WebSphere Enterprise Service Bus 7.0 Augmentation, aggregation, and retry tutorial series Lab One – Message augmentation

What this exercise is about

The objective of this lab is to provide you with an understanding of how to use the service invoke mediation primitive to augment the contents of a message within a mediation flow.

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 service invoke mediation primitive to call a service from within a mediation flow
- Use the returned values from a service invoke mediation primitive to augment the contents of a message

Introduction

This lab exercise is the first of 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

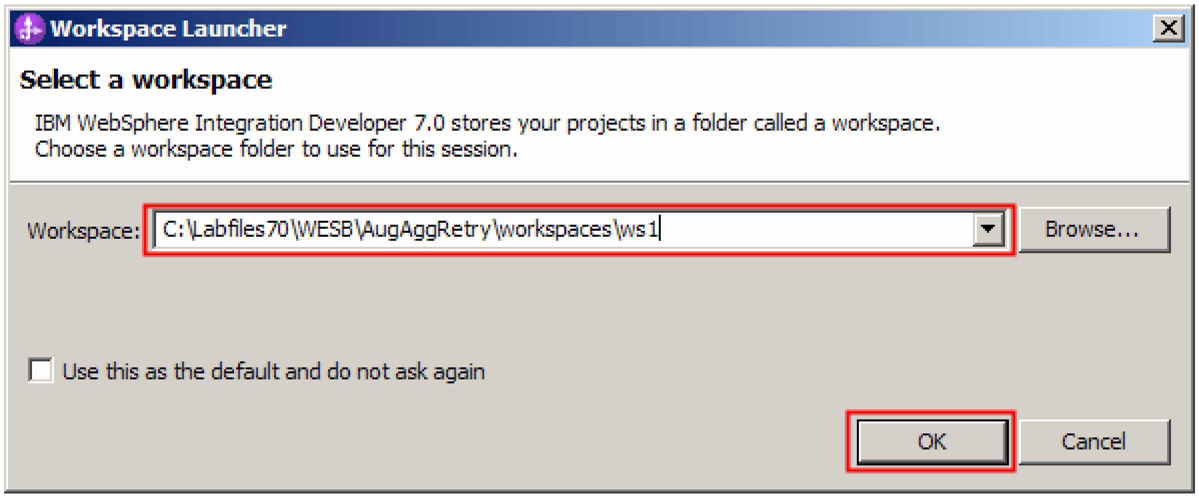
What you will do in this part: This part of the lab gets the environment ready to do the lab exercise.

____ 1. Start WebSphere Integration Developer, preferably using a new workspace

- **`Suggested location: <LAB_FILES>\workspaces\ws1`**
- **`-------------------------------------------------------------------`**

__ 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 **<LAB_FILES>/workspaces/ws1** for the 'Workspace' field and hit **OK**



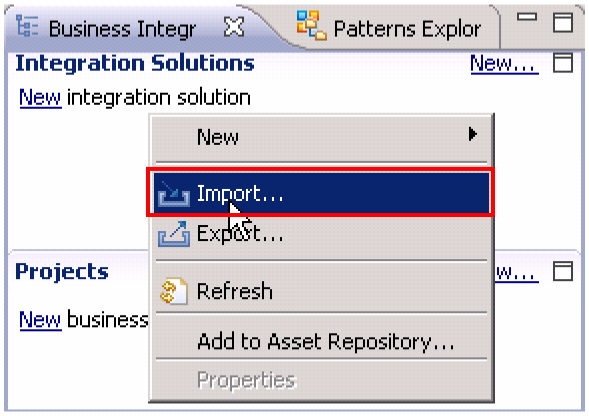
IBM WebSphere Integration Developer 7.0 stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Labfiles70\WESB\AugAggRetry\workspaces\ws1

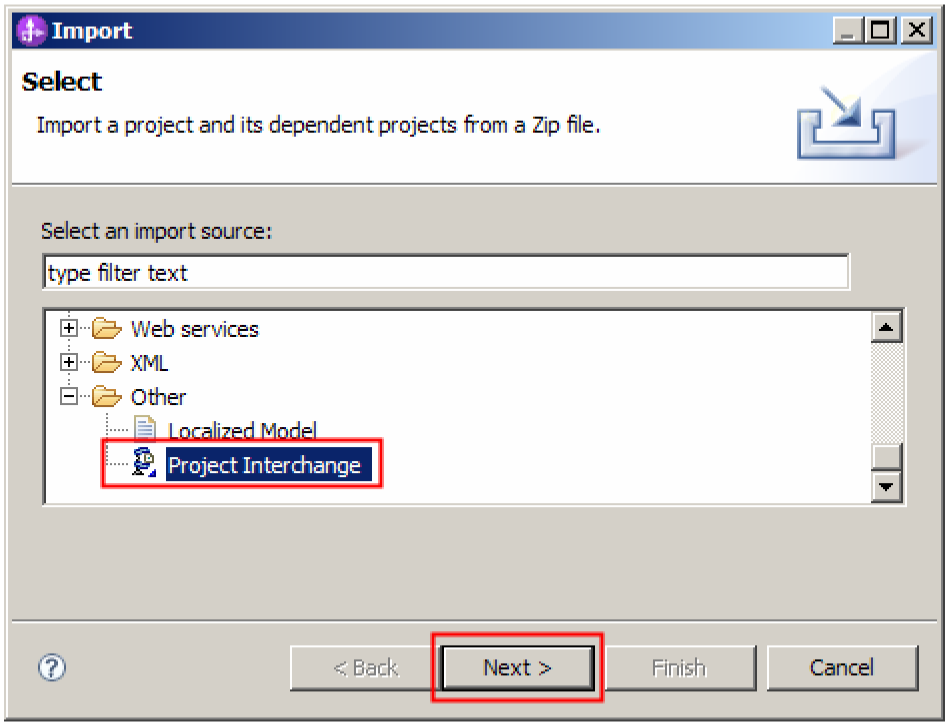
____ 2. Import Project Interchange file, **PI1-AugmentStart.zip**, into the development environment

- **`<LAB_FILES>/PI1-AugmentStart.zip`**
- **`-------------------------------------------------------------------`**

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

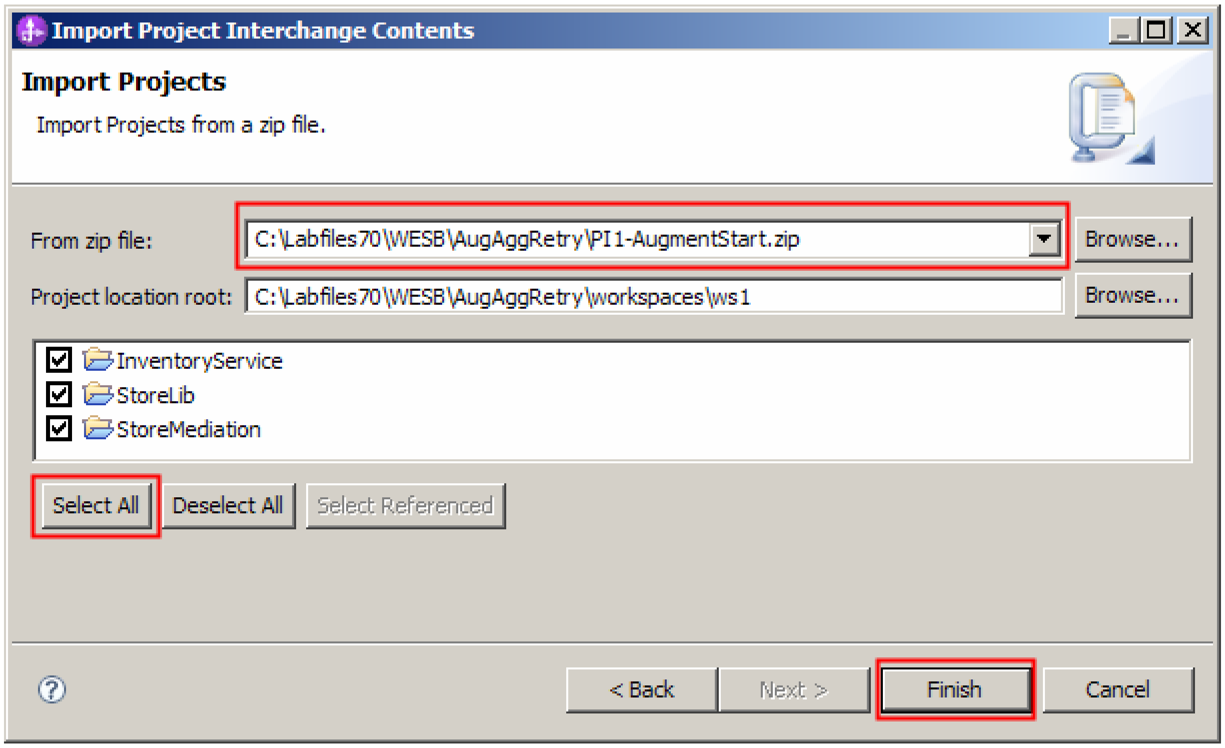


__ b. In the **Import** dialog, expand **Other** and select **Project Interchange** from the list



__ c. Click **Next**

__ d. In the next panel, click the **Browse** button for **From zip file** and navigate to **<LAB_FILES>/PI1-AugmentStart.zip** and click **Open**



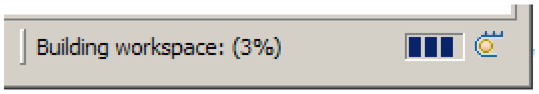
From zip file: C:\Labfiles70\WESB\AugAggRetry\PI1-AugmentStart.zip

Project location root: C:\Labfiles70\WESB\AugAggRetry\workspaces\ws1

- [✓] InventoryService
- [✓] StoreLib
- [✓] StoreMediation

__ 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



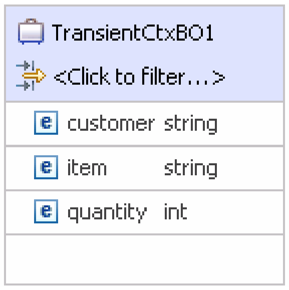
Building workspace: (3%)

Part 2: Authoring the mediation flow for message augmentation

What you will do in this part: In this part of the lab you will develop the mediation flow used to augment the message through use of a service invoke primitive. This will involve using a service invoke primitive with XSL transformation primitives before and after. See the presentation entitled Augmentation, aggregation and retry labs to better understand what this part is doing.

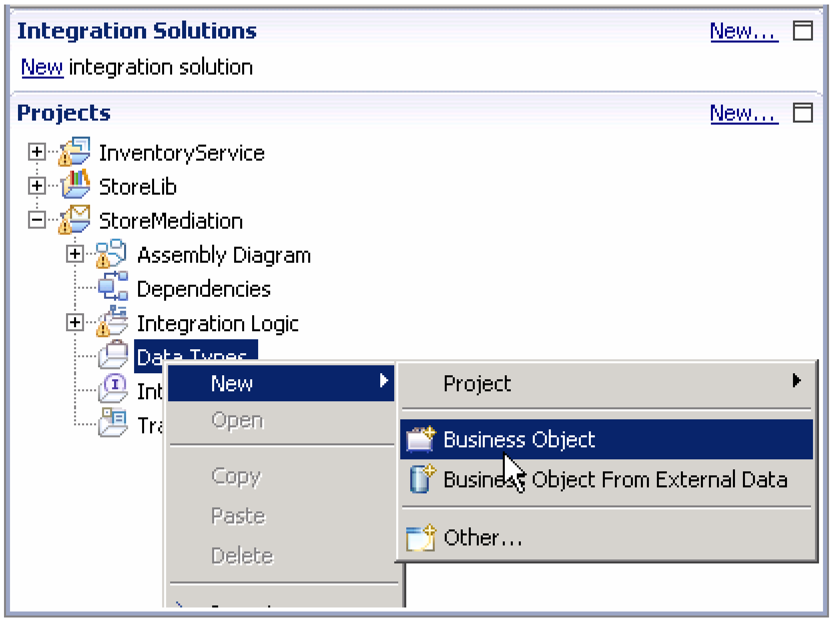
____ 1. Create a new business object that is used as the transient context for the flow. This is used to temporarily store information that otherwise would be lost during the flow.

- **`Module     : StoreMediation`**
- **`Name       : TransientCtxBO1`**
- **`Attributes: customer    string`**
- **`            item        string`**
- **`            quantity    int`**

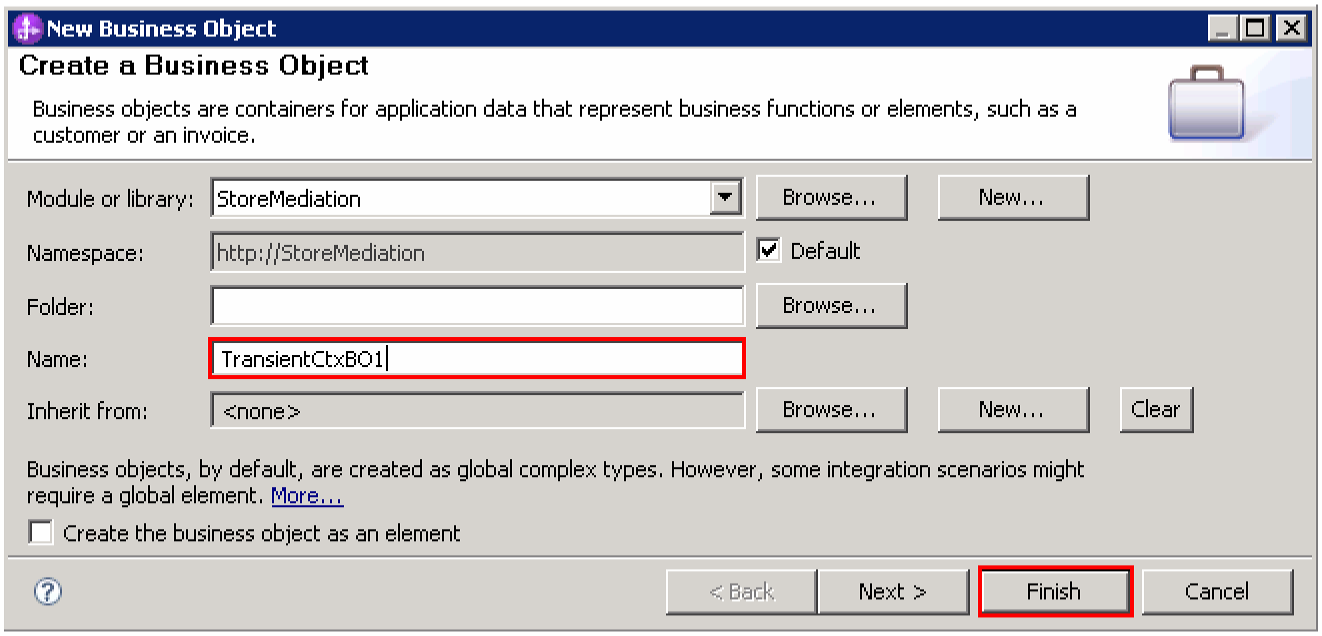


- **`-----------------------------------------------------------------`**

__ **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** wizard, provide the name, **TransientCtxBO1**



New Business Object

Create a Business Object

Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module or library: StoreMediation

Namespace: http://StoreMediation ☑ Default

Folder:

Name: TransientCtxBO1

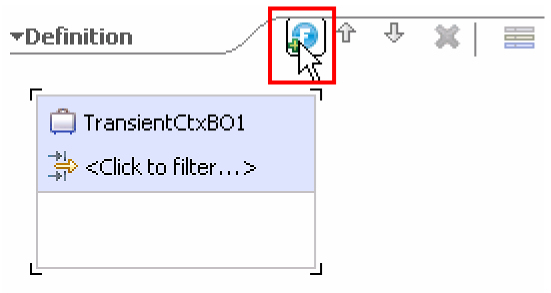
Inherit from: <none>

Business objects, by default, are created as global complex types. However, some integration scenarios might require a global element. More...

