Service Bus (12.1.3) Tutorial

Version 1.0

September 2014

Sivakumar Gonugunta

Contents

Overview of Tutorial

The purpose of this tutorial is to get hands-on experience with Service Bus 12c development using JDeveloper.

The tutorial assumes a little conceptual understanding of Service Bus such as Proxy Service, Business Service and Message Flows. If you don't have this initial understanding, it is recommended to go through the section 'Service Bus Components' in this link. By the time you finish this tutorial, you will learn:

- Creating Service Bus Application, Project, Proxy Service and Business services.
- Working with resources such as WSDL, XSD, XSLT, XQuery, Service Account etc.
- Sharing the OSB resources across Service Bus projects to enable reuse and modularity.
- Creating Pipelines, Pipeline Templates and using nodes like Pipeline Pair, Stage, Routing and Operational Branches.
- Using Domain Value Maps (DVM) for storing configuring properties and using them in message flow.
- Creating Service Accounts and accessing them in message flow at runtime.
- Enriching the messages using intermediate service calls in Pipelines.
- Routing messages to business services using Service Callout, Routing and Routing Table.
- Working with Split-Joins.
- Using service error handlers in Pipelines.
- Deploying and Testing Proxy and Business services.
- Securing Proxy service.

Scenario in Tutorial

XYZ Bank is a Financial Services organization operates in several countries having numerous branches. Very recently, the bank is recording disappointing results for the last few quarters and constantly losing customers satisfaction. The top brass management identified issues with current IT infrastructure lacking agility which is unable to meet growing business and customer needs. The recent acquisitions also had its share of new IT systems getting added to existing system landscape.

Because of this, the management has decided to go with revamp of existing IT infrastructure. As part of this, an integration strategy is being chalked out using SOA as the underlying architecture to improve agility and bring in more reusability as customers uses different channels to avail bank services. Added to this, a new portal is also been planned to improve user experience.

In the proposed architecture, an Enterprise Service Bus plays pivotal role providing virtualization layer to consumers and also takes care of Validation, Transformation and routing all incoming requests to the actual services provided by end (legacy) systems. And Oracle Service Bus (OSB) has been chosen meeting the requirements.

Business Requirements

For milestone 1, IT team wants to expose data services on different objects Customer, Accounts etc. These services will be exposed to consumers using Service Bus that provides features like Service Virtualization, Authentication, Validation, Transformation, Message Enrichment and

Routing. In this tutorial, we will develop such data services for Customer data object and the requirements are listed below.

Create a Service Bus Proxy Service exposing single interface to perform the following:

CreateCustomer - Creates a new customer.
UpdateCustomer - Updates an existing customer.
DeleteCustomer - Deletes an existing customer.
MergeCustomer - Updates an existing customer if exists or create a new customer.

Common Requirements

1. Single interface (WSDL) to be exposed for all above operations.
2. Transform the input request to the structure expected by the actual end system.
3. Route the input request to appropriate business service by selecting suitable operation.
4. In positive case, return the output containing **Customer ID** and **Status.**
5. In all negative cases, return the fault containing **Error Code** (in any) and **Error Message**.
6. **Secure** the proxy service using **OWSM policy** so that only valid users can invoke.

CreateCustomer

- The input structure should include the attributes First Name, Last Name, Email, Phone, Date of Birth, Customer Type and Address fields. Restrict the Customer Type values to 'Individual' and 'Corporate'.
- Validate the input against XSD definition and return fault on validation failure.
- Validate **Email** element for proper format and return fault on validation failure.
- Default **CustomerType** element to '**Individual**', if it's not sent in the input.
- Validate **Address** elements by calling the **Address Validation** service and return fault if address is not valid. Address validation should be performed based on a configurable property.
- The actual legacy system expects **CustomerID** element to be populated in the input. Call **Create User ID** service to get unique id and populate it as **CustomerID**.
- Create User ID service expects the consumers to send credentials in SOAP header in the following format. Use Service Accounts to store these credentials.

```
<usr:UserIDConfiguration
xmlns:usr="http://xmlns.xyzbank.com/User/Configuration">
   <usr:UserName>admin</usr:UserName>
   <usr:Password>password1</usr:Password>
</usr:UserIDConfiguration>
```

- Complete the common requirements 3-5 mentioned above.

UpdateCustomer

- The input structure should be same as **CreateCustomer** operation.
- Validate the input against XSD definition and return fault on validation failure.
- Complete the common requirements 3-5 mentioned above. Since the actual end system updates the fields for which values are sent, the elements with null values have to be removed from input before routing the request.

DeleteCustomer

- The input structure should have **CustomerID** as mandatory attribute.
- Validate the input against XSD definition and return fault on validation failure.

- Complete the common requirements 3-5 mentioned above.

MergeCustomer

- The input structure should be same as **CreateCustomer** operation.
- Call f**ind_customer** in **Customer** service to find existence of Customer using **CustomerID**. The service response element '**CustomerExists'** value 'Y' denotes existing customer and value 'N' denotes non-existing customer .
- For existing Customer, route request to perform **Update**.
- In case no Customer exists, route request to perform **Create**. Make sure that all of the above requirements mentioned for '**CreateCustomer'** are considered.
- Complete common requirements 3-5 mentioned above.

MigrateCustomer

- The input structure should have **CustomerID** as mandatory attribute.
- Validate the input against XSD definition and return fault on validation failure.
- Call **Customer**, **Customer Order** and **Customer Accounts** services in parallel to fetch Customer, Orders and Accounts information respectively.
- Call **migrate_customer** in **Customer** service to migrate customer information.
- Complete common requirements 3-5 mentioned above.

Error Handling

- Convert System or Application errors to the SOAP Fault structure as defined in WSDL.

Following are the WSDLs to be used for business services creation:

Purpose	Service WSDL
Unique ID Creation	CreateUserIDService.wsdl
Address Validation	AddrValidationService.wsdl
Customer Object operations	CustomerService.wsdl
Customer Order operations	CustomerOrderService.wsdl
Customer Account operations	CustomerAccountService.wsdl

You can create SOAP UI mock services using the above WSDLs or your own implementation to imitate behavior of the actual services exposed by service provider. This tutorial uses these mock services for demonstration. Refer to SOAP UI documentation here for more information on **Mock Services**.

The mock service used for this tutorial can be downloaded from here. Modify the payload for different operations as required for both positive and negative test cases described later in this tutorial.

The Configuration jar of this tutorial can be downloaded from here.

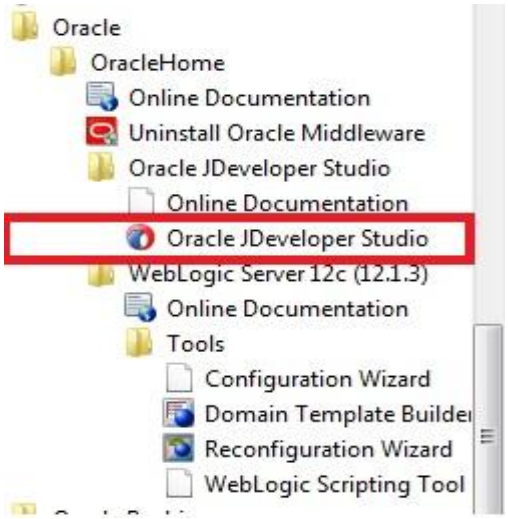
Installation

This tutorial uses Service Bus 12.1.3 for demonstration purpose and this is the first release of SOA Suite 12c and also the first release where Service Bus development is integrated into JDeveloper. For detailed installation steps of SOA Suite 12c, you can refer to this link.

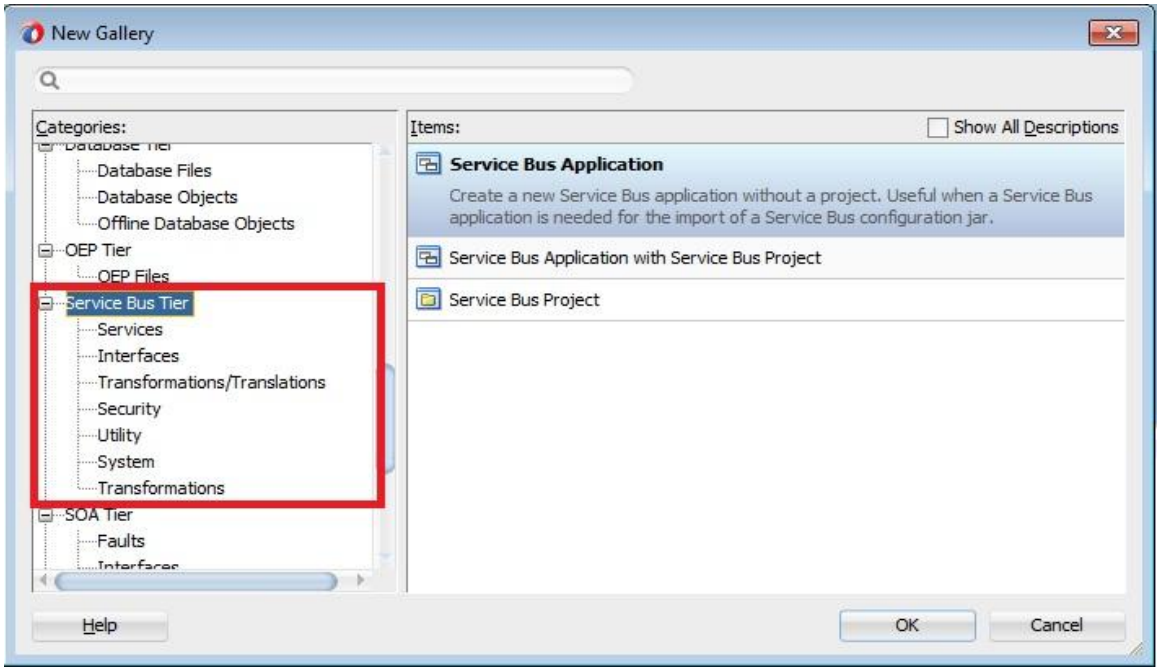
You will use **JDeveloper 12.1.3** for this tutorial, which gets automatically installed as part of SOA Suite 12c installation.

Verifying the Installation

Open JDeveloper IDE from SOA Suite 12c installation.

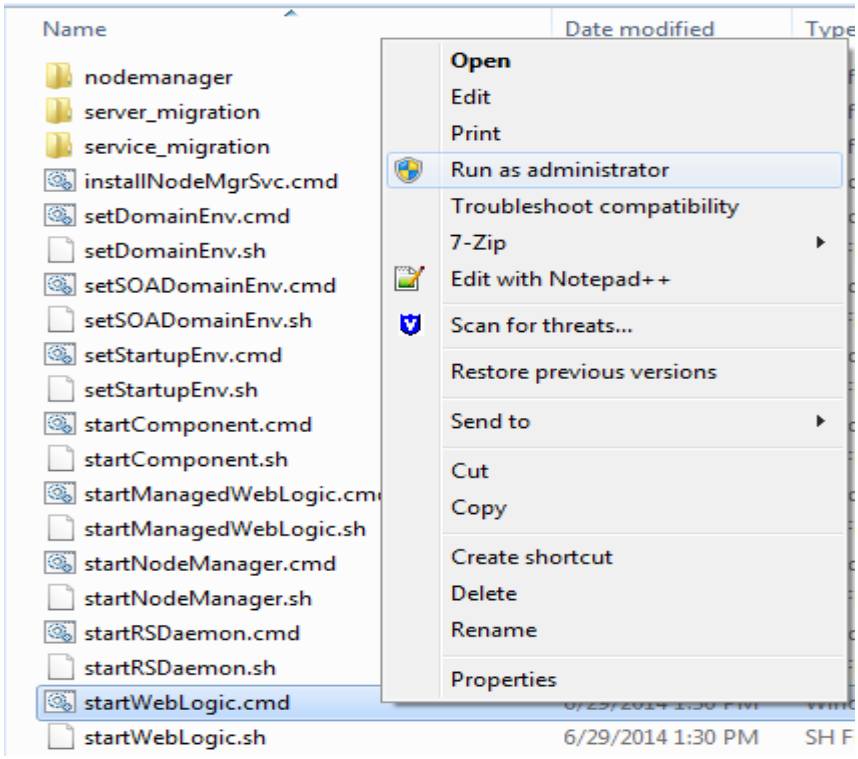


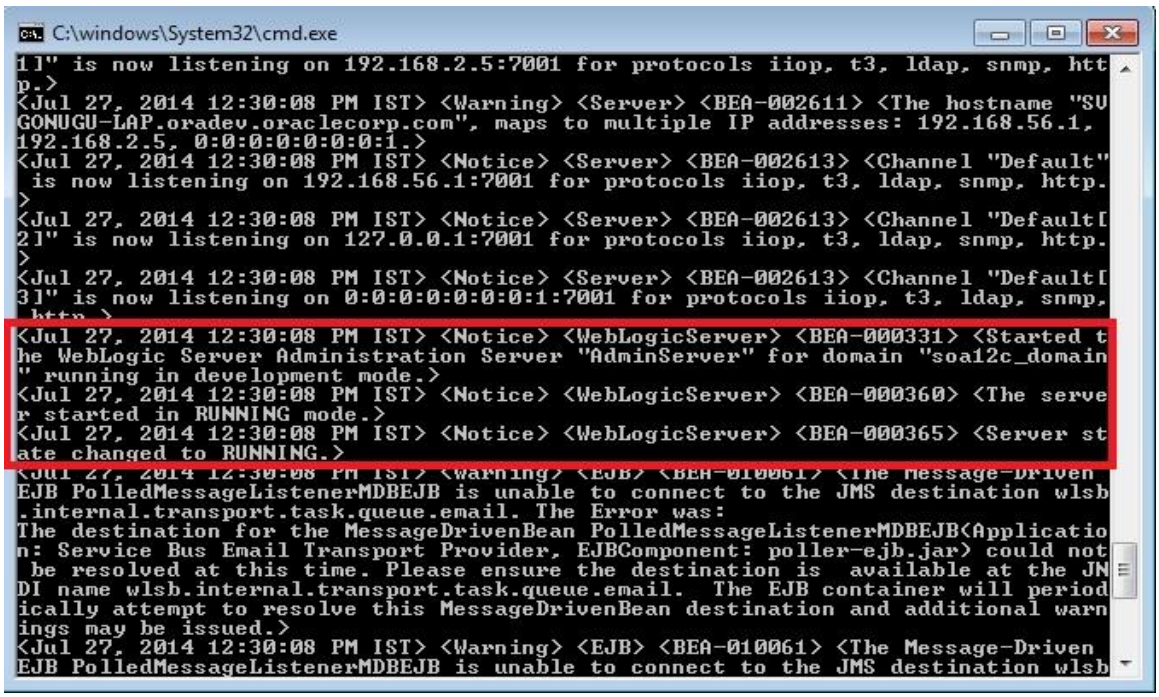
Go to **File -> New** and verify that **Service Bus Tier** is available.



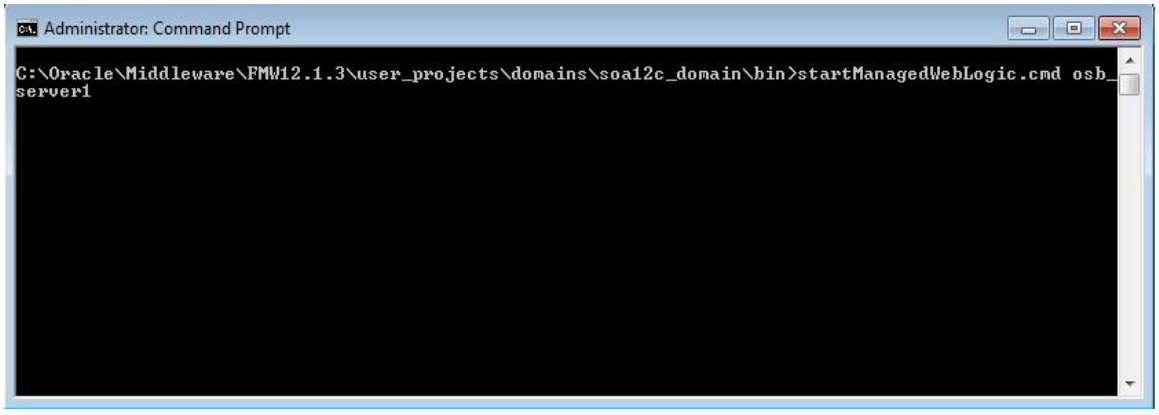
Create a SOA domain using steps given here. Navigate to **DOMAIN_HOME** and start WLS Admin server by executing **startWebLogic.cmd**.

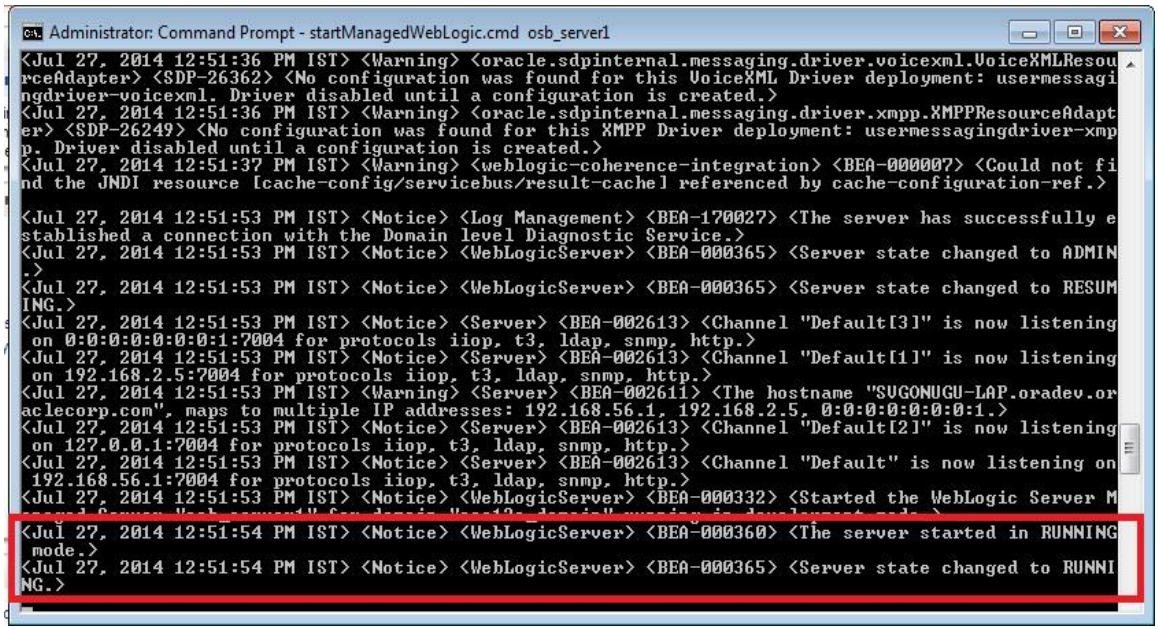
Note: *Going forward we will use the terms DOMAIN_HOME, WLS_HOME, and MW_HOME to refer to domain directory, WLS installation directory and Service Bus installation directory respectively.*



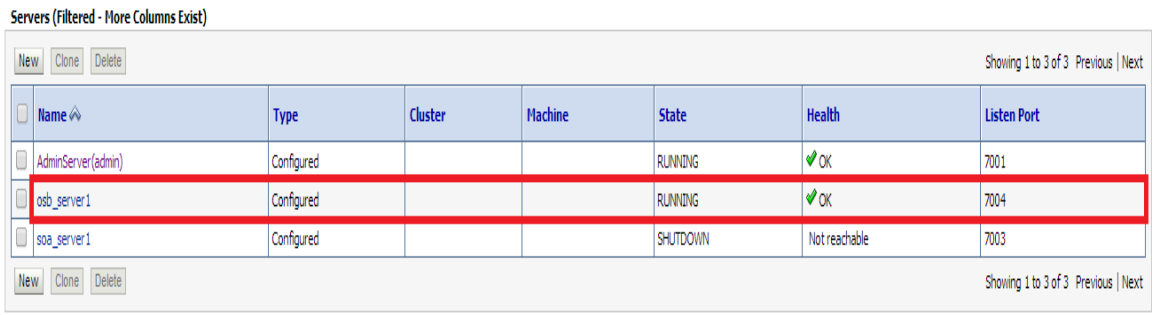


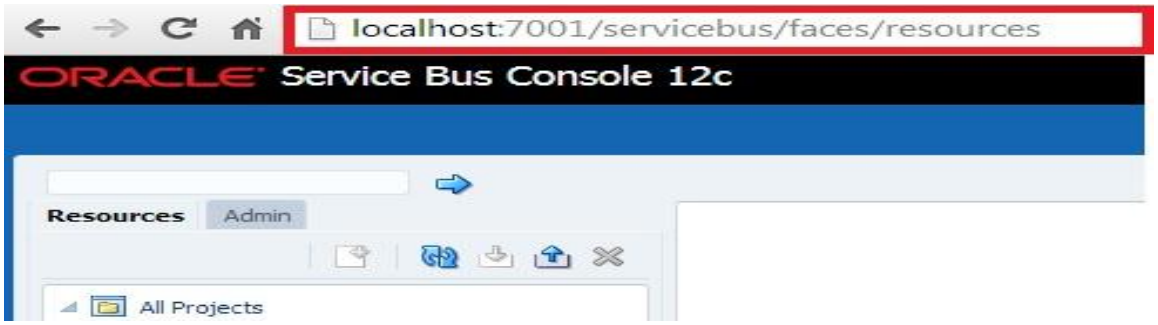
12c domain creation creates managed servers for both SOA and OSB by default, so you should start Managed server too. Navigate to DOMAIN_HOME and start managed server by executing **startManagedWeblogic.cmd** as shown below and give **weblogic** credentials on prompt.





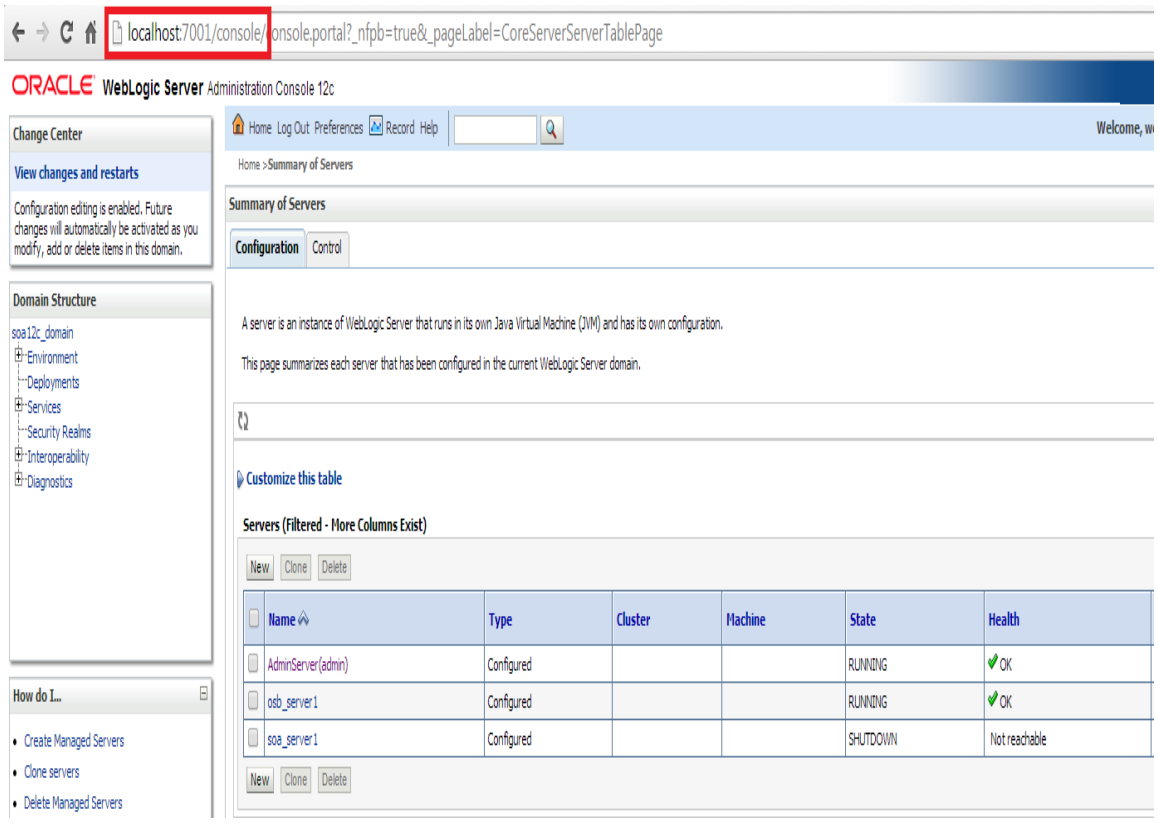
Verify that OSB server is up and you are able to access Service Bus console using URL **http://localhost:<port>/sbconsole** with credentials given during domain creation. Note that **sbconsole** is accessible even when just **AdminServer** is running but to test or deploy any Service Bus project you should start corresponding OSB managed server.





In 12c, **sbconsole** URL will be translated to url as shown above, when it's accessed.

You can also verify WLS **Admin** console by launching URL **http://localhost:<port>/console** with credentials given during domain creation.

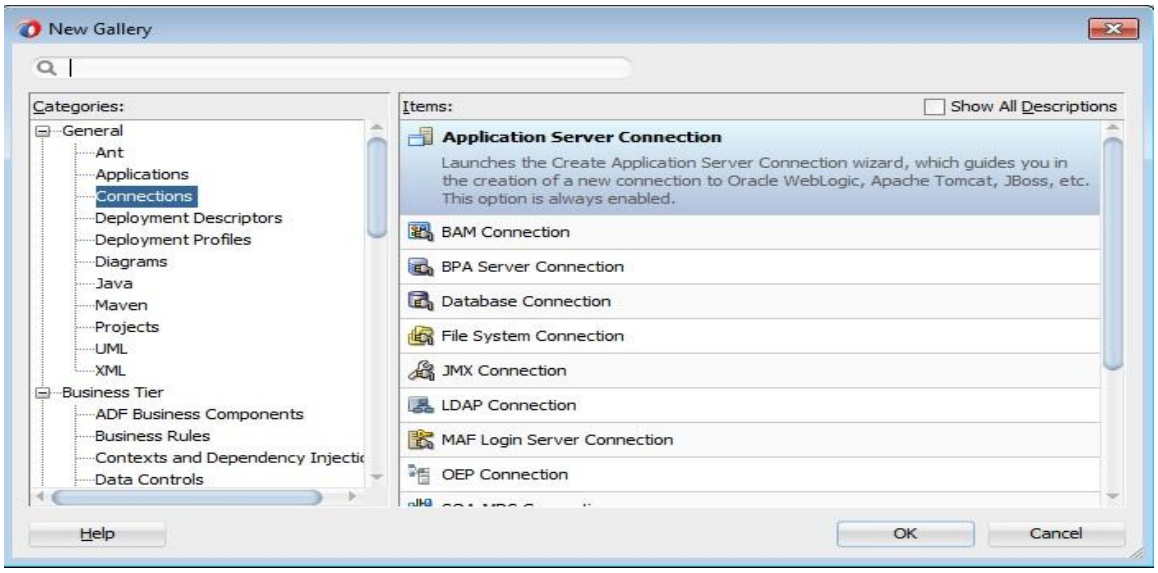


Getting Started

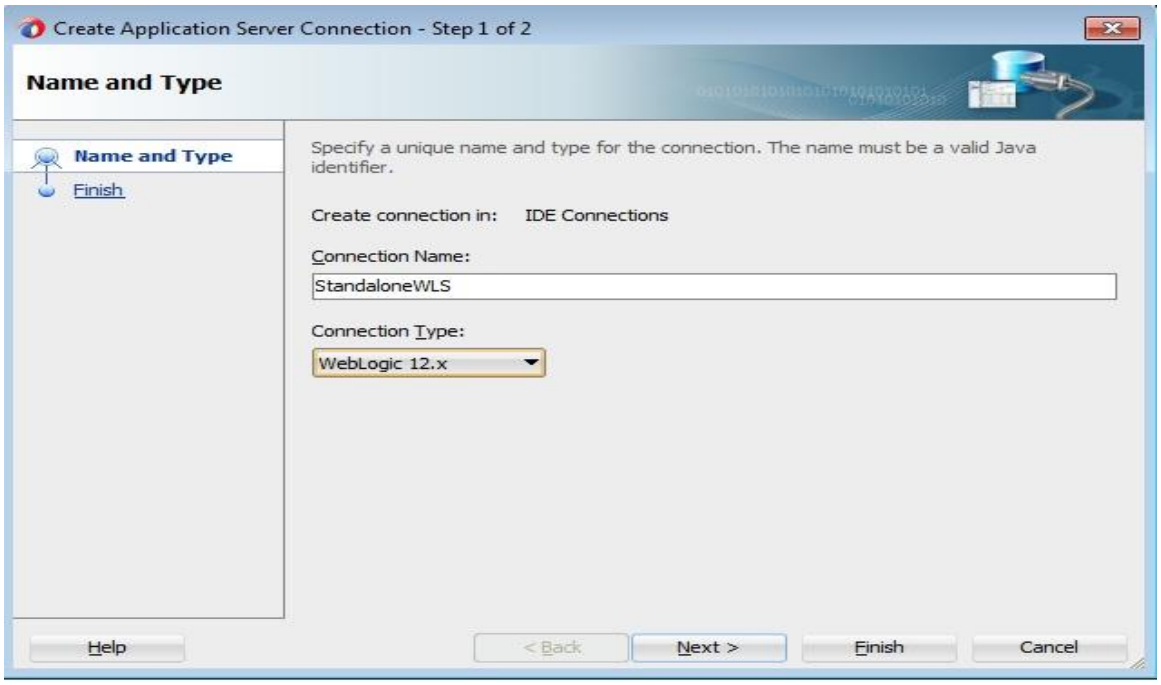
As mentioned, you will be using **JDeveloper IDE** for Service Bus development. You can create **Weblogic server connection** in JDeveloper itself to make the deployment easier or you can use integrated Weblogic server,

Adding Weblogic Server in IDE

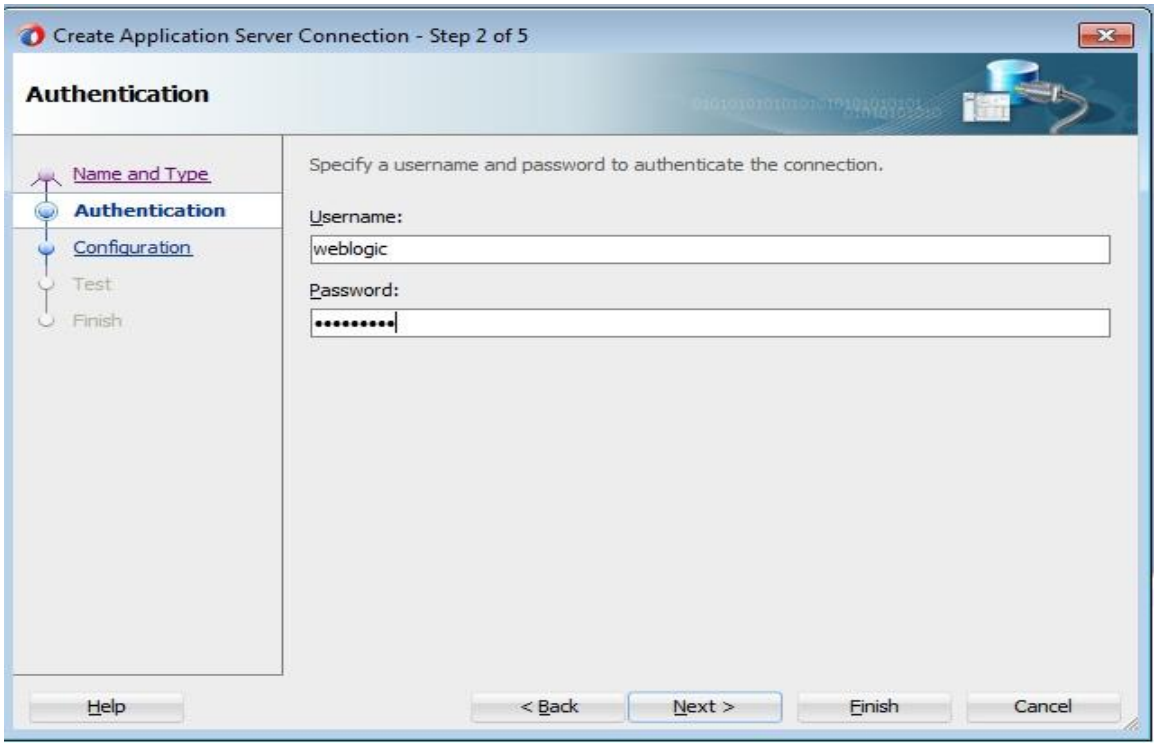
Go to **File -> New -> From Gallery** and navigate to **General -> Connections**. Select **Application Server Connection** and click **OK**.



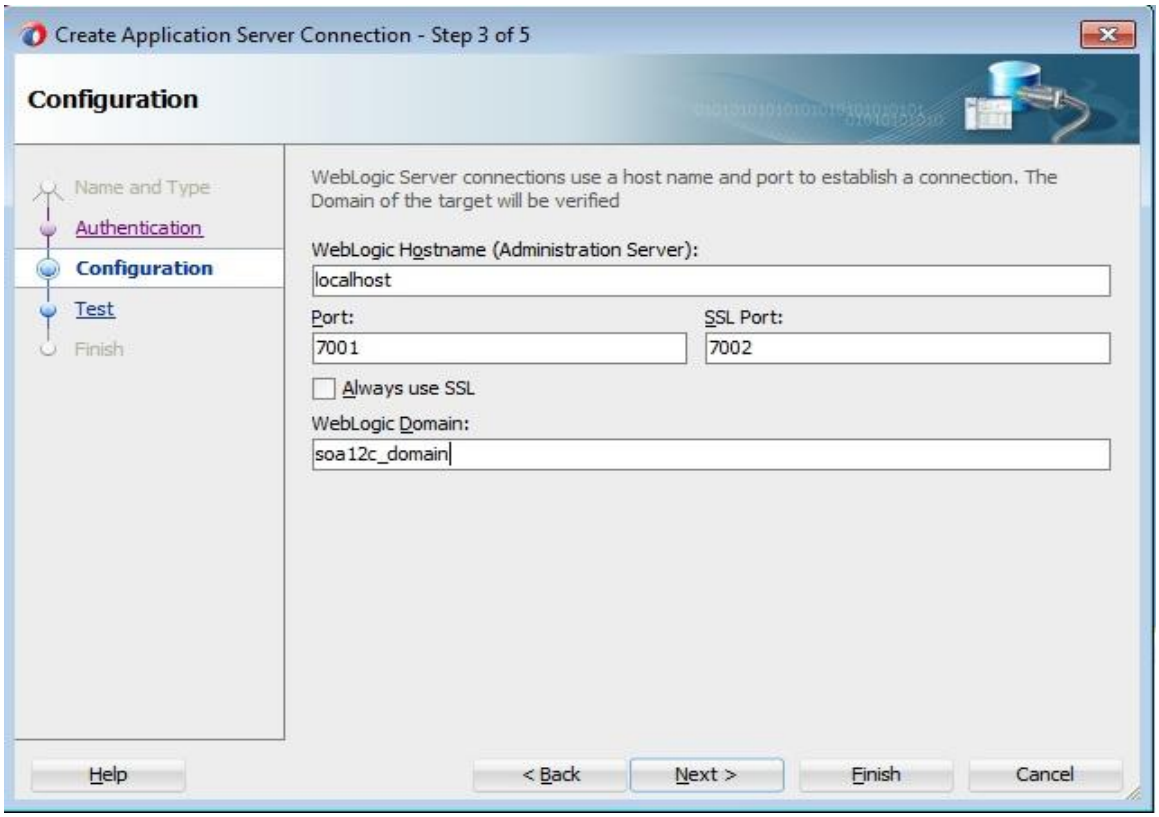
Enter **Connection Name** and select appropriate **Connection Type** as shown below**.**



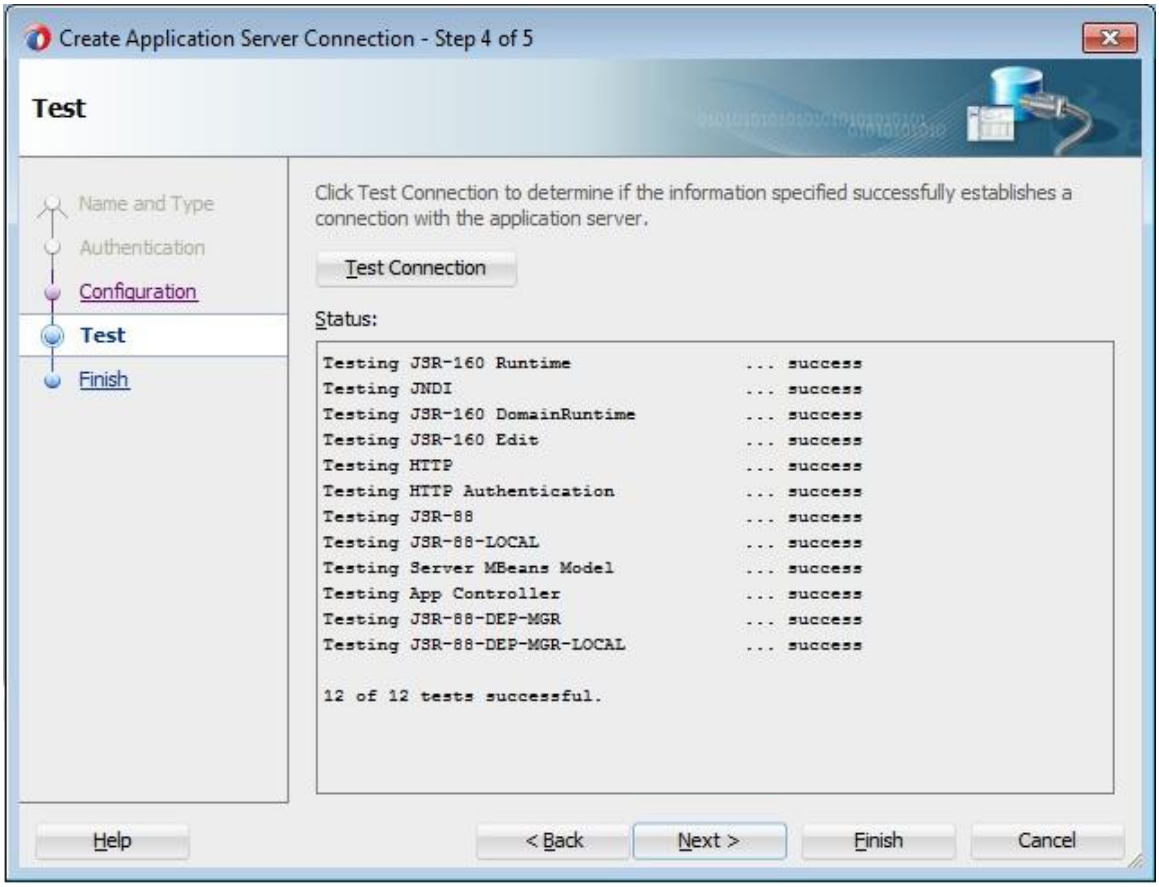
Click **Next** and give valid credentials.



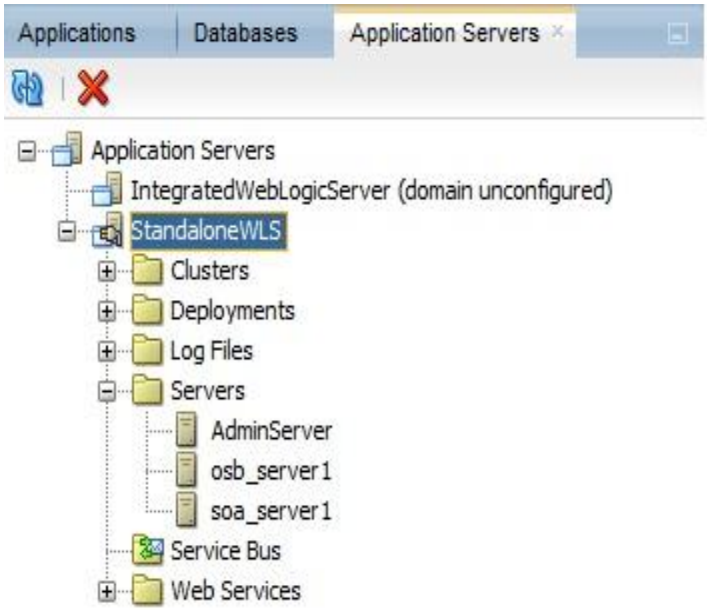
Click **Next** and give the server details and **Weblogic Domain** as shown below.



Click **Next** and do **Test Connection**.



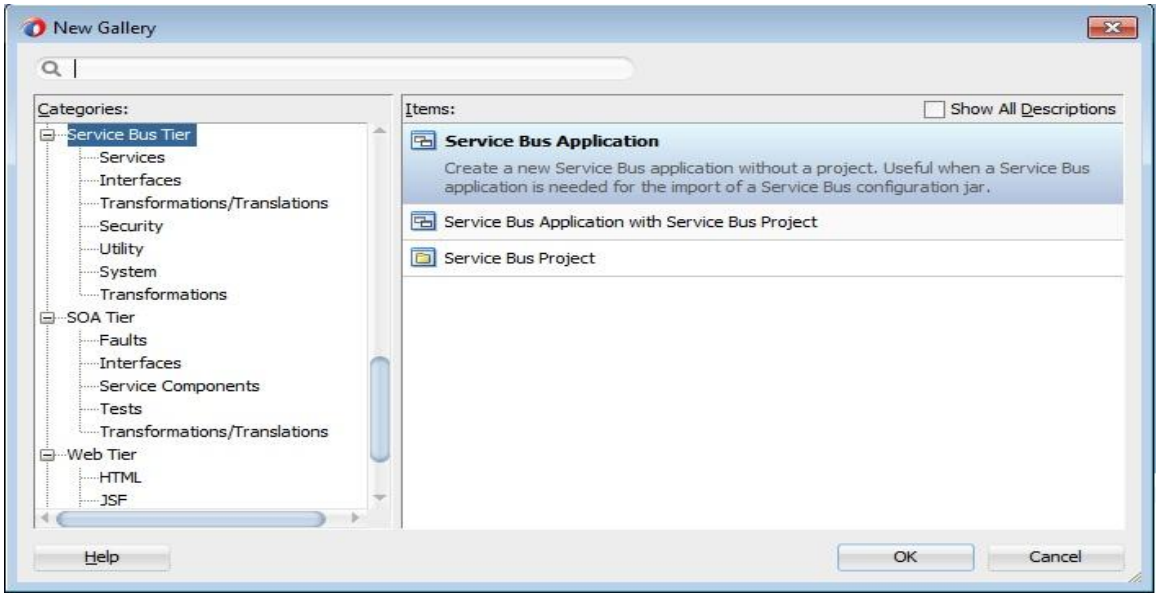
Click **Finish** and verify navigating to **Window -> Application Servers**.

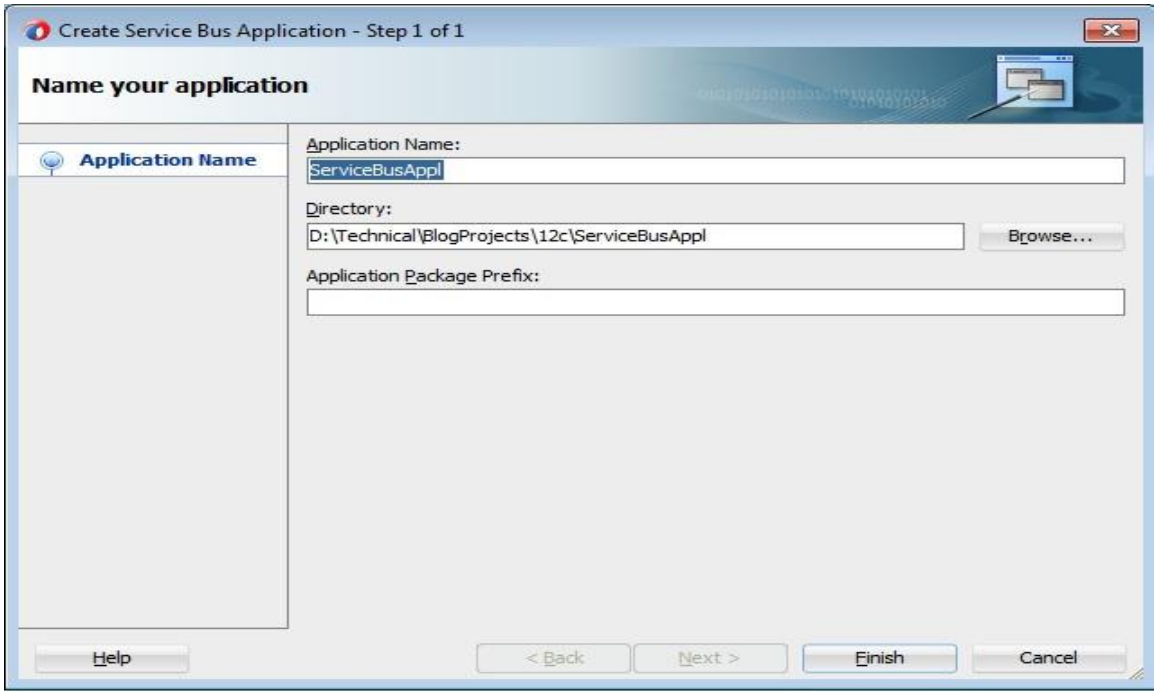


Creating Service Bus Application

In Service Bus development, you make use of different resources like **Proxy Service, Business Service, WSDL, XSD, XQuery, XSLT** etc. Also there are other resources like **SMTP Server, JNDI Provider** called **Global resources**. **Service Bus Project** acts as a container for different resources and **Service Bus Application** is a high-level container for all these Service Bus projects and Global resources.

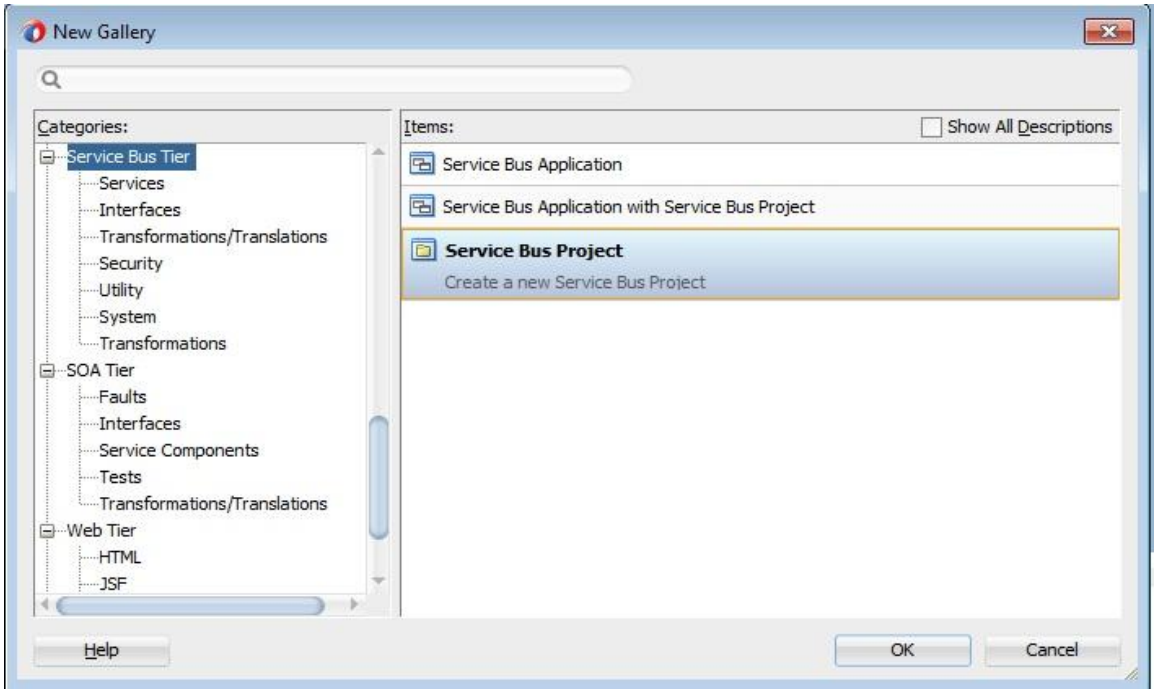
You usually start development activity with creation of new Service Bus Application. So create one by navigating to **File -> New -> From Gallery -> Service Bus Tier.** Select **Service Bus Application** and give name as **ServiceBusAppl** and click **Finish.**

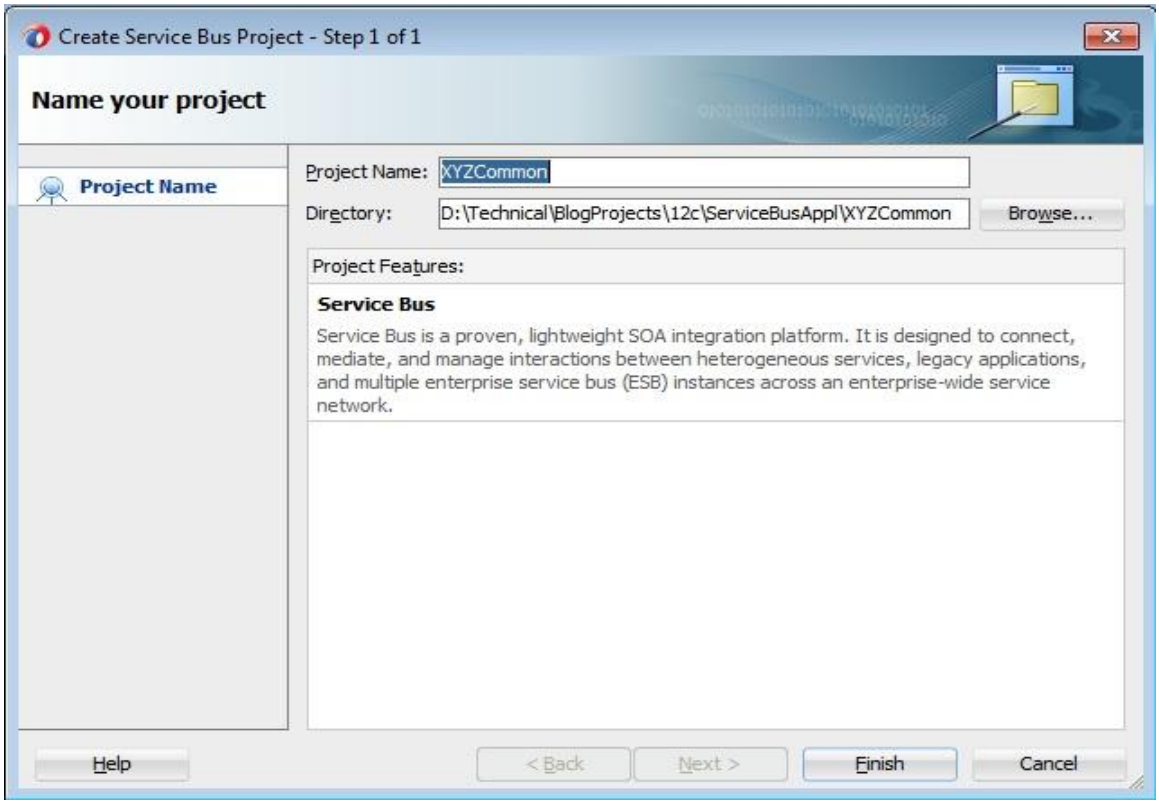




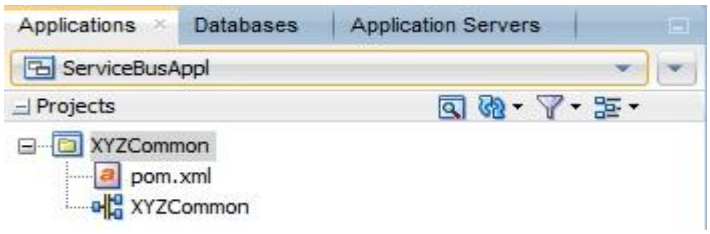
Creating Service Bus Projects

Now you proceed with creation of **Service Bus Projects** considering reusability and modularity. Create new Service Bus Project by navigating to **File -> New -> From Gallery -> Service Bus Tier.** Select **Service Bus Project** and give the name as **XYZCommon** and click **Finish.**

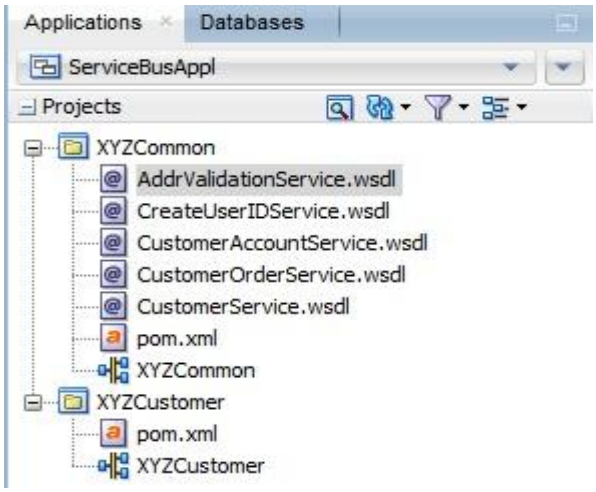




Now your **Project Explorer** should look like below showing **Service Bus Application** and **Service Bus Project** containing 2 files (**pom.xml and XYZCommon**). You will see a file with same name as your project called project overview file which opens up **Service Bus Overview editor** when opened.

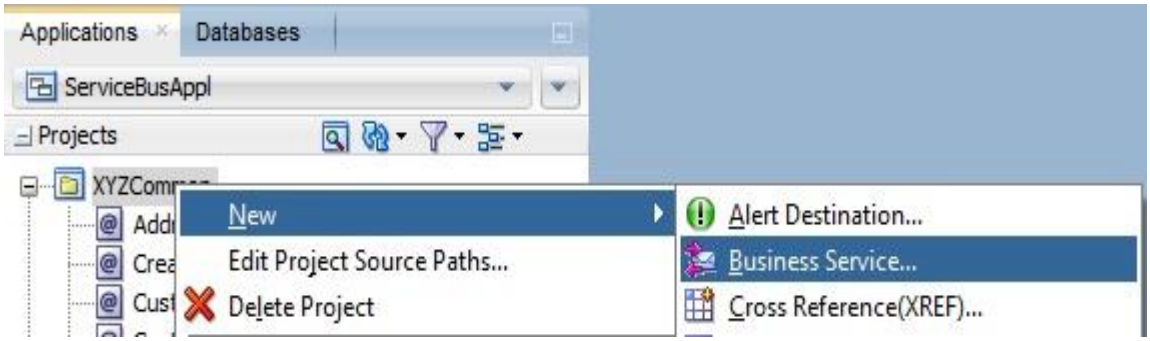


Copy all given WSDLs in **XYZCommon** Service Bus project. You can do this by copying WSDLs into project directory directly in file system. Click **Refresh** in **Project Explorer** so that these resources will be shown. Create another Service Bus Project **XYZCustomer** to be used later in the tutorial. Now your **Project Explorer** should look like below.

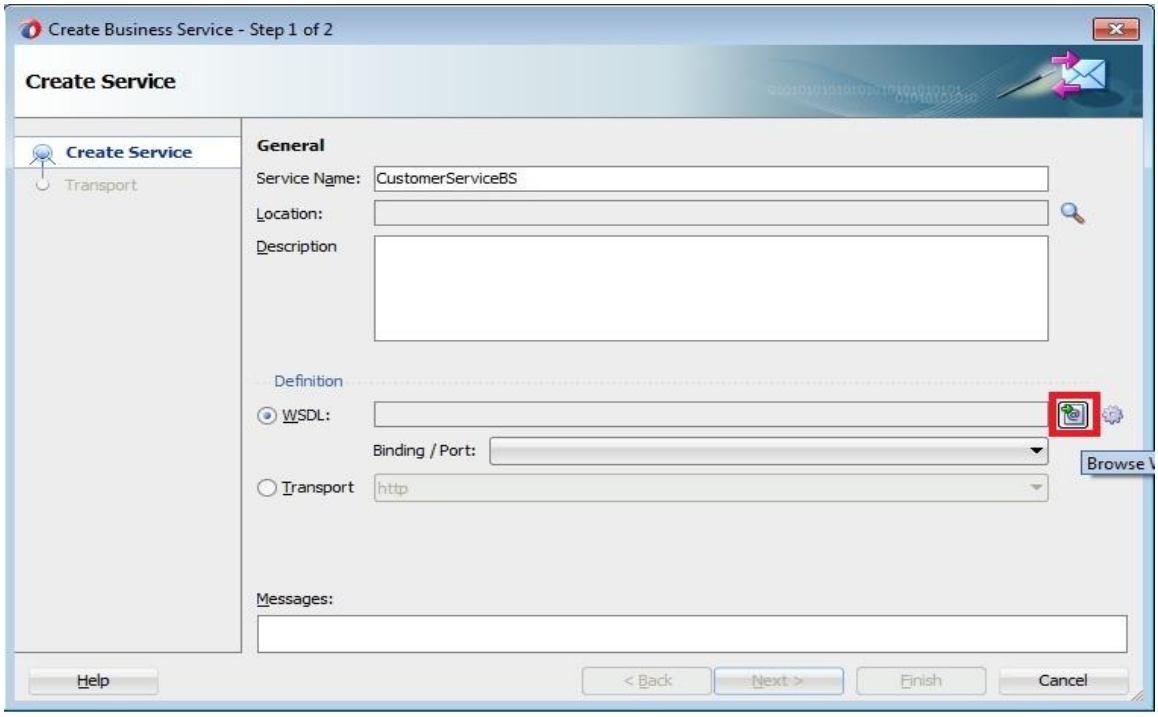


Creating Business Services

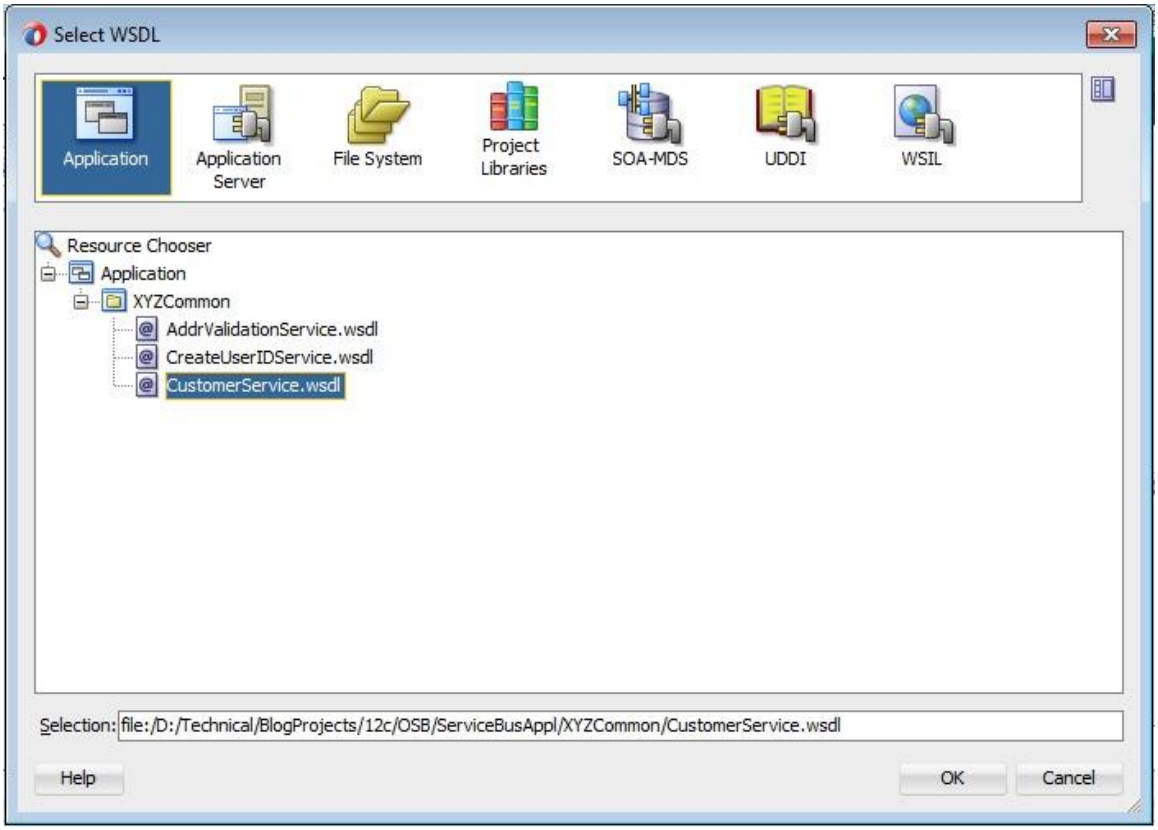
In Service Bus, Business Services provide abstraction layer and take care of communication with Service Providers. So you have to create 3 business services using the given 3 WSDLs. To create business service, right click on **XYZCommon** project and select **New -> Business Service**.



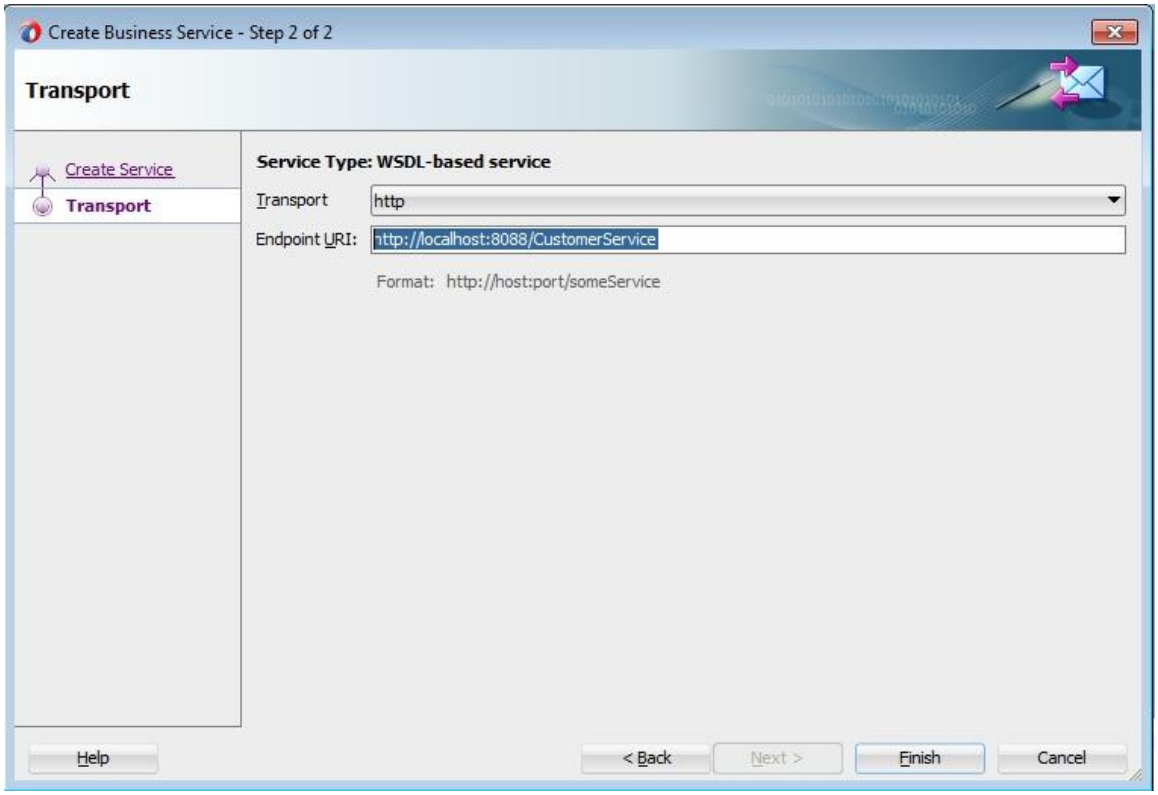
Give name as **CustomerServiceBS.**



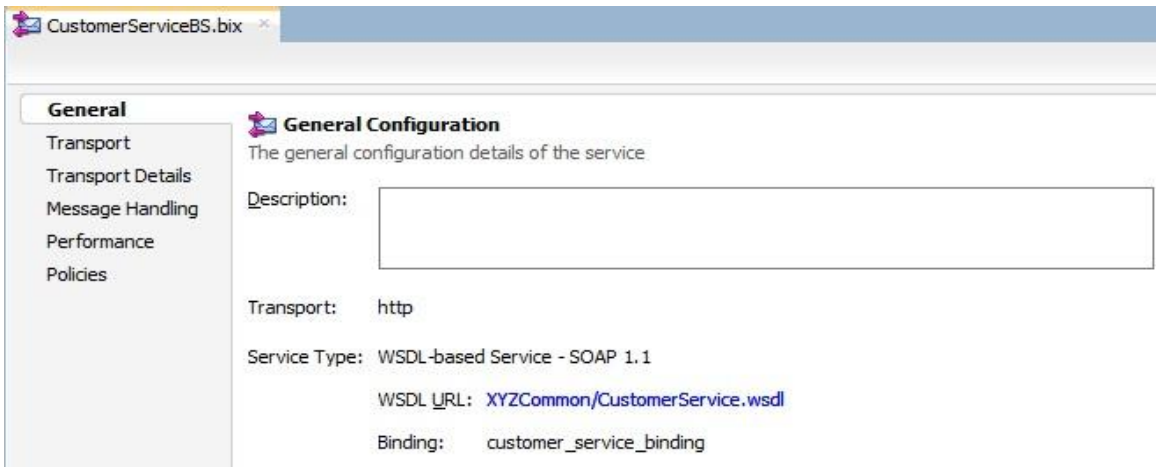
Click **Browse WSDLs** icon (shown above) and choose **CustomerService** WSDL as shown below.



Go back to **Create Business Service** window by clicking **OK**. Click **Next** and enter **Endpoint URI** of SOAP UI Mock Service or any other service implementation.



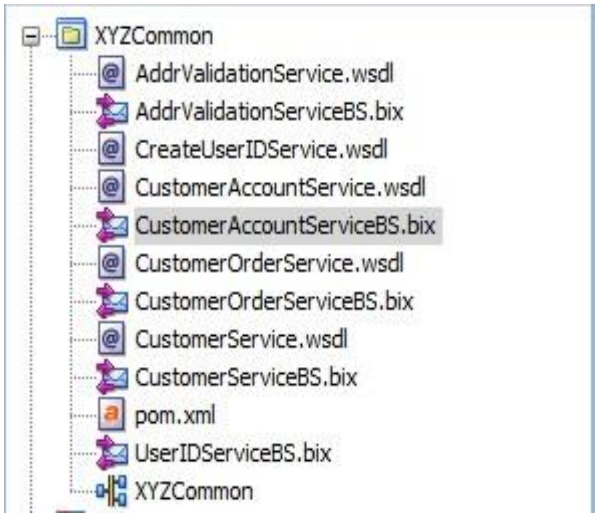
Click **Finish** to bring up window showing new business service **CustomerServiceBS.bix**.



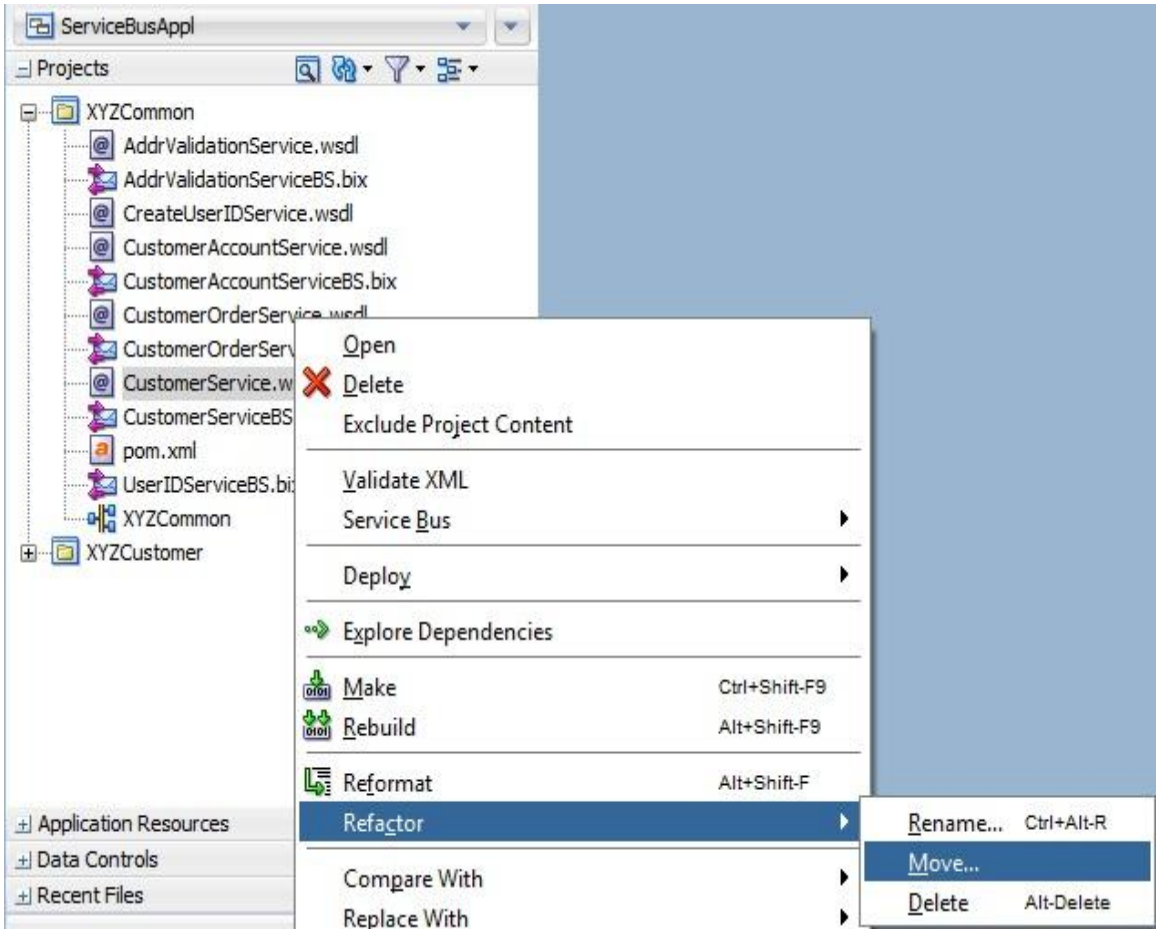
Retain default values for all parameters in all other tabs **Transport Details, Message Handling, Performance and Policies** of business service.

In similar fashion, create all other business services **AddrValidationServiceBS, UserIDServiceBS, CustomerOrderServiceBS** and **CustomerAccountServiceBS** within same service bus project using

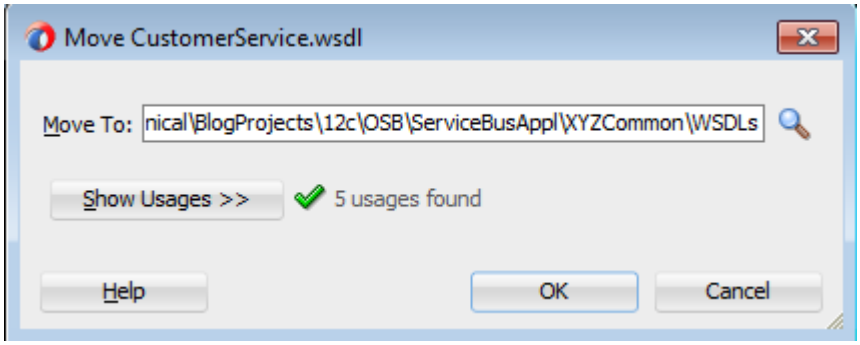
WSDLs **AddrValidationService.wsdl**, **CreateUserIDService.wsdl, CustomerOrderService.wsdl** and **CustomerAccountService.wsdl** respectively.



You can better organize WSDLs and business services in their respective folders in Service Bus project. Since you already created these resources, you can **Refactor** them into desired folders. Right click **CustomerService.wsdl** and select **Refactor -> Move** as shown below.

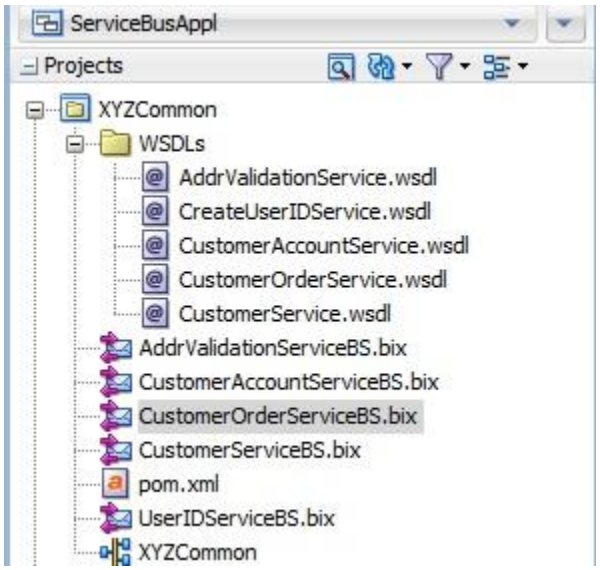


Give folder as WSDLs as shown below.

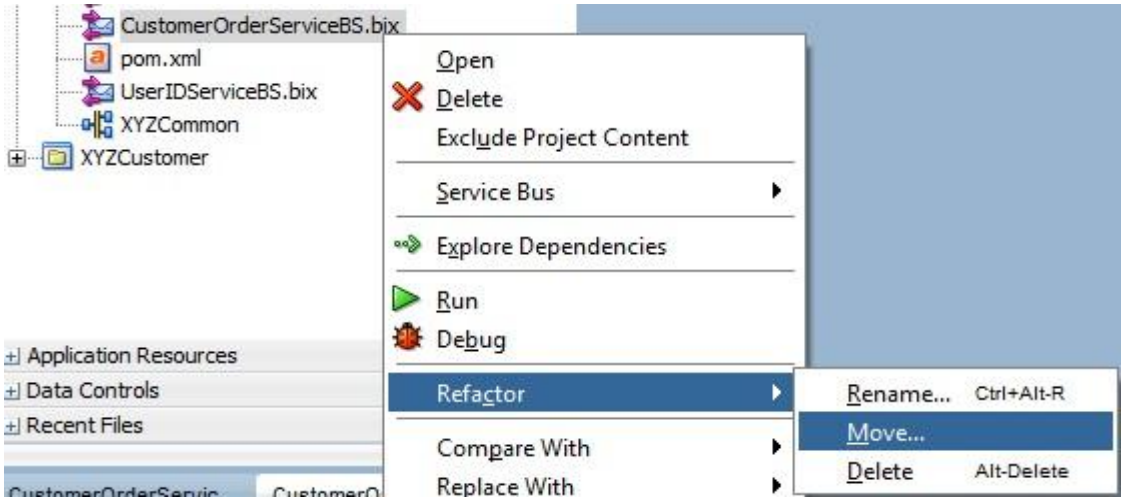


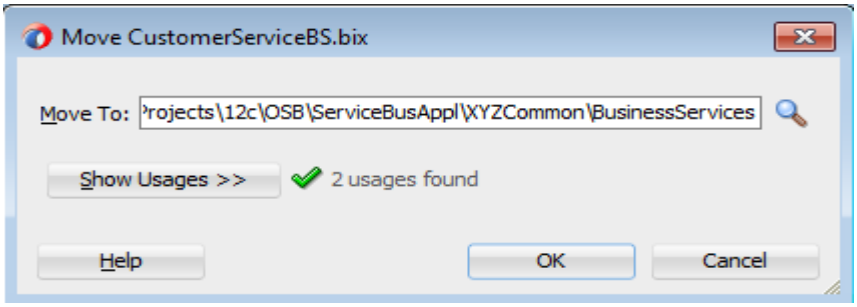
Click **OK** and verify that business service is updated with correct path of WSDL.

Similarly **Refactor** all other WSDLs. Now your **Project Explorer** should look like below.

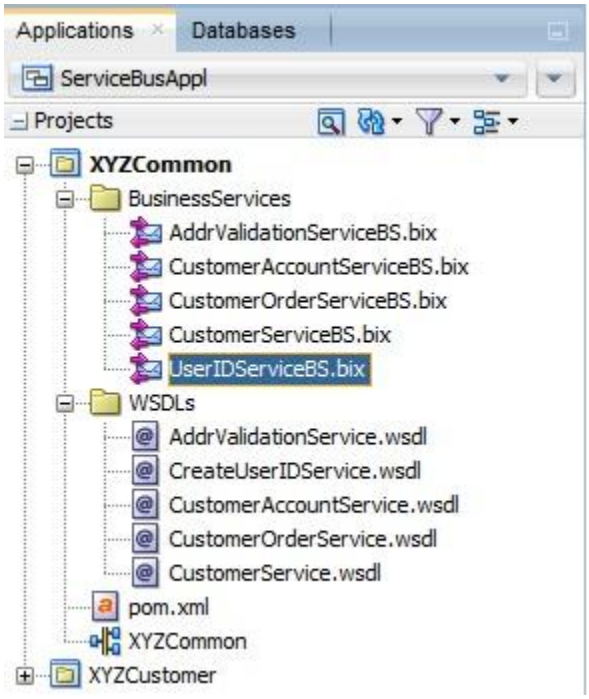


Also **Refactor** all your business services using steps mentioned below.





Now your **Project Explorer** should look like below.



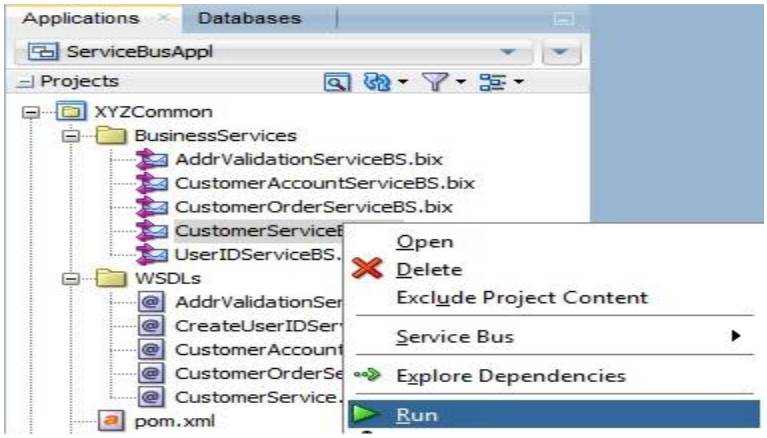
Section 1

Deploying and Testing

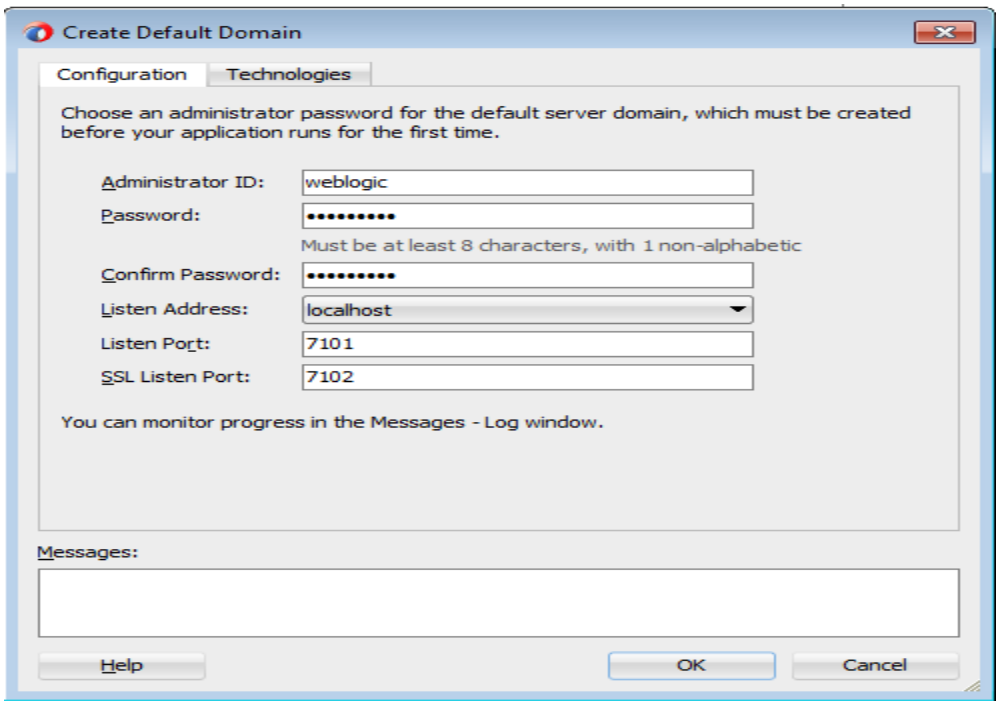
It's always recommended to run your Business/Proxy services to verify that you are able to send request and receive response as expected. Typically, if the service provider is external to your organization, then you have to configure business service to use HTTP proxy server. For this tutorial, you don't need any proxy server configuration.

*Note: HTTP Proxy Server is a global resource to be created in Service Bus Application and can be attached to business service in **Advanced Settings** of **Transport Details** tab.*

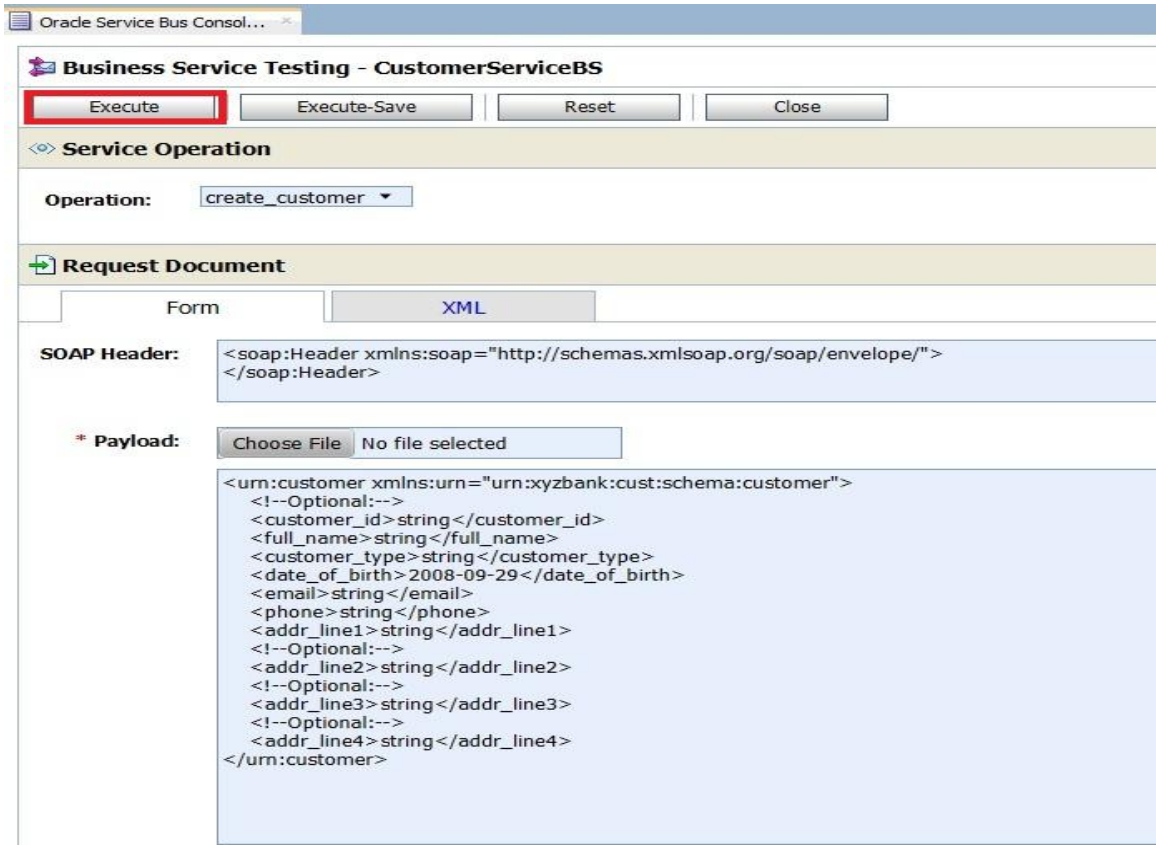
In 12c, you can use either **Integrated** or **Standalone** WLS to test business/proxy services. Right click your Business/Proxy service and select **Run.**



If **Default Domain** does not exist, it will prompt your for credentials to create an Integrated WLS. Enter credentials as shown below and click **OK** to start server.



After server starts, verify that test console is opened as shown below. The test console will show you all available operations (based on WSDL), Request Document section (to enter payload), Transport section (to enter transport headers) and Attachments section.



Click **Execute** and observe that you are able to send request and response in test console as shown below. You can use **Back** button to repeat your test scenarios. In similar manner, you can verify all other business services.

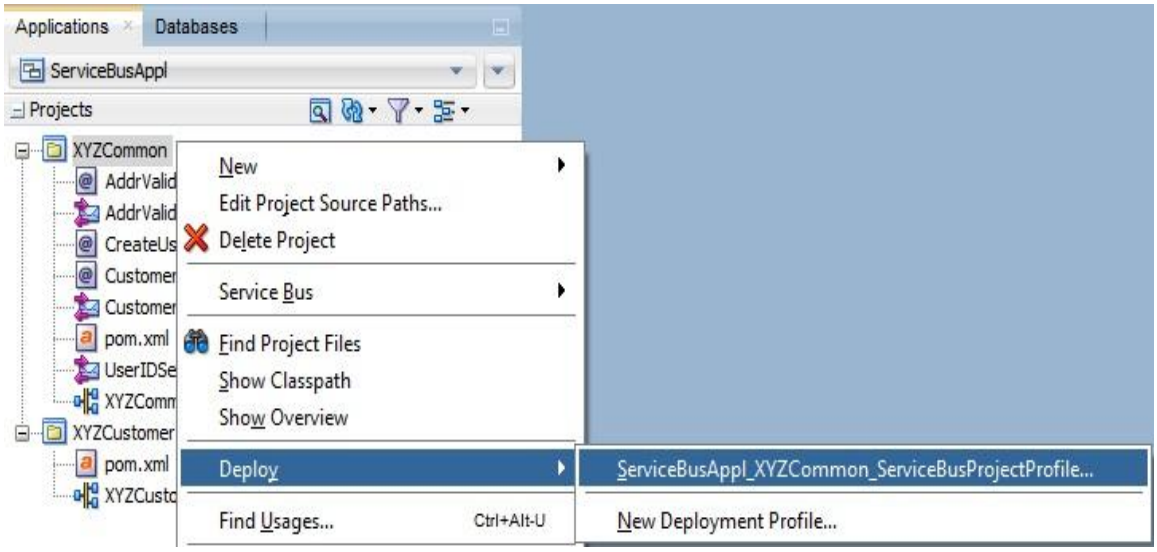
Request Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <urn:customer xmlns:urn="urn:xyzbank:cust:schema:customer">
            <!--Optional:-->
            <customer_id>string</customer_id>
            <full_name>string</full_name>
            <customer_type>string</customer_type>
            <date_of_birth>2008-09-29</date_of_birth>
            <email>string</email>
            <phone>string</phone>
            <addr_line1>string</addr_line1>
            <!--Optional:-->
            <addr_line2>string</addr_line2>
            <!--Optional:-->
            <addr_line3>string</addr_line3>
            <!--Optional:-->
            <addr_line4>string</addr_line4>
        </urn:customer>
    </soapenv:Body>
</soapenv:Envelope>
```

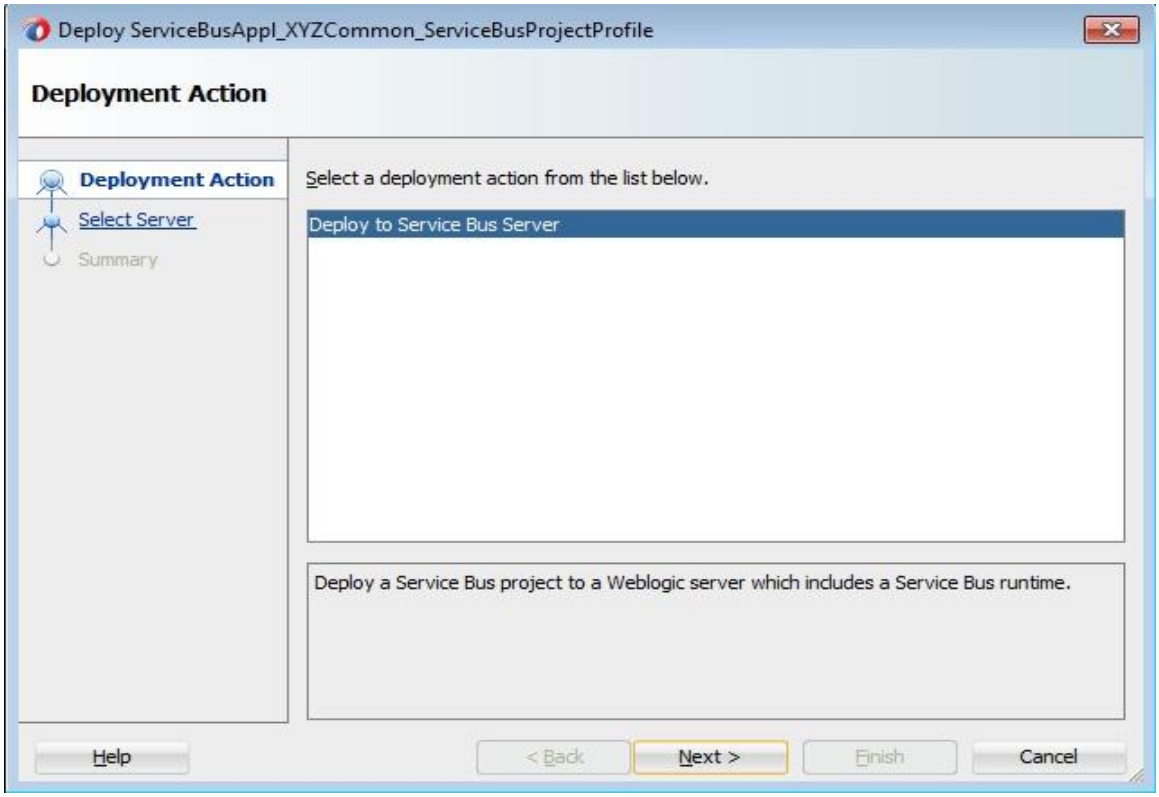
Response Document

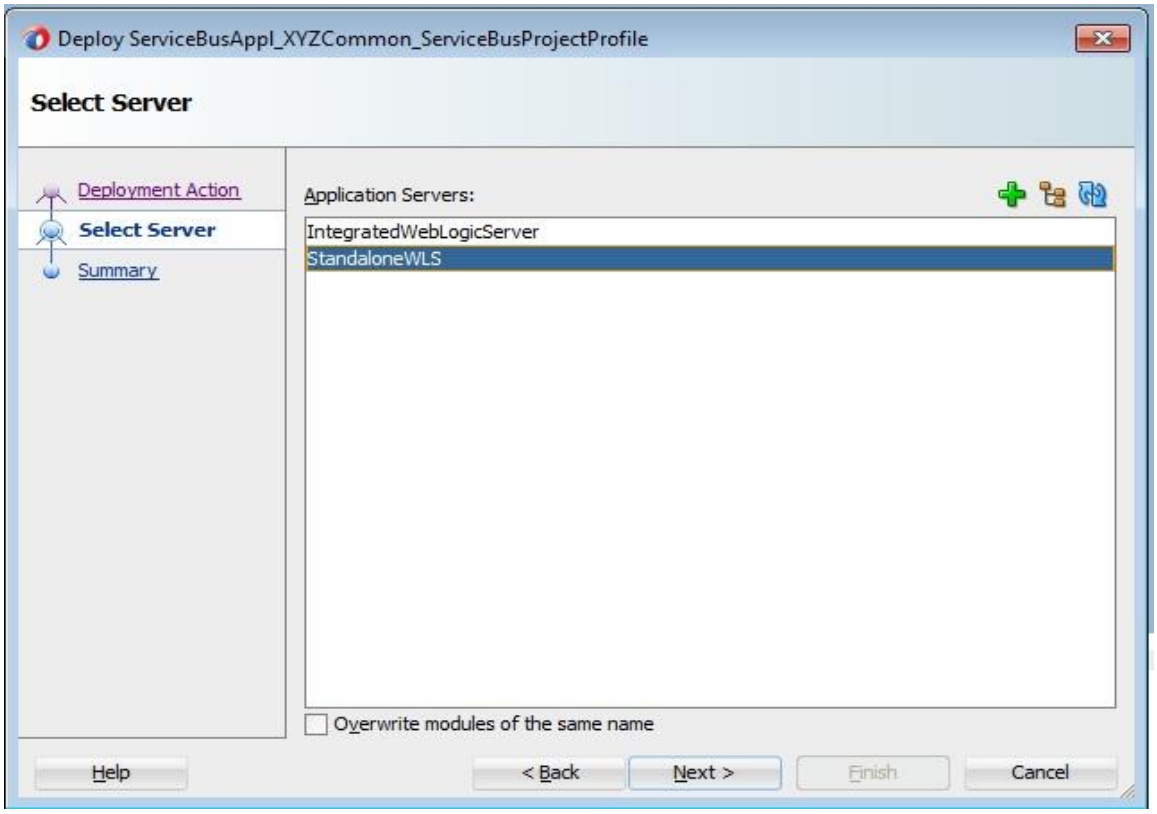
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:xyzbank:cust:schema:customer">
    <soapenv:Header/>
    <soapenv:Body>
        <urn:status_msg>
            <customer_id>1234</customer_id>
            <status>S</status>
        </urn:status_msg>
    </soapenv:Body>
</soapenv:Envelope>
```

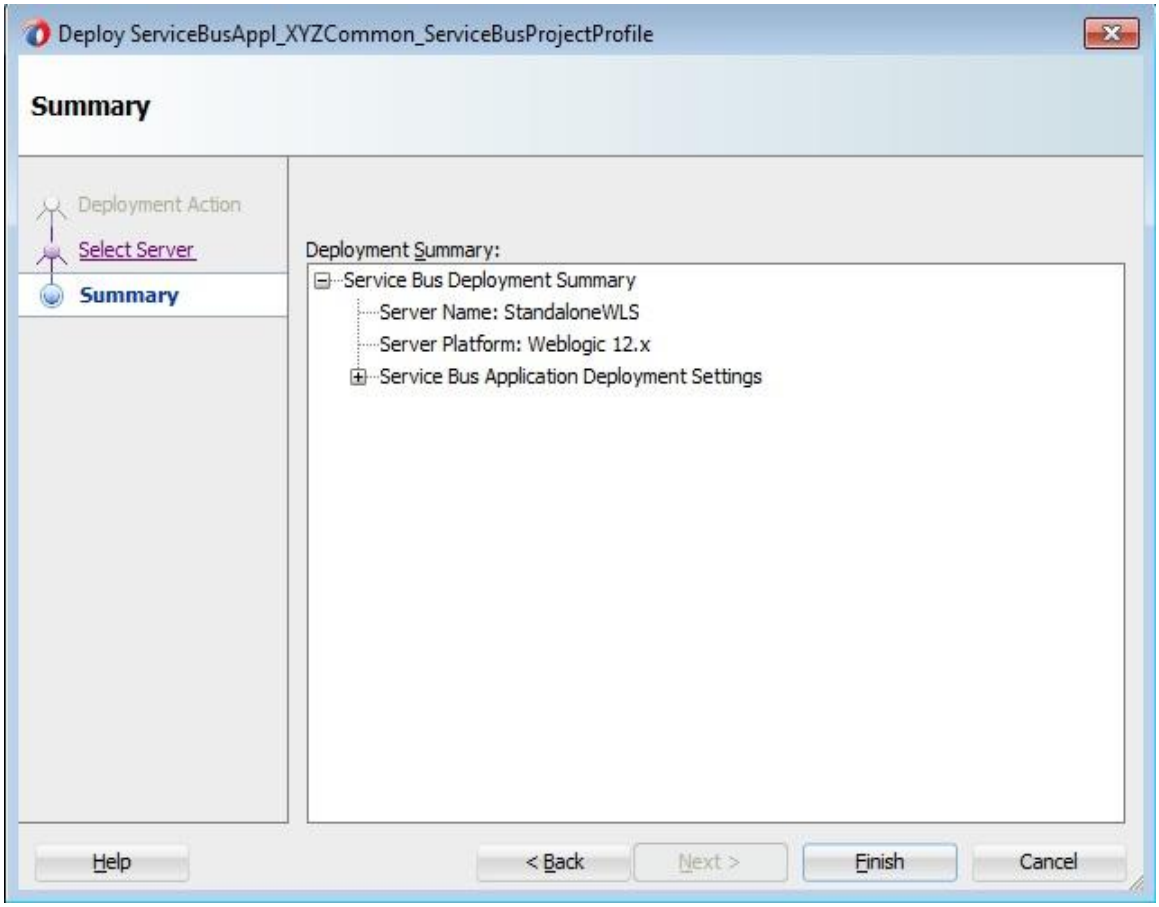
You can also deploy your Service Bus project to standalone WLS from JDeveloper and test from **sbconsole**. Right click **XYZCommon** project and choose deploy as shown below.



