            itemID = item009
0
            orderQuantity = 5
0
             inventoryQuantity = 45
0
             inventoryStatus = OK - sufficient stock levels
0
         items[2] = <ShipItem@5f155f15>
0
            itemID = item002
0
            orderQuantity = 15
0
            inventoryQuantity = 10
0
            inventoryStatus = Backorder - insufficient stock to fill o
                          _____
0 -- Ship object dump ends
0 -----
```

• -----

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 (**W**) 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

🤀 Add and Remove Projects	
Add and Remove Projects Modify the projects that are configured on the server	
Move projects to the right to configure them on the serve	er
Available projects:	Configured projects:
Add > < Remove Add All >> << Remove All	
✓ If server is started, publish changes immediately	
Back Next >	Finish Cancel

- \_\_\_\_ 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 (I) from the toolbar

🔀 Task Flows ᢙ Build Activities 💷 Properties 🔝 Problems 🔞 Server Lo	gs 👫 Servers 🛽	3 - 8
	参 🜔	🖉 🔳 🐏 🔚
Server 🔺	State	Status
😹 WebSphere Business Monitor Server v7.0 on WebSphere Process Server	遣 Stopped	
💌 🧱 WebSphere ESB Server v7.0 at localhost	🛼 Started	Republish
🗱 WebSphere Process Server v7.0 at localhost	遣 Stopped	

- \_\_\_\_b. Wait until the server Status shows as Stopped
- 4. Exit from WebSphere Integration Developer.

### , \_\_\_\_\_

\_\_\_\_a. From the main menu, select File  $\rightarrow$  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 <u>Augmentation, aggregation and retry tutorials</u> 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