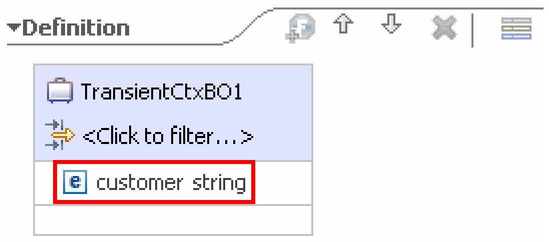
☐ Create the business object as an element

__ c. Click **Finish.** The business object editor opens

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



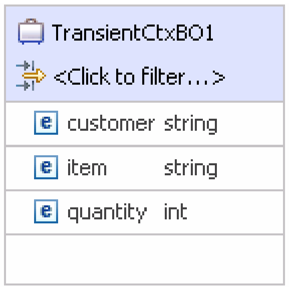
__ e. Edit the new field to have a name of **customer** and a type of **string**



__ f. Similarly, add the field **item** of type **string**

__ g. Also add the field **quantity** of type **int**, using the drop down box to select the type

__ h. Check that your business object looks like the following screen capture:



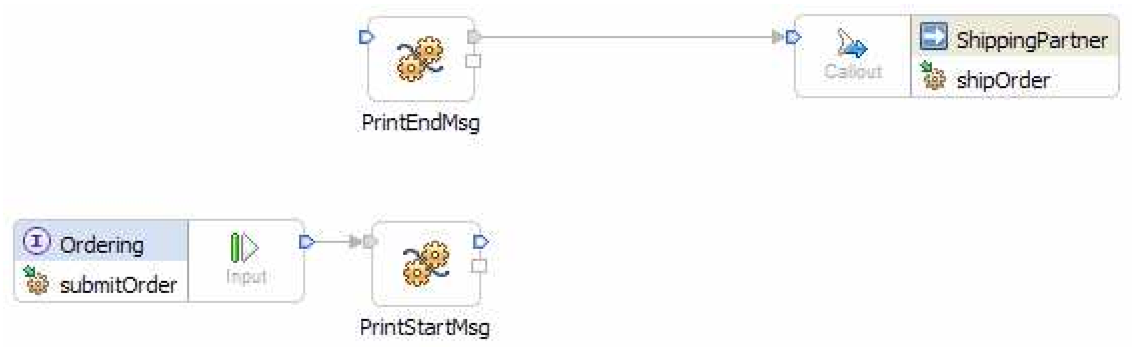
TransientCtxBO1
<Click to filter…>

customer	string
item	string
quantity	int

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

__ j. Close the **TransientCtxBO1**, business object editor

____ 2. Open the StoreMediation flow found in the StoreMediation module. The flow already contains two custom mediation primitives that are used to write start and end messages to the log.



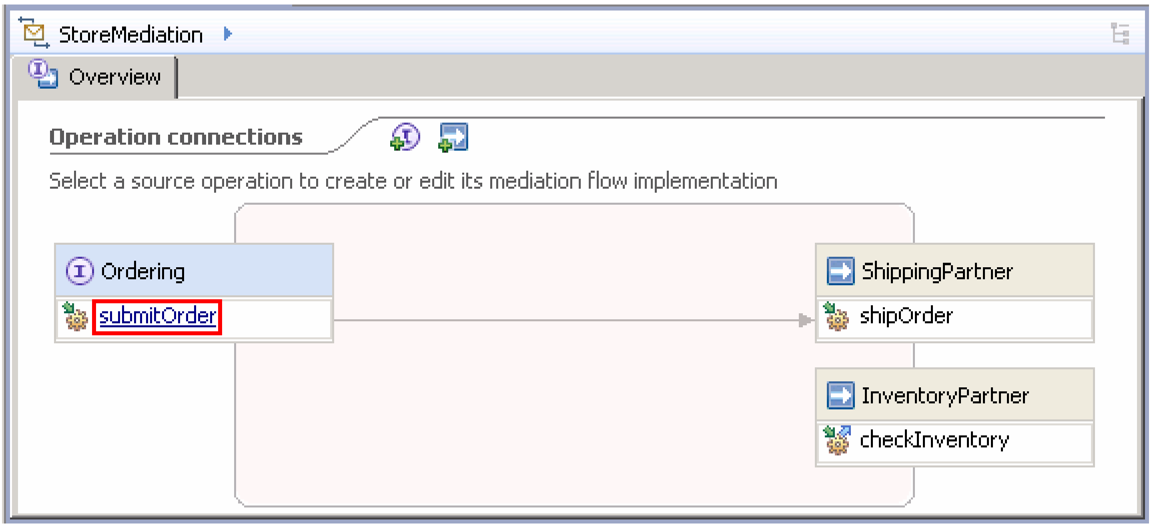
NOTE: The primitives used to write the start and end messages to the log are custom mediation primitives implemented in Java. In version 7 there is also a trace primitive that can be used to write to the log. In a case where you want write a single line or want to dump some of the message contents to the log, the trace primitive is generally a better choice. However, in this case, multiple lines of text are being written to the log without dumping any message content and therefore the custom mediation remains a good choice for this situation.

- --

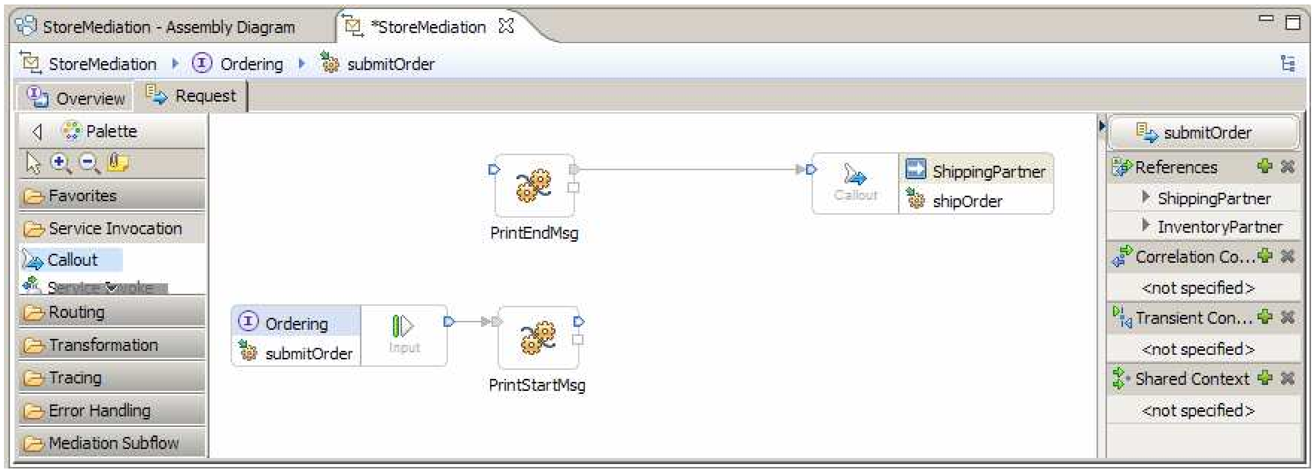
__ a. In the Business Integration view, expand **StoreMediation → Integration Logic → Mediation Flows** and then double-click **StoreMediation** to open it in the mediation flow editor

---

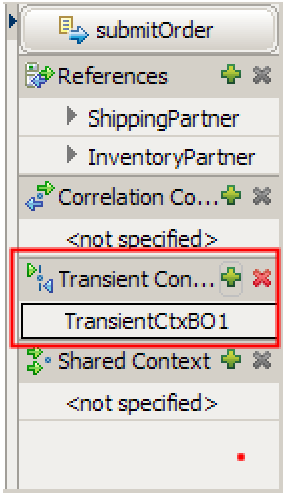
NOTE: If the **Tip** dialog is displayed, you can choose to turn off tips by selecting the **Do not show tips** check box. The instructions throughout this lab series assumes that tips are not shown, but you may want to allow them to continue if you find them useful.



__ 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

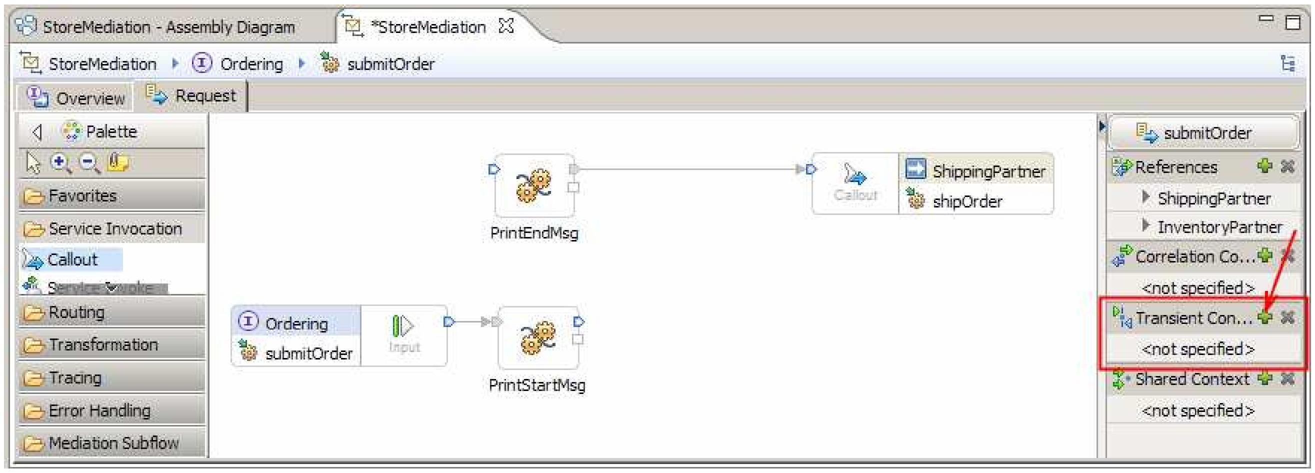


____ 3. In the right side of the mediation flow editor, configure the transient context for the flow to use the **TransientCtxBO1**

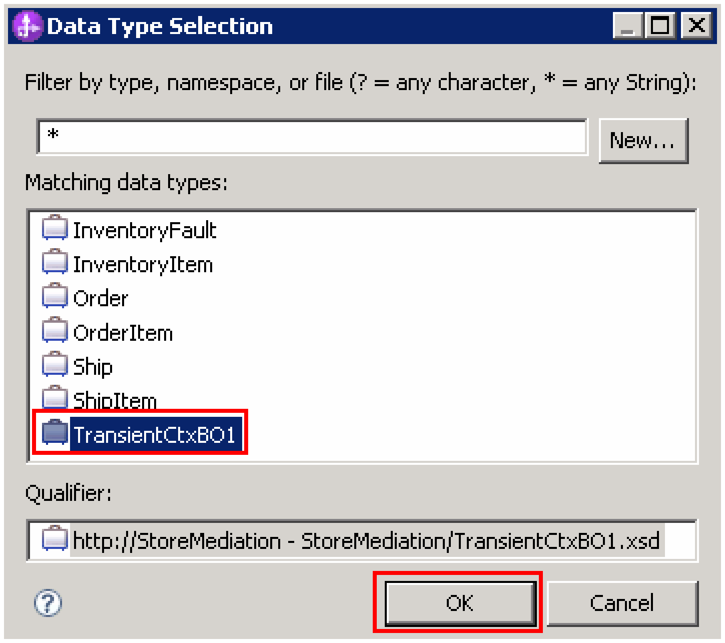


• -

__ a. In the Mediation Flow to the right side, click the **Add** (➕) icon next to **Transient Context**



__ b. In the Data Type Selection window, scroll down to select **TransientCtxBO1** under **Matching data types:**


Data Type Selection

Filter by type, namespace, or file (? = any character, * = any String):

*

Matching data types:

InventoryFault
InventoryItem
Order
OrderItem
Ship
ShipItem
TransientCtxBO1

Qualifier:

http://StoreMediation - StoreMediation/TransientCtxBO1.xsd

__ c. Click **OK**

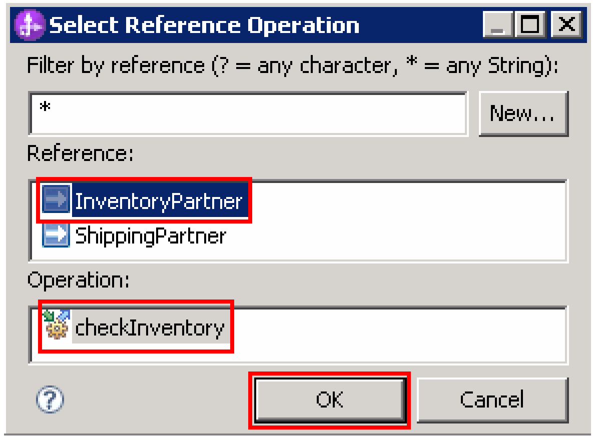
__ d. From the main menu, select **File → Save** (or **Ctrl + S** from your keyboard) to save your changes

____ 4. Add and configure a service invoke primitive to the canvas. This is used to call the inventory service which provides inventory status information about the order item.

```
•  Reference:         InventoryPartner
•  Operation:         checkInventory
•  Display name:      CheckInventory
•  Dynamic endpoint:  Uncheck “Use dynamic endpoint if set in message
•                     header” option on Details panel
•  ---------------------------------------------------------------
```

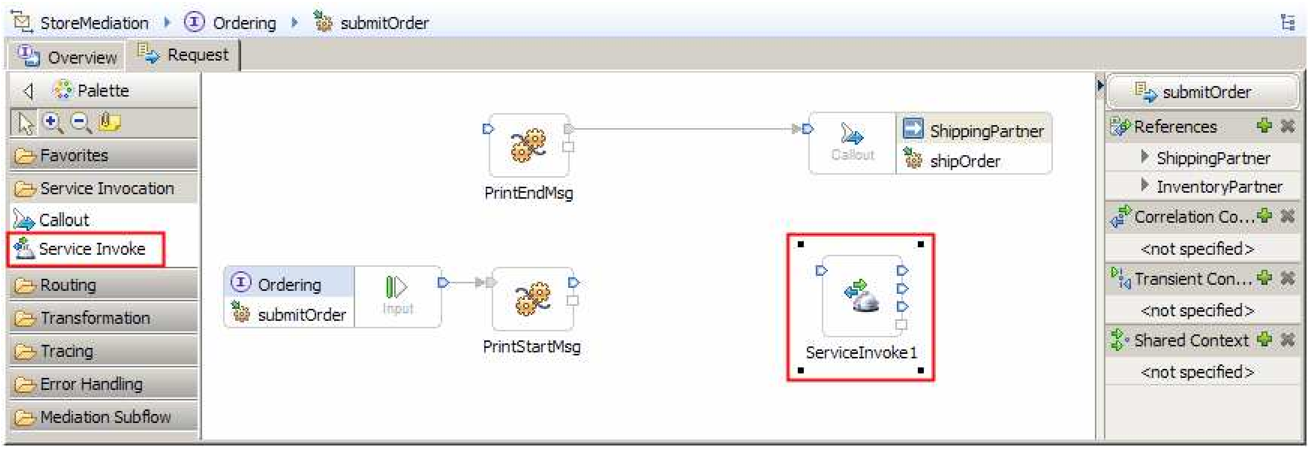
__ a. In the **Palette** on the left side of the mediation flow editor, click to expand the **Service Invocation** tray. Then click to select the **Service Invoke** icon (Service Invoke), move the cursor over the canvas and click to drop it on the canvas.

__ b. In the Select Reference Operation dialog window, select **InventoryPartner** under the **Reference** list and **checkInventory** under the **Operation** list.

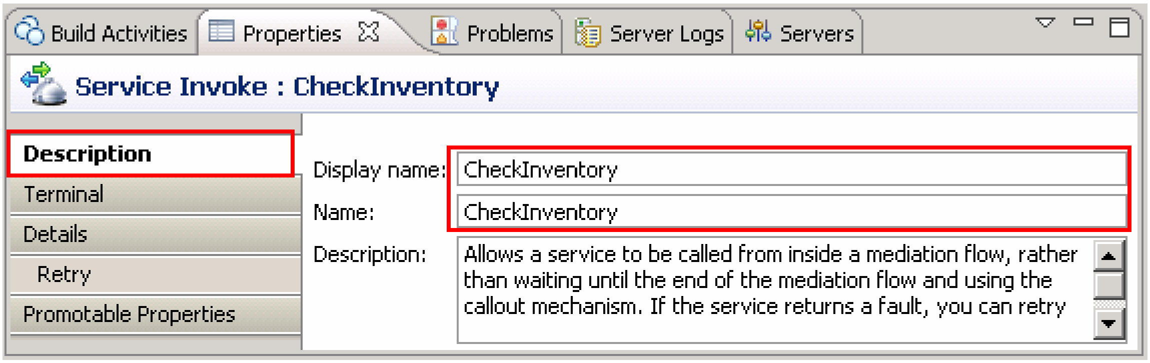

Select Reference Operation
Filter by reference (? = any character, * = any String):
*
New...
Reference:
InventoryPartner
ShippingPartner
Operation:
checkInventory
OK
Cancel

__ c. Click **OK**

__ d. You will now see a new service inventory primitive added to the flow. Ensure that the service invoke primitive is selected – the four dots at the corners of the primitive indicate it is selected.