Finish the deployment as shown below.

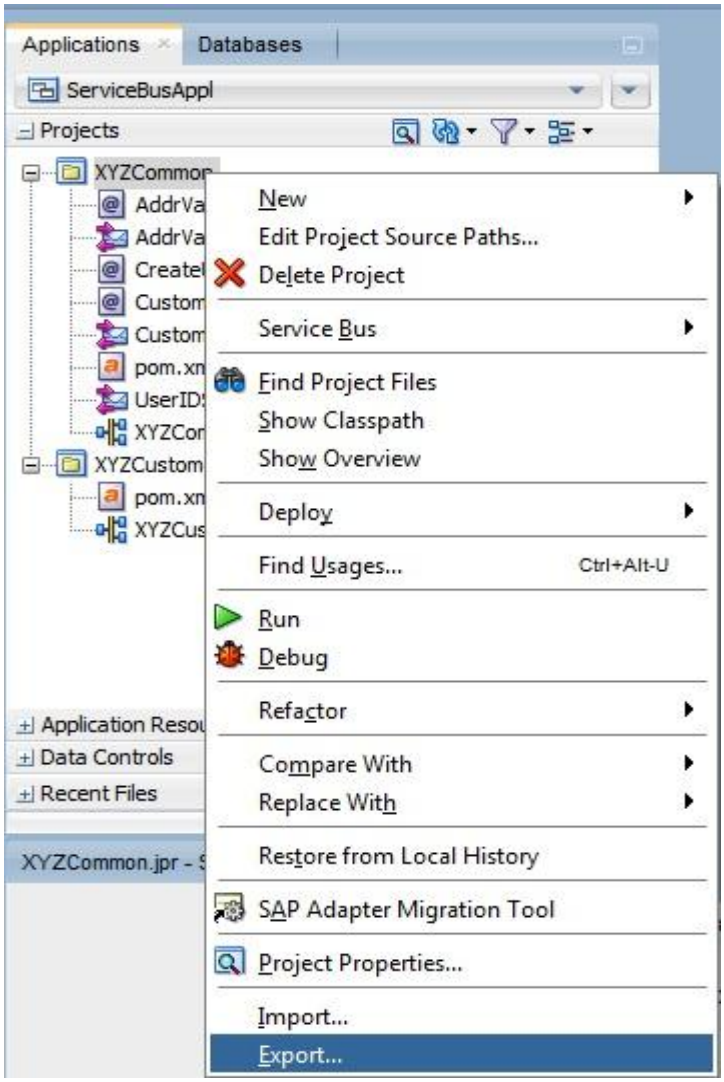




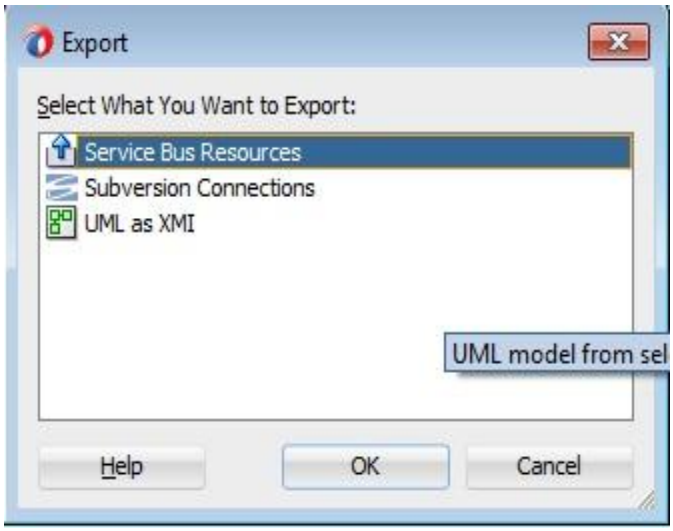


Once deployment is done, you can navigate to **sbconsole** and test your Business/Proxy services. Clicking arrow icon (highlighted below) will bring up the same test console shown earlier.

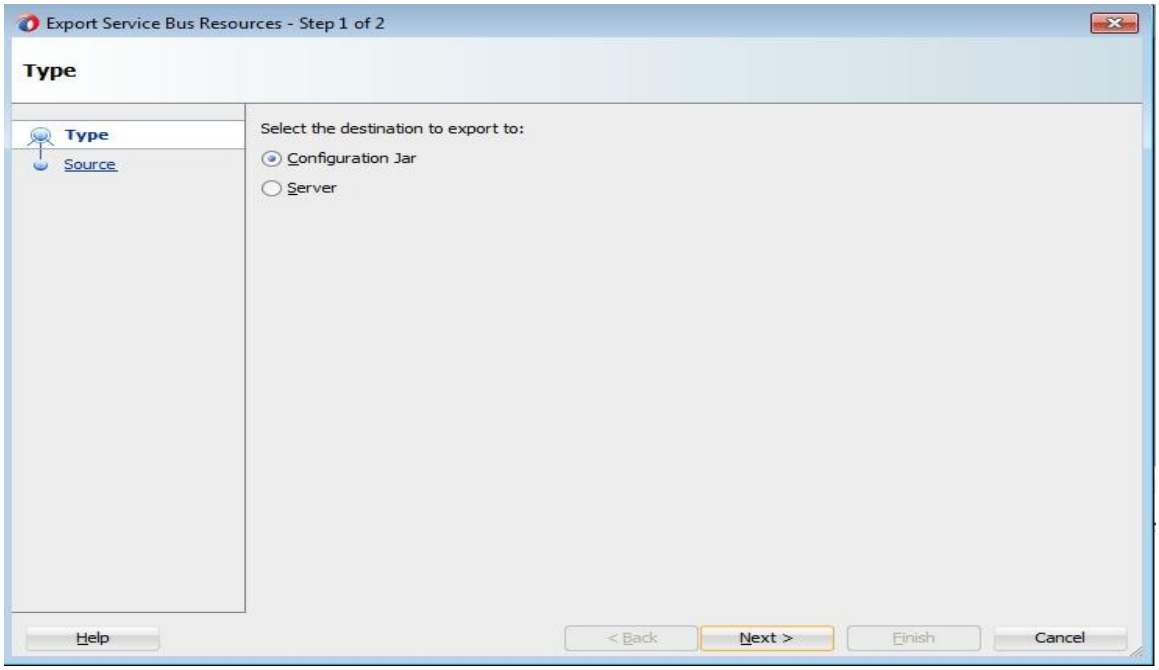
Another option is exporting your projects to **Configuration Jar** and deploy to either **Integrated** or **Standalone** WLS. Right click **XYZCommon** project and select **Export** as shown below.



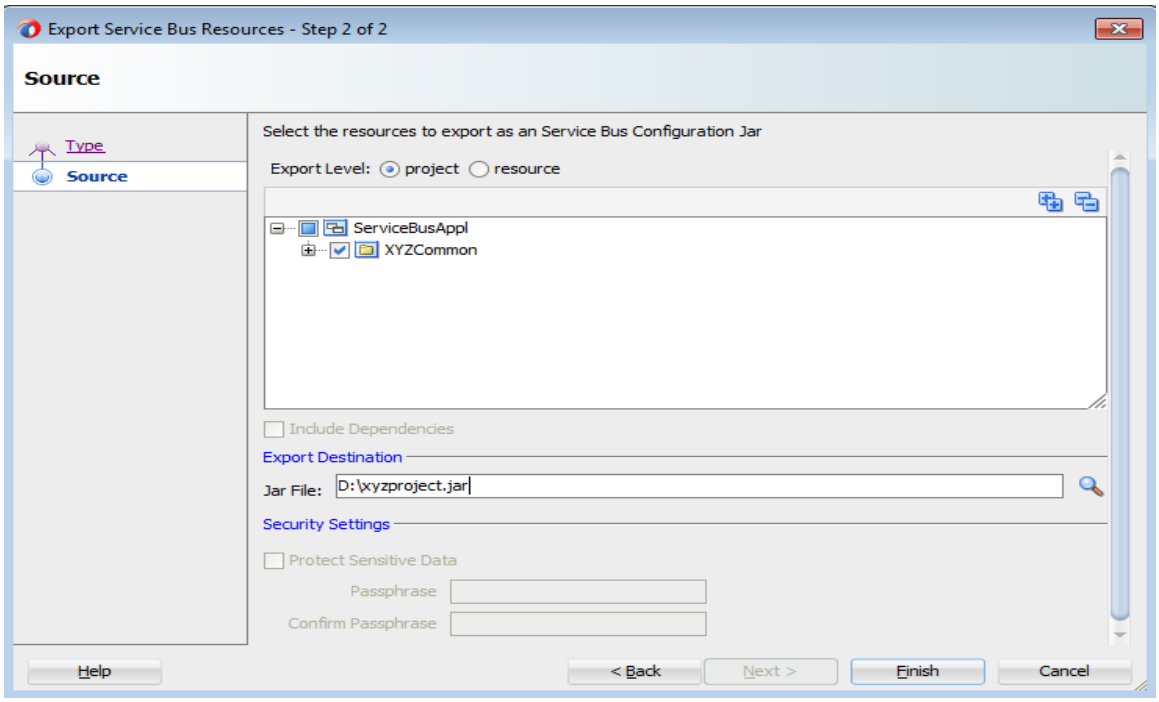
Select **Service Bus Resources** option as shown below.



Select **Configuration Jar** as destination.

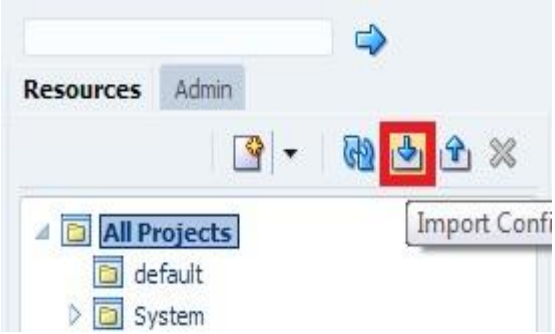


Click **Next** and give Jar file name including file system path as shown below and click **Finish**.



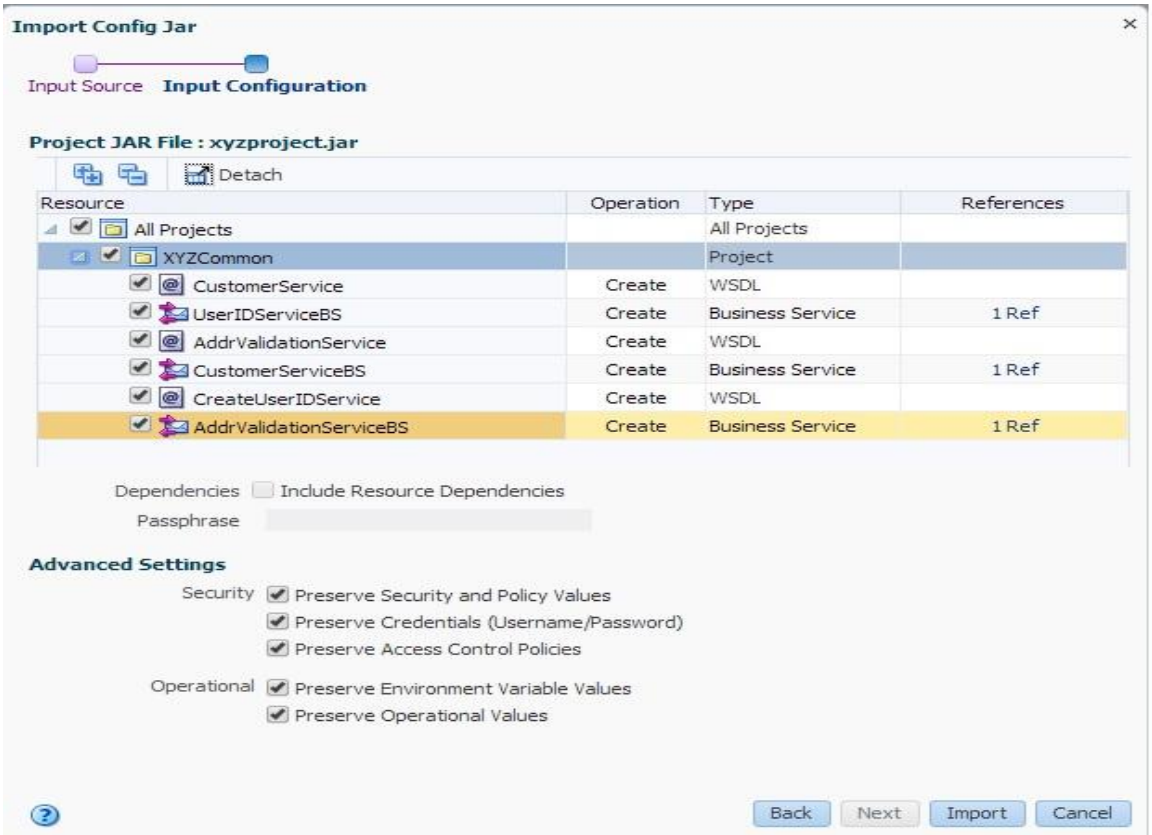
Open **sbconsle** and create new session by clicking **Create** to import configuration jar.

Click **Import** icon in **Resources** tab.



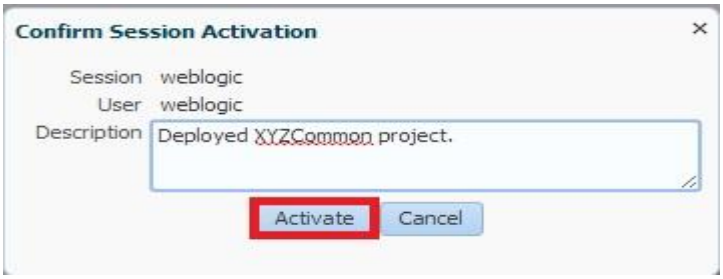
Select configuration jar using **Choose File** button as show below.

Click **Next.** Accept defaults and click **Import.**

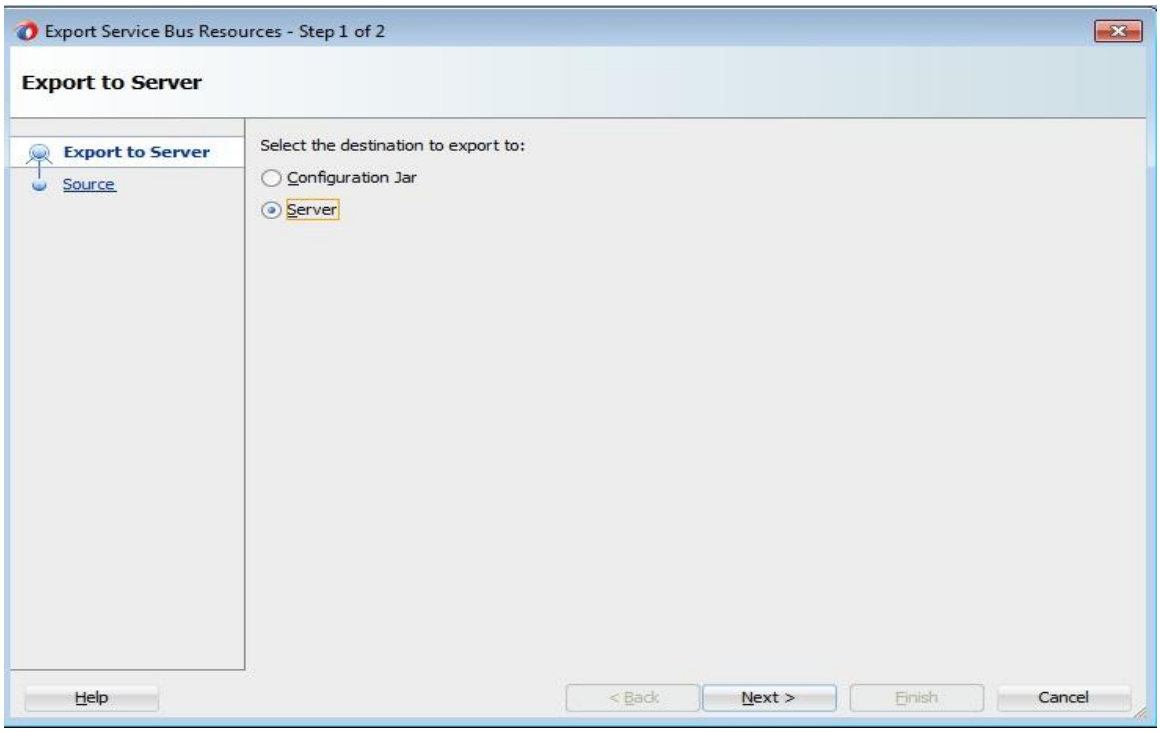


Click **Close** on confirmation screen and verify new project is shown as below in **Resources** tab.

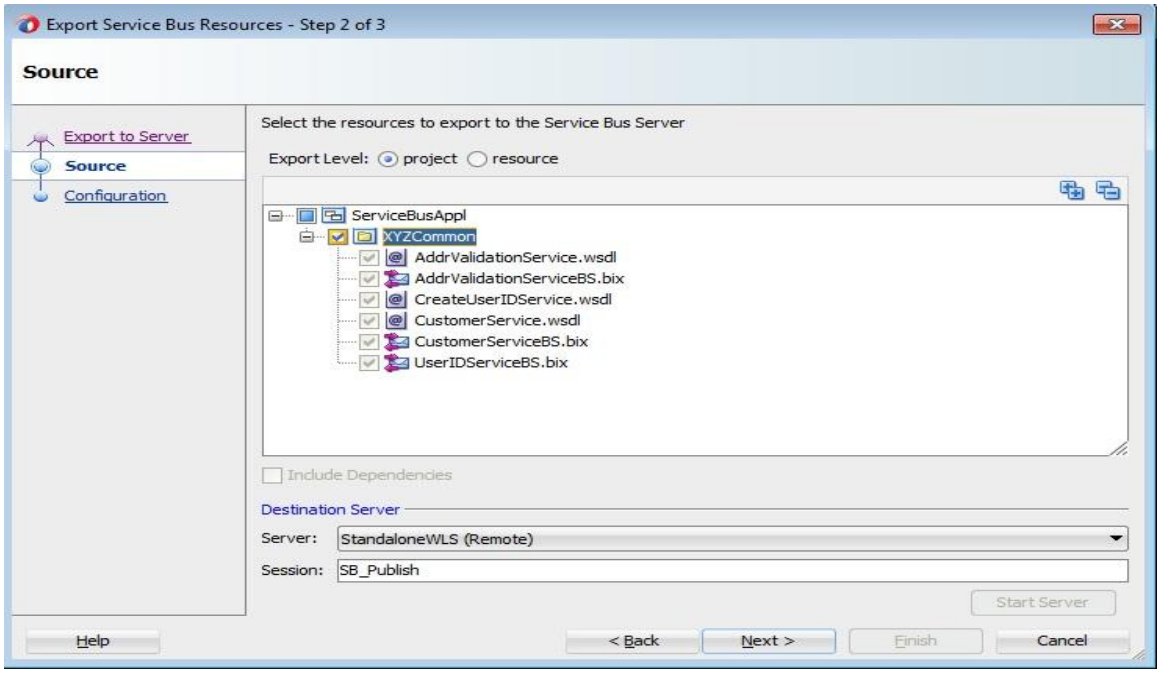
Click **Activate** and confirm the session so that your changes will be effective.



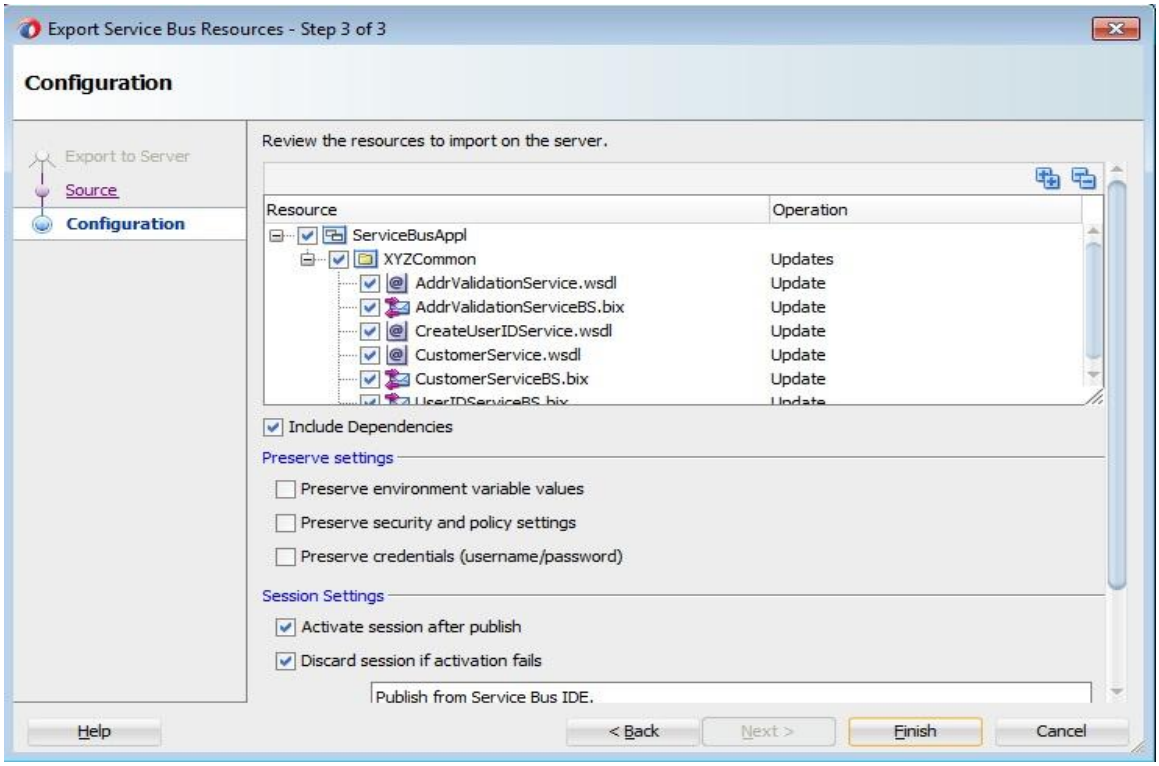
In the same way, to export Service Bus project directly to server choose **Server** as shown below in **Export Service Bus Resources** window shown earlier.



Click **Next** and choose **Destination Server** as shown below.



Click **Next.** Accept defaults and click **Finish.** Once export is successful, you can launch **sbconsole** and test your Business/Proxy services.

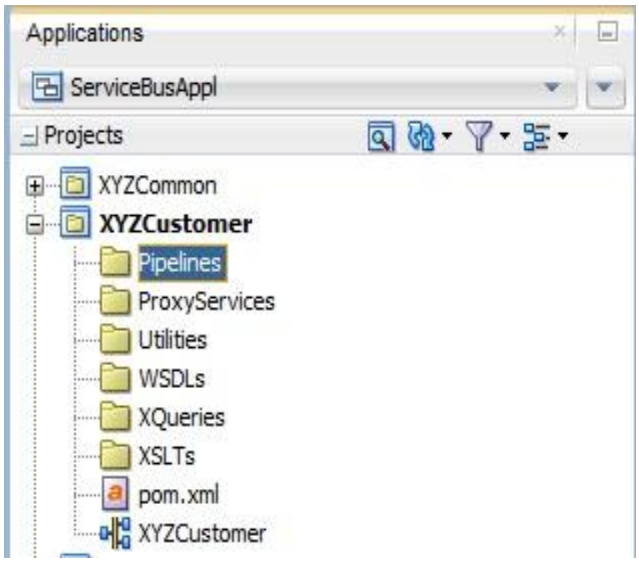


Note: Developer can choose any of above options to deploy and test business/proxy services.

Creating Pipeline Template

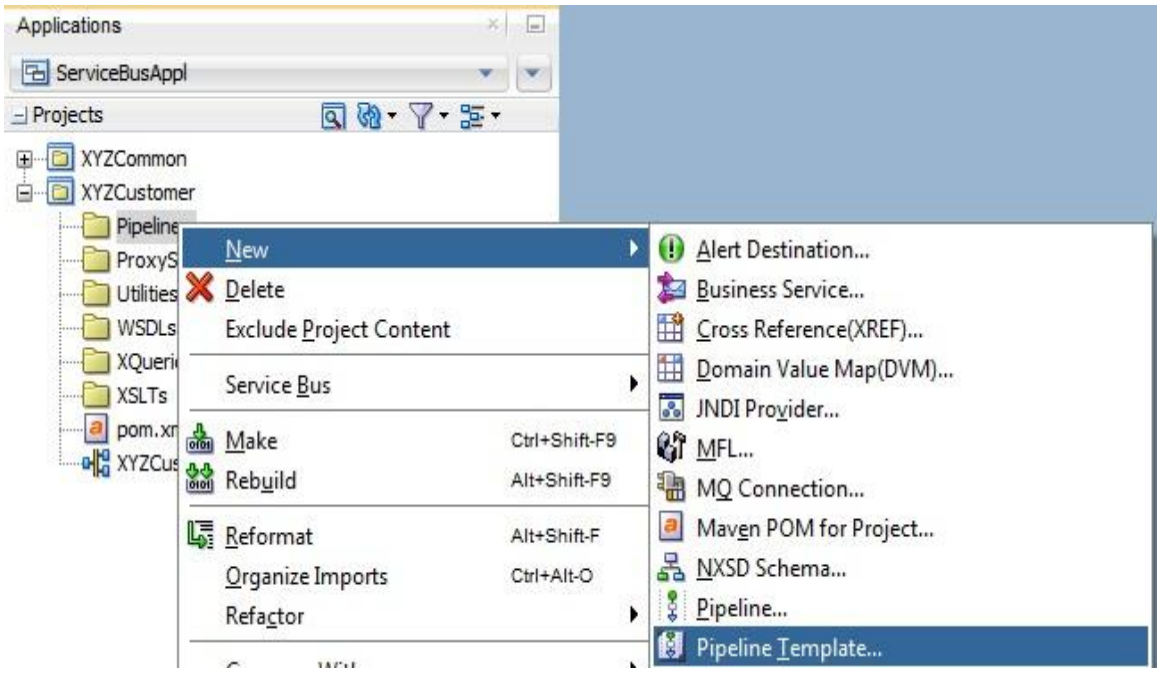
In this section, you will learn how to create pipeline templates considering requirements. A pipeline template defines the general shape or pattern of your message flow.

Use **XYZCustomer** project to create all of your resources unless explicitly mentioned. Before proceeding with Pipeline template creation, create different folders **Pipelines**, **XQueries**, **XSLTs**, **WSDLs** and **ProxyServices** as shown below. You can create new folders by navigating to **File -> New -> From Gallery -> General -> Folder**.

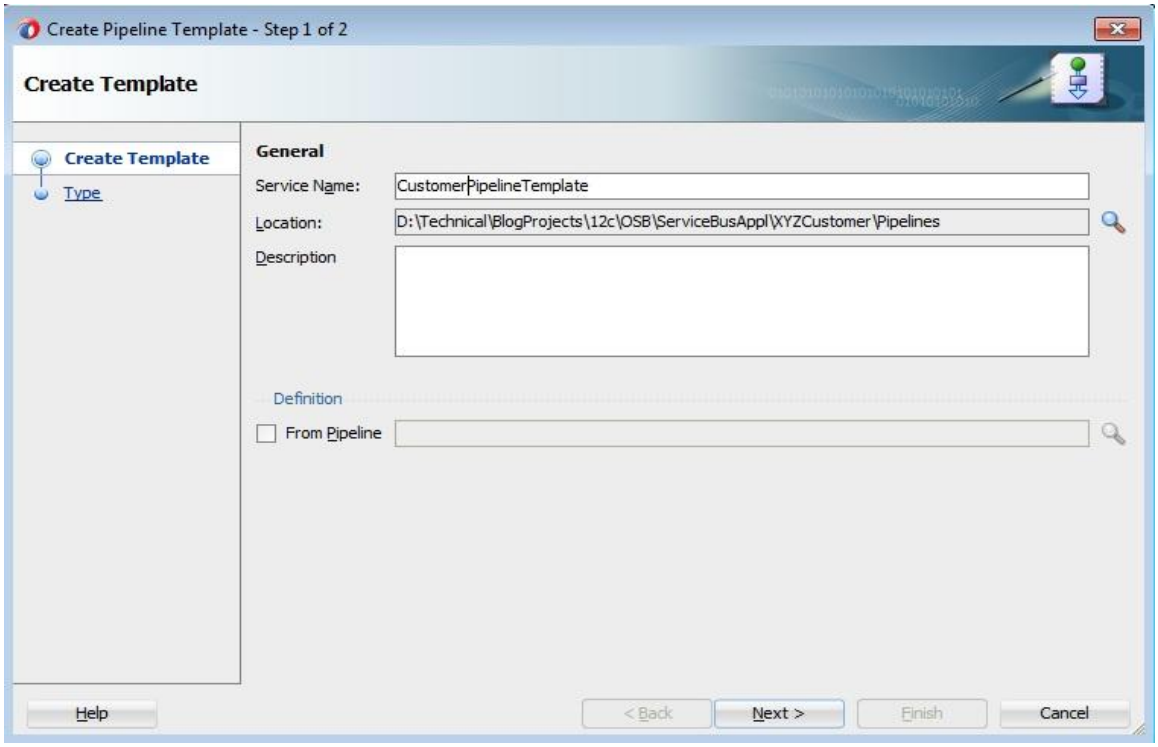


Section 2

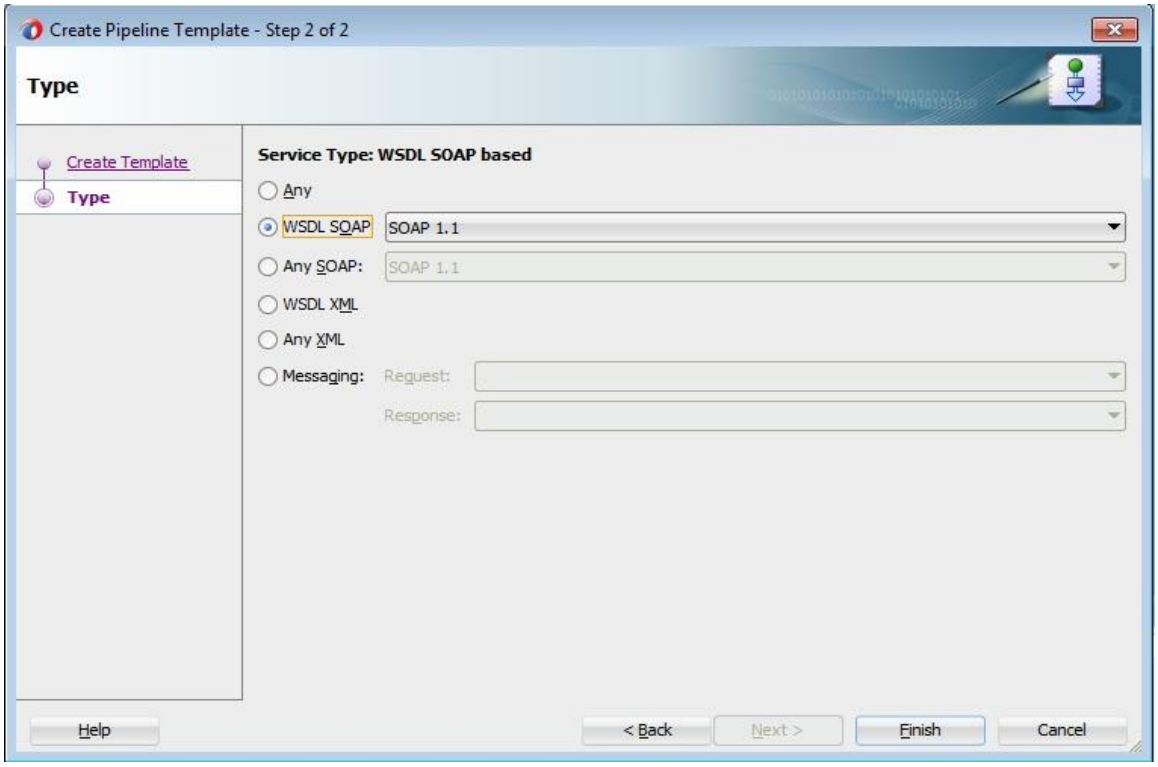
Right click Pipelines folder and select **New -> Pipeline Template**.



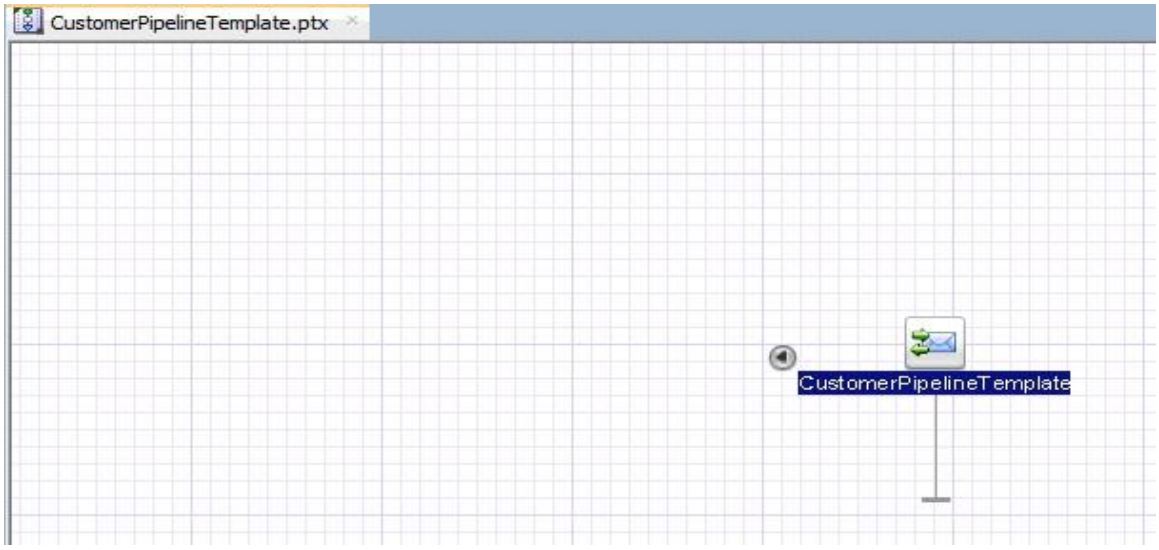
Give name as **CustomerPipelineTemplate.**



Click **Next** and choose **Service Type** as **WSDL SOAP** as shown below. You have to choose this option as you are going to create all of your Proxy Services based on a WSDL.



Click **Finish** and verify that new pipeline template has been created and is opened as a separate tab as shown below.



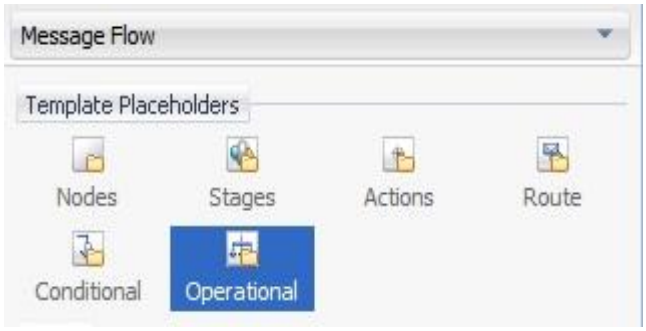
Pipeline templates are the right candidates to keep common logic or enforce pattern of message flow that should be followed across Proxy services. For this tutorial, you may not be creating number of Proxy Services but following are common activities for different operations that your Proxy Service should support. So we will incorporate the following functionality in this Pipeline template.

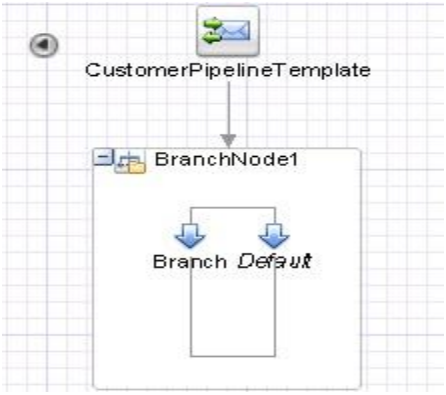
- Validation of Payload against schema
- Routing to **CustomerServiceBS** to perform CRUD operations
- Error Handling

Along with above functionalities, your Pipeline template needs to have nodes like Operational branch, Pipeline Pair, Stages etc... So let us get started with Pipeline template message flow suiting these requirements.

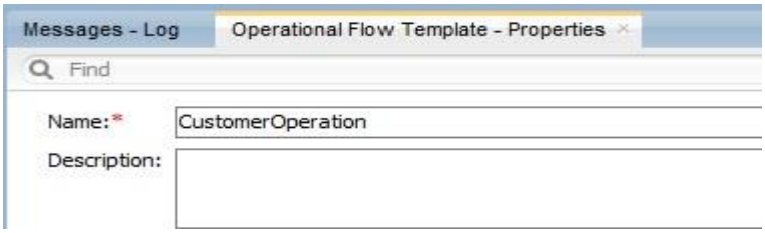
When consumers call proxy service, it has to take different execution paths based on operation being called. The Service Bus provides **Operational Branch** node serving this purpose. So drag **Operational** from **Template Placeholders** onto Pipeline template where yellow circle is shown.

*Note: All Nodes or activities used in message flow for Pipeline or Pipeline template are available in **Components -> Message Flow** under different sections. If you don't see **Components** window, make it visible by selecting **Window -> Components**.*

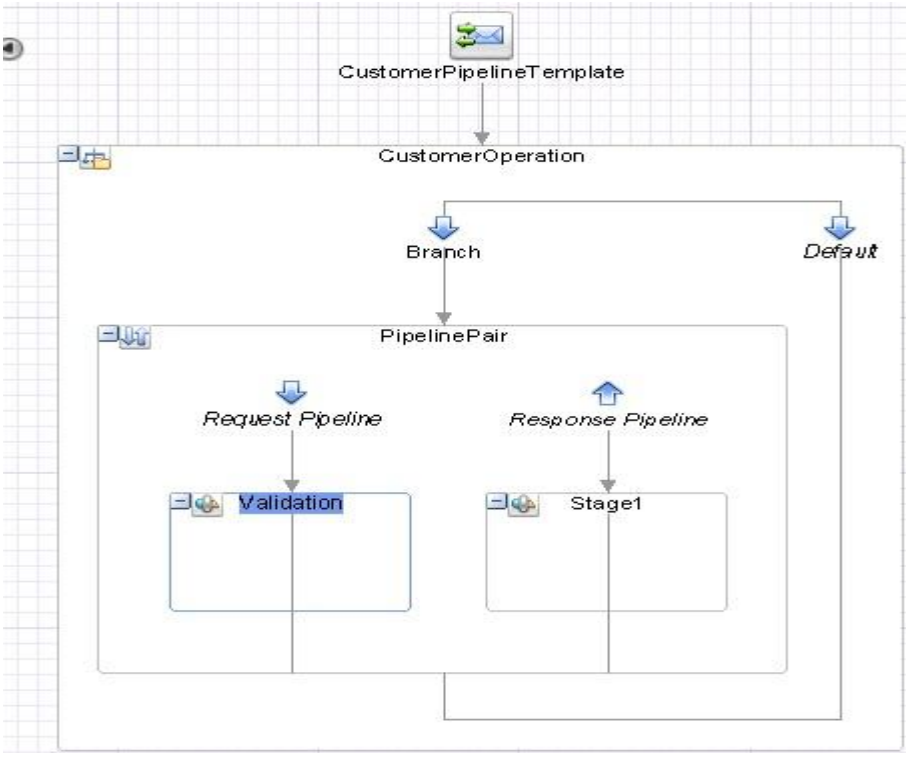




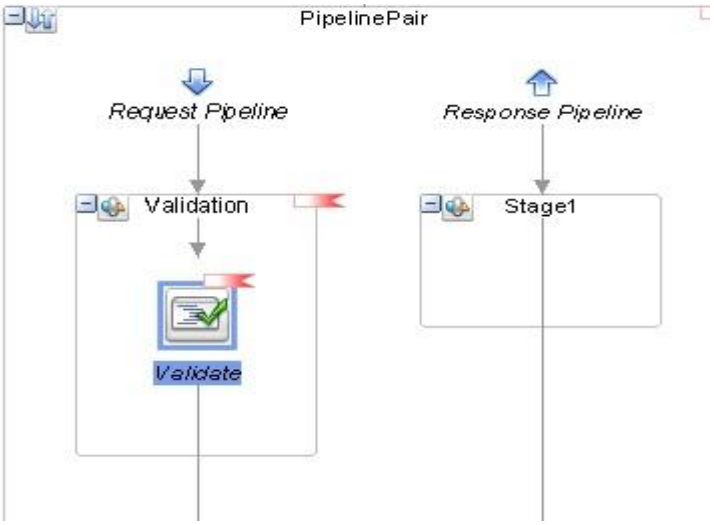
Click **BranchNode1** and set name as **CustomerOperation** in **Properties** tab as shown below. As a best practice, always give meaningful labels for Nodes and other activities (wherever applicable).



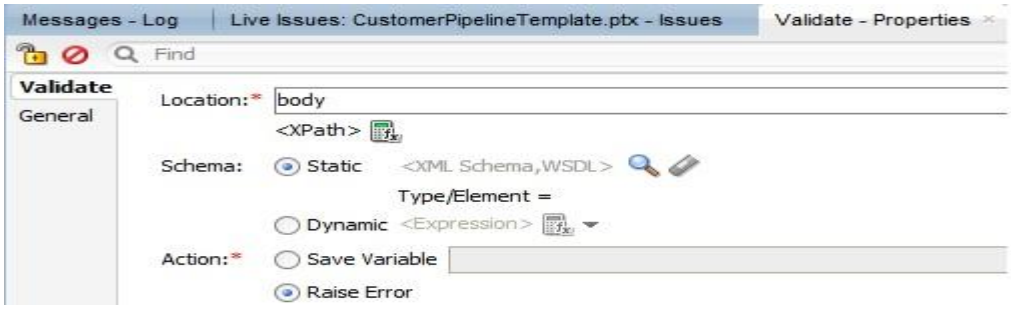
Now drag **Pipeline Pair** from **Nodes** into **Branch** and set name as **Pipeline** in **Properties** tab. Also rename **Stage1** node in **Request Pipeline** to **Validation** in **Properties** tab. Now your pipeline template should look like below.



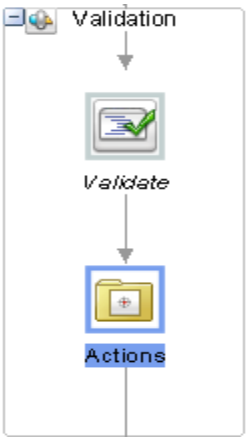
Service Bus provides **Validate** activity to do payload validation against schema. So drag **Validate** activity into **Validation** stage from **Message Processing**.



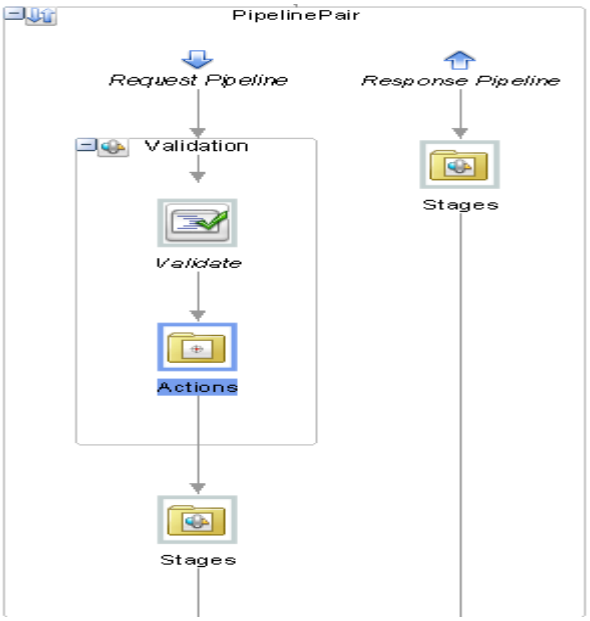
In **Properties** tab, set **Location** property value as **body** and other properties as shown below. Here we don't select any schema or WSDL for validate activity as it's just template and we will not have any payload available.



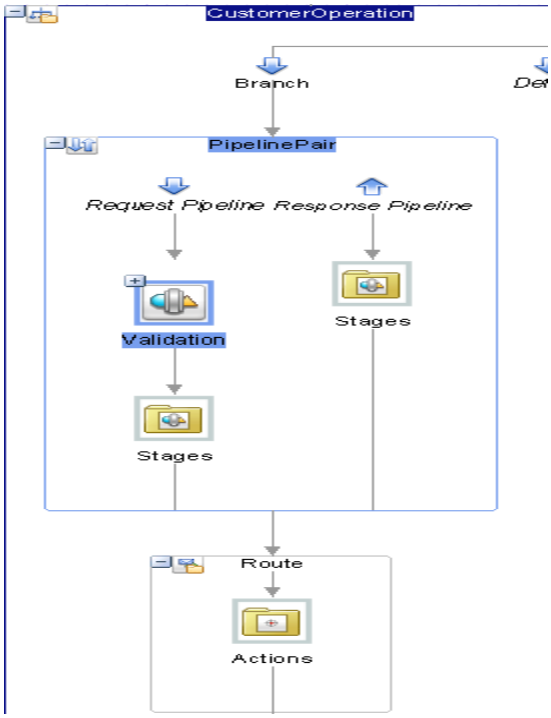
There may be additional validations to be performed while designing message flow for concrete pipelines and requires a place holder. So drag **Actions** into **Validation** stage as shown below from **Template Placeholders**.



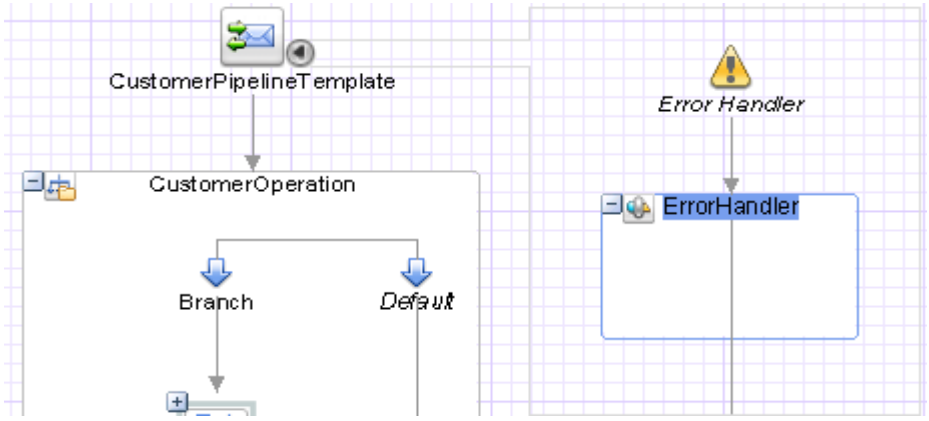
Delete **Stage1** in **Response Pipeline** by selecting **Delete** option on right click. Also you may need to create several Stage nodes in your concrete Pipeline's message flow. So create place holder for the same in template. Drag **Stages** into pipeline after **Validation** stage and in **Response Pipeline** from **Template Placeholders**. Now your Pipeline should look like below.



Routing node always depicts end of message flow and you can't place any other activities after this node. So drag and place **Route** after **PipelinePair** from **Template Placeholders** and set name as **Route** in **Properties** tab. Observe **Actions** placeholder which allows user to place any type of activity while designing message flow for your concrete pipelines.



Error handling is another aspect of message flow where you should follow similar approach for different Proxy Services that you create. So you can always create placeholder in your template. Drag **Error Handler** onto **CustomerPipelineTemplate** from **Nodes** and set name as **ErrorHandler** for **Stage1** as shown below in **Properties** tab.

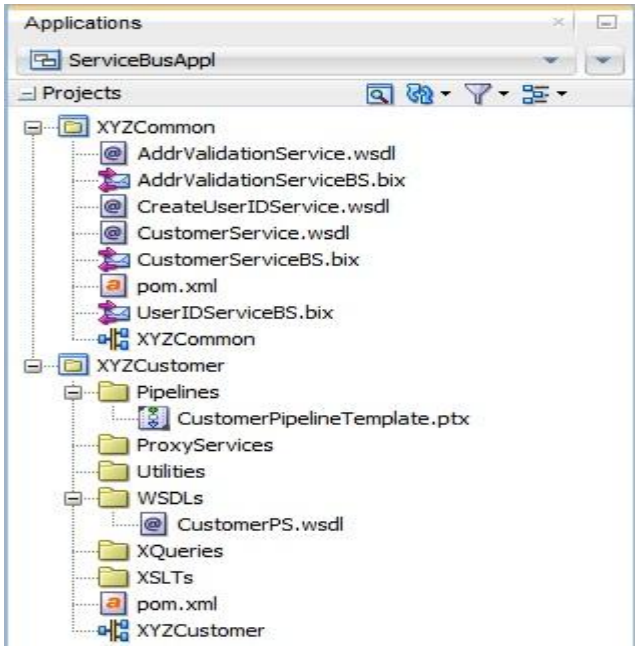


You will finish this **ErrorHandler** stage later in **Error Handling** section. With this, you are done with pipeline template and can proceed with creation of other pipelines using this template.

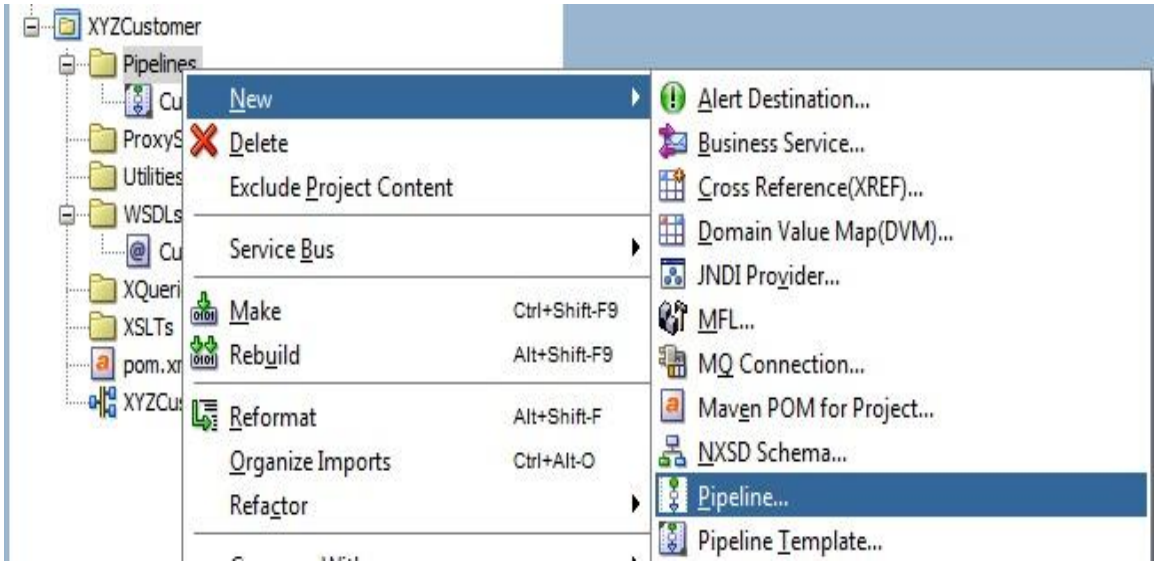
Creating Proxy Service

In this section, you will learn how to create pipeline using pipeline template and expose it as Proxy Service.

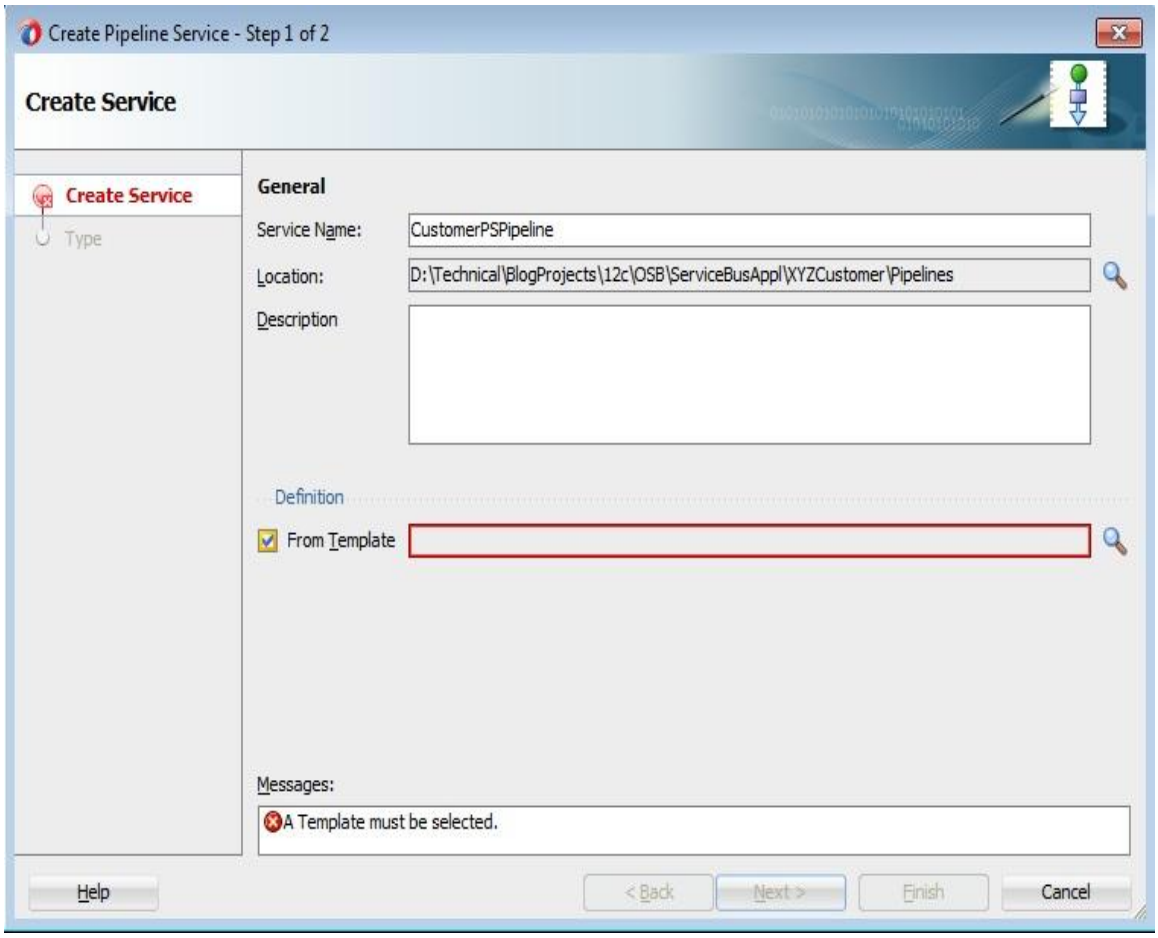
First things is, you should come up with a WSDL having **Create, Update, Delete and Merge** operations as mentioned in business requirements section that can be used for creating Proxy Service. You can come up with a WSDL or can get it from here. Copy the WSDL in **WSDLs** folder of **XYZCustomer** project using steps mentioned earlier.



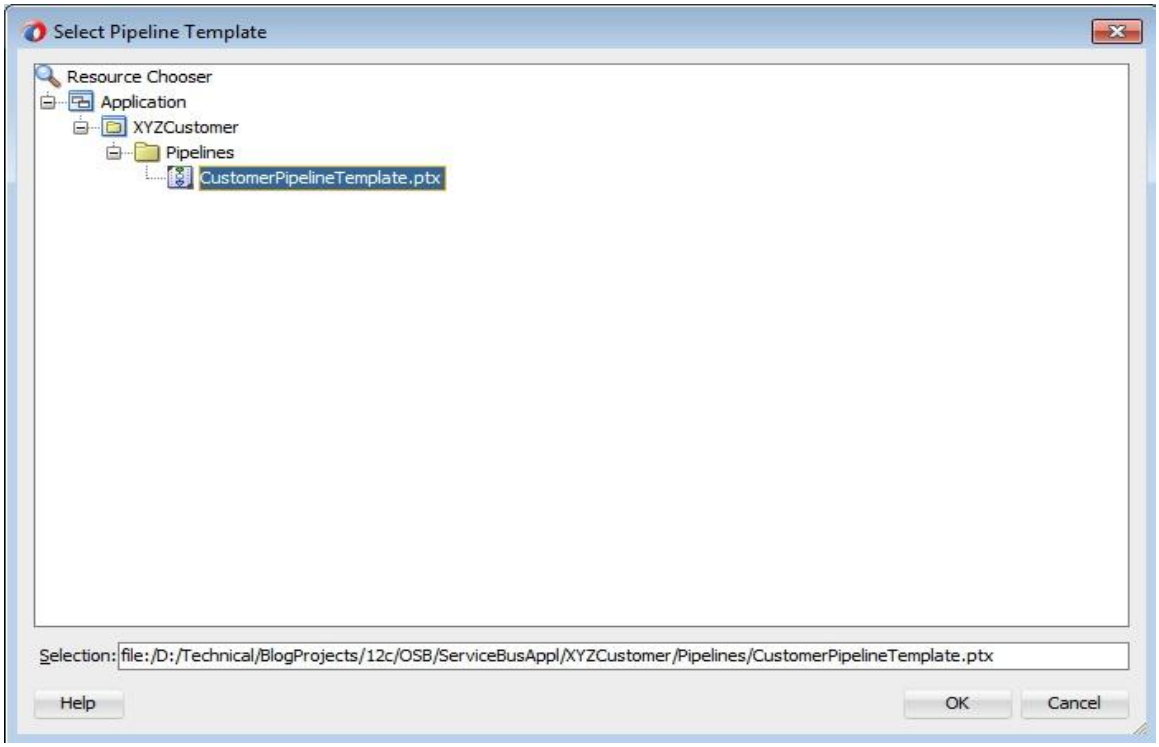
Right click **Pipelines** folder and select **New -> Pipeline**.



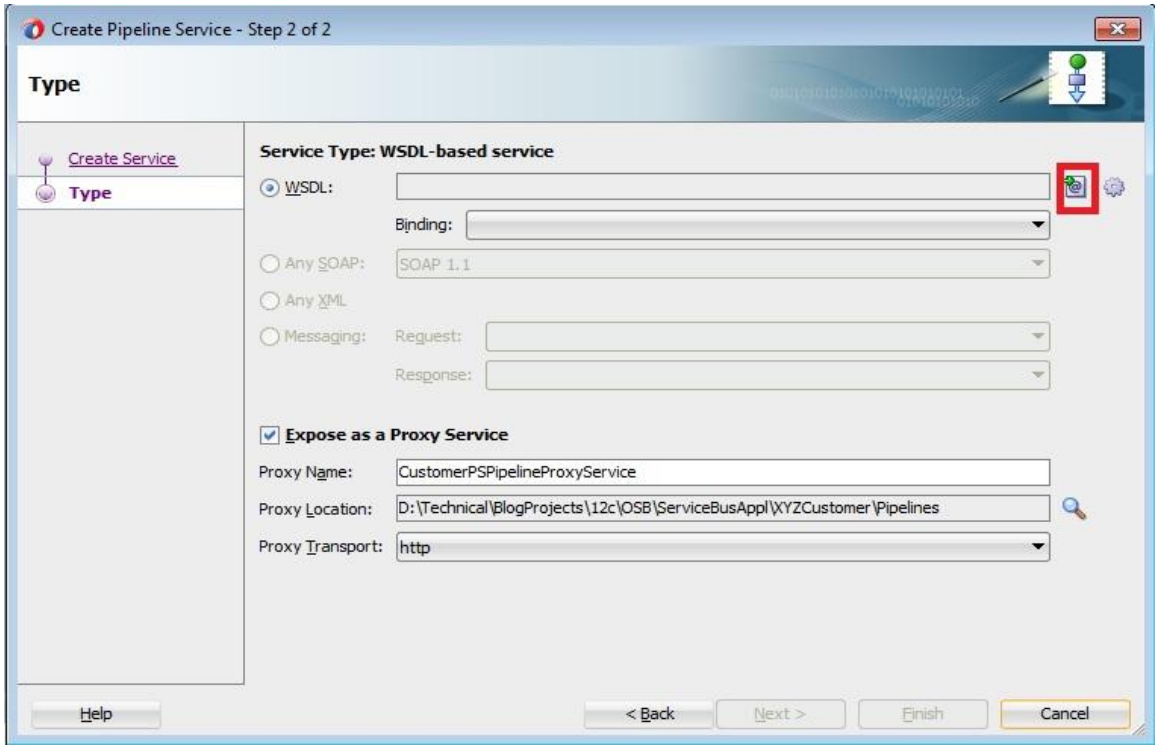
Give name as **CustomerPSPipeline** and select the option **From Template** as shown below.



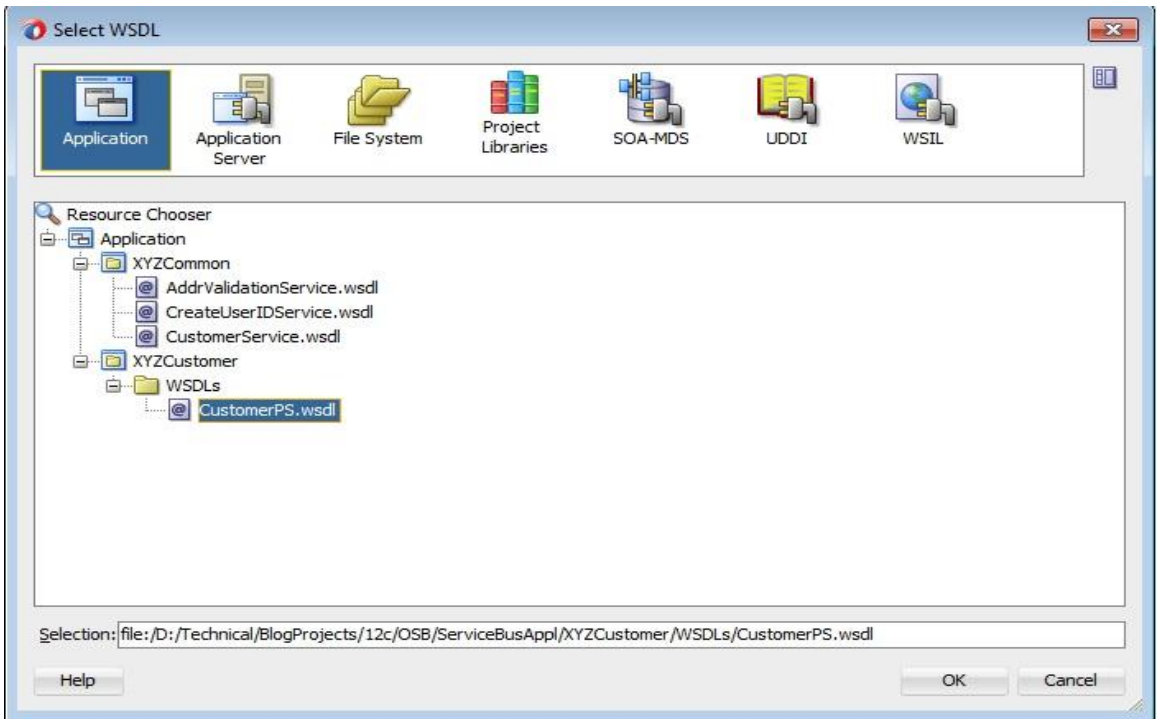
Click **Search** icon to bring up **Resource Chooser** and select pipeline template created in previous section.



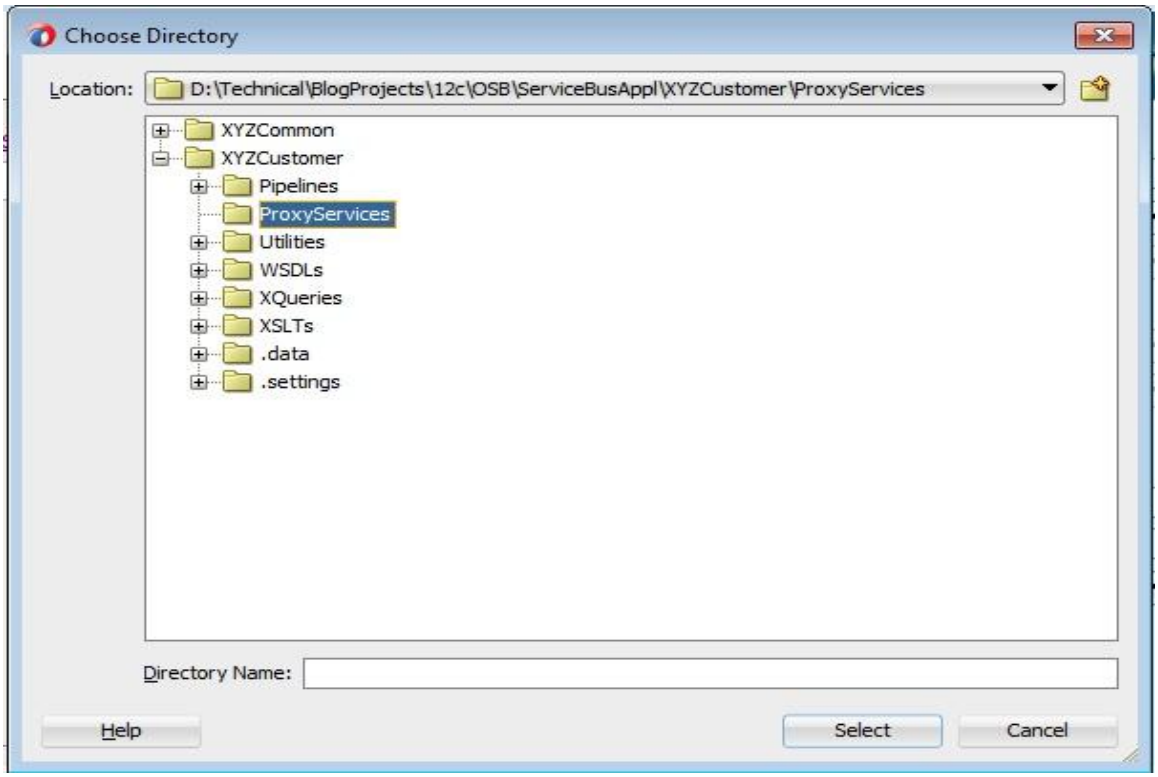
Click **OK** to go back to **Create Pipeline Service** window and click **Next**.



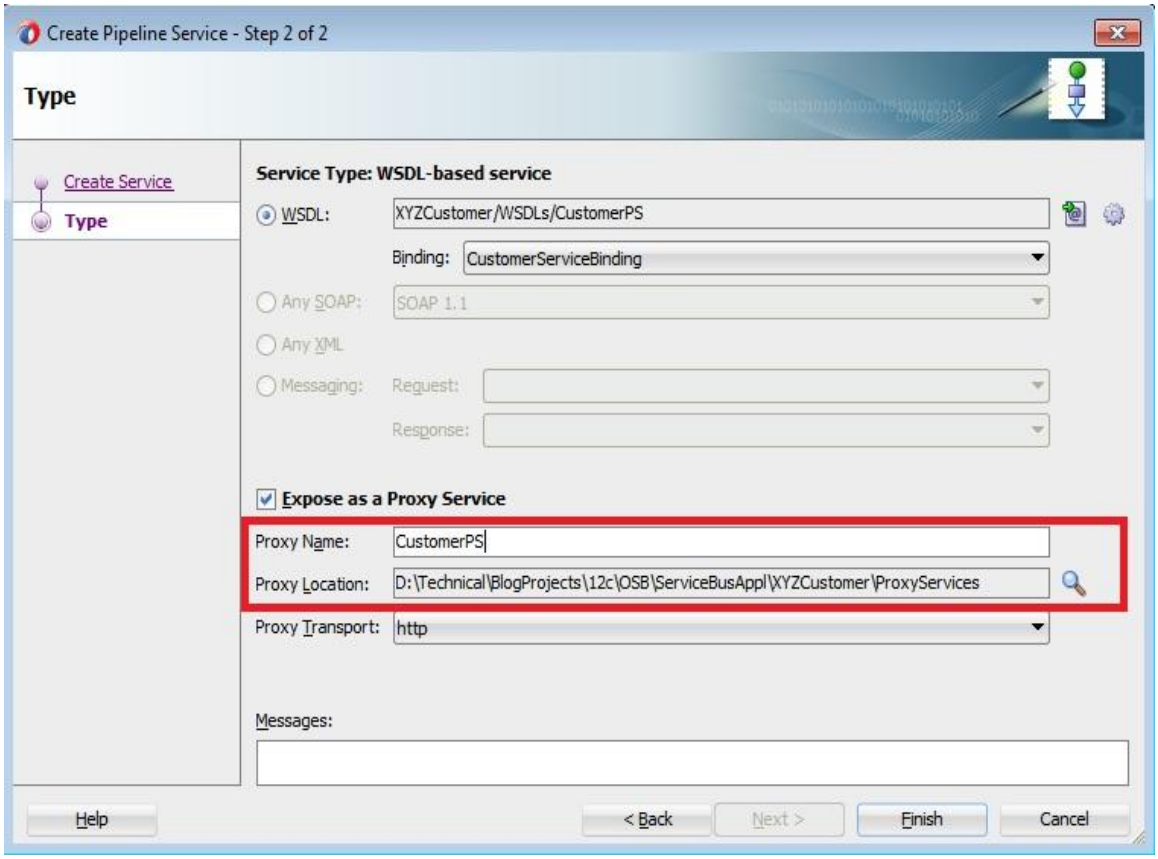
Click **Browse WSDLs** icon (highlighted above) to select WSDL defining your pipeline interface. Choose **CustomerPS.wsdl** from **XYZCustomer** project as shown below and click **OK**.



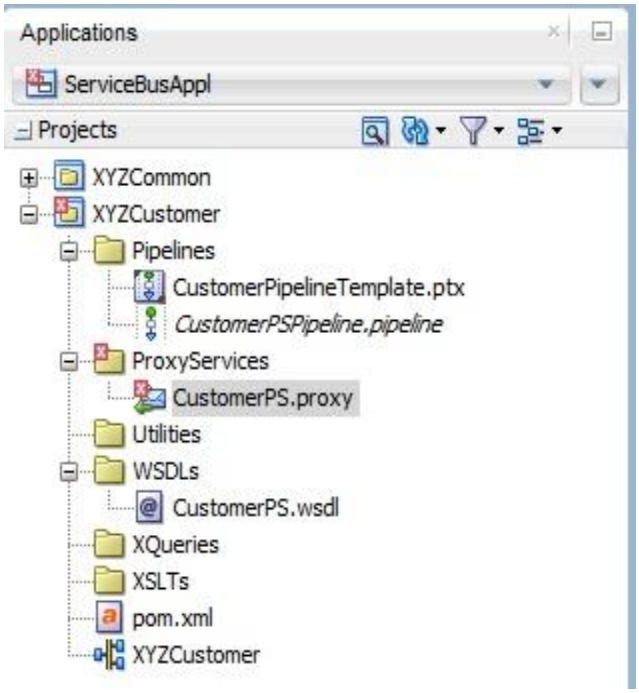
Verify that WSDL and binding is being selected. Click **Search** icon for **Proxy Location** and select **ProxyServices** folder as shown below.



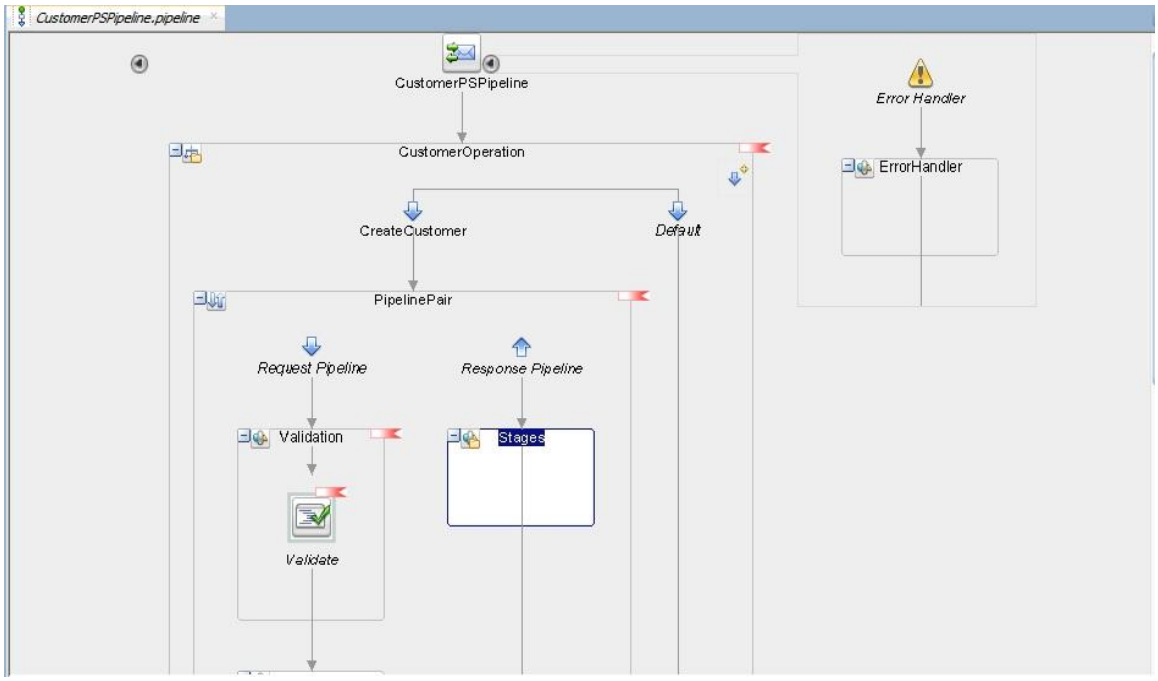
Click **Select** and modify **Proxy Name** to **CustomerPS**.



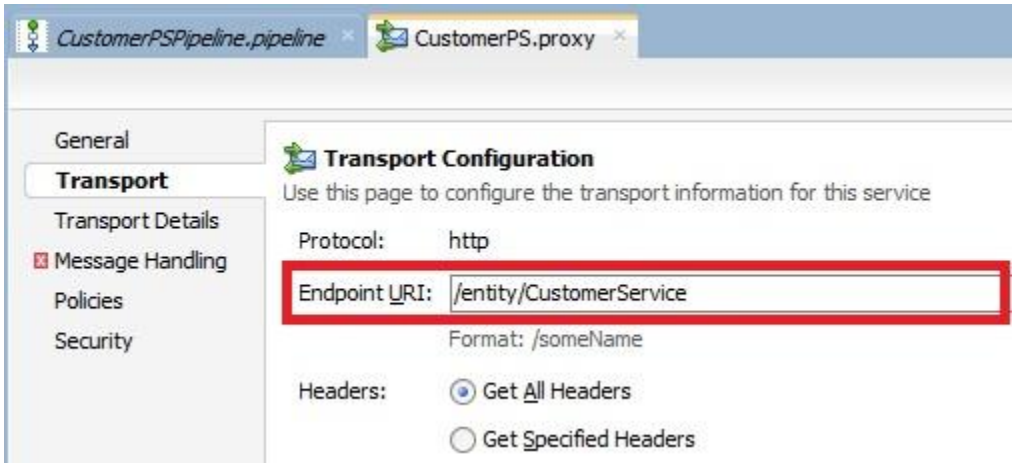
Click **Finish** and verify both **Pipeline** and **Proxy Service** are and shown in **Project Explorer**.



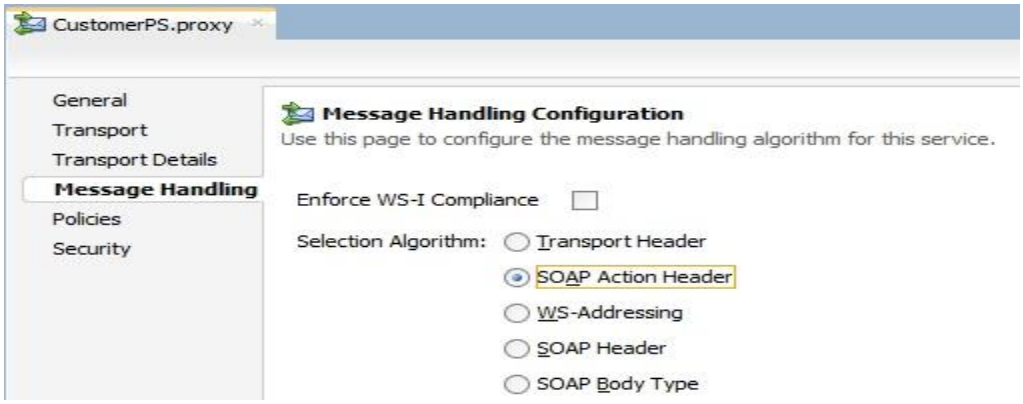
Also observe that new tab is opened showing **CustomerPSPipeline**. Observe that your pipeline had inherited all placeholders, names and properties from pipeline template. And you will be allowed to modify/add activities to placeholders and properties of the activities inherited from template. Since a few of the properties which are mandatory in concrete pipelines but not mentioned in template, you are seeing red marks both in editors and **Project Explorer**.



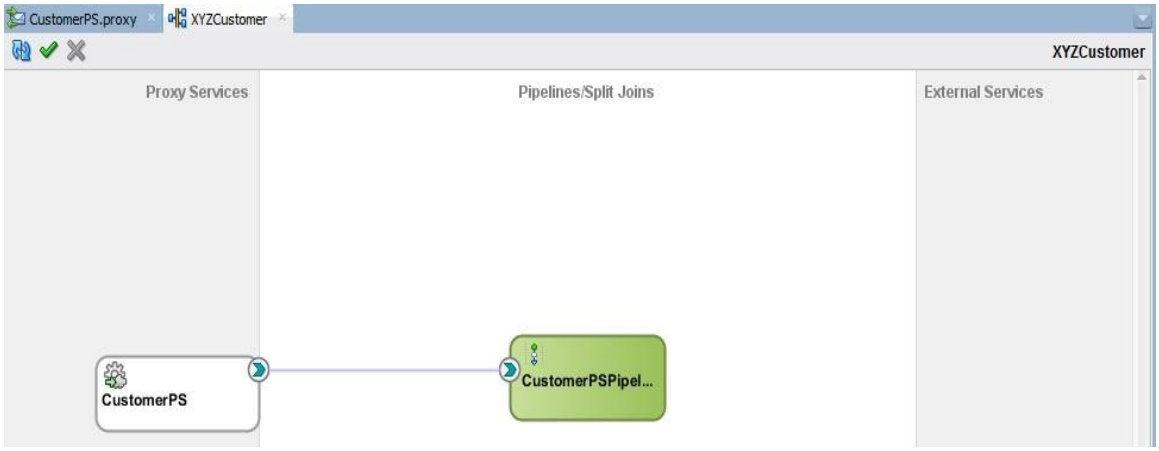
Open **CustomerPS** and navigate to **Transport** configuration as shown below. Modify Endpoint URI as **/entity/CustomerService**. So service consumers can access your proxy service using URL **http://<host>:<port>/entity/CustomerService.**



Go to **Message Handling** tab and select **SOAPAction Header** as **Selection Algorithm.** This would enable proxy service to determine operation at runtime based on **SOAP Action** in HTTP headers. These changes would resolve the errors in Proxy Service and observe all red marks are gone.



Now your **Service Bus Overview Editor** should look like below. Left swim lane is for services providing an entry point to application so limited to **Proxy Services**. Middle swim lane is for components providing routing and transformation so limited to **Pipelines** and **Split-joins**. Right swim lane is for references providing communication with actual service providers so limited to **Business Services** or **Proxy Services**. For more information refer to this link.

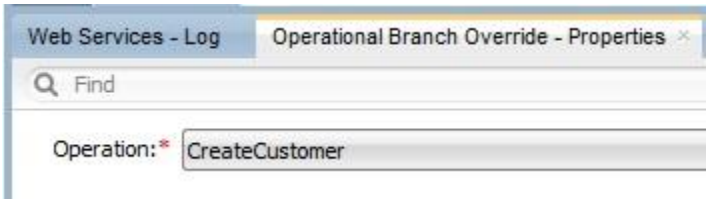


Creating Message Flow

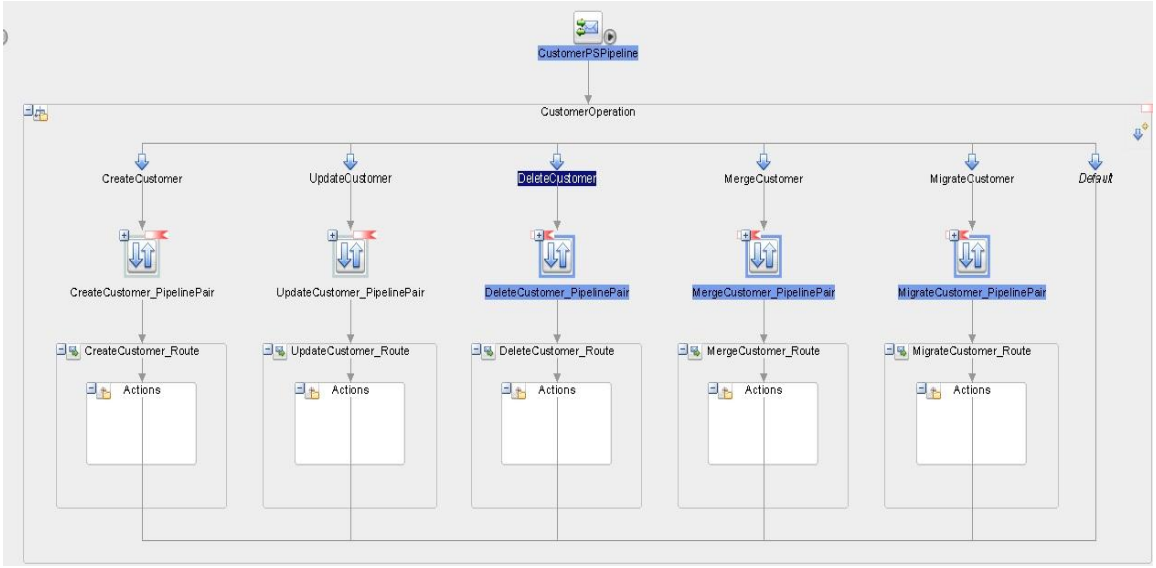
In 12c, message flow is separated from Proxy Service and incorporated in separate component called **Pipeline**. In this section, you will see creating message flow in Pipeline.

As observed in previous section, the **CustomerPSPipeline** already inherited the initial message flow from pipeline template. So in this section, you will start modifying this message flow and add more activities in placeholders provided by pipeline template.

Select **CreateCustomer** branch and set operation as **CreateCustomer** in Properties tab as shown below.



Click icon  to add 4 more additional branches and select **UpdateCustomer**, **DeleteCustomer** **MergeCustomer** and **MigrateCustomer** operations respectively. Now your message flow should look like below. You can also observe that each of these operational branches inherited same set of activities/placeholders as created in pipeline template.

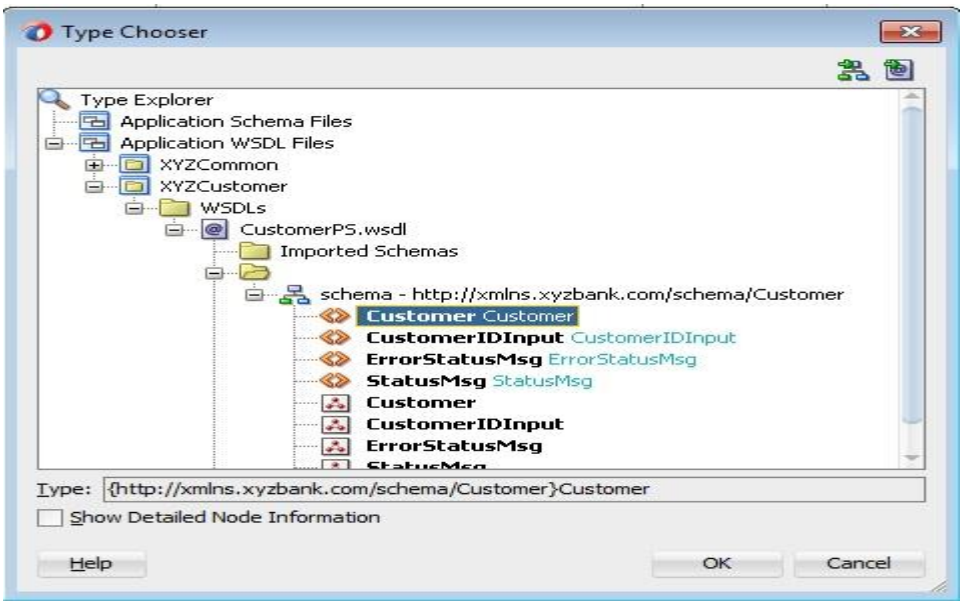


Create Customer

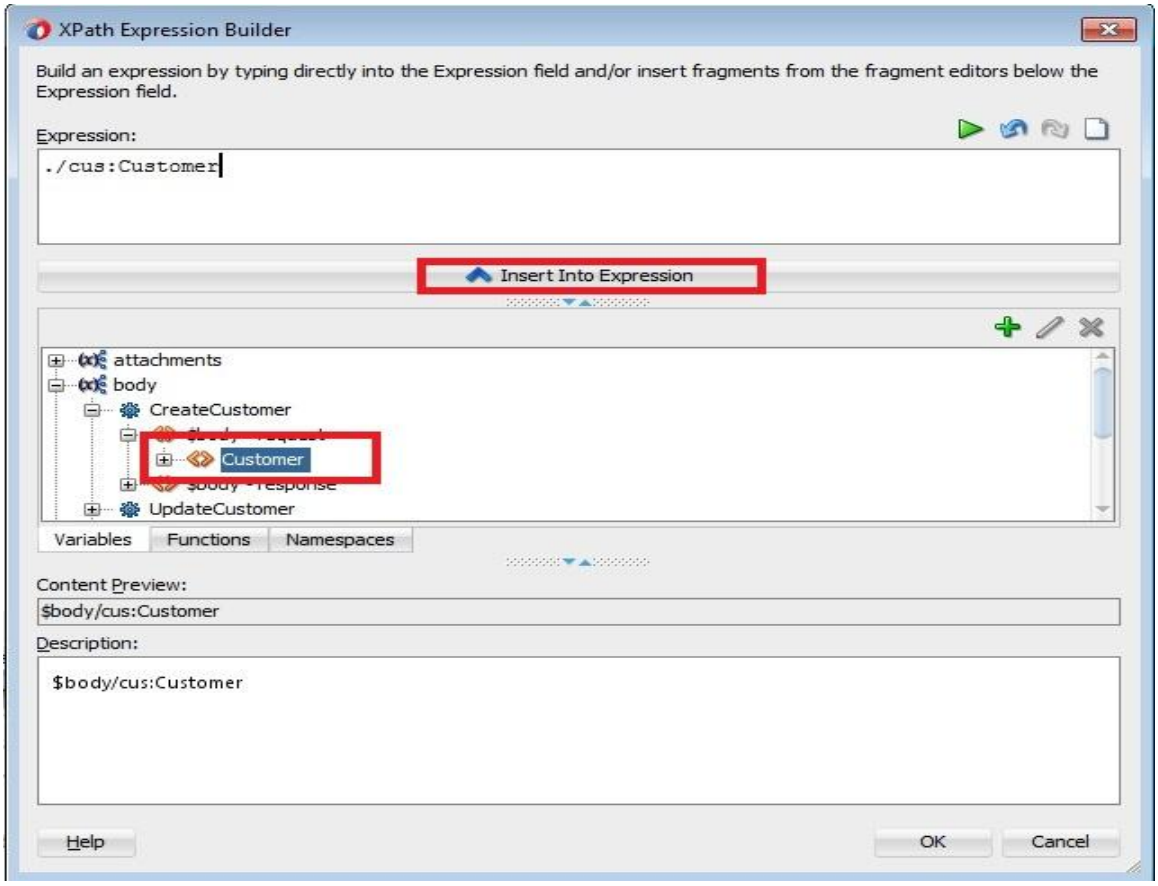
In this section, you will complete message flow for **CreateCustomer** branch.

Validating Payload

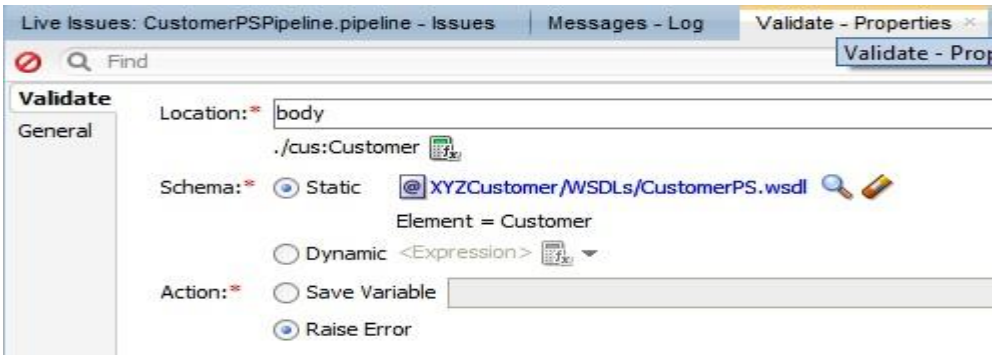
In this section, you will complete the **Payload** and **Email** format validation. In **Properties** tab of Validate activity, for **Schema** property click **Search** icon and select **Customer** element from proxy service WSDL as shown below.



For **Location** property bring up expression editor by clicking icon. Drag or shuttle **Customer** element into **Expression** field as shown below and click **OK**. Now you can observe all red marks are no longer visible for **CreateCustomer** branch as you set required properties.



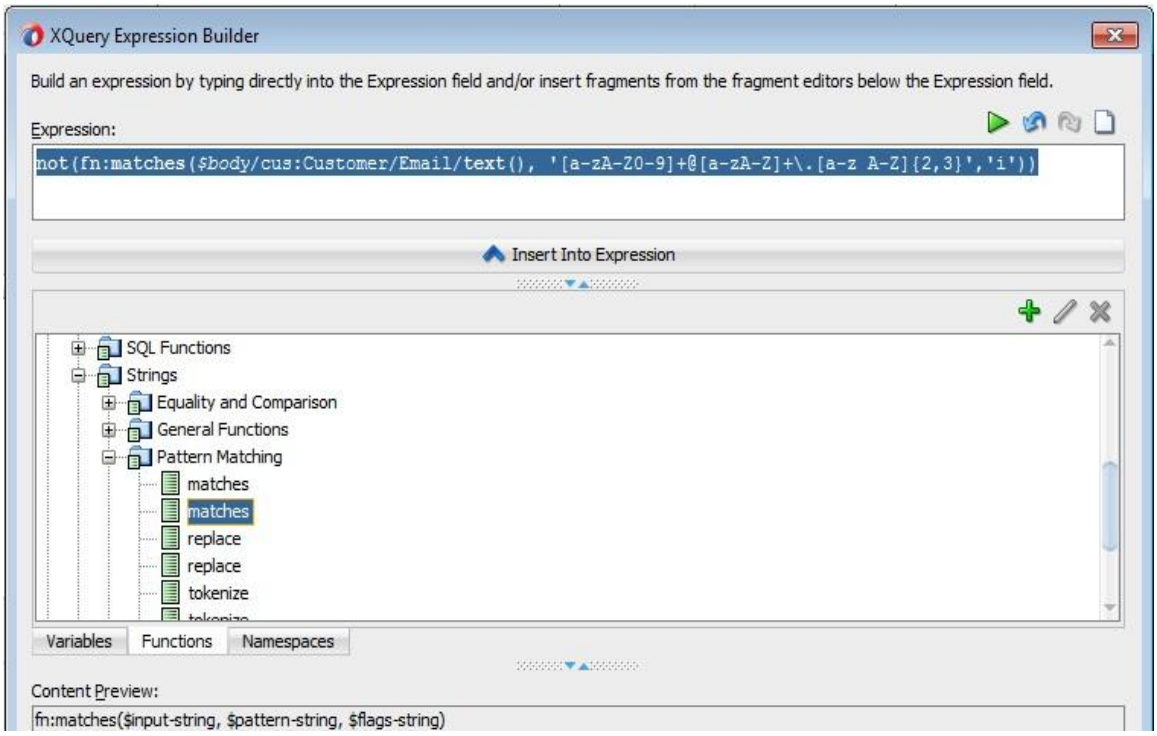
Now your properties tab for **Validate** activity should look like below.