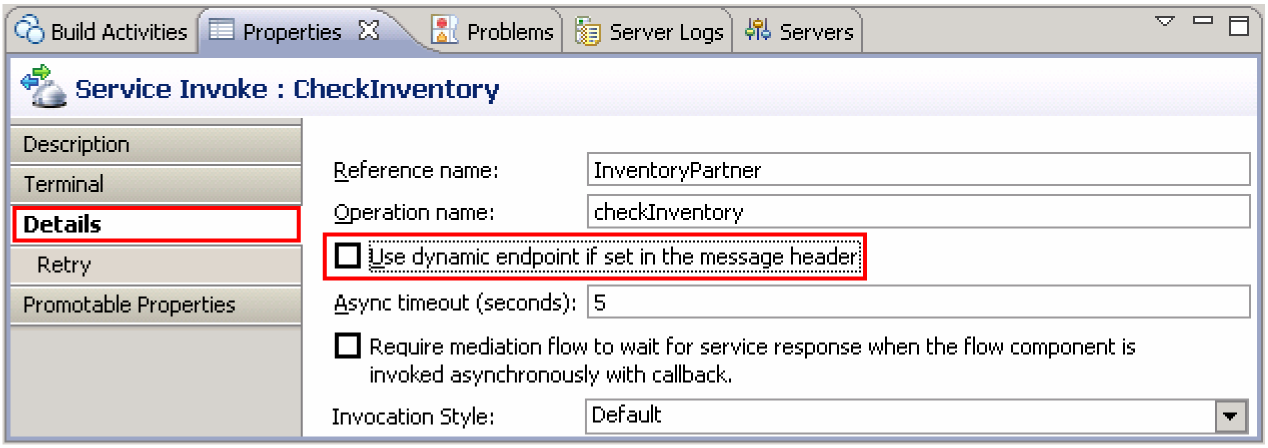
__ e. Now select **Properties → Description** and change the **Display name** to **CheckInventory**. This will also change the Name to CheckInventory



Display name: CheckInventory
Name: CheckInventory
Description: Allows a service to be called from inside a mediation flow, rather than waiting until the end of the mediation flow and using the callout mechanism. If the service returns a fault, you can retry

__ f. For the same service invoke primitive, select **Properties → Details**

__ g. **Clear** the check box for **Use dynamic endpoint if set in the message header**



Service Invoke : CheckInventory

Reference name: InventoryPartner

Operation name: checkInventory

☐ Use dynamic endpoint if set in the message header

Async timeout (seconds): 5

☐ Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback.

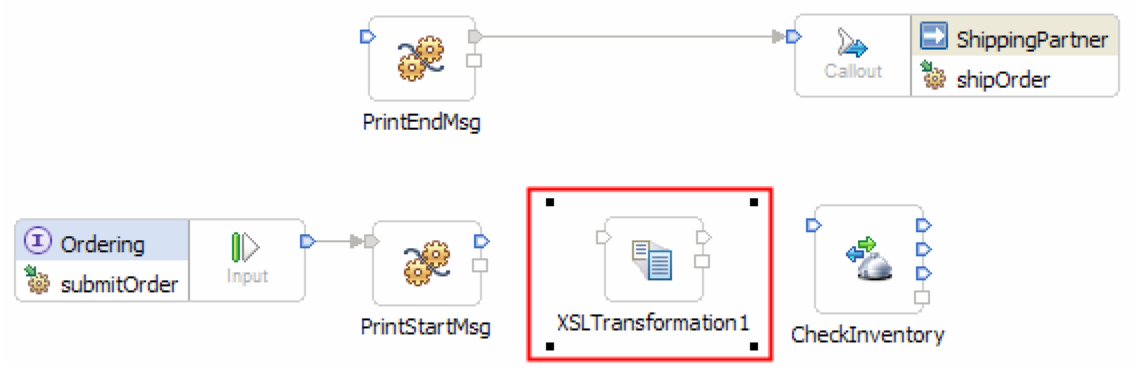
Invocation Style: Default

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

____ 5. Add an XSL transformation primitive to the canvas and wire it into the flow before the CheckInventory, service invoke primitive. This primitive provides the required transformations to call the inventory service.

- **`Display name: Order2Inventory`**
- **`Wire         : PrintStartMsg to Order2Inventory`**
- **`Wire         : Order2Inventory to CheckInventory`**
- **`---------------------------------------------------------------------`**

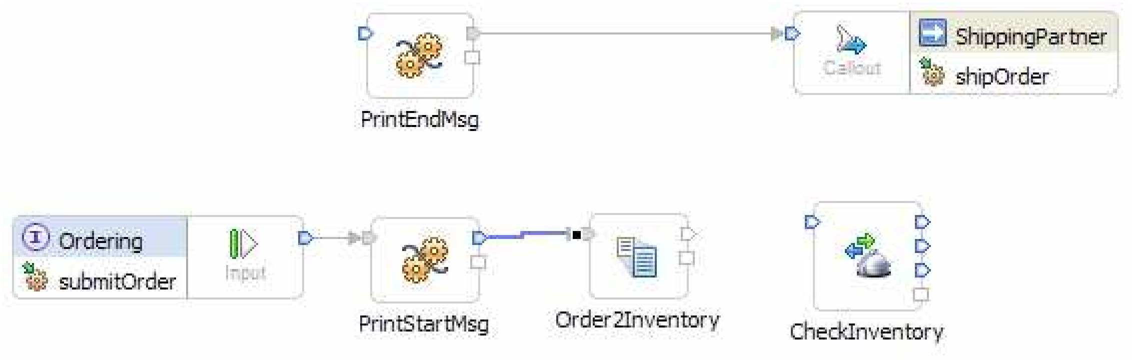
__ a. From the **Palette**, select **Transformation → XSL Transformation** and then click the canvas to drop the primitive to the left side of the CheckInventory primitive



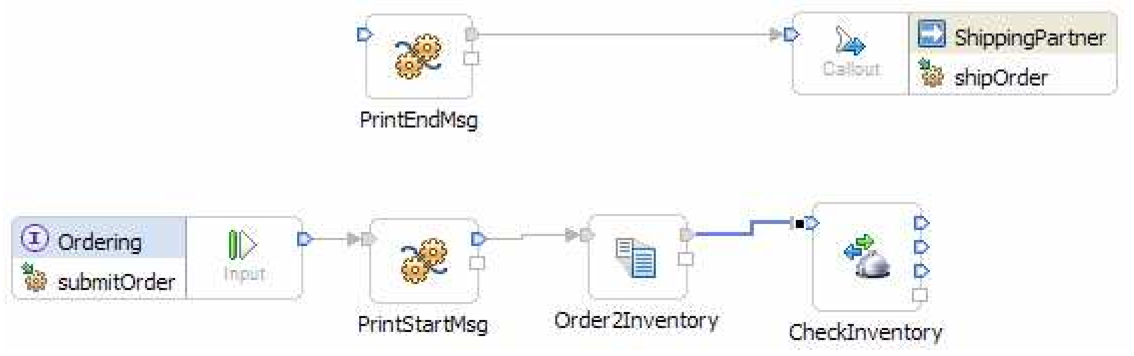
__ b. You will now see a new XSL Transformation primitive added to the flow. Ensure that this primitive is selected and navigate to **Properties → Description**

__ c. Change the **Display name** to **Order2Inventory**

__ d. Add a connection from **PrintStartMsg** to **Order2Inventory**. This is done by clicking on the **output terminal** of **PrintStartMsg** and dragging it to the **input terminal** of **Order2Inventry**



__ e. Similarly, add a connection from **Order2Inventory** to **CheckInventory.** You flow should now look like this:



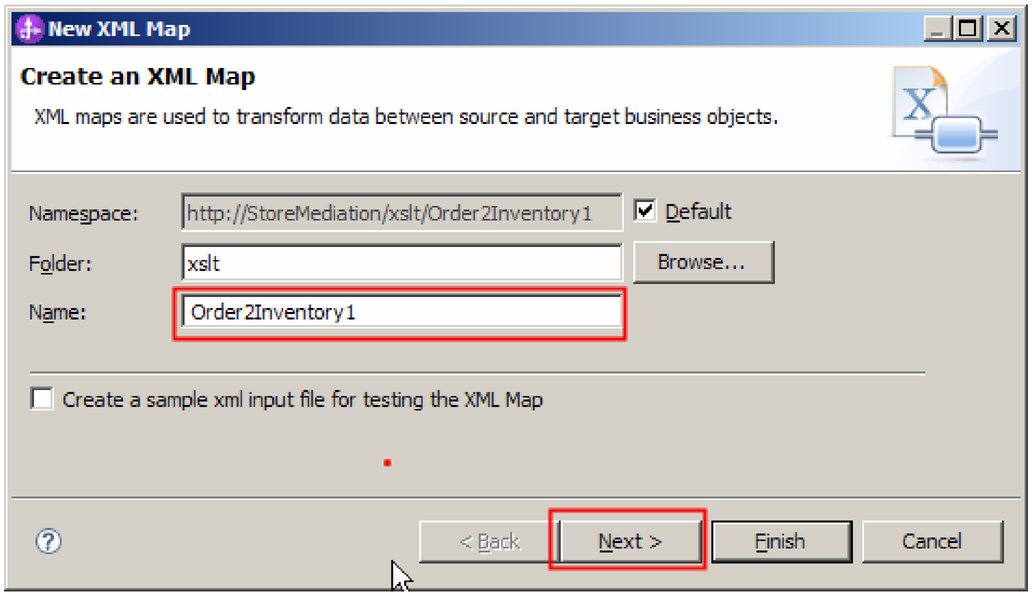
____ 6. In the Order2Inventory XML transformation primitive, create a new XML mapping file used to define the transformation.

- **`Map Name            :    Order2Inventory1`**
- **`Message Root        :    /`**
- **`Input Message Body :    submitOrderRequestMsg`**
- **`Output Message Body:    checkInventoryRequestMsg`**
- **`--------------------------------------------------------------------`**

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

__ b. Click **New…** for **Mapping file**. The New XML Map dialog opens

__ c. For **Name** enter **Order2Inventory1**



XML maps are used to transform data between source and target business objects.

Namespace: http://StoreMediation/xslt/Order2Inventory1

Folder: xslt

Name: Order2Inventory1

Create a sample xml input file for testing the XML Map

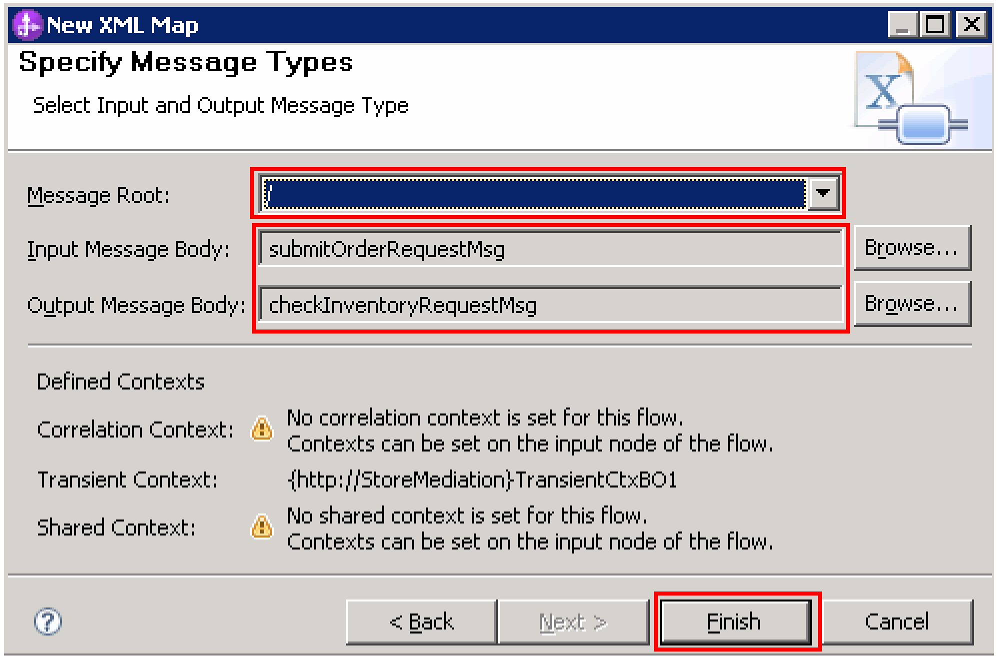
__ d. Click **Next**

__ e. From the Specify Message Types panel, enter :

1) For **Message Root**, select '/' from the drop down list

2) For **Input Message Body**, accept the default selection: **sumbitOrderRequestMsg**

3) For **Output Message Body**, accept the default selection: **checkInventoryRequestMsg**



Select Input and Output Message Type

Message Root: /

Input Message Body: submitOrderRequestMsg

Output Message Body: checkInventoryRequestMsg

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}TransientCtxBO1

Shared Context: No shared context is set for this flow. Contexts can be set on the input node of the flow.

__ f. Click **Finish**. The Order2Inventory1.map file is opened in the XML Mapping editor.

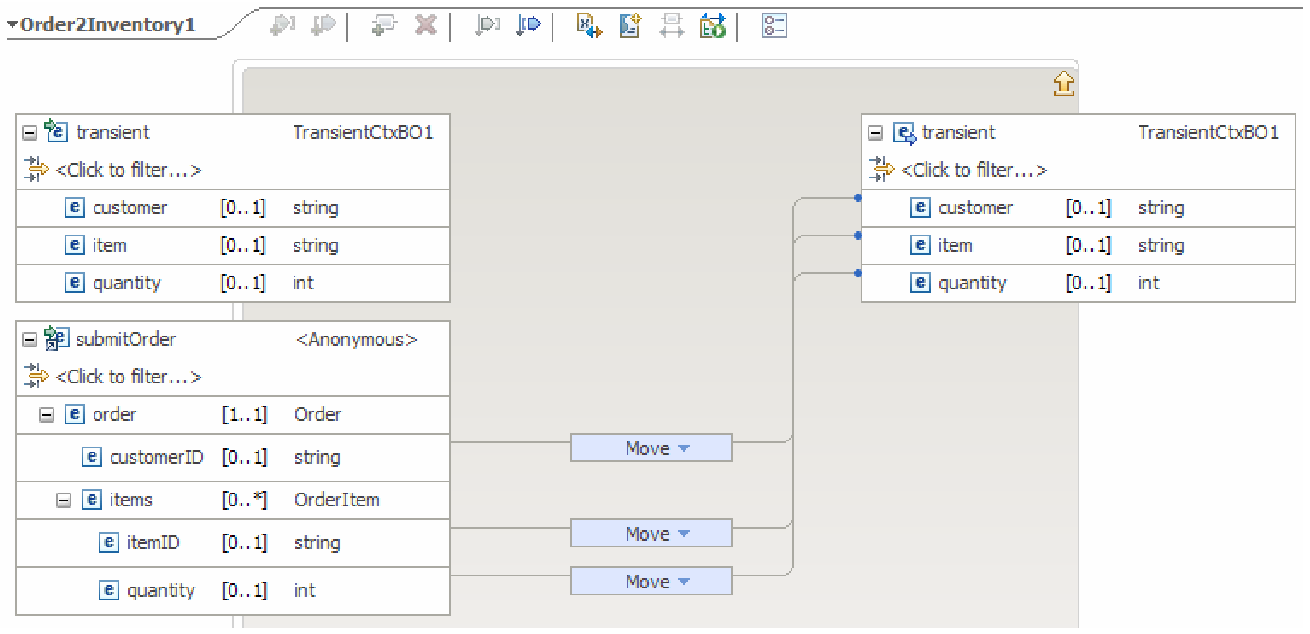
____ 7. Define the mapping for the Order2Inventory1 map. The mapping must address: (1) Moving fields between the source and target SMOs that will not change. (2) Saving information in the transient context from the source message body that is needed later in the flow. (3) Setting up the target message body needed to call the inventory service.

- **(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' to generate all the mappings for like elements.**
- **(2) Saving information in the transient context from the source message body that is needed later in the flow.**
  - **Convert the local map for context to context to a merge of source context and body to target context.**
  - **In the merge, convert the local map for transient to transient to a merge of source transient and submitOrder to target transient.**
  - **In the merge, remove the three existing move transforms and add move transforms for the following:**

Source (left side)	**Target (right side)**
body/submitOrder/order/customerID	context/transient/customer
body/submitOrder/order/items[1]/itemID	context/transient/item
body/submitOrder/order/items[1]/quantity	context/transient/quantity

NOTE: In the XPath expressions, if there is a specific index for an element in an array (such as **items[1]**), you must use the **Properties → Cardinality** view to specify the index.

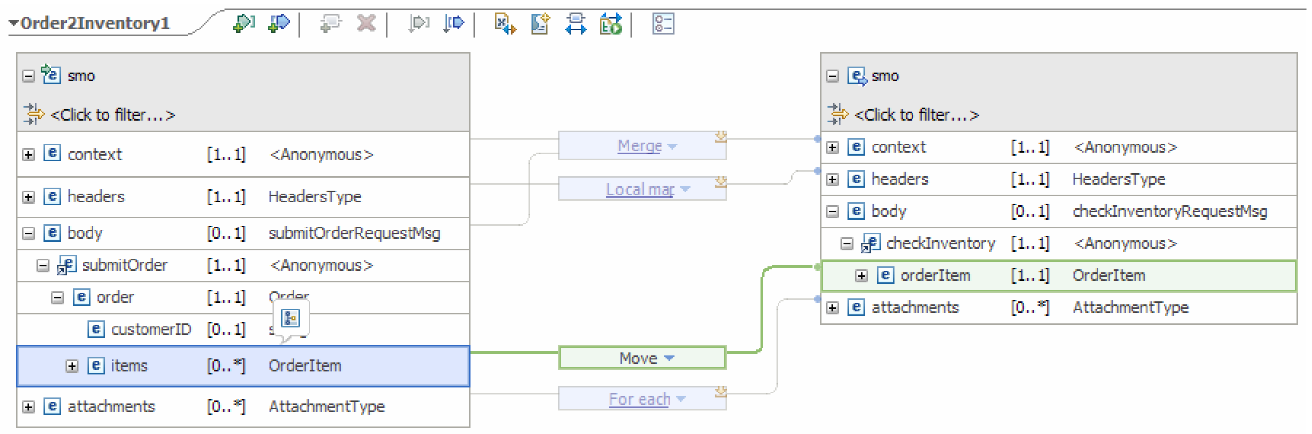
-
- **The resulting merge should look like this:**



- **(3) Setting up the target message body needed to call the inventory service.**
  - **At the SMO level (top level map) add this move transform**

Source (left side)	Target (right side)
body/submitOrder/order/items[1]	body/checkInventory/orderItem

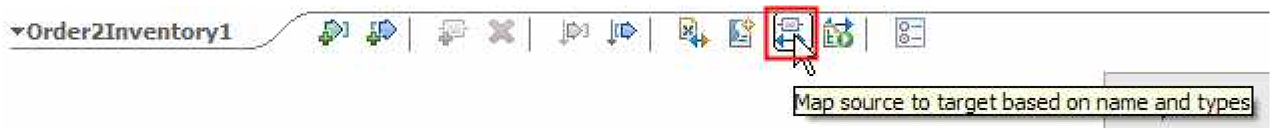
-
- **The resulting map should look like this:**


Order2Inventory1
smo
<Click to filter...>
context [1..1] <Anonymous>
headers [1..1] HeadersType
body [0..1] submitOrderRequestMsg
submitOrder [1..1] <Anonymous>
order [1..1] Order
customerID [0..1]
items [0..*] OrderItem
attachments [0..*] AttachmentType
Merge
Local map
Move
For each
smo
<Click to filter...>
context [1..1] <Anonymous>
headers [1..1] HeadersType
body [0..1] checkInventoryRequestMsg
checkInventory [1..1] <Anonymous>
orderItem [1..1] OrderItem
attachments [0..*] AttachmentType

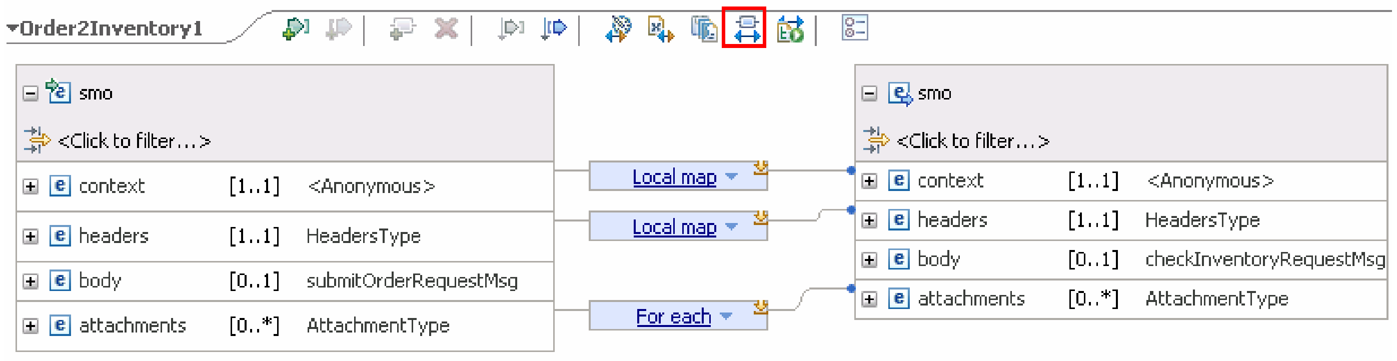
- --

__ 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 ( )

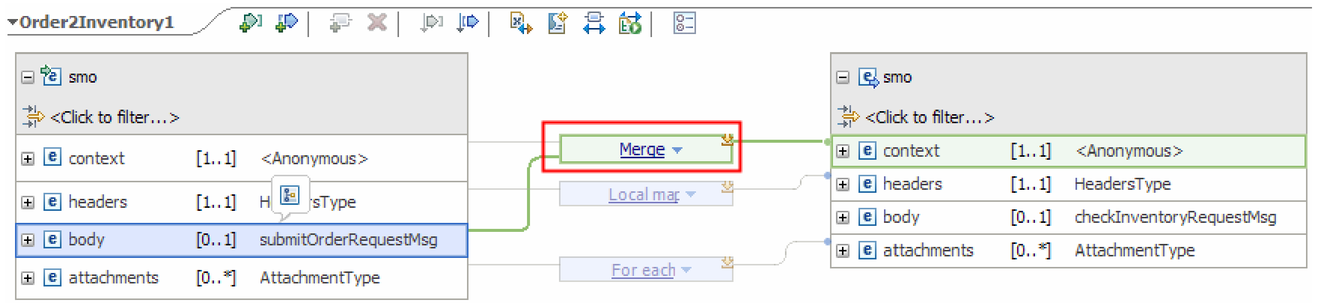


2) This maps the context and headers using local maps. The body is not mapped because the source and target bodies are different. The attachments are mapped with a for each transform. The result is shown below:


Order2Inventory1
smo
<Click to filter...>
context [1..1] <Anonymous>
headers [1..1] HeadersType
body [0..1] submitOrderRequestMsg
attachments [0..*] AttachmentType
Local map
Local map
For each
smo
<Click to filter...>
context [1..1] <Anonymous>
headers [1..1] HeadersType
body [0..1] checkInventoryRequestMsg
attachments [0..*] AttachmentType

__ b. These steps address saving information in the transient context from the source message body. This information is needed again later in the flow and therefore must be saved. The following steps are used to map those elements by using merge rather than local maps.

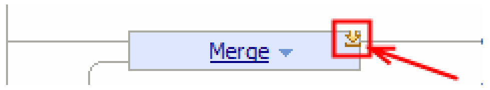
1) On the source (left) side, click anywhere in the **body** and drag it to the **Local Map** for context to context. This converts the **Local Map** to a **Merge** transform. The result look like this:



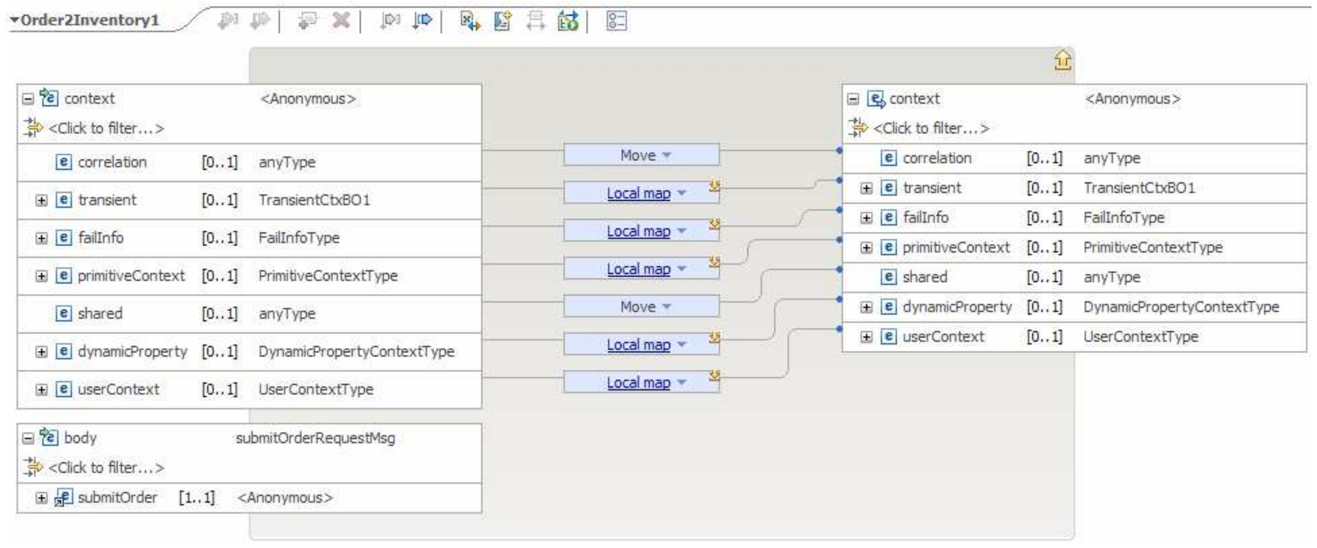
smo		
<Click to filter...>		
context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	submitOrderRequestMsg
attachments	[0..*]	AttachmentType

smo		
<Click to filter...>		
context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	checkInventoryRequestMsg
attachments	[0..*]	AttachmentType

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



3) The mapping for the merge looks like this:

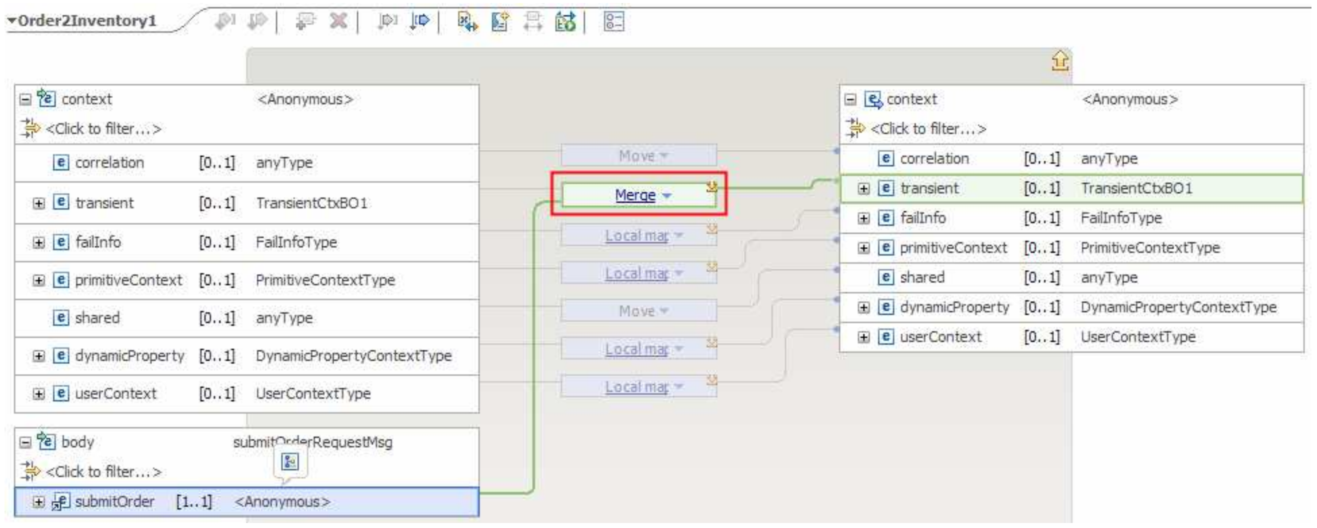


context		<Anonymous>
<Click to filter...>		
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO1
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	anyType
dynamicProperty	[0..1]	DynamicPropertyContextType
userContext	[0..1]	UserContextType

body		submitOrderRequestMsg
<Click to filter...>		
submitOrder	[1..1]	<Anonymous>

context		<Anonymous>
<Click to filter...>		
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO1
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	anyType
dynamicProperty	[0..1]	DynamicPropertyContextType
userContext	[0..1]	UserContextType

4) In a similar manner, click **submitOrder** in the source body and drag it to the **Local map** from transient to transient, converting it to a **Merge**.

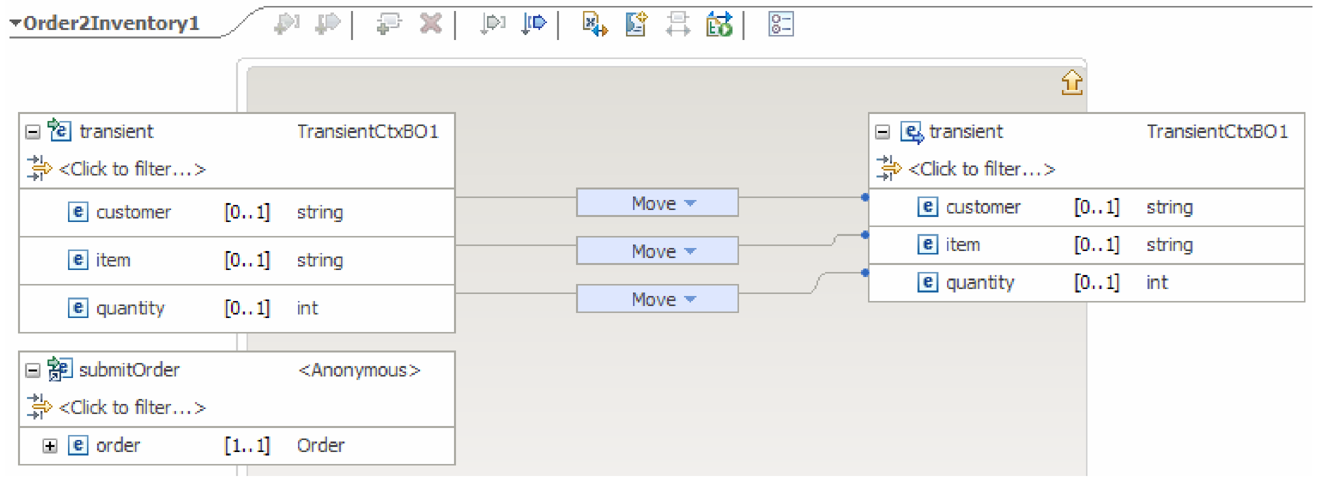


context		<Anonymous>
<Click to filter...>		
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO1
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	anyType
dynamicProperty	[0..1]	DynamicPropertyContextType
userContext	[0..1]	UserContextType

body		submitOrderRequestMsg
<Click to filter...>		
submitOrder	[1..1]	<Anonymous>

context		<Anonymous>
<Click to filter...>		
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO1
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	anyType
dynamicProperty	[0..1]	DynamicPropertyContextType
userContext	[0..1]	UserContextType

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



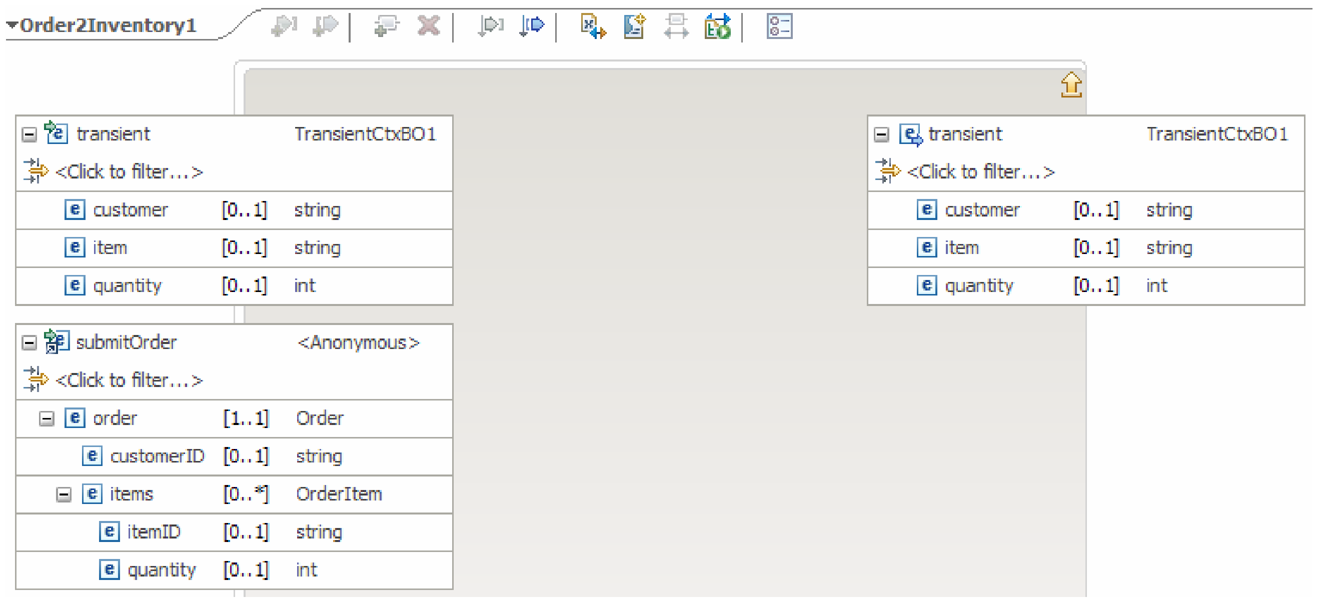
transient		TransientCtxBO1
<Click to filter...>		
customer	[0..1]	string
item	[0..1]	string
quantity	[0..1]	int

submitOrder		<Anonymous>
<Click to filter...>		
order	[1..1]	Order

transient		TransientCtxBO1
<Click to filter...>		
customer	[0..1]	string
item	[0..1]	string
quantity	[0..1]	int

6) Delete all three of the **Move** transforms by selecting the transform and hitting the **Delete** key.