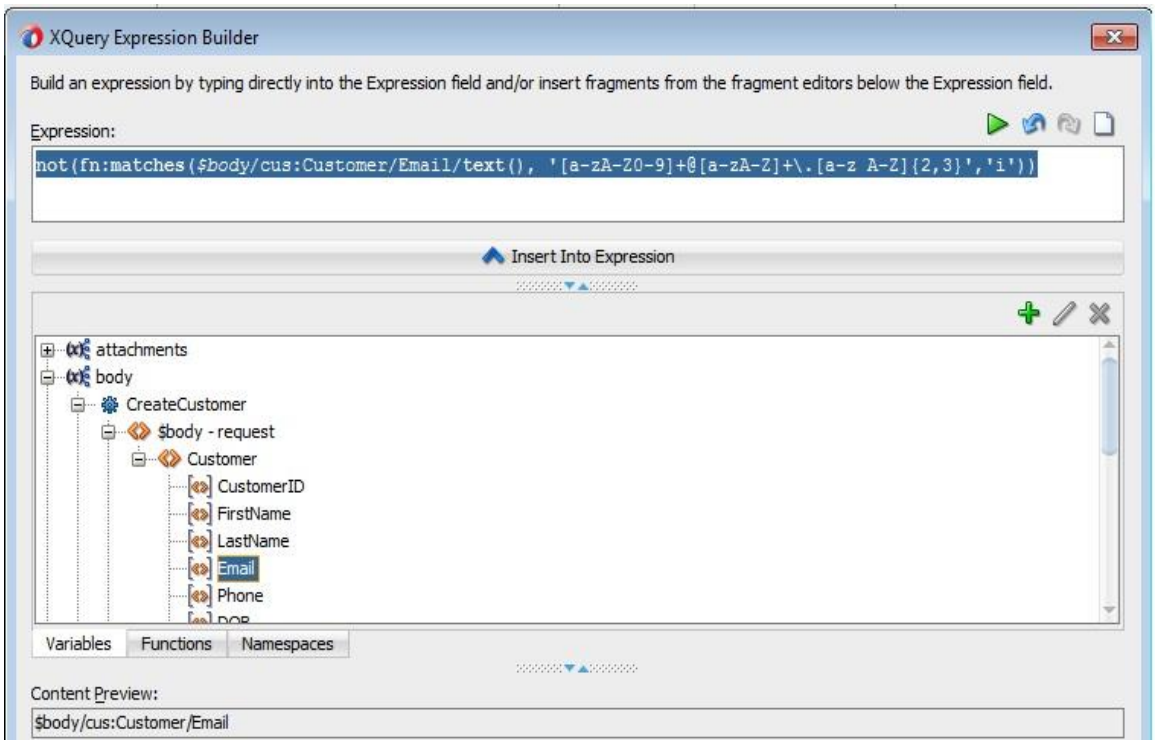
To validate **Email** element, drag **If-Then** activity into **Actions** placeholder of **Validation** stage from F**low Control.** In **Properties** tab, bring up expression editor for **Condition** property and give the **Expression** as:

not(fn:matches($body/cus:Customer/Email/text(), '[a-zA-Z0-9]+@[a-zA-Z]+\.[a-z A-Z]{2,3}','i'))

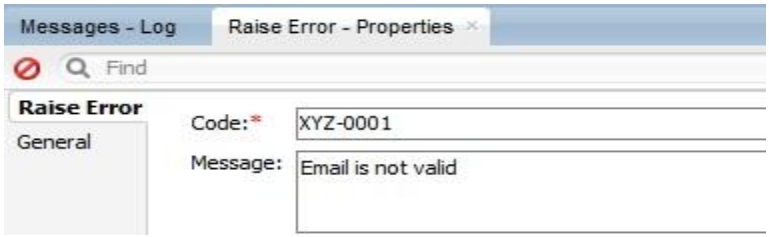
Alternatively, you can drag **fn:matches** function from **Functions->XQuery 1.0 Functions -> Strings -> Pattern Matching -> matches** as shown below.



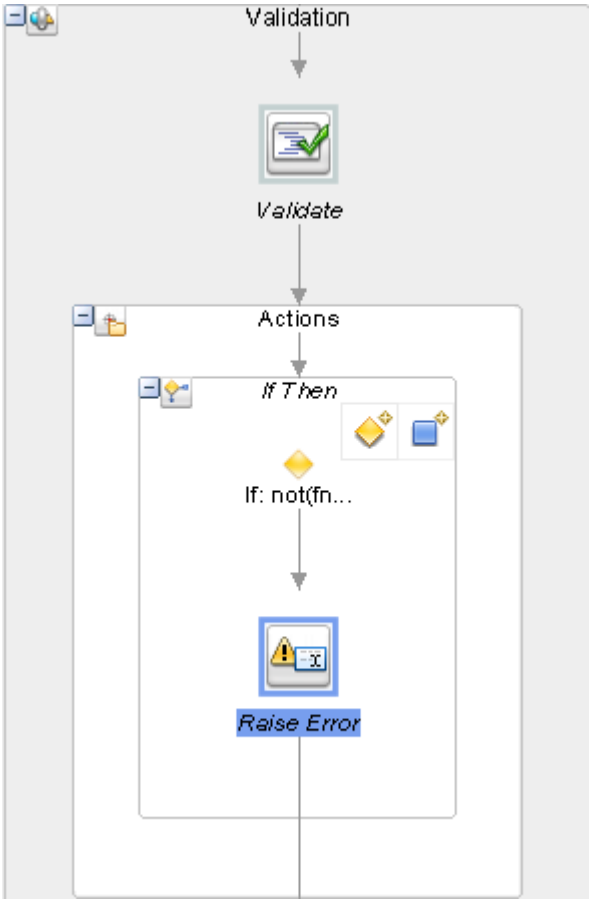
Also you can drag **Email** element from **Variables -> body -> CreateCustomer -> $body – request -> Customer -> Email** as shown below and modify expression to above condition.



Drag **Raise Error** activity into **If** branch from **Flow Control**, and set properties as shown below. You can remove **Else** branch by selecting **Delete on** right click.



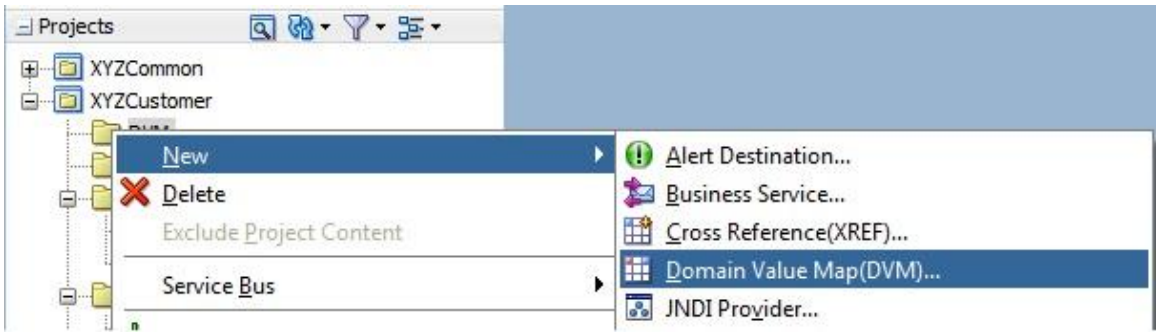
Now your **Validation** stage should look like below.



Validating Address

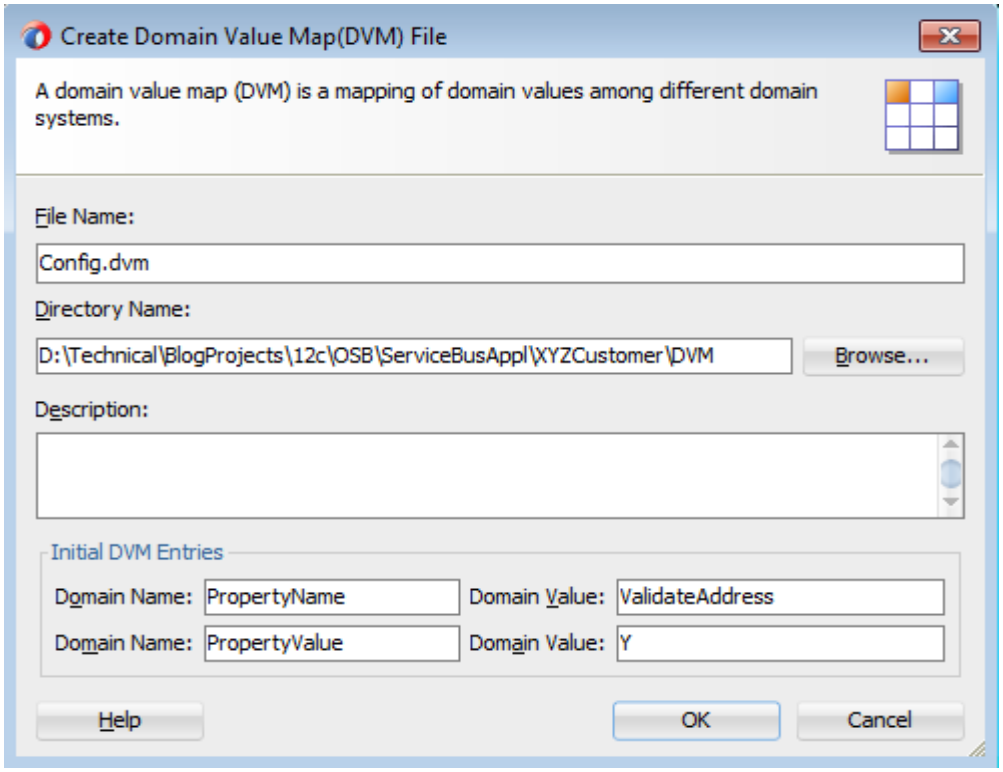
In this section, you will perform **Address Validation** by invoking **AddrValidationServiceBS** based on a configurable property.

You will use **Domain Value Map (DVM)** to store configurable properties. Create a new folder named **DVM** in **XYZCustomer** project. Right click **DVM** folder and select **New -> Domain Value Map**.

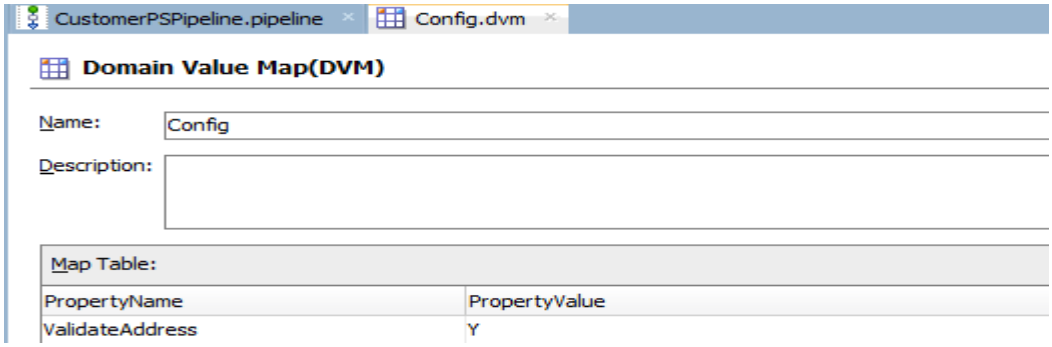


Give **Config** as **File Name** and give initial DVM entries as shown below. This way, basically you are defining different properties that can be used in Message Flow along with initial values. The domain names **PropertyName** represent configurable properties and **PropertyValue** represents property values respectively.

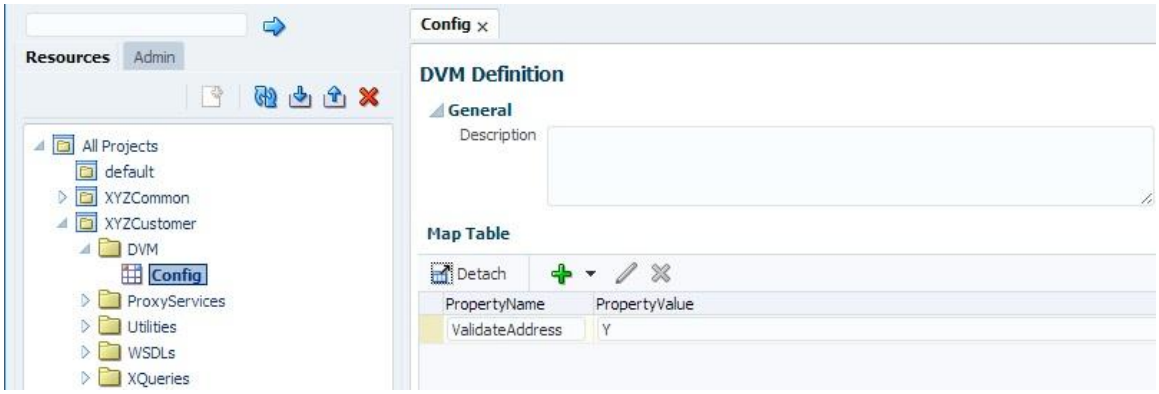
Using DVM, you can extend same logic to any number of configurable properties in your Service Bus Application. For current scenario, give property name as **ValidateAddress** and its value as **Y.**



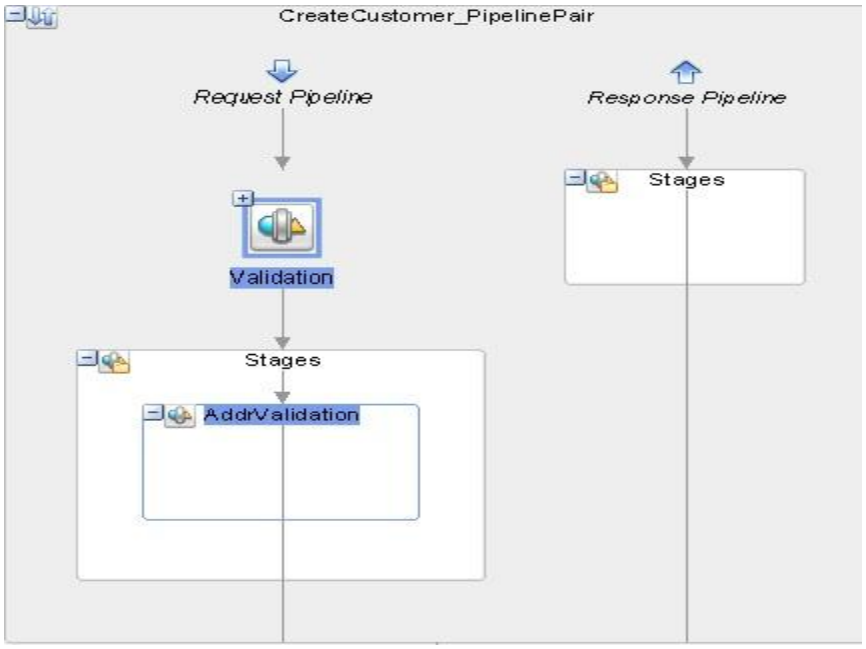
Click **OK** to bring up DVM editor.



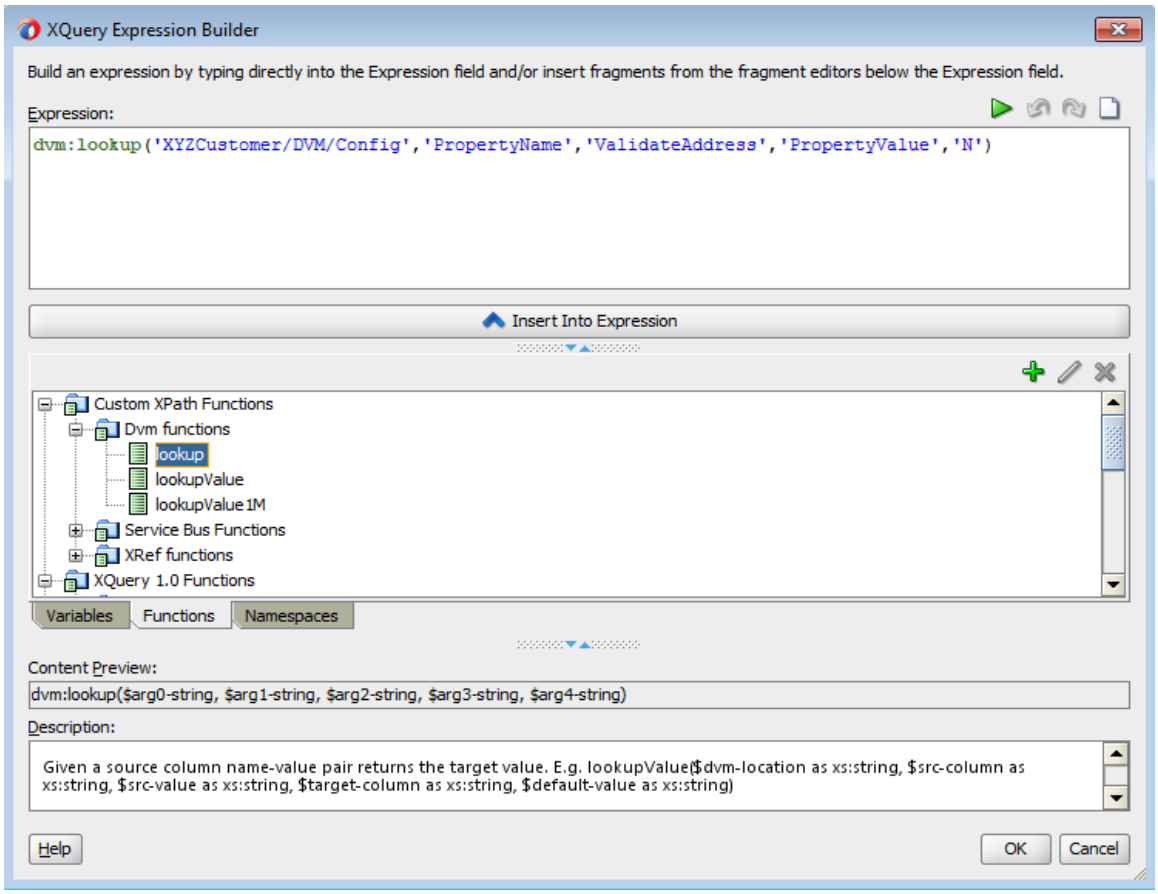
You can always modify DVM in your IDE and deploy to your server or you can also modify it in **sbconsole.** Remember you need to create a session before modifying anything.



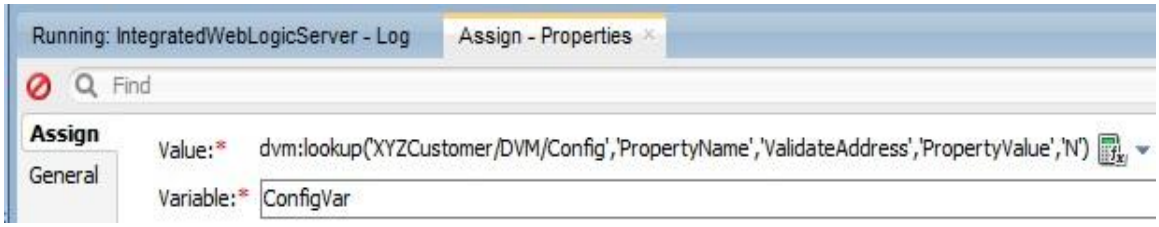
Drag **Stage** node into **Stages** placeholder of **Request Pipeline** and set name as **AddrValidation** in **Properties** tab.



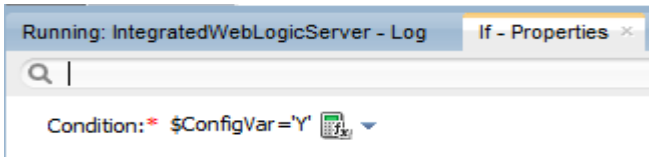
Drag **Assign** activity into **AddrValidation** stage from **Message Processing.** Bring up expression builder for **Value** property and give expression as shown below. Alternatively you can select the function from **Functions -> Custom XPath Functions -> Dvm functions**. Make sure that you give complete qualified path of DVM containing Service Bus Project name too.



Set **Variable** property as **ConfigVar.** Now your **Properties** tab should look like below.

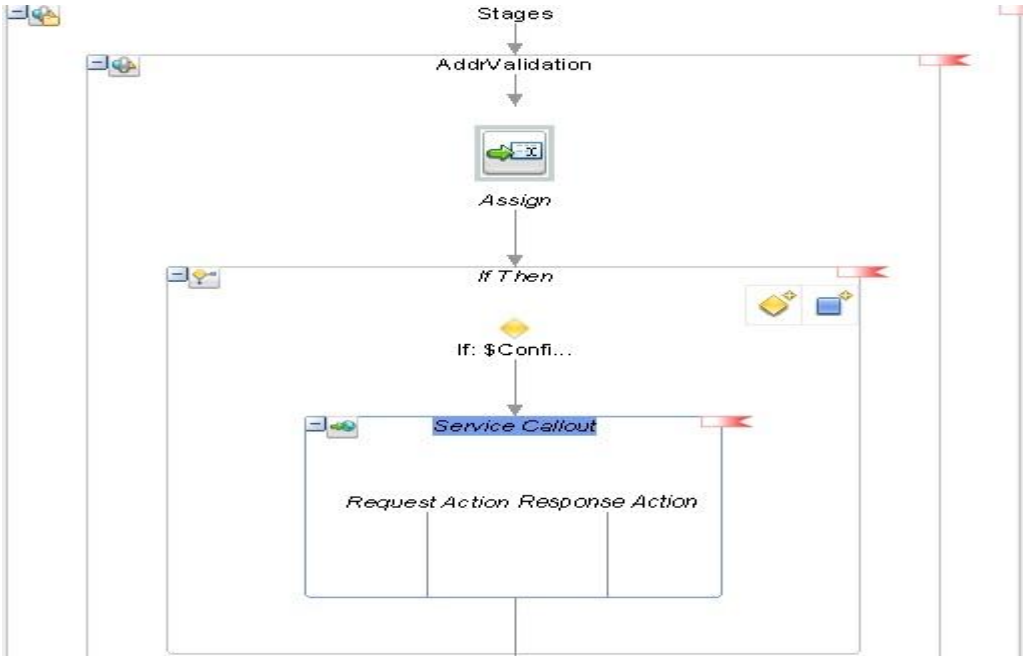


Address Validation business service should be invoked only when the value of this configuration property 'Y'. So drag **If Then** activity into **AddrValidation** stage after **Assign** from **Flow Control** and set condition as ***$ConfigVar='Y'***.

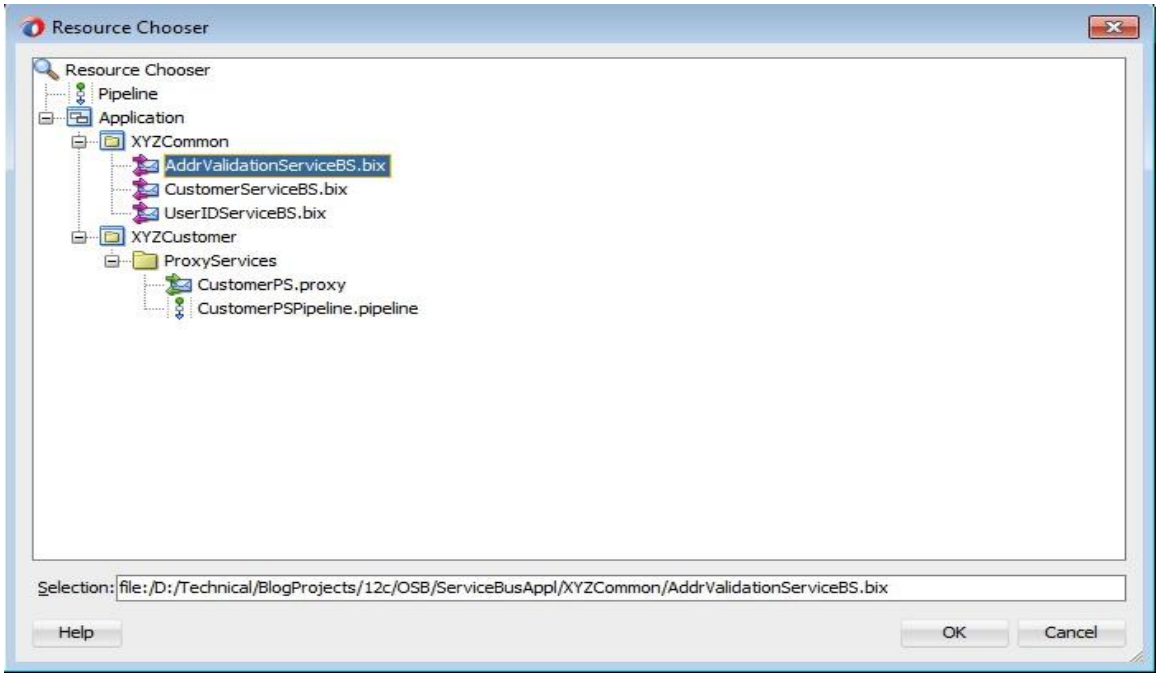


Service Bus provides **Service Callout, Routing** and **Publish** activities to route incoming request to an appropriate business service. **Routing** and **Service Callout** are used for Synchronous where as **Publish** activity is used for one-way communication.

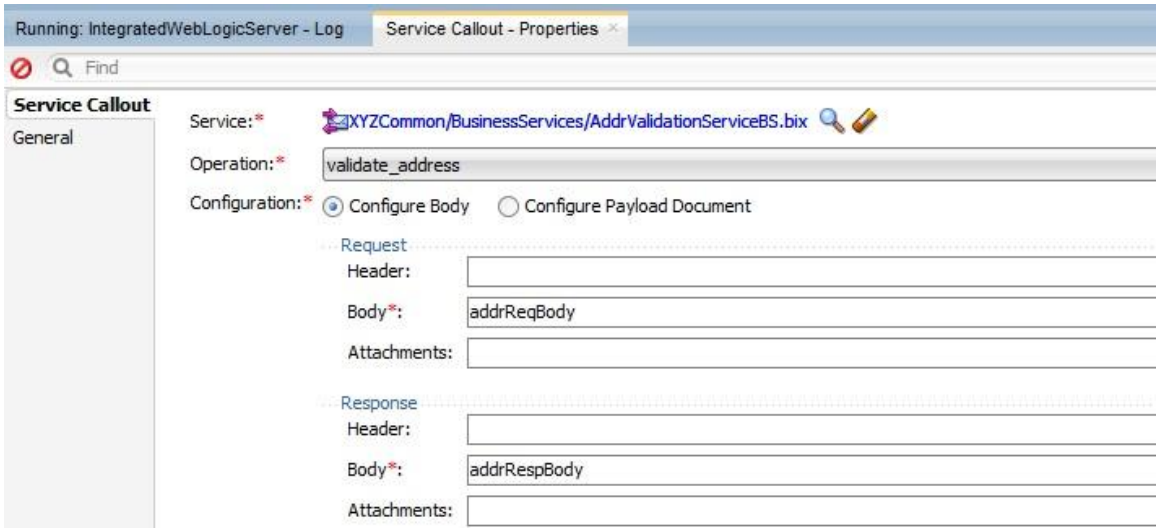
As mentioned earlier, **Routing** node always depicts end of message flow so you can't place any activities after **Routing** node. Hence we use **Service Callout** for all intermediate service calls in pipeline. So drag **Service Callout** from **Communication** into **If Then** branch as shown below.



In **Properties** tab, browse and select **AddrValidationServiceBS** for **Service** property.



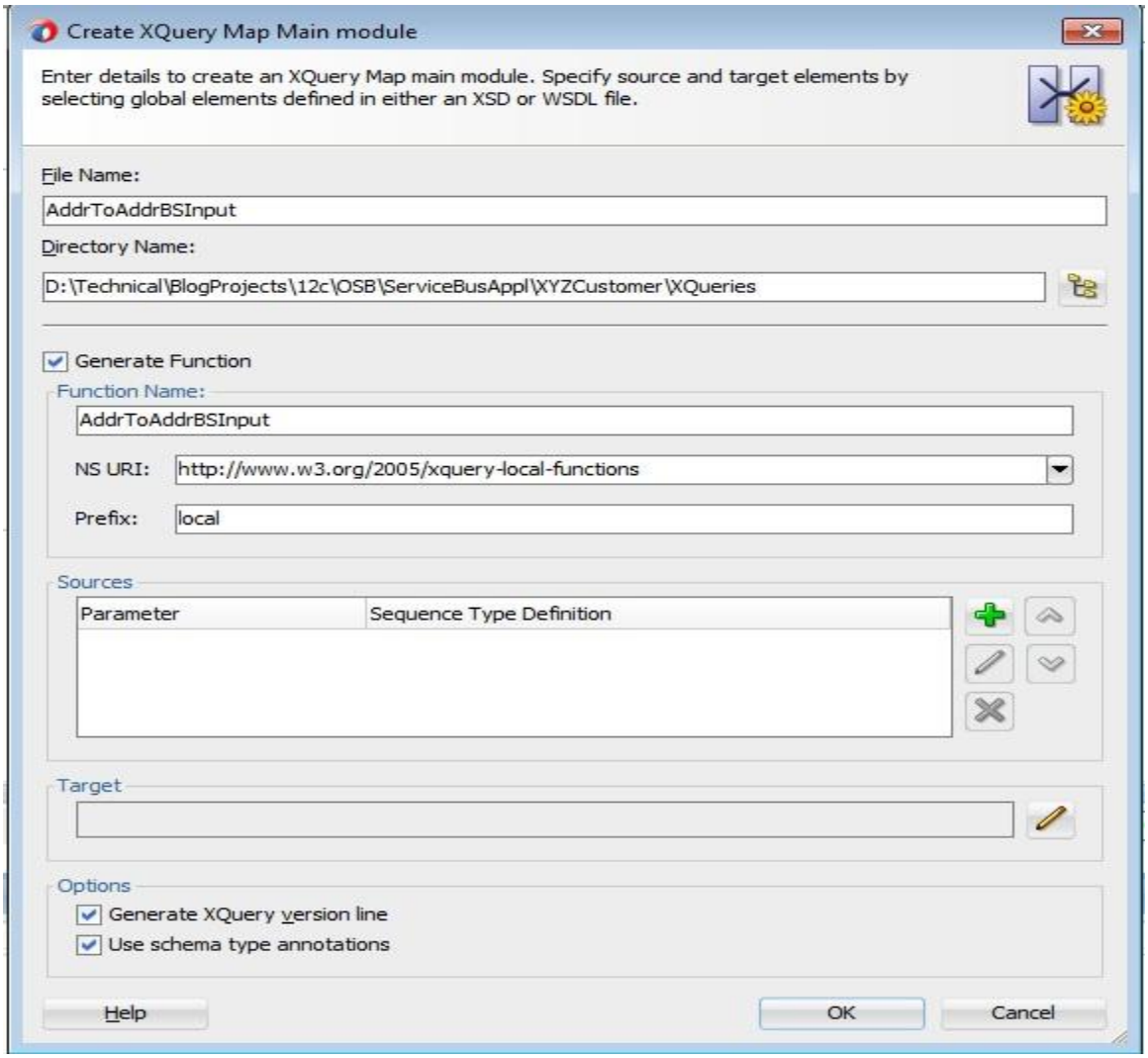
Select **validate_address** operation and set other properties of **Service Callout** as shown below.



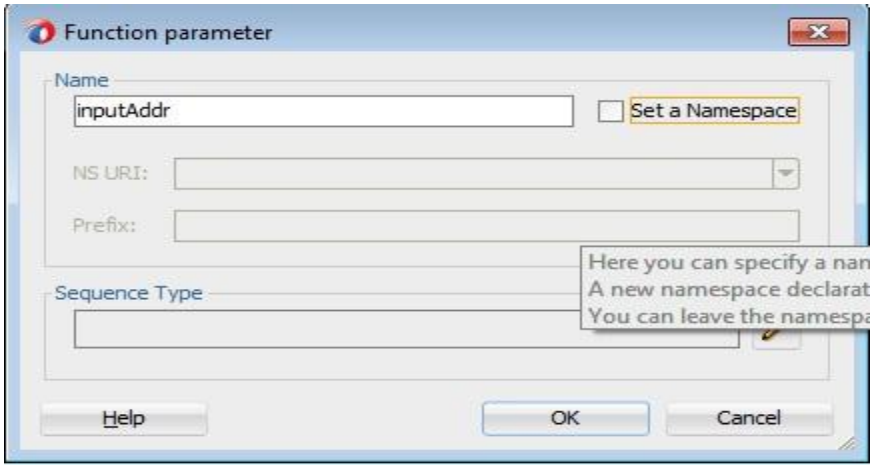
addrReqBody is variable to be populated with the payload for **valdiate_address** operation and **addrRespBody** is variable that contains response returned by business service after service call. Since you are using the '**Configure Soap Body**' option, make sure that request variable is populated with complete payload including SOAP Body element (**soap-env:Body).** In the same way, response variable contains SOAP Body element when response is received.

Next task is transforming the received payload to the structure expected by address validation business service. To achieve this, you can either use **XQuery** or **XSLT** maps. One of the added features in 12c is JDeveloper has complete design time support for XQuery as well. First we will get hands-on in using XQuery files so let us start creating one serving your purpose.

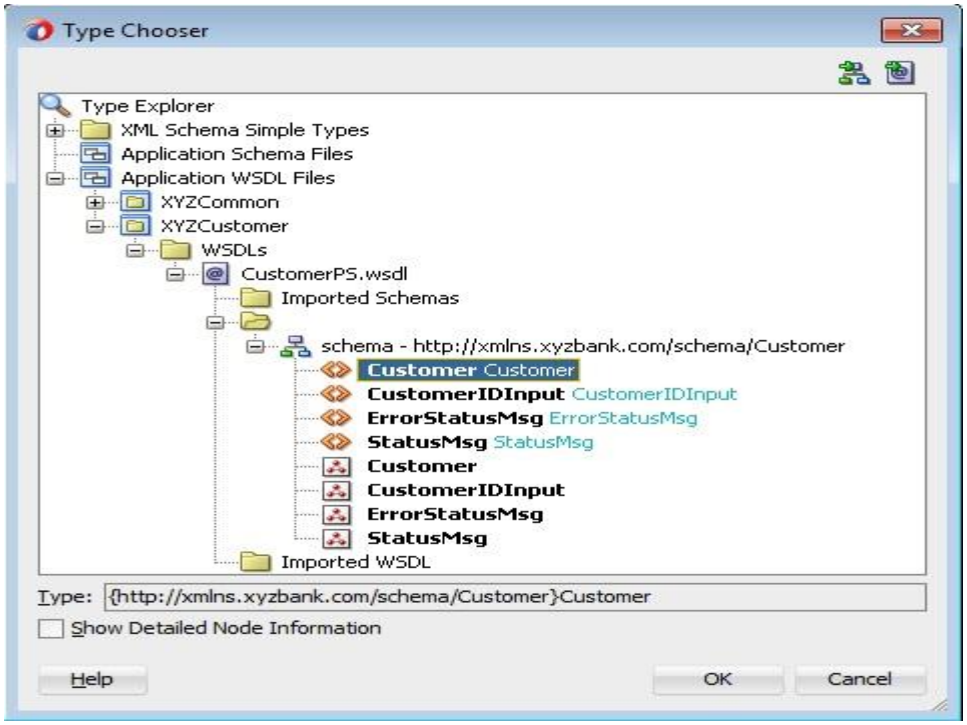
Right click on **XQueries** folder in **XYZCustomer** project and select **New -> XQuery File ver1.0** to create new **XQuery.** Give both file and function name as **AddrToAddrBSInput** as shown below.



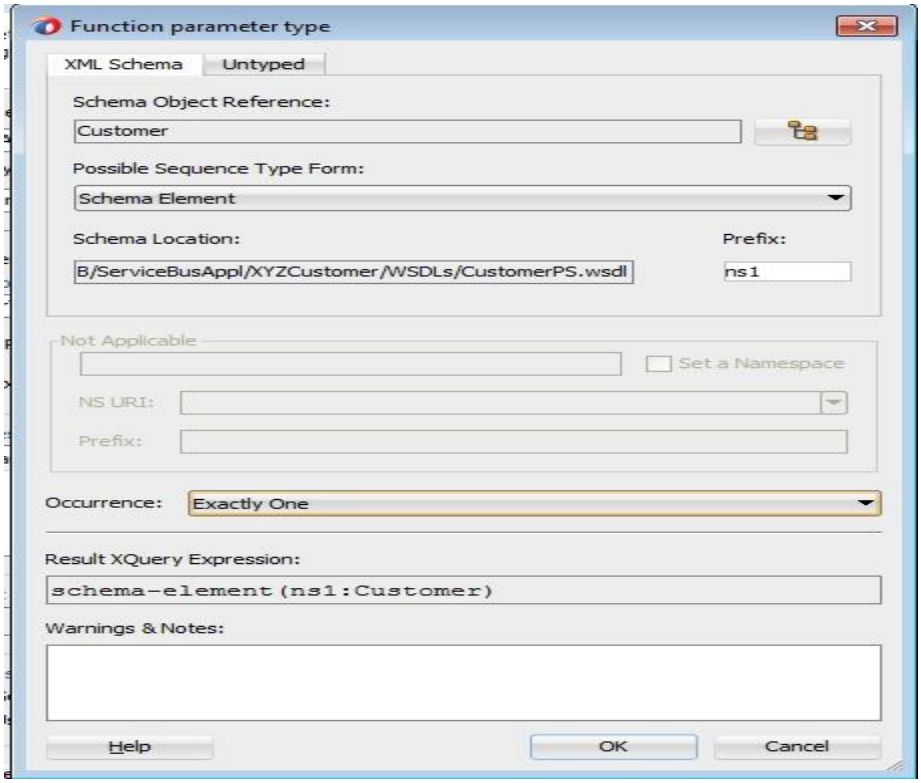
Click **+** icon to add Source parameter and give name as **inputAddr**.



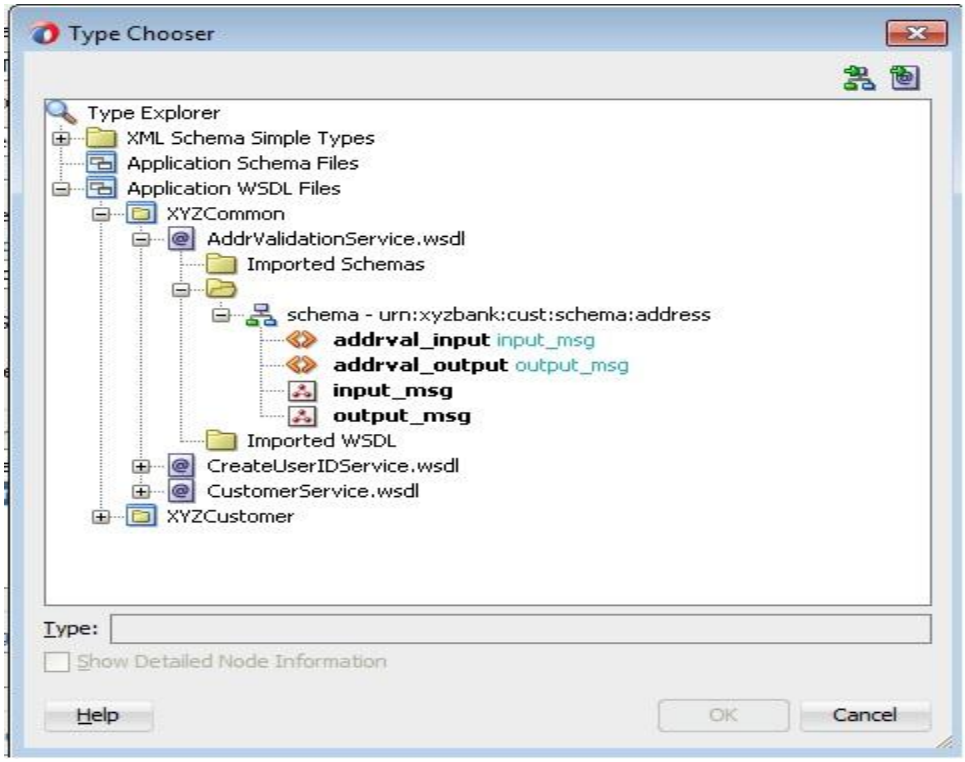
Click pencil icon for **Sequence Type** to bring up **Function Parameter Type** window. Click browse icon and select **Customer** element in **CustomerPS.wsdl** as shown below.



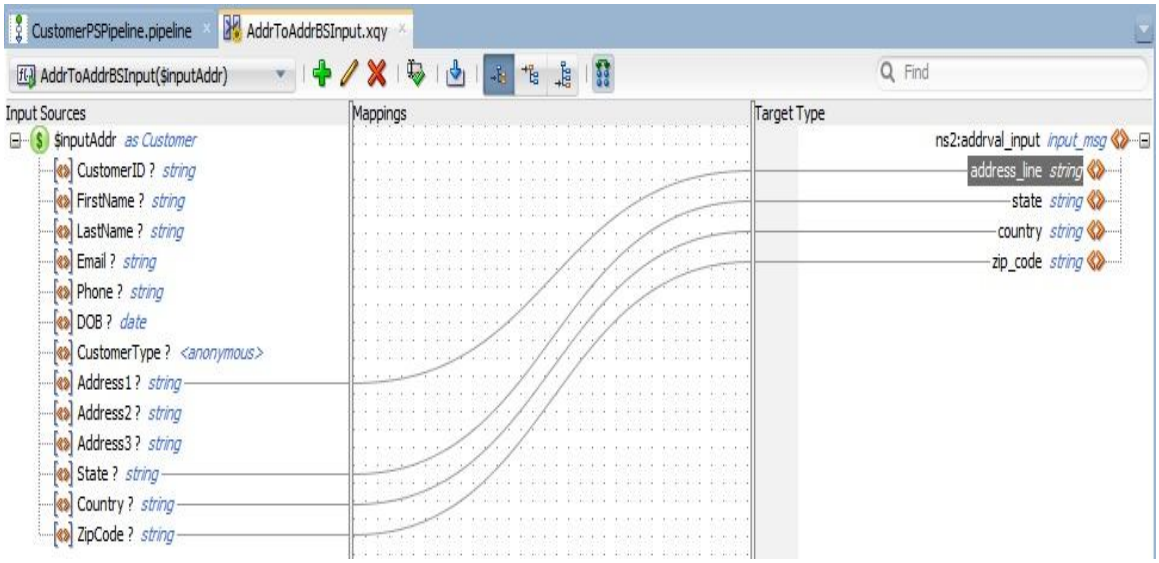
Click **OK** and retain default values for all other properties in **Function Parameter Type** window.



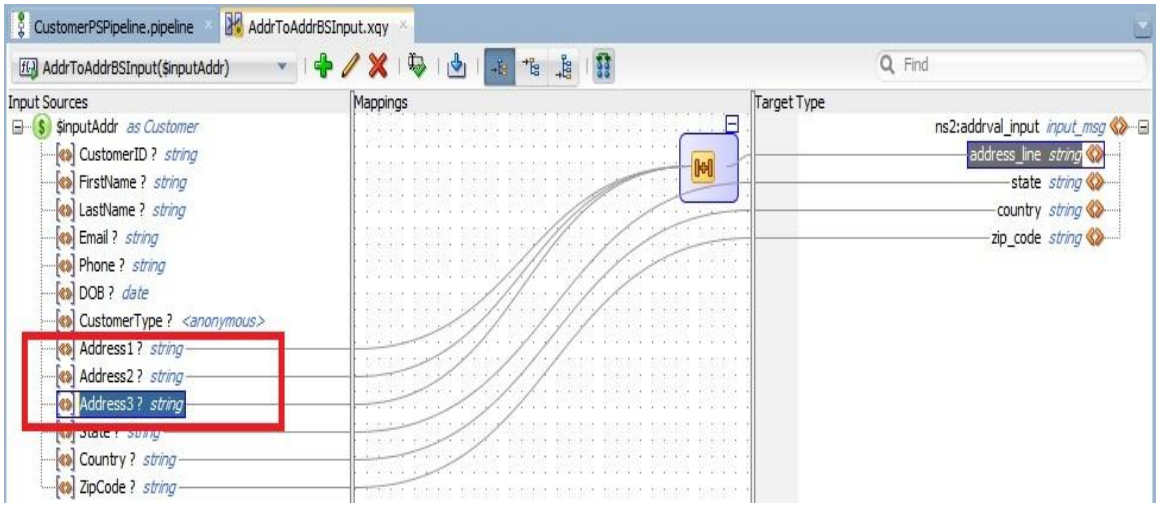
Go back to **Create XQuery Map** window by clicking **OK** twice. Now click pencil icon for **Target** to bring up **Function Result Type** window. Click browse icon and select **addrval_input** element as shown below.



Click **OK** thrice to bring up transformation editor. Do the initial mapping of attributes as shown below. You can map the fields by connecting source element to target element.

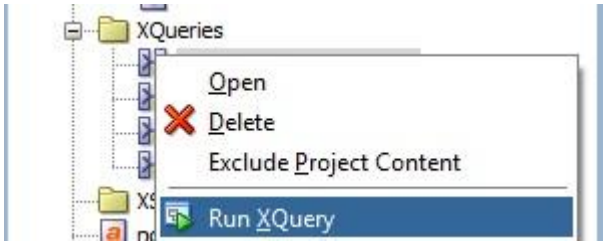


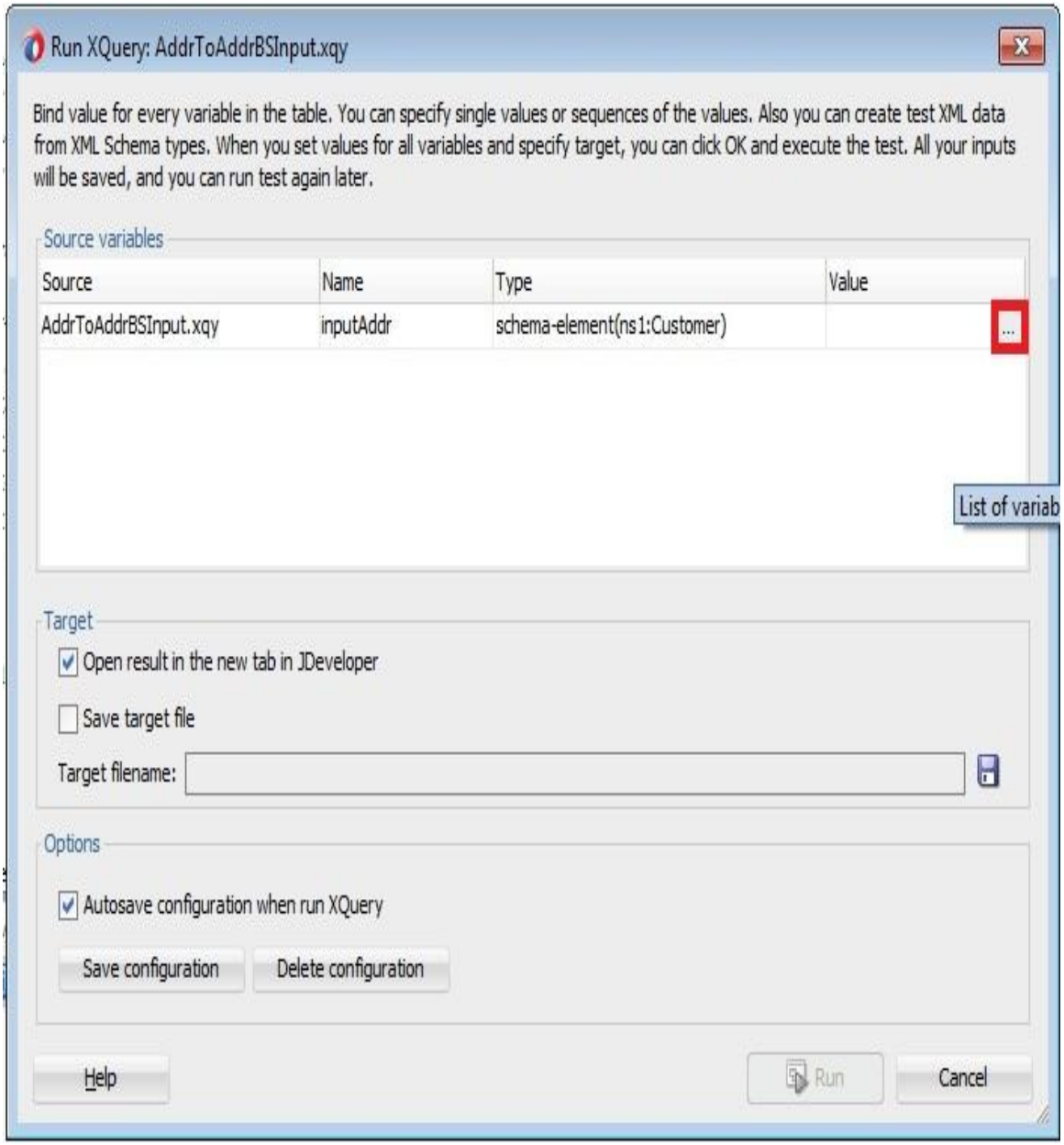
Drag **concat** function from **Components -> XQuery Functions -> String Functions** into **Mappings** swim lane as shown below and connect **Address2** and **Address3** input elements to this function.



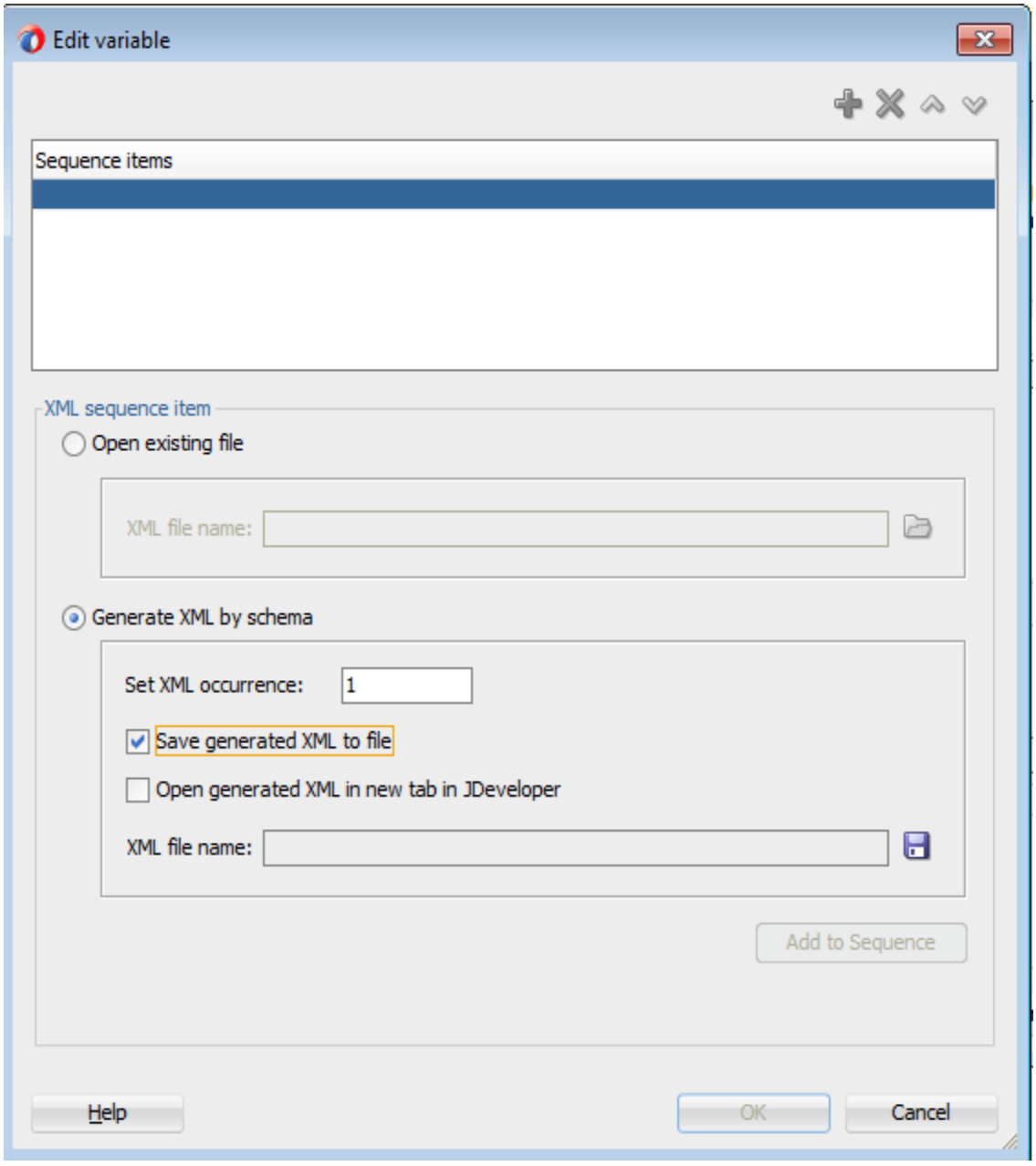
Address Validation service expects address lines with **','** as delimiter, so go to **XQuery Source** tab and modify **concat** function as shown below.

It's always advisable to test your **XQuery** mapping to verify that mapping is working as expected. Right click **XQuery** file in **XYZCustomer** project and select **Run XQuery**.

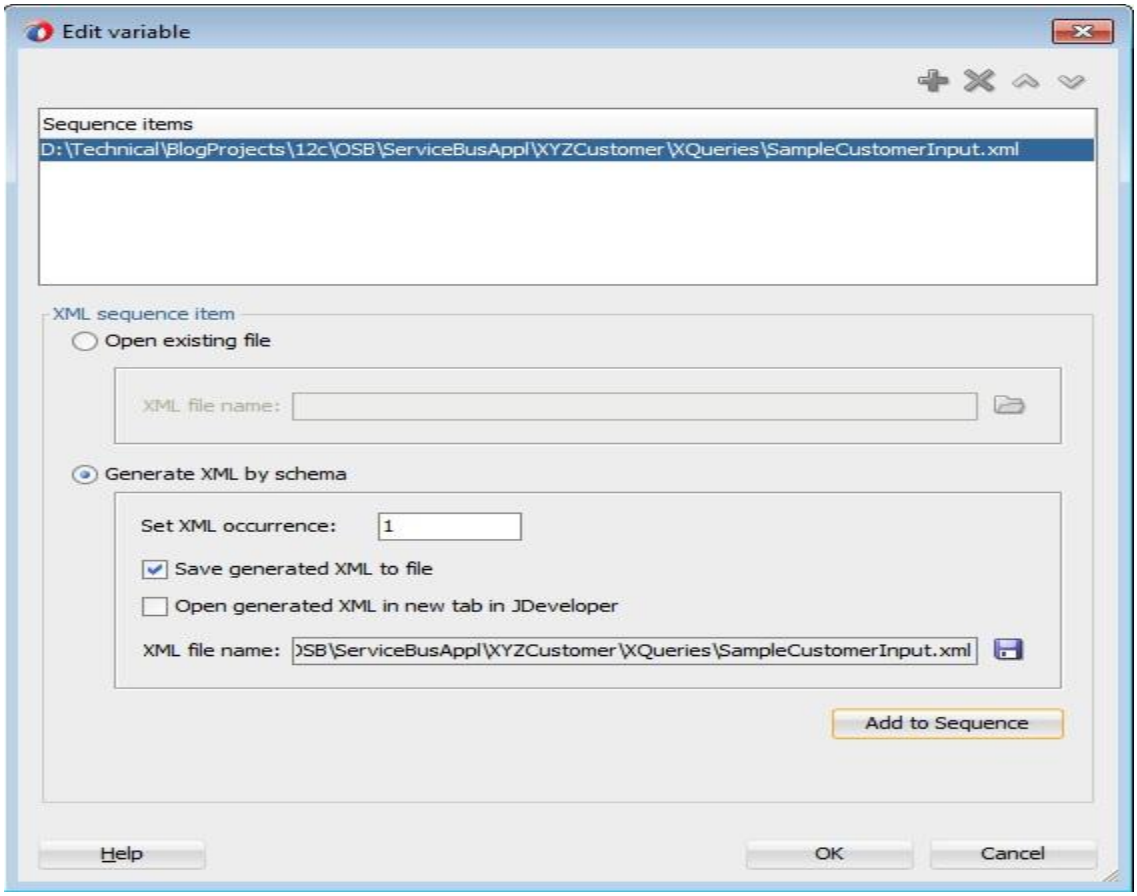




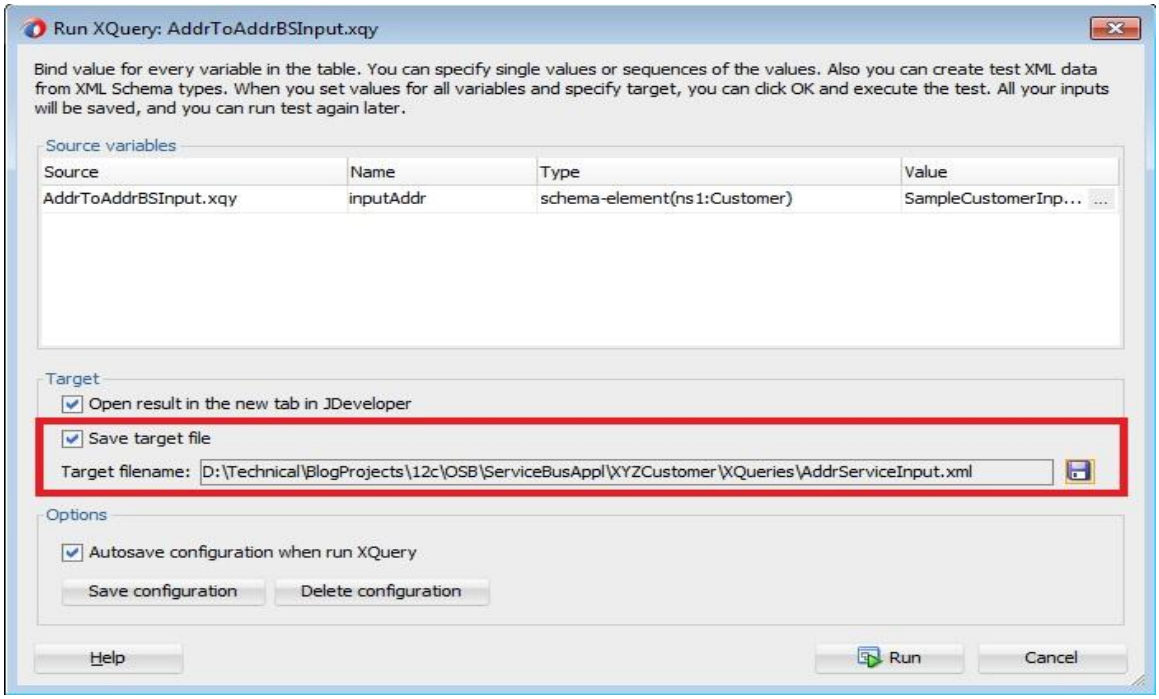
Click **List of variables** button (highlighted above) to bring **Edit Variable** window and select the options as shown below so that sample input will be generated automatically.



Click **save** icon and give file name as **SampleCustomerInput.xml** and click **Add to Sequence.**

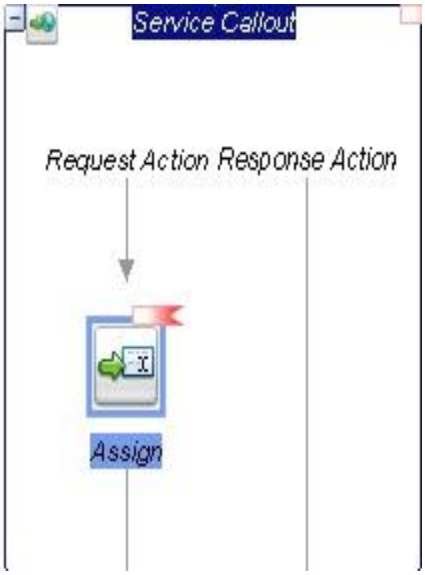


Go back to **Run XQuery** window by clicking **OK**. Now give **Target file name** as **AddrServiceInput** by clicking **Save** icon.

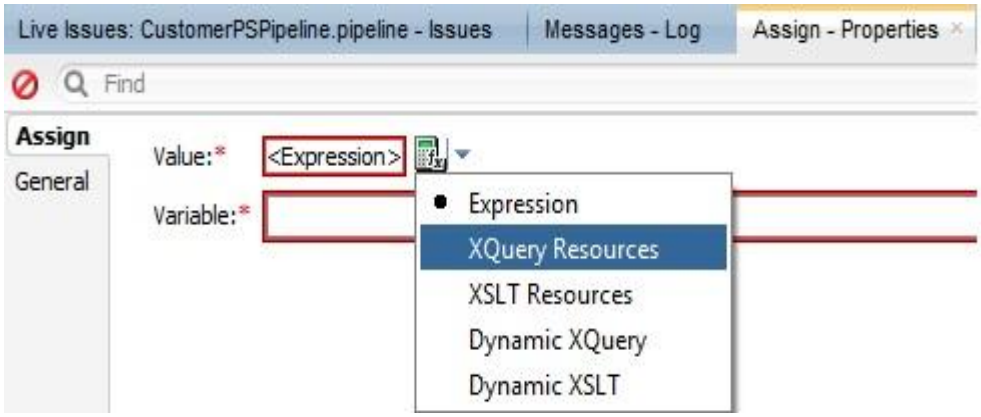


Click **Run** and verify the mapping is working as expected. You can also verify sample source and target files created in your **XQueries** folder.

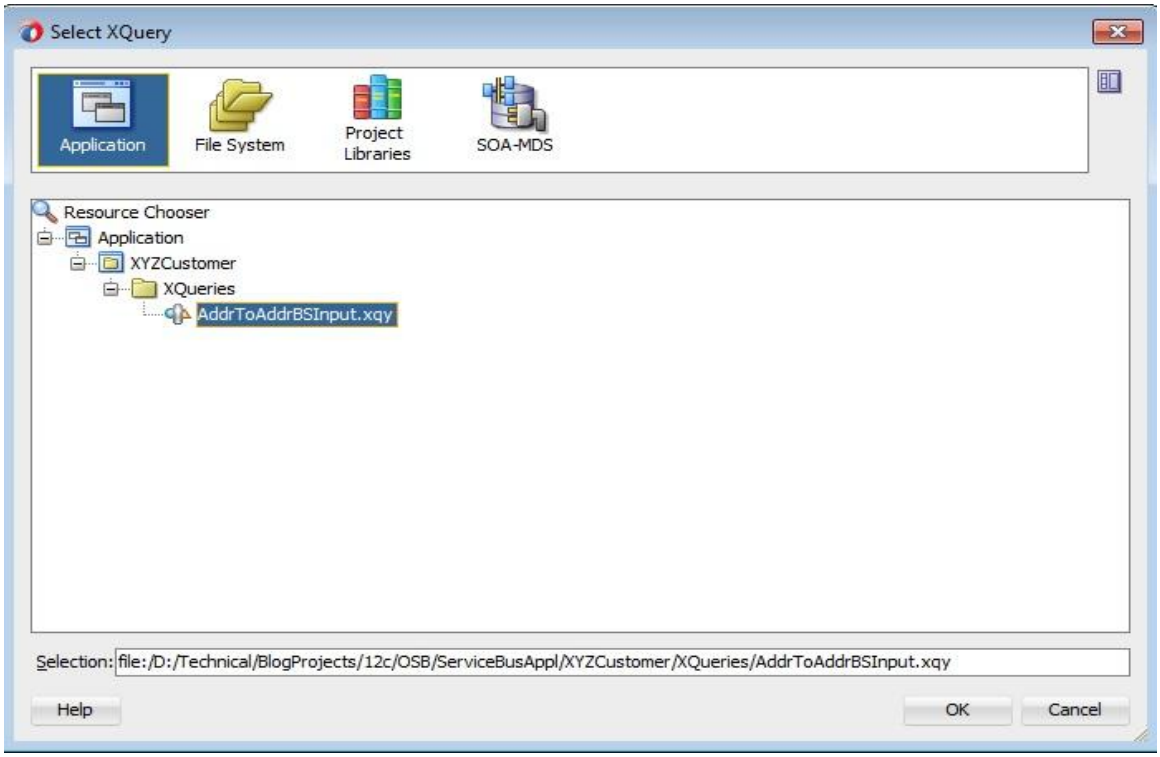
To use this **XQuery** mapping, drag **Assign** activity into **Service Callout** from **Message Processing.**



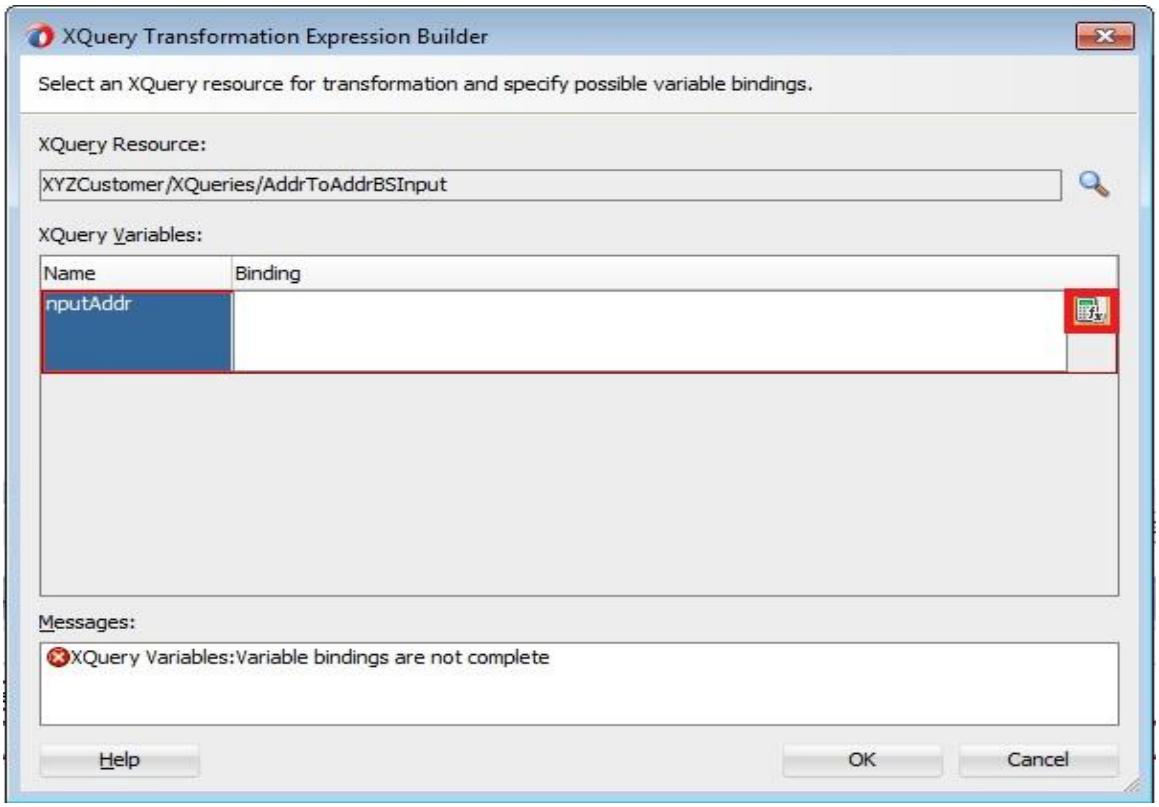
In **Properties** tab, select **XQuery Resources** for **Value** property as shown below.

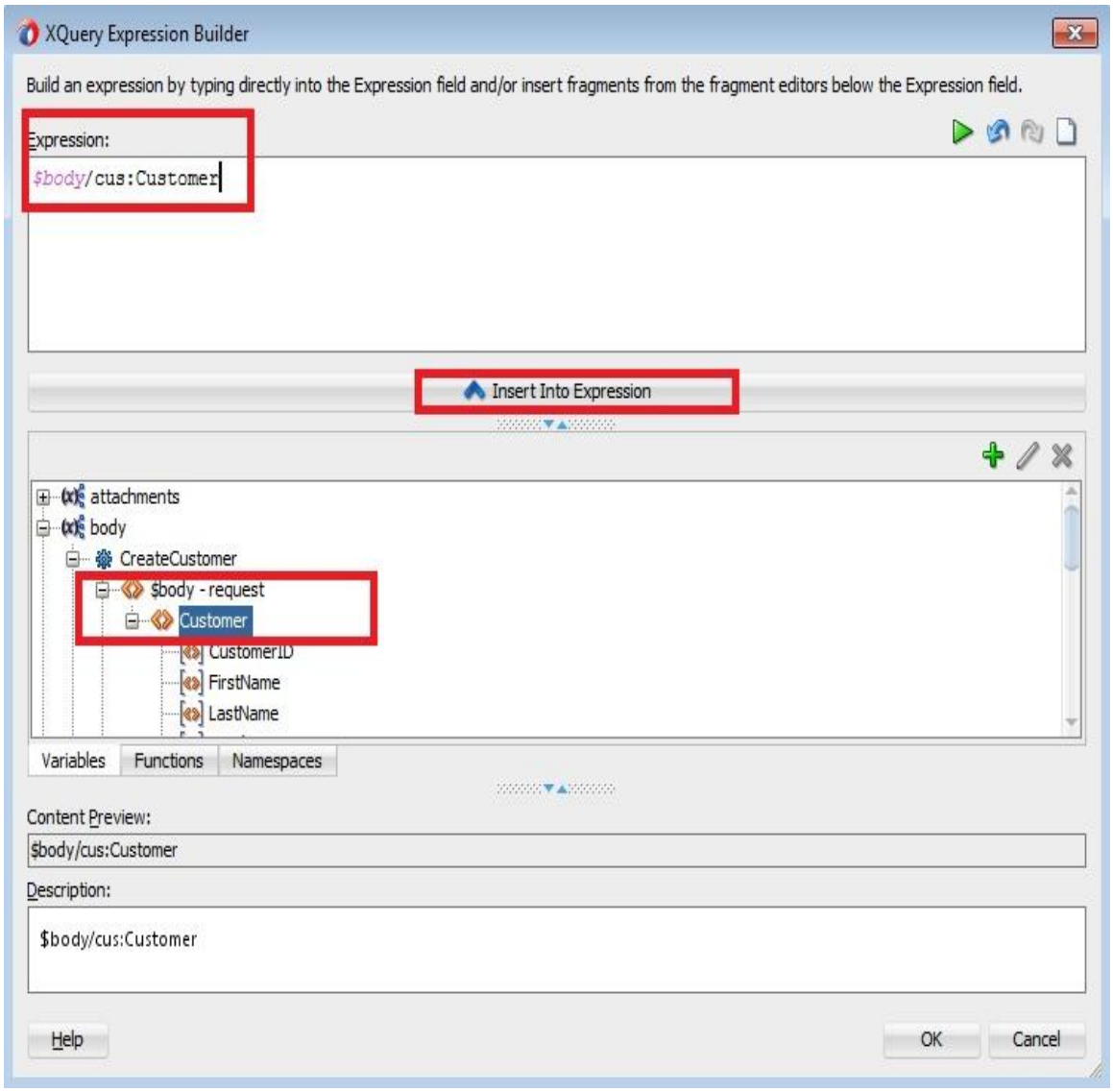


In expression builder, click **Browse** icon and select above **XQuery** mapping.



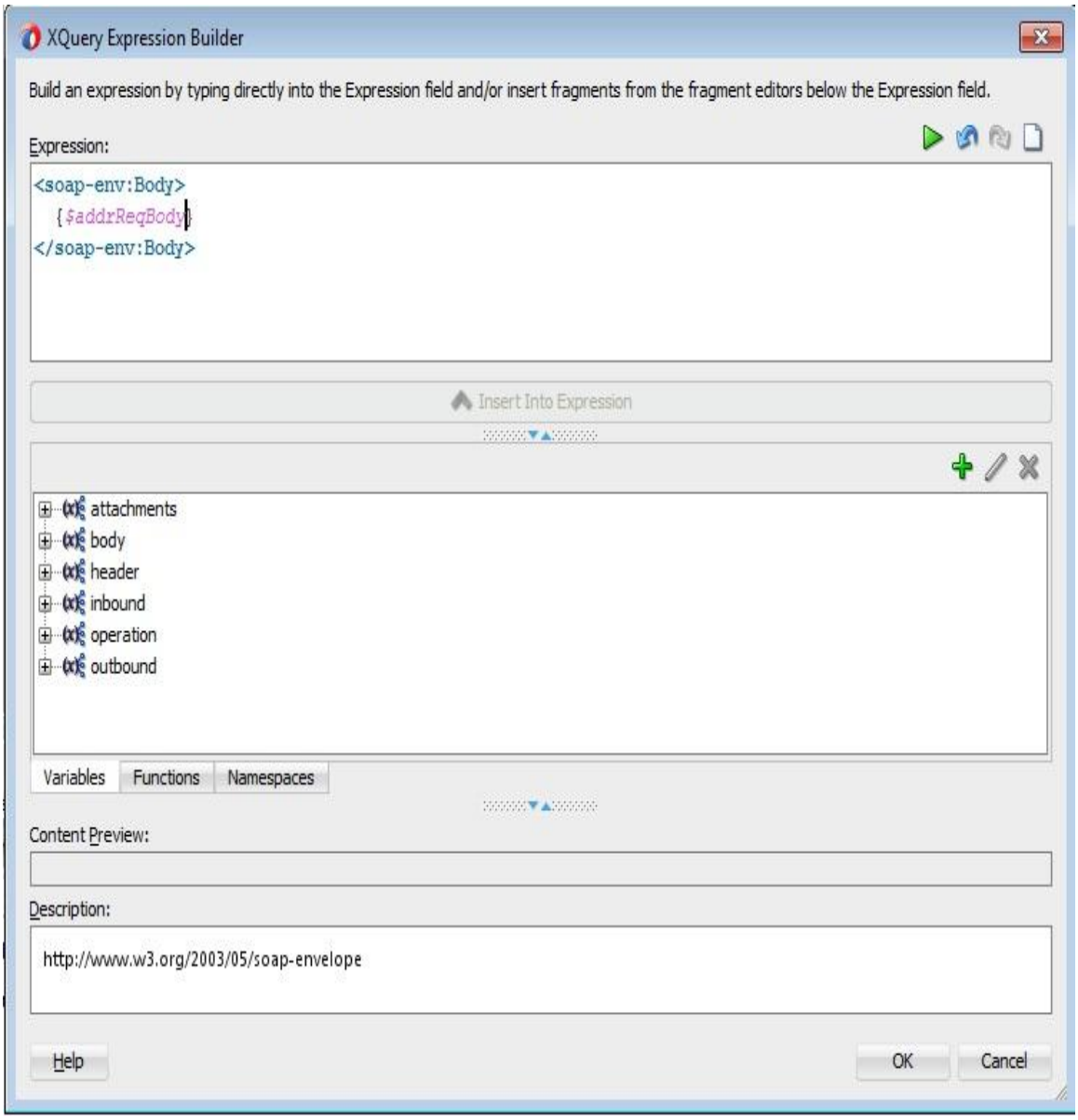
Click **OK** and use following steps to set value for **Binding** property.





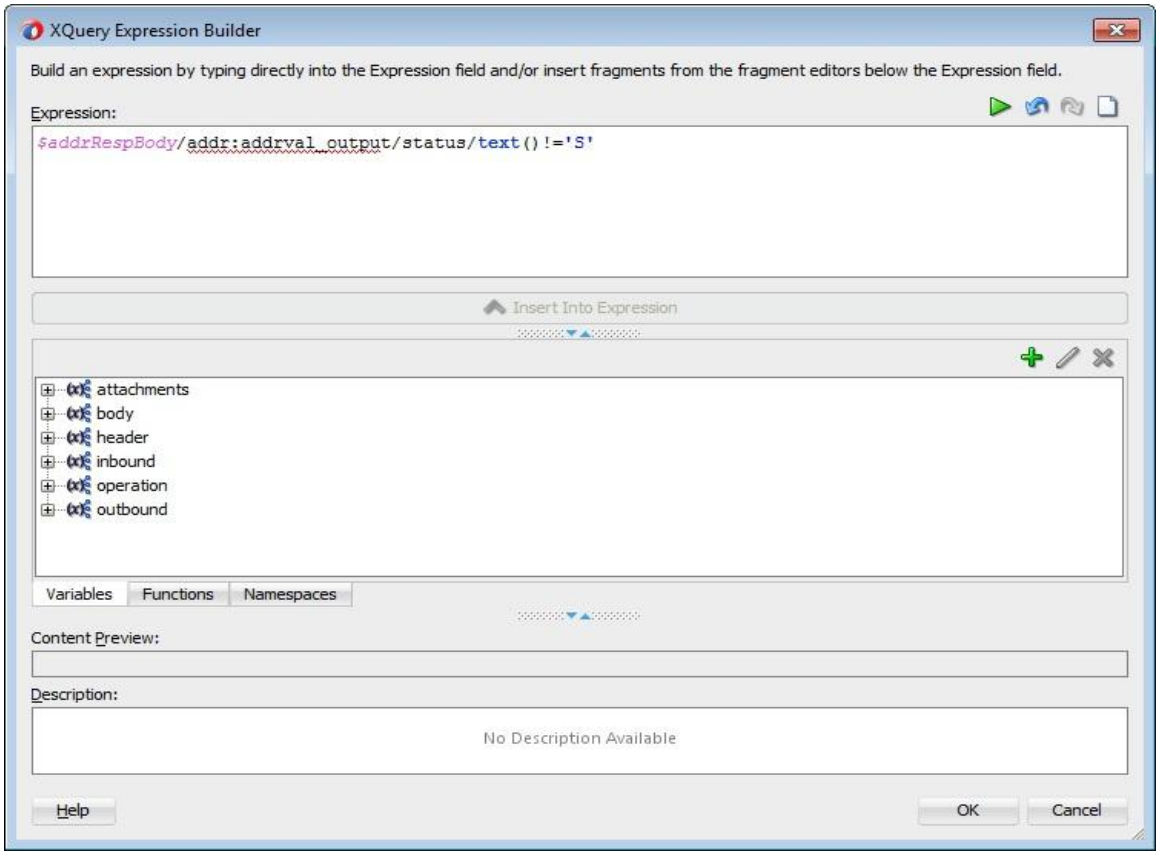
Click **OK** twice and set **addrReqBody** as value for **Variable** property.

Now use another **Assign** activity to enclose the payload in **soap-env:Body** element. Bring up expression builder for **Value** property and give the expression as shown below.

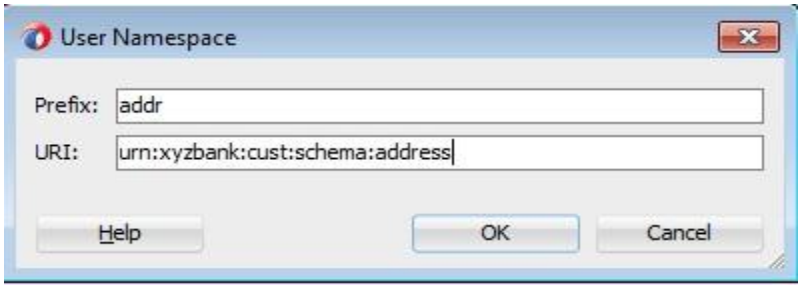


Click **OK** and set **addrReqBody** as value for **Variable** property.

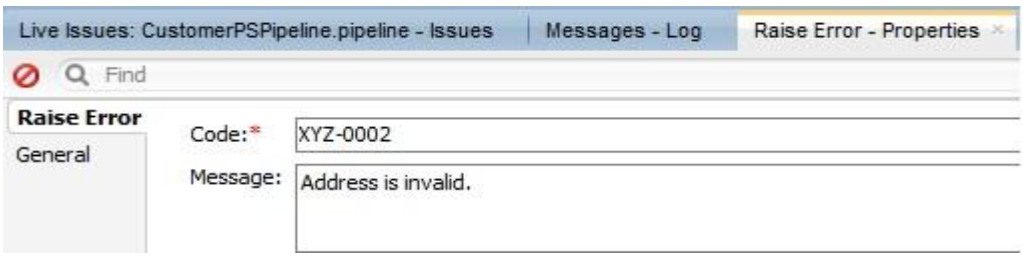
Once response is received, you have to raise error or proceed with message flow depending on status. So drag **If-Then** activity into **AddrValidation** stage after **Service Callout** from **Flow Control** and give condition as shown below.



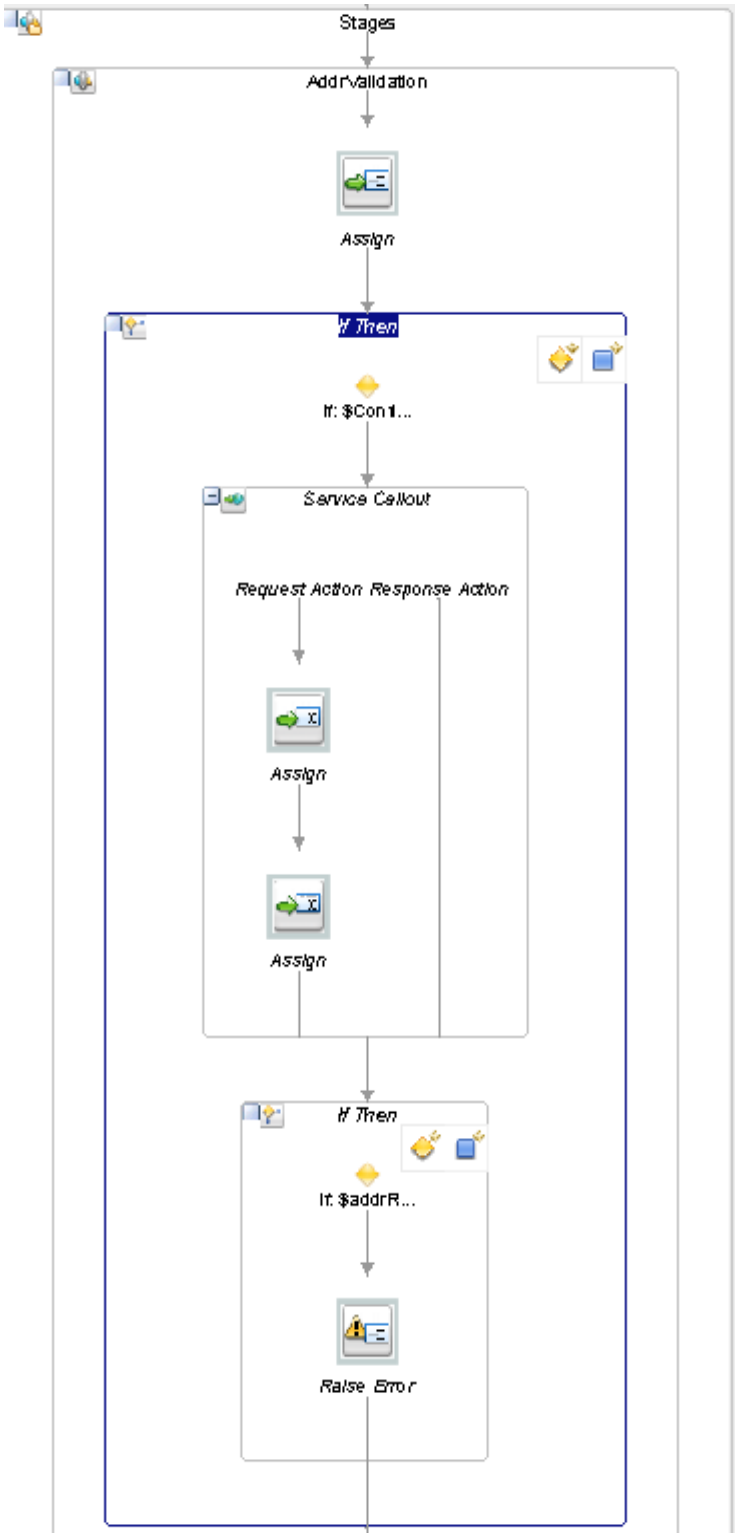
The red line denotes namespace alias **addr** is not a valid one. To register new namespace, go to **Namespaces** tab and click **+** icon and fill in the details as shown below. Here you need to give the namespace used by response schema of address validation service.



Click **OK** and observe red line is no longer visible. Now drag **Raise Error** activity into **If** branch from **Flow Control** and set properties as below.



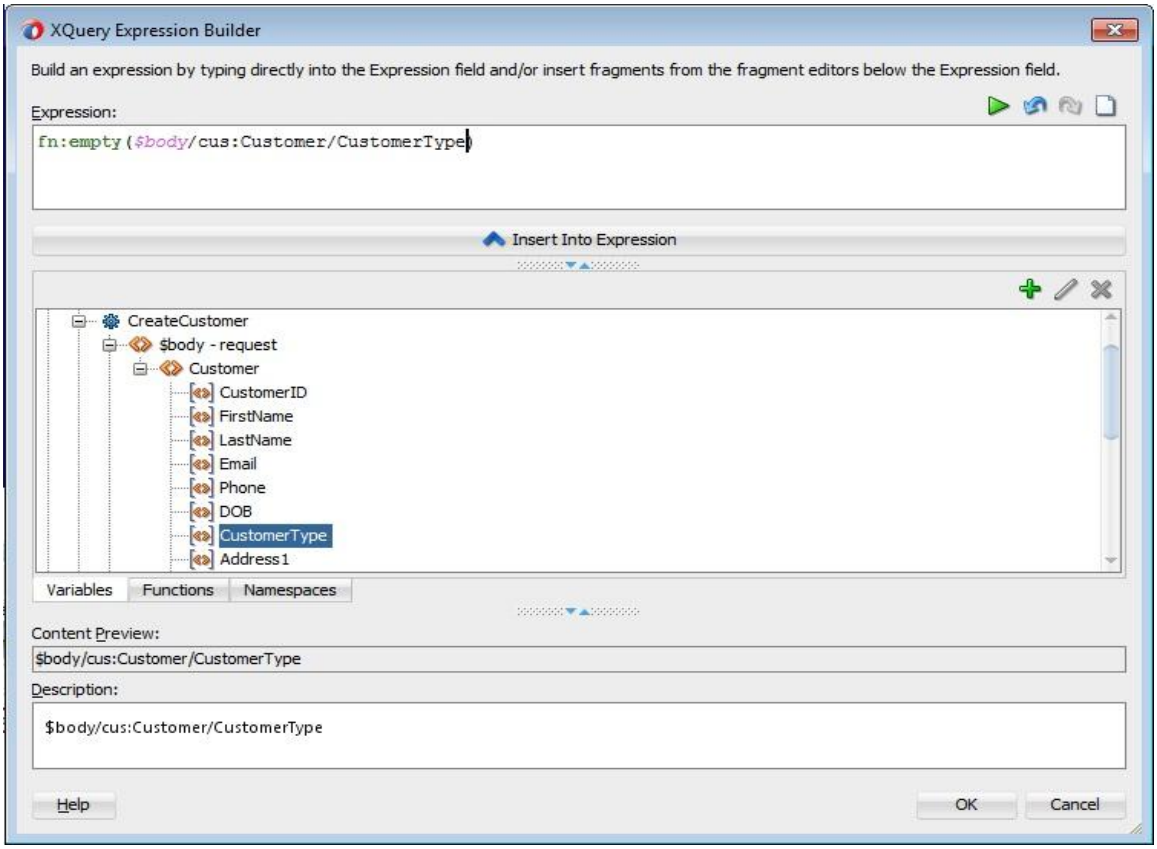
Now your **AddrValidation** stage should look like below.



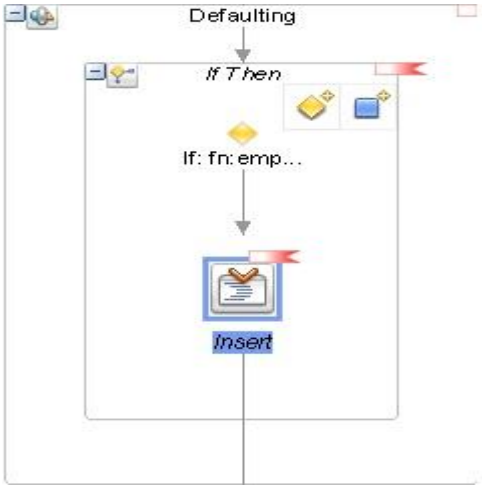
Defaulting

In this section, you will default **CustomerType** element if the value is not sent.

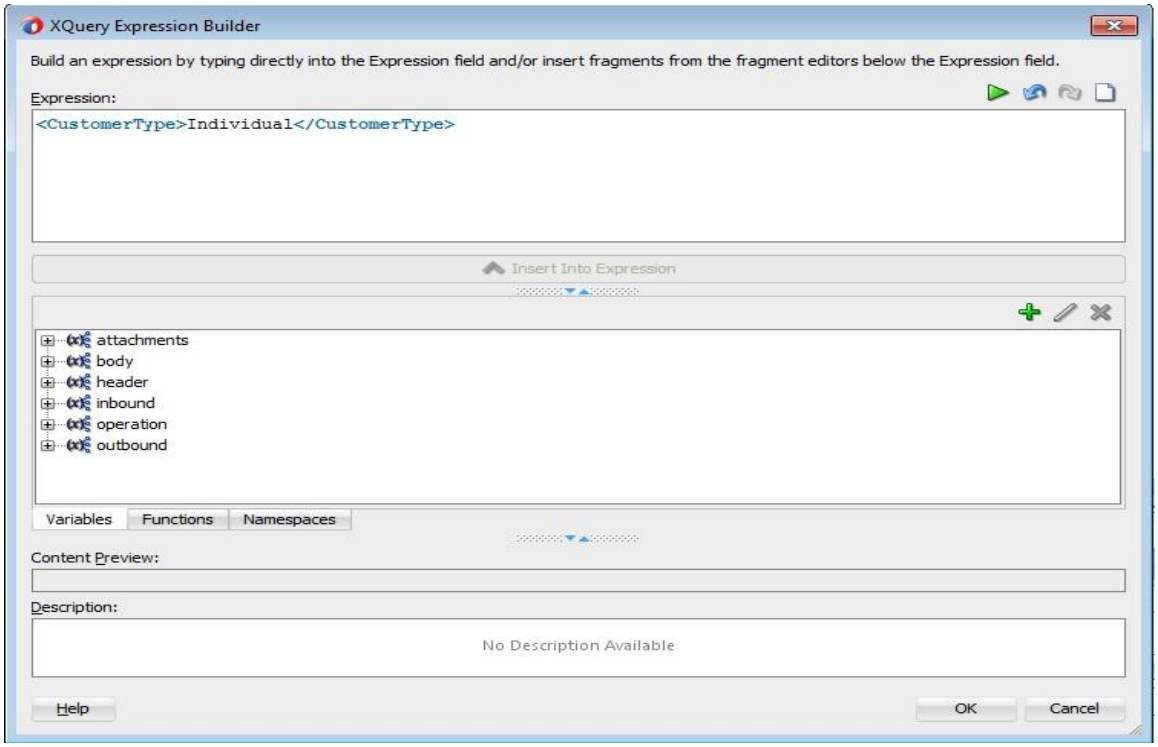
Create another **Stage** after **AddrValidation** stage and set name as **Defaulting** in Properties tab. Drag **If-Then** activity into this stage from **Flow Control** and give condition as shown below. Alternatively, you can select **fn:empty** function from **Functions -> XQuery 1.0 Functions -> Sequences -> General Functions**.



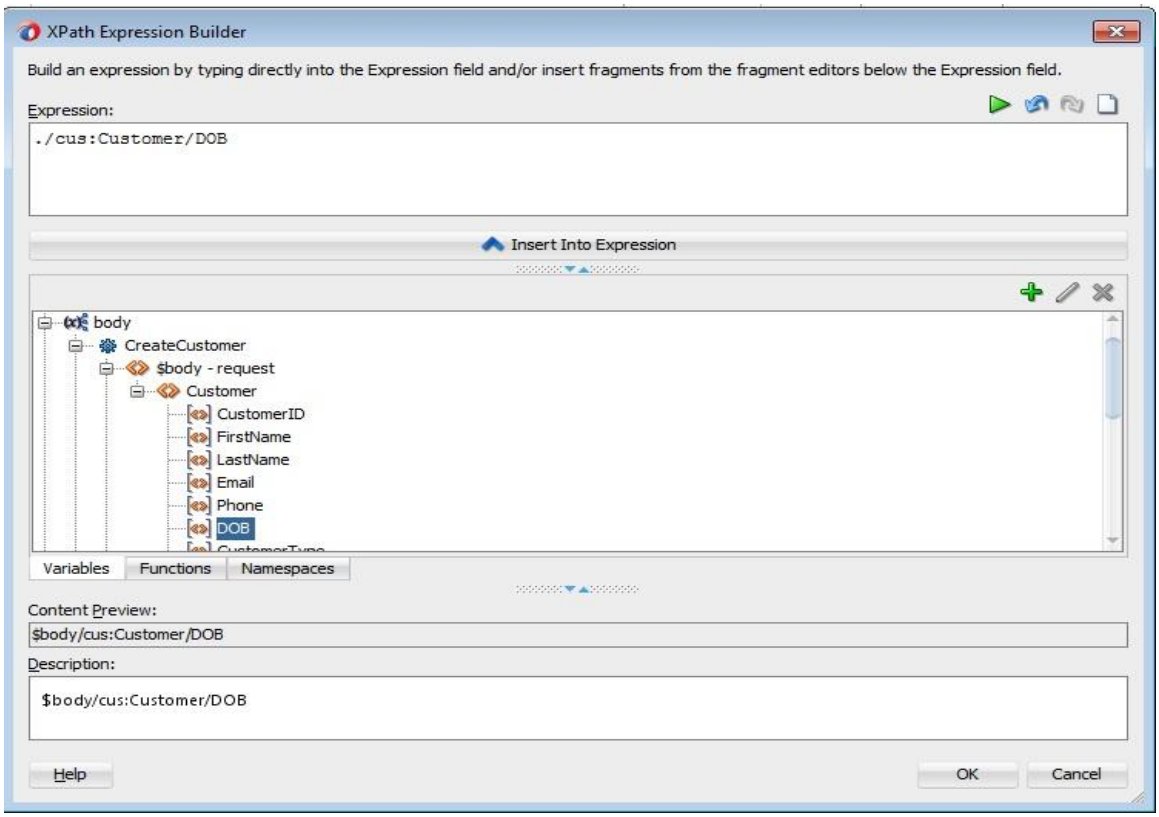
Drag **Insert** activity into **If** branch from **Message Processing.**



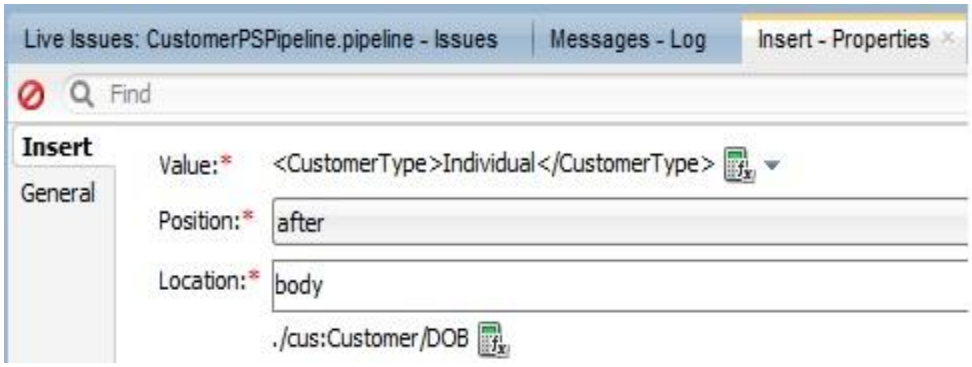
Bring up expression builder for **Value** property and set **expression** as shown below.



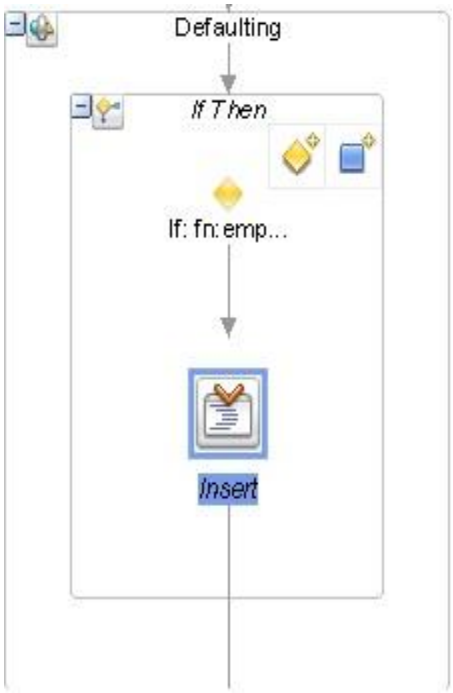
Bring up expression builder for **Location** property and set **expression** as shown below.



Set other properties as shown below so that **CustomerType** element will be inserted in **body** context variable after DOB element in request payload.



Now your **Defaulting** stage should look like below.

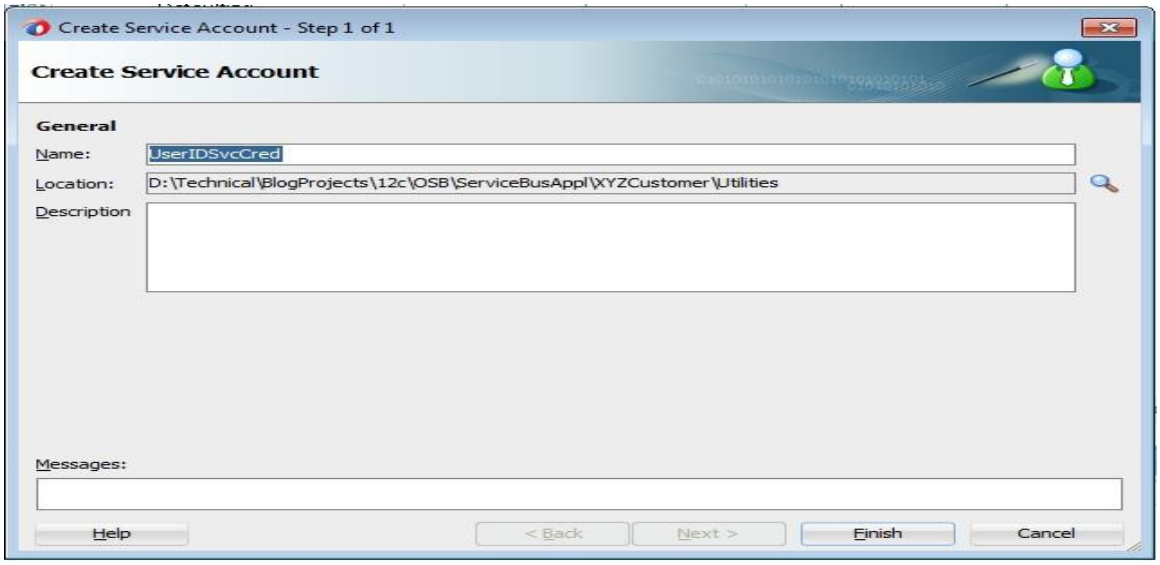


Getting User ID

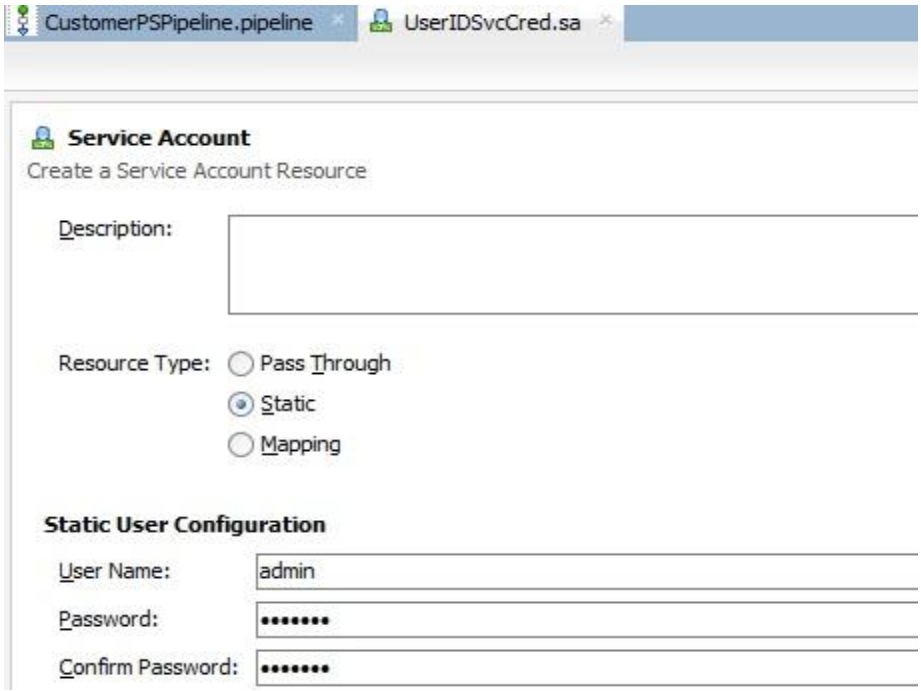
In this section, you will invoke **CreateUserID** service to generate Unique ID for **CustomerID**.

CreateUserID service expects credential information in SOAP header. Instead of hard-coding these values in your message flow, you can use **Service Account** feature provided by Service Bus to store credentials which can be referenced in message flow using XQuery function.

Create a new folder named **Utilities** in **XYZCustomer** project. Right click this folder and select **New->Service Account.** Give name as **UserIDSvcCred** and click **Finish**.

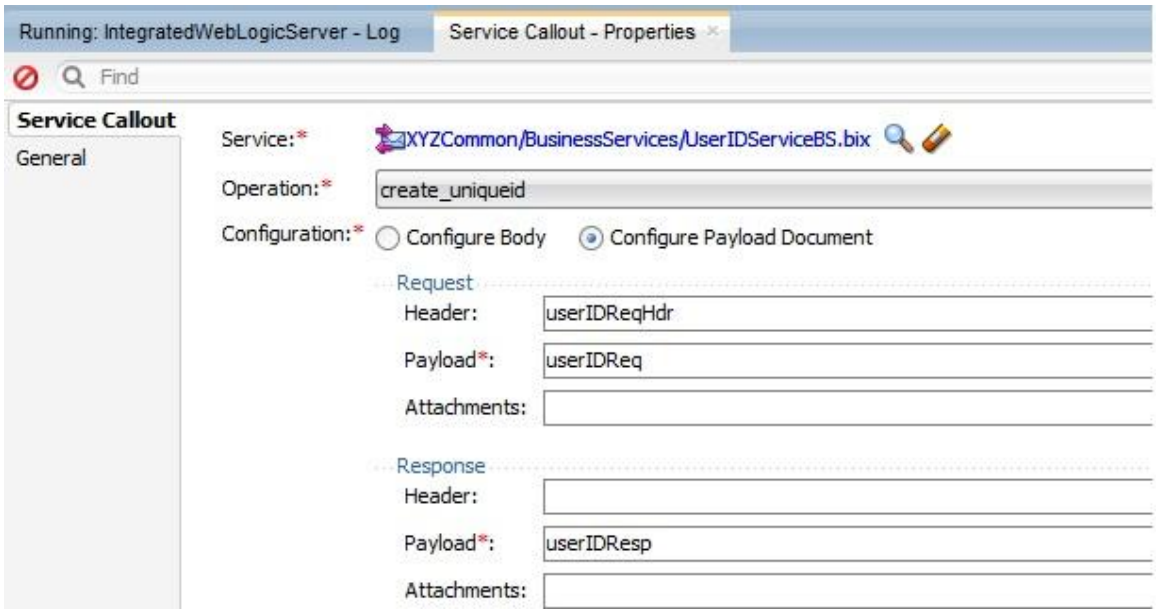


Select **Resource Type** as **Static** and give your username and password as shown below.



Drag a **Service Callout** into **Defaulting** stage after **If-Then** activity from **Communication**. And in **Properties** tab, browse and select **UserIDServiceBS** and select **create_uniqueid** as operation.

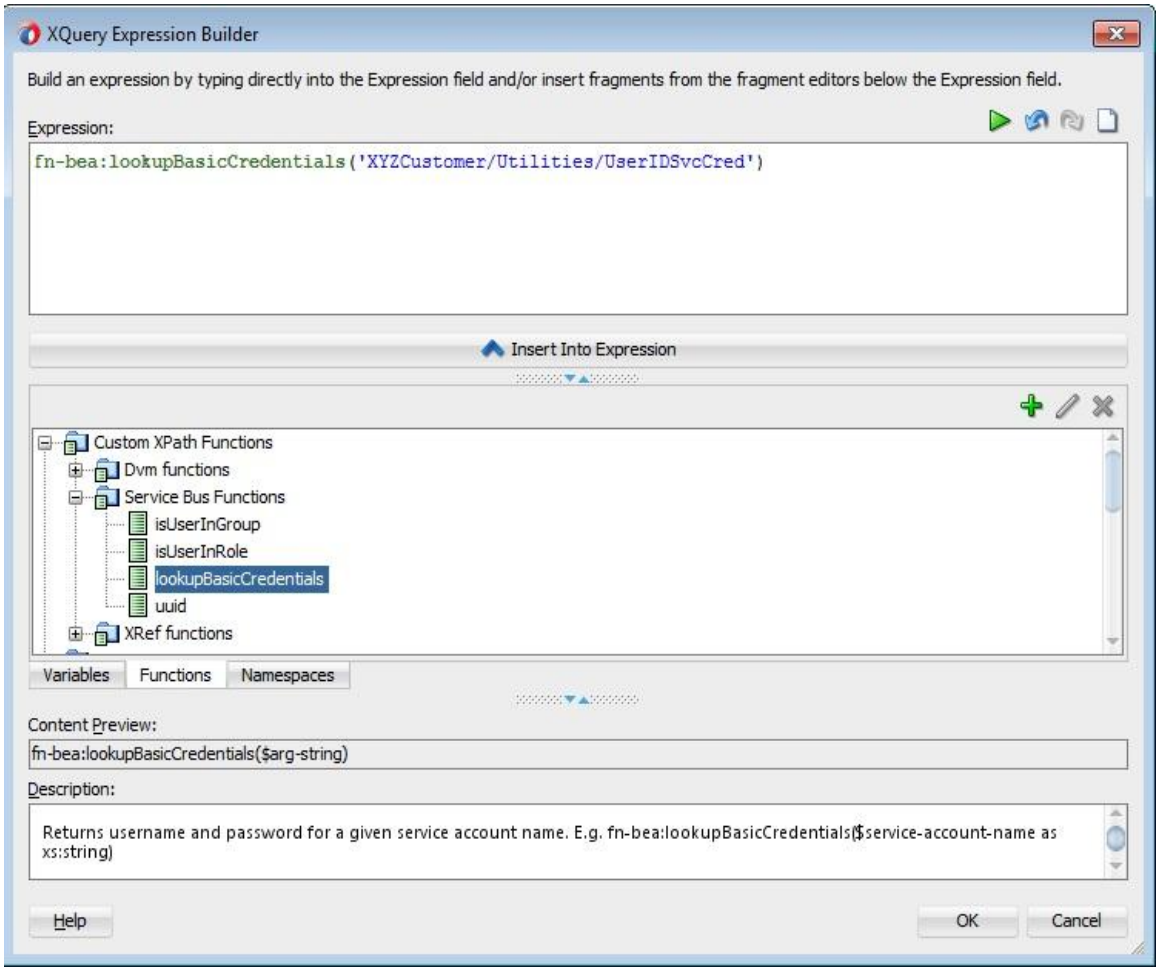
Set other properties as shown below including **Request Header** property as you need to send credential information in SOAP Header. Choosing the option '**Configure Payload Document**' allows you to send request and receive response payload without **soap-env:Body** tag. In this case, Service Bus will add SOAP Body element at runtime while calling business service.

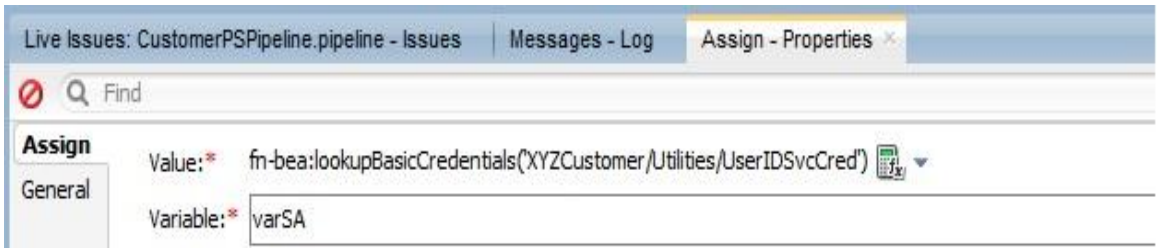


The XQuery function **fn-bea:lookupBasicCredentials()** can be used to get credentials stored in your service account. This function returns following XML fragment.

```
<con:UsernamePasswordCredential xmlns:con="http://www.bea.com/wli/sb/services/security/config">
        <con:username>Username value given in service account</con:username>
        <con:password>Password value given in service account</con:password>
</con:UsernamePasswordCredential>
```

Drag **Assign** activity into **Request Action** of **Service Callout** from **Message Processing** and set properties as shown below. Alternatively, you can select this **XQuery** function by navigating to **Functions -> Custom XPath Functions -> Service Bus Functions**.





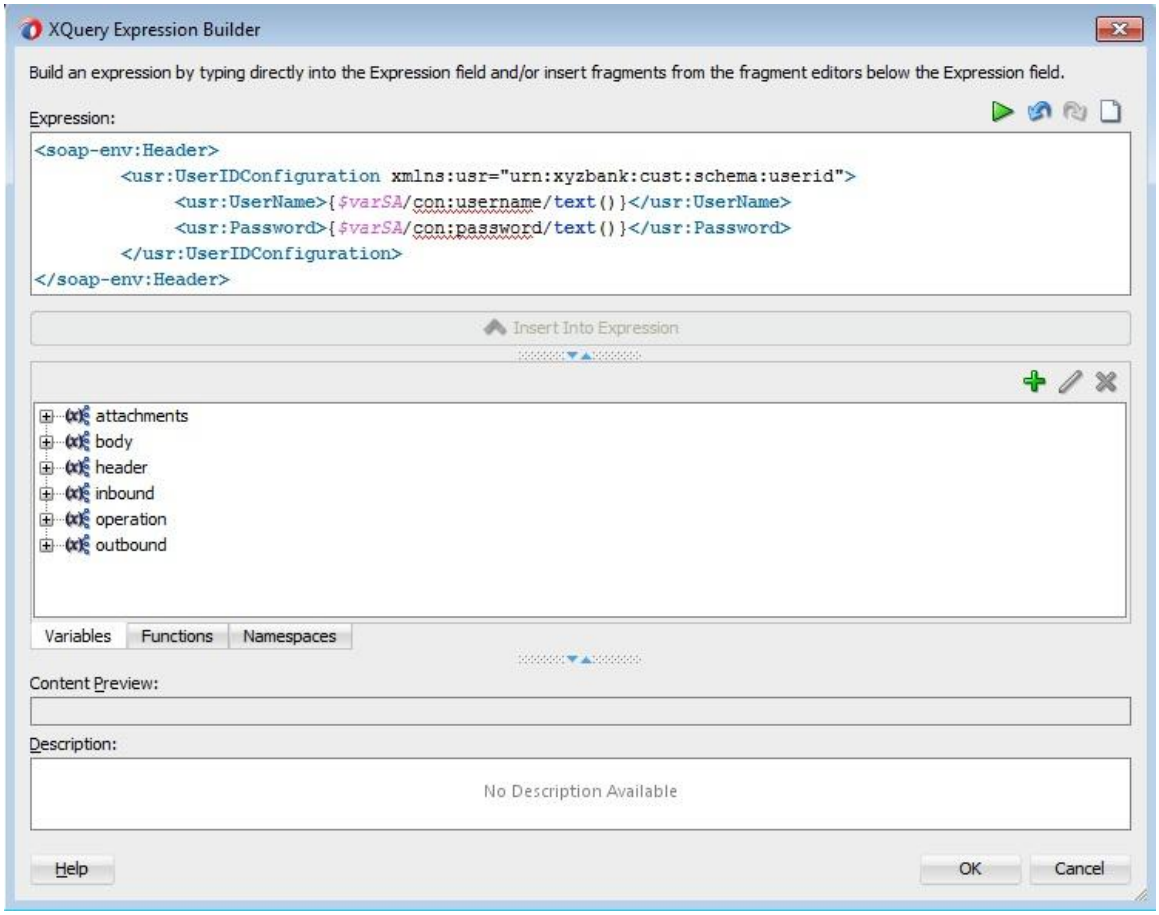
Drag **Assign** activity and set properties as given below populating the SOAP header. Note the inclusion of **<soap-env:Header>** tag in expression and has to be included always irrespective of the **Configure** option selected for **Service Callout**.

Value:

```
<soap-env:Header>
        <usr:UserIDConfiguration xmlns:usr="urn:xyzbank:cust:schema:userid">
           <usr:UserName>{$varSA/con:username/text()}</usr:UserName>
           <usr:Password>{$varSA/con:password/text()}</usr:Password>
        </usr:UserIDConfiguration>
</soap-env:Header>
```

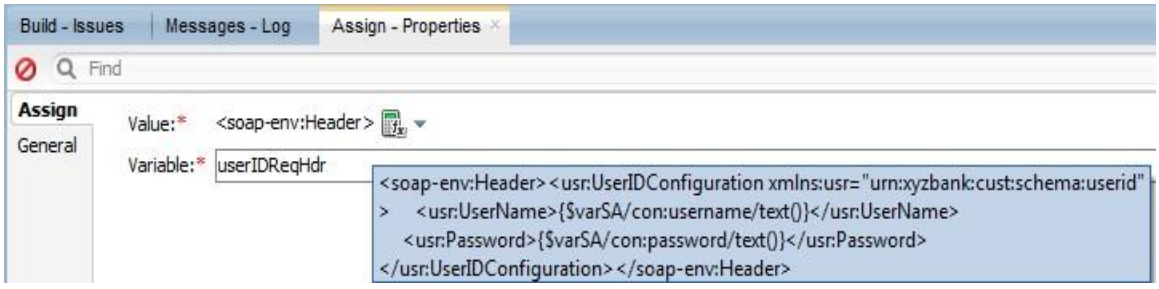
Variable:

userIDReqHdr



If you observe red lines, register new namespace **http://www.bea.com/wli/sb/services/security/config** with alias **con** as shown below.

Now your properties window should be shown like below.



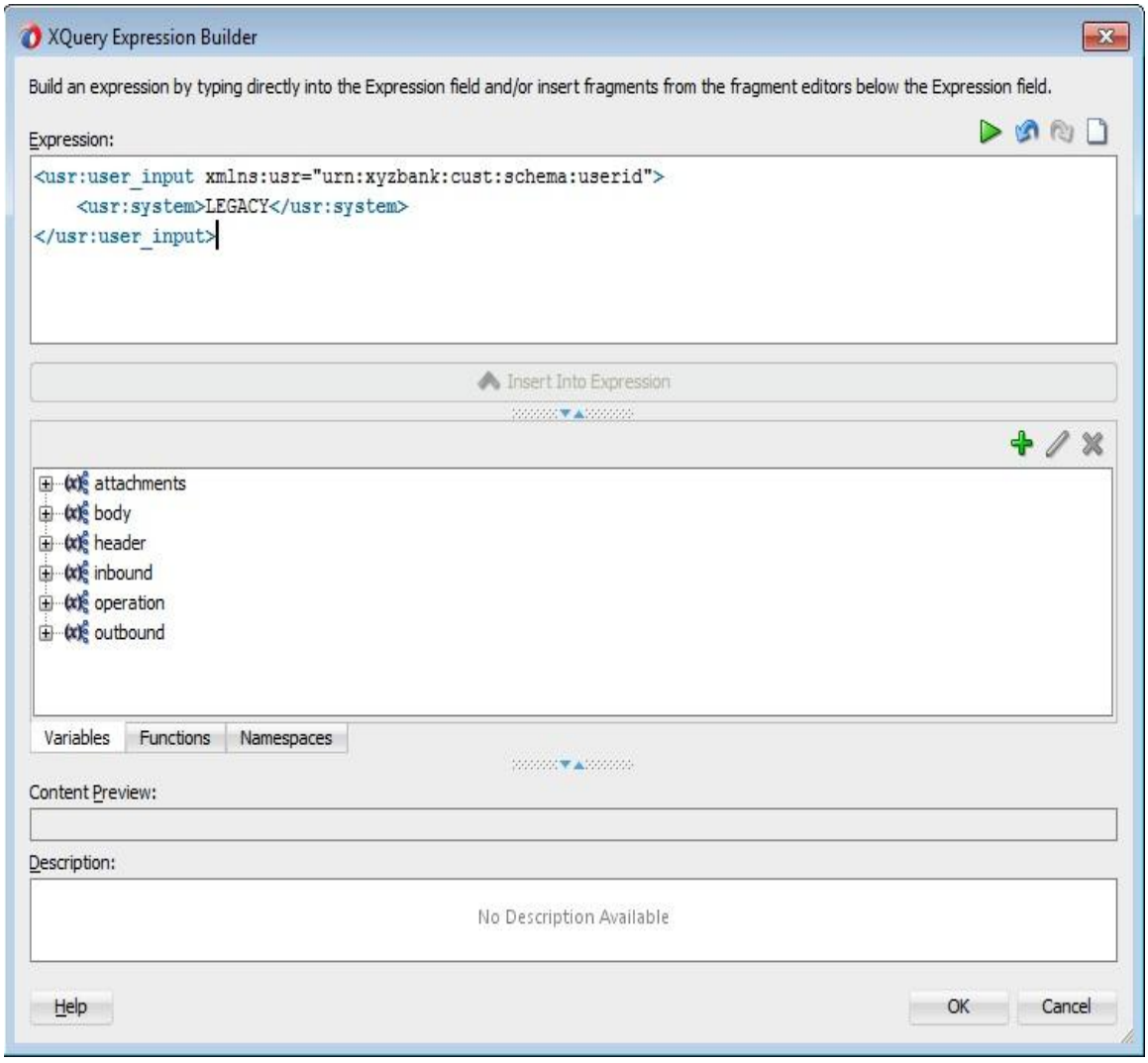
Drag another **Assign** activity and set properties as given below to populate payload for business service.

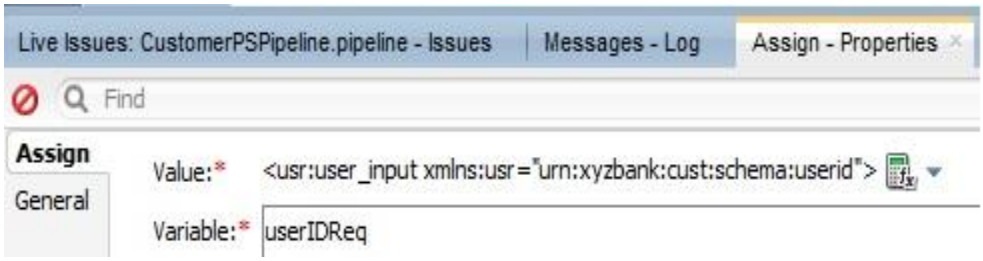
Expression:

```
<usr:user_input xmlns:usr="urn:xyzbank:cust:schema:userid">
  <usr:system>LEGACY</usr:system>
</usr:user_input>
```

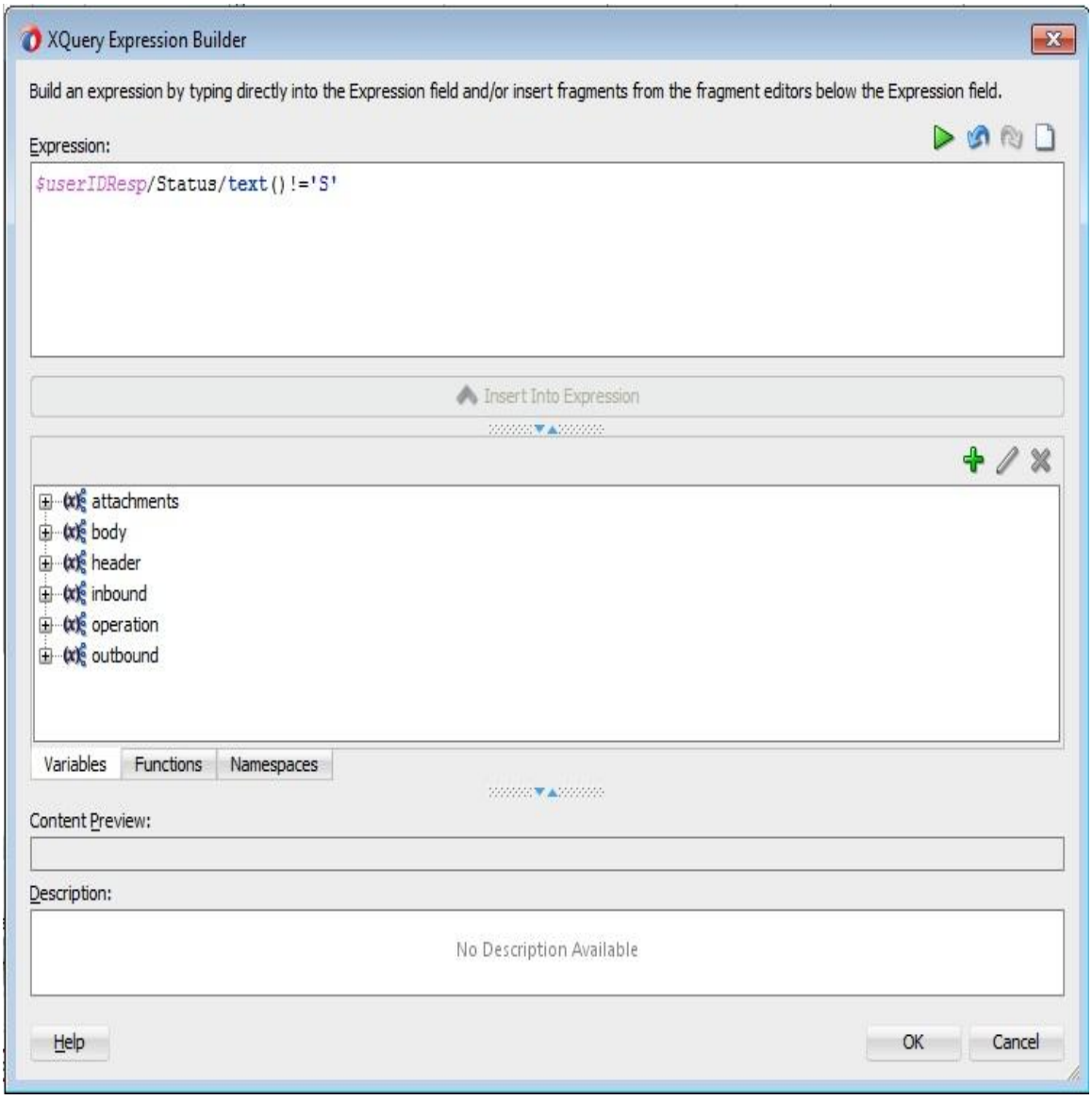
Variable:

userIDReq

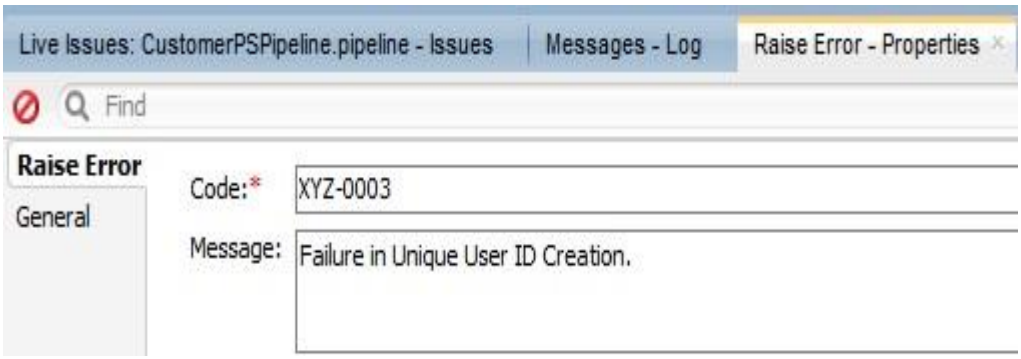




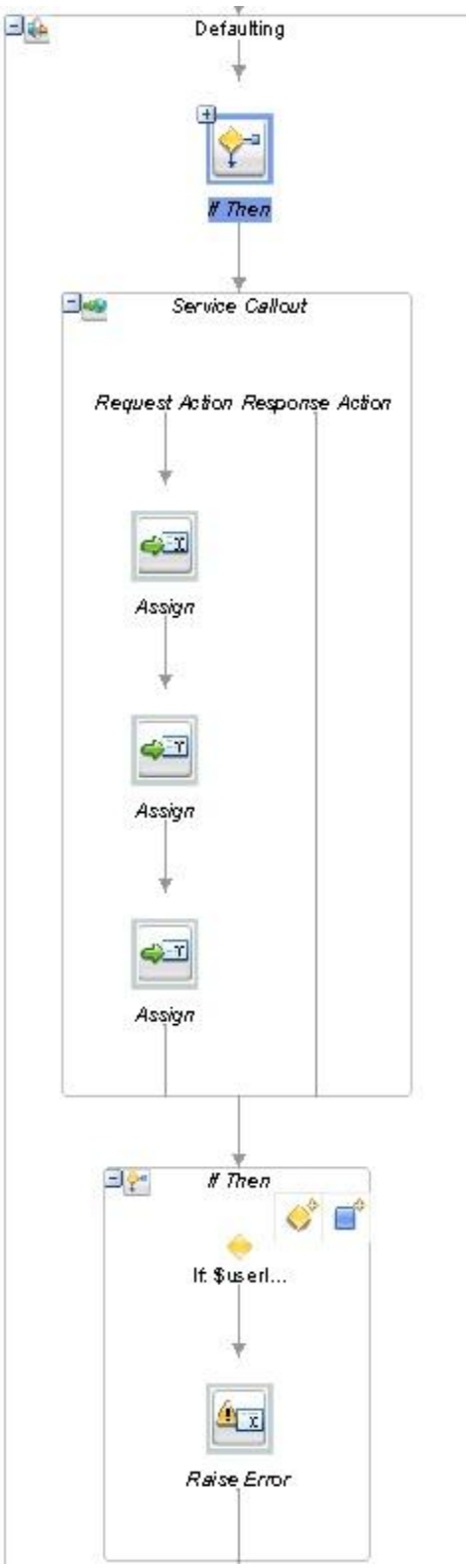
Drag **If-Then** activity after **Service Callout** from **Flow Control** and set condition as shown below to verify status of **UserID** Creation.



Drag **RaiseError** activity into **If** branch from **Flow Control** and set properties as shown below.



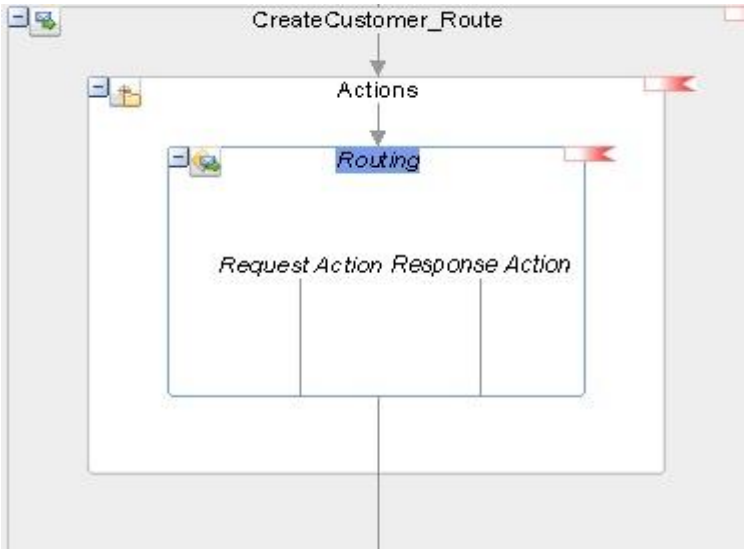
Now your **Defaulting** stage should look like below.



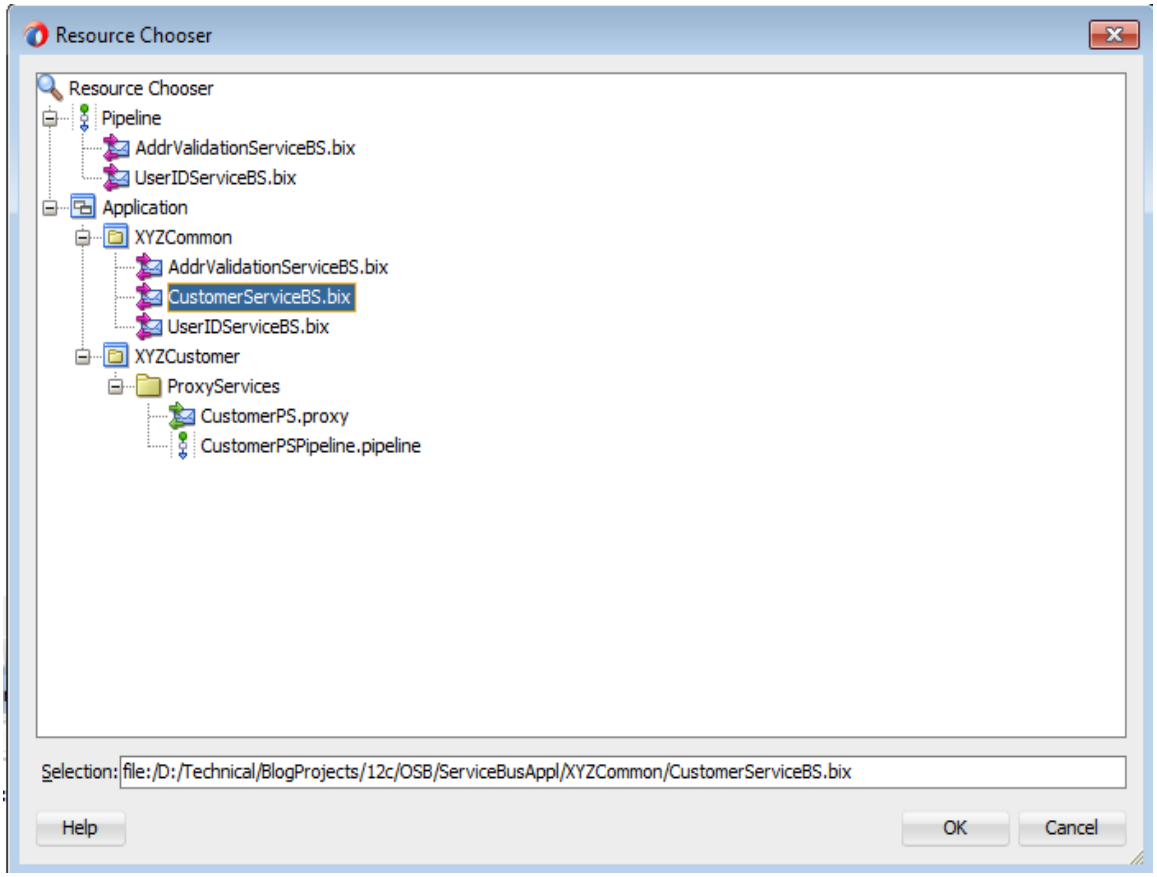
Routing

In this section, you will invoke **CustomerServiceBS** to finish message flow for **Create Customer.**

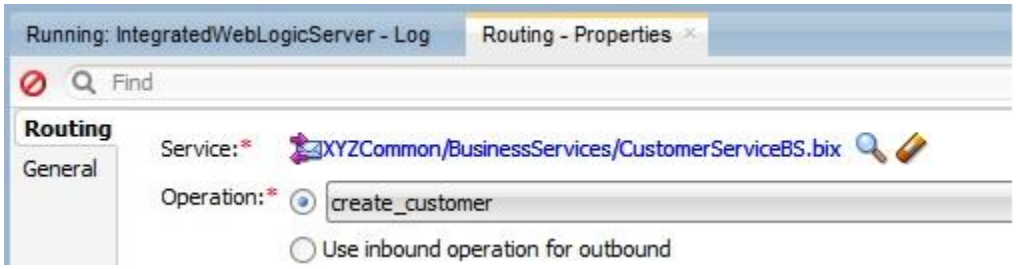
Drag **Routing** activity into **Actions** placeholder of **CreateCustomer_Route.**



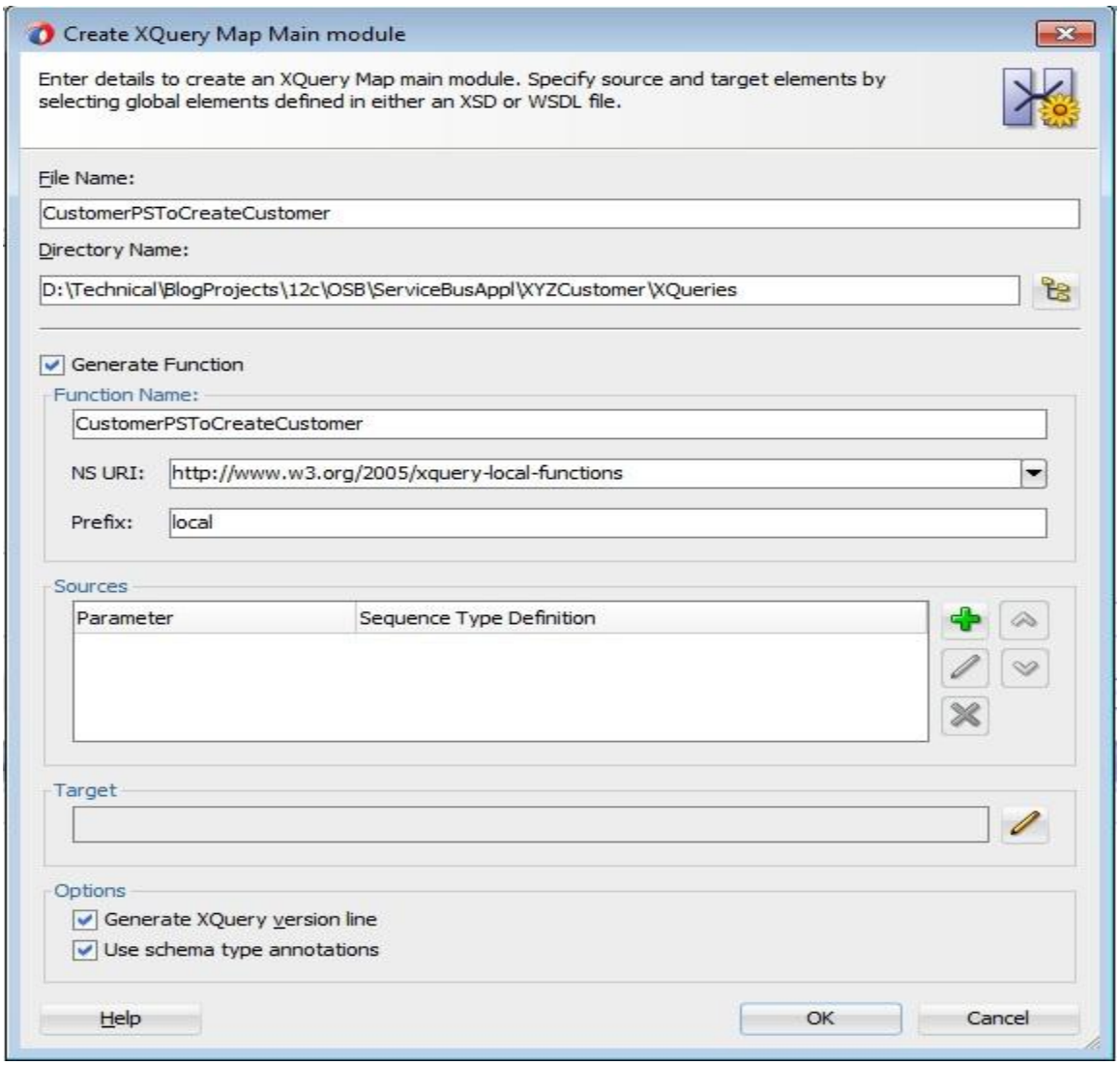
In **Properties** tab, click browse and select **CustomerServiceBS** as shown below.



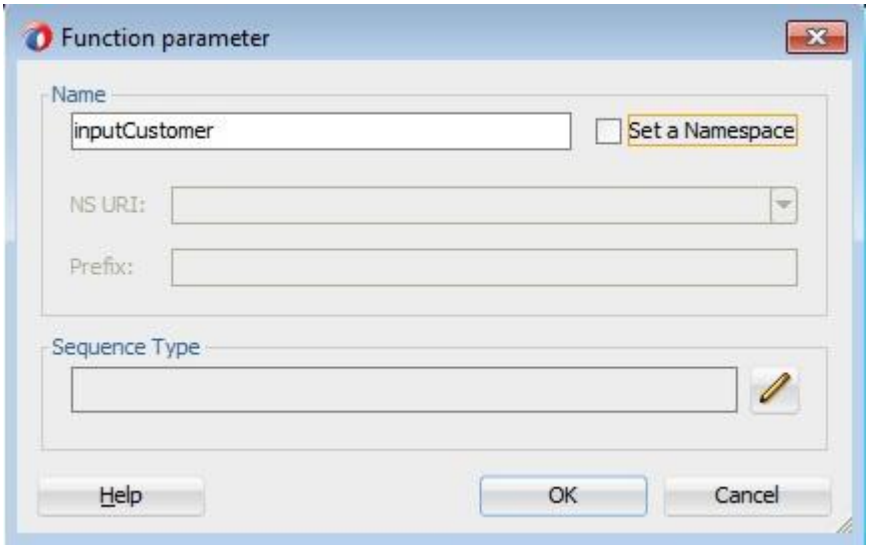
Select **create_customer** as **Operation**.

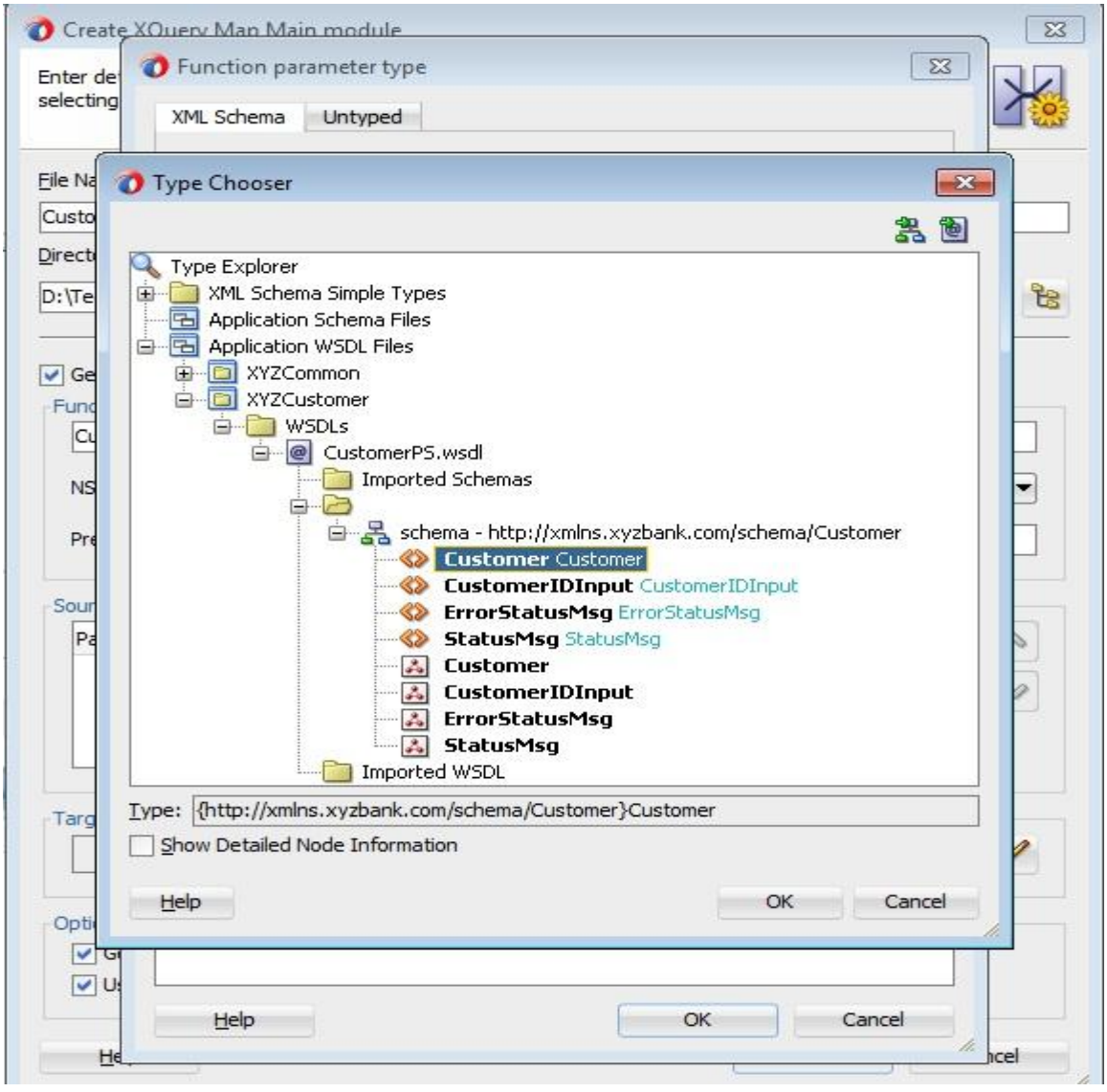


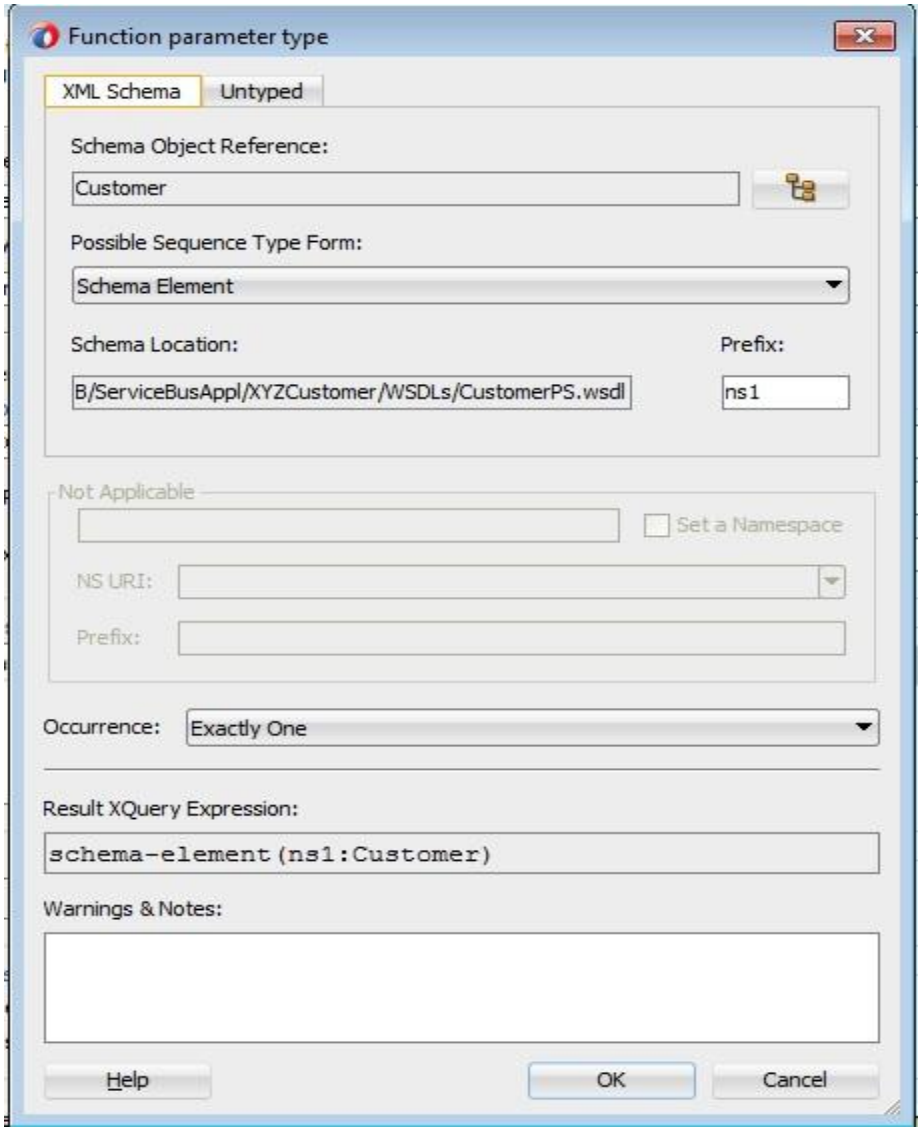
Next task is transforming the received payload to the structure expected by **CustomerService** business service. So create a new XQuery map in **XQueries** folder of **XYZCustomer** project and give both **File Name** and **Function Name** as **CustomerPSToCreateCustomer**.

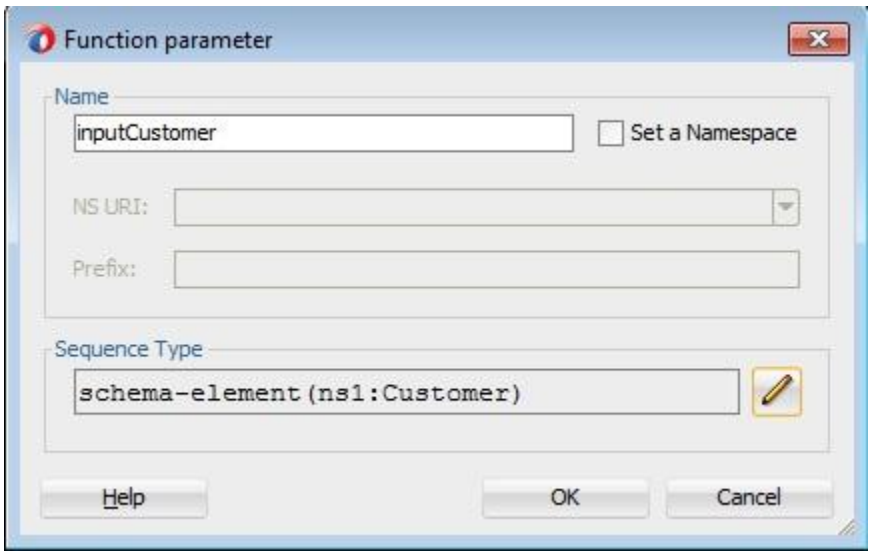


Complete **Source** parameter selection as shown below and you can refer to previous sections for actual navigation steps.

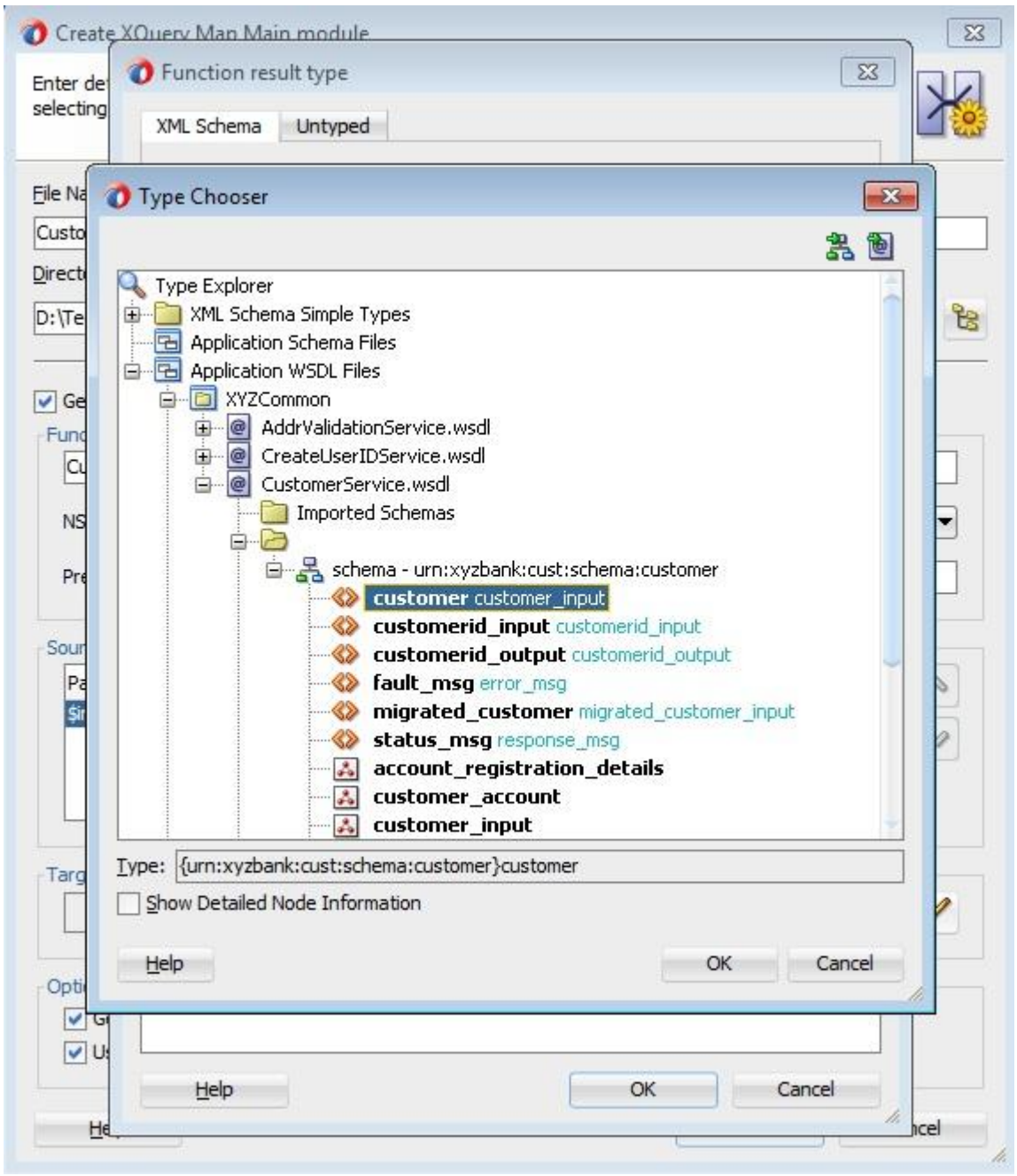

Function parameter
Name
inputCustomer
Set a Namespace
NS URI:
Prefix:
Sequence Type
Help
OK
Cancel

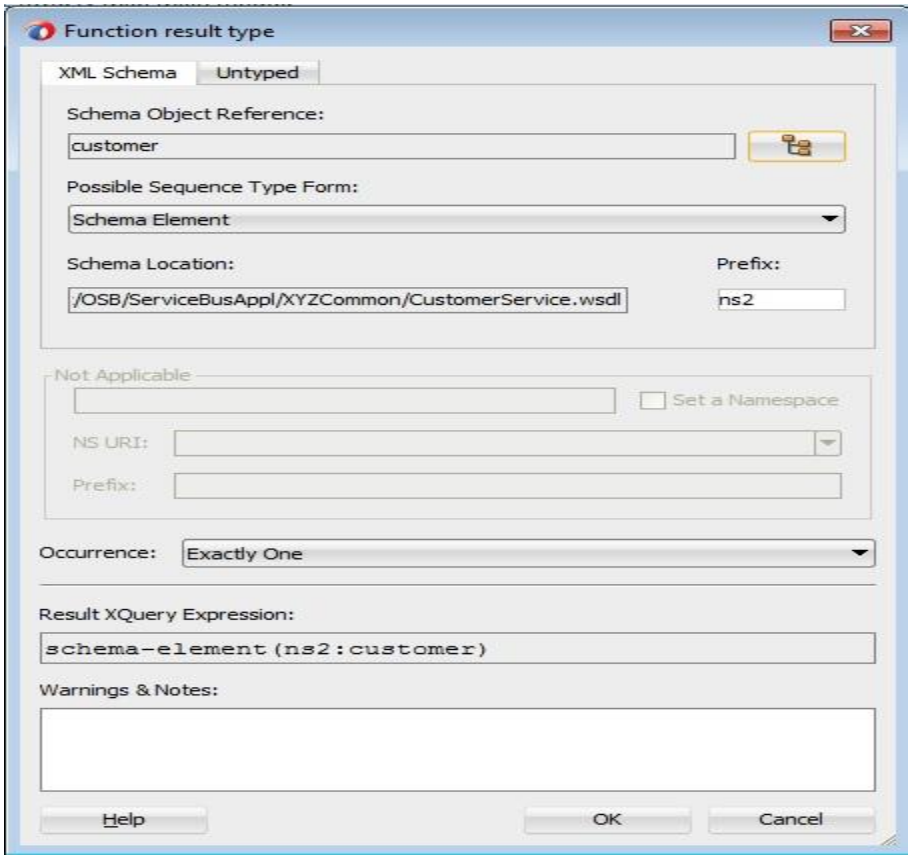

Create XQuery Map Main module
Function parameter type
XML Schema
Untyped
Type Chooser
Type Explorer
XML Schema Simple Types
Application Schema Files
Application WSDL Files
XYZCommon
XYZCustomer
WSDLs
CustomerPS.wsdl
Imported Schemas
schema - http://xmlns.xyzbank.com/schema/Customer
Customer Customer
CustomerIDInput CustomerIDInput
ErrorStatusMsg ErrorStatusMsg
StatusMsg StatusMsg
Customer
CustomerIDInput
ErrorStatusMsg
StatusMsg
Imported WSDL
Type: {http://xmlns.xyzbank.com/schema/Customer}Customer
Show Detailed Node Information
Help
OK
Cancel
Help
OK
Cancel

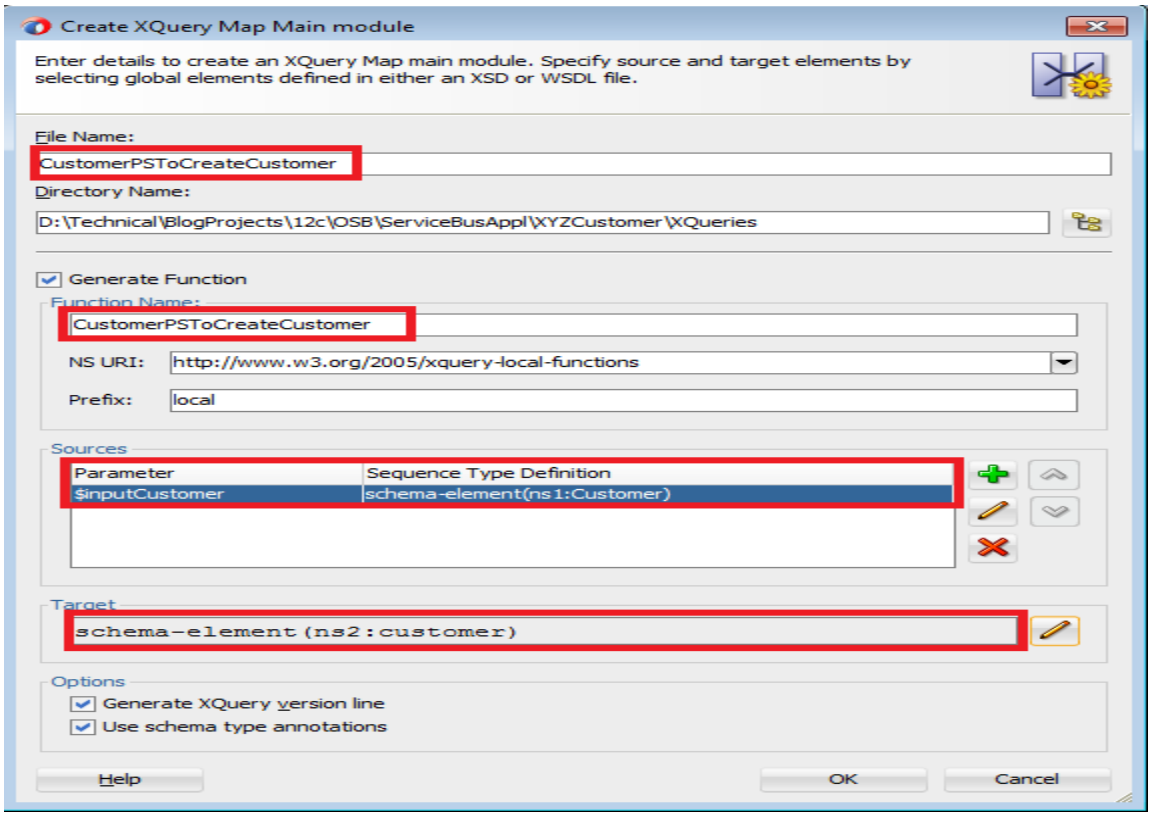

Function parameter type
XML Schema
Untyped
Schema Object Reference:
Customer
Possible Sequence Type Form:
Schema Element
Schema Location:
Prefix:
B/ServiceBusAppl/XYZCustomer/WSDLs/CustomerPS.wsdl
ns1
Not Applicable
Set a Namespace
NS URI:
Prefix:
Occurrence:
Exactly One
Result XQuery Expression:
schema-element(ns1:Customer)
Warnings & Notes:
Help
OK
Cancel


Function parameter
Name
inputCustomer
Set a Namespace
NS URI:
Prefix:
Sequence Type
schema-element(ns1:Customer)
Help
OK
Cancel

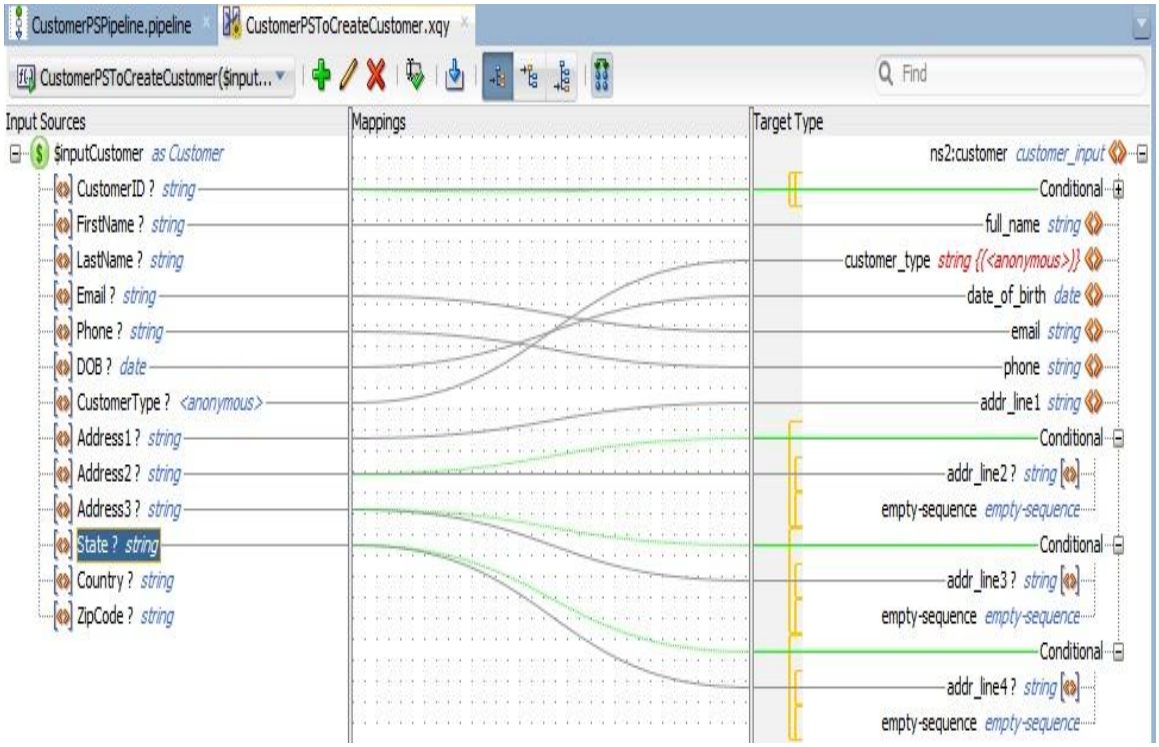
Complete **Target** selection as shown below and you can refer to previous sections for actual navigation steps.



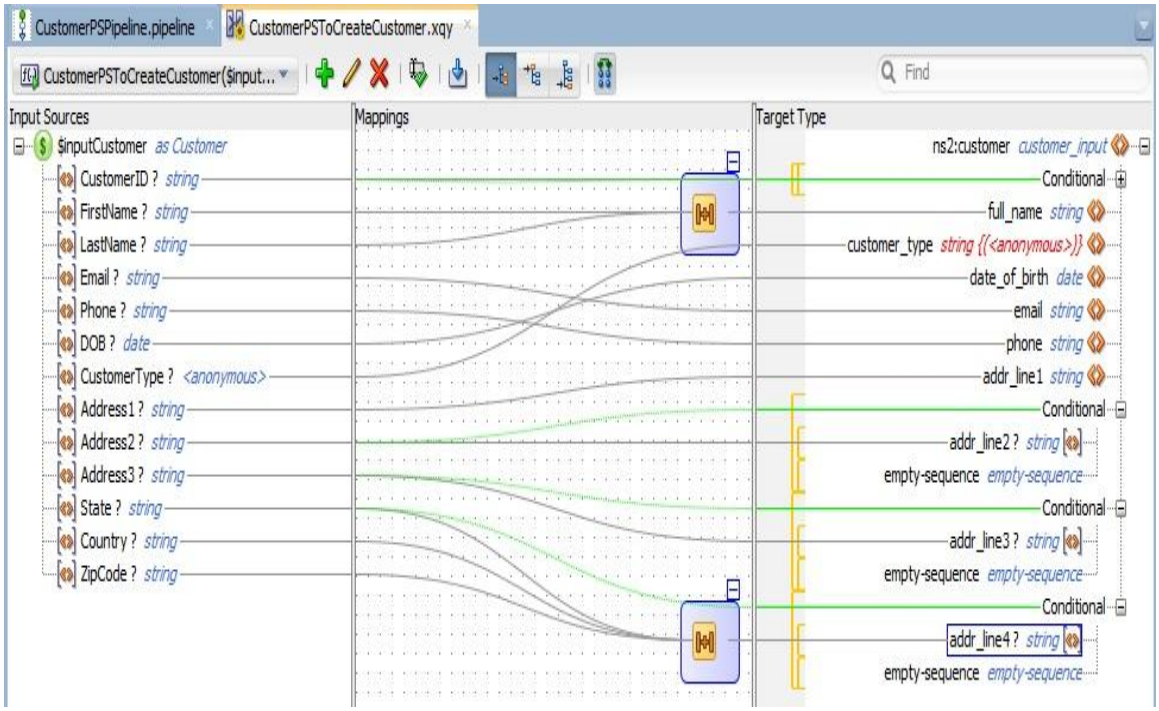

Function result type
XML Schema
Untyped
Schema Object Reference:
customer
Possible Sequence Type Form:
Schema Element
Schema Location:
Prefix:
/OSB/ServiceBusAppl/XYZCommon/CustomerService.wsdl
ns2
Not Applicable
Set a Namespace
NS URI:
Prefix:
Occurrence:
Exactly One
Result XQuery Expression:
schema-element(ns2:customer)
Warnings & Notes:
Help
OK
Cancel


Create XQuery Map Main module
Enter details to create an XQuery Map main module. Specify source and target elements by selecting global elements defined in either an XSD or WSDL file.
File Name:
CustomerPSToCreateCustomer
Directory Name:
D:\Technical\BlogProjects\12c\OSB\ServiceBusAppl\XYZCustomer\XQueries
Generate Function
Function Name:
CustomerPSToCreateCustomer
NS URI:
http://www.w3.org/2005/xquery-local-functions
Prefix:
local
Sources
Parameter
Sequence Type Definition
$inputCustomer
schema-element(ns1:Customer)
Target
schema-element(ns2:customer)
Options
Generate XQuery version line
Use schema type annotations
Help
OK
Cancel

Click **OK** and do initial mappings as shown below.



Drag **concat** function on to mapping as shown below from **Components -> XQuery Functions -> String Functions** to concatenate elements **FirstName/LastName** and **State/Country/ZipCode** to map to **full_name** and **addr_line4** respectively.

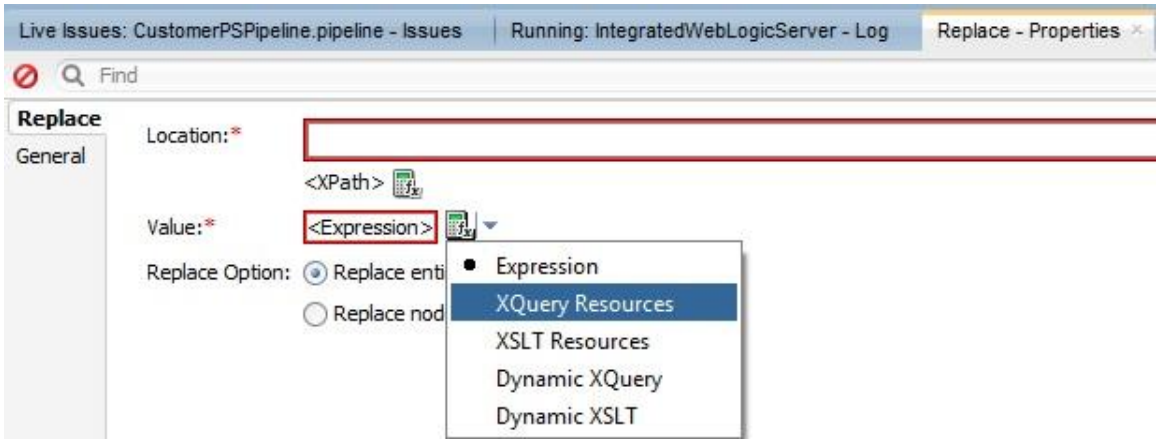


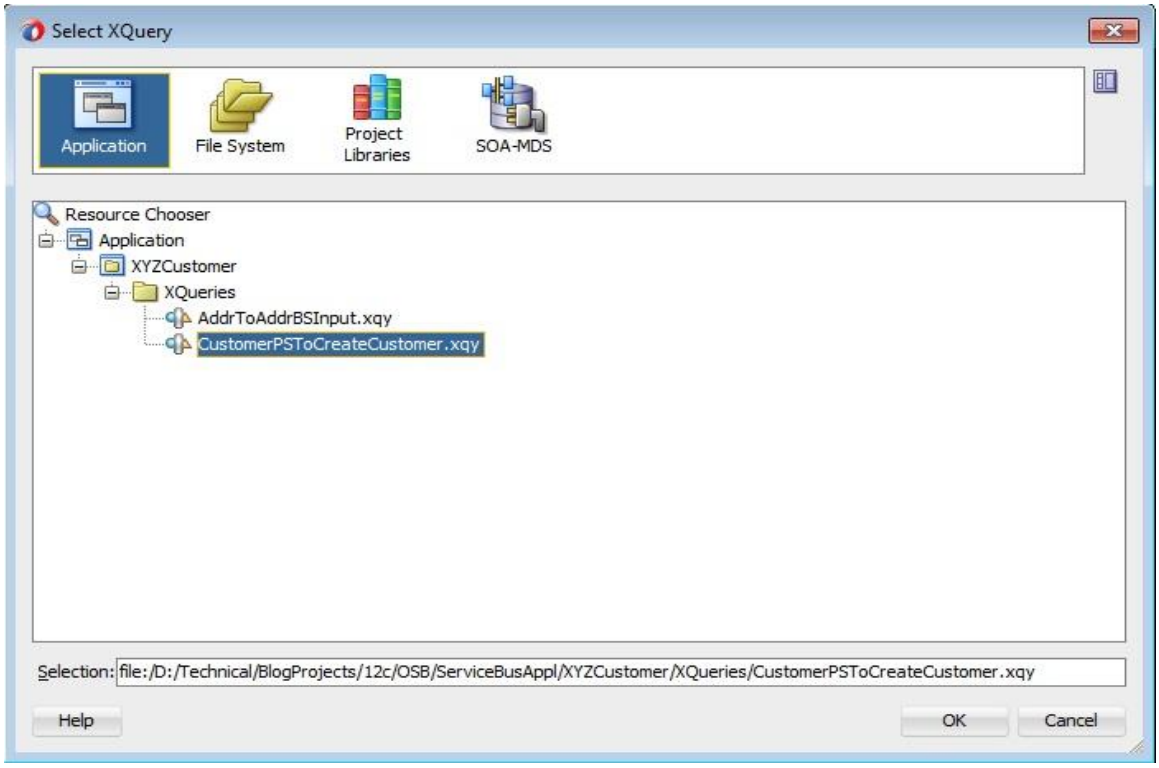
Go to **Source** tab and do modifications as highlighted below.

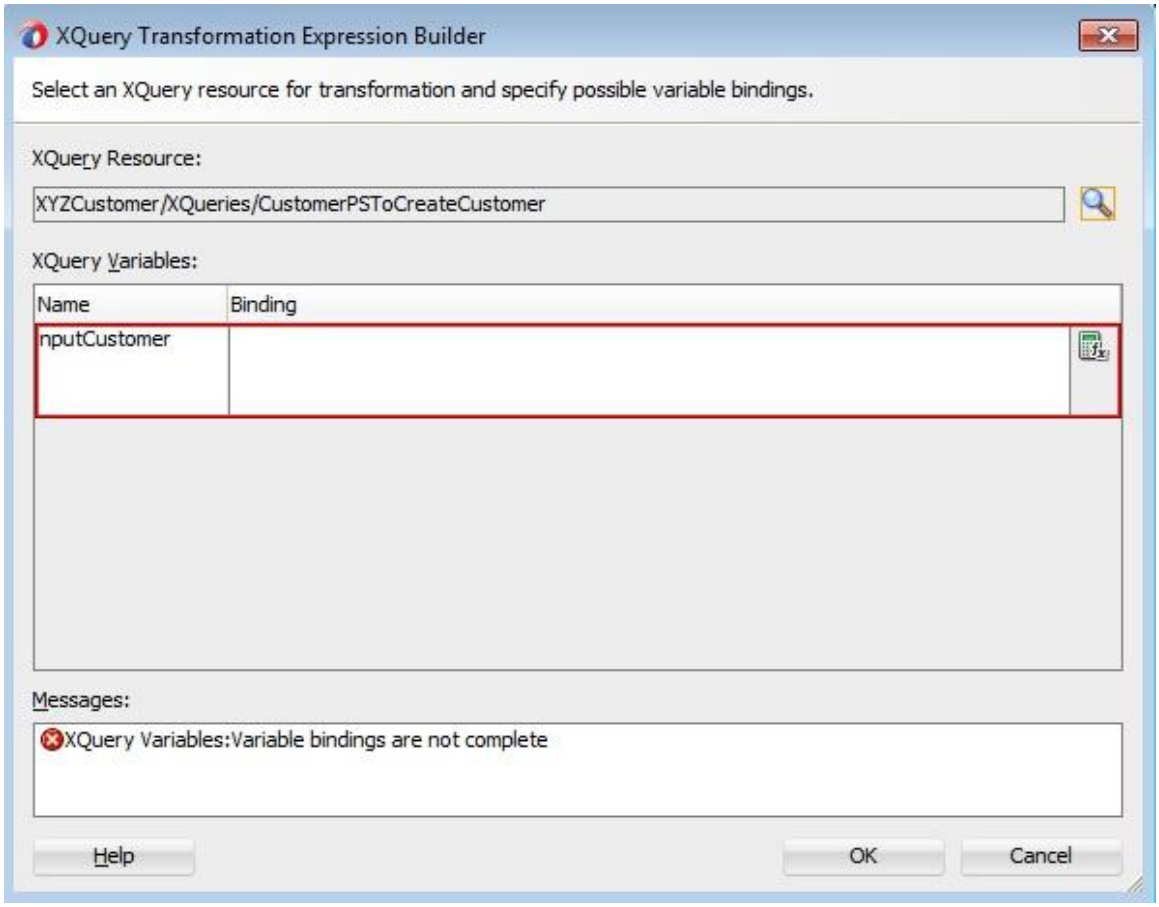
```
declare function local:CustomerPSToCreateCustomer($inputCustomer as element() (:: schema-element(ns1:Customer) ::)) as element() (:: schema-element(ns2:
    <ns2:customer>
        {
            if ($inputCustomer/CustomerID)
            then <customer_id>{fn:data($inputCustomer/CustomerID)}</customer_id>
            else ()
        }
        <full_name>{fn:concat(fn:data($inputCustomer/FirstName),',', $inputCustomer/LastName)}</full_name>
        <customer_type>{fn:data($inputCustomer/CustomerType)}</customer_type>
        <date_of_birth>{fn:data($inputCustomer/DOB)}</date_of_birth>
        <email>{fn:data($inputCustomer/Email)}</email>
        <phone>{fn:data($inputCustomer/Phone)}</phone>
        <addr_line1>{fn:data($inputCustomer/Address1)}</addr_line1>
        {
            if ($inputCustomer/Address2)
            then <addr_line2>{fn:data($inputCustomer/Address2)}</addr_line2>
            else ()
        }
        {
            if ($inputCustomer/Address3)
            then <addr_line3>{fn:data($inputCustomer/Address3)}</addr_line3>
            else ()
        }
        {
            if ($inputCustomer/State or $inputCustomer/Country or $inputCustomer/ZipCode  )
            then <addr_line4>{fn:concat(fn:data($inputCustomer/State),',', $inputCustomer/Country,',', fn:data($inputCustomer/ZipCode))}</addr_line4>
            else ()
        }
    </ns2:customer>
};

local:CustomerPSToCreateCustomer($inputCustomer)
```

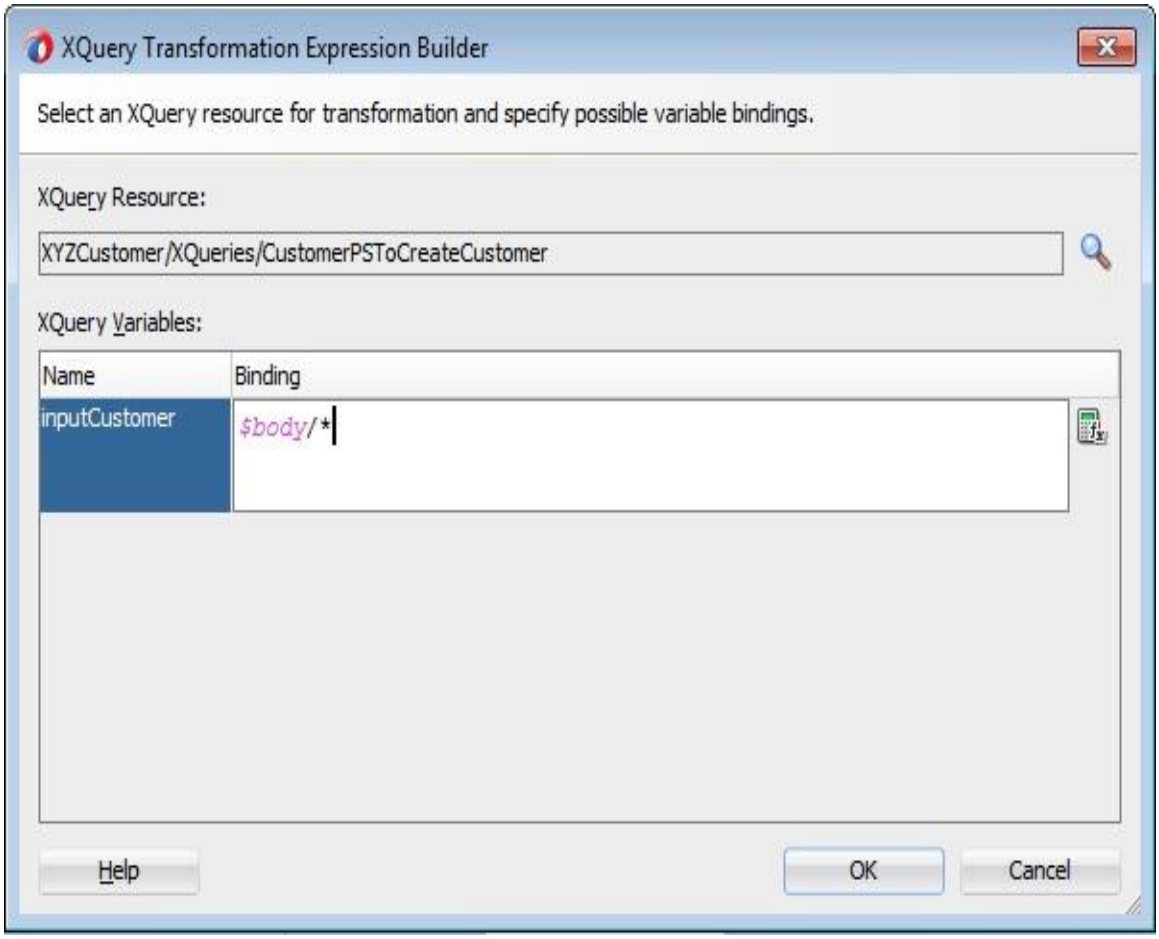
Drag **Replace** activity into **Request Action** of **Routing** from **Message Processing**. In **Properties** tab, bring up expression editor for **Value** property and choose above transformation.



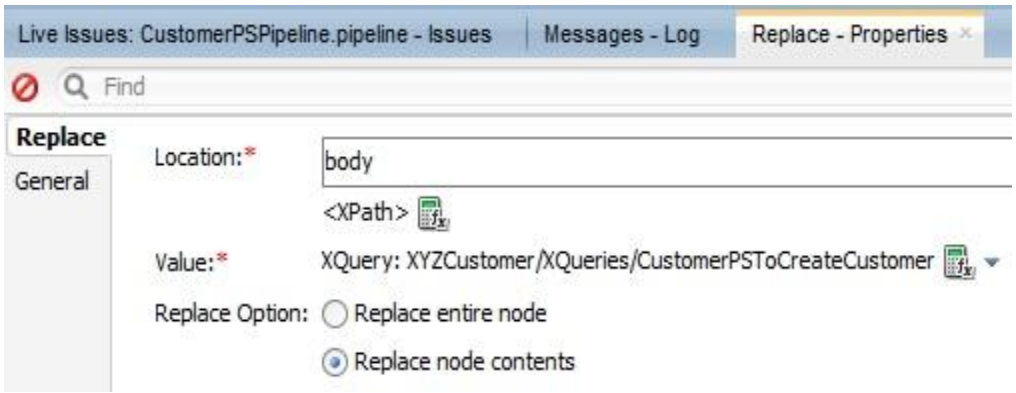

Select XQuery
Application
File System
Project Libraries
SOA-MDS
Resource Chooser
Application
XYZCustomer
XQueries
AddrToAddrBSInput.xqy
CustomerPSToCreateCustomer.xqy
Selection: file:/D:/Technical/BlogProjects/12c/OSB/ServiceBusAppl/XYZCustomer/XQueries/CustomerPSToCreateCustomer.xqy
Help
OK
Cancel


XQuery Transformation Expression Builder
Select an XQuery resource for transformation and specify possible variable bindings.
XQuery Resource:
XYZCustomer/XQueries/CustomerPSToCreateCustomer
XQuery Variables:
Name
Binding
nputCustomer
Messages:
XQuery Variables:Variable bindings are not complete
Help
OK
Cancel

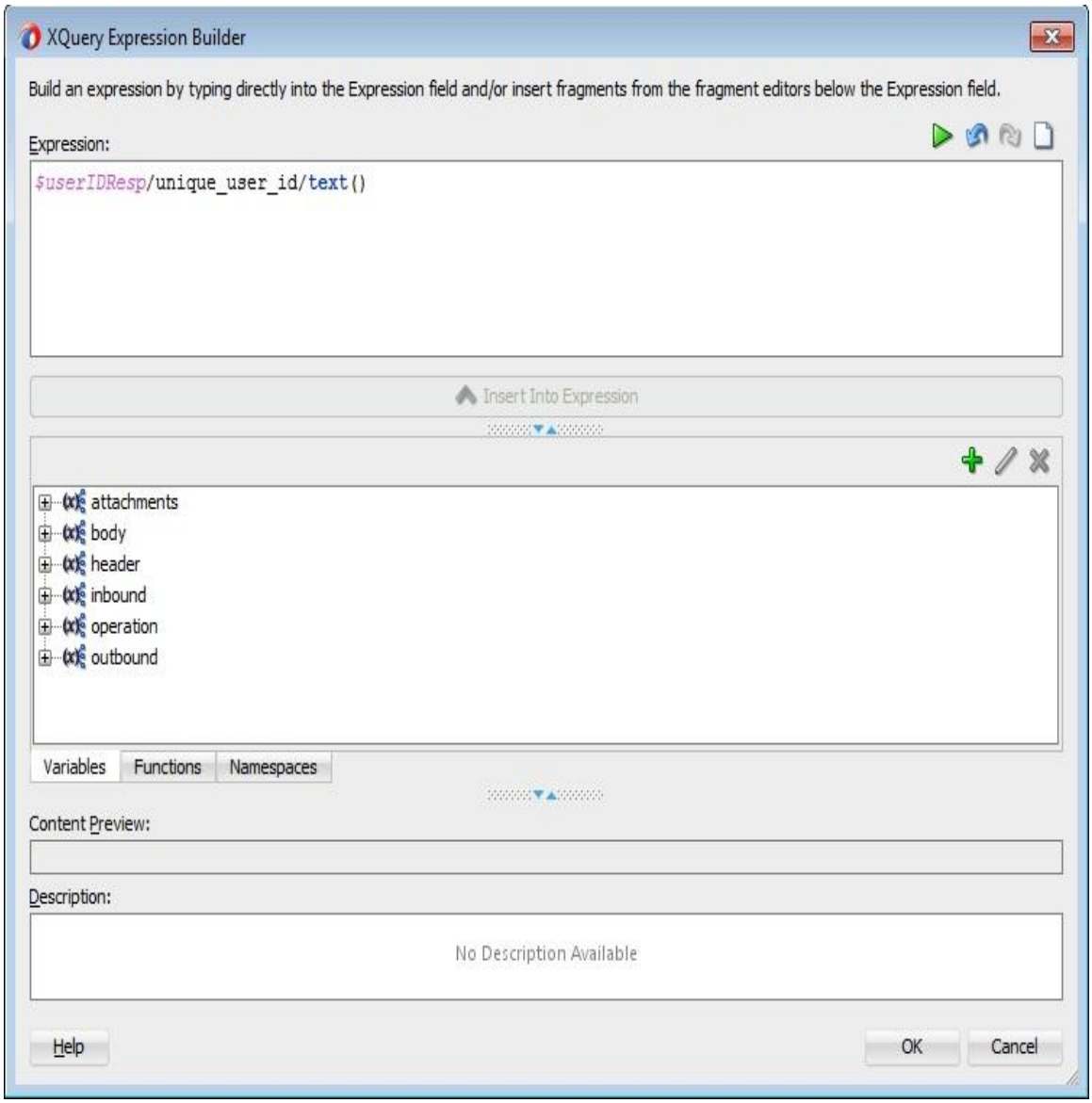
Give bind variable value as **$body/***.

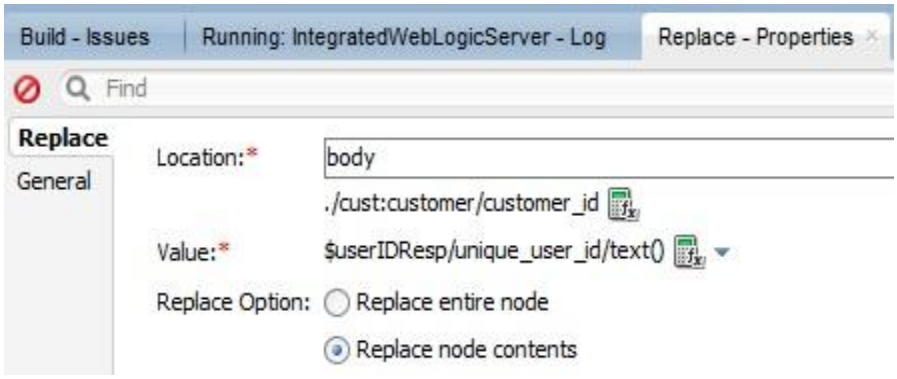


Set other properties as shown below. This action will replace contents of **$body** context variable with transformed payload as expected by business service.

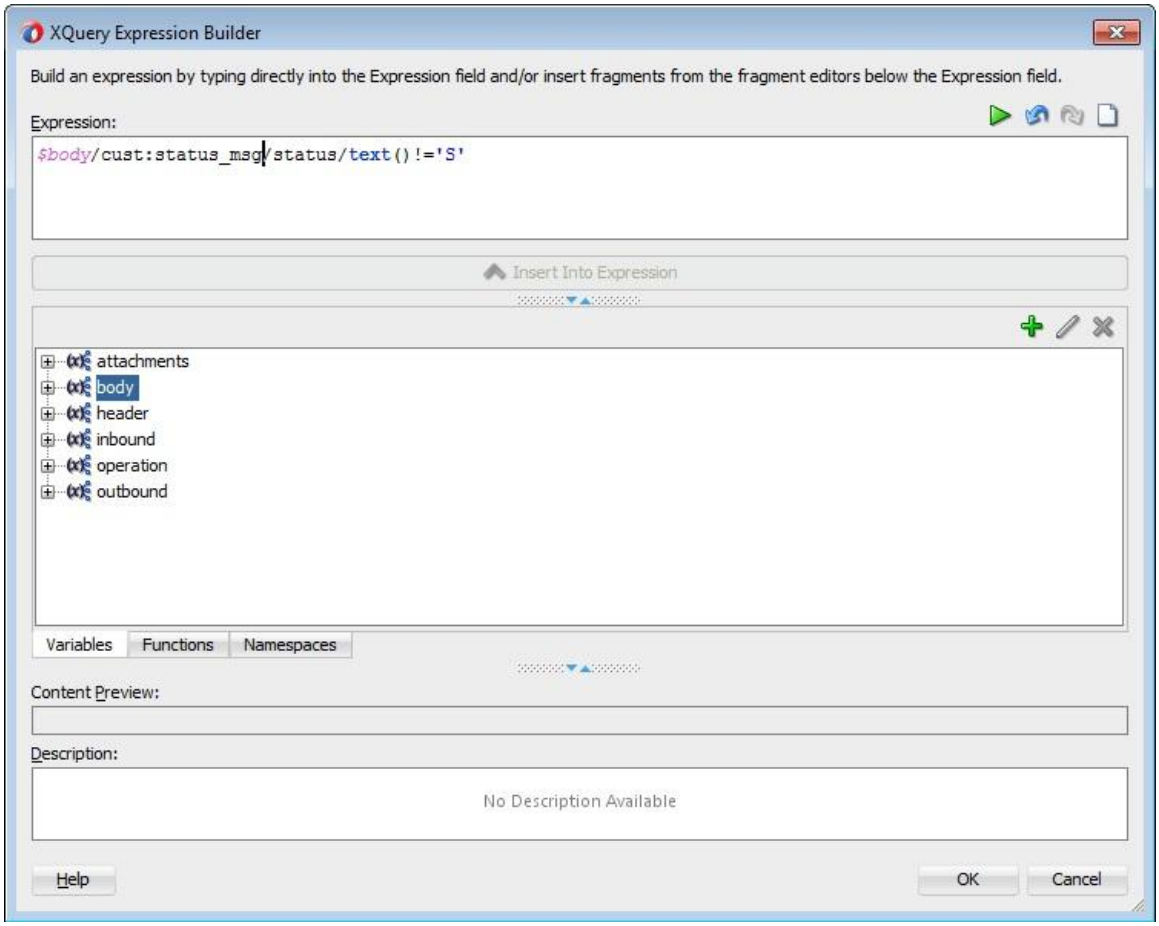


Drag another **Replace** activity into **Request Action** of **Routing** to replace contents of **Customer ID** with Unique ID got it from **UserIDServiceBS**. In **Properties** tab, bring up expression editor for Value property and give **expression** as **$userIDResp/unique_user_id/text().** Set other properties as shown below.