7) On the source side, expand **order** and then expand **items**. The map should now look like this:



Order2Inventory1

transient		TransientCtxBO1
<Click to filter...>		
customer	[0..1]	string
item	[0..1]	string
quantity	[0..1]	int

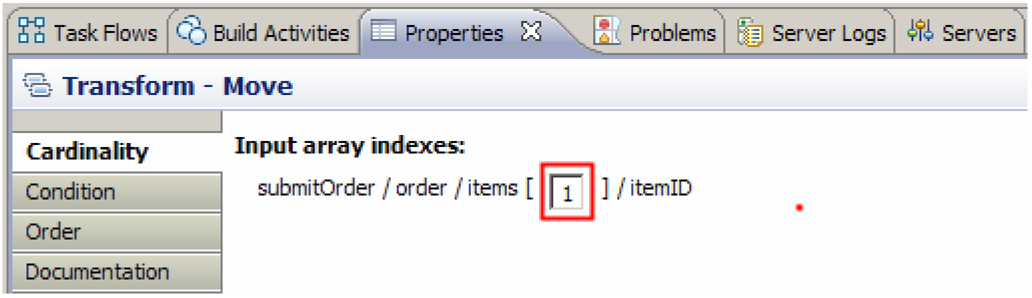
submitOrder		<Anonymous>
<Click to filter...>		
order	[1..1]	Order
customerID	[0..1]	string
items	[0..*]	OrderItem
itemID	[0..1]	string
quantity	[0..1]	int

transient		TransientCtxBO1
<Click to filter...>		
customer	[0..1]	string
item	[0..1]	string
quantity	[0..1]	int

8) Click **customerID** on the source side and drag it to **customer** on the target side, creating a **Move** transform.

9) Similarly, create a **Move** transform from **itemID** to **item**.

10) Since **itemID** is contained in the **items** array, an array index must be specified. Ensure that the **Move** transform is selected, and In the **Cardinality** panel of the **Properties** view, set the index to 1.



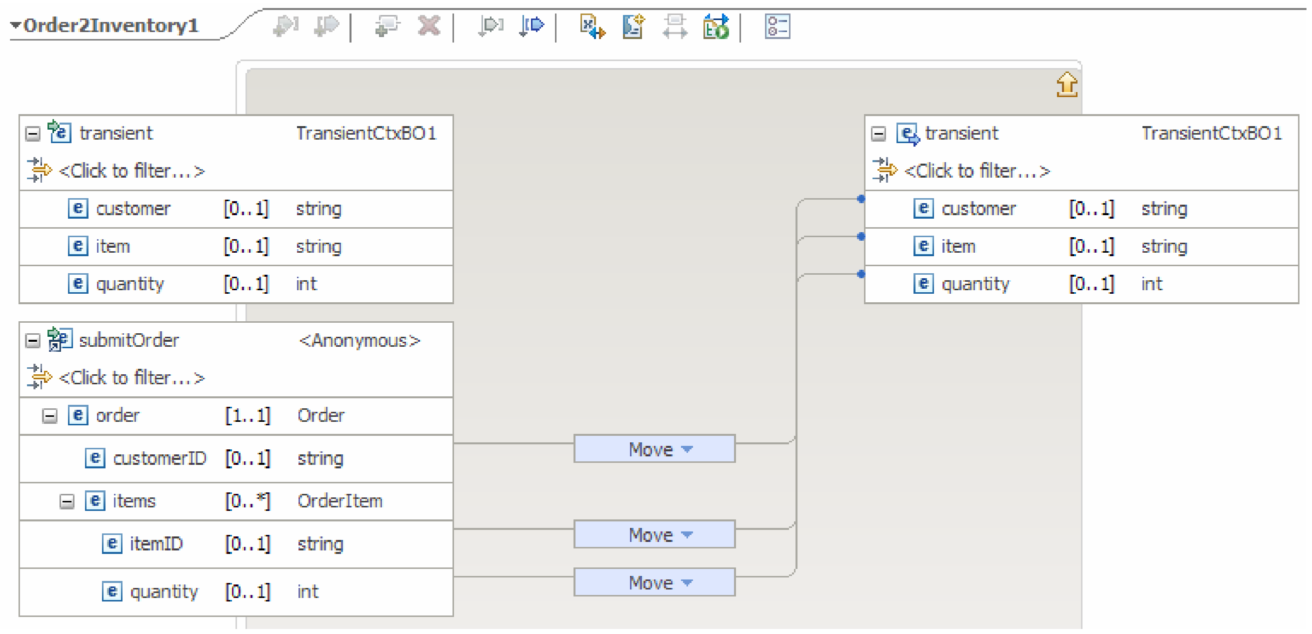
Transform - Move

Input array indexes:

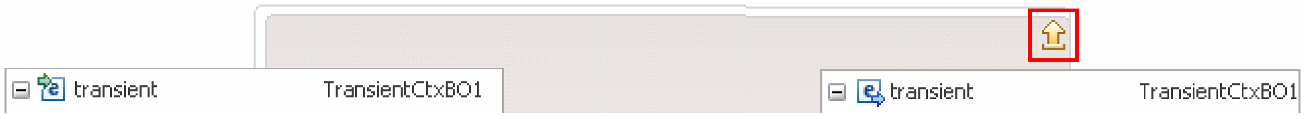
submitOrder / order / items [1] / itemID

11) Similarly, create a **Move** transform from **quantity** to **quantity** and set the index to 1 in the **Cardinality** pane.

12) The resulting map should look like this:



13) Click the '**Up a level**' ( ) icon, which will bring you one level up to the context mapping



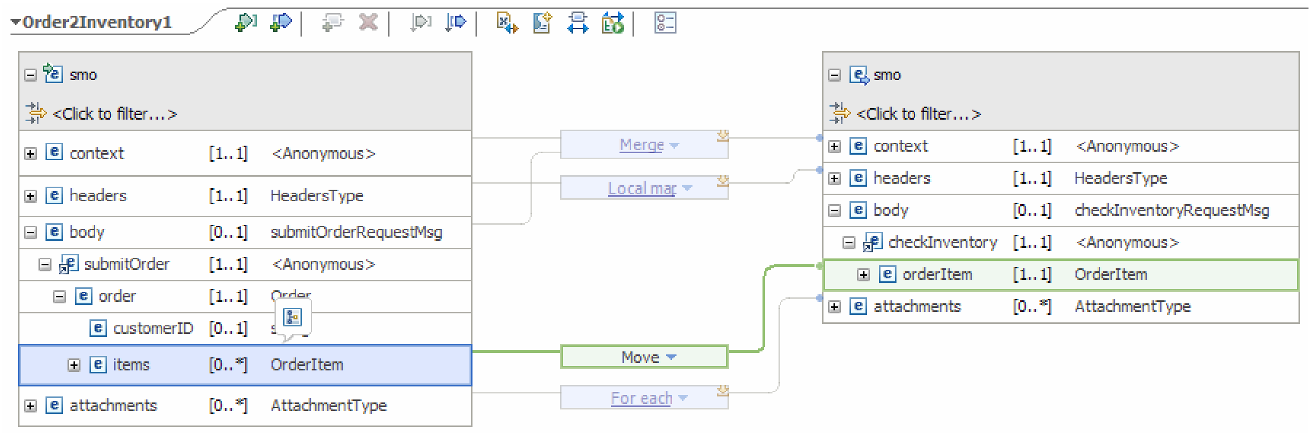
14) Click the **Up a level** ( ) icon one more time. This will now bring you back the smo mapping level

__ c. These steps are needed to set up the target message body needed to call the inventory service. Essentially, the source element body/submitOrder/order/items[1] need to be moved to the target element body/checkInventory/orderItem

1) In the source (left) smo, expand **body → submitOrder → order**

2) In the target (right) smo, expand **body → checkInventory**

3) Click **items** and drag it to **orderItem.** The resulting map should look like this:



context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	submitOrderRequestMsg
submitOrder	[1..1]	<Anonymous>
order	[1..1]	Order
customerID	[0..1]	s
items	[0..*]	OrderItem
attachments	[0..*]	AttachmentType

context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	checkInventoryRequestMsg
checkInventory	[1..1]	<Anonymous>
orderItem	[1..1]	OrderItem
attachments	[0..*]	AttachmentType

4) Since **items** is an array, an array index must be specified. Ensure that the **Move** transform is selected, and In the **Cardinality** panel of the **Properties** view, set the index to 1.

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

__ e. Close Order2Inventory1.map file

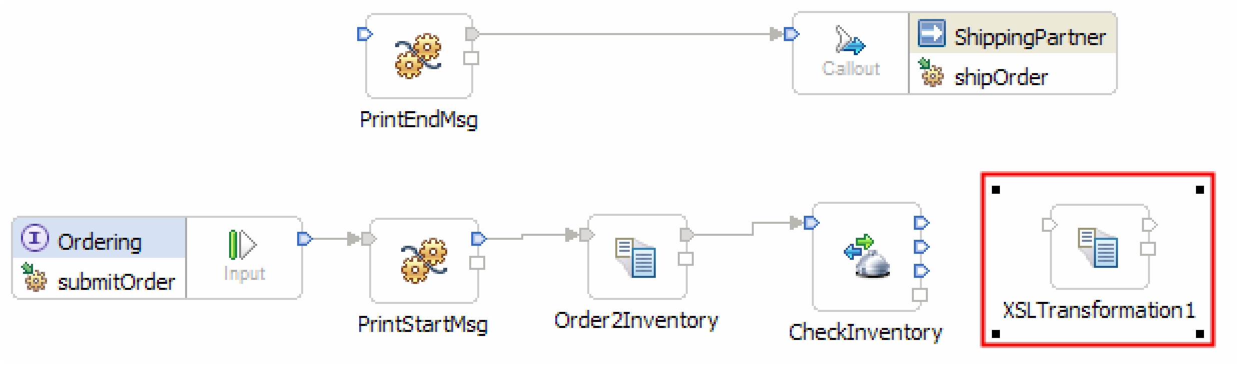
__ f. From the menu select **File → Save** (or **Ctrl + S** from your keyboard) to save your changes to the mediation flow

____ 8. Add an XSL transformation primitive to the canvas and wire it into the flow following the CheckInventory primitive. This primitive provides the required transformations to call the shipping service.

- **`Display Name: Inventory2Ship`**
- **`Wire         : CheckInventory(out terminal) to Inventory2Ship`**
- **`Wire         : Inventory2Ship to PrintEndMsg`**
- **`------------------------------------------------------------------`**

__ a. From the **Palette**, select **Transformation → XSL Transformation** and then click the canvas to drop the primitive to the right side of CheckInventory primitive

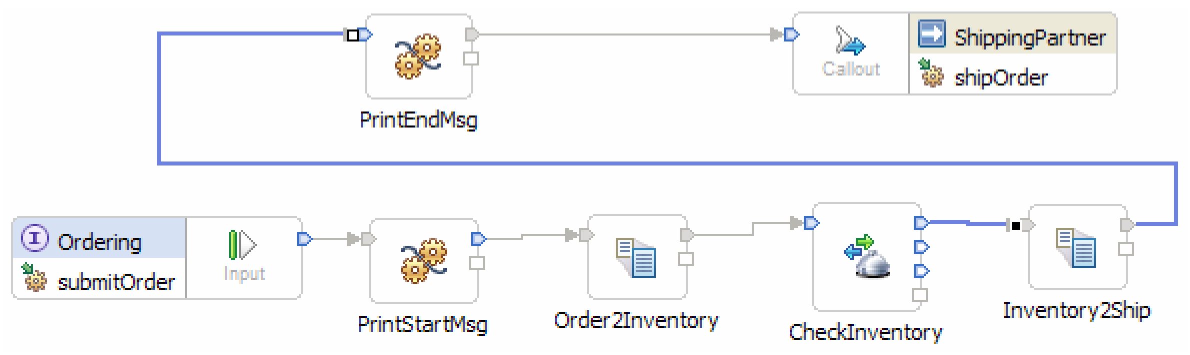
__ b. You will now see a new XSL Transformation primitive added to the flow. Ensure that this primitive is highlighted and select **Properties → Description**



__ c. Change the **Display name** to **Inventory2Ship**

__ d. Add a connection from **CheckInventory** to **Inventory2Ship**.

__ e. Similarly, add a connection from the **Inventory2Ship** primitive to the **PrintEndMsg** primitive, resulting in a flow that looks like this.



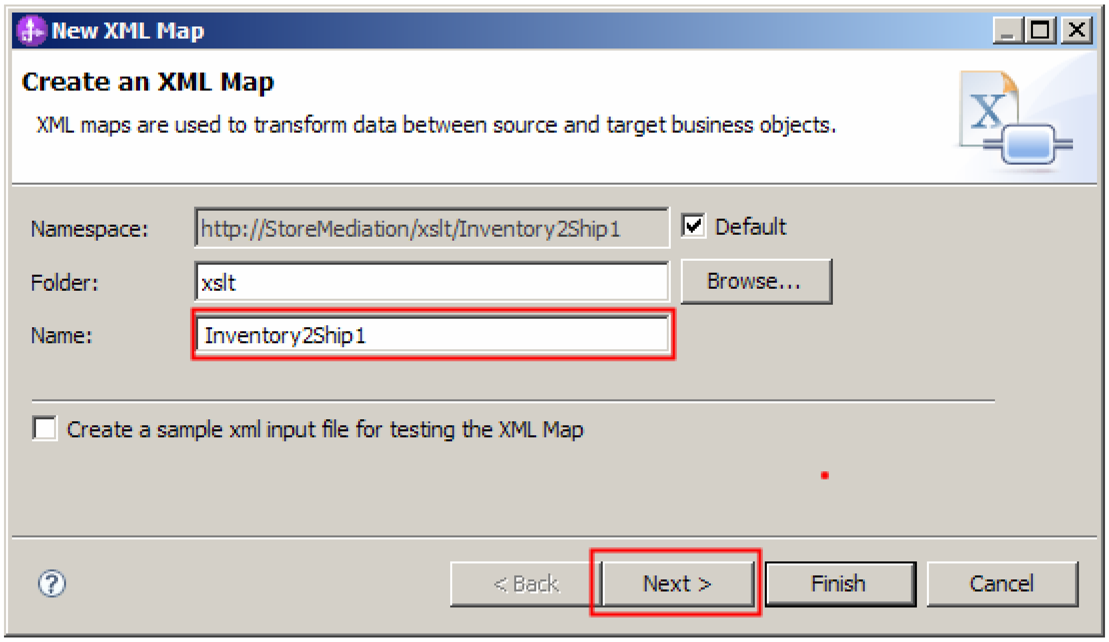
____ 9. In the Inventory2Ship XML transformation primitive, create a new XML mapping file used to define the transformation.

- **`Map Name            :    Inventory2Ship1`**
- **`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…** for **Mapping file**. The New XML Map dialog opens.

__ c. For **Name** enter **Inventory2Ship1**


New XML Map

Create an XML Map

XML maps are used to transform data between source and target business objects.

Namespace: http://StoreMediation/xslt/Inventory2Ship1 [✓ Default]

Folder: xslt

Name: Inventory2Ship1

☐ Create a sample xml input file for testing the XML Map

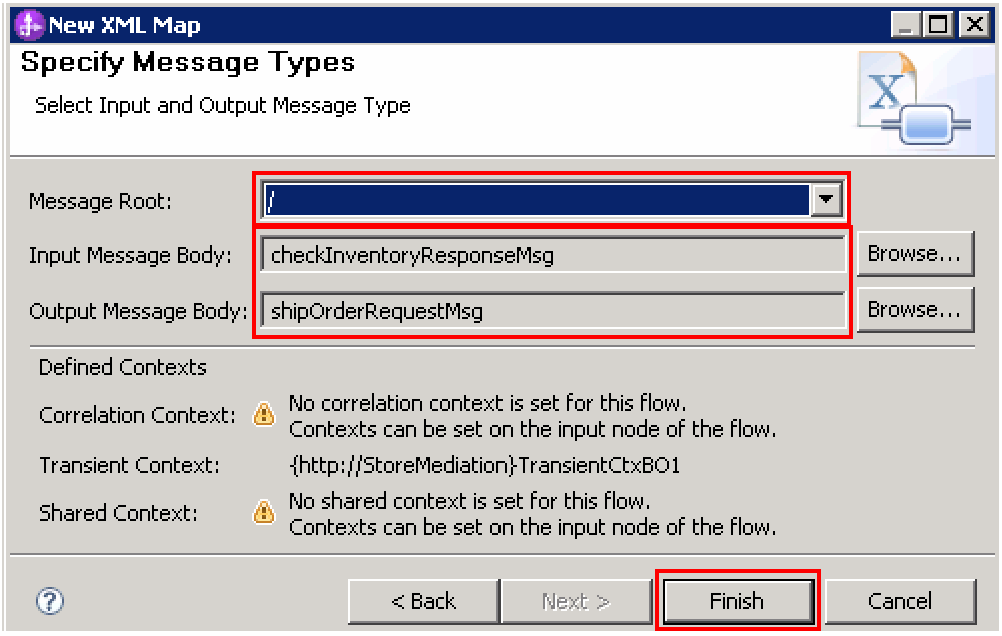
__ d. Click **Next**

__ e. From the Specify Message Types panel, enter:

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 Output Message Type

Message Root: /

Input Message Body: checkInventoryResponseMsg

Output 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: {http://StoreMediation}TransientCtxBO1

Shared Context: No shared context is set for this flow. Contexts can be set on the input node of the flow.