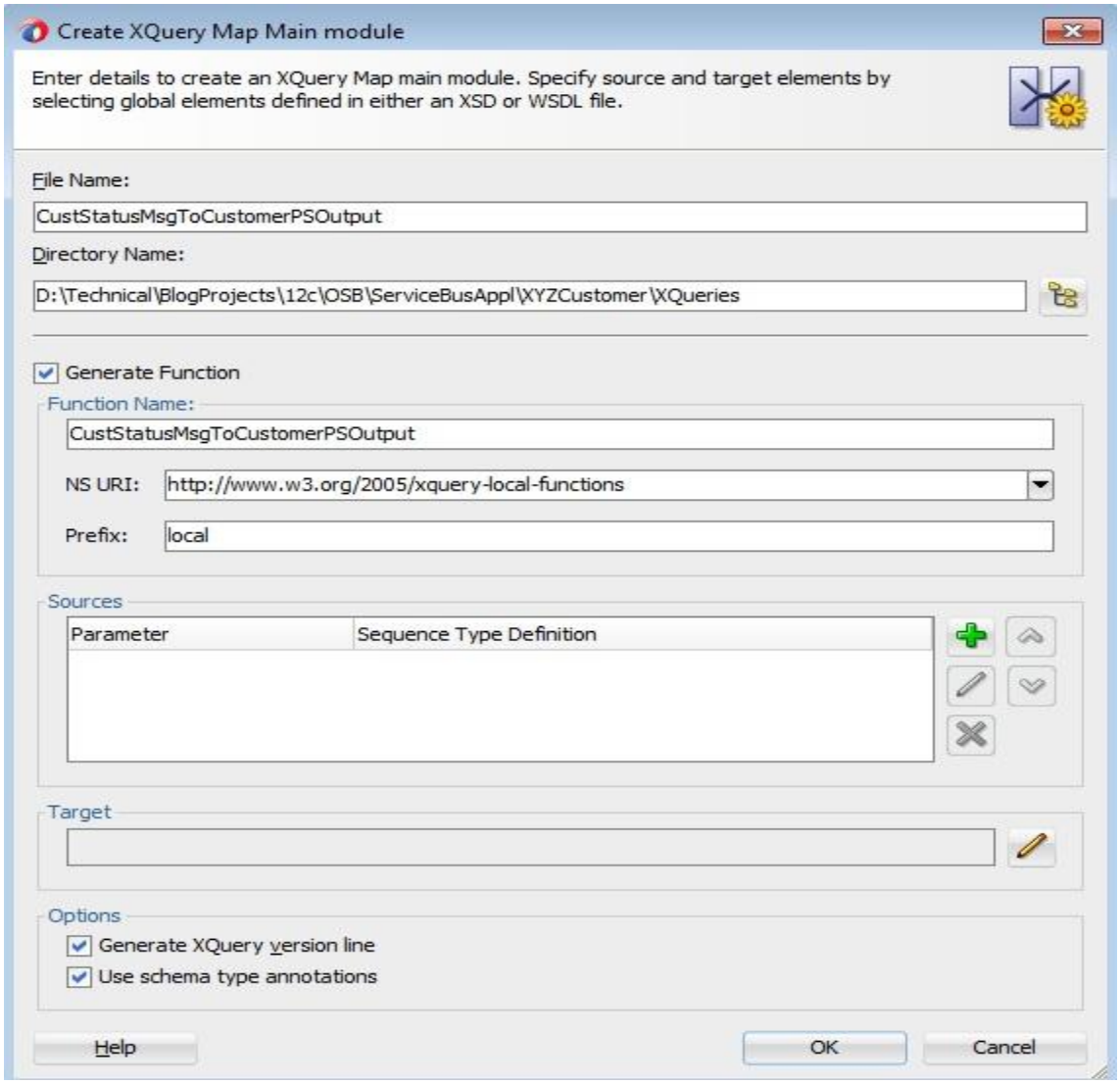


Drag **If-Then** activity into **Response Action** of **Routing** activity from **Flow Control** to verify status returned by business service. Set condition as shown below and add namespace alias **cust** with namespace **urn:xyzbank:cust:schema:customer** in **Namespaces** tab.

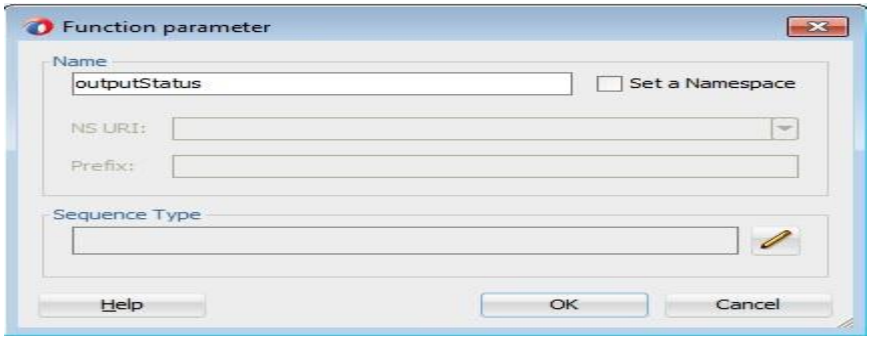


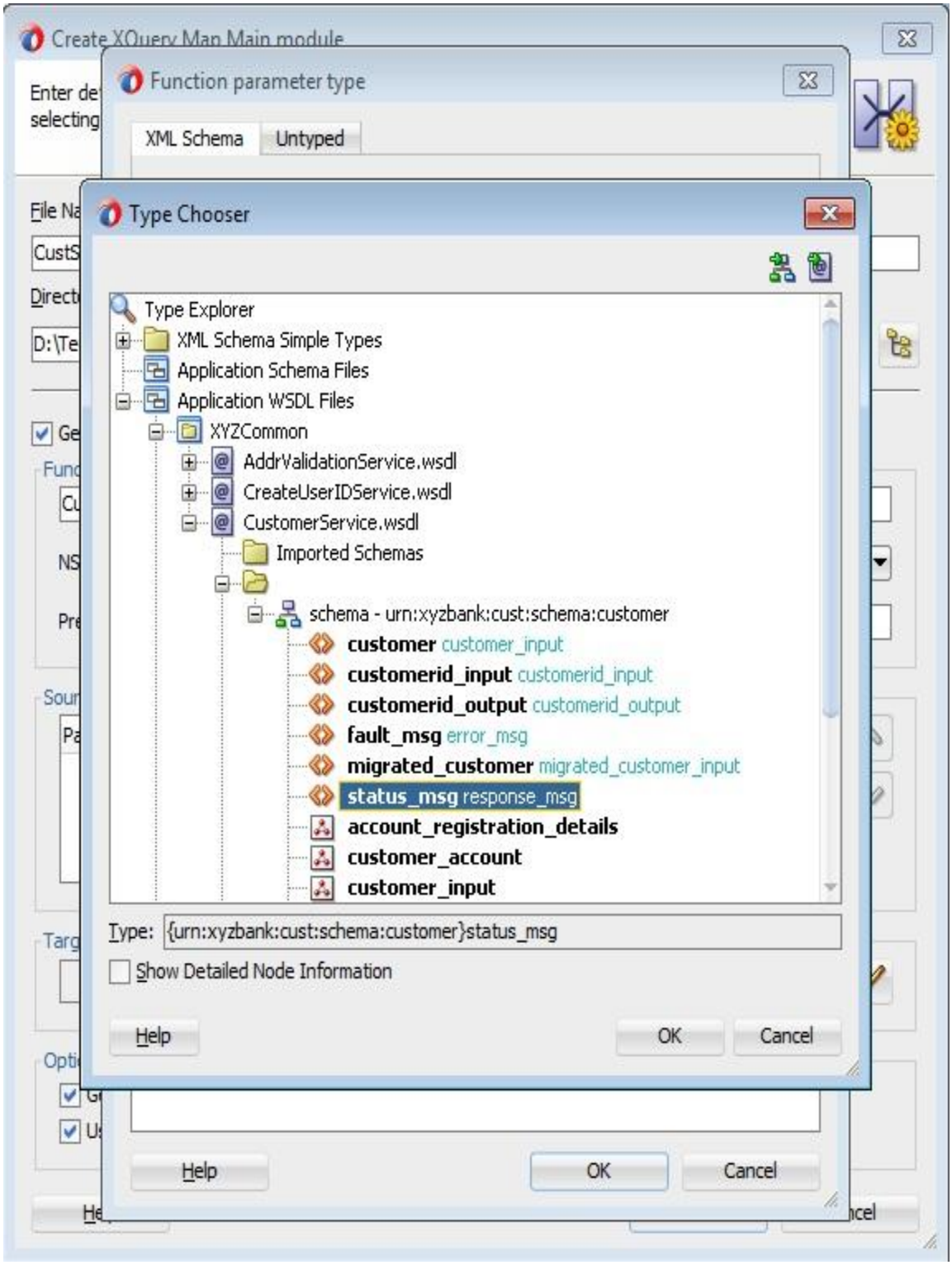
Drag **Assign** activity into **If** branch from **Message Processing** and set properties as shown below to store unsuccessful response in **faultVar** variable. Drag **RaiseError** activity from **Flow Control** and give error code as **XYZ-0004**. The actual Fault Response will be sent from Proxy Service Error Handler that will be discussed later in this tutorial.

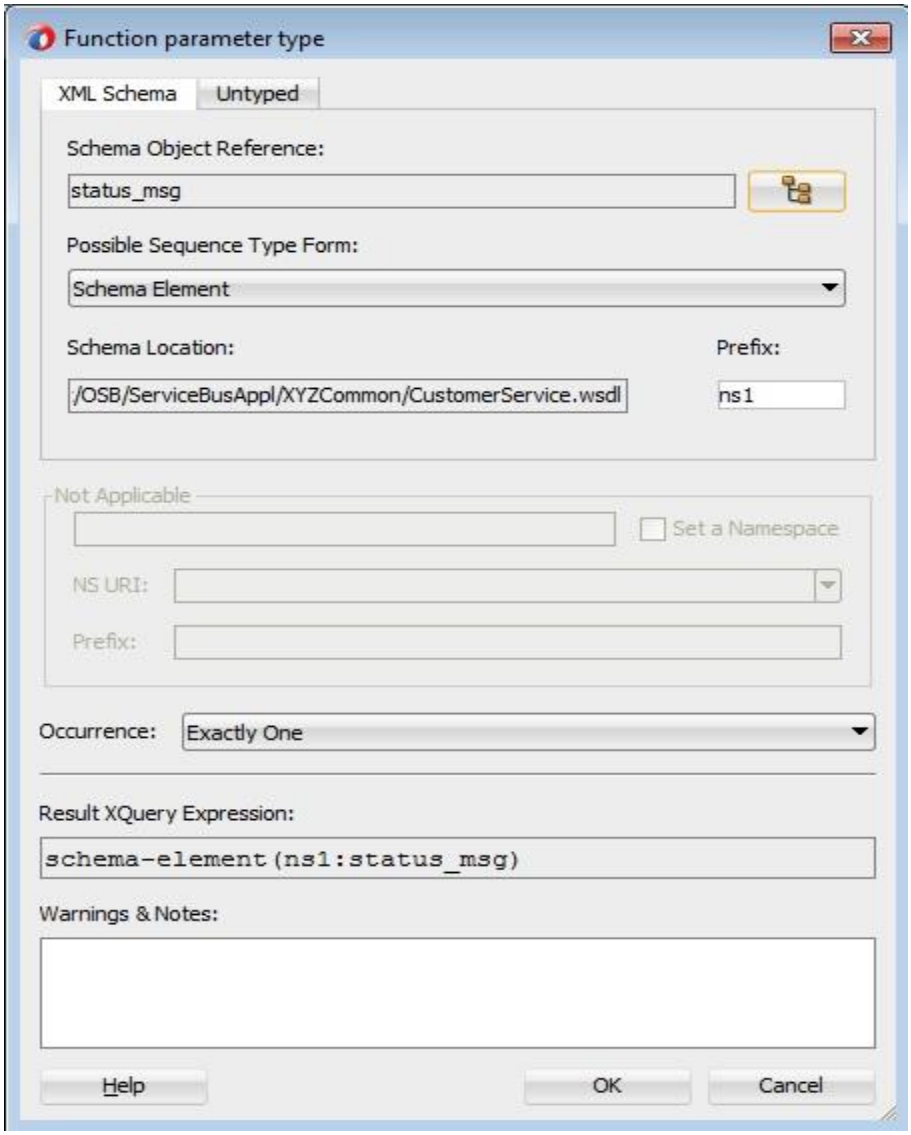
Now create another **XQuery** map in **XQueries** folder of **XYZCustomer** project and give both **File Name** and **Function Name** as **CustStatusMsgToCustomerPSOutput**, to transform response of business service to response structure of Proxy Service on successful customer creation.

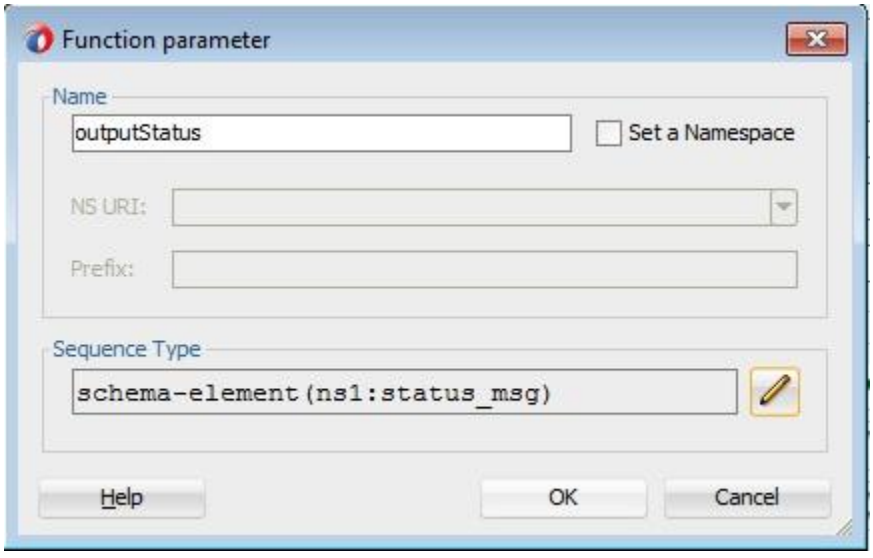


Complete **Source** parameters selection as shown below and you can refer to previous sections for actual navigation steps.

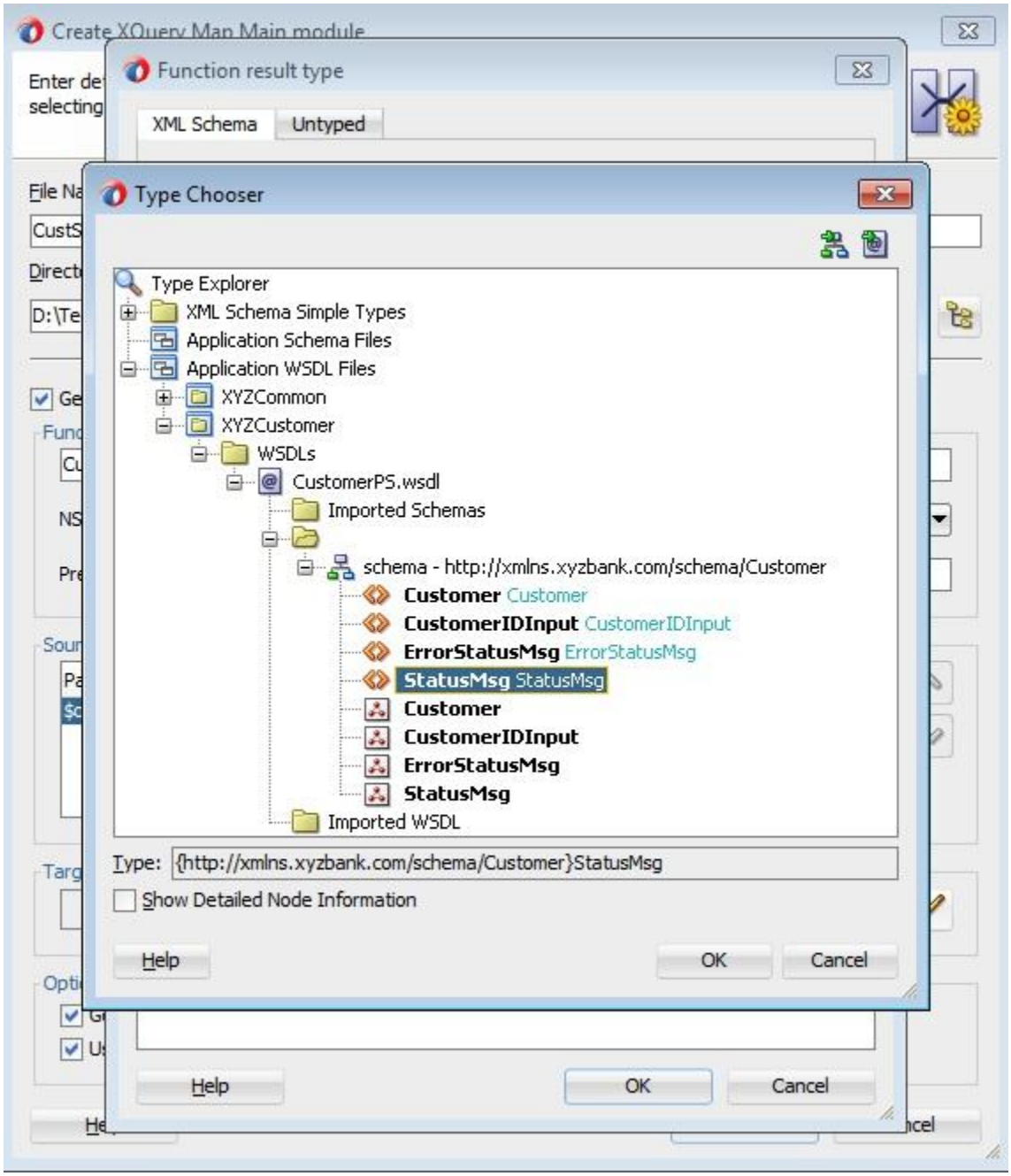


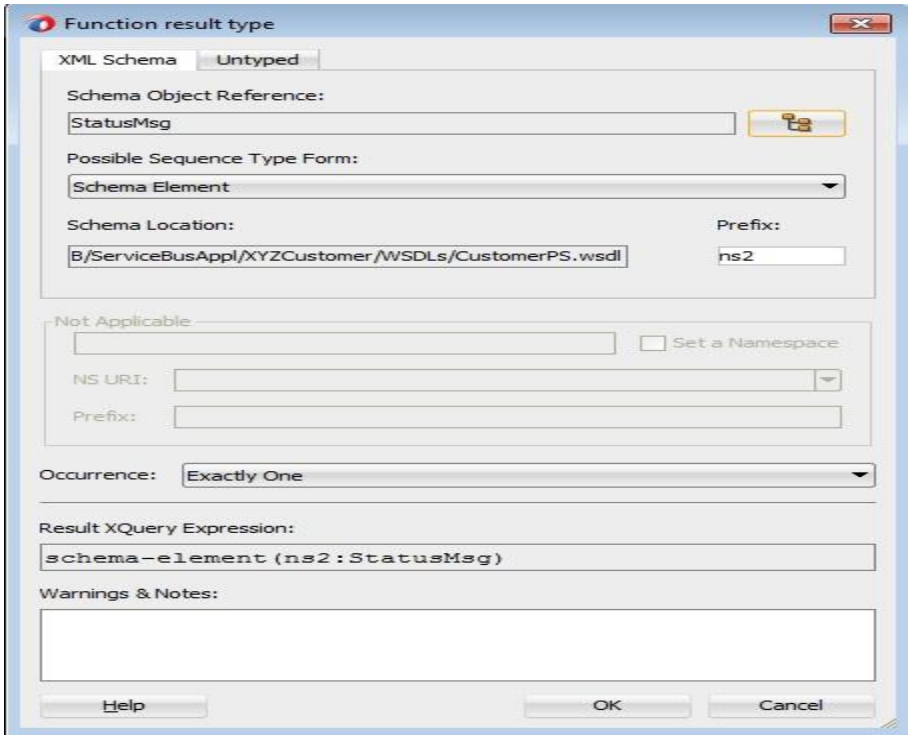

Create XQuery Map Main module
Function parameter type
XML Schema
Untyped
Type Chooser
Type Explorer
XML Schema Simple Types
Application Schema Files
Application WSDL Files
XYZCommon
AddrValidationService.wsdl
CreateUserIDService.wsdl
CustomerService.wsdl
Imported Schemas
schema - urn:xyzbank:cust:schema:customer
customer customer_input
customerid_input customerid_input
customerid_output customerid_output
fault_msg error_msg
migrated_customer migrated_customer_input
status_msg response_msg
account_registration_details
customer_account
customer_input
Type: {urn:xyzbank:cust:schema:customer}status_msg
Show Detailed Node Information
Help
OK
Cancel
Help
OK
Cancel

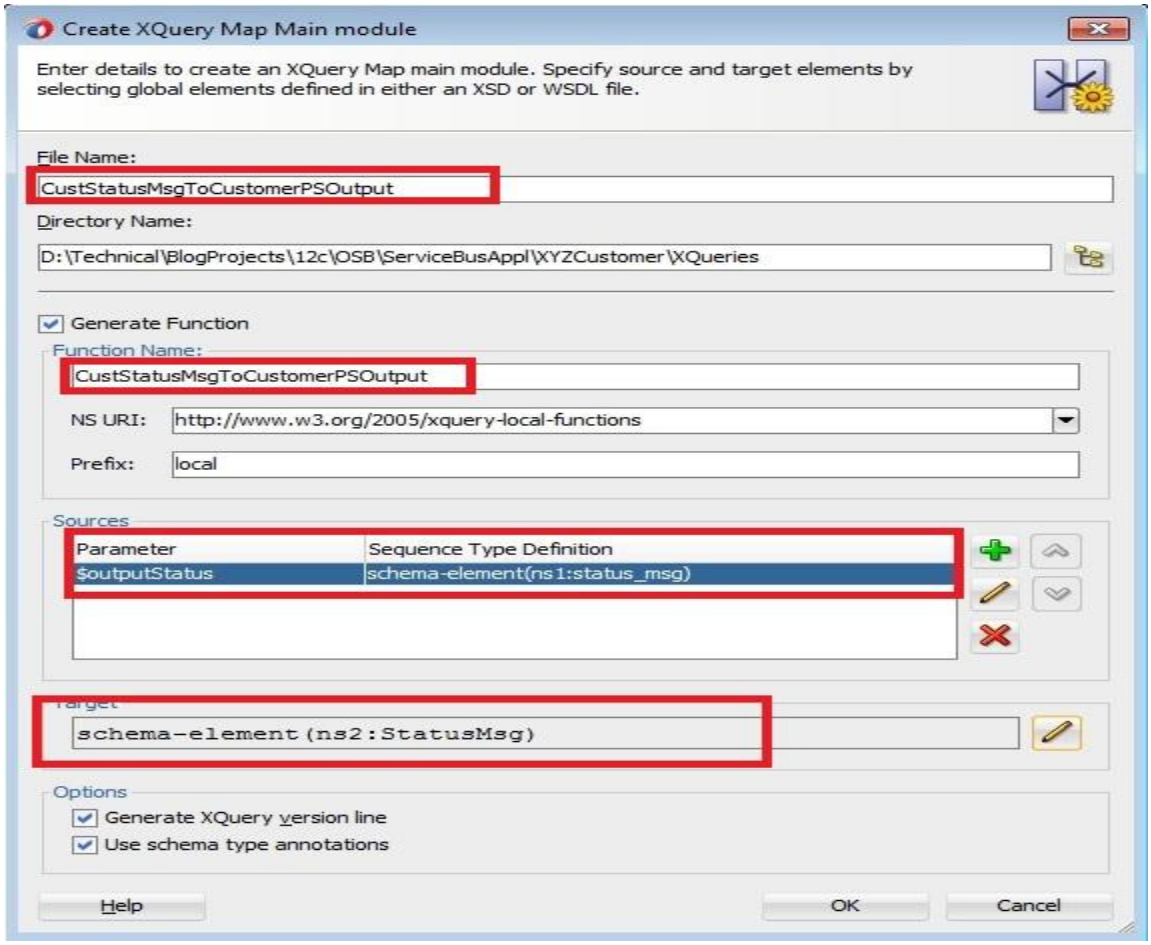

Function parameter type
XML Schema
Untyped
Schema Object Reference:
status_msg
Possible Sequence Type Form:
Schema Element
Schema Location:
Prefix:
/OSB/ServiceBusAppl/XYZCommon/CustomerService.wsdl
ns1
Not Applicable
Set a Namespace
NS URI:
Prefix:
Occurrence:
Exactly One
Result XQuery Expression:
schema-element(ns1:status_msg)
Warnings & Notes:
Help
OK
Cancel


Function parameter
Name
outputStatus
Set a Namespace
NS URI:
Prefix:
Sequence Type
schema-element(ns1:status_msg)
Help
OK
Cancel

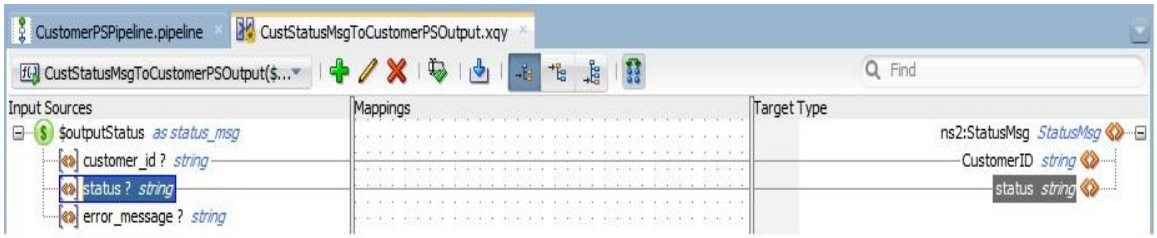
Complete **Target** selection as shown below and you can refer to previous sections for actual navigation steps.



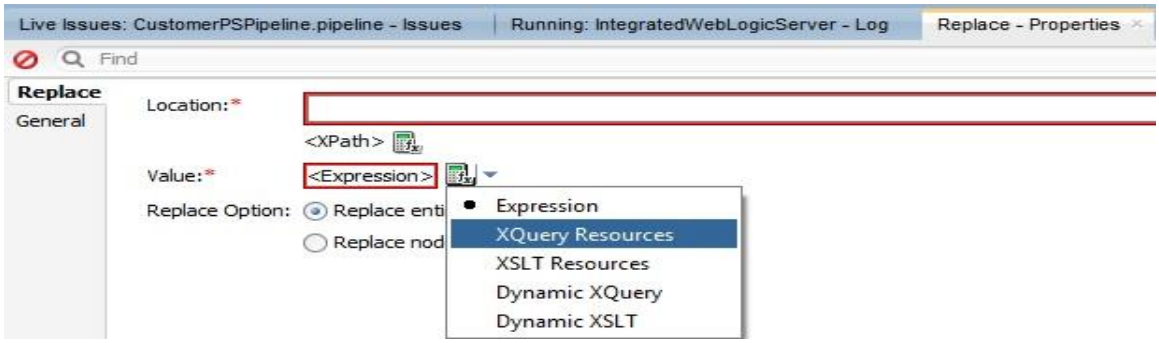

Function result type
XML Schema
Untyped
Schema Object Reference:
StatusMsg
Possible Sequence Type Form:
Schema Element
Schema Location:
Prefix:
B/ServiceBusAppl/XYZCustomer/WSDLs/CustomerPS.wsdl
ns2
Not Applicable
Set a Namespace
NS URI:
Prefix:
Occurrence:
Exactly One
Result XQuery Expression:
schema-element(ns2:StatusMsg)
Warnings & Notes:
Help
OK
Cancel

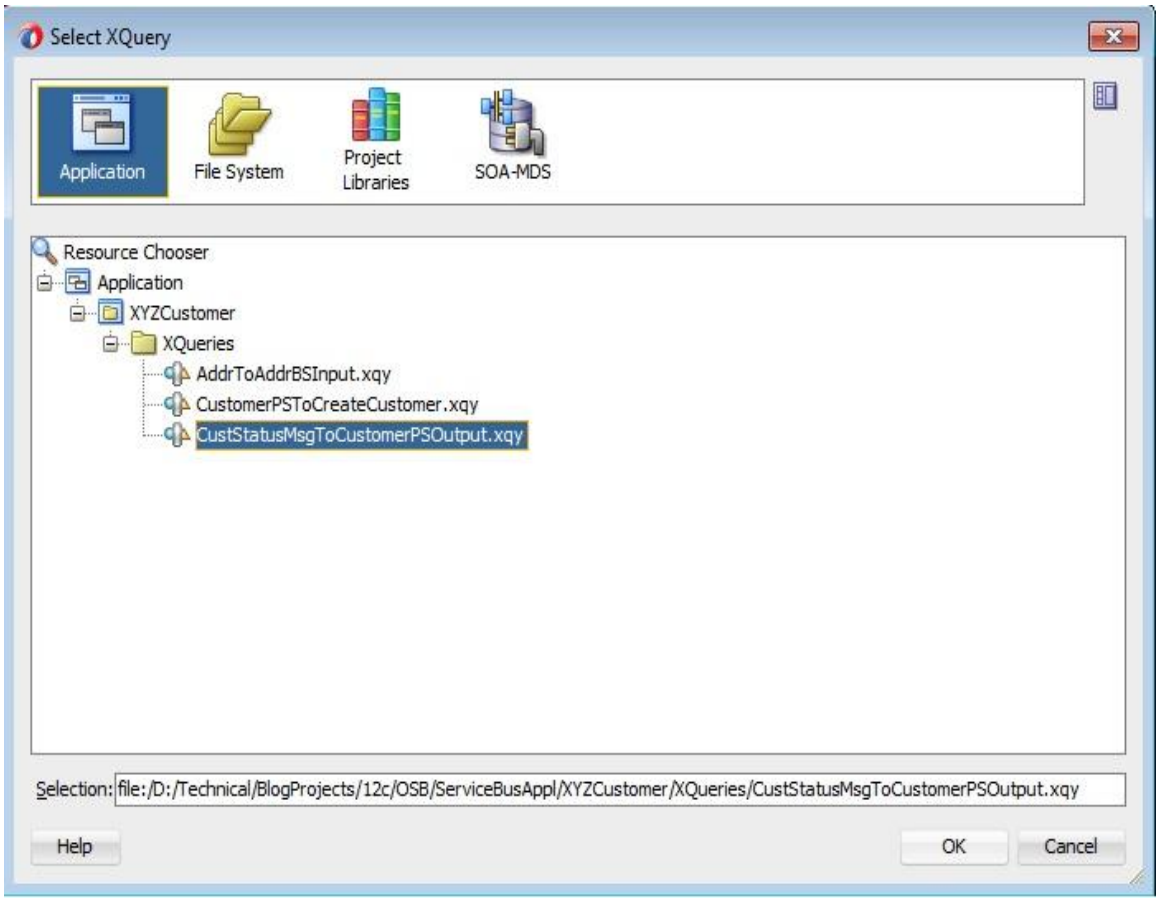

Create XQuery Map Main module
Enter details to create an XQuery Map main module. Specify source and target elements by selecting global elements defined in either an XSD or WSDL file.
File Name:
CustStatusMsgToCustomerPSOutput
Directory Name:
D:\Technical\BlogProjects\12c\OSB\ServiceBusAppl\XYZCustomer\XQueries
Generate Function
Function Name:
CustStatusMsgToCustomerPSOutput
NS URI:
http://www.w3.org/2005/xquery-local-functions
Prefix:
local
Sources
Parameter
Sequence Type Definition
$outputStatus
schema-element(ns1:status_msg)
Target
schema-element(ns2:StatusMsg)
Options
Generate XQuery version line
Use schema type annotations
Help
OK
Cancel

Click **OK** and finish the mapping as shown below.

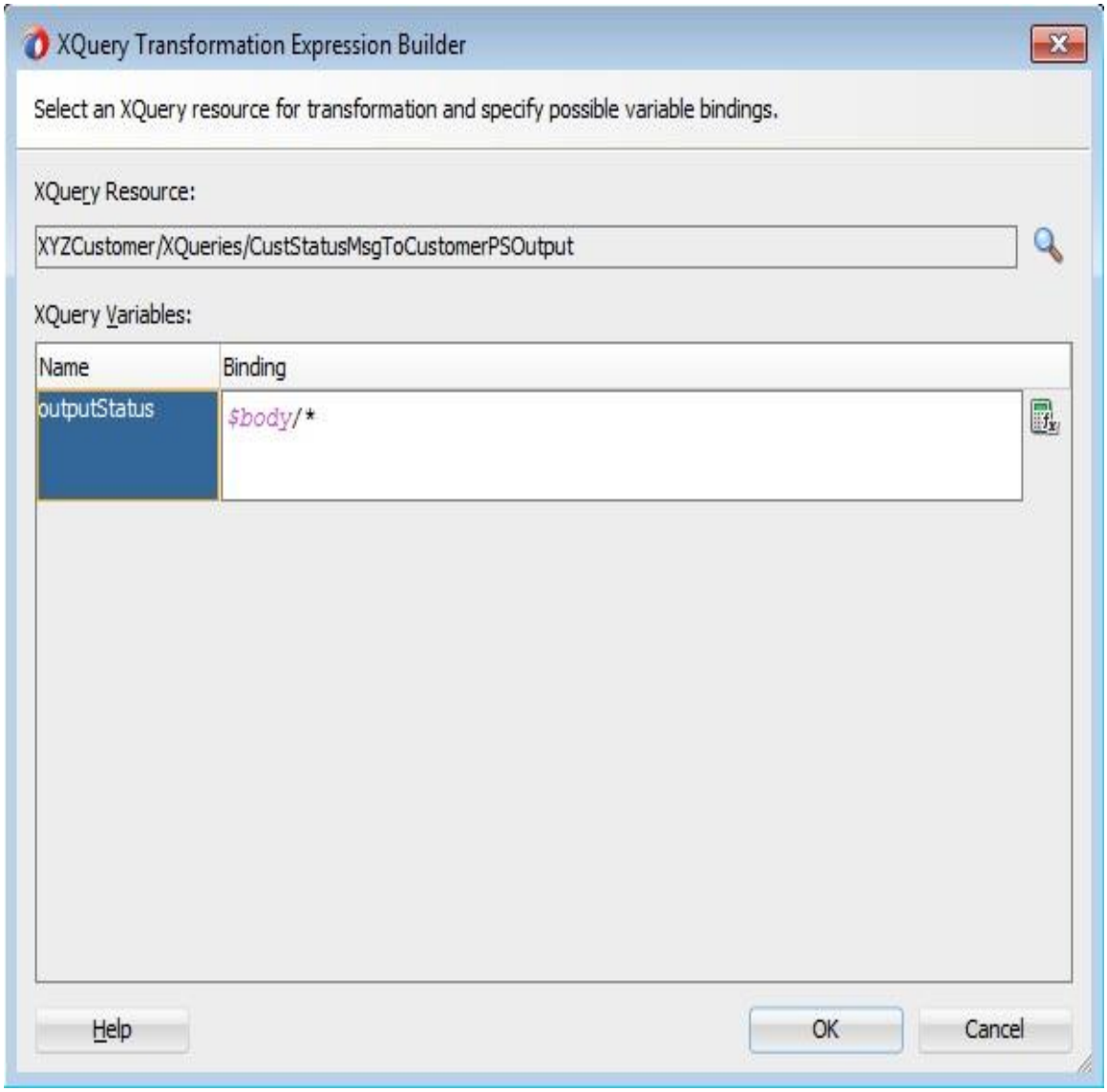


Drag **Replace** activity into **Response Action** of **Routing** after If-Then from **Message Processing**. Bring up expression builder for **Value** property and select above XQuery map as shown below.

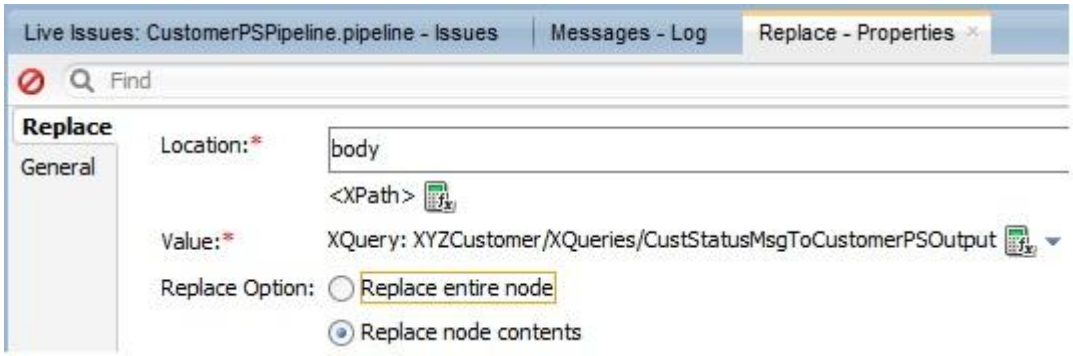




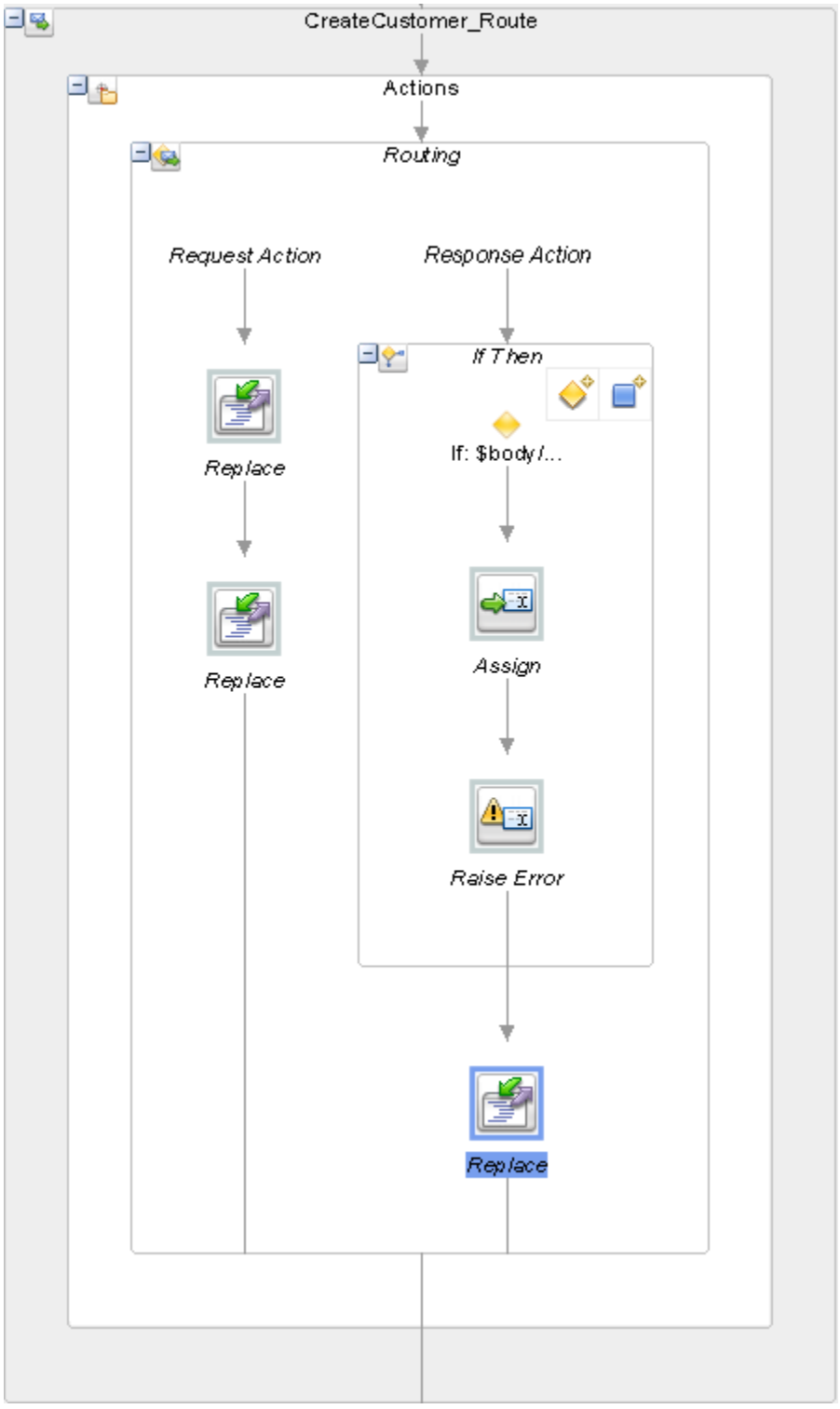
Give **$body/*** as value for **Binding**.



Set other properties as shown below. This action will replace contents of **$body** context variable which will be ultimately send to your Proxy service consumer.



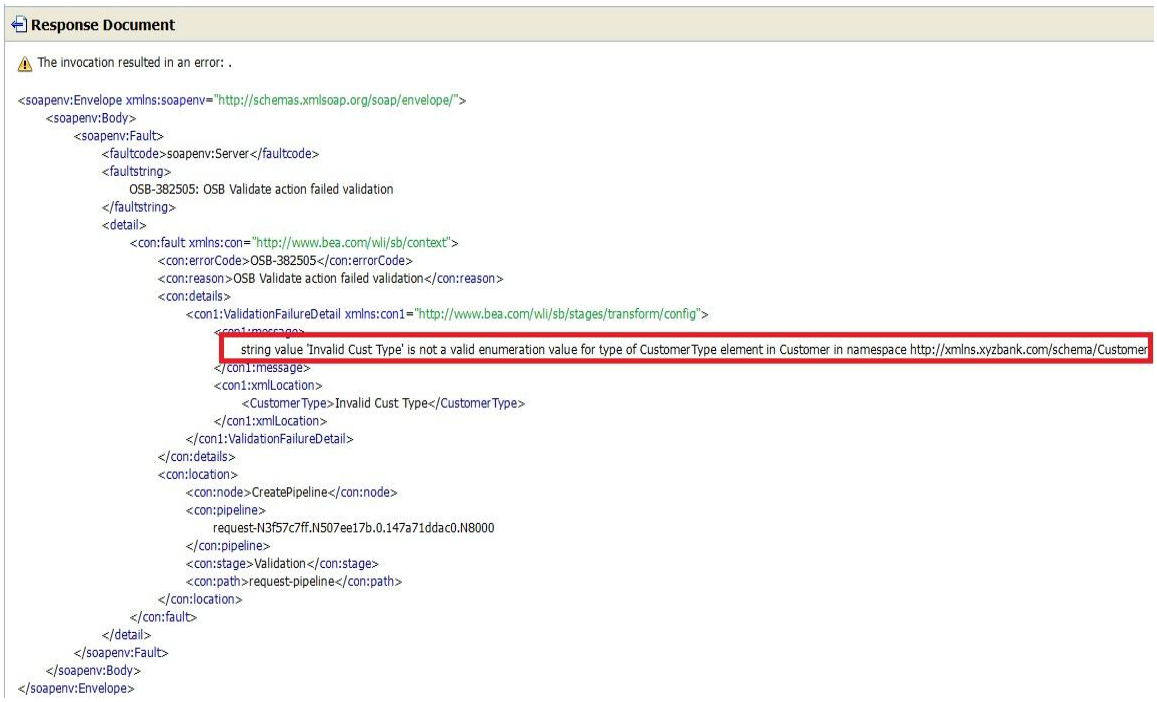
Now your **Routing** node should look like below.



Testing

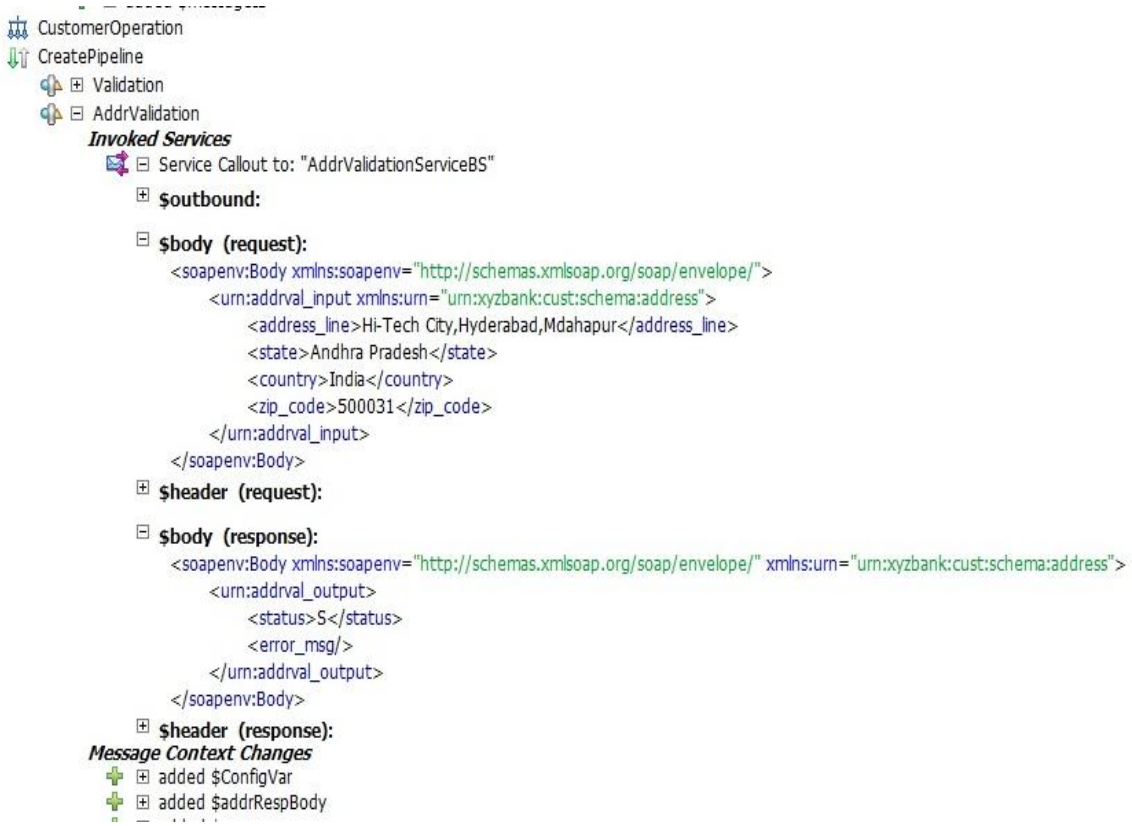
Deploy both projects or run **Pipeline** directly as shown in **Deploying and Testing** section. You may want to frequently test your proxy service/pipeline during development. Run your **pipeline** with sample payloads given here and observe **Flow Trace and Variables** as shown below. You can also run Proxy Service but you will not observe any **Flow Trace**.

CustomerType element Validation:



Email element Validation:

Positive Case - ValidateAddress with configuration property value as Y:



AddrValidation

Invoked Services

Service Callout to: "AddrValidationServiceBS"

$outbound:

$body (request):

$header (request):

$body (response):

$header (response):

Message Context Changes

added $ConfigVar

Y

added $addrRespBody

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:xyzbank:cust:schema:address">
    <urn:addrval_output>
        <status>S</status>
        <error_msg/>
    </urn:addrval_output>
</soapenv:Body>
```

added $addrReqBody

```
<soap-env:Body xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <ns2:addrval_input xmlns:ns2="urn:xyzbank:cust:schema:address">
        <address_line>Hi-Tech City,Mdahapur,Hyderabad</address_line>
        <state>Andhra Pradesh</state>
        <country>India</country>
        <zip_code>500031</zip_code>
    </ns2:addrval_input>
</soap-env:Body>
```

changed $body

changed $inbound

Service Callout to: "UserIDServiceBS"

$outbound:

$body (request):

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <usr:user_input xmlns:usr="urn:xyzbank:cust:schema:userid">
        <usr:system>LEGACY</usr:system>
    </usr:user_input>
</soapenv:Body>
```

$header (request):

```
<soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <usr:UserIDConfiguration xmlns:usr="urn:xyzbank:cust:schema:userid">
        <usr:UserName>admin</usr:UserName>
        <usr:Password>password1</usr:Password>
    </usr:UserIDConfiguration>
</soapenv:Header>
```

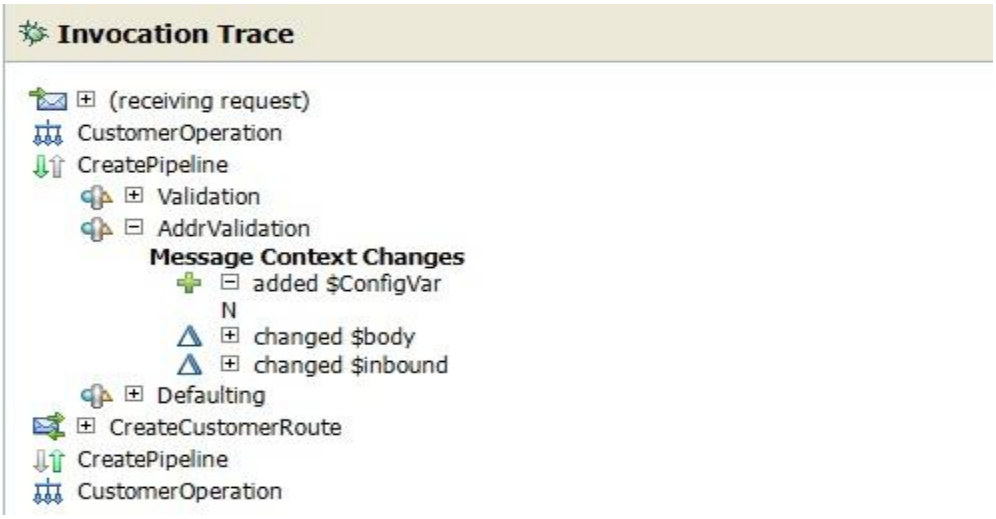
$body (response):

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:xyzbank:cust:schema:userid">
    <urn:user_output>
        <unique_user_id>AHHHHH8787QDJHJH299</unique_user_id>
        <status>S</status>
        <error_msg/>
    </urn:user_output>
</soapenv:Body>
```

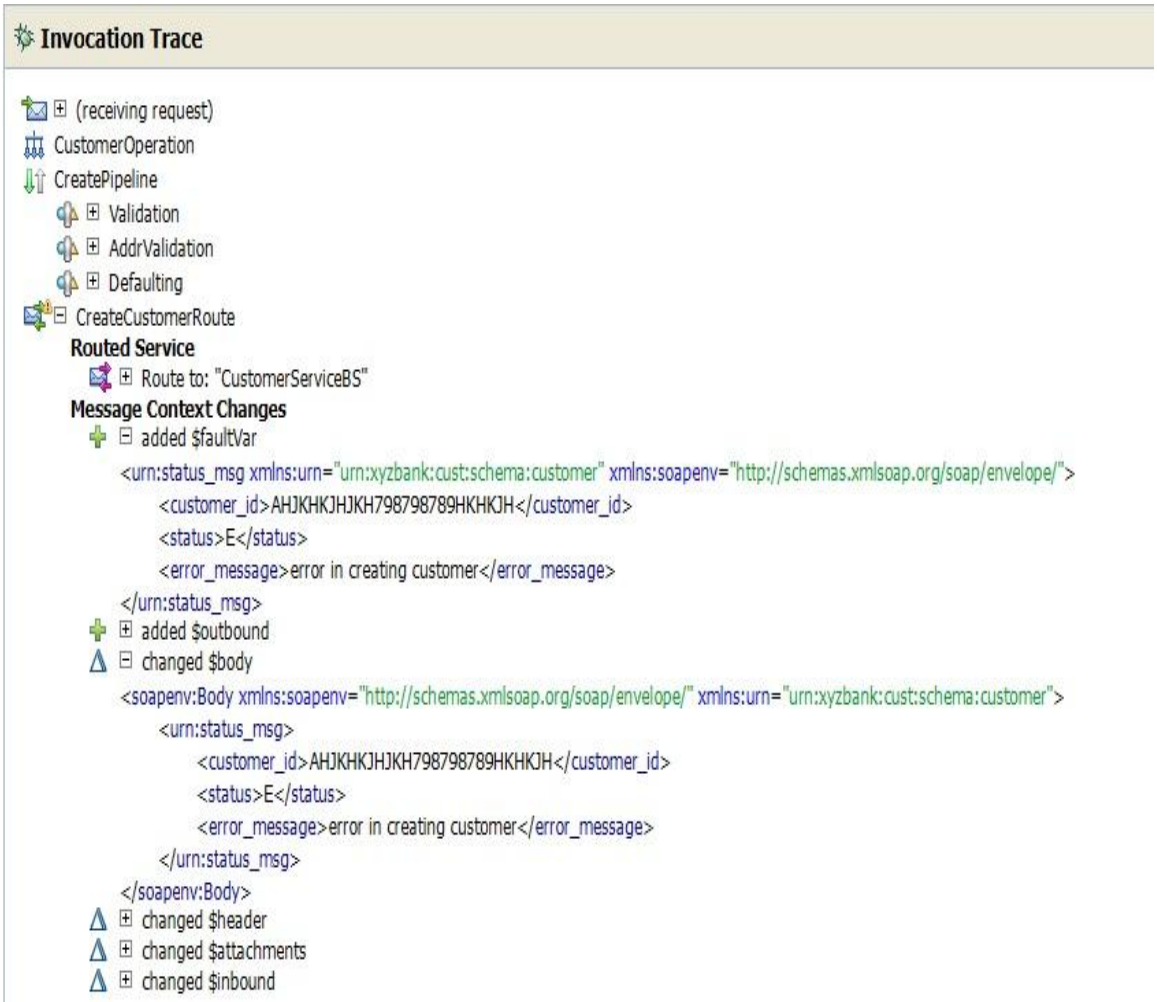
$header (response):


Message Context Changes
added $userIDResp
<urn:user_output xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:xyzbank:cust:schema:userid">
<unique_user_id>AHHHHH8787QDJHJH299</unique_user_id>
<status>S</status>
<error_msg/>
</urn:user_output>
added $varSA
<con:UsernamePasswordCredential xmlns:con="http://www.bea.com/wli/sb/services/security/config">
<con:username>admin</con:username>
<con:password>password1</con:password>
</con:UsernamePasswordCredential>
added $userIDReq
<usr:user_input xmlns:usr="urn:xyzbank:cust:schema:userid">
<usr:system>LEGACY</usr:system>
</usr:user_input>
added $userIDReqHdr
<soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<usr:UserIDConfiguration xmlns:usr="urn:xyzbank:cust:schema:userid">
<usr:UserName>admin</usr:UserName>
<usr:Password>password1</usr:Password>
</usr:UserIDConfiguration>
</soapenv:Header>
changed $body
changed $inbound
(receiving request)
CustomerOperation
CreatePipeline
Validation
AddrValidation
Defaulting
CreateCustomerRoute
Routed Service
Route to: "CustomerServiceBS"
$outbound:
$body (request):
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<urn:customer xmlns:urn="urn:xyzbank:cust:schema:customer">
<customer_id>AHHHHH8787QDJHJH299</customer_id>
<full_name>S.N. RAO</full_name>
<customer_type>Individual</customer_type>
<date_of_birth>1976-09-29</date_of_birth>
<email>abc@gmail.com</email>
<phone>+91 9999999999</phone>
<addr_line1>Hi-Tech City</addr_line1>
<addr_line2>Mdahapur</addr_line2>
<addr_line3>Hyderabad</addr_line3>
<addr_line4>Andhra Pradesh,India-500031</addr_line4>
</urn:customer>
</soapenv:Body>
$header (request):
$attachments (request):
Message Context Changes
added $outbound
changed $body
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:xyzbank:cust:schema:customer">
<cus:StatusMsg xmlns:cus="http://xmlns.xyzbank.com/schema/Customer">
<CustomerID>AHHHHH8787QDJHJH299</CustomerID>
<status>S</status>
</cus:StatusMsg>
</soapenv:Body>
changed $header
changed $attachments
changed $inbound
CreatePipeline
CustomerOperation

Positive Case - ValidateAddress configuration property value as N:



Negative Case - Status E in business service response:



Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>XYZ-0004: Error in Customer Creation.</faultstring>
            <detail>
                <con:fault xmlns:con="http://www.bea.com/wli/sb/context">
                    <con:errorCode>XYZ-0004</con:errorCode>
                    <con:reason>Error in Customer Creation.</con:reason>
                    <con:location>
                        <con:node>CreateCustomerRoute</con:node>
                        <con:path>response-pipeline</con:path>
                    </con:location>
                </con:fault>
            </detail>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Update Customer

In this section, you will complete message flow for **UpdateCustomer** operation. Since we have discussed each and every step in detail and extensive manner in previous sections, we will go with brief explanation from here onwards.

Validating Payload

Finish **Validation** stage in **UpdateCustomer** operational branch similar to **CreateCustomer** for validating **Payload Structure** using **Validate** activity. But you should select **Customer** element from **$body/UpdateCustomer** in expression builder for **Location** property.

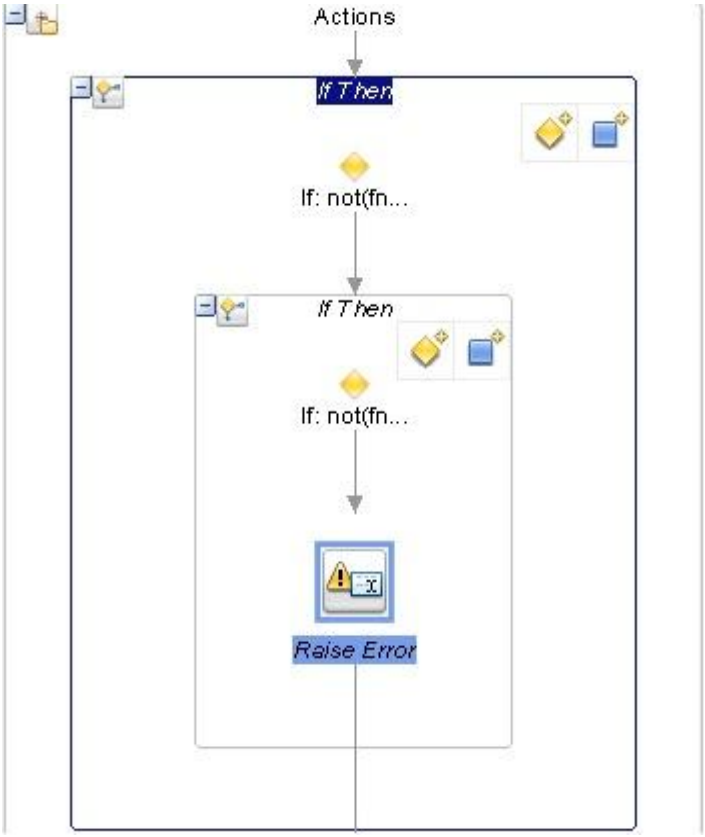
In this case, **Email** element validation has to be done only when it's sent in request. Use **Actions** placeholder in **Validation** stage to do this. Drag two **If-Then** activities as shown below in **Actions** placeholder from **Flow Control**. Give the following expression in top-level **If-Then** condition to find whether **Email** element is available and length is greater than zero.

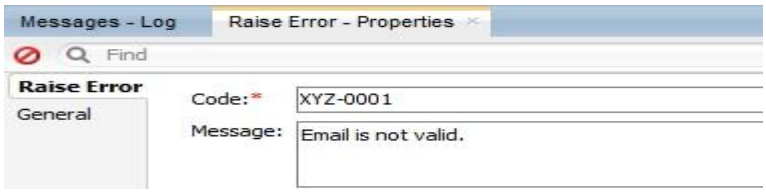
not(fn:empty($body/cus:Customer/Email)) and
fn:string-length(fn-bea:trim($body/cus:Customer/Email/text()))>0

Give the following expression in inner **If-Then** condition to validate against regular expression:

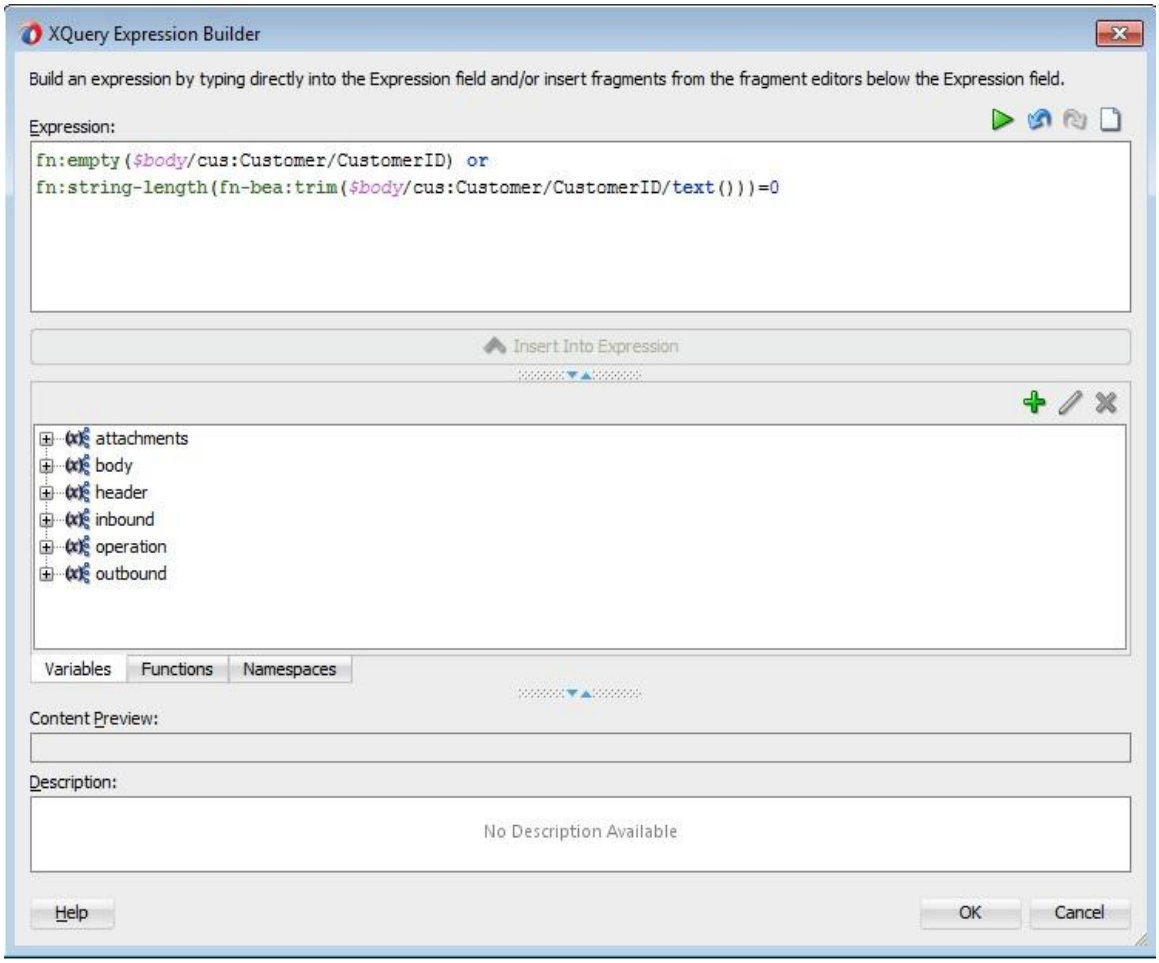
not(fn:matches($body/cus:Customer/Email/text(),
'[a-zA-Z0-9]+@[a-zA-Z]+\.[a-zA-Z]{2,3}','i'))

Also drag **Raise Error** activity into **If** branch from **Flow Control** and set properties as shown below.

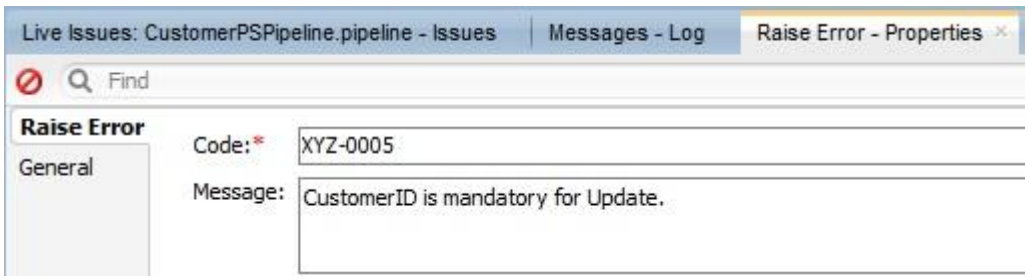




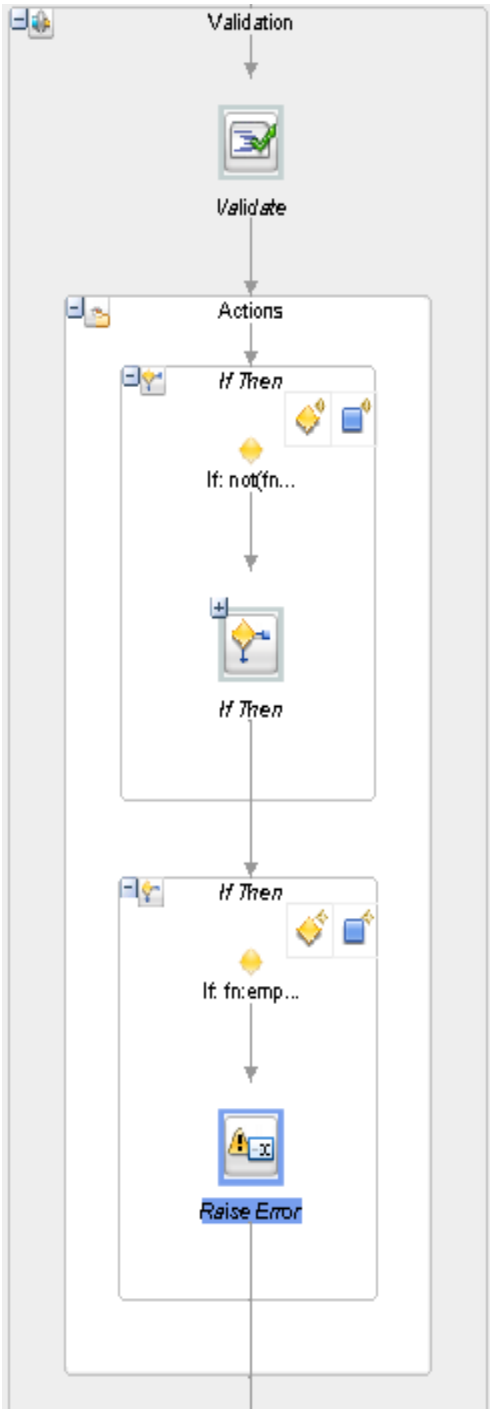
Since this is **Update** operation, service consumer should send **CustomerID** mandatorily in input and your message flow should enforce the same. So drag **If-Then** activity into **Validation** stage from **Flow Control** and set condition as shown below. Alternatively, you can select these string functions from **Functions -> XQuery 1.0 Functions -> Strings -> General Functions.**



Drag **Raise Error** activity into **If** branch from **Flow Control** and set properties as shown below.



Now your **Validation** stage should look like below.

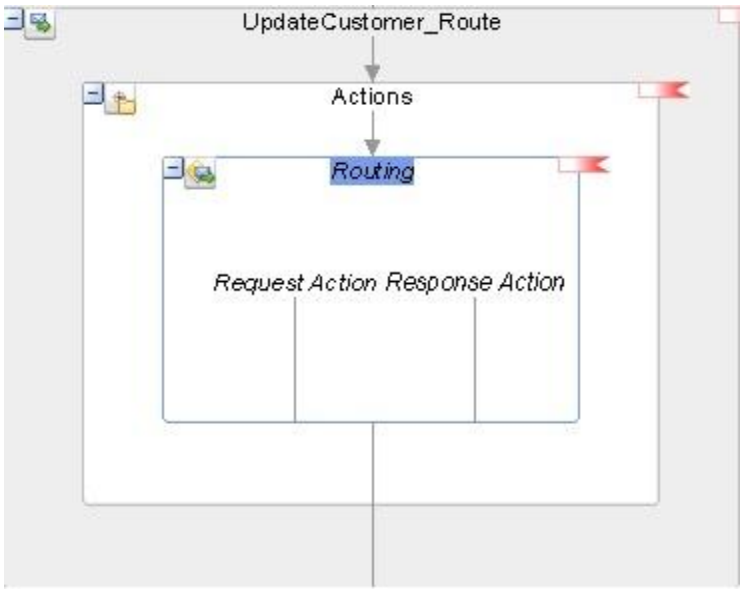


You can also do the **Address Validation** similar to **CreateCustomer** operation, as user can always update the address. But we are not going to discuss this again in this section.

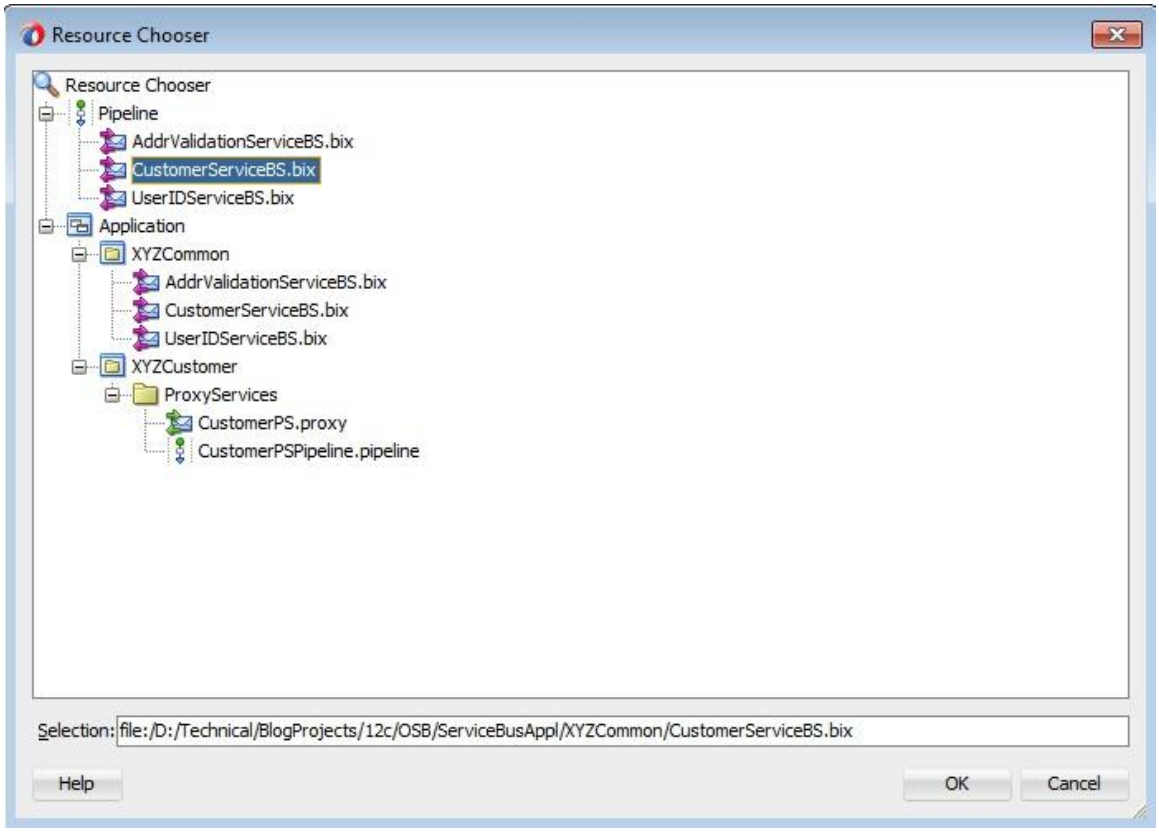
Routing

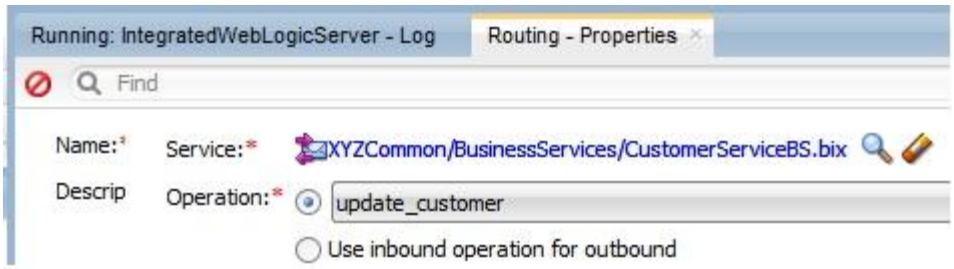
In this section, you will invoke **CustomerServiceBS** to finish message flow for **Update Customer.**

Drag **Routing** activity into **Actions** placeholder of **UpdateCustomer_Route**.

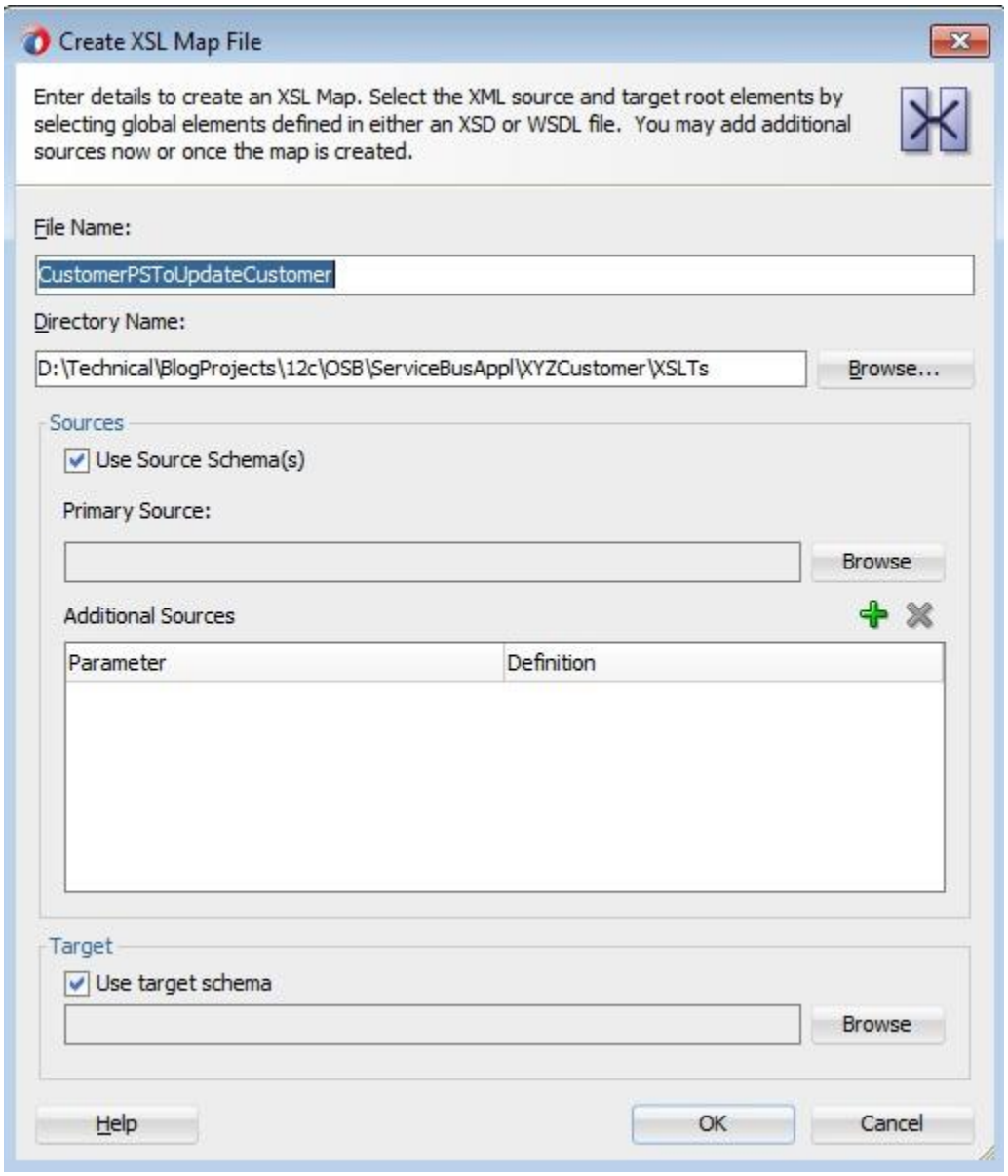


In **Properties** tab, browse and select **CustomerServiceBS** and **update_customer** operation.

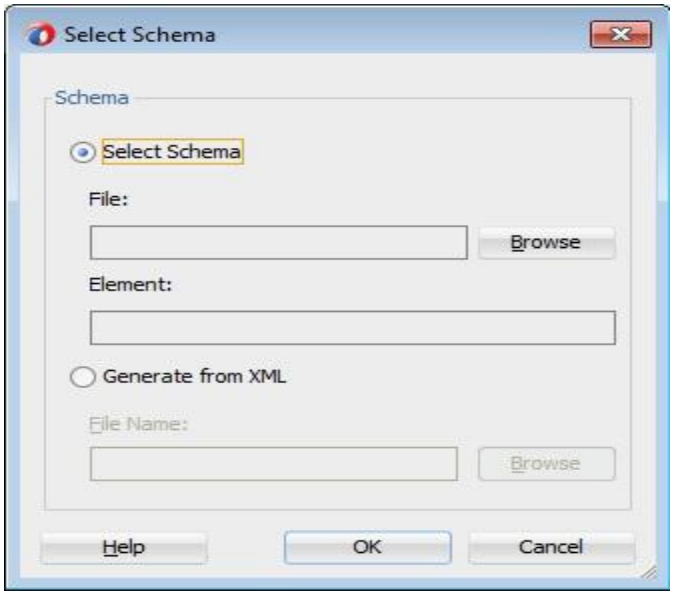




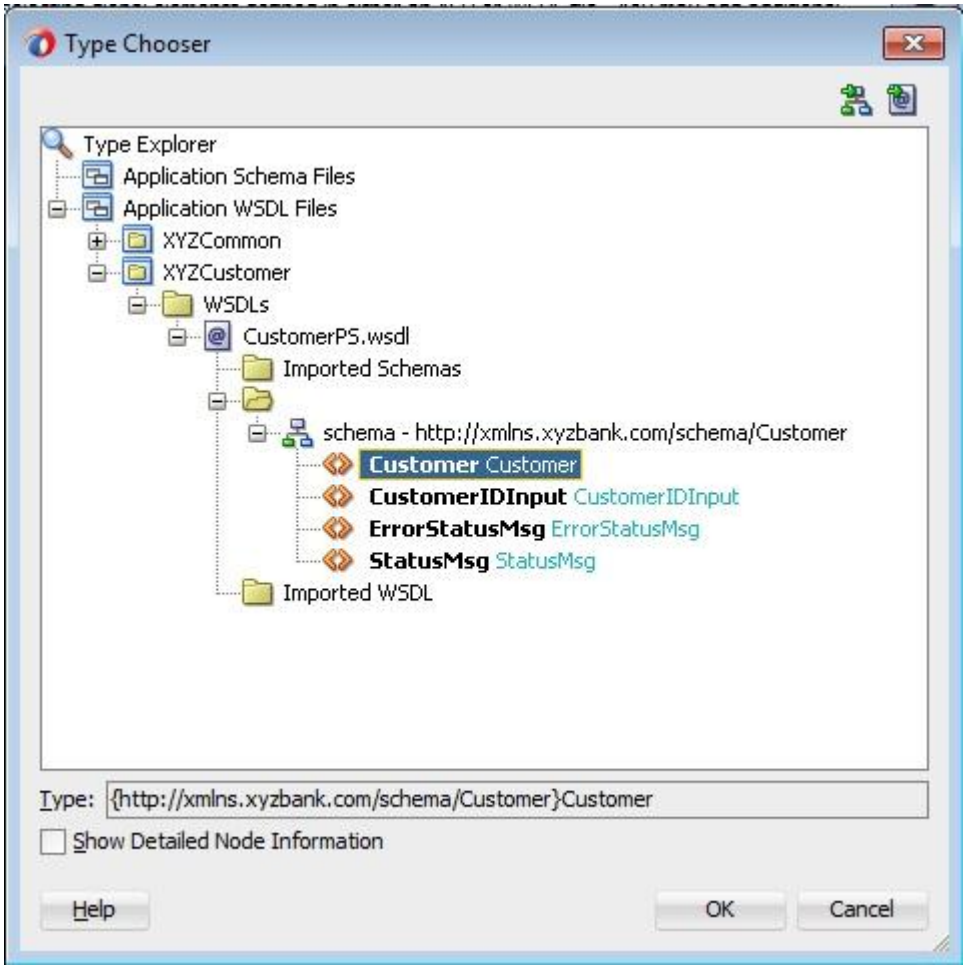
In this section, you will use XSLT Map instead of XQuery for payload transformation. Right click **XSLTs** folder in **XYZCustomer** project and select **New -> XSL Map**. In Create XSL Map File window give **File Name** as **CustomerPSToUpdateCustomer**.



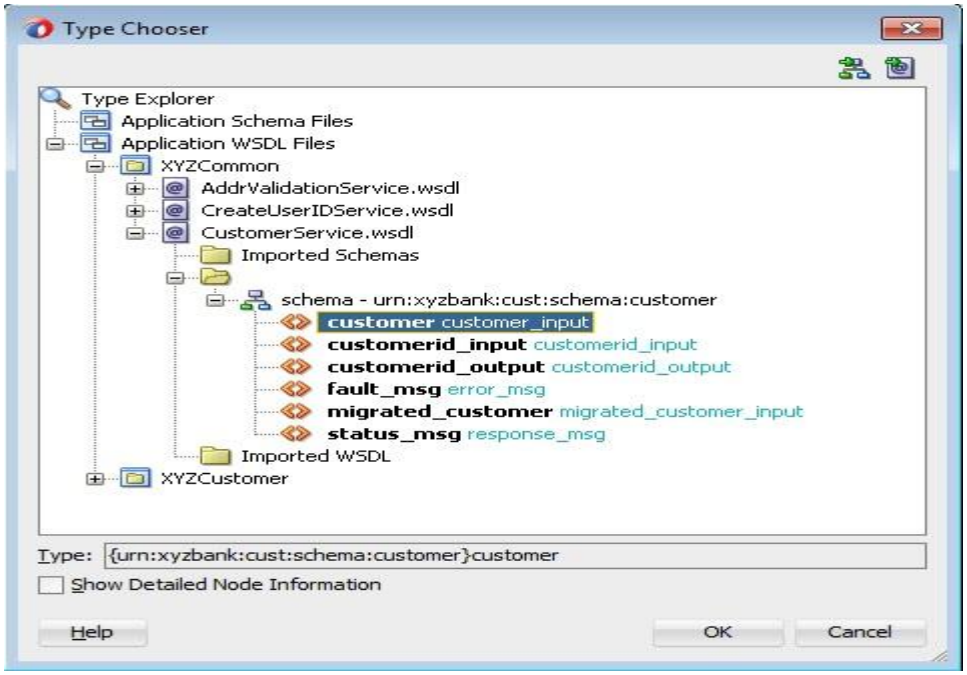
Click **Browse** for **Primary Source** to bring up **Select Schema** screen.



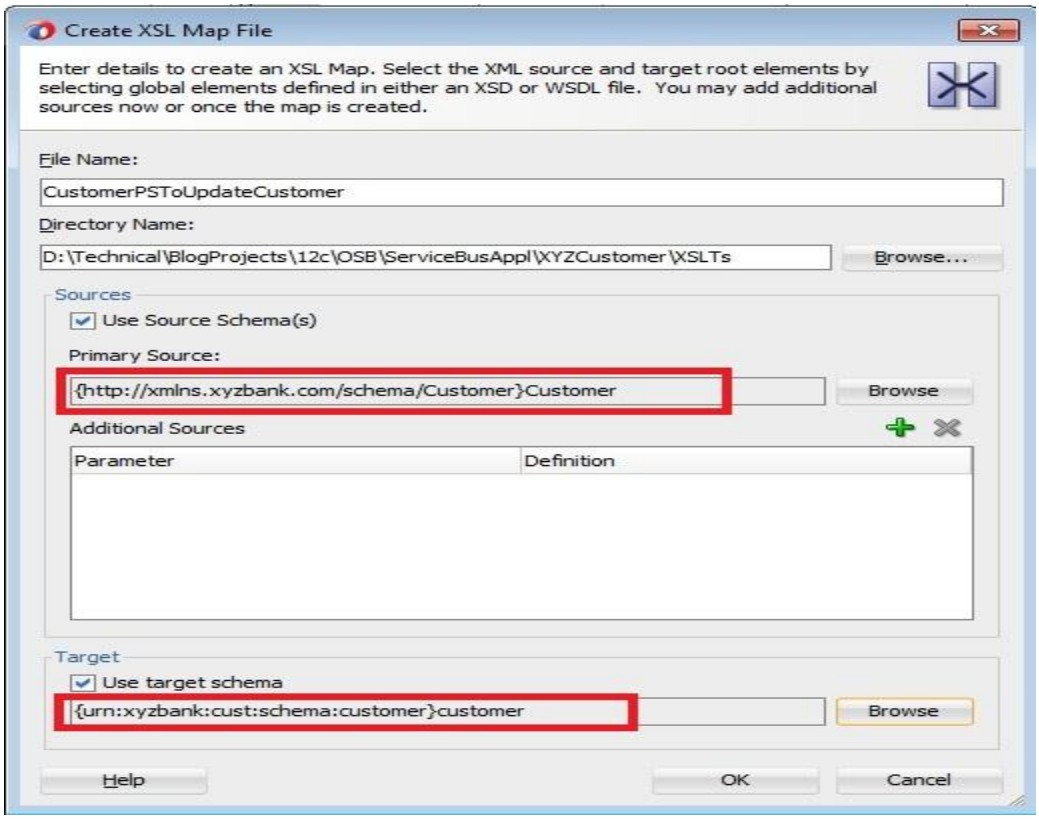
Retain the default selection and click **Browse** to select primary source element as shown below.



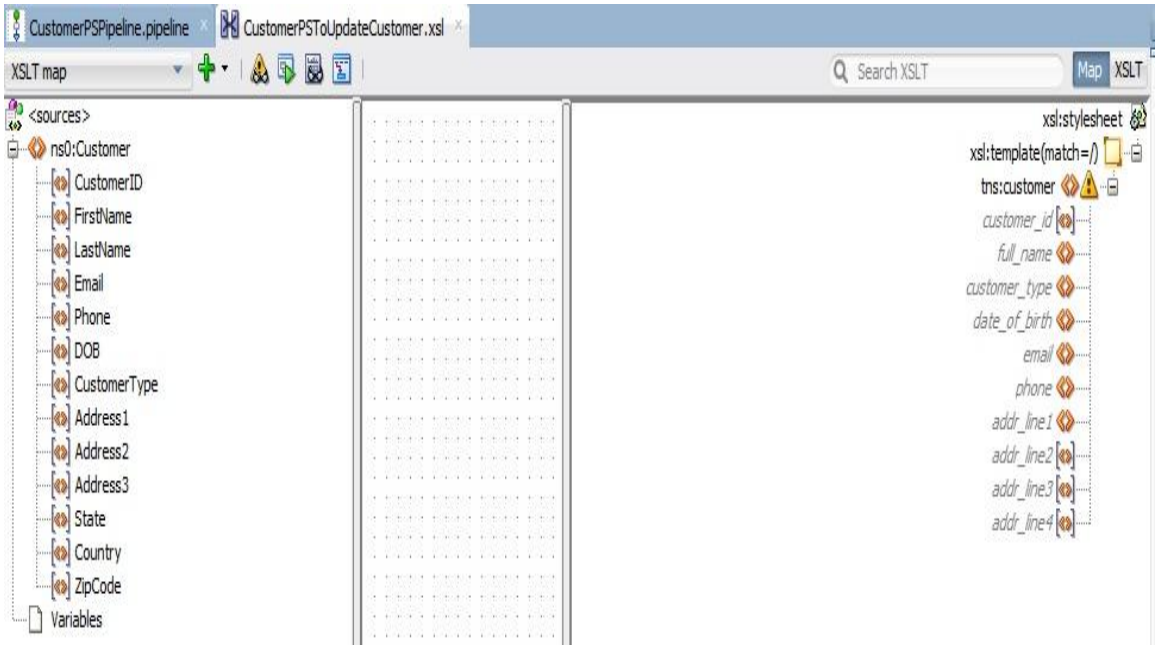
Go back to **Create XSL Map** window by clicking **OK** twice. Now click **Browse** for **Target** to bring up **Select Schema** screen. And click **Browse** for **Select Schema** and select the target element as shown below.



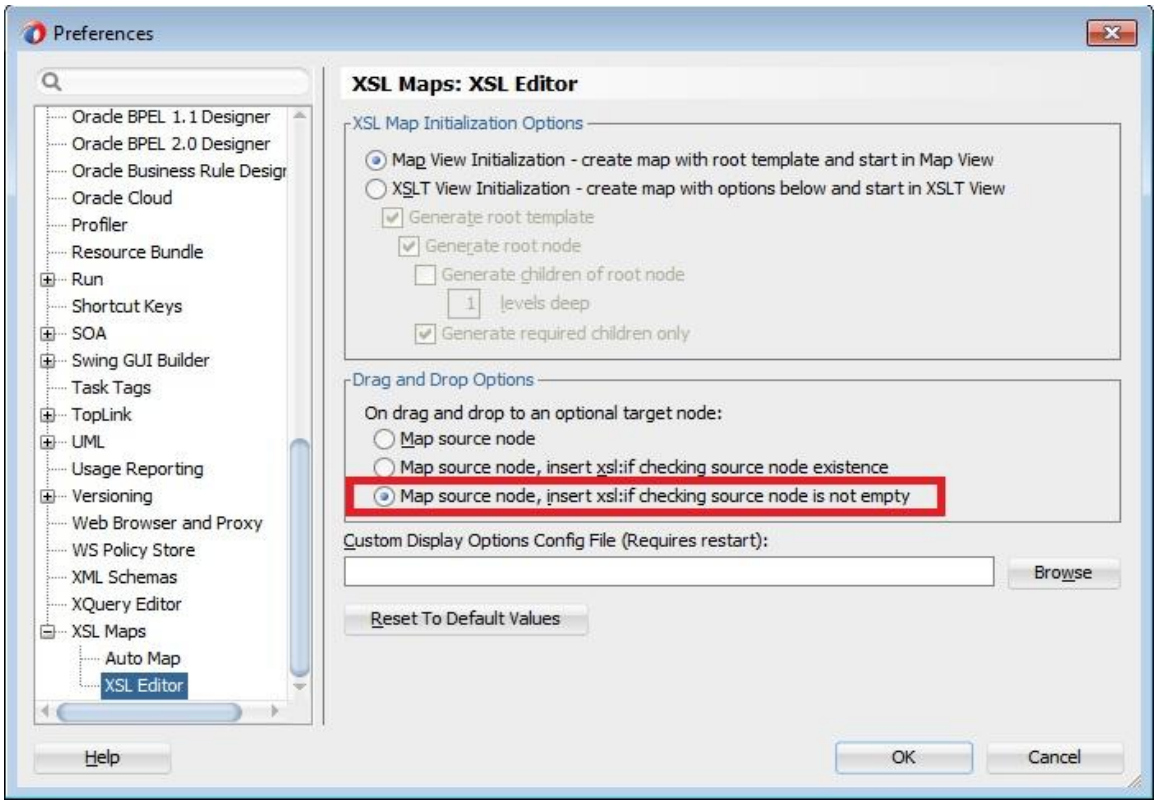
Go back to **Create XSL Map** screen by click **OK** twice and observe Source and Target elements.



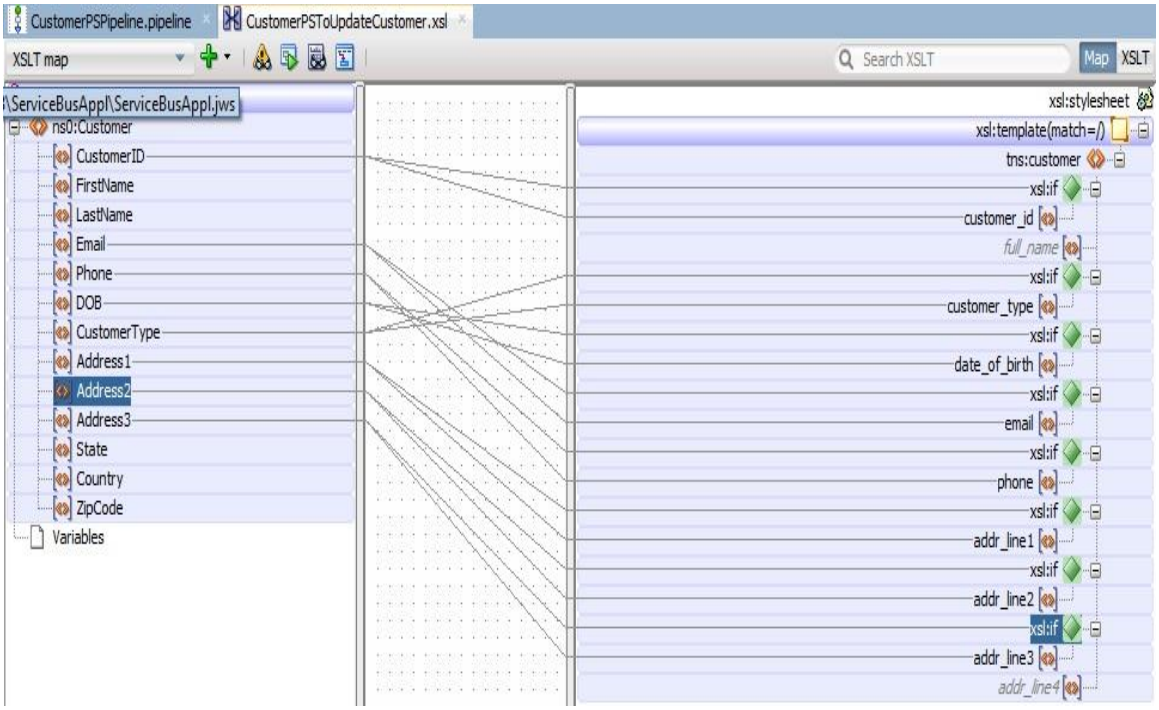
Click **OK** to finish the creation which brings up XSLT Transformation editor as shown below.



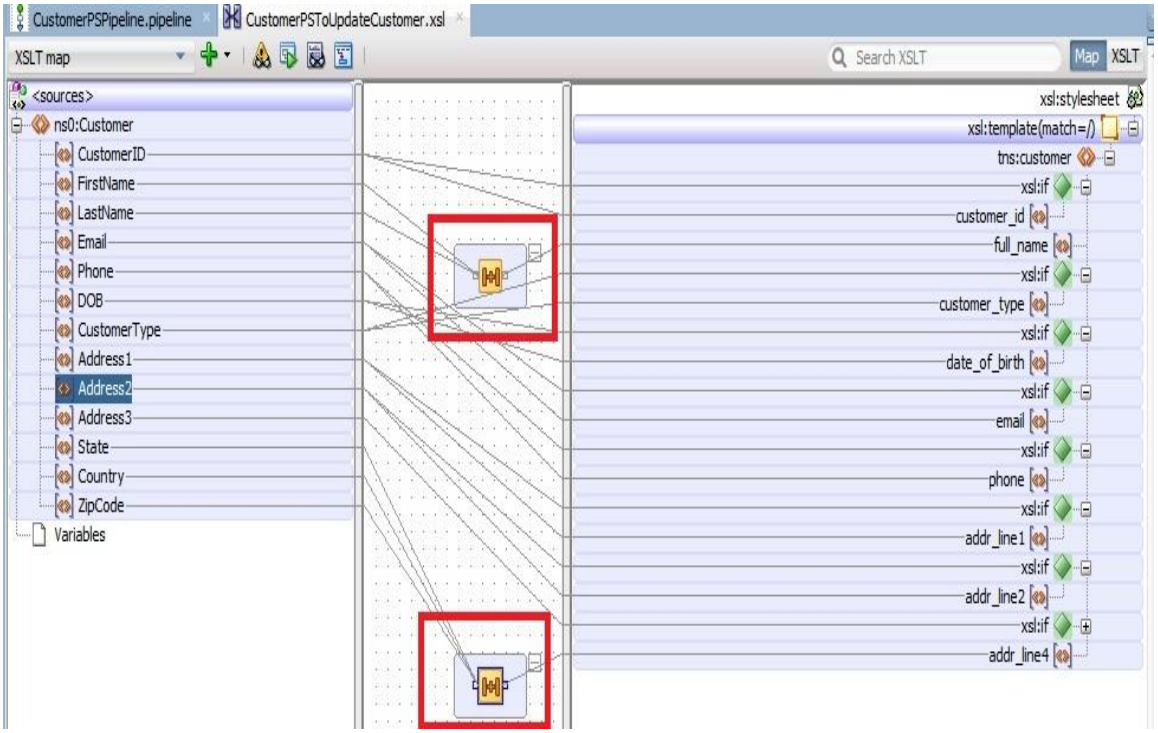
Since we are doing update of customer information, you should not send null values to business service; hence target elements should be mapped only when corresponding source element is available. You can achieve this behavior automatically in **XSL Map editor** by setting a preference. Go to **Tools -> Preferences** and set property as shown below.



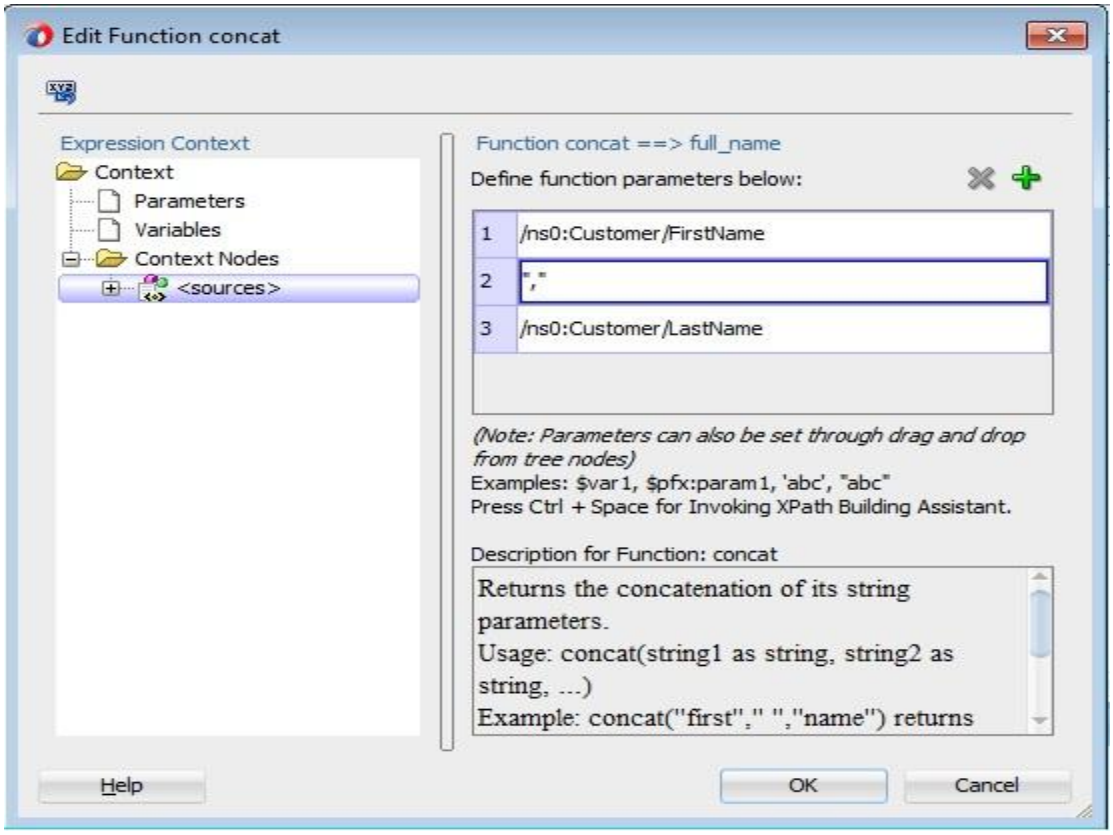
Go back to **XSL Map editor** and finish the initial mapping by mapping respective source and target elements as shown below. Observe that **xsl:if** condition gets added automatically.

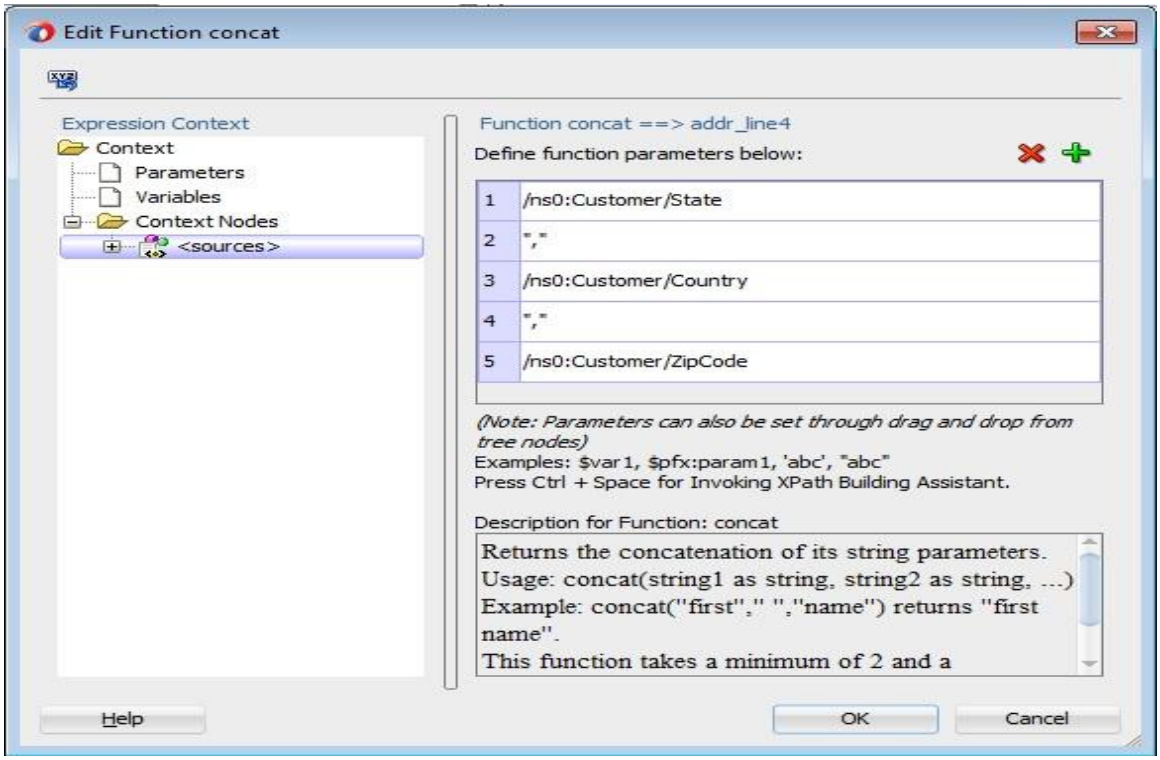


Drag **concat** function into middle swim lane from **Components-> General XPath -> String Functions** to concatenate **FirstName, Lastame** and map to **full_name**. Also concatenate **State, Country and ZipCode** and map to **addr_line4** as shown below.



Click on **concat** functions and modify them as shown below by adding new elements. This would append **‘,’** between the elements and is expected by business service.



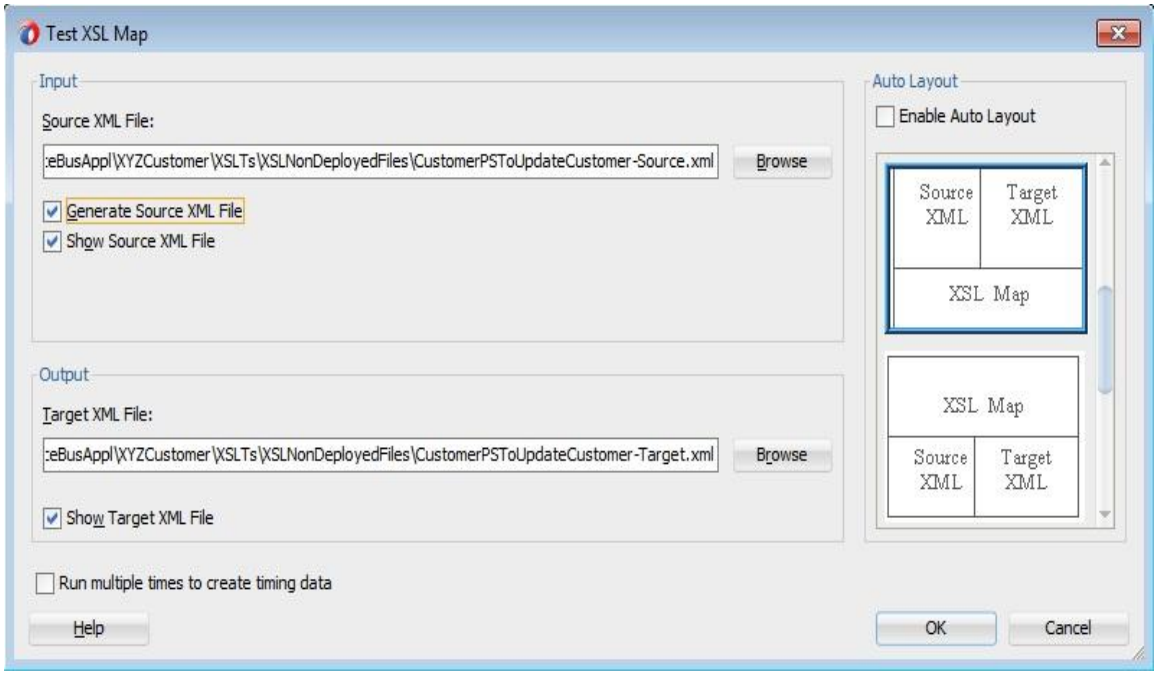


Now go to **Source** for XSL map. Modify **full_name** and **addr_line4** elements mapping to include **xsl:if** condition so that target will be mapped only when all of corresponding source elements are present.

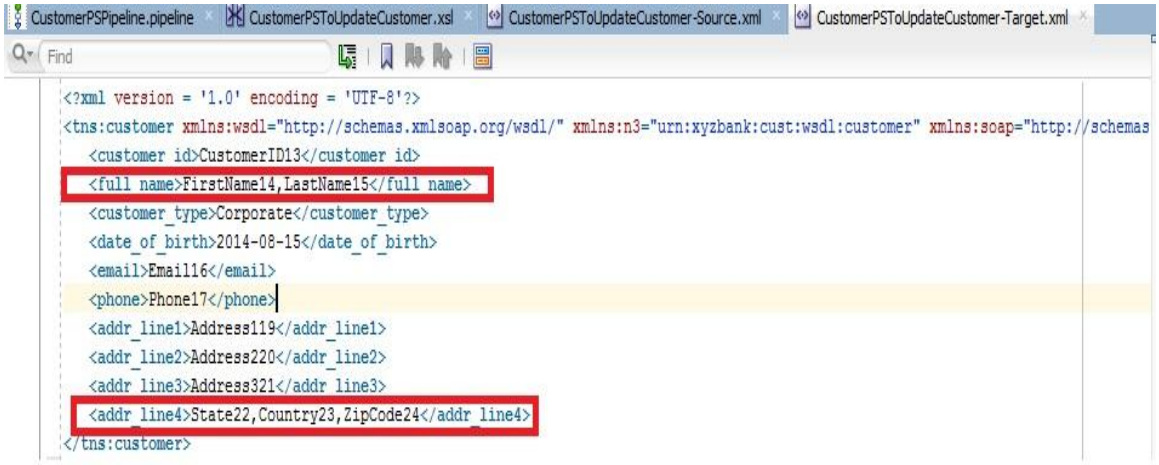
```
<xsl:template match="/">
  <tns:customer>
    <xsl:if test="/ns0:Customer/CustomerID">
      <customer_id>
        <xsl:value-of select="/ns0:Customer/CustomerID"/>
      </customer_id>
    </xsl:if>
    <xsl:if test="/ns0:Customer/FirstName and /ns0:Customer/LastName">
    <full_name>
      <xsl:value-of select='concat (/ns0:Customer/FirstName, ",", /ns0:Customer/LastName )'/>
    </full_name>
    </xsl:if>
    <xsl:if test="/ns0:Customer/CustomerType">
      <customer_type>
        <xsl:value-of select="/ns0:Customer/CustomerType"/>
      </customer_type>
    </xsl:if>
```