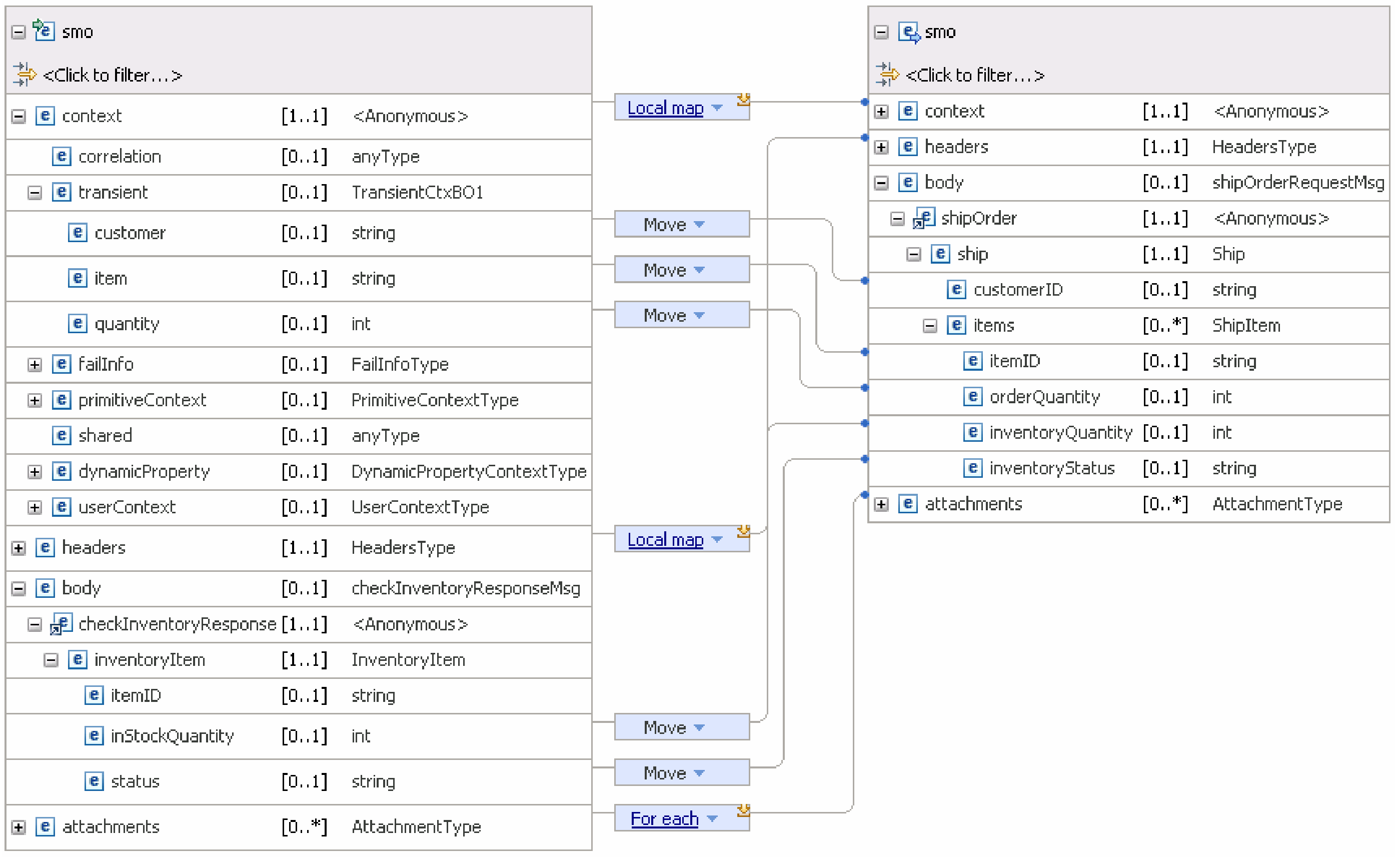
__ f. Click **Finish**. The Inventory2Ship1.map file is opened in the XML Mapping editor

____ 10. Define the mapping for the Inventory2Ship1 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.

- **`Use toolbar icon () 'Map source to target based on name and types' to generate all the mappings for like elements`**
- **`At the SMO level (top level map) add these move transforms`**

Source (left side)	Target (right side)
context/transient/customer	body/shipOrder/ship/customerID
context/transient/item	body/shipOrder/ship/items[1]/itemID
context/transient/quantity	body/shipOrder/ship/items[1]/orderQuantity
body/checkInventoryResponse/inventoryItem/inStockQuantity	body/shipOrder/ship/items[1]/inventoryQuantity
body/checkInventoryResponse/inventoryItem/status	body/shipOrder/ship/items[1]/inventoryStatus

-



smo

<Click to filter...>

context	[1..1]	<Anonymous>
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO1
customer	[0..1]	string
item	[0..1]	string
quantity	[0..1]	int
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	anyType
dynamicProperty	[0..1]	DynamicPropertyContextType
userContext	[0..1]	UserContextType
headers	[1..1]	HeadersType
body	[0..1]	checkInventoryResponseMsg
checkInventoryResponse	[1..1]	<Anonymous>
inventoryItem	[1..1]	InventoryItem
itemID	[0..1]	string
inStockQuantity	[0..1]	int
status	[0..1]	string
attachments	[0..*]	AttachmentType

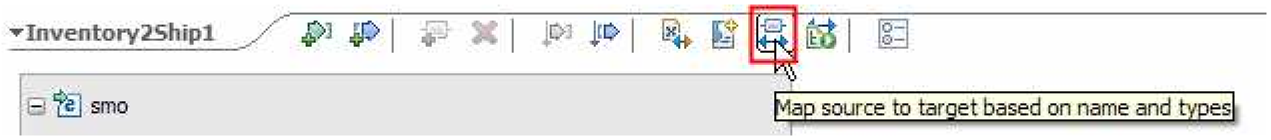
smo

<Click to filter...>

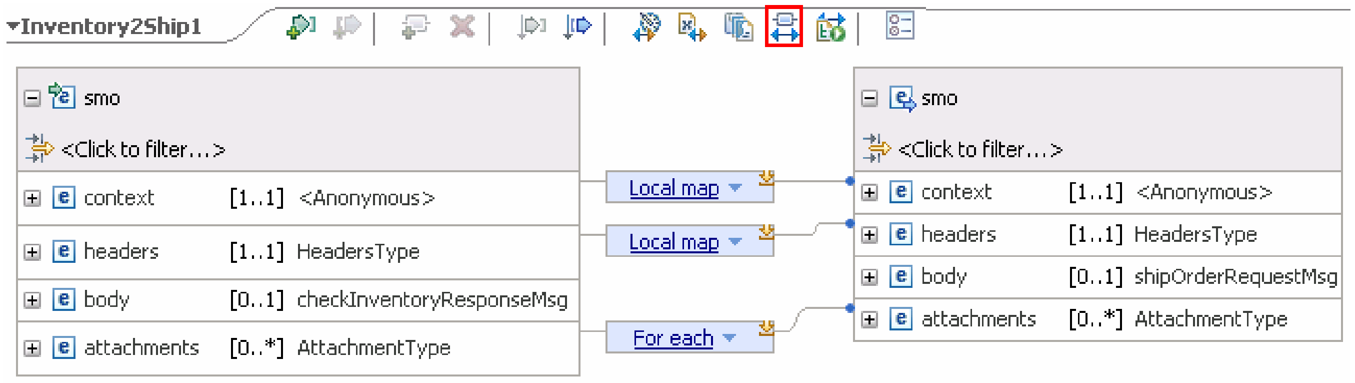
context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	shipOrderRequestMsg
shipOrder	[1..1]	<Anonymous>
ship	[1..1]	Ship
customerID	[0..1]	string
items	[0..*]	ShipItem
itemID	[0..1]	string
orderQuantity	[0..1]	int
inventoryQuantity	[0..1]	int
inventoryStatus	[0..1]	string
attachments	[0..*]	AttachmentType

- **`--------------------------------------------------------------------`**

__ a. Click '**Map source to target based on name and types**' icon



__ b. 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


Inventory2Ship1
smo
<Click to filter...>
context [1..1] <Anonymous>
headers [1..1] HeadersType
body [0..1] checkInventoryResponseMsg
attachments [0..*] AttachmentType
Local map
Local map
For each
smo
<Click to filter...>
context [1..1] <Anonymous>
headers [1..1] HeadersType
body [0..1] shipOrderRequestMsg
attachments [0..*] AttachmentType

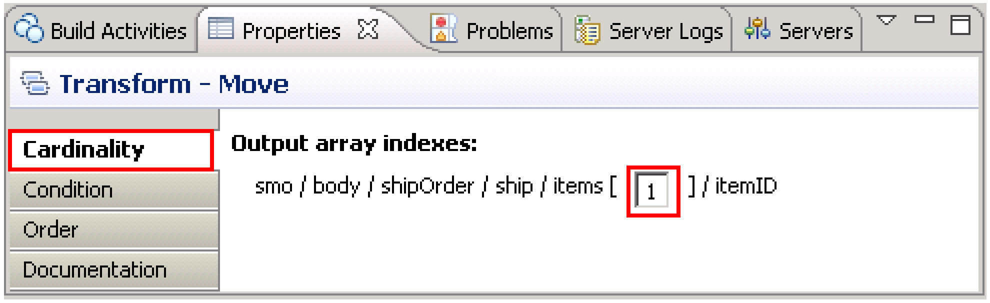
__ c. Move elements from the source transient context to the target message body.

1) In the source (left) smo, expand **context → transient**

2) In the target (right) smo, expand **body → shipOrder → ship → items**

3) Click **customer** in the source and drag to **customerID** in the target to create a **Move** transform.

4) Similarly, create a **Move** transform from **item** to **itemID**. Ensure that the **Move** transform is highlighted and set the array index to 1 in the **Cardinality** panel of the **Properties** view.

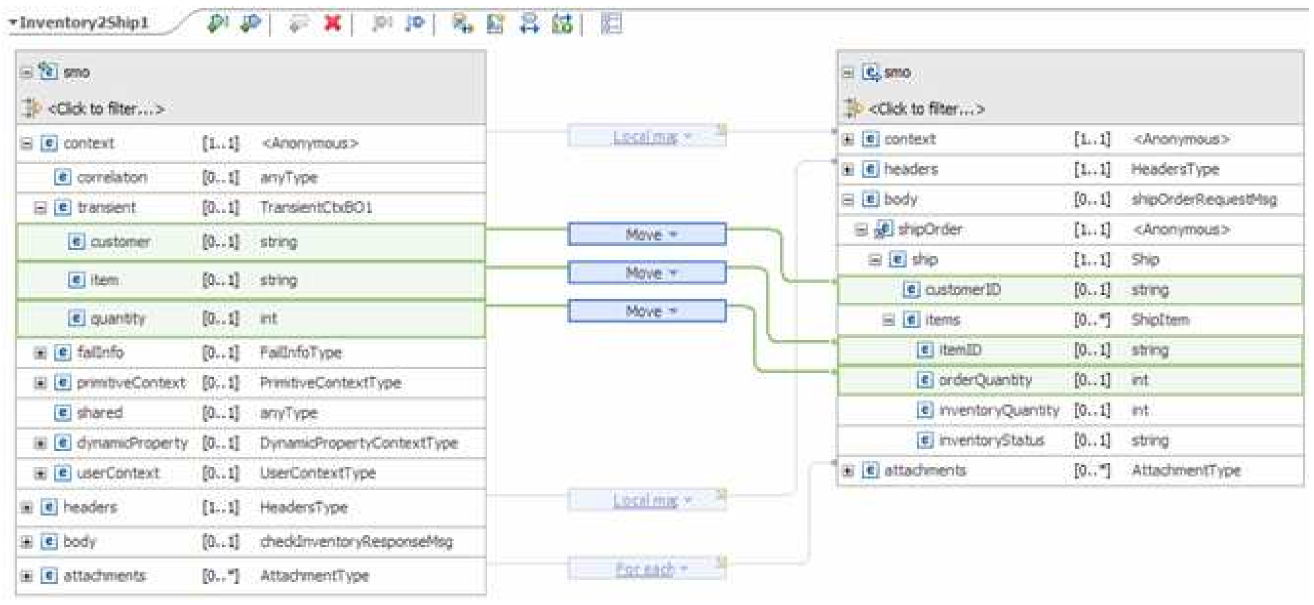


Output array indexes:

smo / body / shipOrder / ship / items [1] / itemID

5) Similarly, create a **Move** transform from **quantity** to **orderQuantity**. Set the cardinality to 1.

6) You map should now look like this:



context	[1..1]	<Anonymous>
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO1
customer	[0..1]	string
item	[0..1]	string
quantity	[0..1]	int
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	anyType
dynamicProperty	[0..1]	DynamicPropertyContextType
userContext	[0..1]	UserContextType
headers	[1..1]	HeadersType
body	[0..1]	checkInventoryResponseMsg
attachments	[0..*]	AttachmentType

context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	shipOrderRequestMsg
shipOrder	[1..1]	<Anonymous>
ship	[1..1]	Ship
customerID	[0..1]	string
items	[0..*]	ShipItem
itemID	[0..1]	string
orderQuantity	[0..1]	int
inventoryQuantity	[0..1]	int
inventoryStatus	[0..1]	string
attachments	[0..*]	AttachmentType

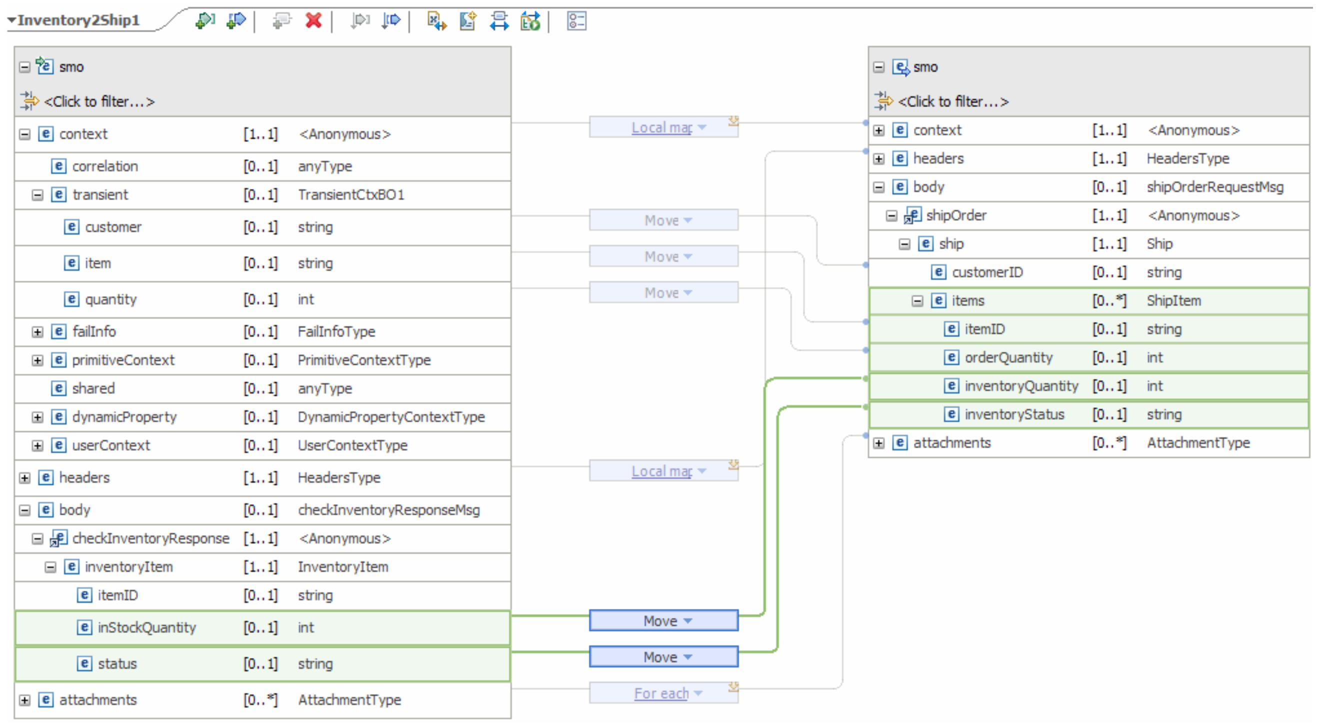
__ d. Move elements from the source message body to the target message body.

1) In the source (left) smo, expand **body → checkInventoryResponse → InventoryItem**

2) Create a **Move** transform from **inStockQuantity** to **inventoryQuantity**. Set the cardinality to 1.

3) Create a **Move** transform from **status** to **inventoryStatus**. Set the cardinality to 1.