```
    <xsl:if test="/ns0:Customer/Address3">
      <addr_line3>
        <xsl:value-of select="/ns0:Customer/Address3"/>
      </addr_line3>
    </xsl:if>
    <xsl:if test="/ns0:Customer/State and /ns0:Customer/Country and /ns0:Customer/ZipCode">
    <addr_line4>
      <xsl:value-of select='concat (/ns0:Customer/State, ",", /ns0:Customer/Country, ",", /ns0:Customer/ZipCode )'/>
    </addr_line4>
    </xsl:if>
  </tns:customer>
</xsl:template>
</xsl:stylesheet>
```

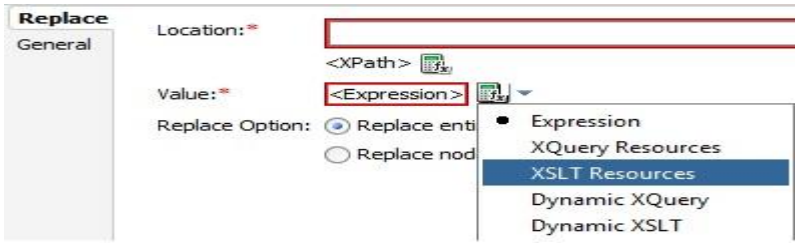
It's always advisable to test your XSL map to see whether mapping is working as expected. Click **Test XSL Map** icon in editor to bring up the following screen.

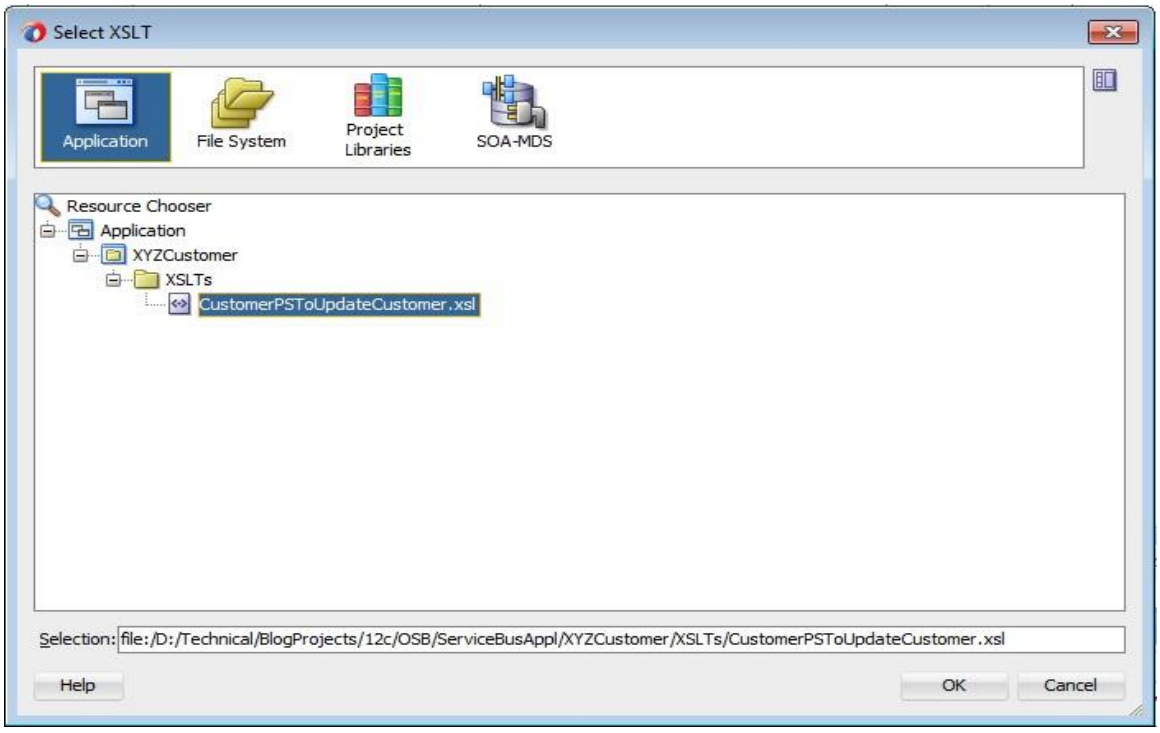


Click **OK** and observe both source and target xml files get created. You can also modify source xml and retest your XSL map again.

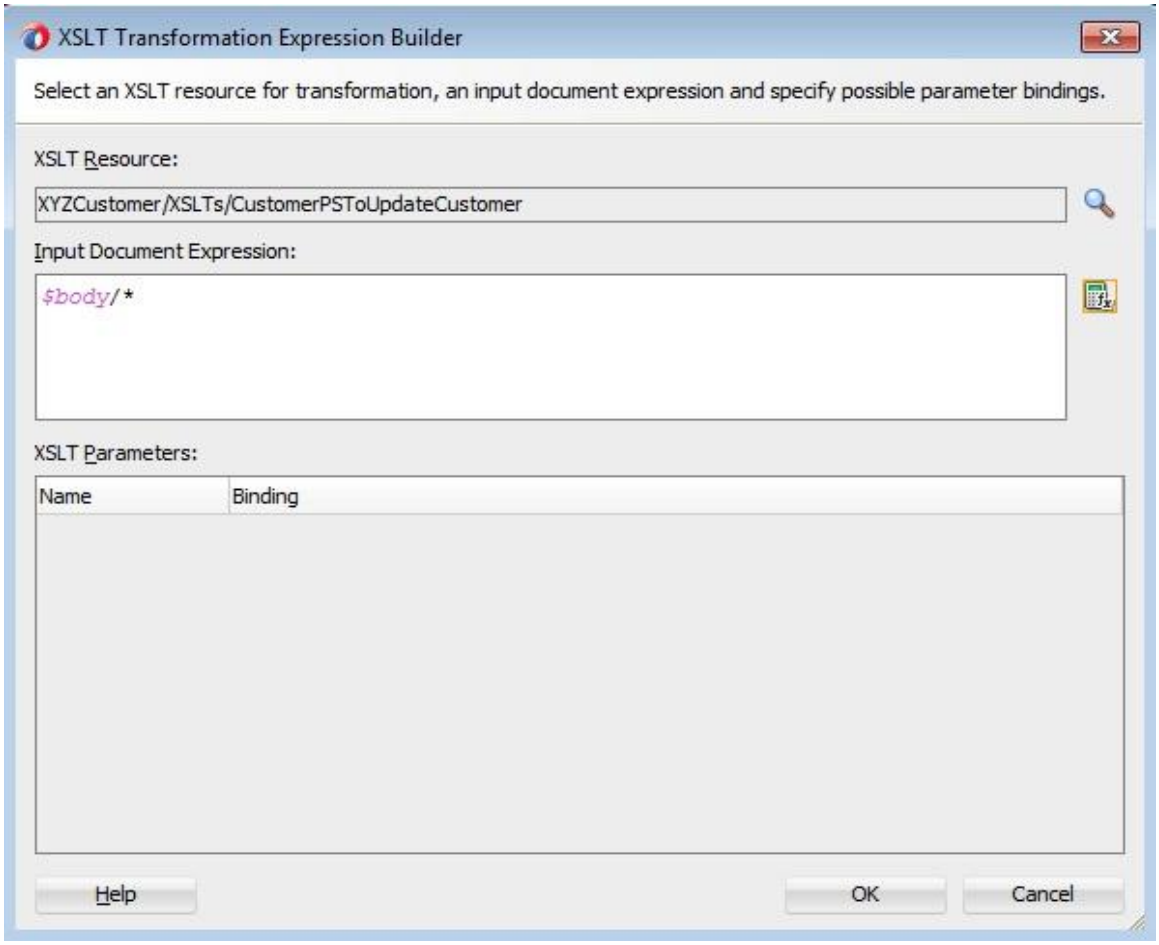


Drag **Replace** activity into **Request Action** of **Routing** from **Message Processing**. In **Properties** tab, bring up expression editor for **Value** property and choose XSL map as shown below.

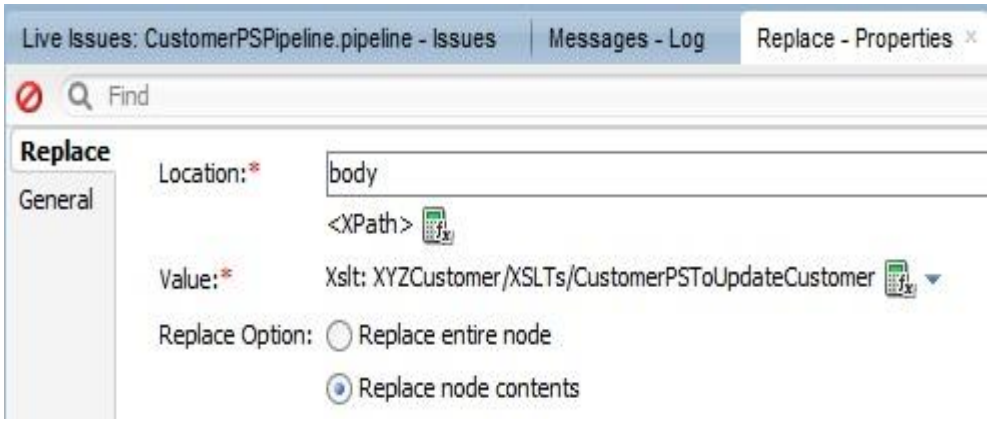




Click **OK** and give **Input Document Expression** as **$body/***.

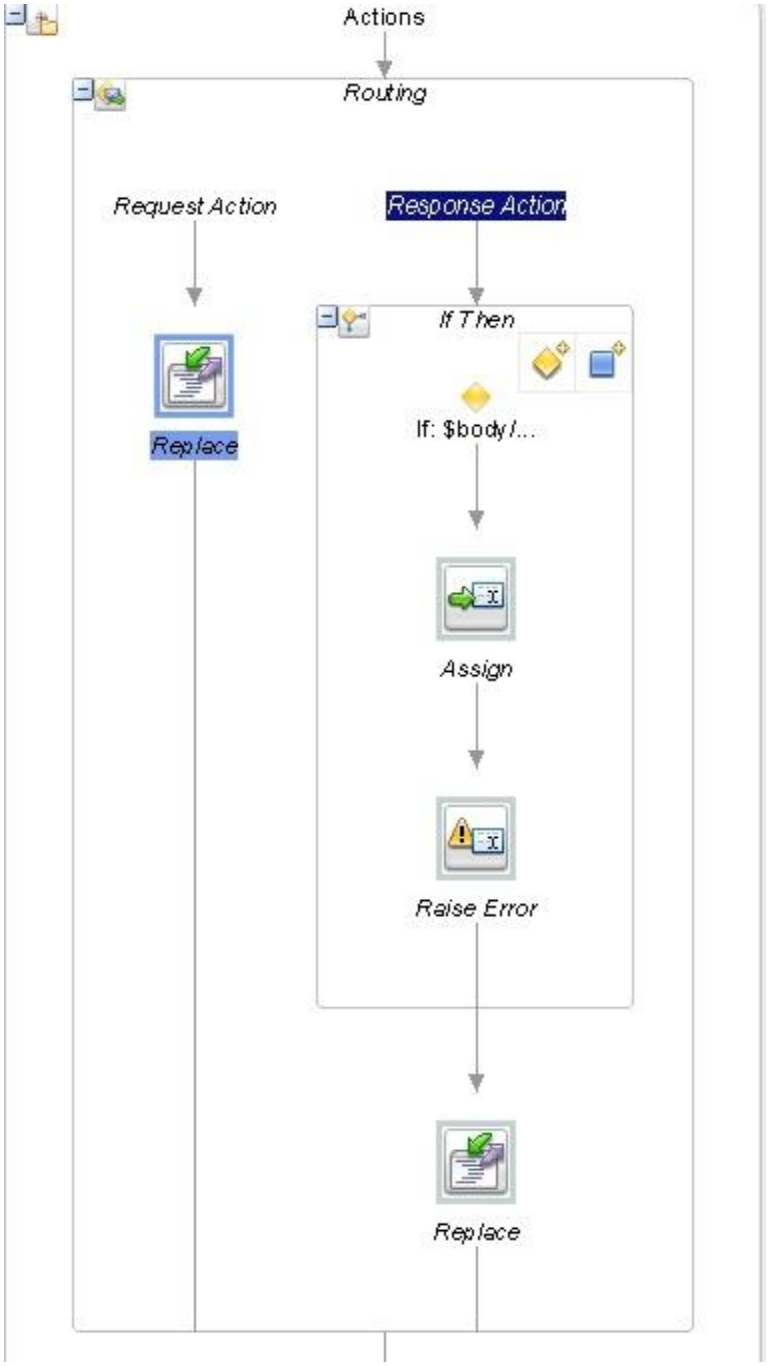


Set other properties as shown below. This action will replace contents of **$body** context variable with payload structure required by business service.



Finish **Response Action** in **Routing** similar to **CreateCustomer** as response structure of business service is same for all the operations. So you can reuse **CustStatusMsgToCustomerPSOutput XQuery Map.**

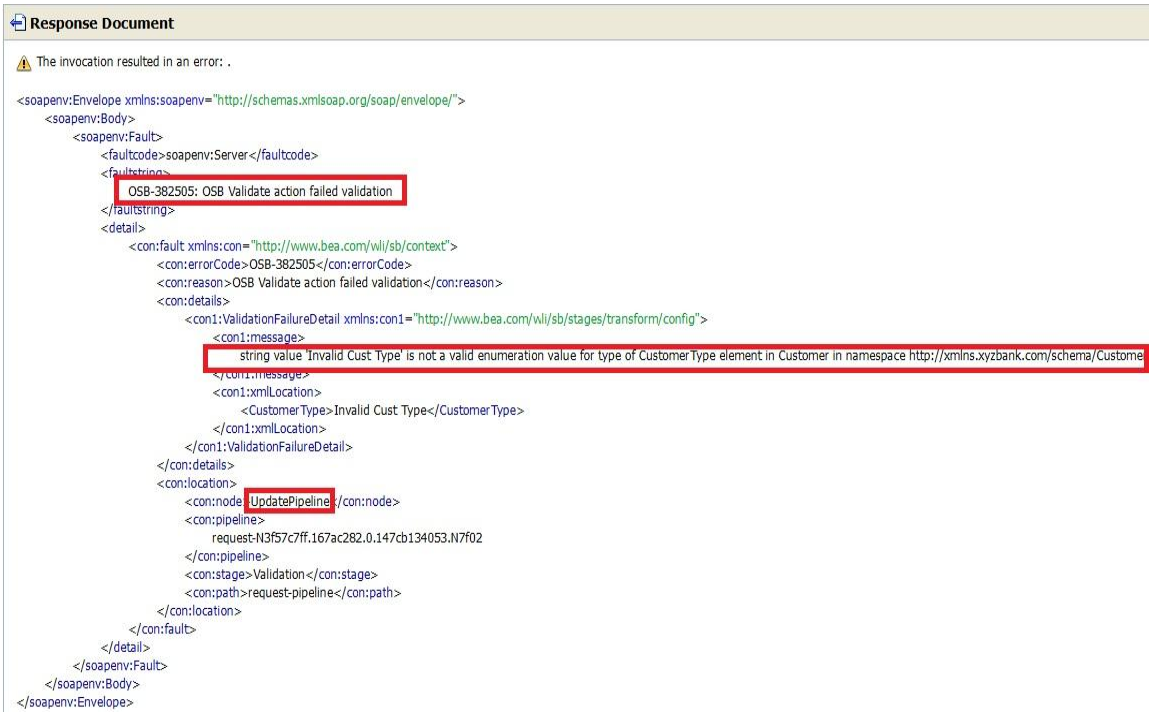
Now your **Routing** node should look like below.



Testing

Deploy both projects or run **Pipeline** directly as shown in **Deploying and Testing** section. You may want to frequently test your proxy service/pipeline during development. Run your **pipeline** with sample payloads given here and observe **Flow Trace and Variables** as shown below. You can also run Proxy Service but you will not observe any **Flow Trace**.

CustomerType element validation:



Email element validation:

CustomerID element validation:

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>
                XYZ-0005: CustomerID is mandatory for Update.
            </faultstring>
            <detail>
                <con:fault xmlns:con="http://www.bea.com/wli/sb/context">
                    <con:errorCode>XYZ-0005</con:errorCode>
                    <con:reason>CustomerID is mandatory for Update.</con:reason>
                    <con:location>
                        <con:node>UpdatePipeline</con:node>
                        <con:pipeline>
                            request-N3f57c7ff.167ac282.0.147cb134053.N7f02
                        </con:pipeline>
                        <con:stage>Validation</con:stage>
                        <con:path>request-pipeline</con:path>
                    </con:location>
                </con:fault>
            </detail>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Positive Case - FirstName, LastName, Phone, DOB and Email:

UpdateCustomerRoute

Routed Service

Route to: "CustomerServiceBS"

$outbound:

$body (request):

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <ns0:customer xmlns:ns0="urn:xyzbank:cust:schema:customer">
        <customer_id>AHHHHH8787QDJHJH299</customer_id>
        <full_name>S.N.R. RAO</full_name>
        <date_of_birth>1986-09-29</date_of_birth>
        <email>abcnew@gmail.com</email>
        <phone>+91 8888888888</phone>
    </ns0:customer>
</soapenv:Body>
```

$header (request):

$attachments (request):

Message Context Changes

added $outbound

changed $body

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:xyzbank:cust:schema:customer">
    <cus:StatusMsg xmlns:cus="http://xmlns.xyzbank.com/schema/Customer">
        <CustomerID>AHHHHH8787QDJHJH299</CustomerID>
        <status>S</status>
    </cus:StatusMsg>
</soapenv:Body>
```

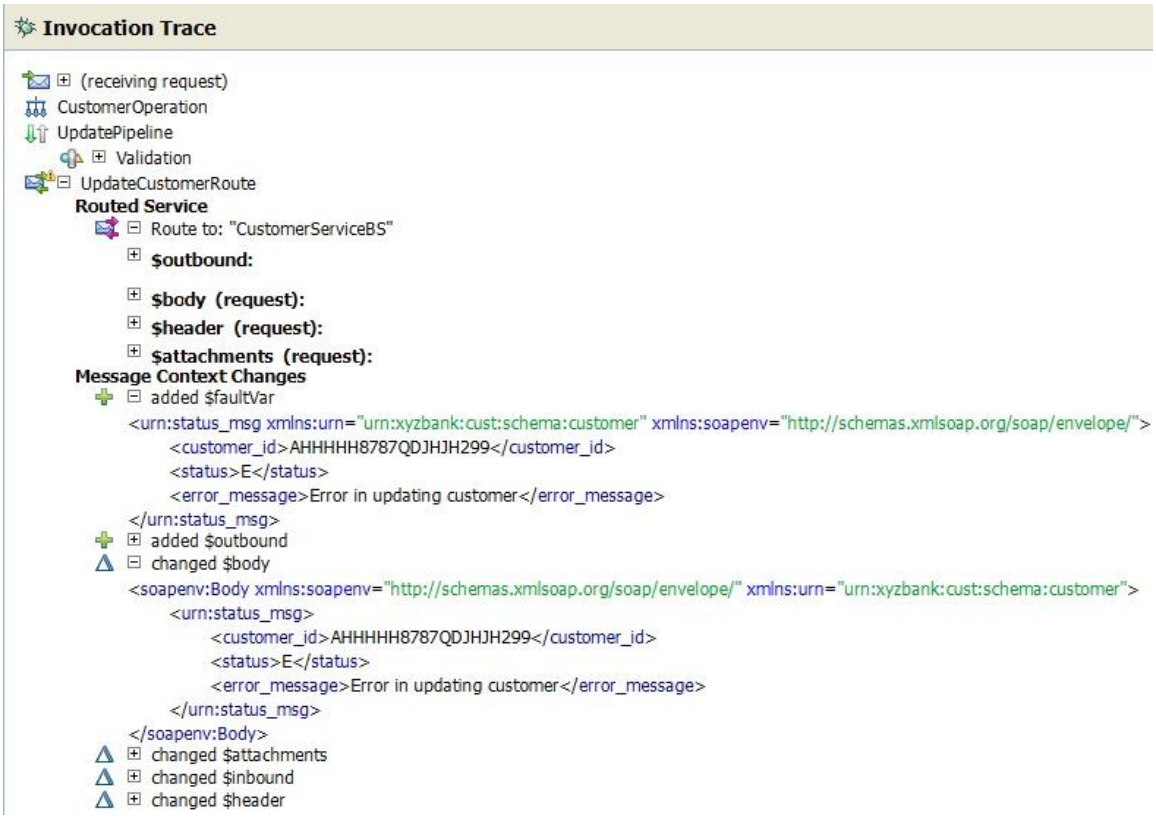
changed $attachments

changed $inbound

changed $header

Positive Case - Address fields update:

Negative Case - Status in business service response:



Response Document

The invocation resulted in an error: .

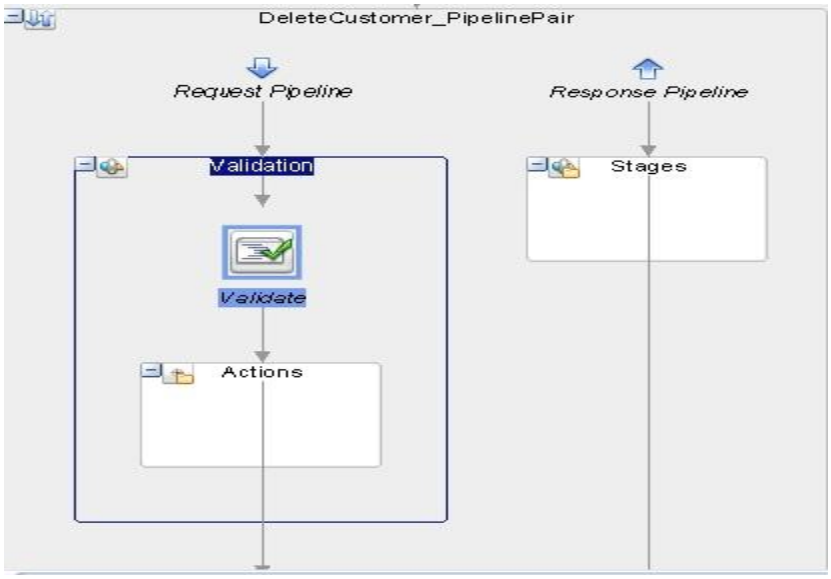
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>XYZ-0004: Error in Customer Update.</faultstring>
            <detail>
                <con:fault xmlns:con="http://www.bea.com/wli/sb/context">
                    <con:errorCode>XYZ-0004</con:errorCode>
                    <con:reason>Error in Customer Update.</con:reason>
                    <con:location>
                        <con:node>UpdateCustomerRoute</con:node>
                        <con:path>response-pipeline</con:path>
                    </con:location>
                </con:fault>
            </detail>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Delete Customer

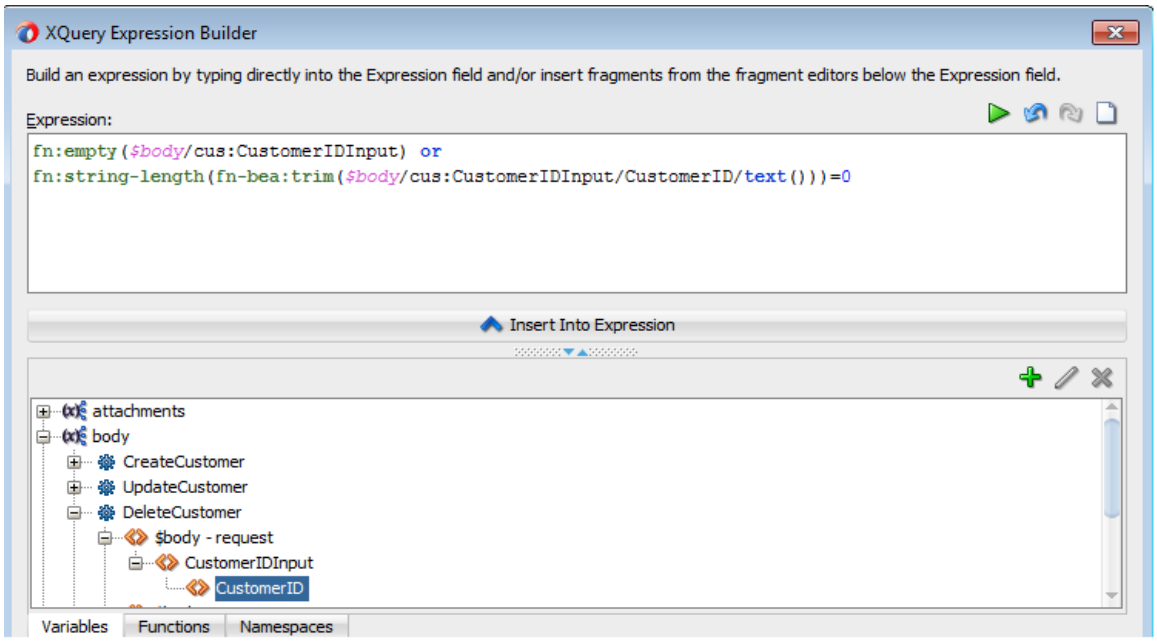
In this section, you will complete message flow for **DeleteCustomer** operation.

Validating Payload

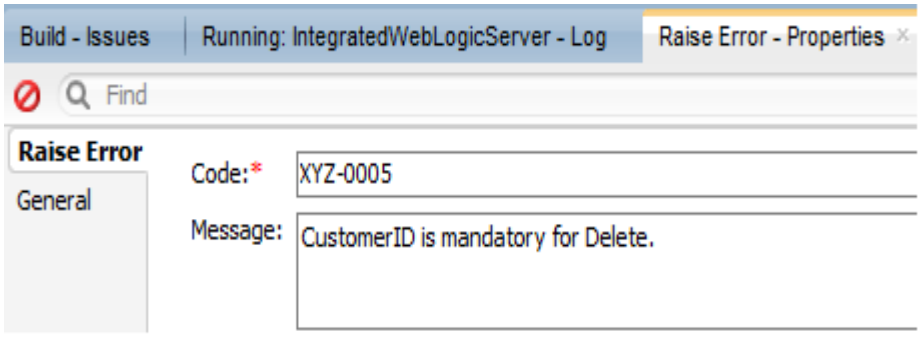
Finish **Validation** stage in **DeleteCustomer** operational branch similar to **CreateCustomer** for validating **Payload Structure** using **Validate** activity. But you should select **CustomerIDInput** element from **$body/DeleteCustomer** in expression builder for **Location** property.



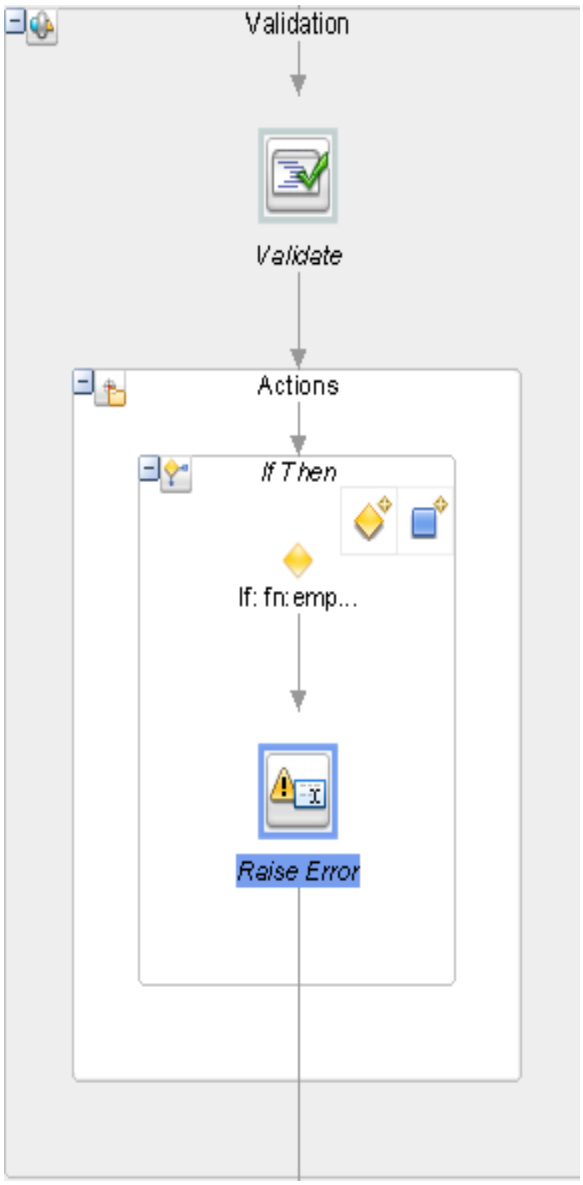
Since this is **Delete** operation, service consumer should send **CustomerID** mandatorily in input and your message flow should enforce the same. So drag **If-Then** activity into **Validation** stage from **Flow Control** and set condition as shown below. Alternatively, you can select these string functions from **Functions -> XQuery 1.0 Functions -> Strings -> General Functions**.



Drag **Raise Error** activity into **If** branch from **Flow Control** and set properties as shown below.



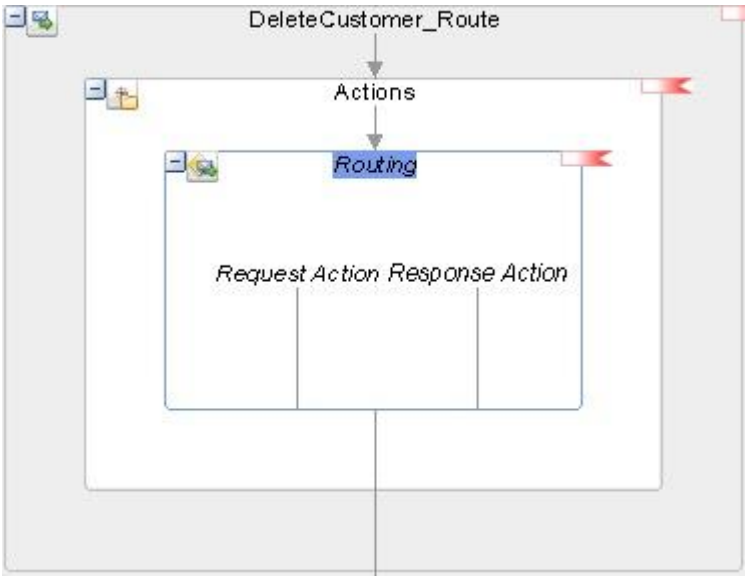
Now your **Validation** stage should look like below.



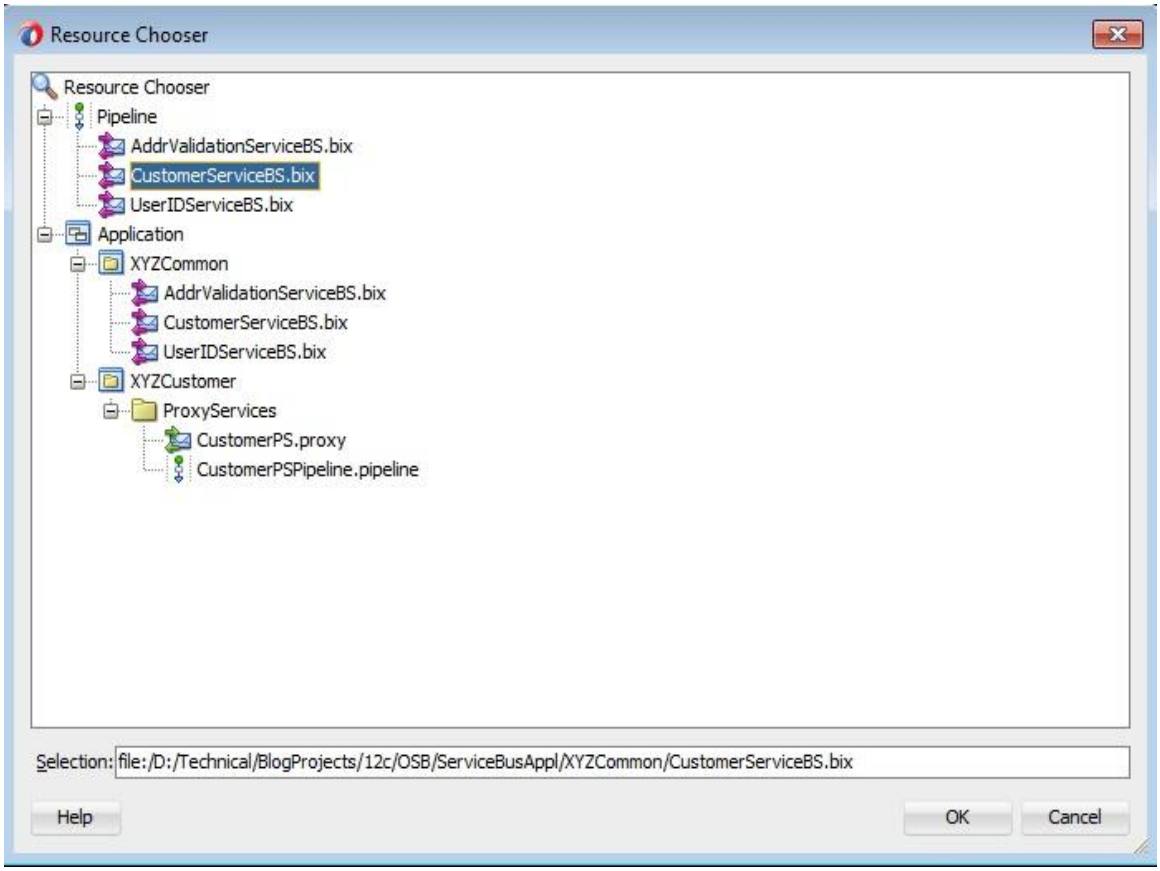
Routing

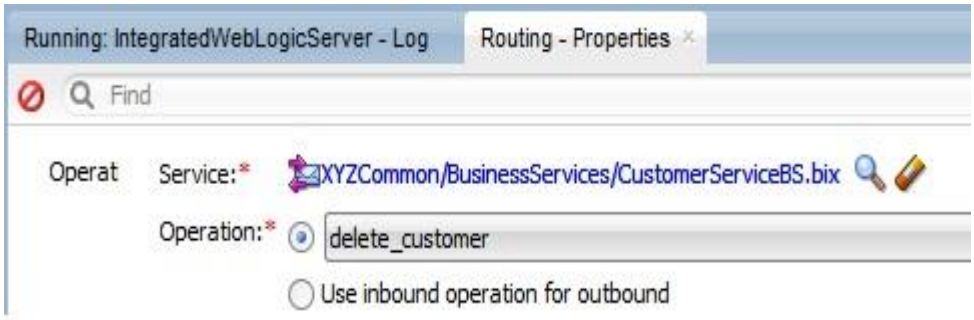
In this section, you will invoke **CustomerServiceBS** to finish message flow for **Delete Customer**.

Drag Routing activity into Actions placeholder of **DeleteCustomer_Route**.

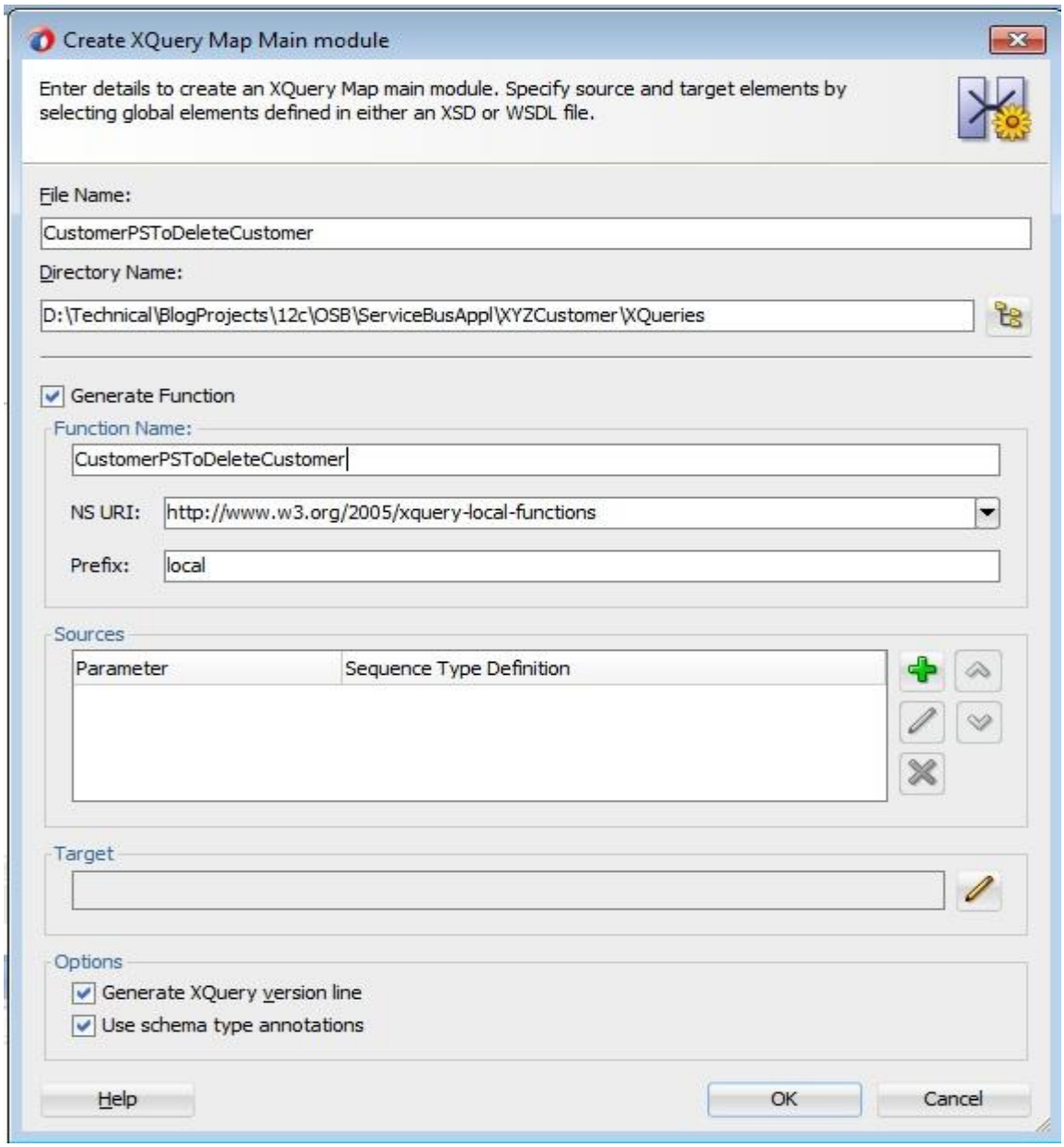


In **Properties** tab, browse and select **CustomerServiceBS** and **delete_customer** operation.

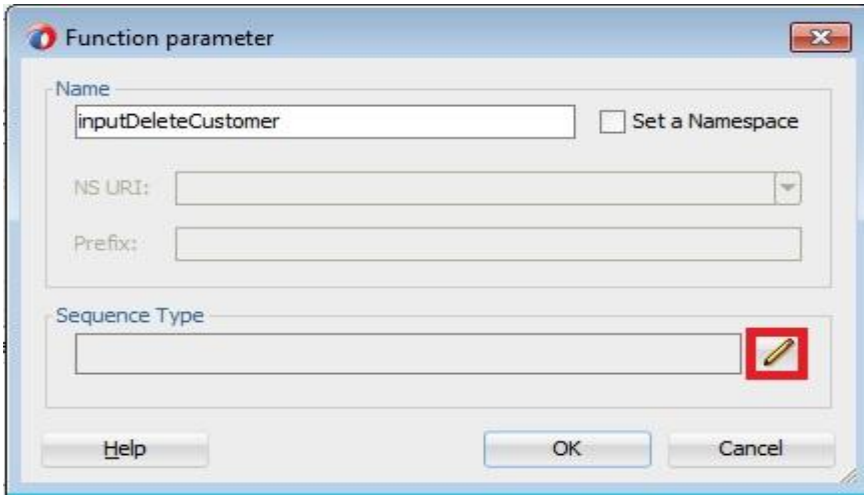


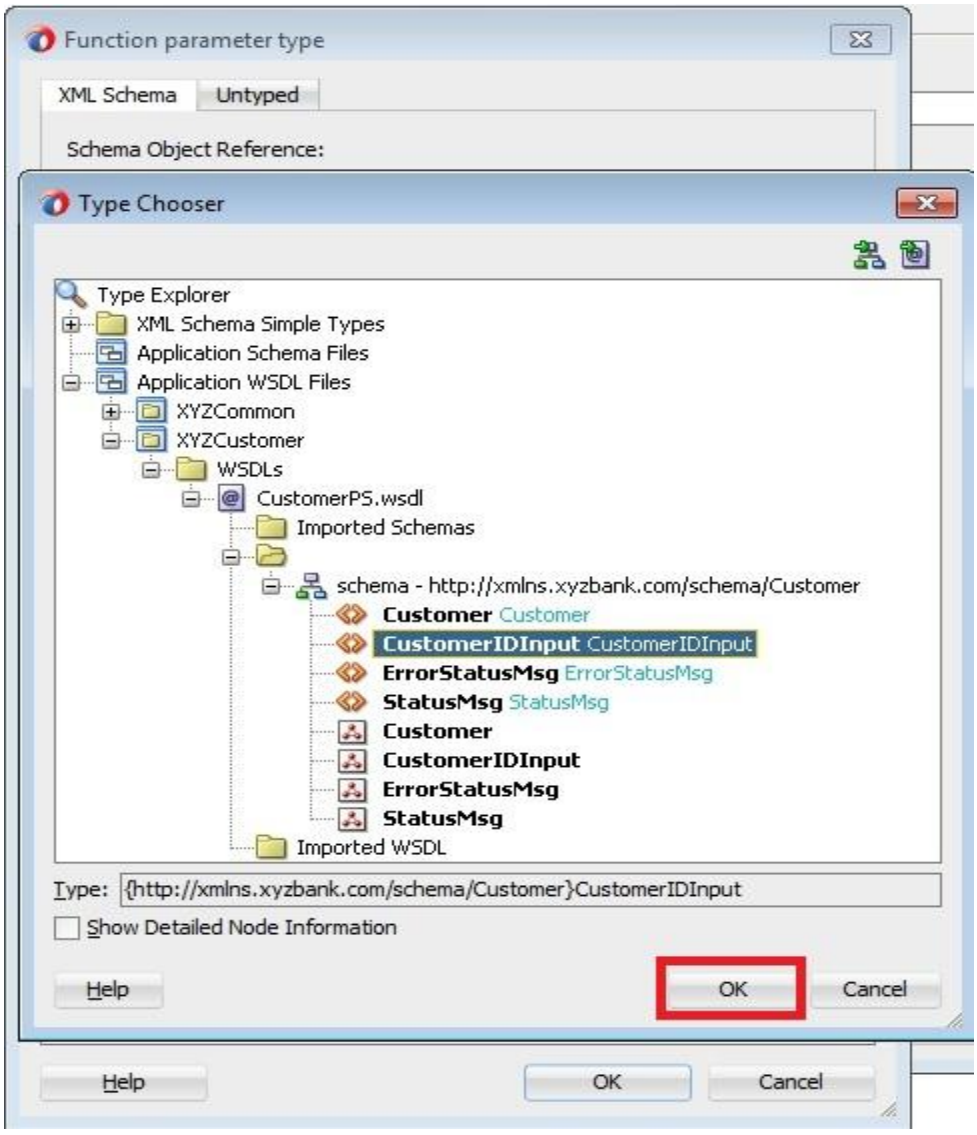


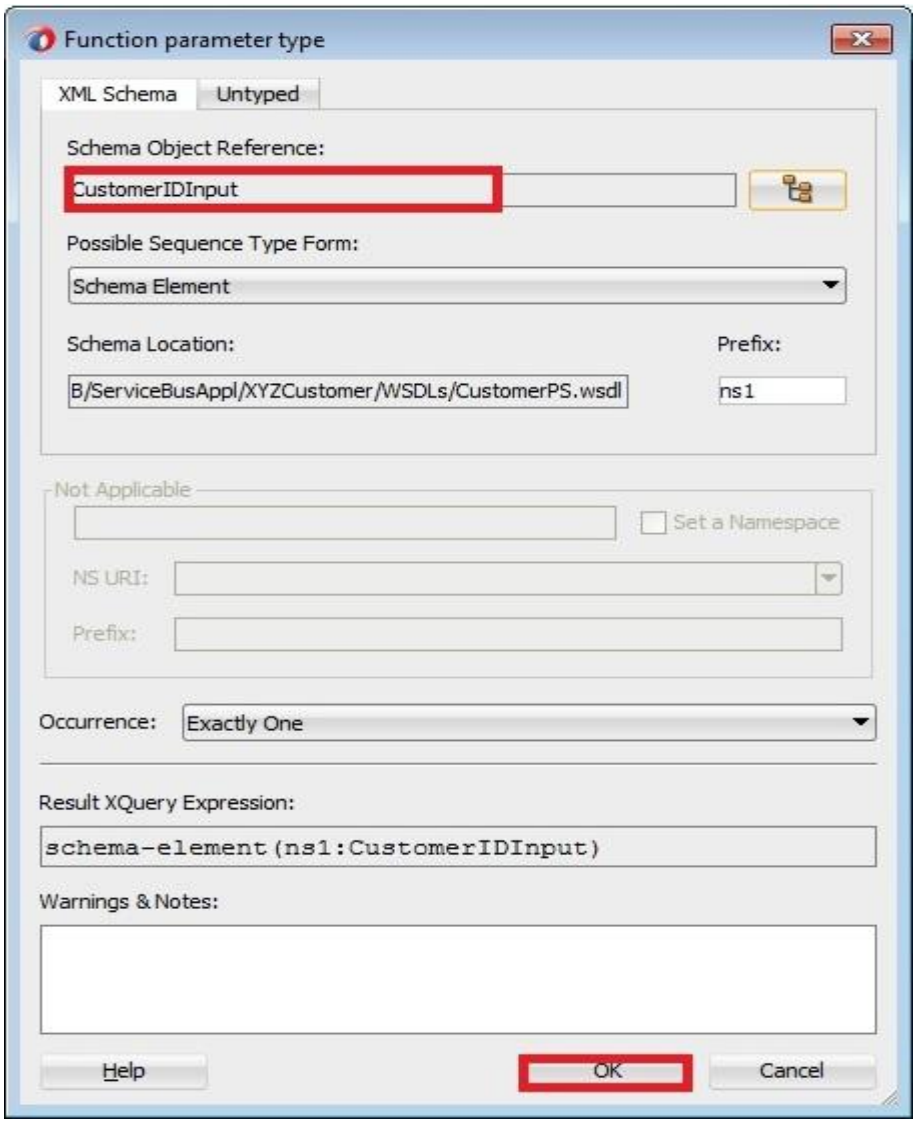
Create a new XQuery map in **XQueries** folder of **XYZCustomer** project and give both **File Name** and **Function Name** as **CustomerPSToDeleteCustomer**.

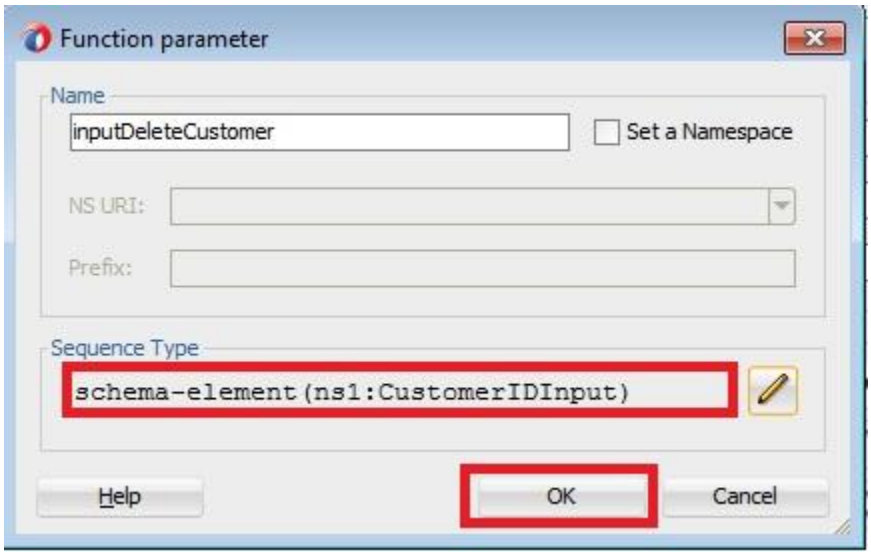


Complete **Source Parameters** selection as shown below.

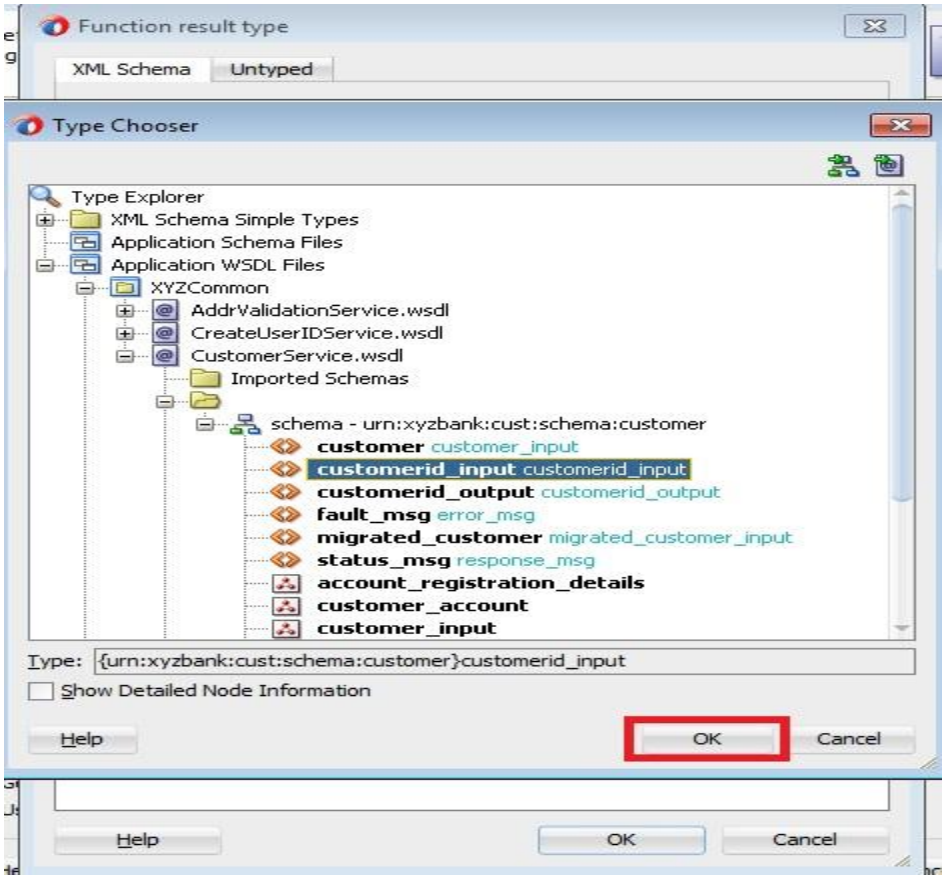


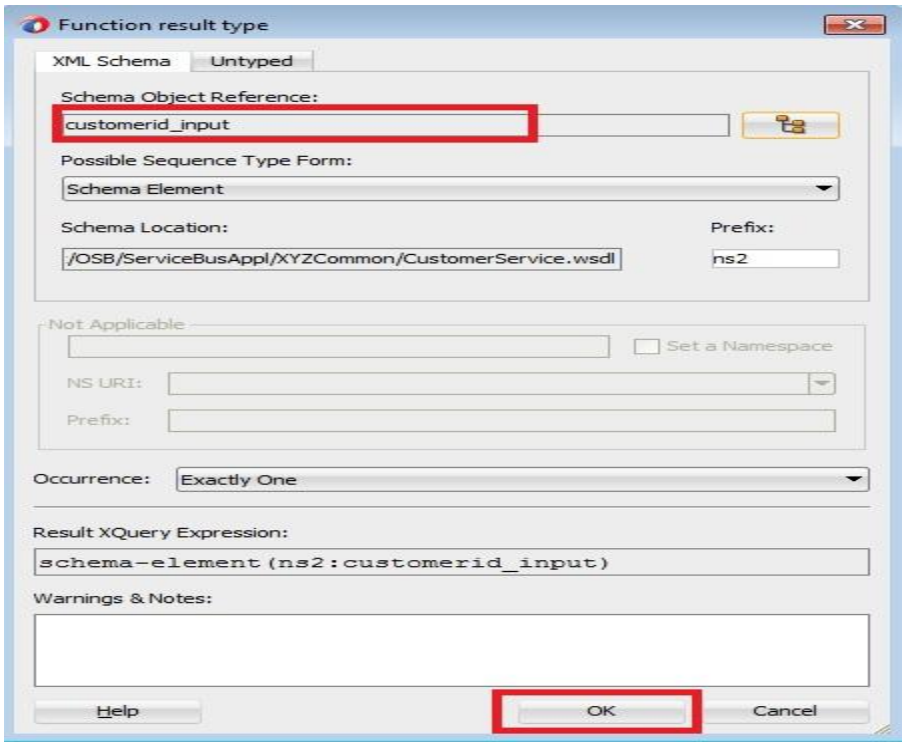


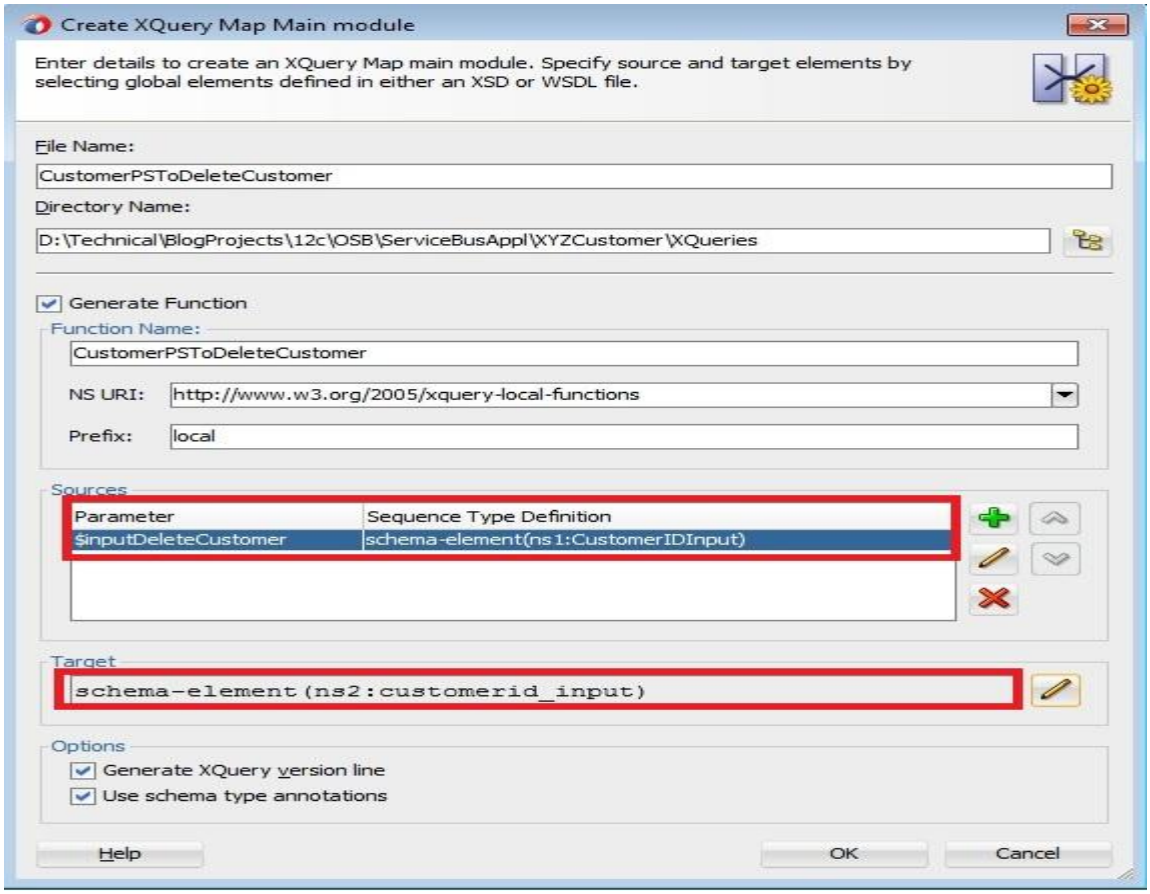

Function parameter type
XML Schema
Untyped
Schema Object Reference:
CustomerIDInput
Possible Sequence Type Form:
Schema Element
Schema Location:
Prefix:
B/ServiceBusAppl/XYZCustomer/WSDLs/CustomerPS.wsdl
ns1
Not Applicable
Set a Namespace
NS URI:
Prefix:
Occurrence:
Exactly One
Result XQuery Expression:
schema-element(ns1:CustomerIDInput)
Warnings & Notes:
Help
OK
Cancel


Function parameter
Name
inputDeleteCustomer
Set a Namespace
NS URI:
Prefix:
Sequence Type
schema-element(ns1:CustomerIDInput)
Help
OK
Cancel

Complete **Target** selection as shown below.

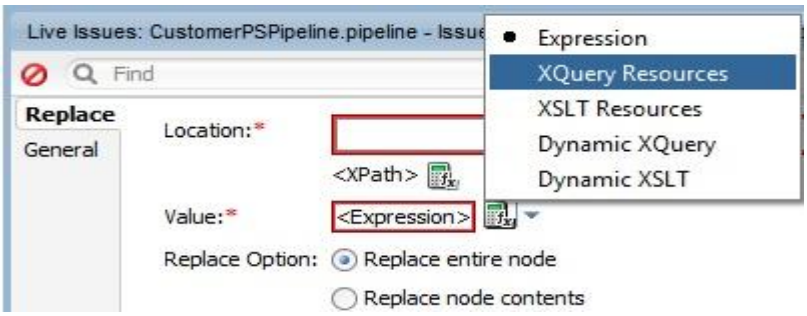


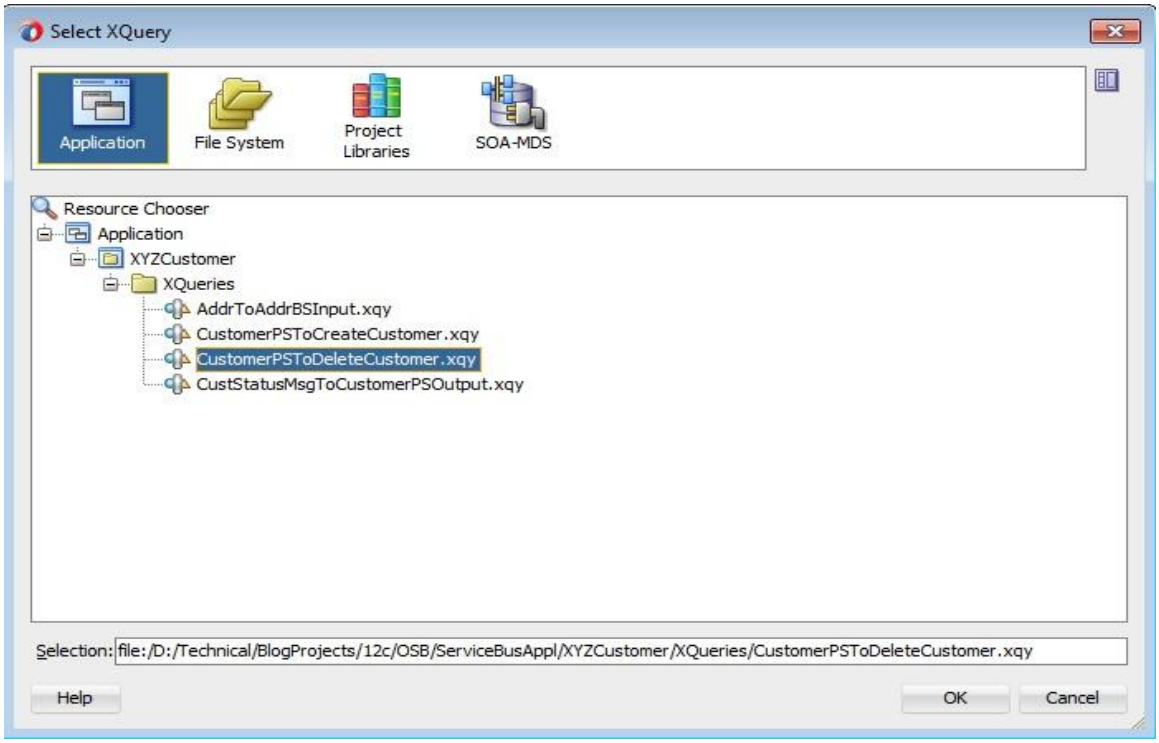




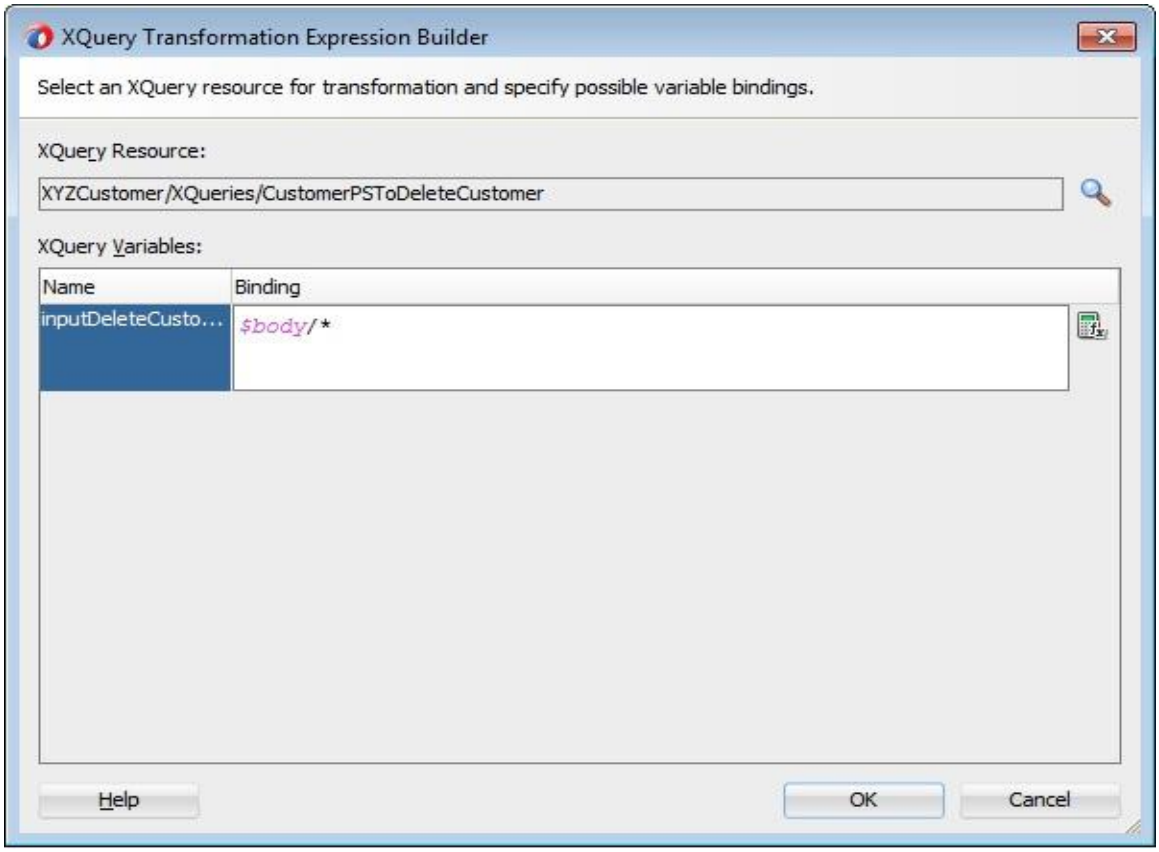
Both source and targets have single element **CustomerID** and finish mapping as shown below.

Drag **Replace** activity into **Request Action** of Routing from **Message Processing**. In **Properties tab**, bring up expression builder for **Value** property and choose above XQuery map.

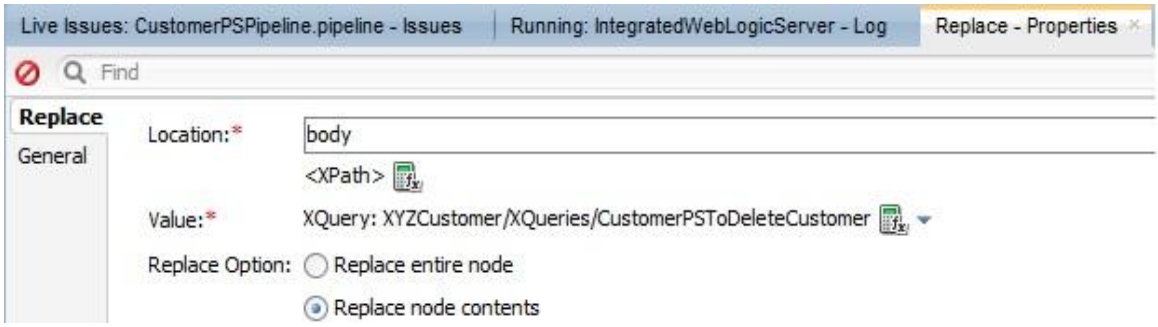




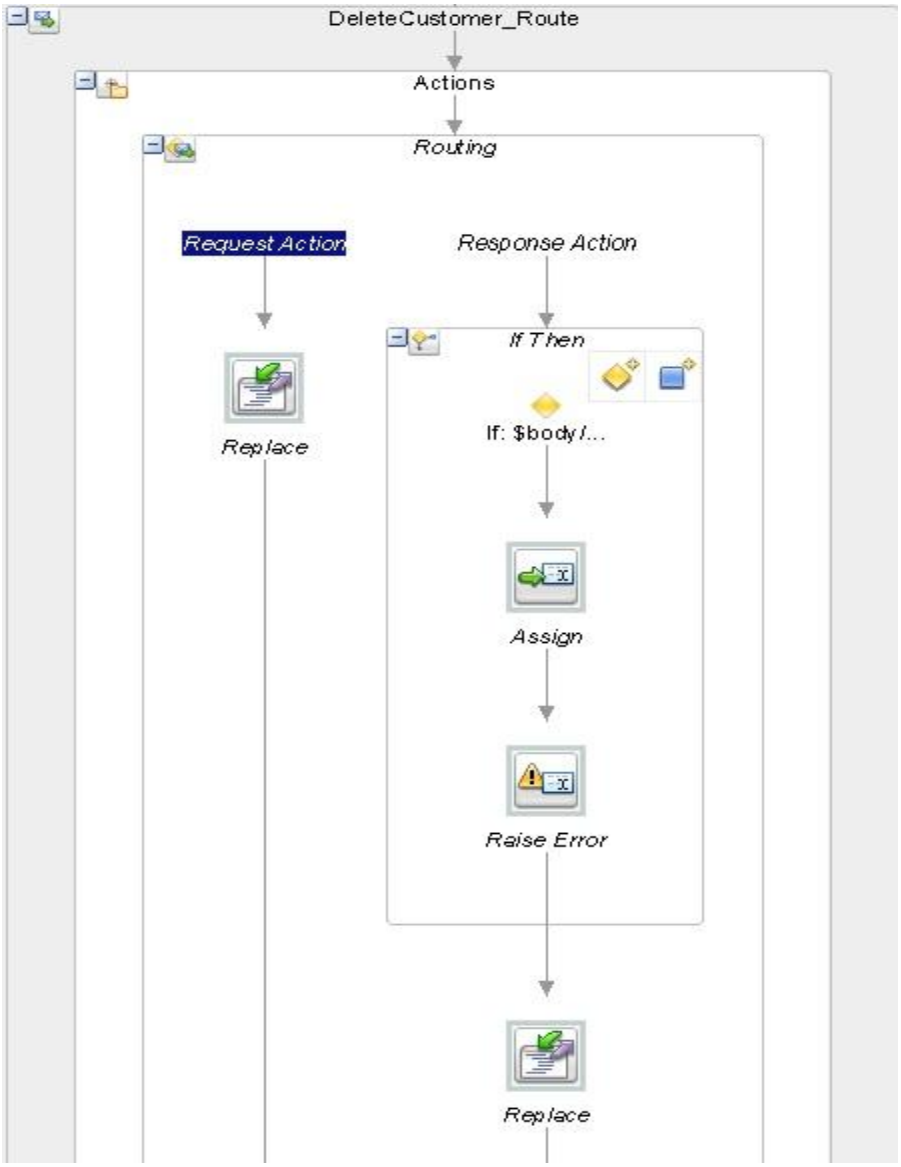
Click **OK** and give bind variable value as **$body/***.



Set other properties as shown below.



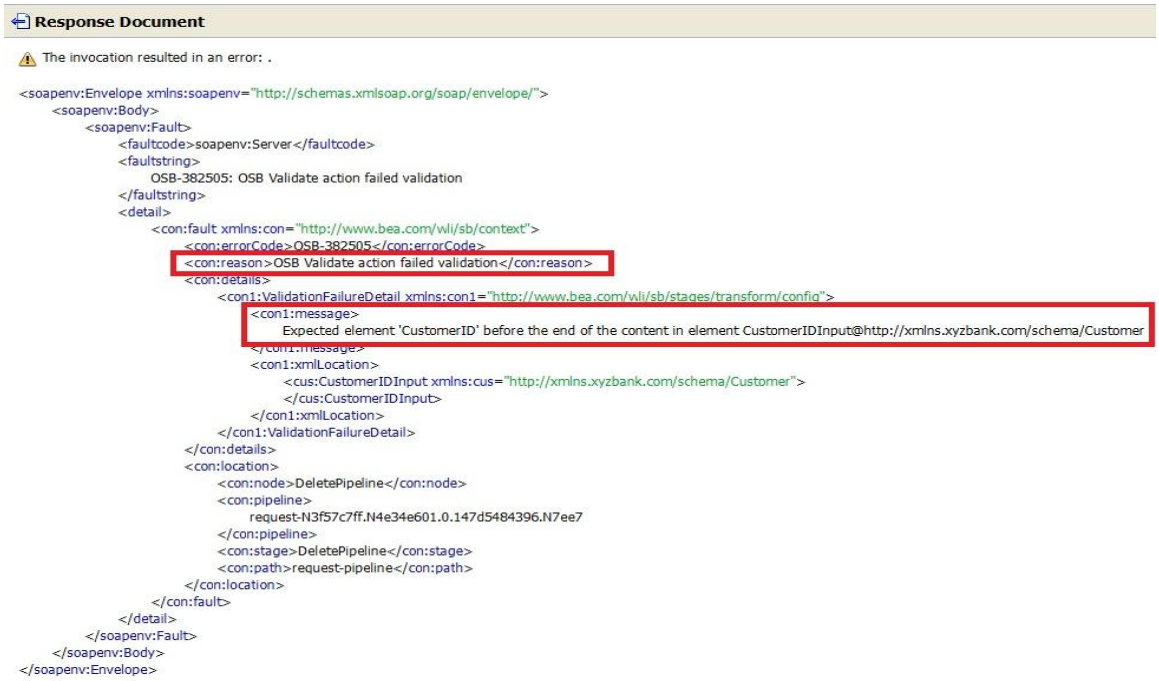
Finish **Response Action** in **Routing** similar to **CreateCustomer** as response structure of business service is same for all the operations. So you can reuse **CustStatusMsgToCustomerPSOutput XQuery Map.** Now your **Routing** node should look like below.



Testing

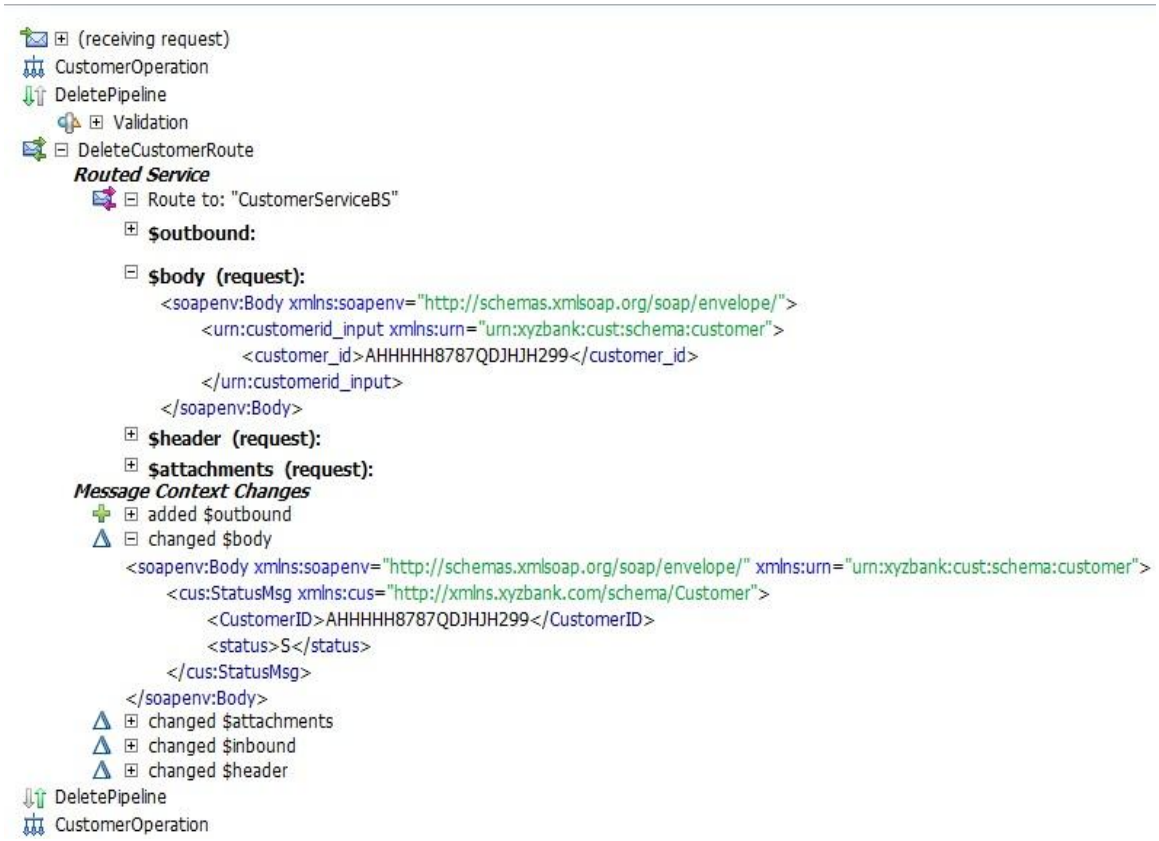
Deploy both projects or run **Pipeline** directly as shown in **Deploying and Testing** section. You may want to frequently test your proxy service/pipeline during development. Run your pipeline with sample payloads given here and observe **Flow Trace and Variables** as shown below. You can also run Proxy Service but you will not observe any **Flow Trace**.

No CustomerID element:

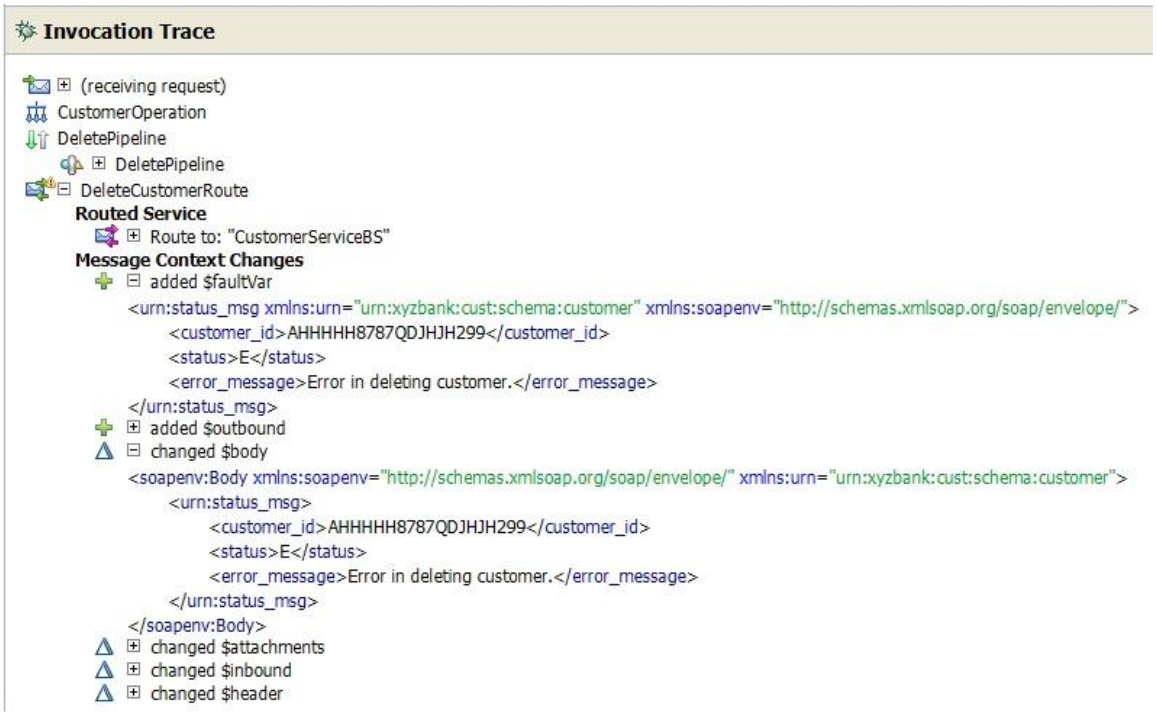


No value for CustomerID:

Positive Case:



Negative Case - Status E in business service response:



Response Document

⚠ The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>XYZ-0004: Error in Customer Deletion.</faultstring>
            <detail>
                <con:fault xmlns:con="http://www.bea.com/wli/sb/context">
                    <con:errorCode>XYZ-0004</con:errorCode>
                    <con:reason>Error in Customer Deletion.</con:reason>
                    <con:location>
                        <con:node>DeleteCustomerRoute</con:node>
                        <con:path>response-pipeline</con:path>
                    </con:location>
                </con:fault>
            </detail>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

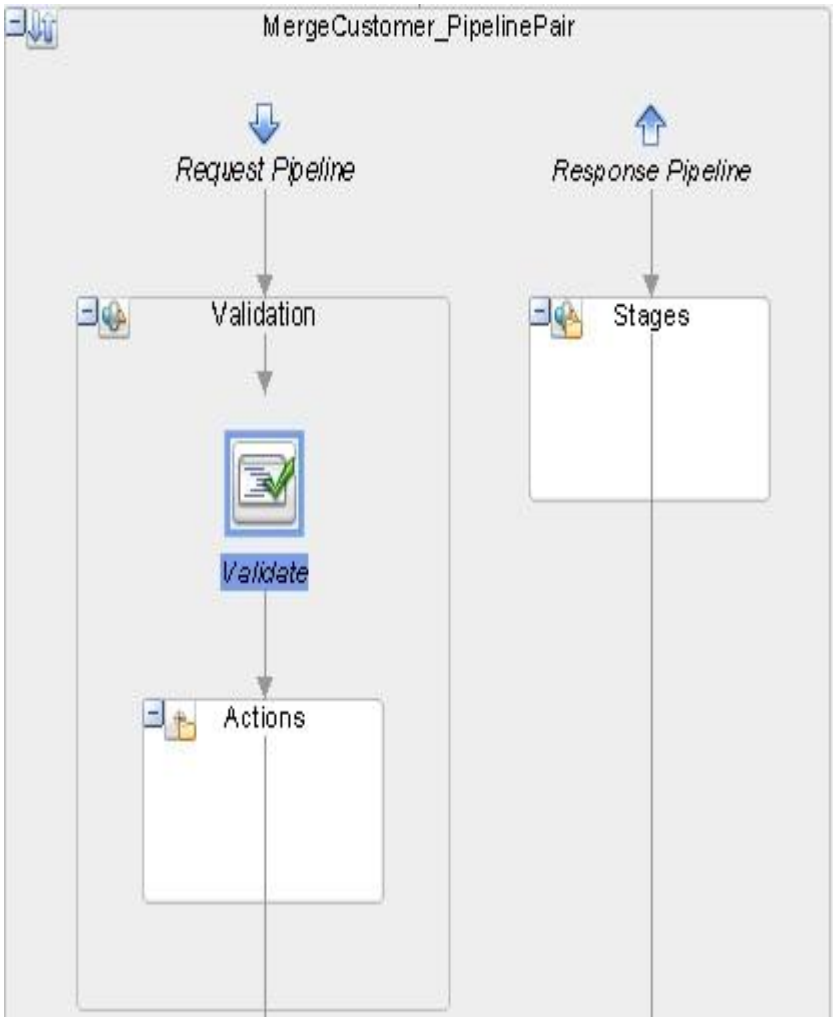
Merge Customer

In this section, you will complete message flow for **MergeCustomer** operation. As per business requirements, merge operation should create customer if it does not exist in system otherwise update existing customer. So you should reuse the existing message flow created for **Create** and **Update** customer operations. The message flow should perform following:

- Call **find_customer** operation of business service to find whether customer exists or not. This operation returns **Y** for existing customer and **N** otherwise.
- Create a new pipeline **CustomerCmnPipeline** using same wsdl as **CustomerPS**.
- Move **CreateCustomer** and **UpdateCustomer** message flows intto this new pipeline.
- Modify **CreateCustomer** and **UpdateCustomer** pipelines to call respective operations of new pipeline.
- Modify **MergeCustomer** operational branch to call **CreateCustomer** or **UpdateCustomer** operations of new pipeline based on customer existence.

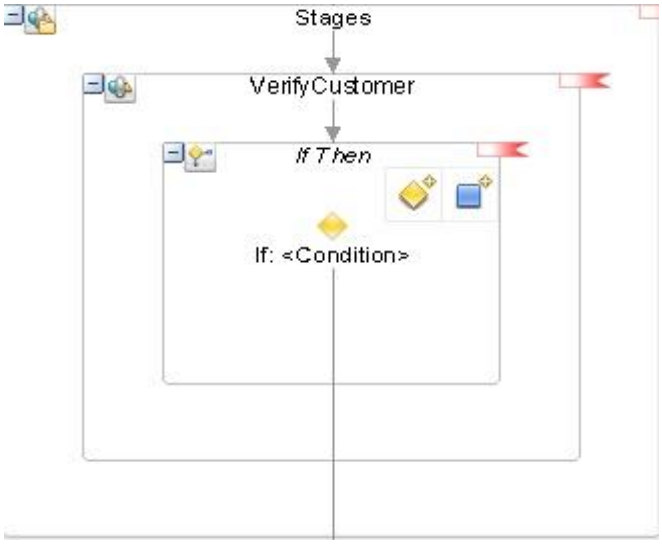
Validating Payload

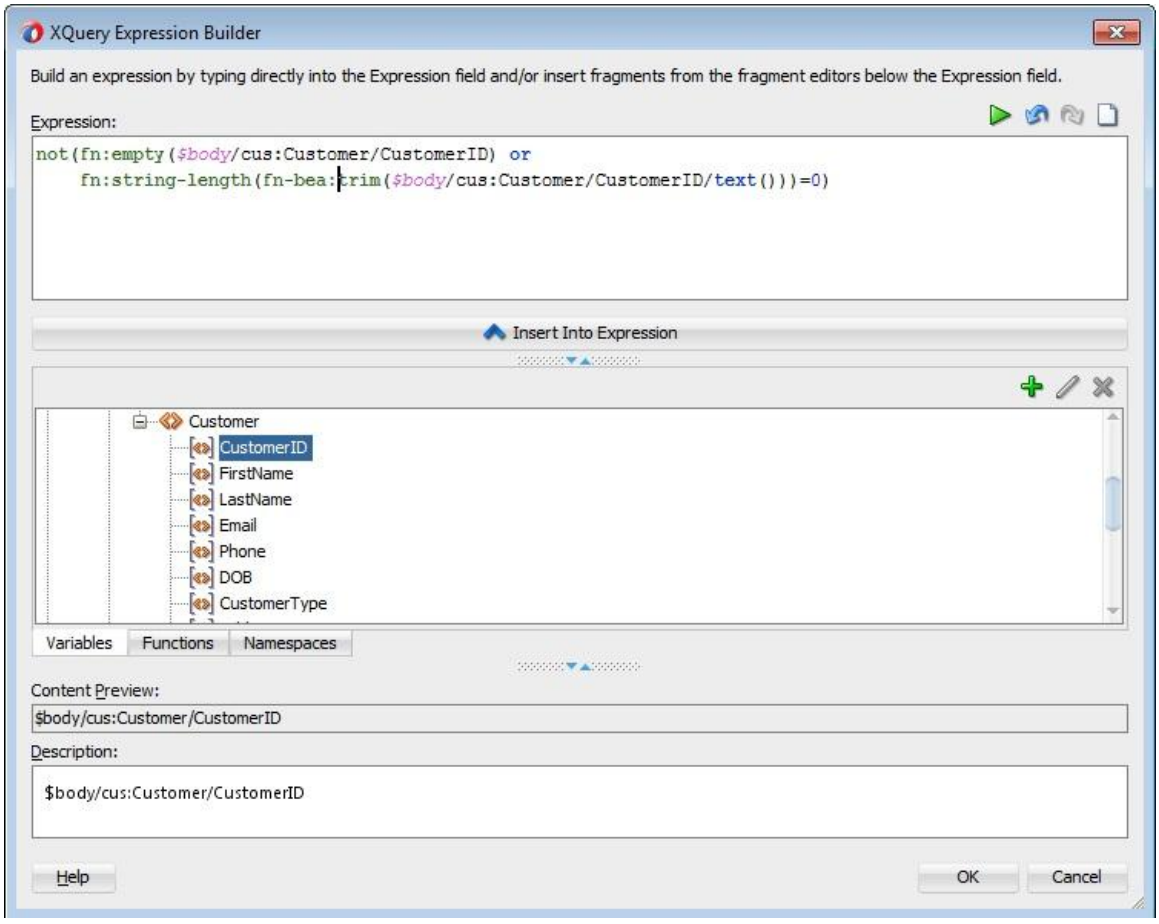
Finish **Validation** stage in **MergeCustomer** operational branch similar to **CreateCustomer** for validating **Payload Structure** using **Validate** activity. But you should select **Customer** element from **$body/MergeCustomer** in expression builder for **Location** property.



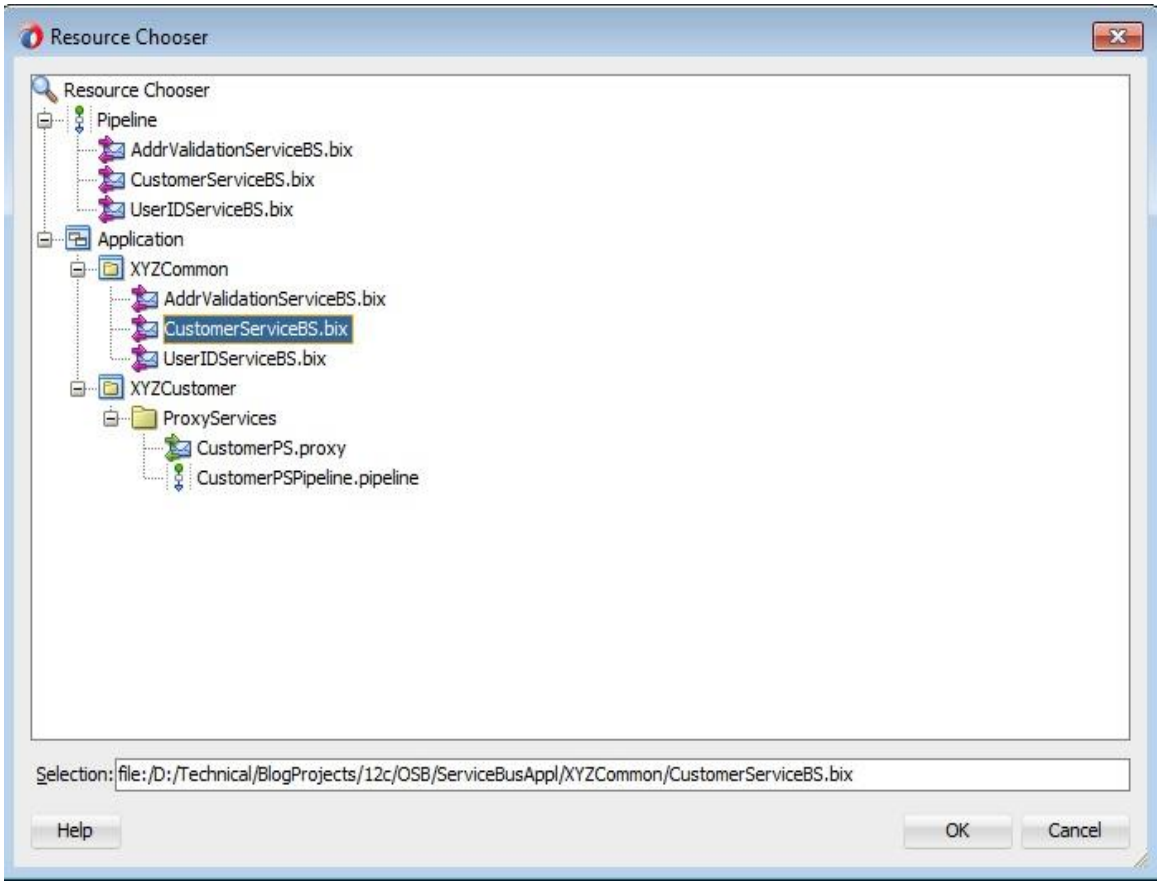
Verifying Customer Existence

Drag **stage** node into **Stages** placeholder in **Request Pipeline** of **MergeCustomer_PipelinePair** from **Nodes** and name it as **VerifyCustomer.** You can verify customer existence only when **CustomerID** is present in input payload. So drag **If-Then** activity from **Flow Control** and set condition as shown below.

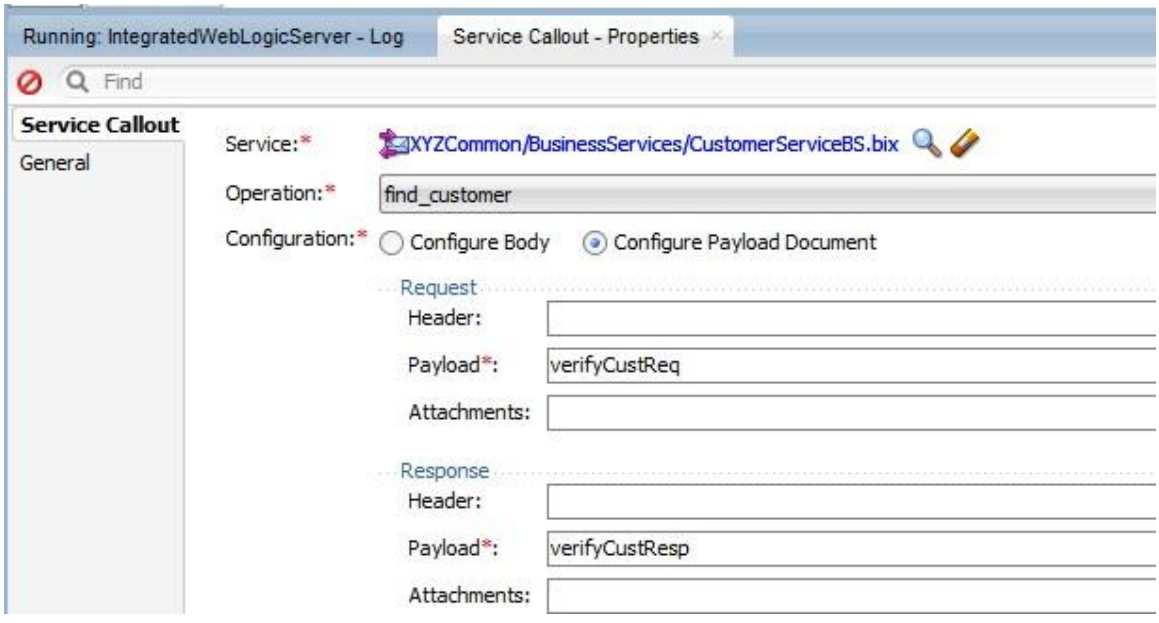




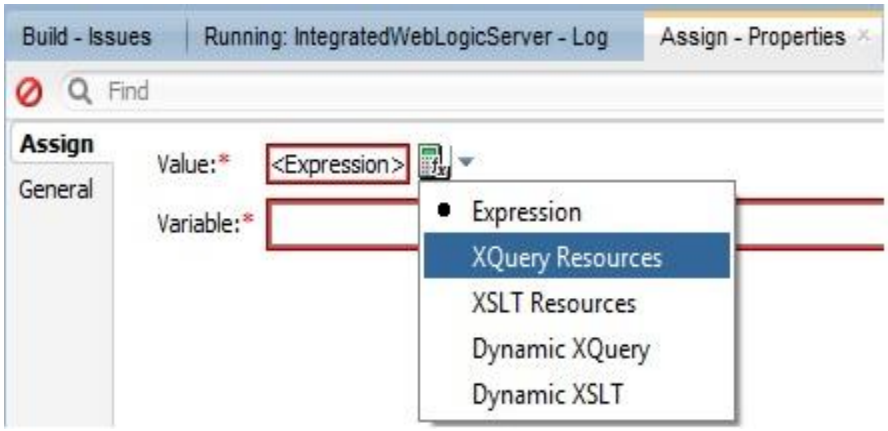
Drag a **Service Callout** into **If** branch from **Communication.** In **Properties** tab, browse and select business service **CustomerServiceBS** and **find_customer** as operation.

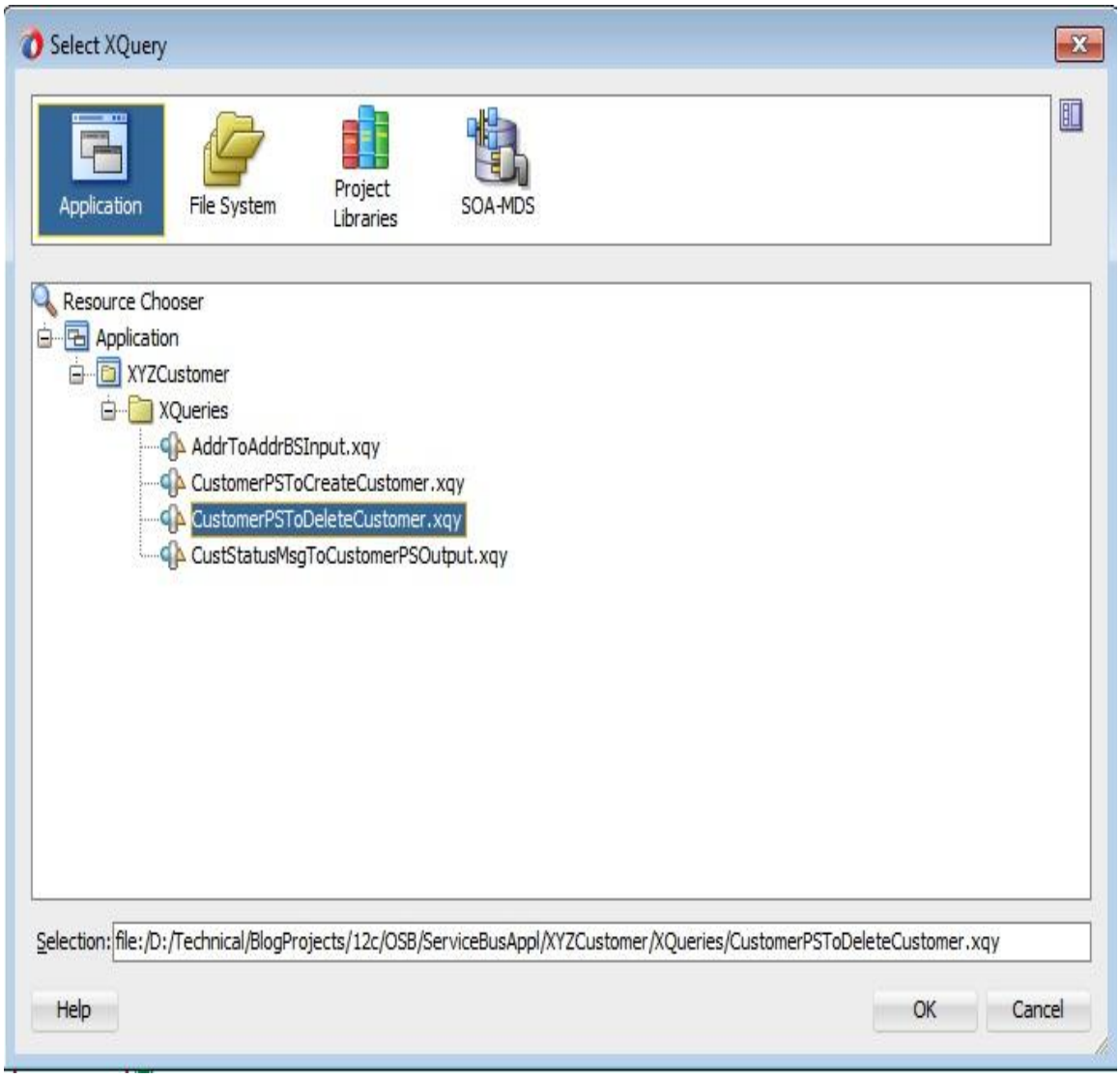


Set other properties as shown below and choose '**Configure Payload Document**' option.

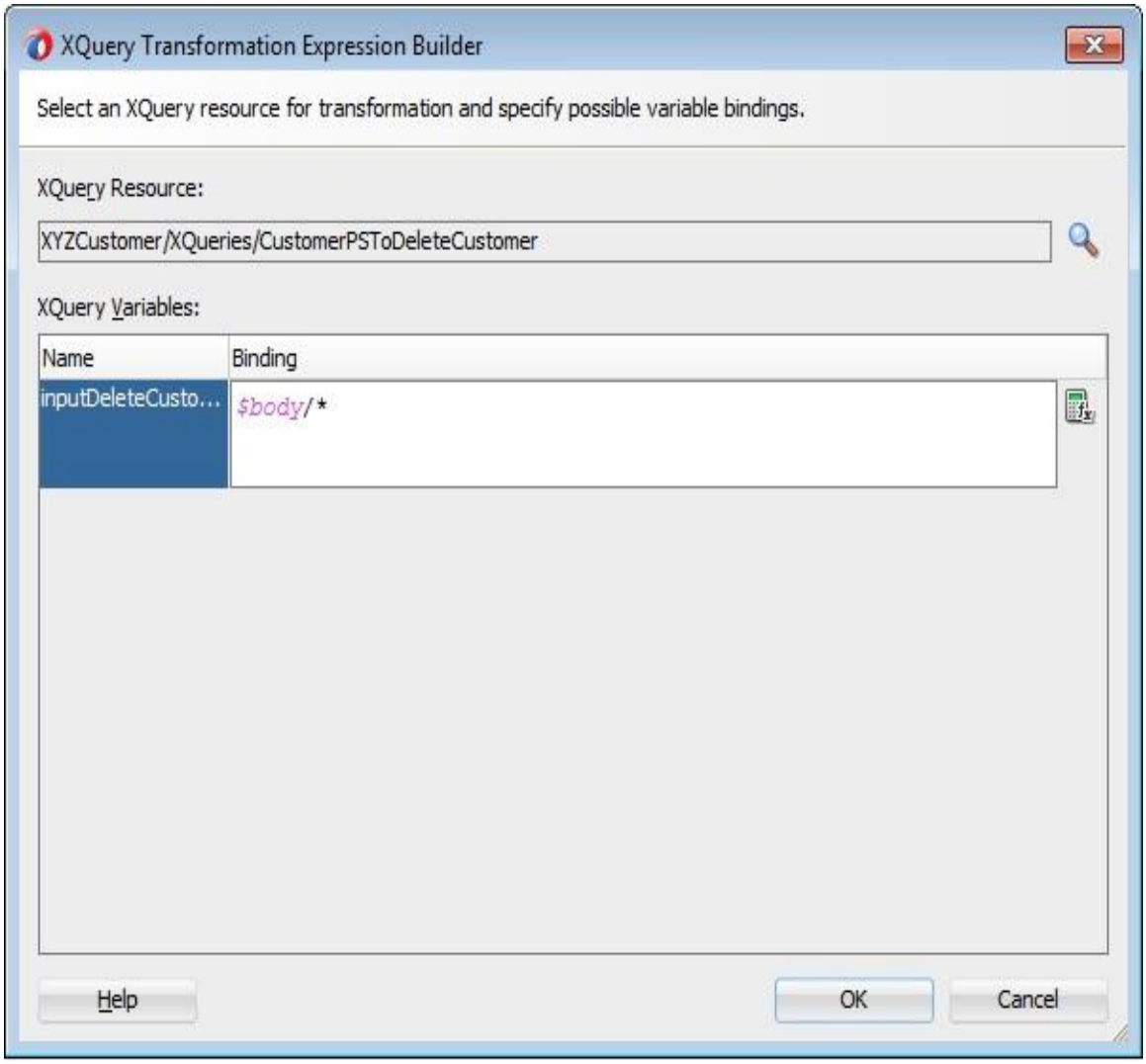


You can reuse XQuery map **CustomerPSToDeleteCustomer** as input structure of **Delete** and **Find** operations is same. So drag **Assign** activity into **Request Action** of **Service Callout** from **Message Processing** and select this XQuery map as shown below.



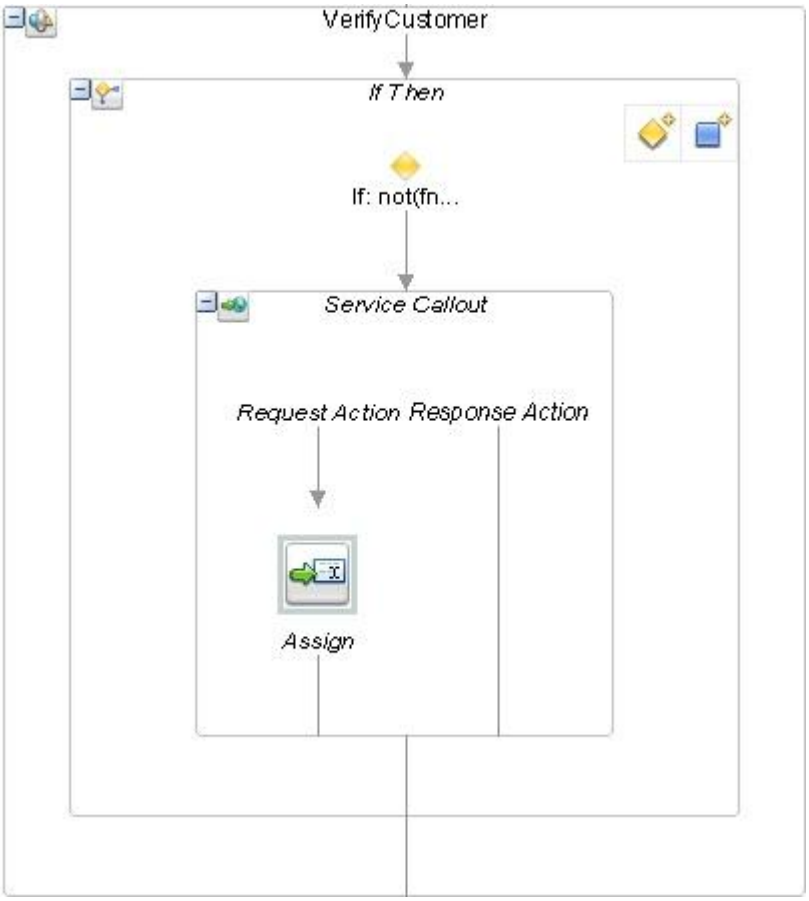


Give **$body/*** as value for **Binding** as shown below.



Give **verifyCustReq** as value for Variable property.

Now your **VerifyCustomer** stage should look like below.

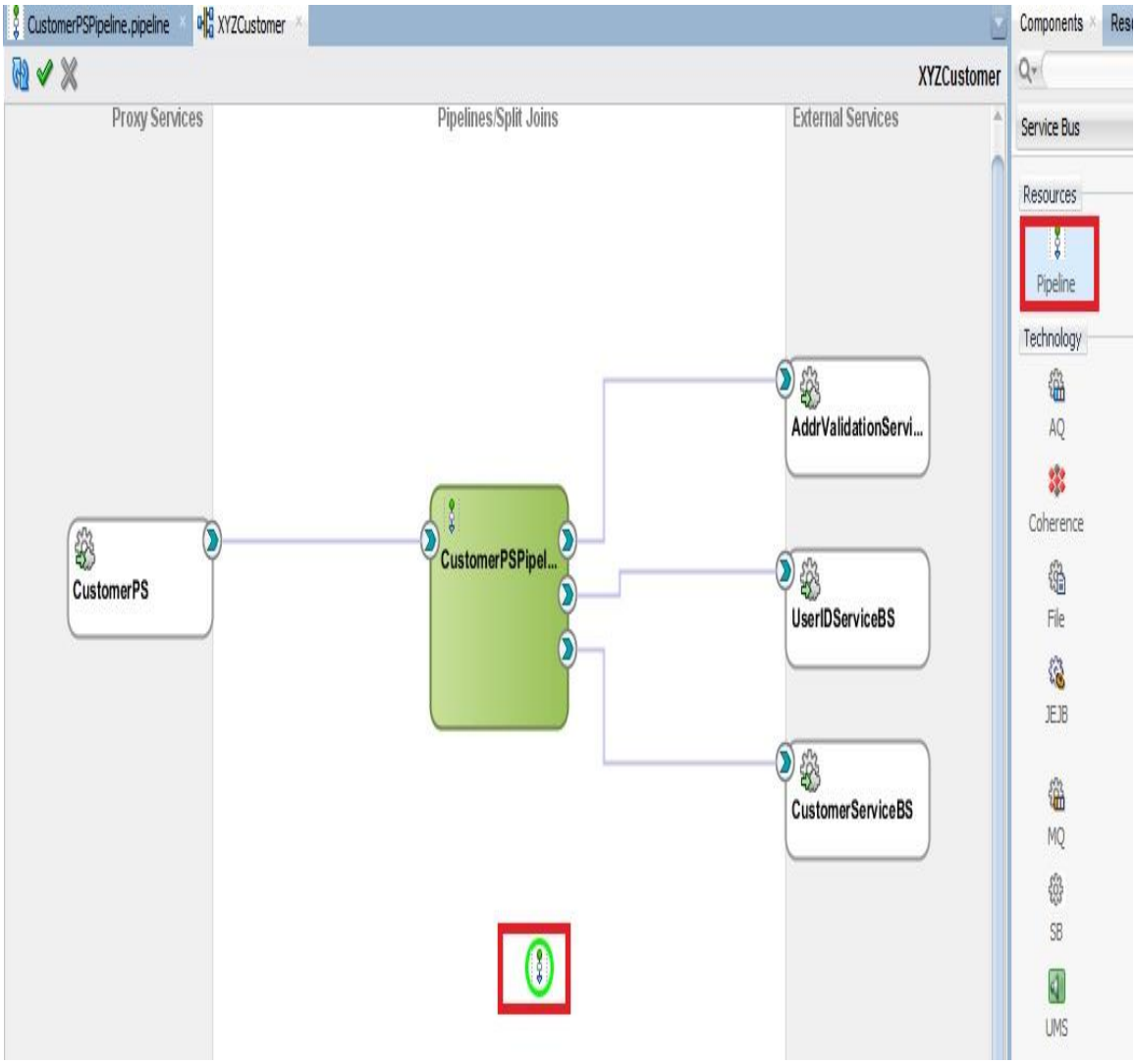


Creating CustomerCmnPipeline

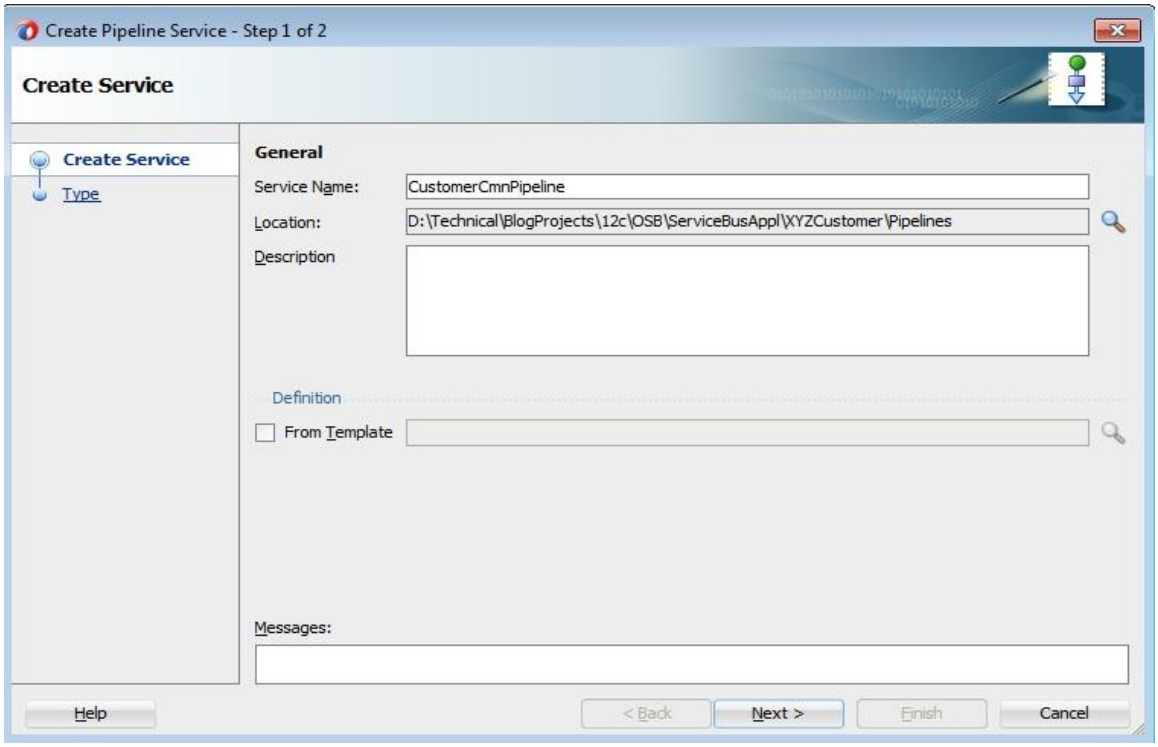
Depending on response of **find_customer** call in previous step, you have to **Create** or **Update** customer information. And you already completed these actions as part of **CreateCustomer** and **UpdateCustomer** operational branches. So it's not a good idea to repeat all of this logic again for this merge operation.

To overcome this, you can create a new Pipeline/Proxy Service having common logic for Create and Update. If your requirement is just within Service Bus project you can create Pipeline having common message flow and call from main Pipeline or ProxyService. If you require this across projects you can create Proxy Service having common message flow with **local** transport. This way, your common proxy service will not be exposed to external consumers and can only be invoked internally in message flow. In this section, you will create new pipeline having common message flow using **Service Bus Overview editor**.

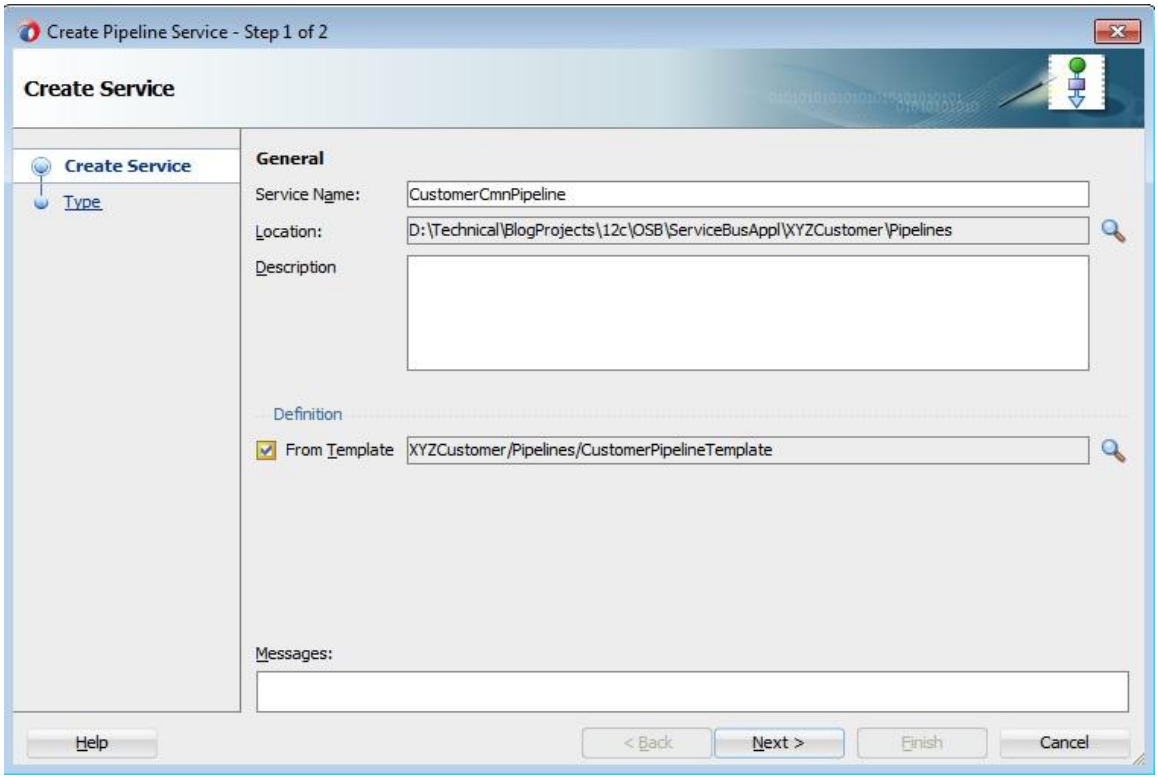
So open Service Bus Overview editor by opening file having same name as Service Bus project. Drag **Pipeline** from **Components -> Service Bus -> Resources** into middle swim lane as shown below.



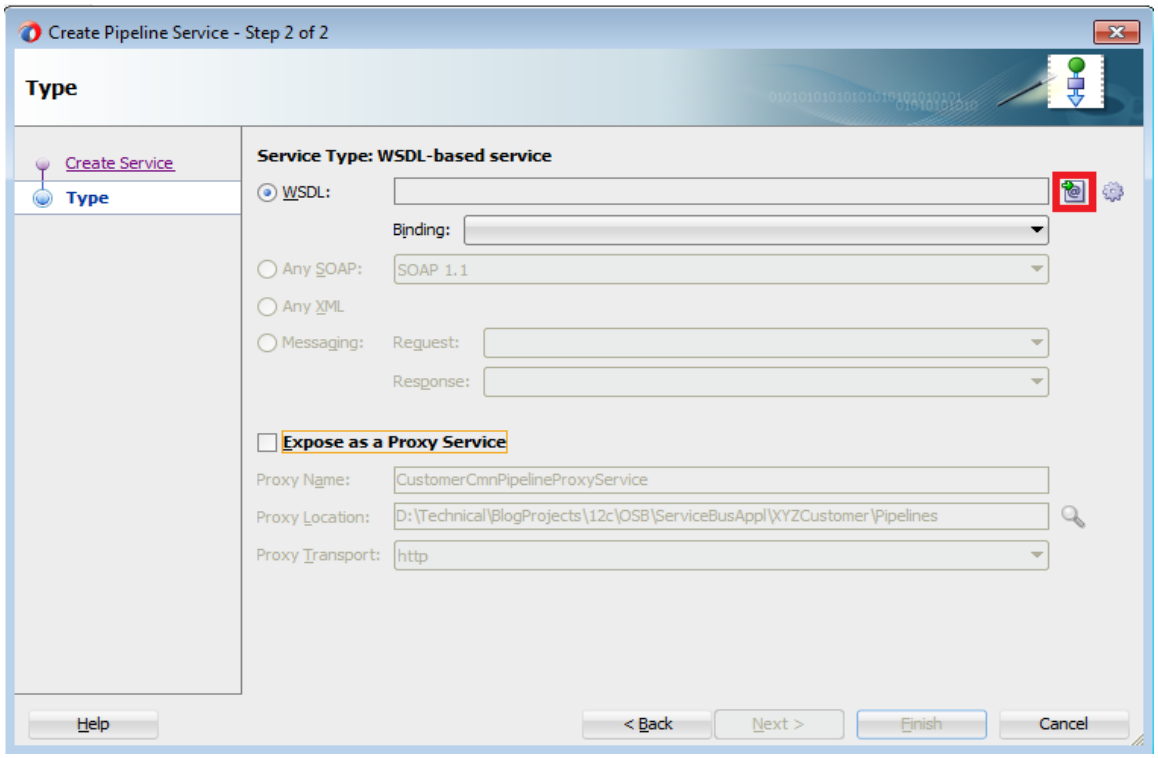
Give **Service Name** as **CustomerCmnPipeline** and select **Pipelines** folder using browse icon for **Location**.



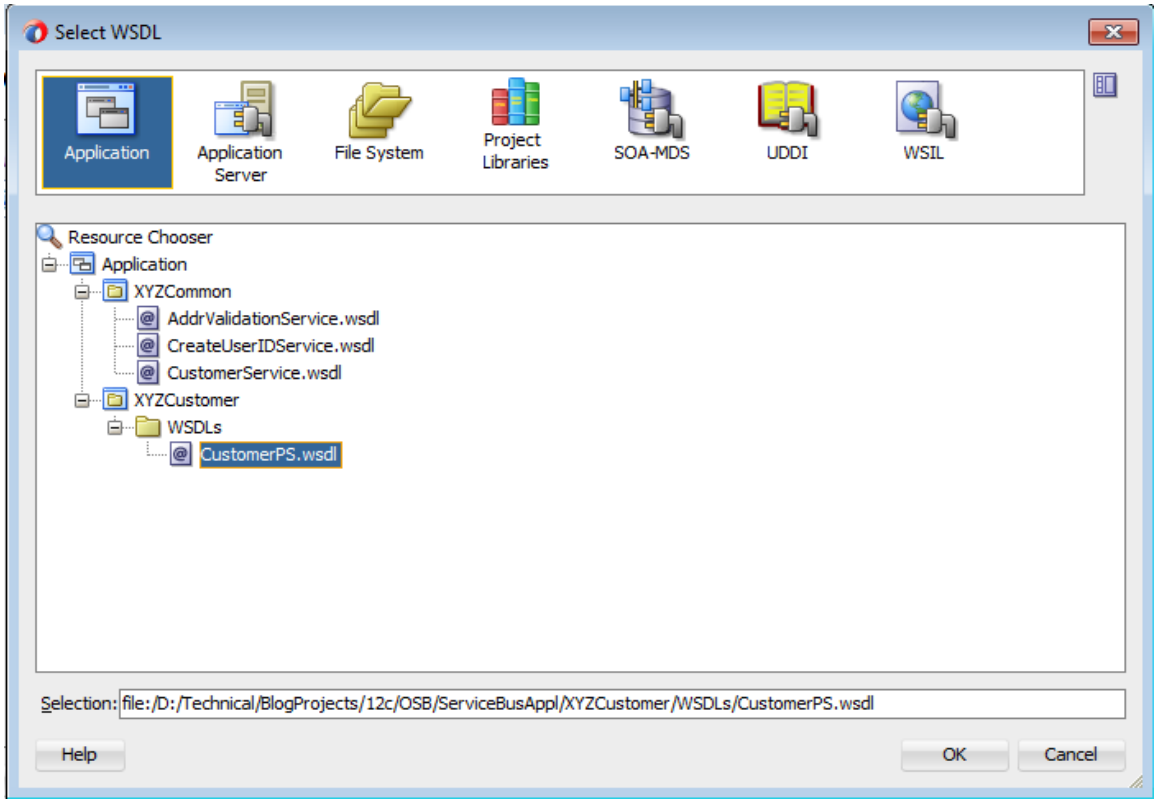
Choose the option **From Template** and select **CustomerPipelineTemplate** using browse icon.



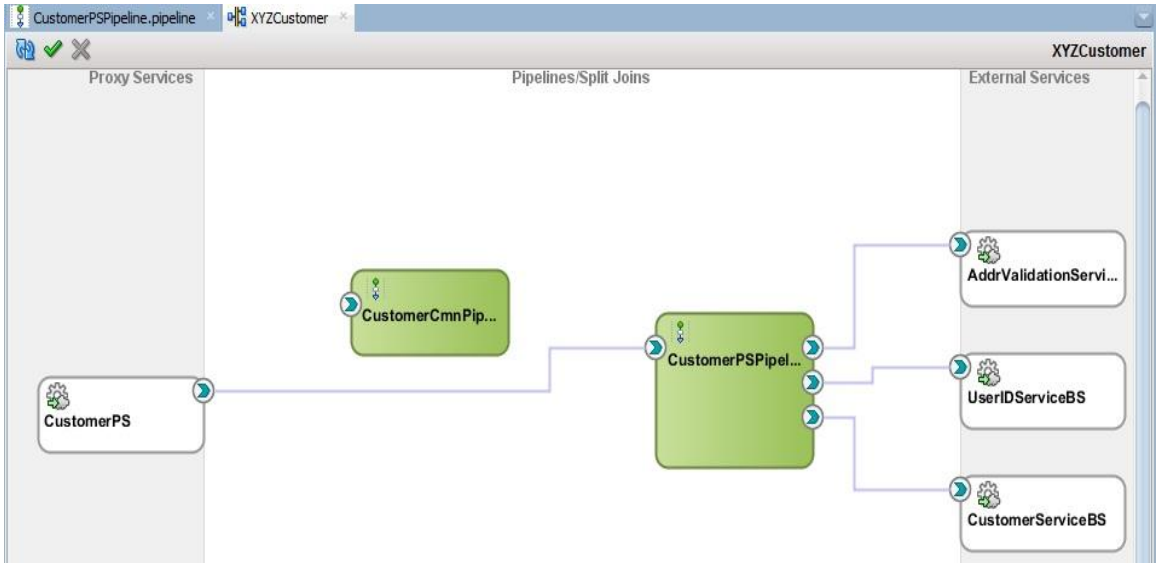
Click **Next.** Retain **Service Type** as **WSDL** and unselect the option **Expose as a Proxy Service.**



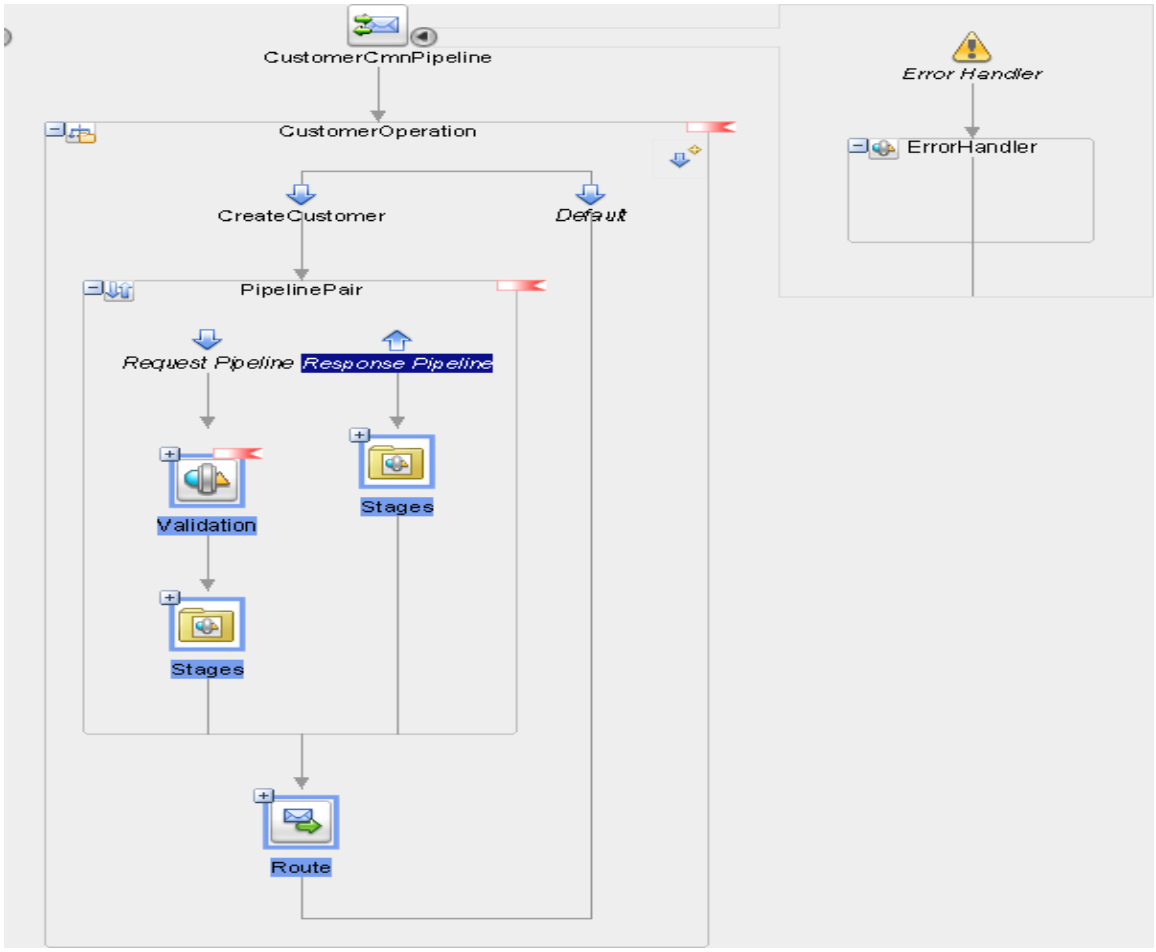
Click **Browse WSDLs** (highlighted above) and select same WSDL used for **CustomerPSPipeline**.



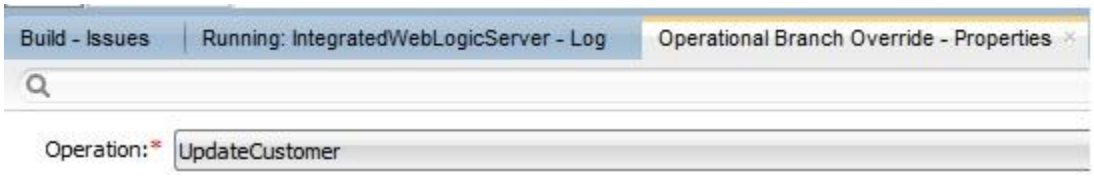
Go back to **Service Bus Overview editor** by finishing pipeline creation wizard and it should look like below. You can also verify new pipeline got created in **Pipelines** folder.



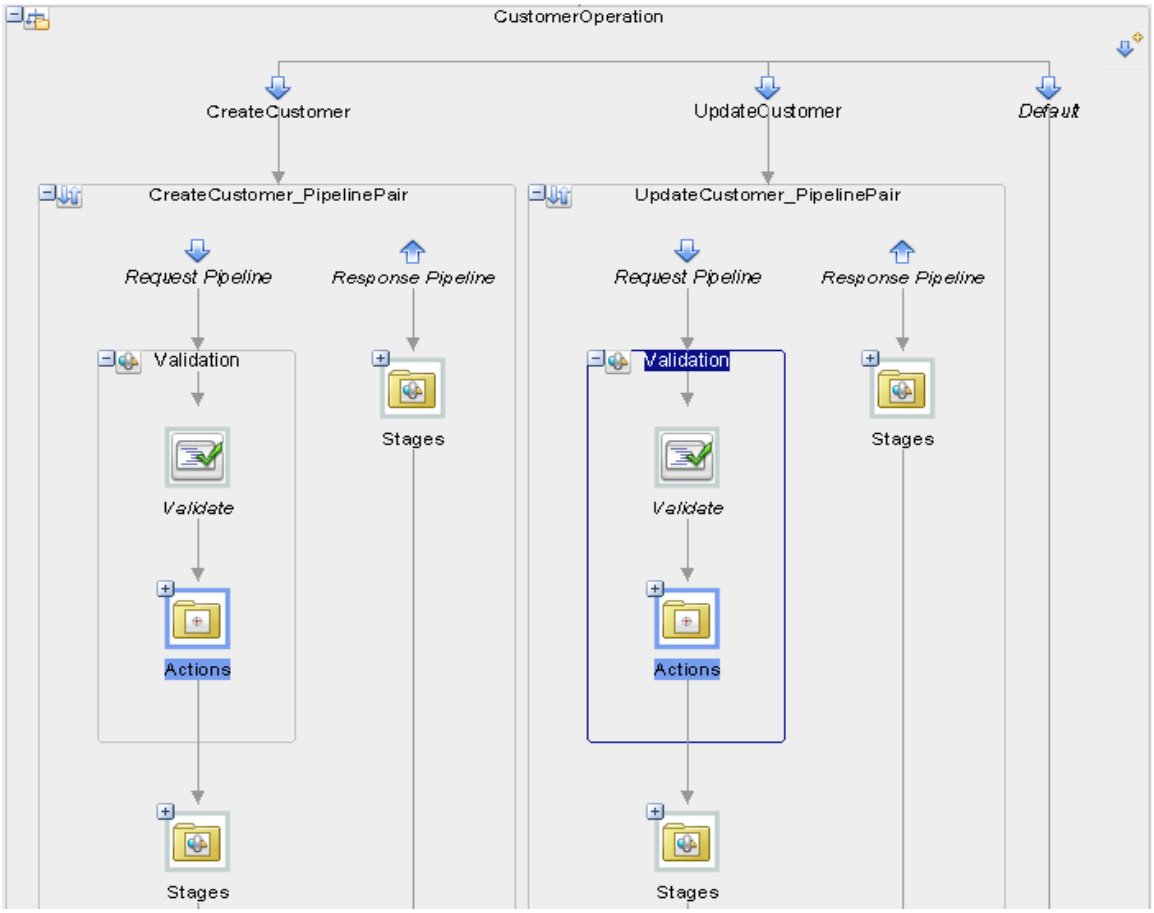
Verify that your new pipeline also inherited the activities from pipeline template as expected.



Add another operational branch and select operation as **UpdateCustomer** in **Properties** tab.



Finish **Validate** activity in both branches as shown previously in respective sections. This would resolve build issues and your message flow should look like below.



Next task is moving the activities from **CustomerPSPipeline** to here. Service Bus allows **Cut** and **Paste** options on activities so that you can move your activities across/within message flow. So move the following activities and nodes to their respective placeholders in this pipeline.

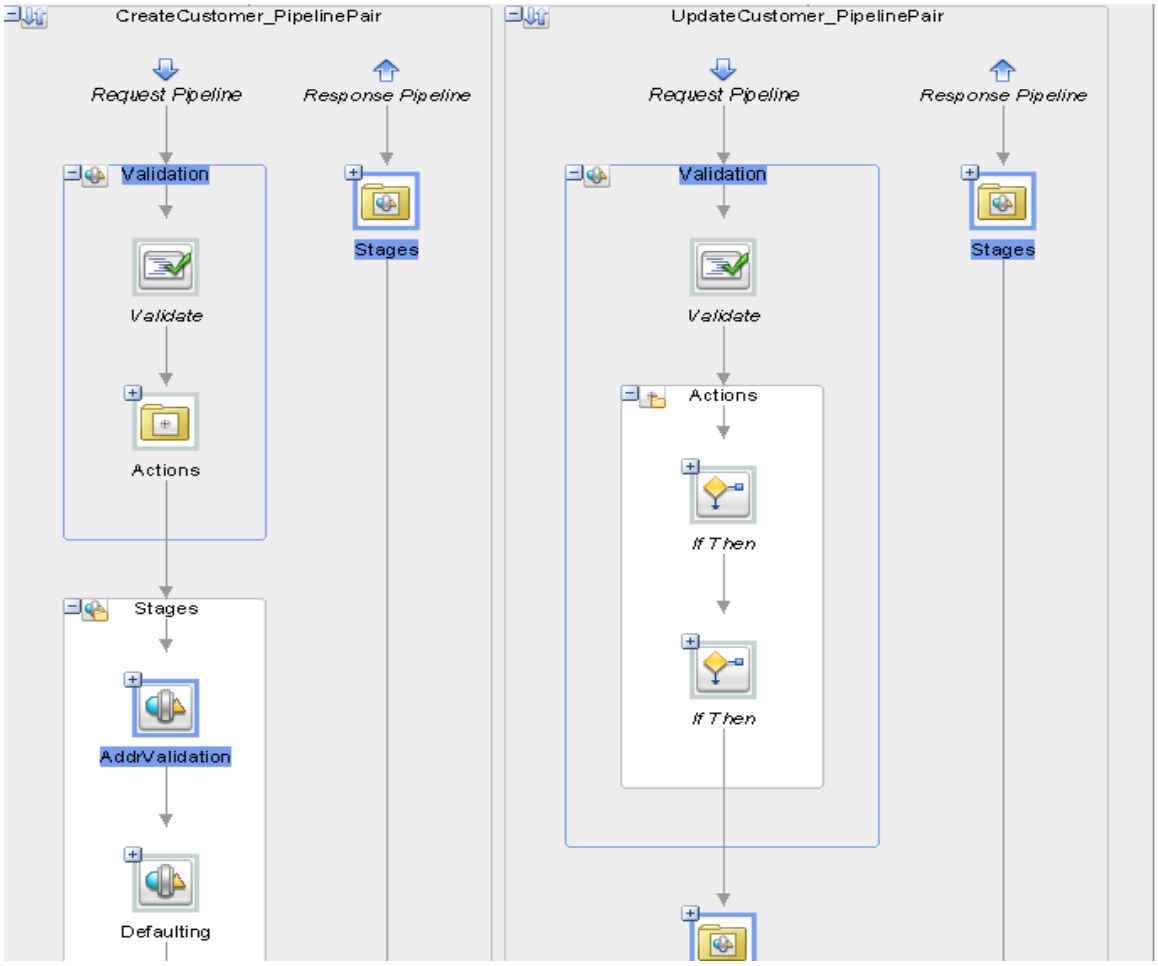
Create Customer:

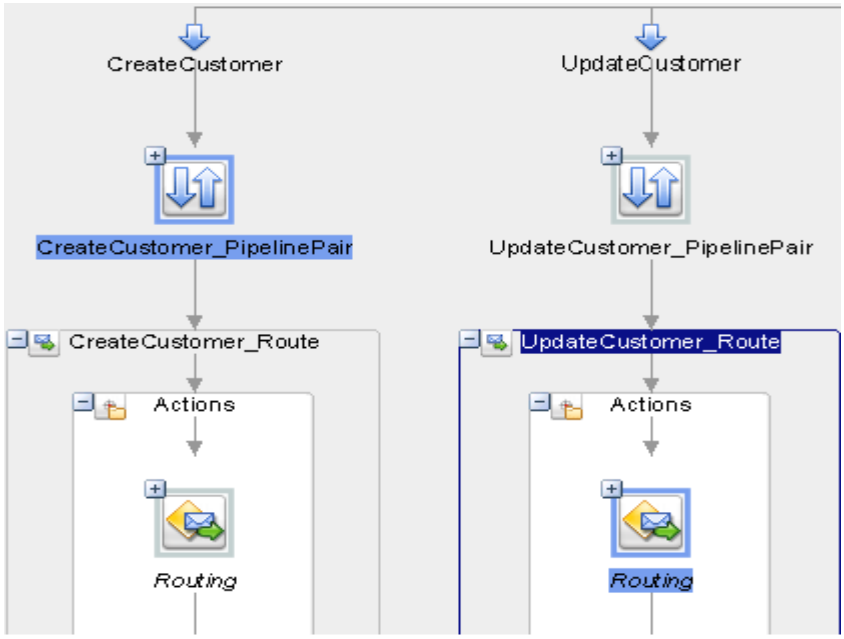
Validation Stage - Actions placeholder contents
AddrValidation Stage
Defaulting Stage
CreateCustomer_Route - Actions placeholder contents

Update Customer:

Validation Stage - Actions placeholder contents
UpdateCustomer_Route - Actions placeholder contents

Now your message flow should look like below.

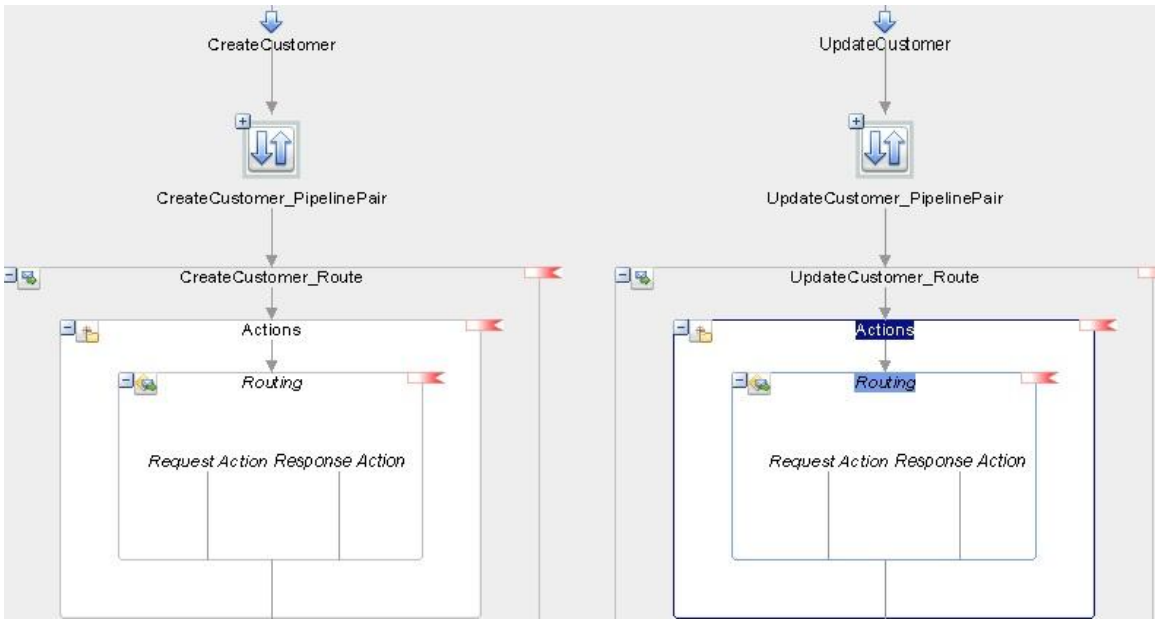




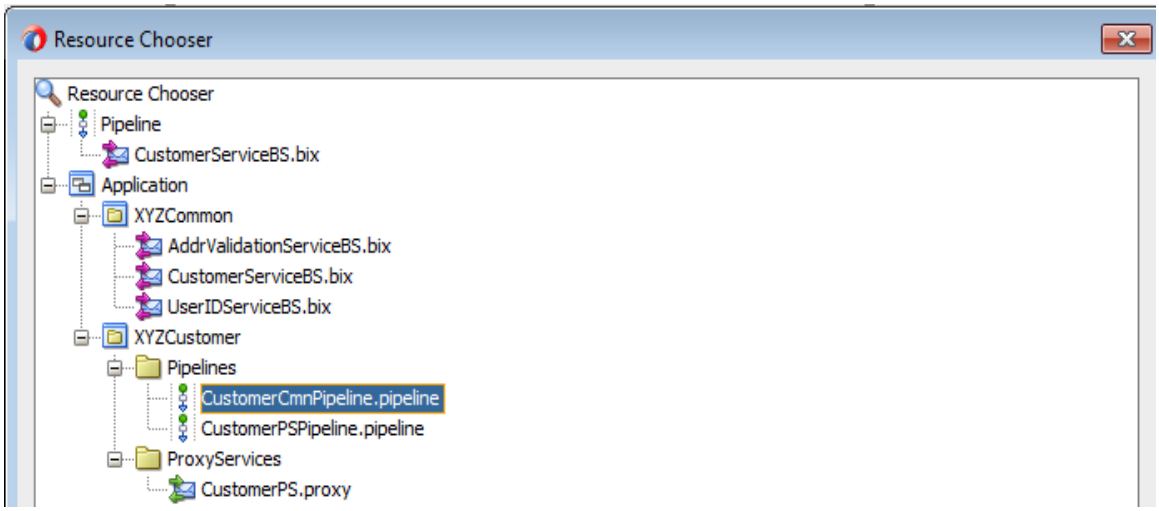
If you see redlines for **If** condition in **Response Action** of the routing activity, add namespace **urn:xyzbank:cust:schema:customer** with alias **cust** as shown in previous sections.

Also observe that, you are unable to copy any of the activities inherited from pipeline templates like **Validate** activity. Because of this, validation will happen twice in **CustomerPSPipeline** and in this pipeline. In your actual development, you should take care of this by creating new pipeline template for common pipelines or some other means. Here, we did this way to demonstrate the features of pipeline template like having common functionality in template.

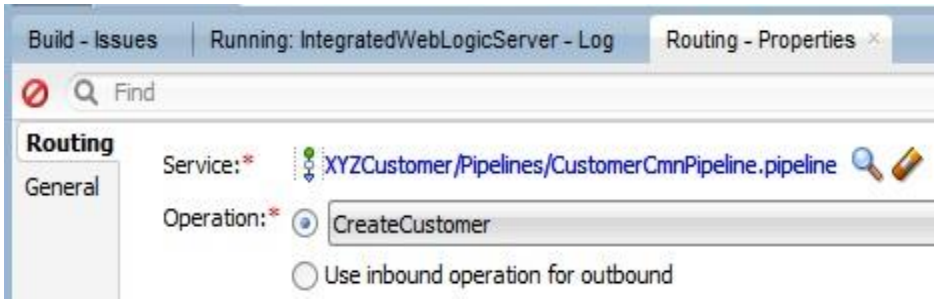
Now let us make changes in **CustomerPSPipeline** to call **CreateCustomer** and **UpdateCustomer** operations of **CustomerCmnPipeline**. Drag **Routing** activity into **Actions** placeholder in both **CreateCustomer_Route** and **UpdateCustomer_Route** as shown below.

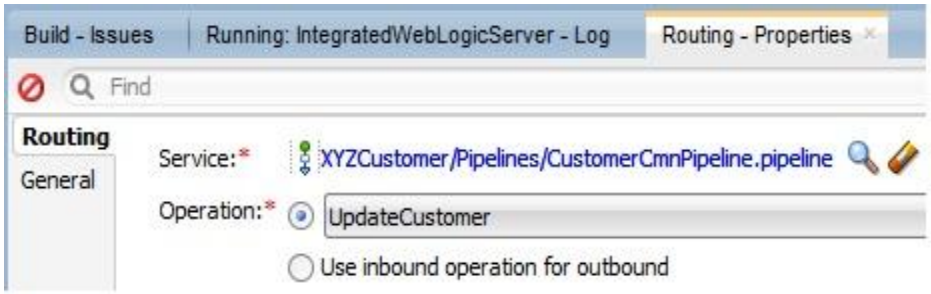


In **Properties** tab, browse and select pipeline **CustomerCmnPipeline** for **Service** property of both **Routing** activities.

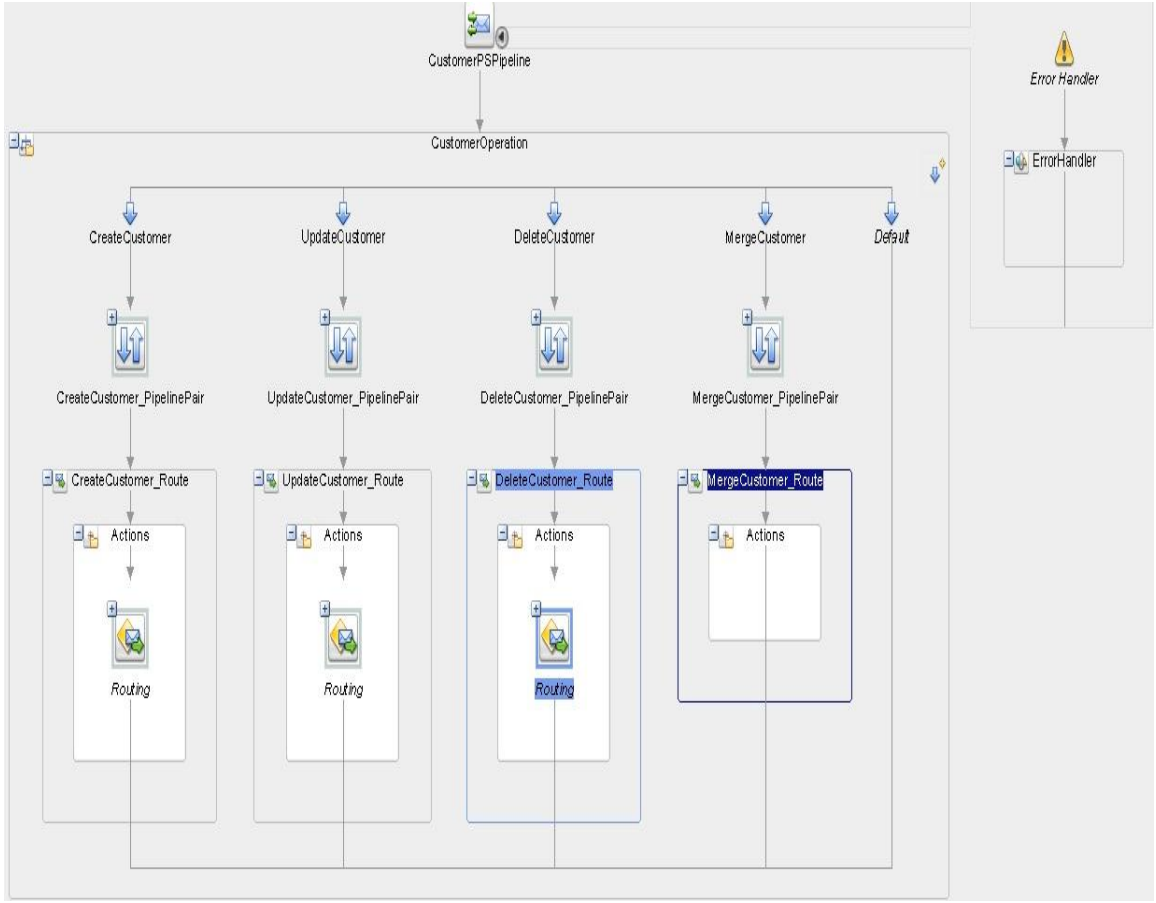


In **Properties** tab, select **operation** as **CreateCustomer** and **UpdateCustomer** respectively.

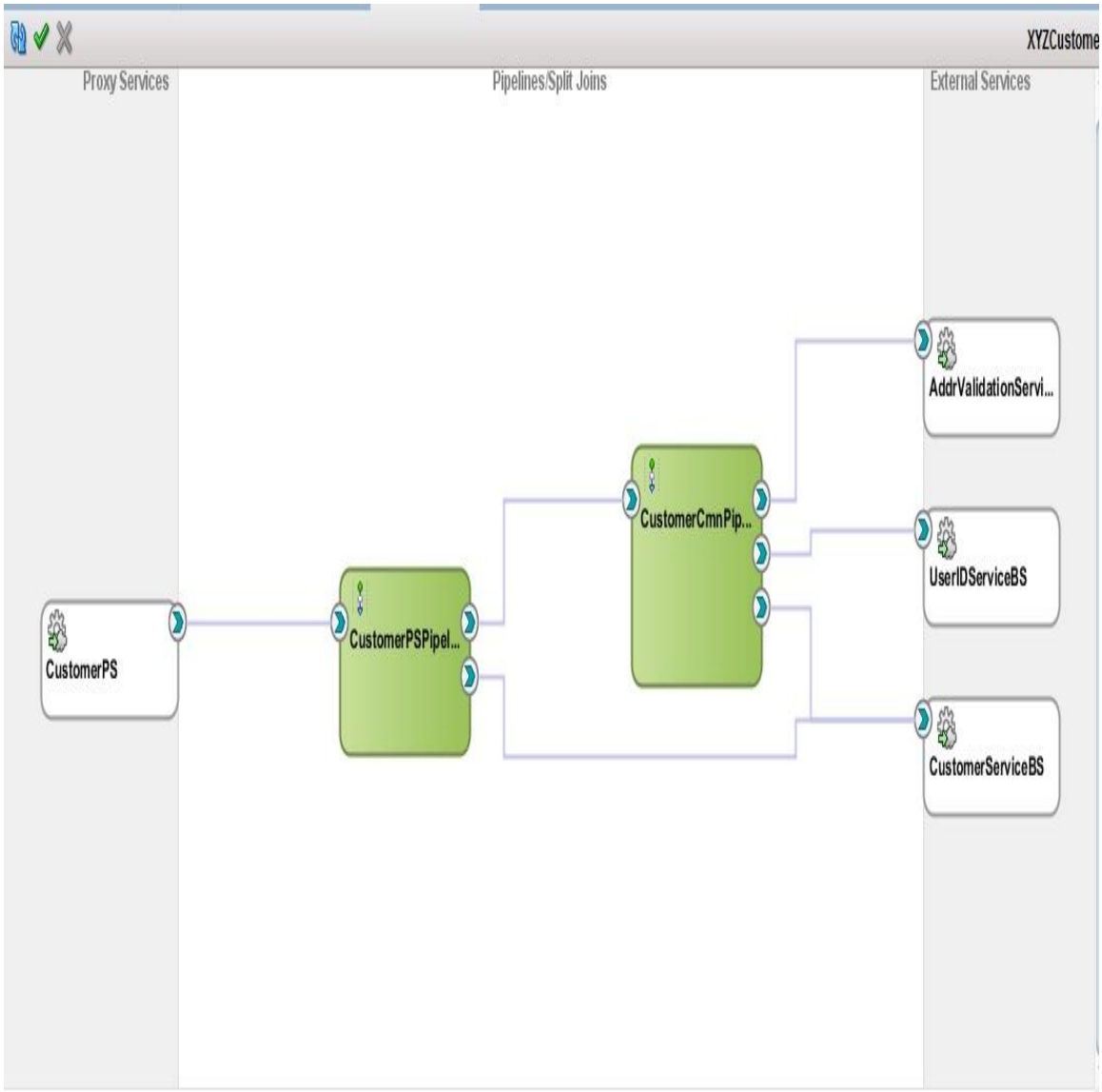




Now your **CustomerPSPipeline** message flow should look like below.



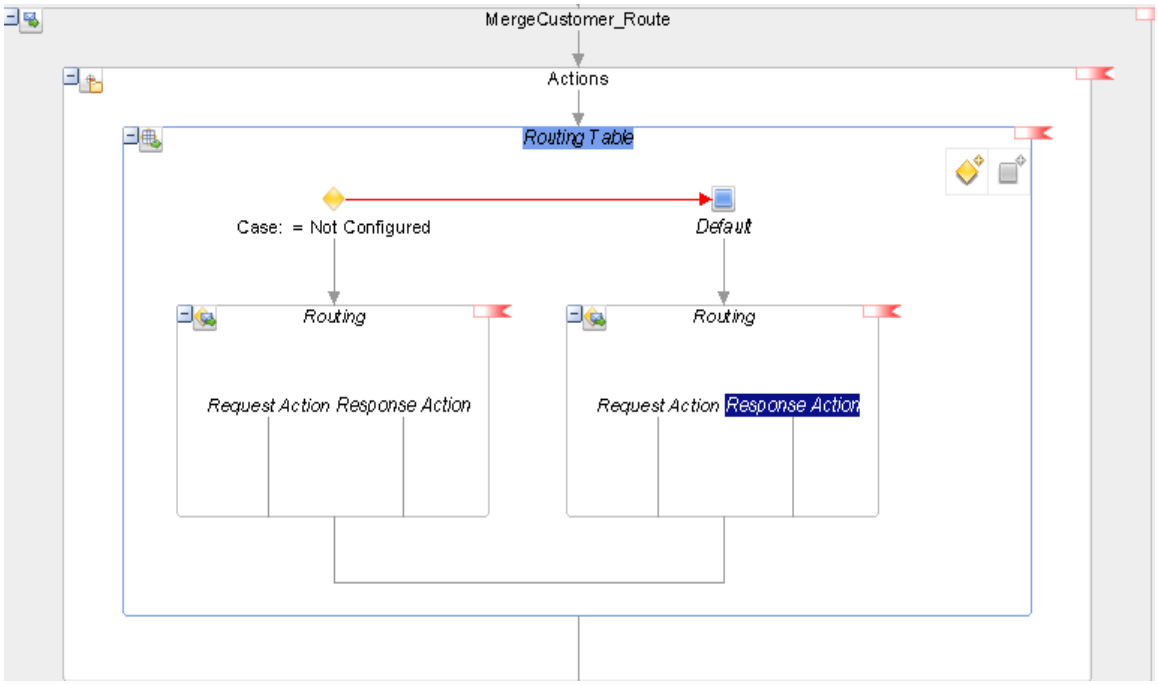