4) The completed map should look like this:

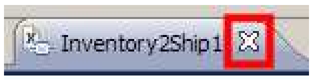


context	[1..1]	<Anonymous>
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO1
customer	[0..1]	string
item	[0..1]	string
quantity	[0..1]	int
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	anyType
dynamicProperty	[0..1]	DynamicPropertyContextType
userContext	[0..1]	UserContextType
headers	[1..1]	HeadersType
body	[0..1]	checkInventoryResponseMsg
checkInventoryResponse	[1..1]	<Anonymous>
inventoryItem	[1..1]	InventoryItem
itemID	[0..1]	string
inStockQuantity	[0..1]	int
status	[0..1]	string
attachments	[0..*]	AttachmentType

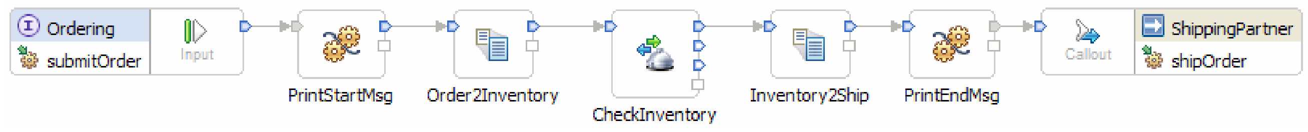
context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	shipOrderRequestMsg
shipOrder	[1..1]	<Anonymous>
ship	[1..1]	Ship
customerID	[0..1]	string
items	[0..*]	ShipItem
itemID	[0..1]	string
orderQuantity	[0..1]	int
inventoryQuantity	[0..1]	int
inventoryStatus	[0..1]	string
attachments	[0..*]	AttachmentType

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

__ f. Click **X** to close Inventory2Ship1.map file



____ 11. Ensure that the resulting flow looks like this:



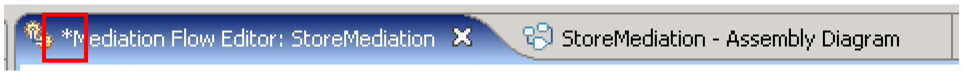
- --

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

____ 12. 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:


*Mediation Flow Editor: StoreMediation
StoreMediation - Assembly Diagram

__ 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.

____ 13. Check that there are no errors reported in the Problems view.

- --

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

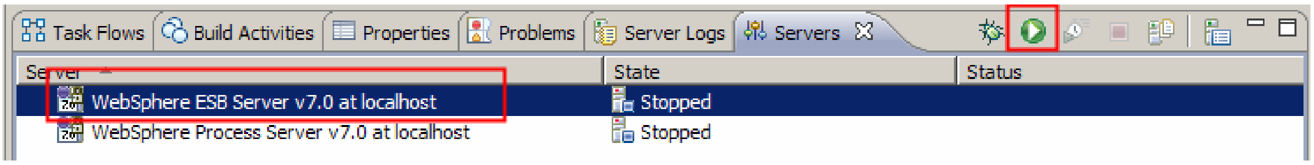
Part 3: Test the message augmentation mediation

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 inventory service implementation has already been provided and contains some hard-coded inventory data useful for testing. The resulting output from the test is explained. See the presentation entitled Augmentation, aggregation and retry labs to better understand what this part is doing.

____ 1. Start the WebSphere Enterprise Service Bus (or WebSphere Process Server) test server.

- -

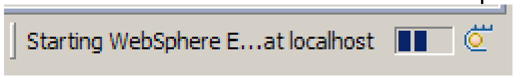
__ a. From the **Servers** view of WebSphere Integration Developer, select **WebSphere ESB Server v7.0** and hit **Start the server** icon () from the toolbar



Server	State	Status
WebSphere ESB Server v7.0 at localhost	Stopped	
WebSphere Process Server v7.0 at localhost	Stopped	

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

NOTE: During startup, the **Server Logs** or **Console** view might grab the focus from the **Servers** view. If this is the case, the status in the lower right should indicate if the server start is still in process. You can switch back to the **Servers** view when startup has completed.

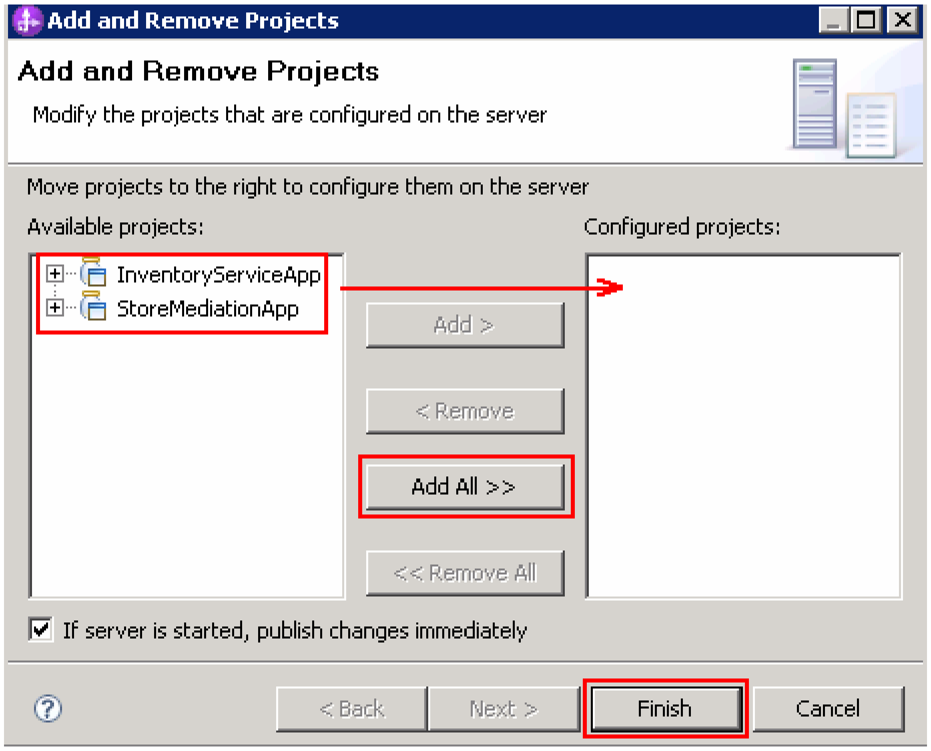

Starting WebSphere E...at localhost

____ 2. Add the InventoryServiceApp and StoreMediationApp to the test server using the Add and Remove Projects dialog from the server pop-up menu.

- -

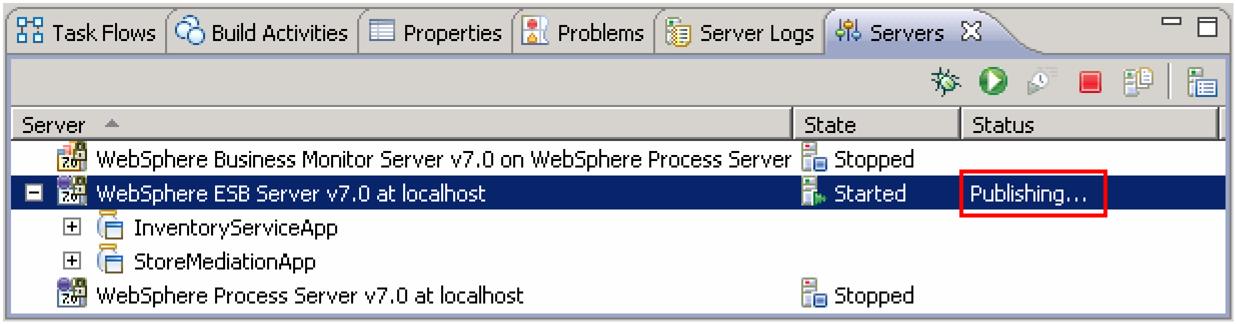
__ a. Right-click **WebSphere ESB Server v7.0** under the Servers view and select **Add and Remove Projects…** from the context menu

__ b. In the Add and Remove Projects window, click **Add All >>** to add the InventoryServiceApp and StoreMediationApp to the Configured projects panel



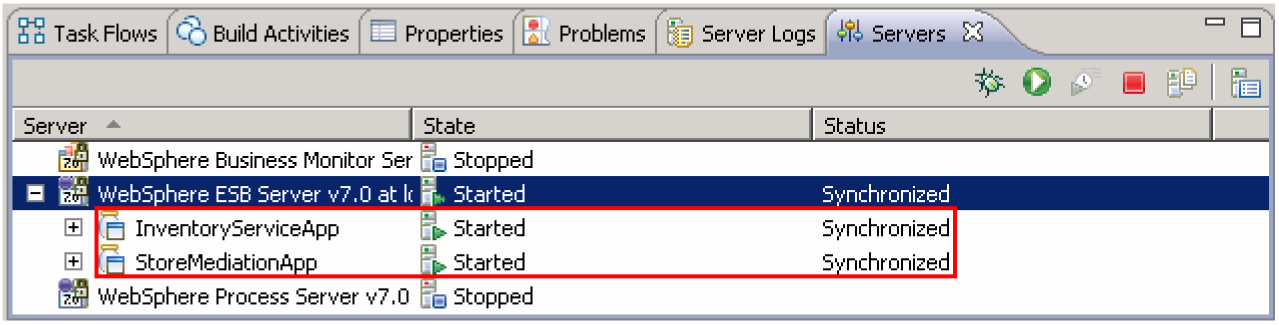
__ c. Click **Finish**

__ d. Click the **+** in front of the server to show the projects and wait while the projects are being published to the server



Server	State	Status
WebSphere Business Monitor Server v7.0 on WebSphere Process Server	Stopped	
WebSphere ESB Server v7.0 at localhost	Started	Publishing...
InventoryServiceApp		
StoreMediationApp		
WebSphere Process Server v7.0 at localhost	Stopped	

__ e. Once the publishing is done you should see the 2 applications started as shown here:



Server	State	Status
WebSphere Business Monitor Ser	Stopped	
WebSphere ESB Server v7.0 at l	Started	Synchronized
InventoryServiceApp	Started	Synchronized
StoreMediationApp	Started	Synchronized
WebSphere Process Server v7.0	Stopped	

____ 3. From the assembly diagram for StoreMediation, start Test Component for the StoreMediation component

- ---

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

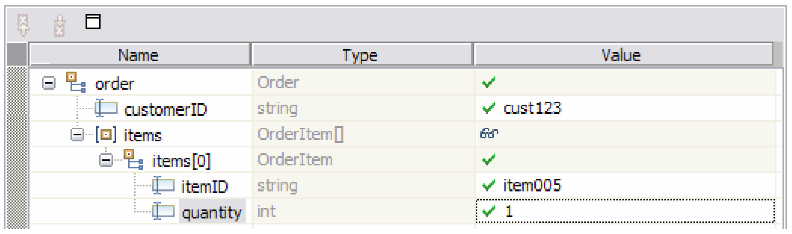
__ b. From the StoreMediation-Assembly Diagram, right-click the **StoreMediation** component and select **Test Component** from the pop-up menu

__ c. The **StoreMediation_Test** window is opened where you enter your test data

____ 4. Initialize the test data

- **`Set customerID to cust123`**
- **`Set items/items[0]/itemID to item005`**
- **`Set items/items[0]/quantity to 1`**

NOTE: To create elements in the items array, right click anywhere on the row and select **Add Elements…** from the pop-up menu, then enter **1** for the number of elements to add.

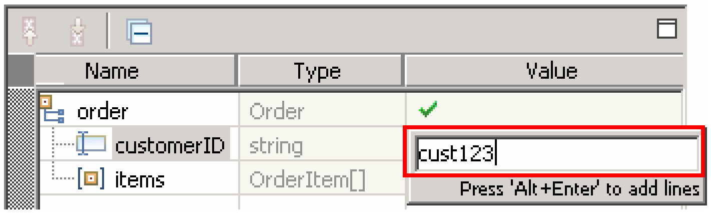


Name	Type	Value
order	Order	✓
customerID	string	✓ cust123
items	OrderItem[]	
items[0]	OrderItem	✓
itemID	string	✓ item005
quantity	int	✓ 1

- ---

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

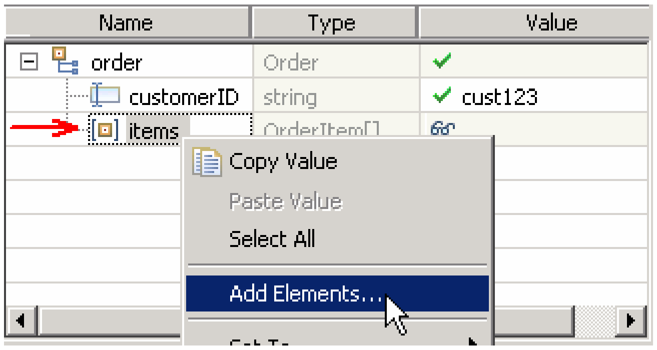
1) For **customerID**, click under Value and enter **cust123**



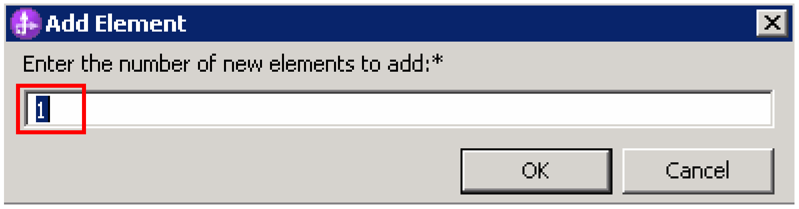
Name	Type	Value
order	Order	✓
customerID	string	cust123
items	OrderItem[]	

Press 'Alt+Enter' to add lines

2) Right-click any where on the row containing **items** and select **Add Elements…** from the pop-up menu



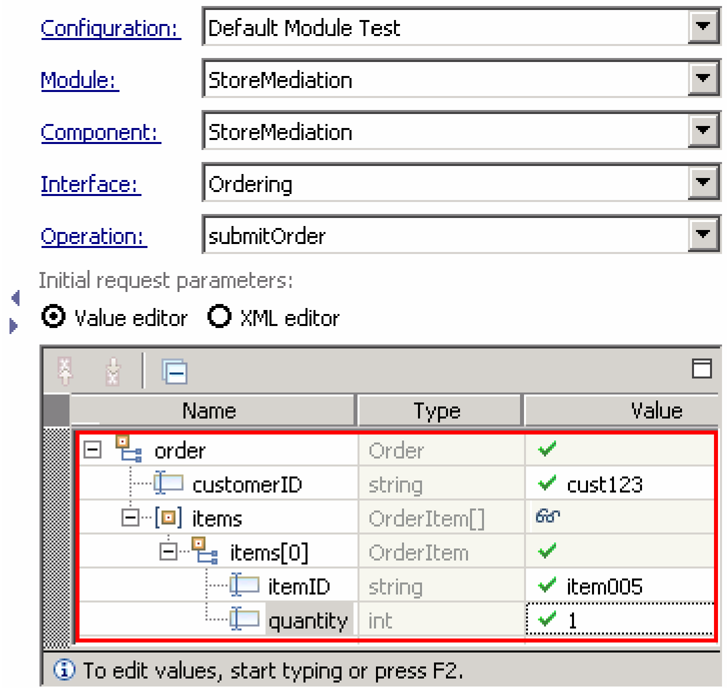
3) Enter '**1**' (default) in the Add Element window:



4) Click **OK**

5) For **itemID**, click under Value and enter **item005**

6) For **quantity**, click under Value and enter **1**



Configuration: Default Module Test

Module: StoreMediation

Component: StoreMediation

Interface: Ordering

Operation: submitOrder

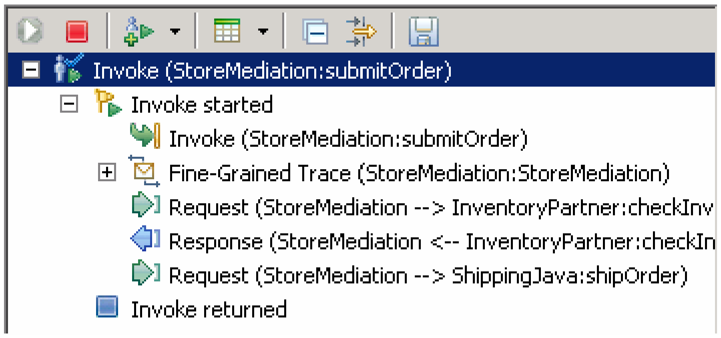
Initial request parameters:

[✓ Value editor] XML editor

Name	Type	Value
order	Order	✓
customerID	string	✓ cust123
items	OrderItem[]	
items[0]	OrderItem	✓
itemID	string	✓ item005
quantity	int	✓ 1

To edit values, start typing or press F2.

____ 5. Run the test by clicking the Continue icon ( ). Results should look like the picture shown below::


Invoke (StoreMediation:submitOrder)
Invoke started
Invoke (StoreMediation:submitOrder)
Fine-Grained Trace (StoreMediation:StoreMediation)
Request (StoreMediation --> InventoryPartner:checkInv
Response (StoreMediation <-- InventoryPartner:checkIn
Request (StoreMediation --> ShippingJava:shipOrder)
Invoke returned

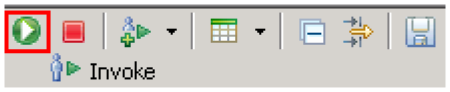
NOTE: If presented with the Deployment Location dialog, select **WebSphere ESB Server V7.0 at localhost** (or whatever server you added the projects to).

NOTE: If presented with the **User Login** dialog, set the **User ID** and **Password** (defaults are **admin** and **admin**). You should select the **'Use the authentication settings in the preference and never ask again'** check box to prevent this dialog from being displayed in the future.

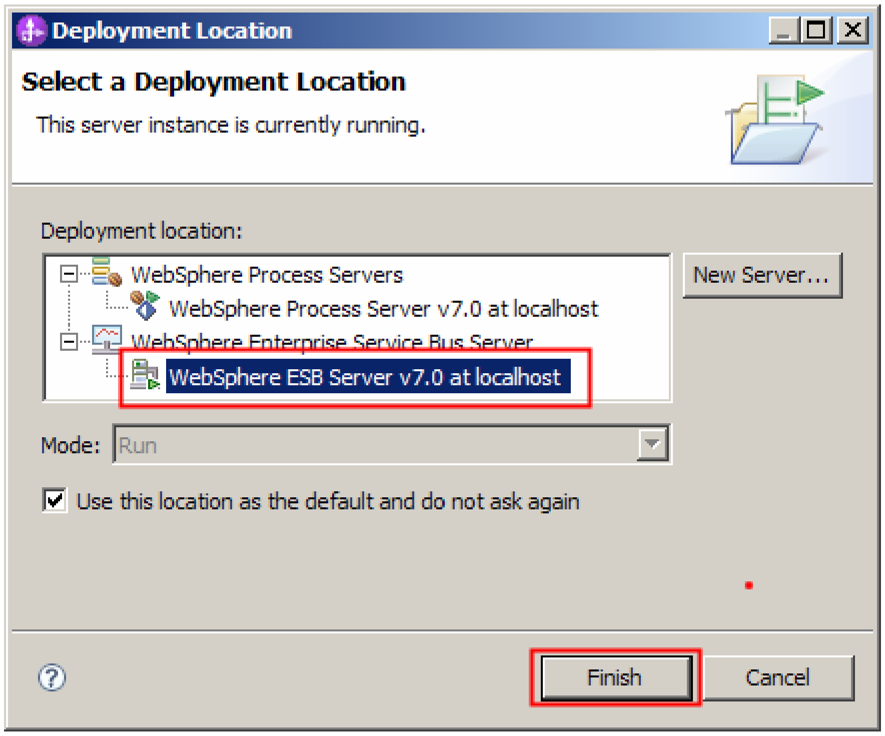
NOTE: The Fine-Grained Trace section has been collapsed. When expanded, it shows all of the nodes and primitives invoked by the flow. Although not discussed or explained in this lab, clicking on any of the entries within the fine grained trace will show the contents of the SMO at that point in the flow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

- --

__ a. Click the **Continue** icon ( ) under Events panel


Invoke

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



Select a Deployment Location

This server instance is currently running.

Deployment location:

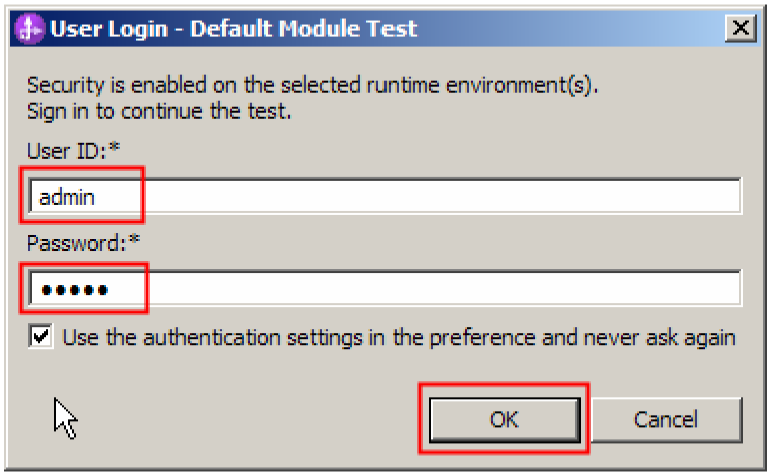
- WebSphere Process Servers
  - WebSphere Process Server v7.0 at localhost
- WebSphere Enterprise Service Bus Server
  - WebSphere ESB Server v7.0 at localhost

Mode: Run

☑ Use this location as the default and do not ask again

__ 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.



Security is enabled on the selected runtime environment(s).
Sign in to continue the test.

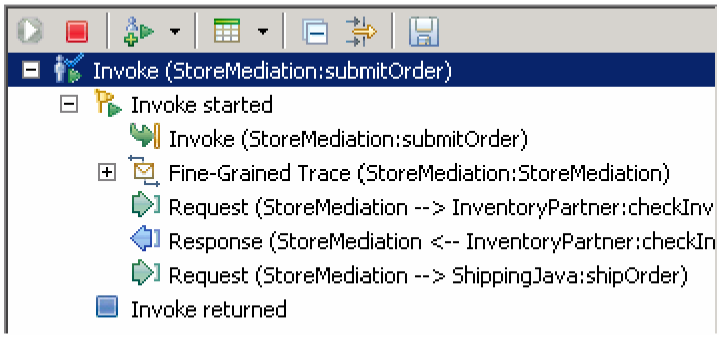
User ID:* admin

Password:* •••••

☑ Use the authentication settings in the preference and never ask again

__ e. Click **OK**. Wait until the integration test client starts

__ f. You should see the following results (there will be additional lines showing if the twistie for Fine-Grained Trace is open):



```
Invoke (StoreMediation:submitOrder)
    Invoke started
        Invoke (StoreMediation:submitOrder)
        Fine-Grained Trace (StoreMediation:StoreMediation)
        Request (StoreMediation --> InventoryPartner:checkInv
        Response (StoreMediation <-- InventoryPartner:checkIn
        Request (StoreMediation --> ShippingJava:shipOrder)
    Invoke returned
```

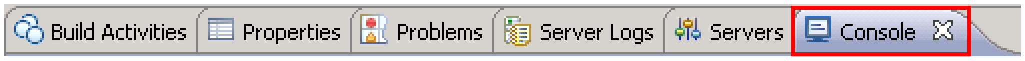
____ 6. Check the output written to the console

- **`Open (if not present) or switch to the Console view`**
- **`Scroll to the bottom and examine the output which should look like this:`**

```
O ******************************************
O ********** START mediation flow **********
I   processMessage
O ***** InvWorks  - returning InventoryItem for itemID = item005
I   processMessage
O ***
O ********** END mediation flow ************
O ******************************************
O ----------------------------------------
O -- Ship object dump begins -------------
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@671a671a
O Value:
O     customerID = cust123
O     items = ShipItem[1]
O         items[0] = <ShipItem@67366736>
O             itemID = item005
O             orderQuantity = 1
O             inventoryQuantity = 25
O             inventoryStatus = OK - sufficient stock levels
O -- Ship object dump ends   -------------
O ----------------------------------------
```

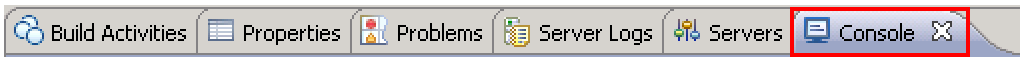
- **`---------------------------------------------------------------------`**

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

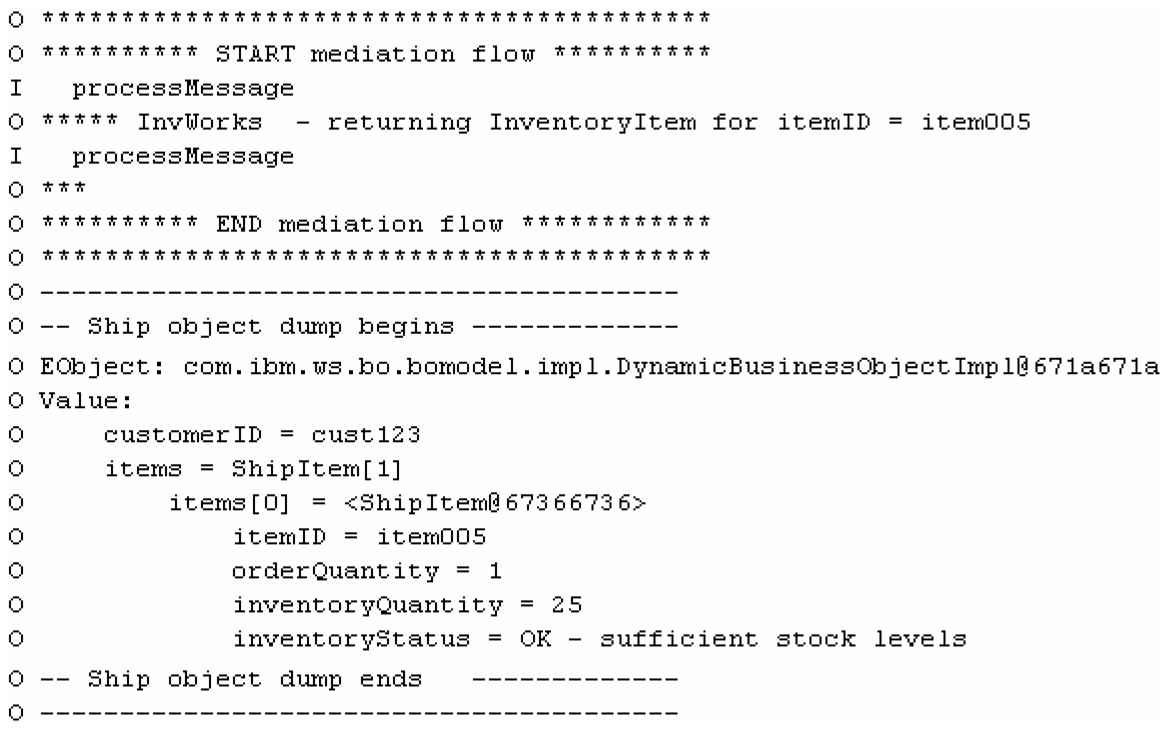


__ b. If it is not showing, open it by selecting menu items **Window → Show View → Console**

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



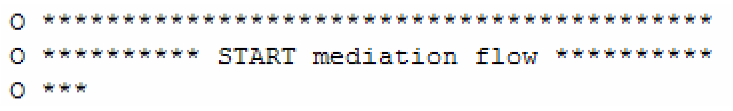
__ d. Scroll to the bottom of the console to see the output from the test run.



```
O ******************************************
O ********** START mediation flow **********
I   processMessage
O ***** InvWorks  - returning InventoryItem for itemID = item005
I   processMessage
O ***
O ********** END mediation flow ************
O ******************************************
O ----------------------------------------
O -- Ship object dump begins -------------
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@671a671a
O Value:
O     customerID = cust123
O     items = ShipItem[1]
O         items[0] = <ShipItem@67366736>
O             itemID = item005
O             orderQuantity = 1
O             inventoryQuantity = 25
O             inventoryStatus = OK - sufficient stock levels
O -- Ship object dump ends   -------------
O ----------------------------------------
```

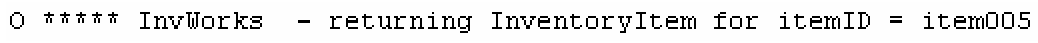
____ 7. Being able to examine and understand the Console output is important throughout this series of labs. To that end, the output above is explained here:

- **`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 labs examining service invoke retry capabilities).`**



```
O ***** InvWorks  - returning InventoryItem for itemID = item005
```

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. It is a dump of the input business object Ship which is composed of a customerID and an array of ShipItem. Each ShipItem in the array is composed of information for the item from the Order and additional data for that item from the inventory service. In subsequent labs in this series, there are multiple elements in this array.`**

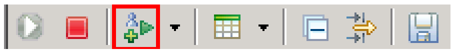
```
O -- Ship object dump begins -------------
O EObject: com.ibm.ws.bo.bomodel.impl.DynamicBusinessObjectImpl@671a671a
O Value:
O     customerID = cust123
O     items = ShipItem[1]
O         items[0] = <ShipItem@67366736>
O             itemID = item005
O             orderQuantity = 1
O             inventoryQuantity = 25
O             inventoryStatus = OK - sufficient stock levels
O -- Ship object dump ends   -------------
```

- **`----------------------------------------------------------------------`**

____ 8. You can run additional tests using different input data to see how the output varies. Key things to note about the data you use:

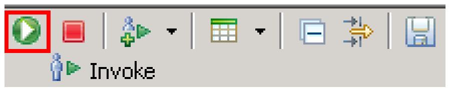
- **`customerID can be any string and should not have any particular affect on the results`**
- **`items should always be a 1 element array`**
- **`itemID values of “item001” through “item010” are recognized by the inventory service, all other values are not recognized`**
- **`inventory status will change according to the relationship between the order and inventory quantities`**
- **`----------------------------------------------------------------------`**

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



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

__ c. Click the **Continue** icon ( ) under Events panel



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

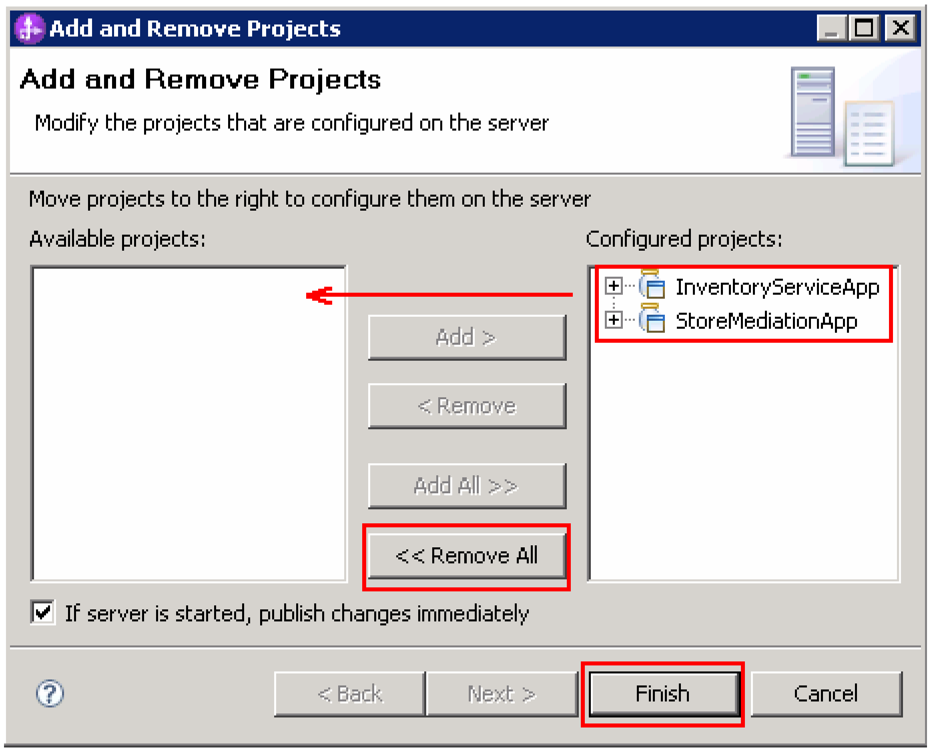
Perform this part only if you are not continuing to the message splitting and aggregation lab (the second lab in this series).

____ 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, click **<< Remove All**



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

__ d. Wait until the applications are removed from the server

____ 2. Close the StoreMediation_Test panel without saving

- --

__ a. Close the StoreMediation_Test tab

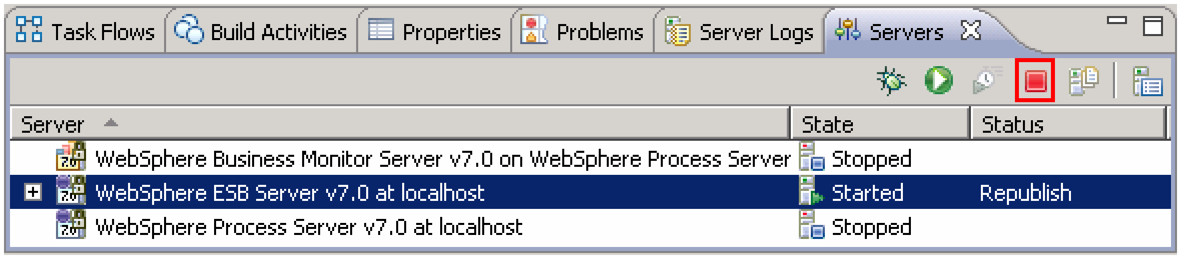
__ b. Click **No** from Save Test Trace window

__ c. Close the StoreMediation – Assembly Diagram tab

____ 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



Server	State	Status
WebSphere Business Monitor Server v7.0 on WebSphere Process Server	Stopped	
WebSphere ESB Server v7.0 at localhost	Started	Republish
WebSphere Process Server v7.0 at localhost	Stopped	

__ b. Wait until the server Status shows as **Stopped**

____ 4. Exit from WebSphere Integration Developer.

- --

__ a. From menu, select **File → Exit**

What you did in this exercise

In this exercise, you used a service invoke primitive to obtain data used to augment a message. In order to use the service invoke primitive, you needed to use XSL transformation primitives before and after to modify the message type and correctly set the data into the message.

Reviewing the presentation entitled **Augmentation, aggregation, and retry labs** 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>/PI2-AugmentSolution-AggregateStart.zip`**

____ 2. Skip to **Part 3: Test the message augmentation mediation** and proceed through the rest of the lab.