Also your service bus overview should look like as shown below showing different interactions and interfaces exposed to consumers.



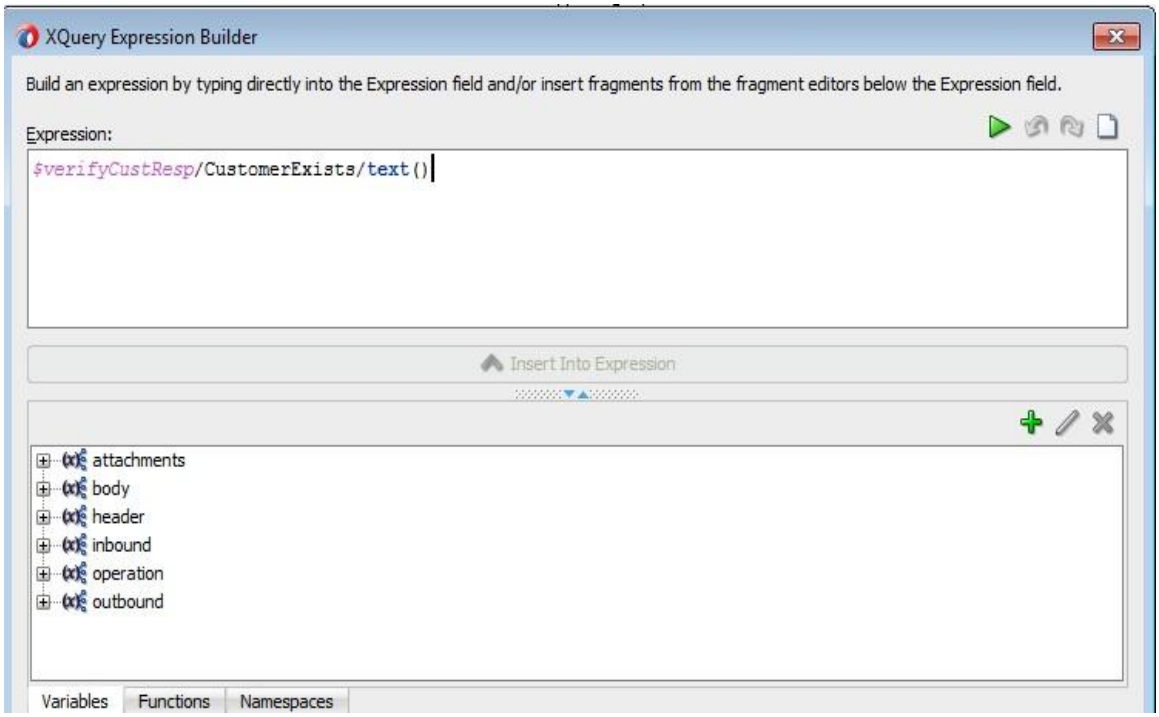
Routing

In this section, you will invoke **CustomerServiceBS** to finish message flow for **Merge Customer**.

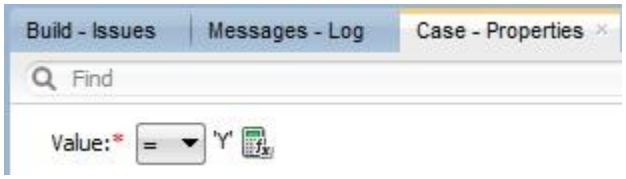
Merge operation has 2 possible routings based on Customer existence. So drag **Routing Table** activity into **Actions** placeholder of **MergeCustomer_Route**.



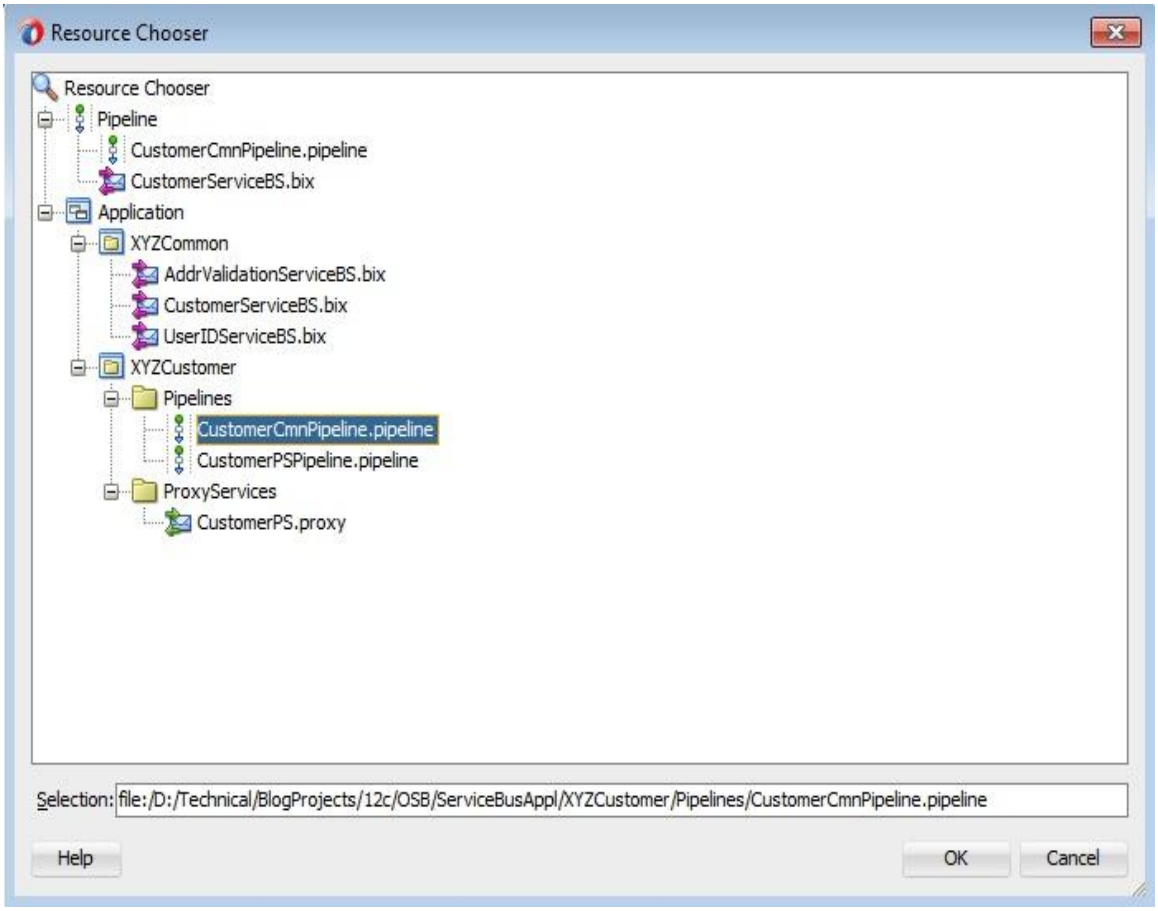
In **Properties** tab, set **Value** in expression builder as shown below to verify customer existence.



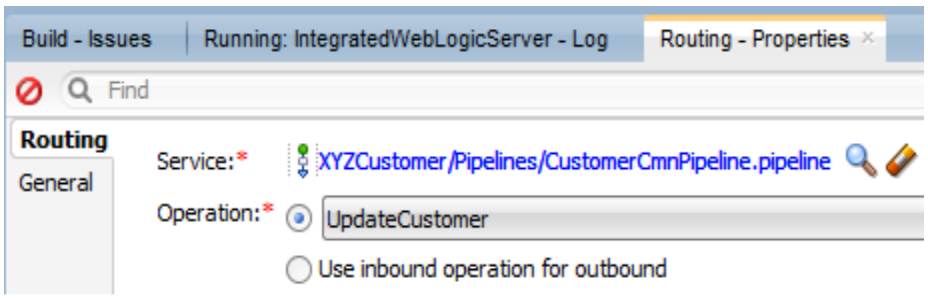
Click **Case** in **Routing Table** and set properties as shown below in **Properties** tab.

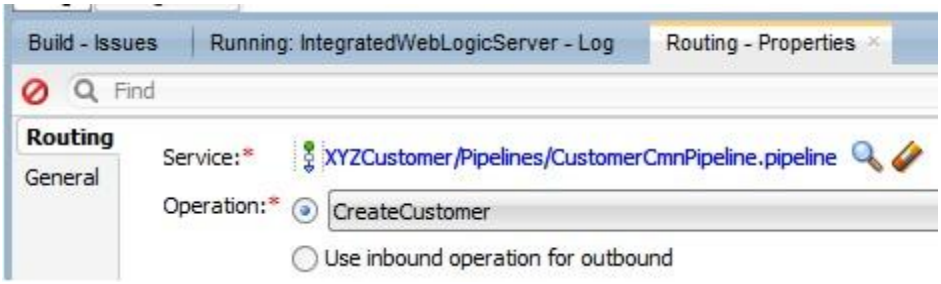


For both **Routings** in **Case** and **Default** branches, in **Properties** tab browse and select pipeline **CustomerCmnPipeline** for **Service** property.

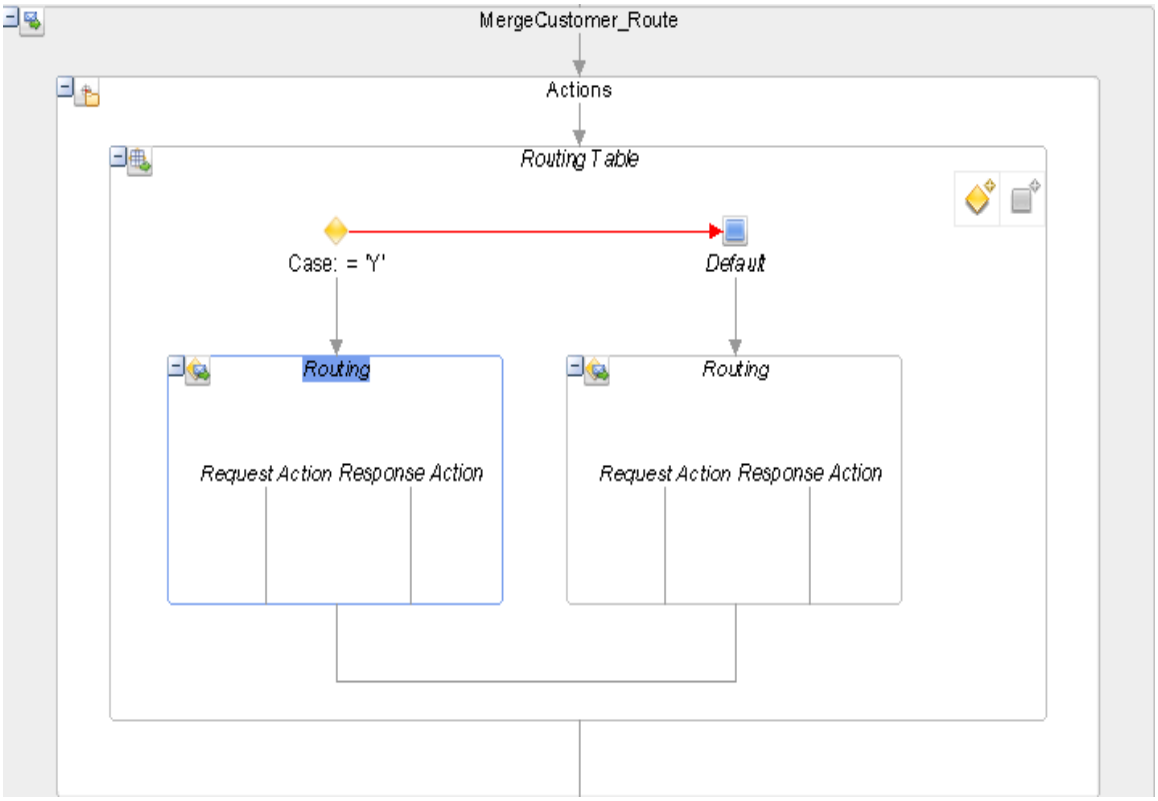


Now select operation as **UpdateCustomer** and **CreateCustomer** for **Case** and **Default** branches respectively as shown below.





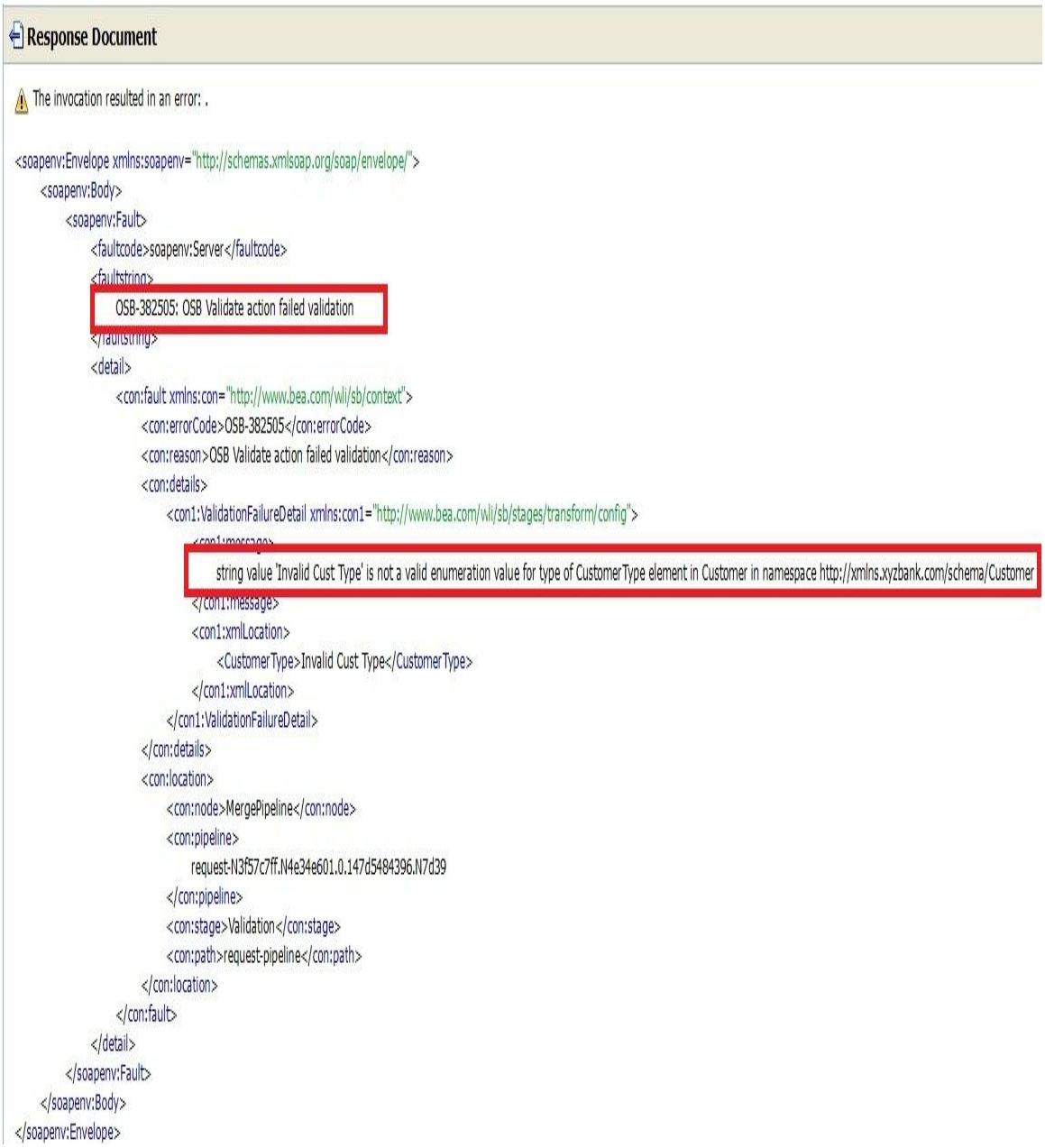
Now your **Routing** node should look like as shown below.



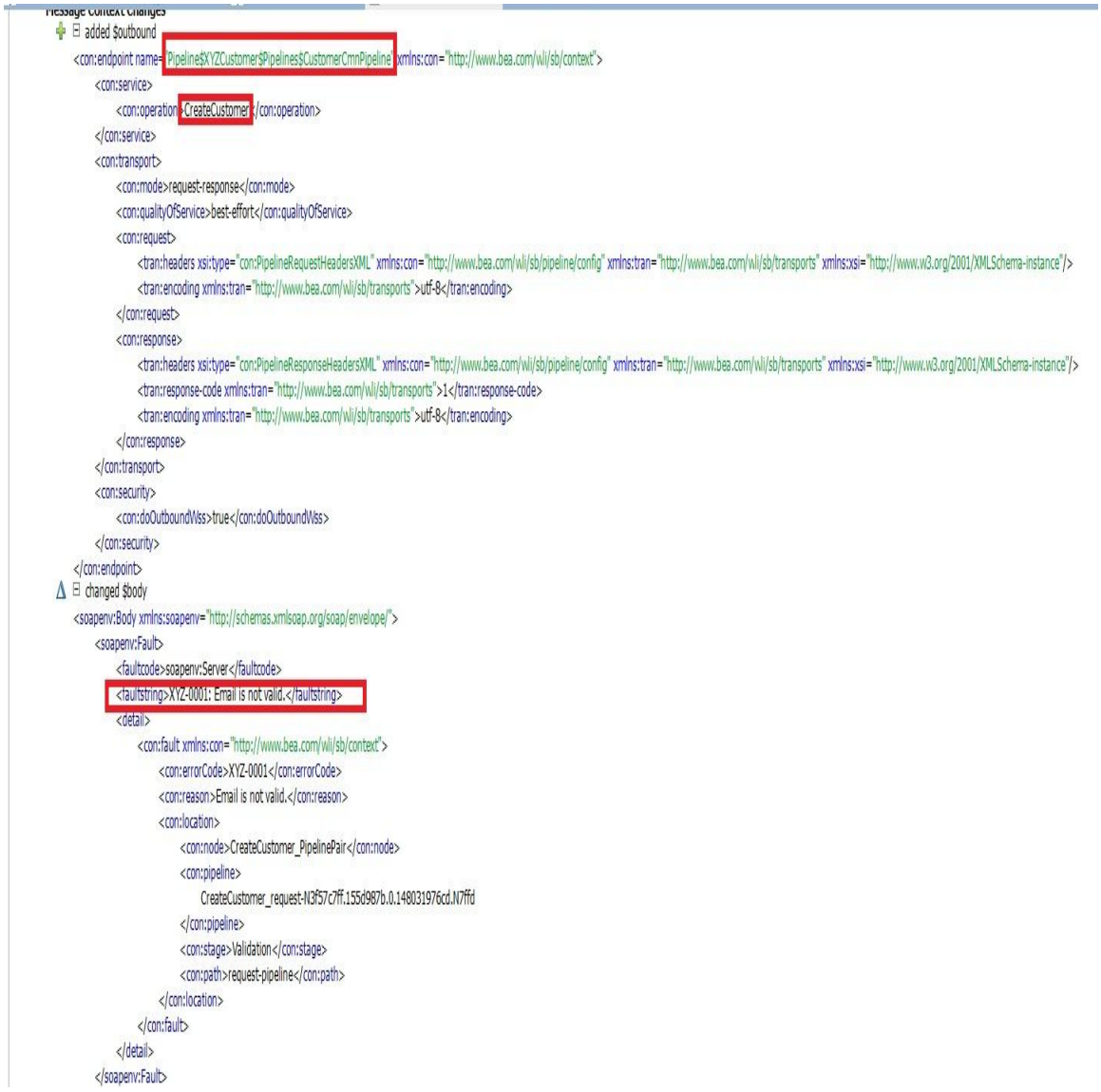
Testing

Deploy both projects or run **Pipeline** directly as shown in **Deploying and Testing** section. You may want to frequently test your proxy service/pipeline during development. Run your pipeline with sample payloads given for **CreateCustomer** and **UpdateCustomer** operations and observe **Flow Trace and Variables** as shown below. You can also run Proxy Service but you will not observe any **Flow Trace**. You can always verify **outbound** context variable in **Flow Trace** to determine operation being called in **Routing/Service Callout**.

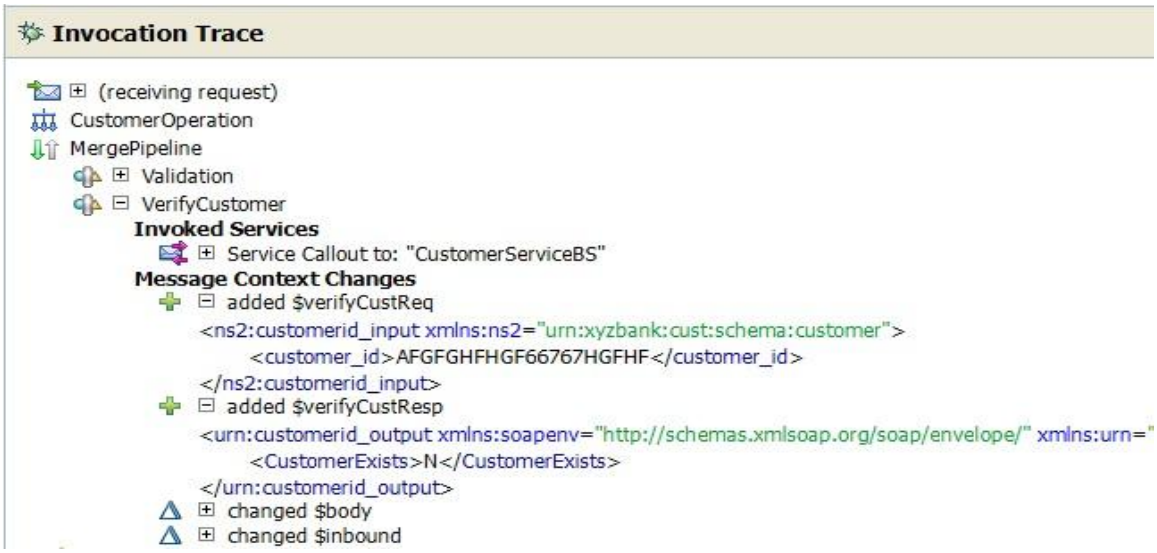
Invalid Customer Type:

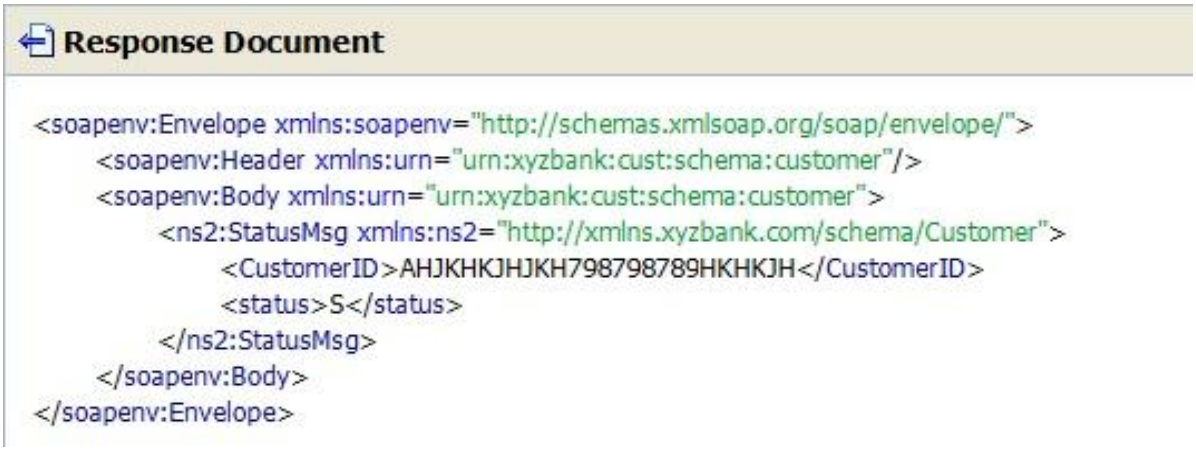


Invalid E-mail and no Customer ID:

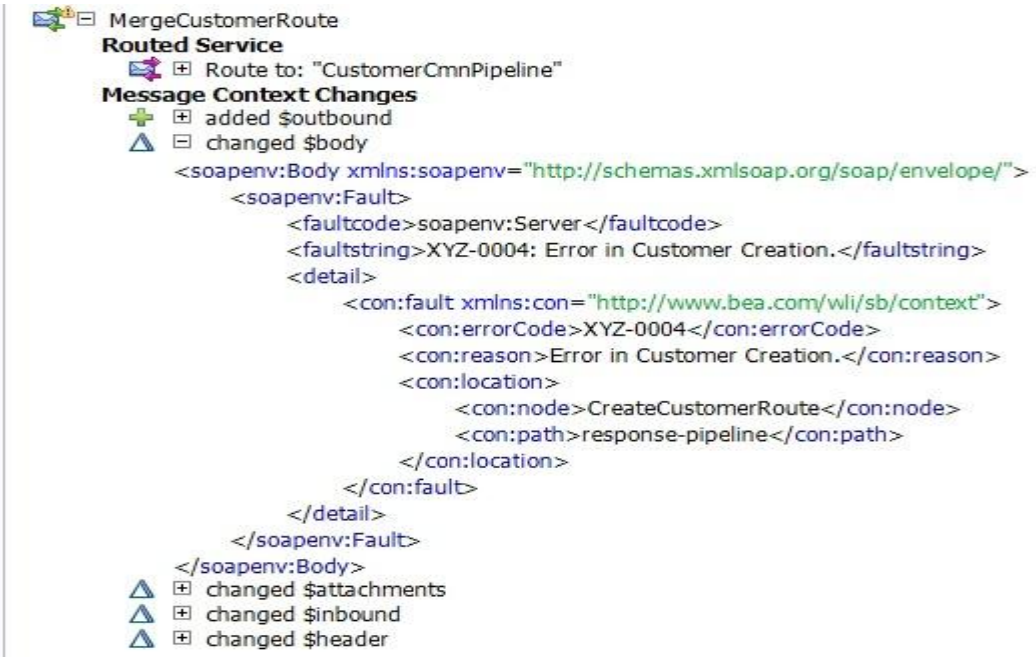


Valid Payload and non-existing Customer ID:





Valid Payload and non-existing Customer ID -Negative Case:



You can repeat test cases for **UpdateCustomer** operation. Remember that you need to send **CustomerID** in input and modify your mock service for **find_customer** to return response **Y**.

Valid Payload and existing Customer ID:

Valid Payload and non-existing Customer ID -Negative Case:

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>OSB-380000</faultstring>
            <detail>
                <con:fault xmlns:con="http://www.bea.com/wli/sb/context">
                    <con:errorCode>OSB-380000</con:errorCode>
                    <con:location>
                        <con:node>MergeCustomerRoute</con:node>
                        <con:path>response-pipeline</con:path>
                    </con:location>
                </con:fault>
            </detail>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

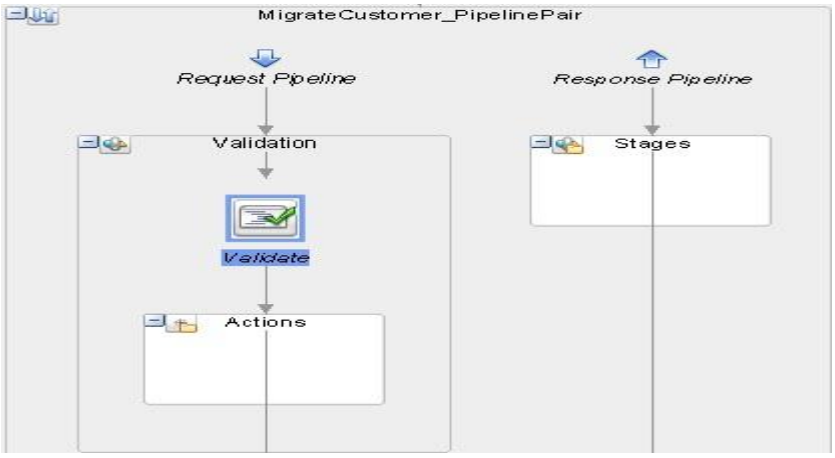
Migrate Customer

In this section, you will complete message flow for **MigrateCustomer** operation.

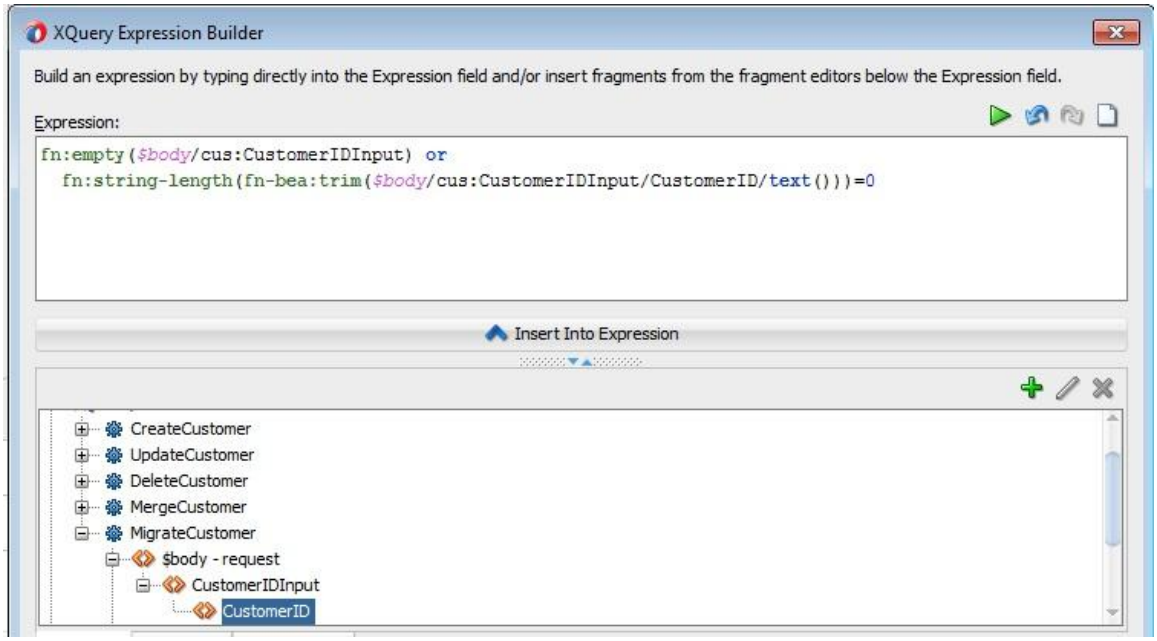
Please note that, migration will not happen this way in real world, and Service Bus may not be the right candidate to perform data migration. Here we consider this case only to demonstrate the parallel processing capabilities of Service Bus.

Validating Payload

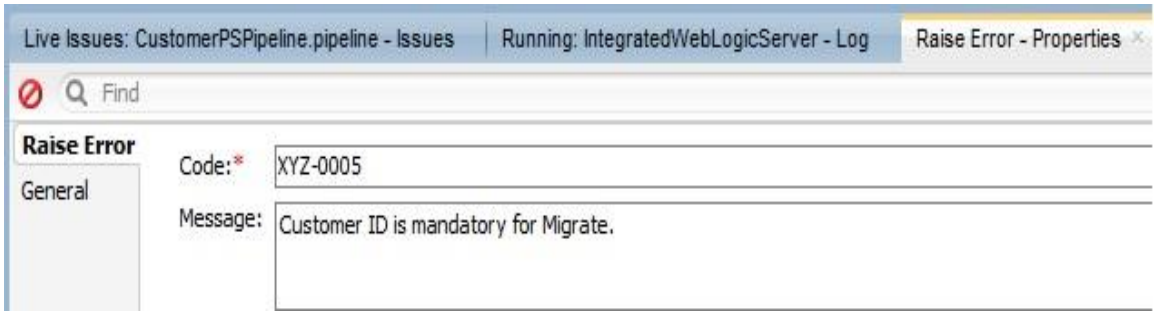
Finish **Validation** stage in **MigrateCustomer** operational branch similar to **DeleteCustomer** for validating **Payload Structure** using **Validate** activity. But you should select **CustomerIDInput** element from **$body/MigrateCustomer** in expression builder for **Location** property.



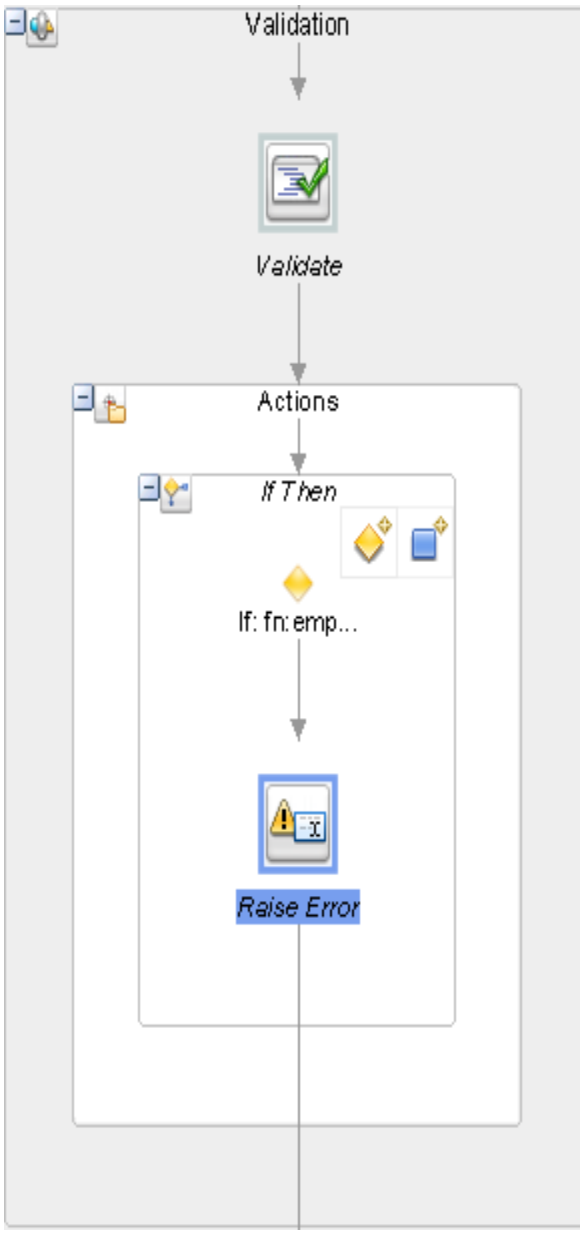
Since this is **Migrate** operation, service consumer should send **CustomerID** mandatorily in input and your message flow should enforce the same. So drag **If-Then** activity into **Validation** stage from **Flow Control** and set condition as shown below. Alternatively, you can select these string functions from **Functions -> XQuery 1.0 Functions -> Strings -> General Functions**.



Drag **Raise Error** activity into **If** branch from **Flow Control** and set properties as shown below.



Now your **Validation** stage should look like below.



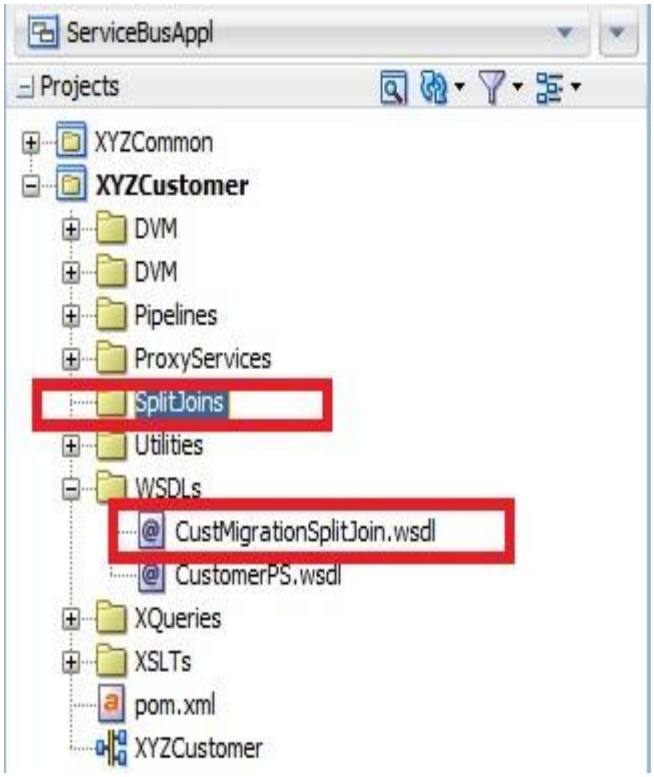
Creating Split Join

In this section, you will aggregate customer information using Split Join by invoking appropriate Business Services in parallel.

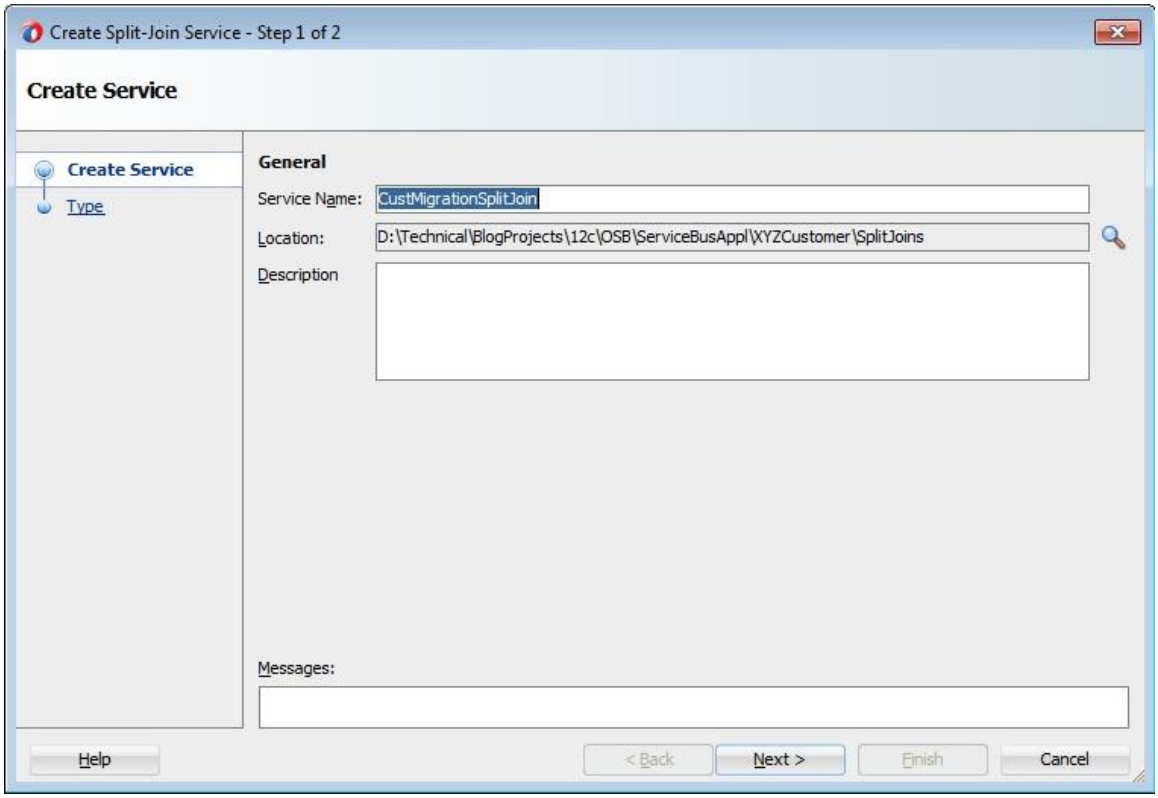
Split join allows you to split request payload so that you can call multiple services concurrently as opposed to sequential processing. This way, you will achieve improved service performance. Split join can only be created based on WSDL so create a new one in WSDLs folder, having single operation with following. You can come up with your own WSDL or use the one provided with this tutorial from here.

- **Customer ID** as input.
- Customer information including **Customer, Orders and Accounts** Info as output.

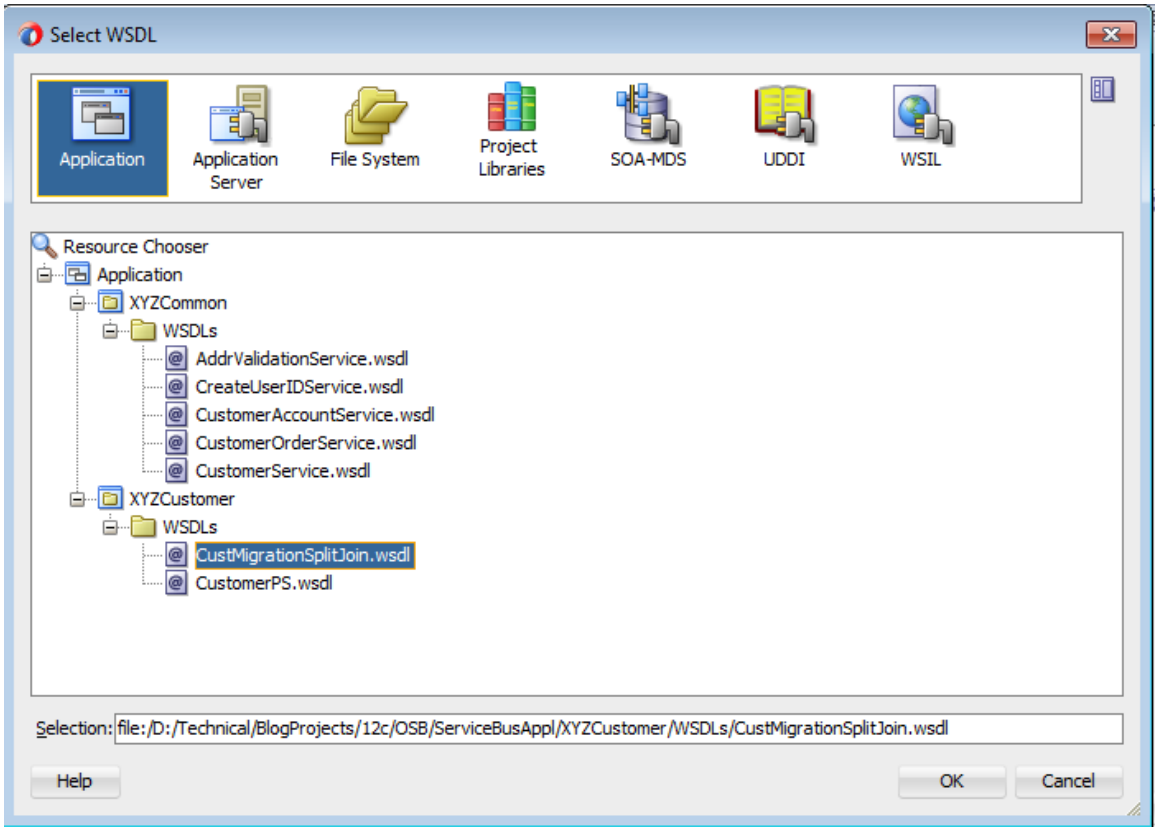
Copy **CustMigrationSplitJoin.wsdl** into **WSDLs** folder of **XYZCustomer** project from here and create a new folder **SplitJoins** as shown below.



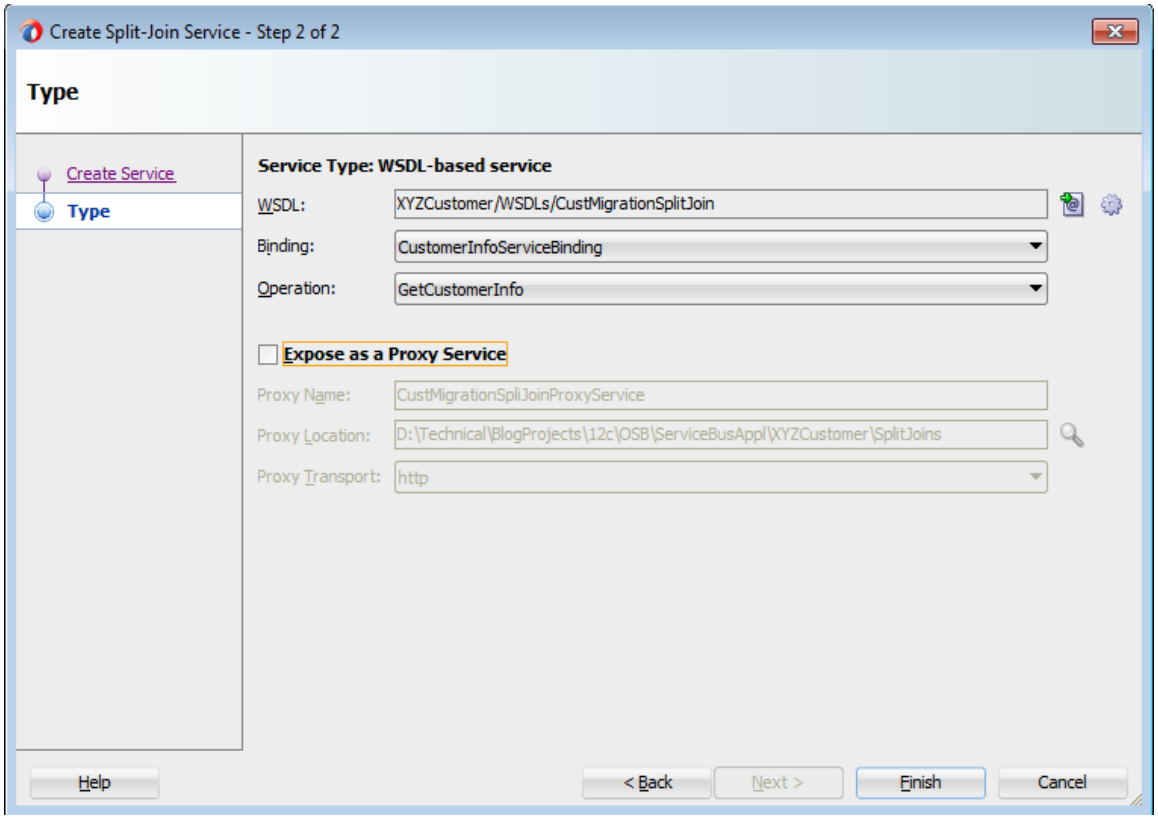
Open Service Bus overview editor and drag **Split Join** into middle swim lane from **Components -> Service Bus -> Resources** and finish the creation using wizard as shown below. Make sure that you select **SplitJoins** folder as the location in wizard.



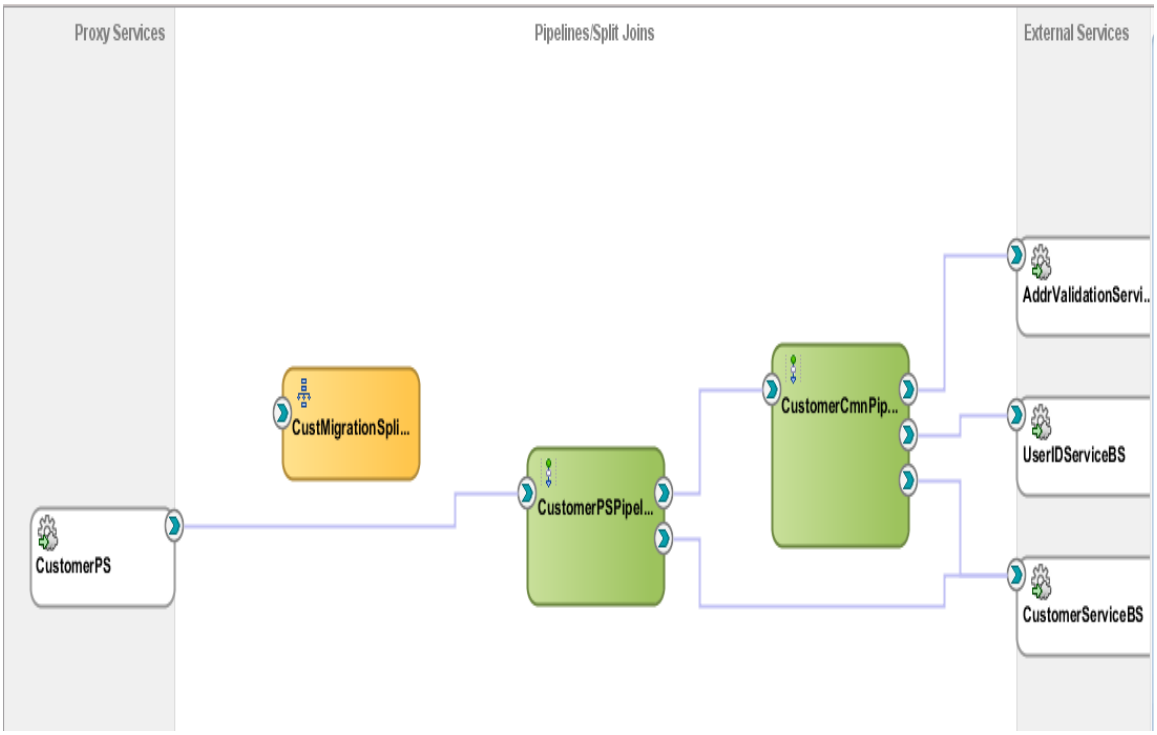
Click **Next** and select WSDL **CustMigrationSplitJoin.wsdl**.



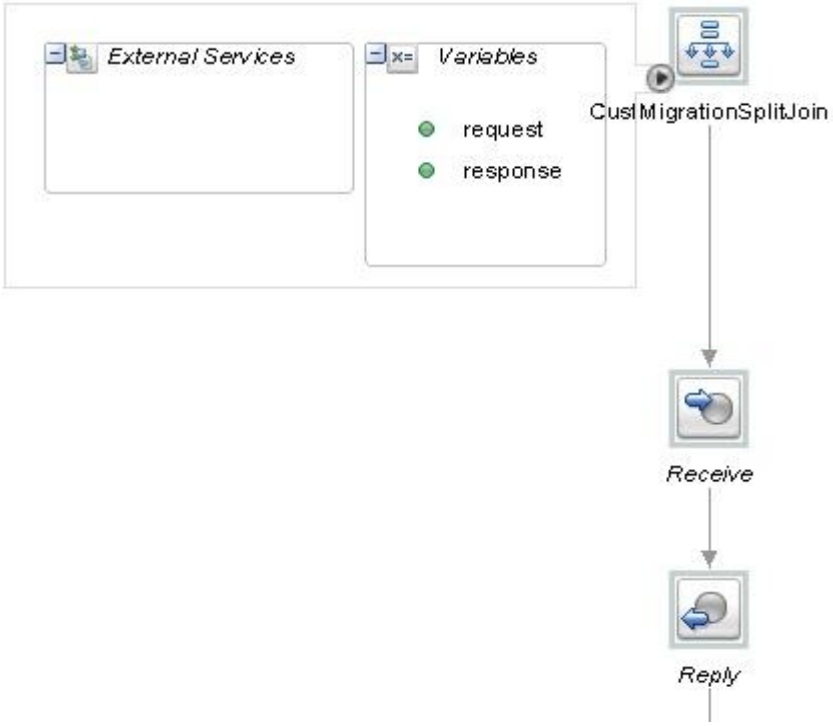
Click **OK** and uncheck **Expose as a Proxy Service.**



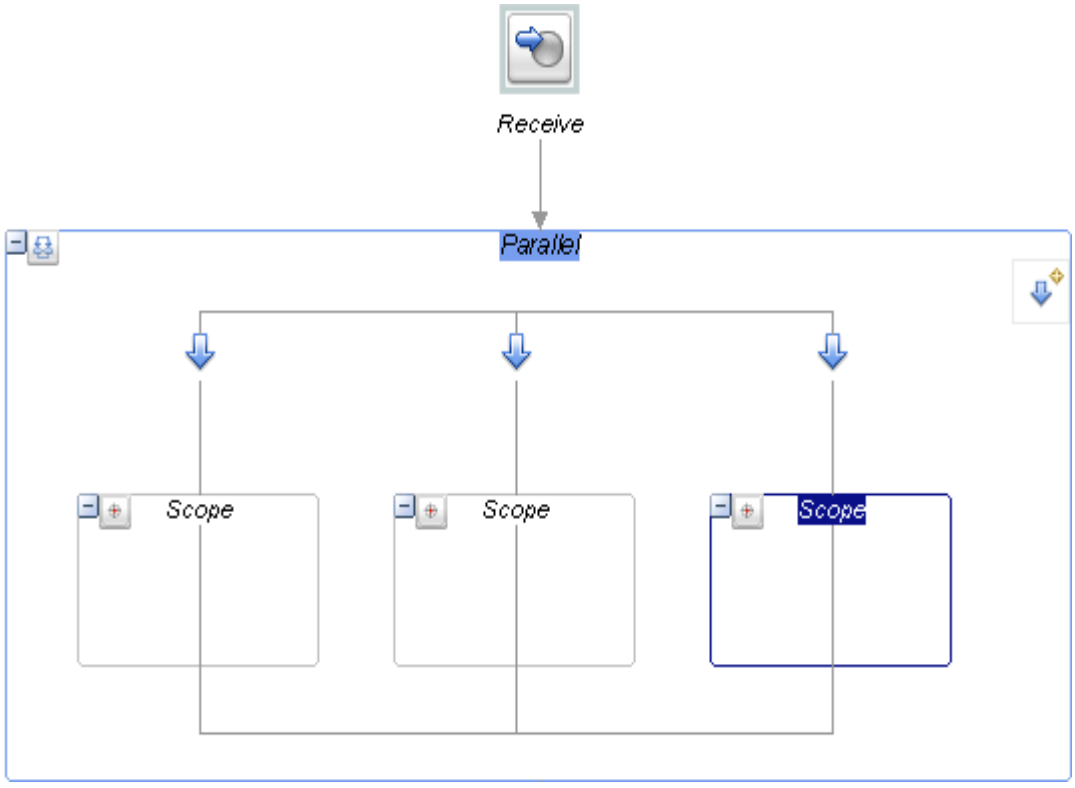
Now your Service Bus overview editor should look like below.



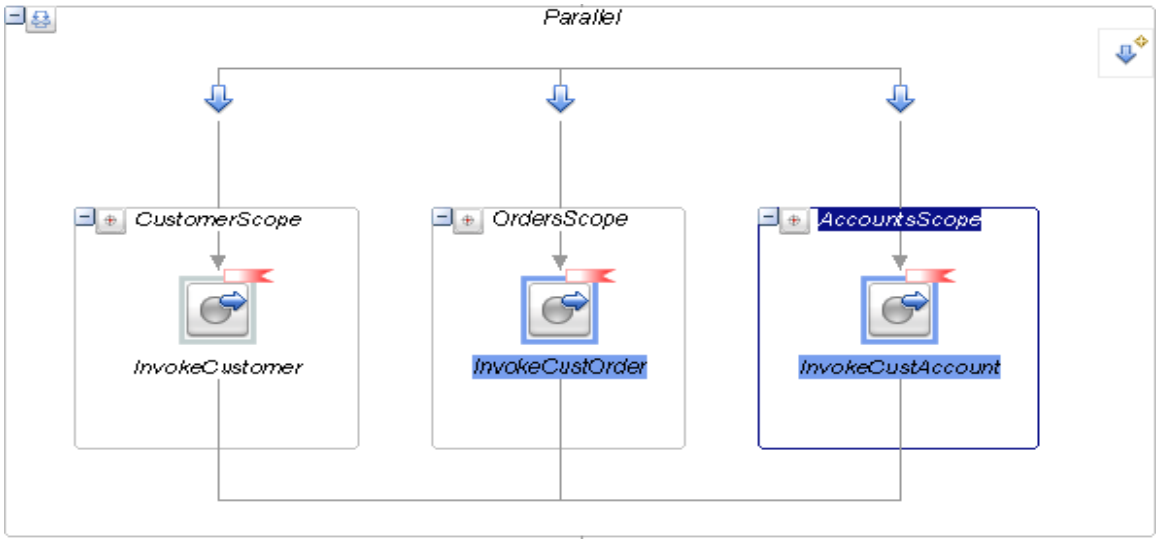
Also your initial split join flow should look like below.



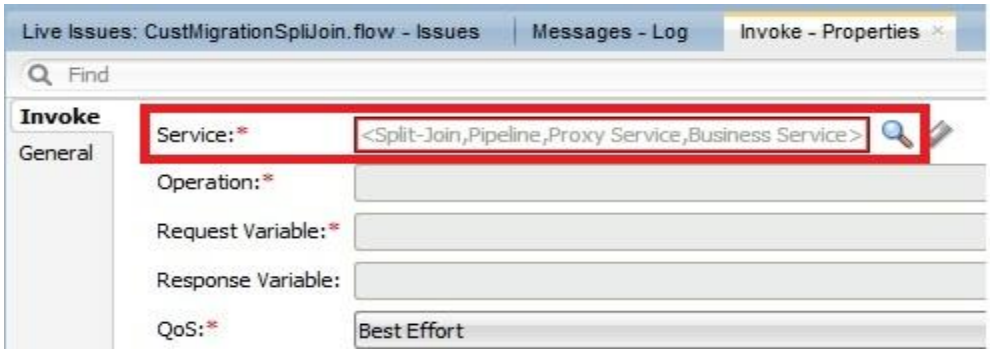
Drag **Parallel** activity after **Receive** as shown below from **Flow Control** and add another branch by clicking  . The **Parallel** activity allows you to invoke different services concurrently.



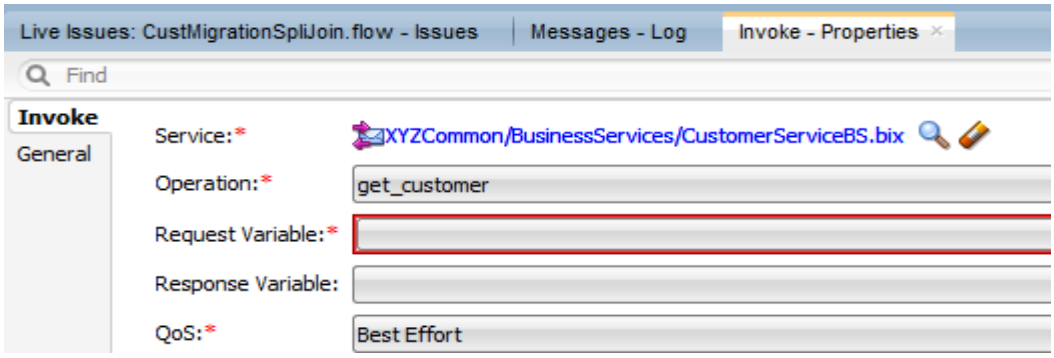
In **Properties** tab, modify **Name** for all the **Scope** activities as shown below. Drag **Invoke Service** activity into each of these **Scope** activities in all 3 parallel branches from **Communication** so that you can invoke Business services. Also modify **Name** property for **Invoke** activities in **Properties** tab.

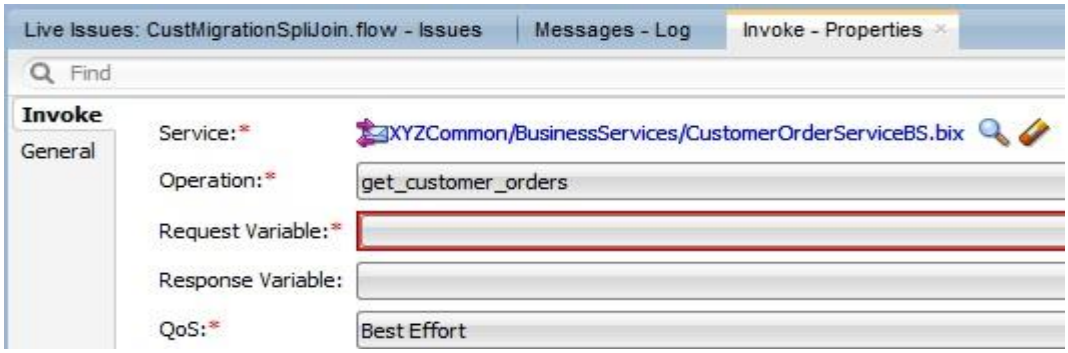


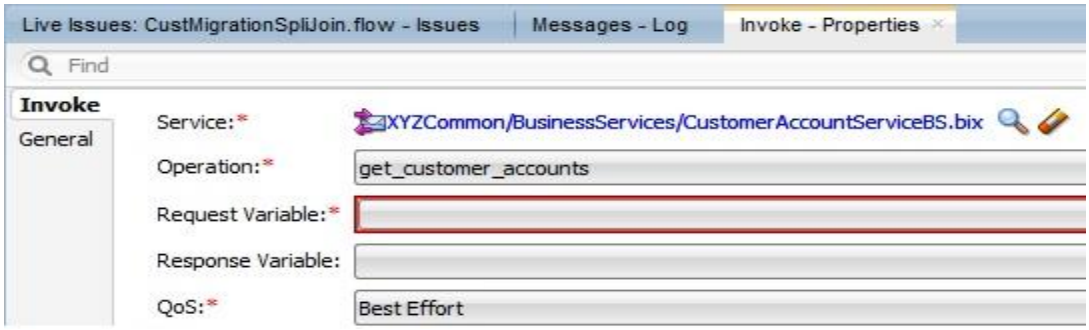
In **Properties** tab, browse and select **CustomerServiceBS, CustomerOrderServiceBS and CustomerAccountServiceBS** for **Service** property of **InvokeCustomer**, **InvokeCustOrder**, **InvokeCustAccount** activities respectively.



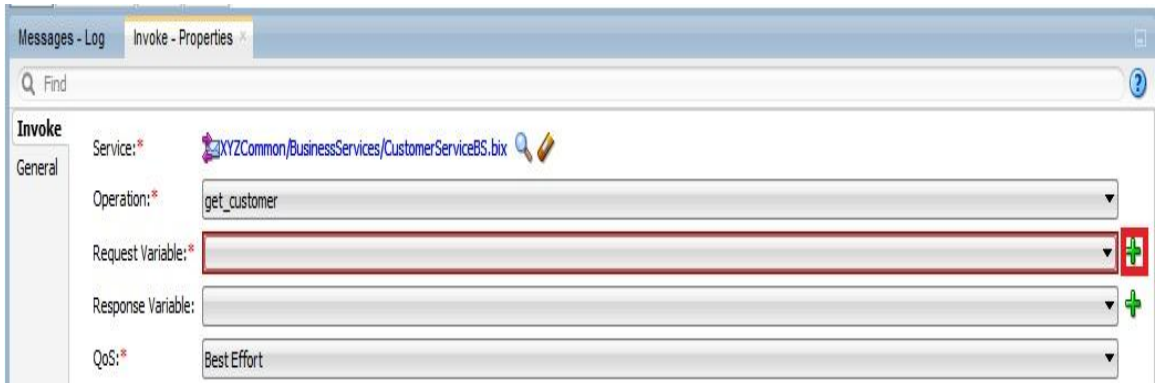
Select **Operation** as shown below for **InvokeCustomer**, **InvokeCustOrder** , **InvokeCustAccount** activities respectively.

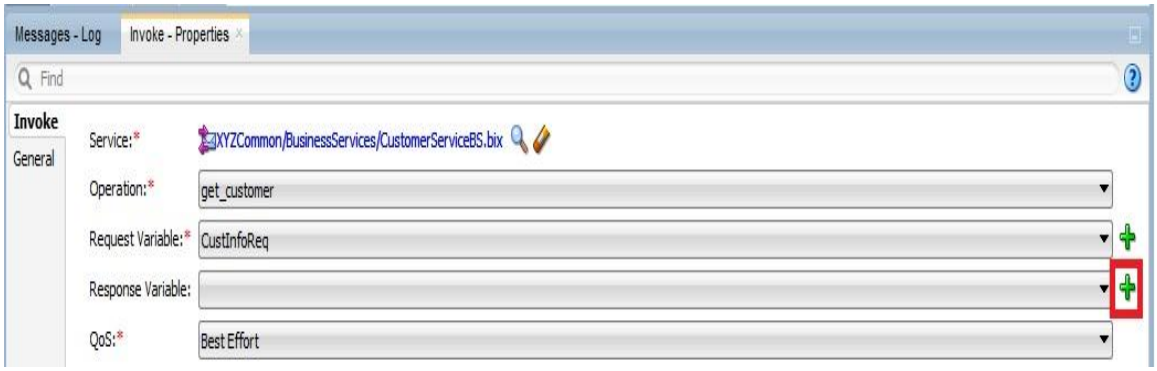


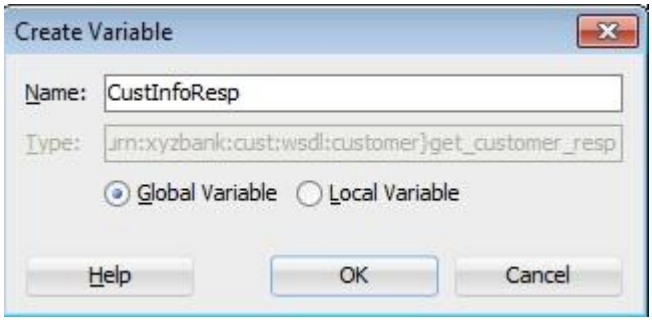




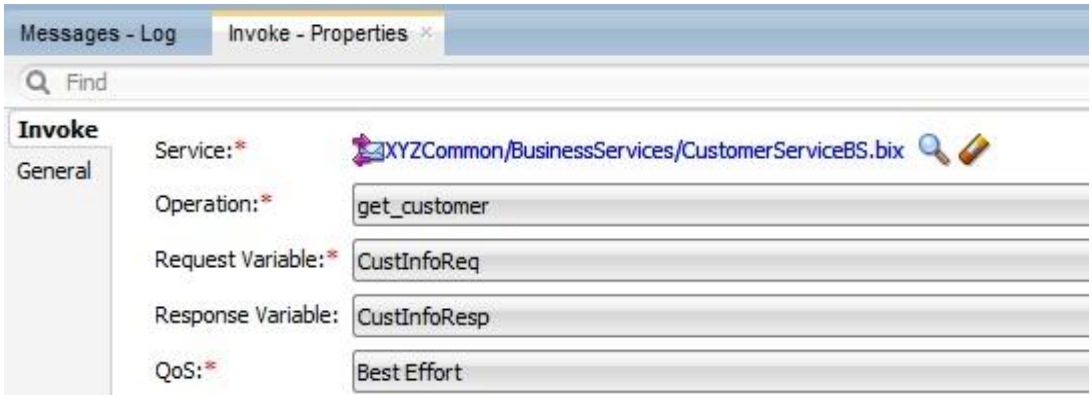
For **InvokeCustomer**, create **Request** and **Response** variables as shown below. **Request Variable** can be created as local variable as you don't use it outside the scope.



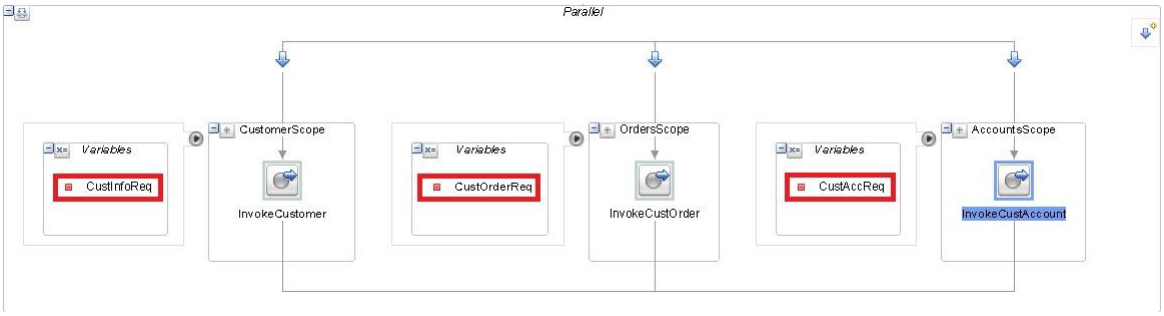




Now your properties tab for **InvokeCustomer** should look like below.

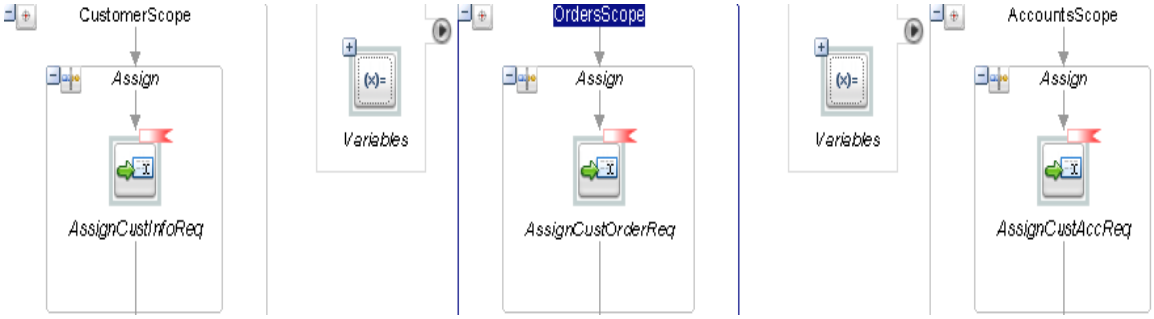


Similarly create variables **CustOrderReq**, **CustOrderResp** for **InvokeCustOrder** and **CustAccReq**, **CustAccResp** for **InvokeCustAccount** respectively. Now local variables created for each **Scope** activity should look like below.



Your **global** variables would be shown at Split Join level.

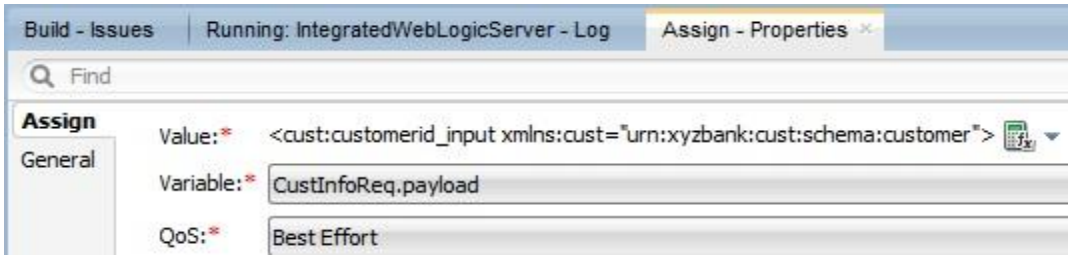
Drag **Assign** activity from **Assign Operations** into each of the scope activity and name them as **AssignCustInfoReq**, **AssignCustOrderReq** and **AssignCustAccReq** respectively in **Properties** tab.



In **Properties** tab, set properties for **AssignCustInfoReq** as shown below. This will populate the request variable with the payload expected by **CustomerServiceBS**. Since there are not many fields to map, we have chosen this way to populate the payload avoiding new transformations.

Value:

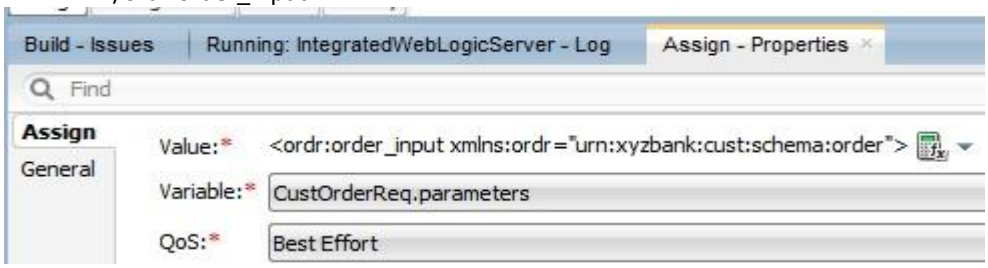
```
<cust:customerid_input xmlns:cust="urn:xyzbank:cust:schema:customer">
        <customer_id>{$request.parameters/customer_id/text()}</customer_id>
</cust:customerid_input>
```



In **Properties** tab, set properties for **AssignCustOrderReq** as shown below. This will populate the request variable with the payload expected by **CustomeOrderServiceBS**.

Value:

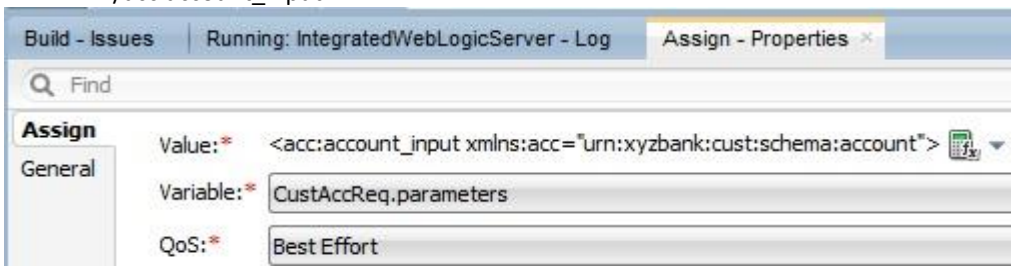
```
<ordr:order_input xmlns:ordr="urn:xyzbank:cust:schema:order">
  <ordr:customer_id>{$request.parameters/customer_id/text()}</ordr:customer_id>
</ordr:order_input>
```



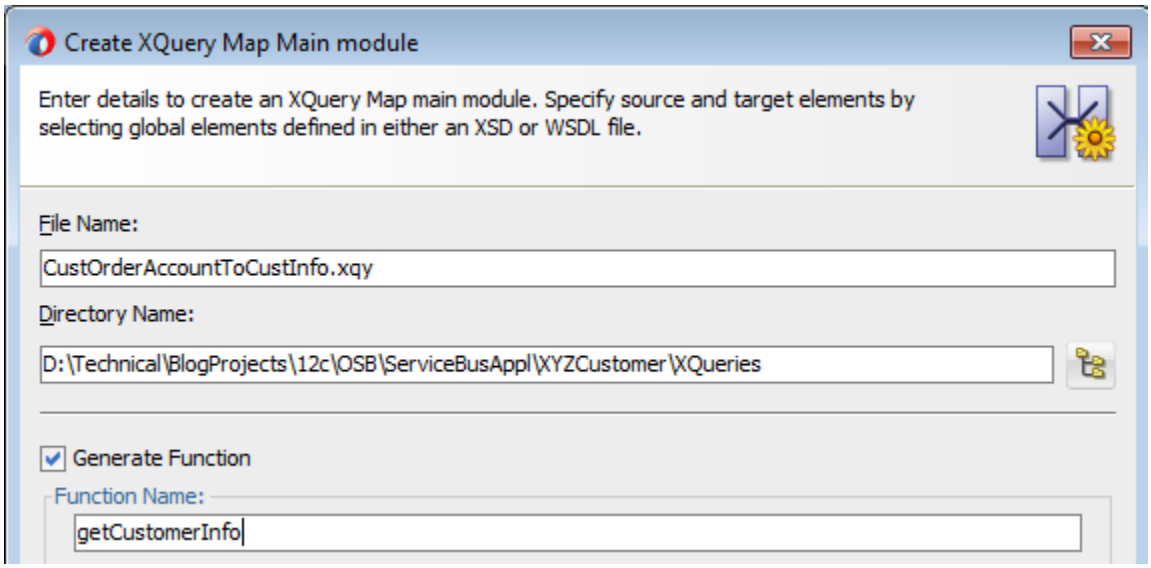
In **Properties** tab, set properties for **AssignCustAccReq** as shown below. This will populate the request variable with the payload expected by **CustomeAccountServiceBS**.

Value:

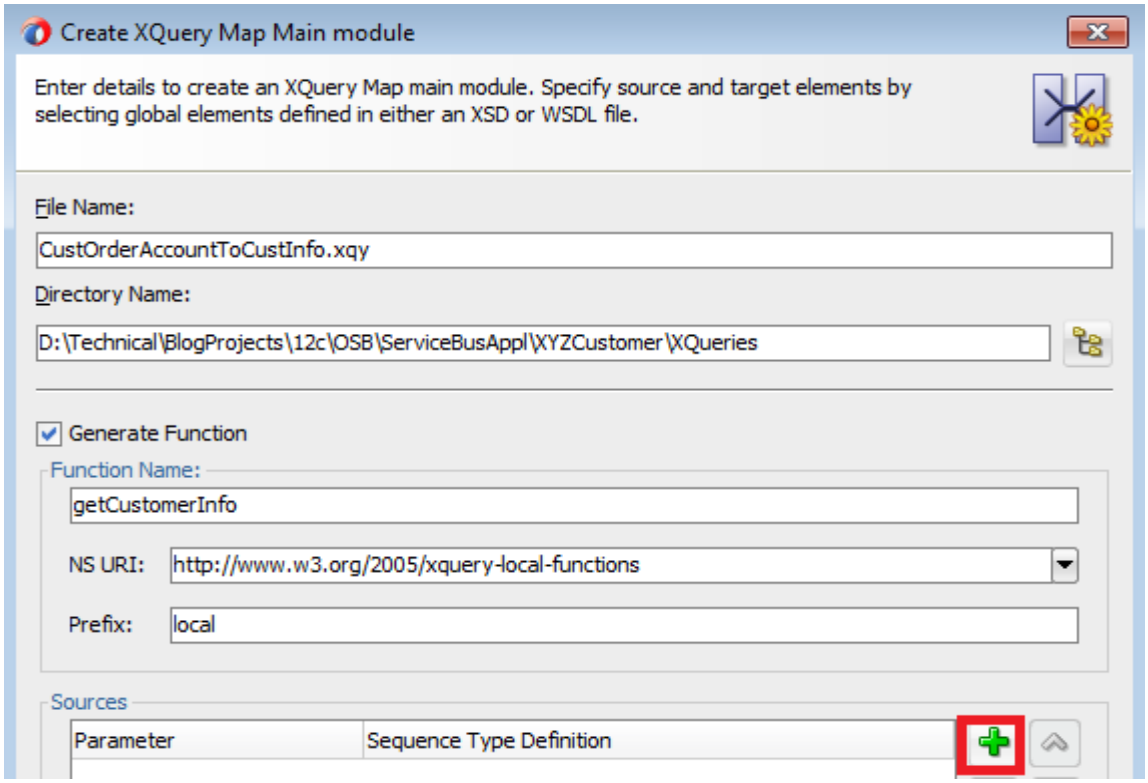
```
<acc:account_input xmlns:acc="urn:xyzbank:cust:schema:account">
        <acc:customer_id>{$request.parameters/customer_id/text()}</acc:customer_id>
</acc:account_input>
```

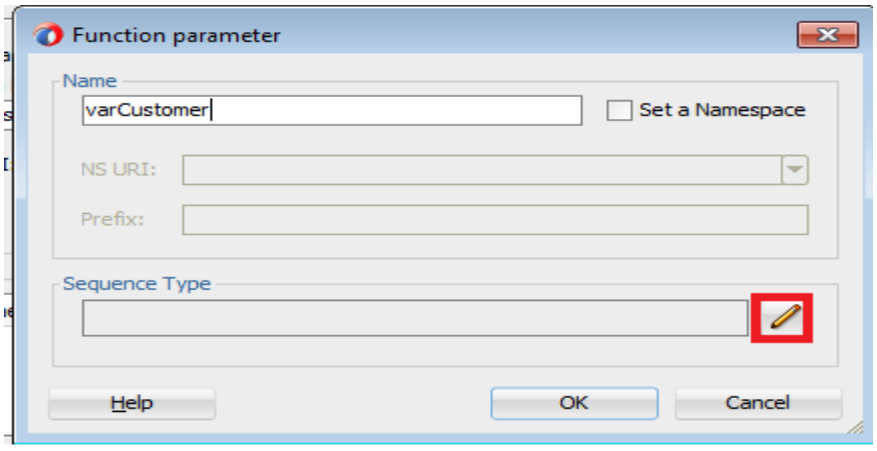


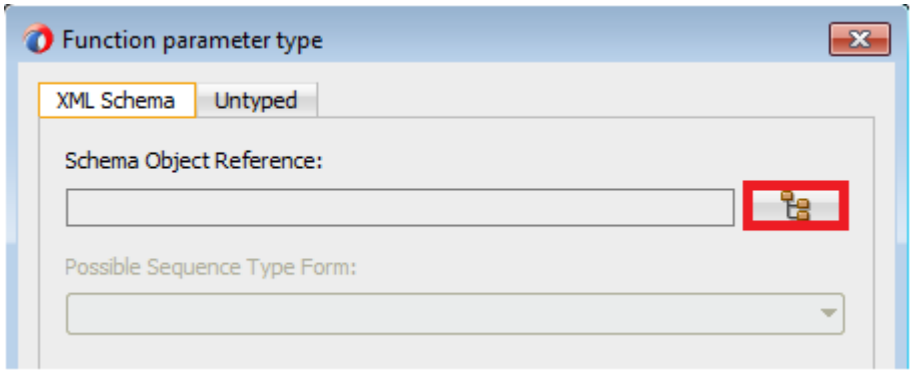
This would fetch **Customer**, **Order** and **Account** information concurrently by invoking respective business services. You need to transform all these responses into one before returning back to caller. So you need a transformation to do this. Create a XQuery map in **XQueries** folder of **XYZCustomer** project as shown below. Give name as **CustOrderAccountToCustInfo.xqy**.

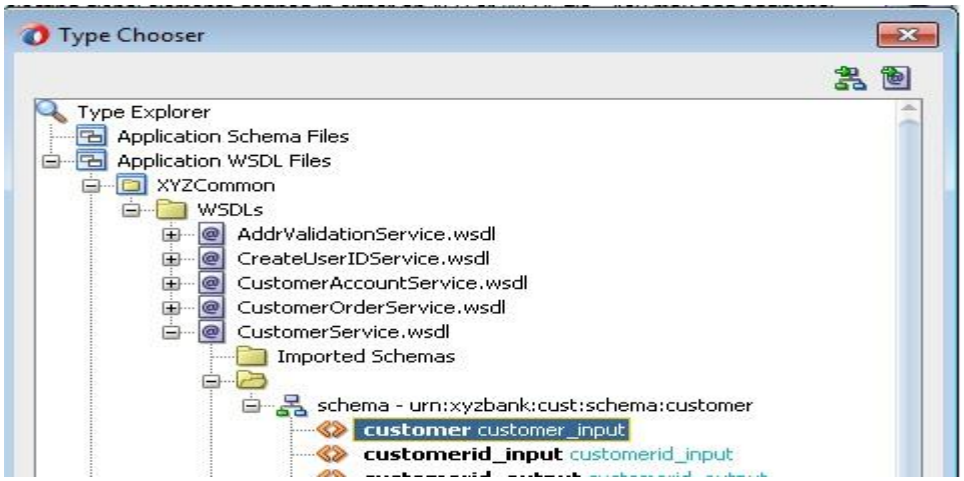


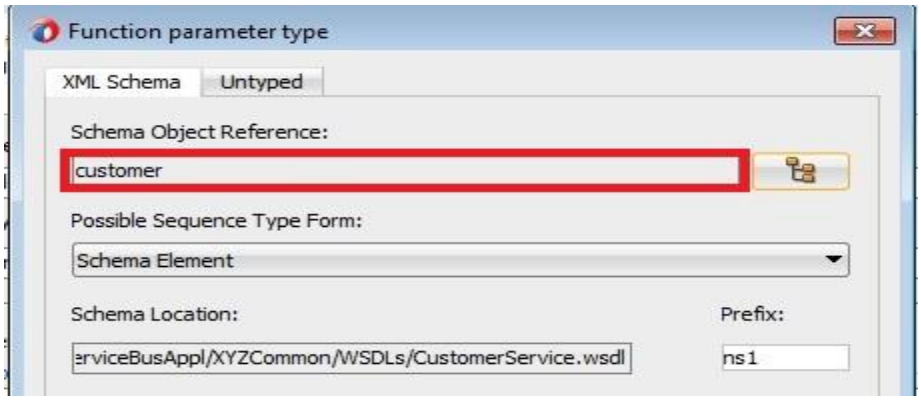
Choose the **Source** for XQuery map using steps shown below.



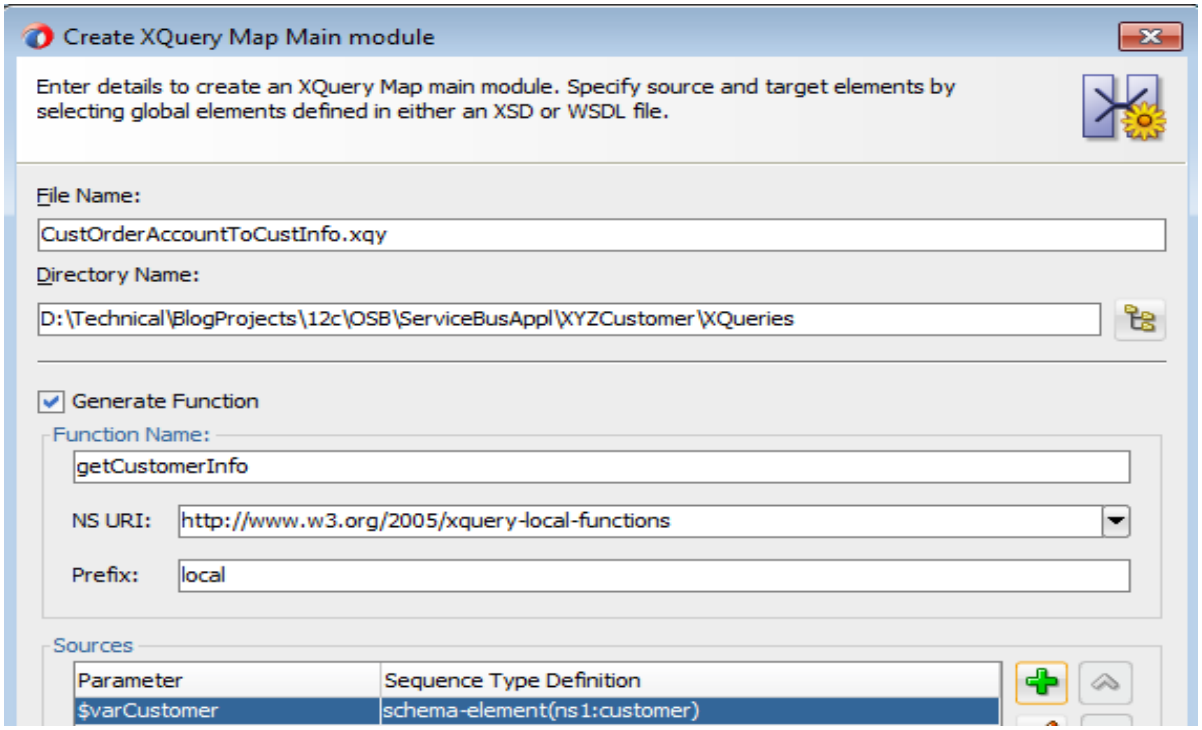




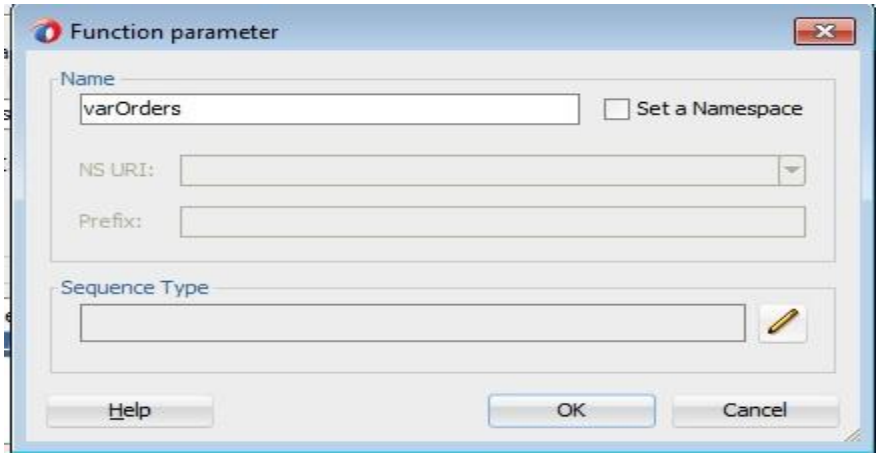


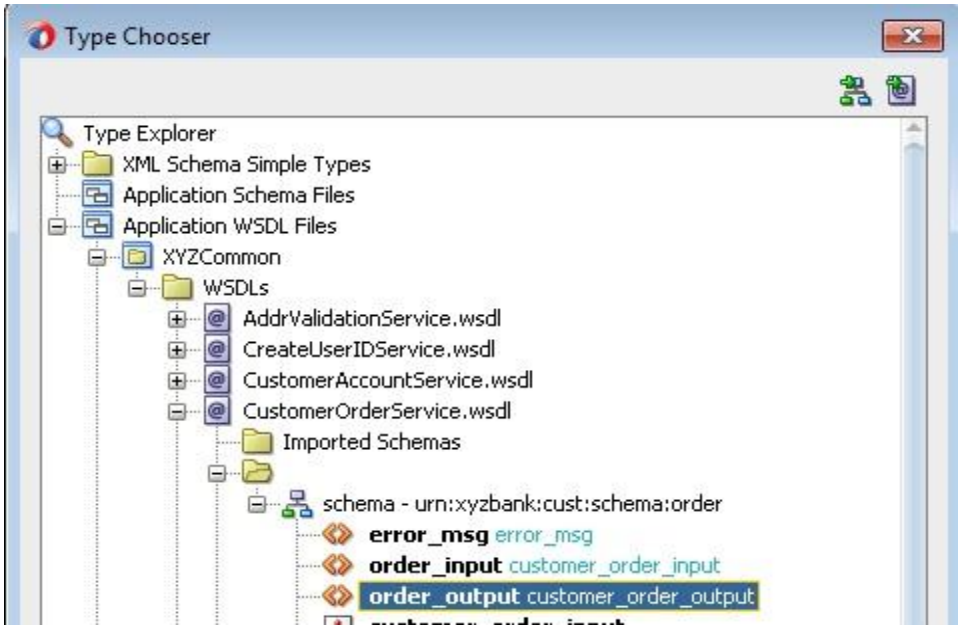


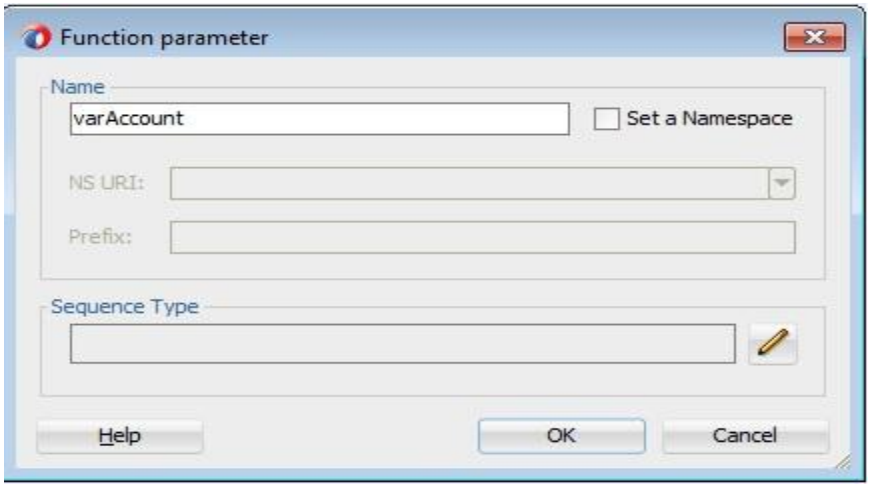
Go back to **Create XQuery Map** window by clicking **OK** twice.

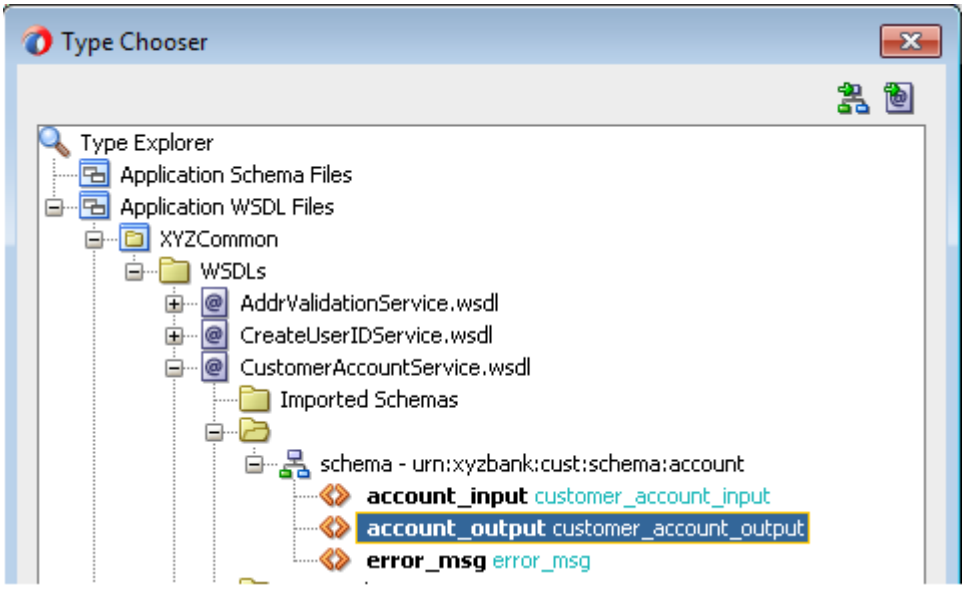


Similarly add 2 more parameters as shown below to accept **Account** and **Orders** information as the input.

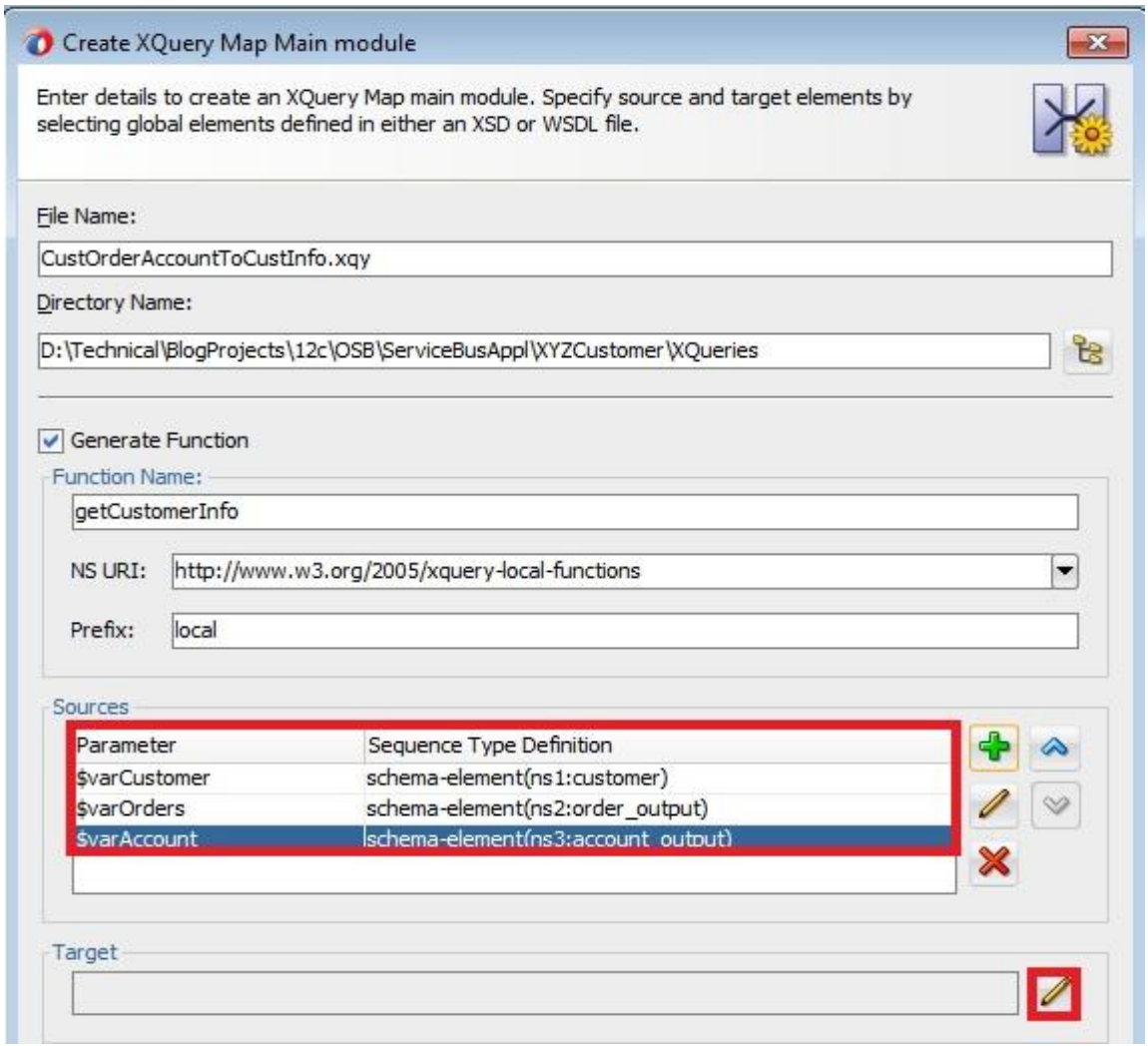




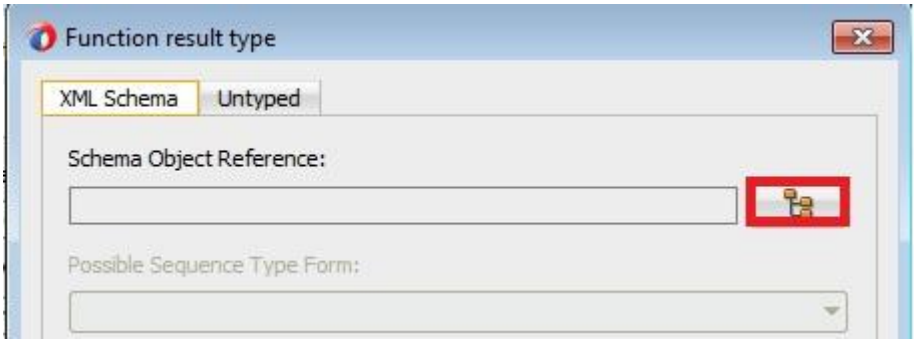


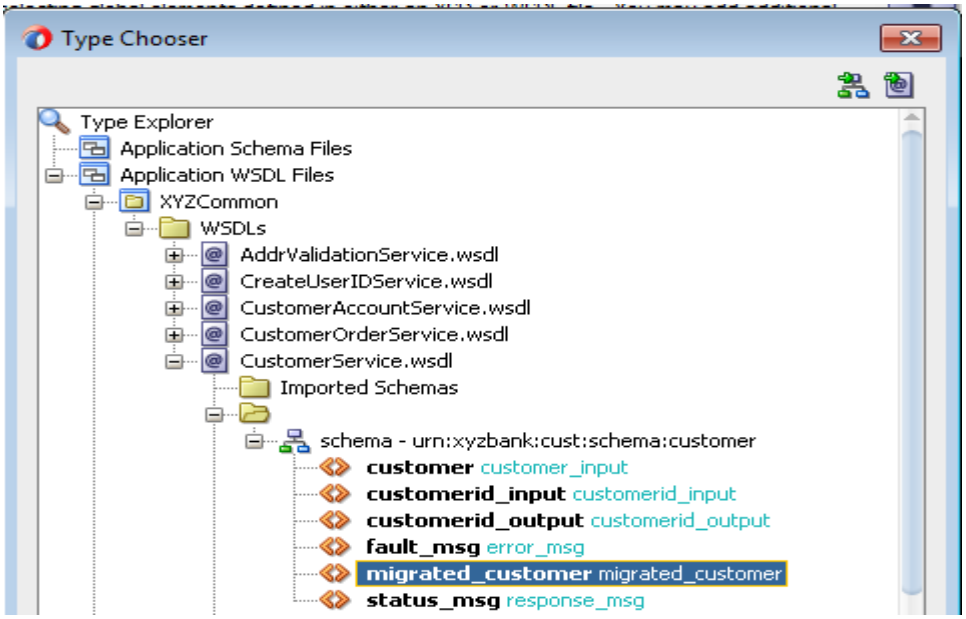


Your Create XQuery Map window should look like below.

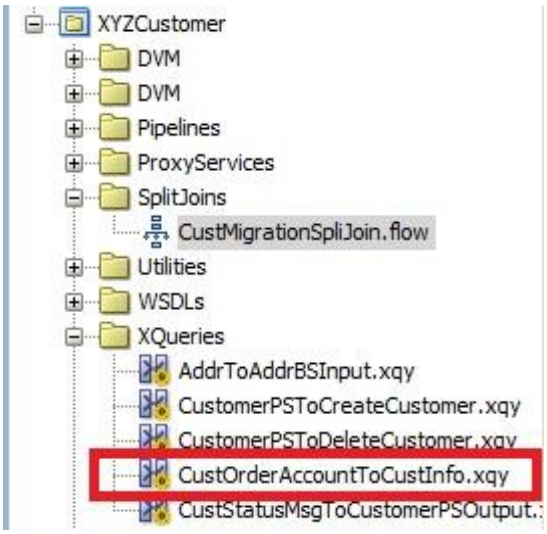


Select **Target** using steps mentioned below.

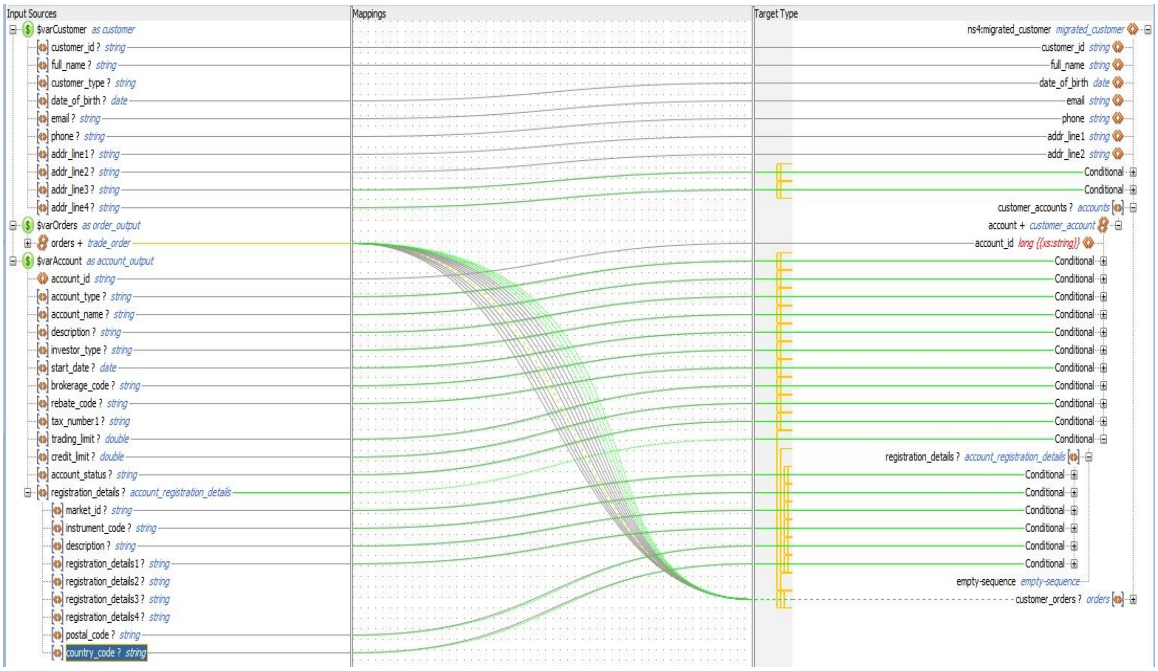




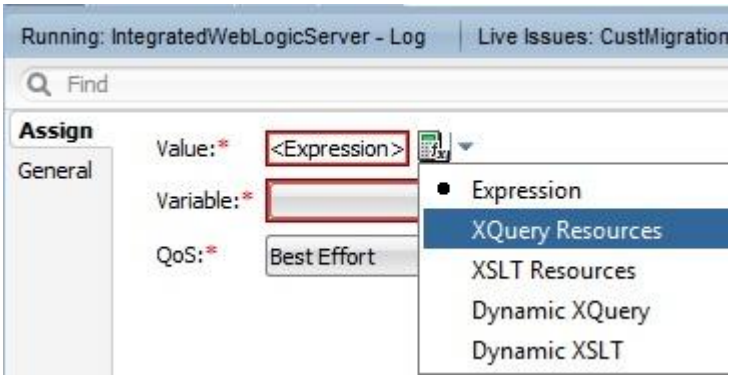
Click **OK** twice to finish XQuery map creation and should be visible in **XQueries** folder as shown below.

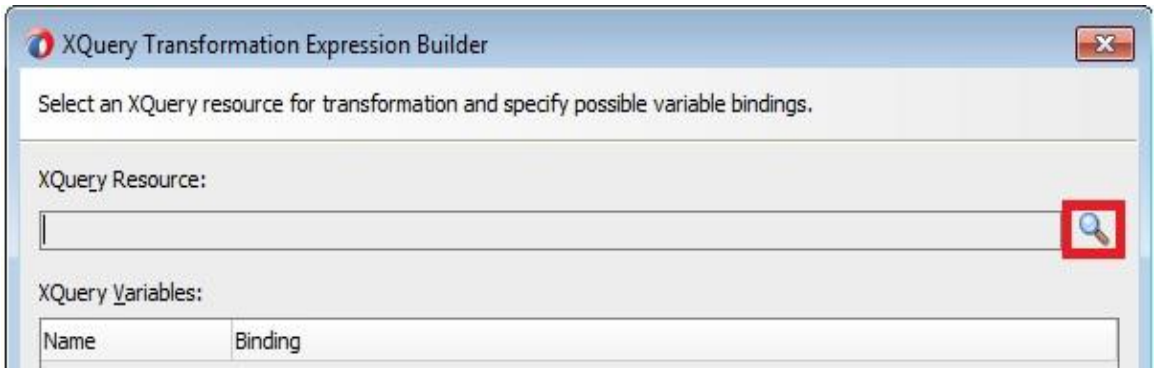


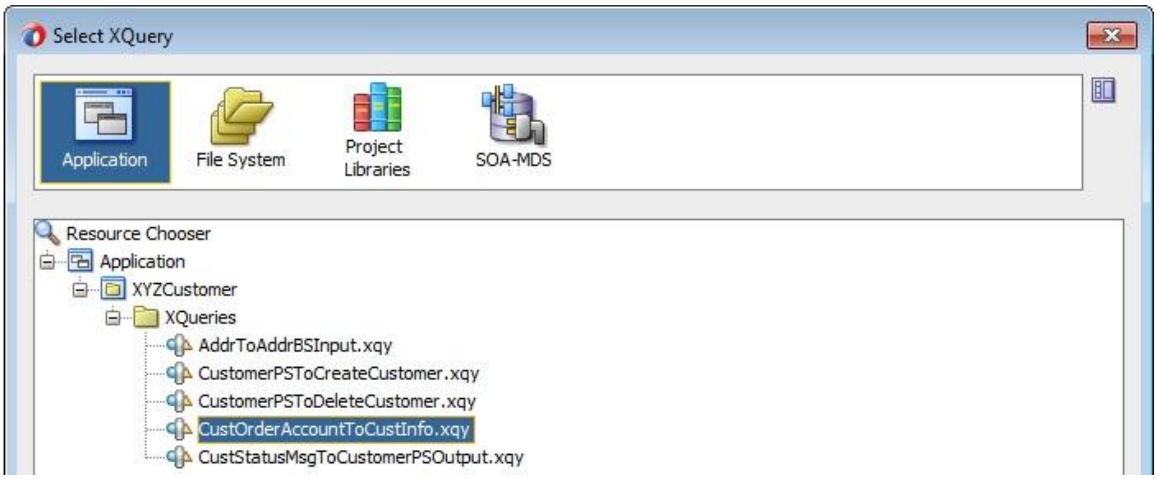
Finish the mapping as required and shown below. Observe the sources and target structures. You may want to test your XQuery using steps mentioned in previous sections.

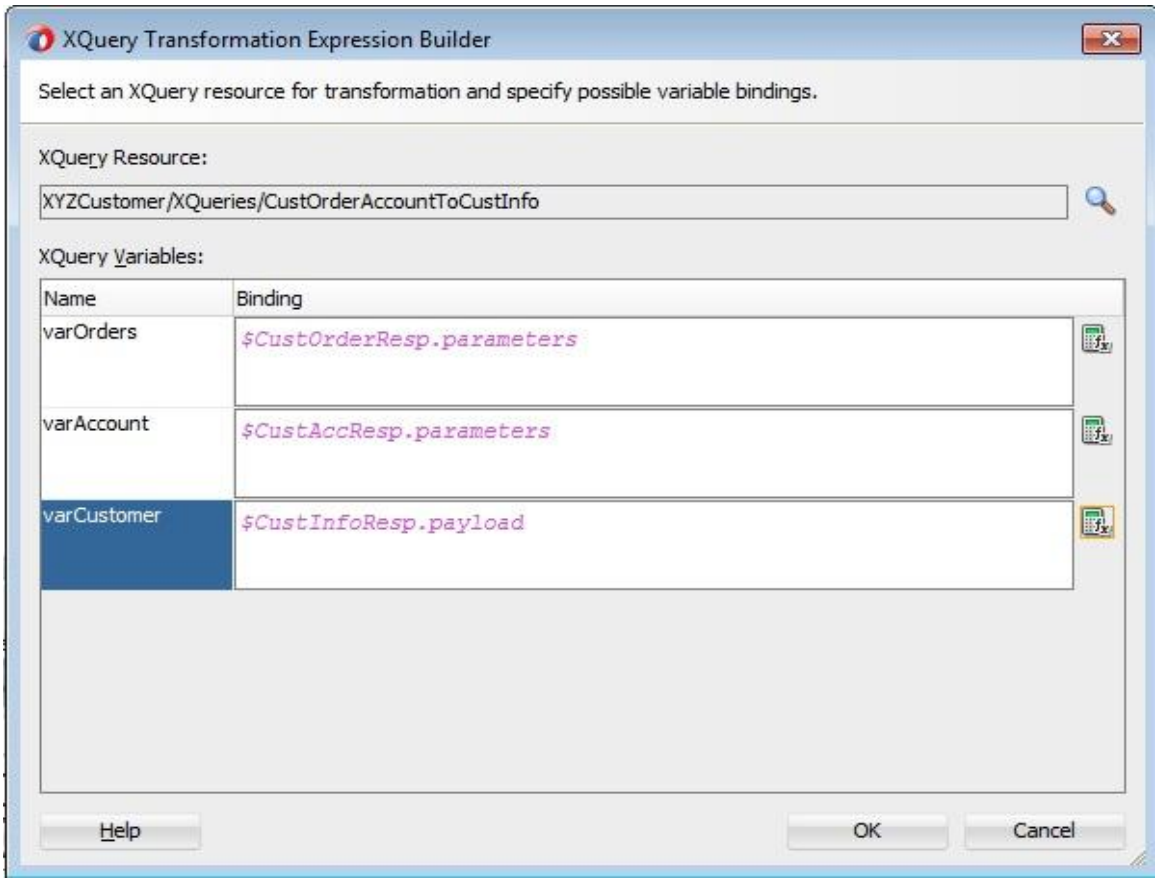


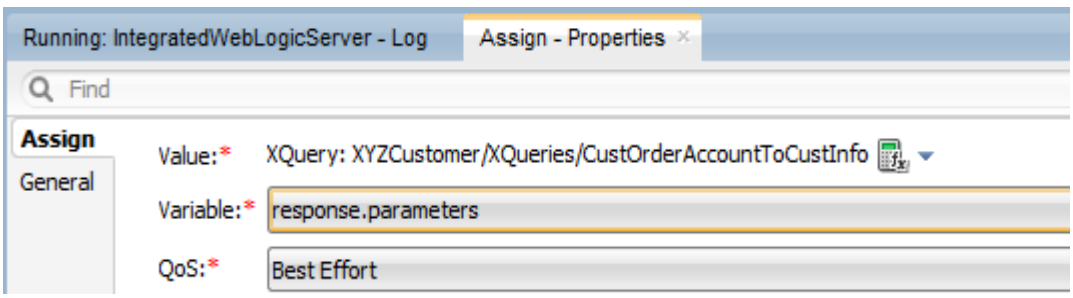
Now you have to use this XQuery map in Split join to merge 3 different responses into one and reply with merged payload. So drag **Assign** after **Parallel** activity from **Assign Operations and** set properties as shown below.



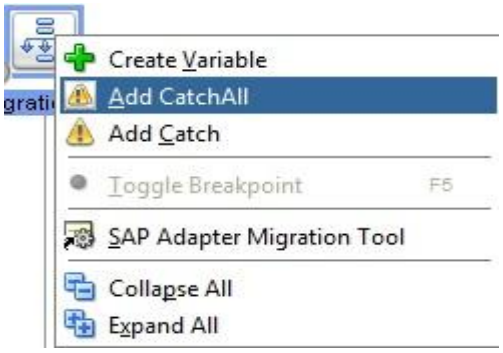


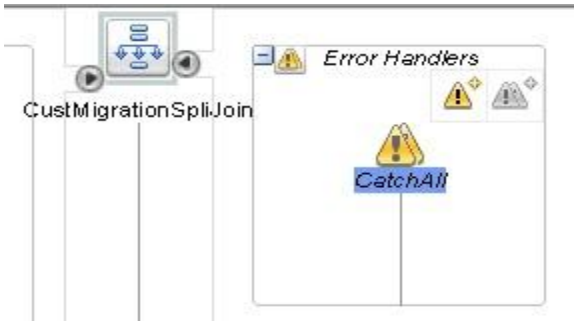

Select XQuery
Application
File System
Project Libraries
SOA-MDS
Resource Chooser
Application
XYZCustomer
XQueries
AddrToAddrBSInput.xqy
CustomerPSToCreateCustomer.xqy
CustomerPSToDeleteCustomer.xqy
CustOrderAccountToCustInfo.xqy
CustStatusMsgToCustomerPSOutput.xqy


XQuery Transformation Expression Builder
Select an XQuery resource for transformation and specify possible variable bindings.
XQuery Resource:
XYZCustomer/XQueries/CustOrderAccountToCustInfo
XQuery Variables:
Name
Binding
varOrders
$CustOrderResp.parameters
varAccount
$CustAccResp.parameters
varCustomer
$CustInfoResp.payload
Help
OK
Cancel

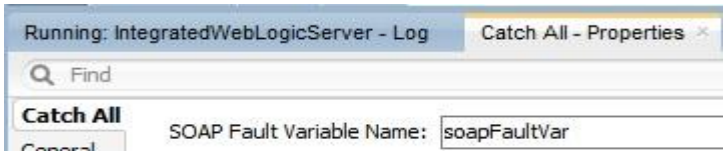

Running: IntegratedWebLogicServer - Log
Assign - Properties
Find
Assign
General
Value:*
XQuery: XYZCustomer/XQueries/CustOrderAccountToCustInfo
Variable:*
response.parameters
QoS:*
Best Effort

With this, you have finished message flow for Split Join and let us finish error handling before proceeding with testing. The WSDL used for split join has generic fault structure and you will use the same structure to return all type of faults to caller. So we will do the fault handling at complete split join level. Right click **CustMigrationSplitJoin** and select **CatchAll** as shown below.

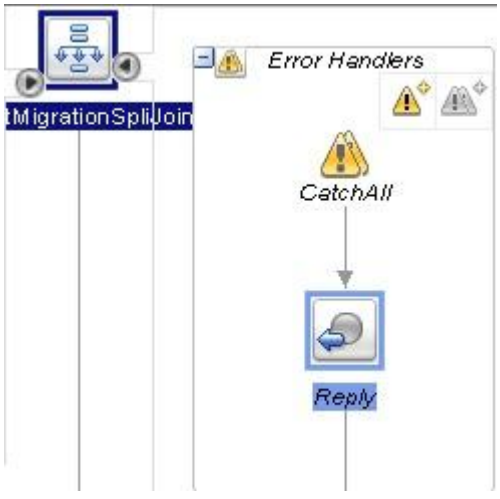




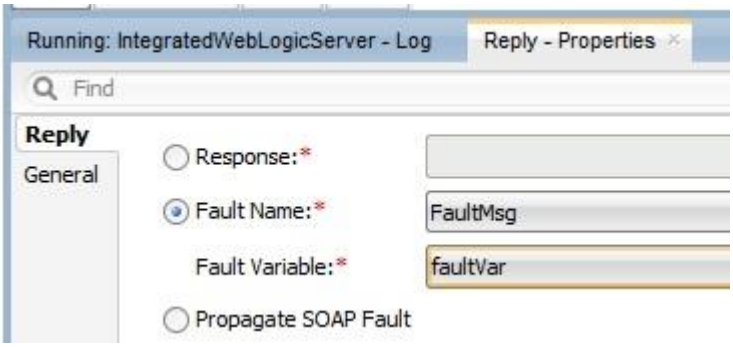
In Properties tab, give the fault variable name as shown below.



Drag **Reply** activity into **Catch All** branch from **Communication**.



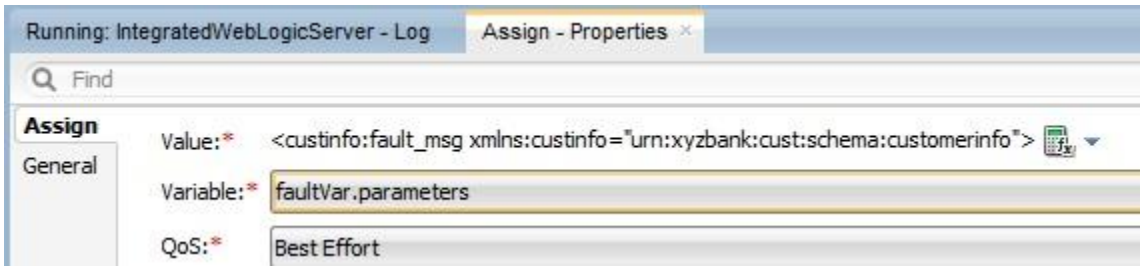
In **Properties** tab, set the properties as shown below by creating new fault variable. This enables split join to return a fault in case of any error to Pipeline.



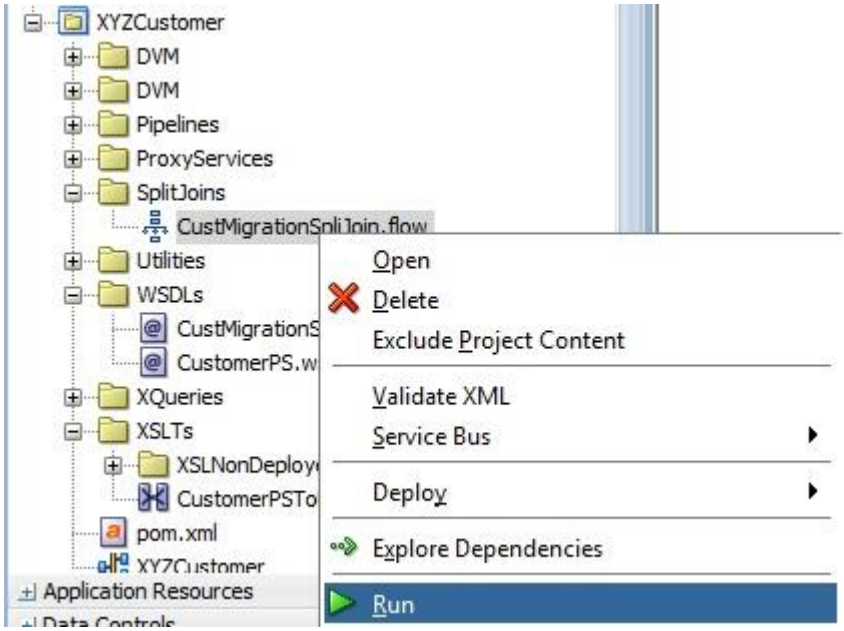
Drag **Assign** activity into **Catch All** branch before **Reply** activity from **Assign Operations**. Modify name to **AssignFault** and set properties as shown below.

Value:

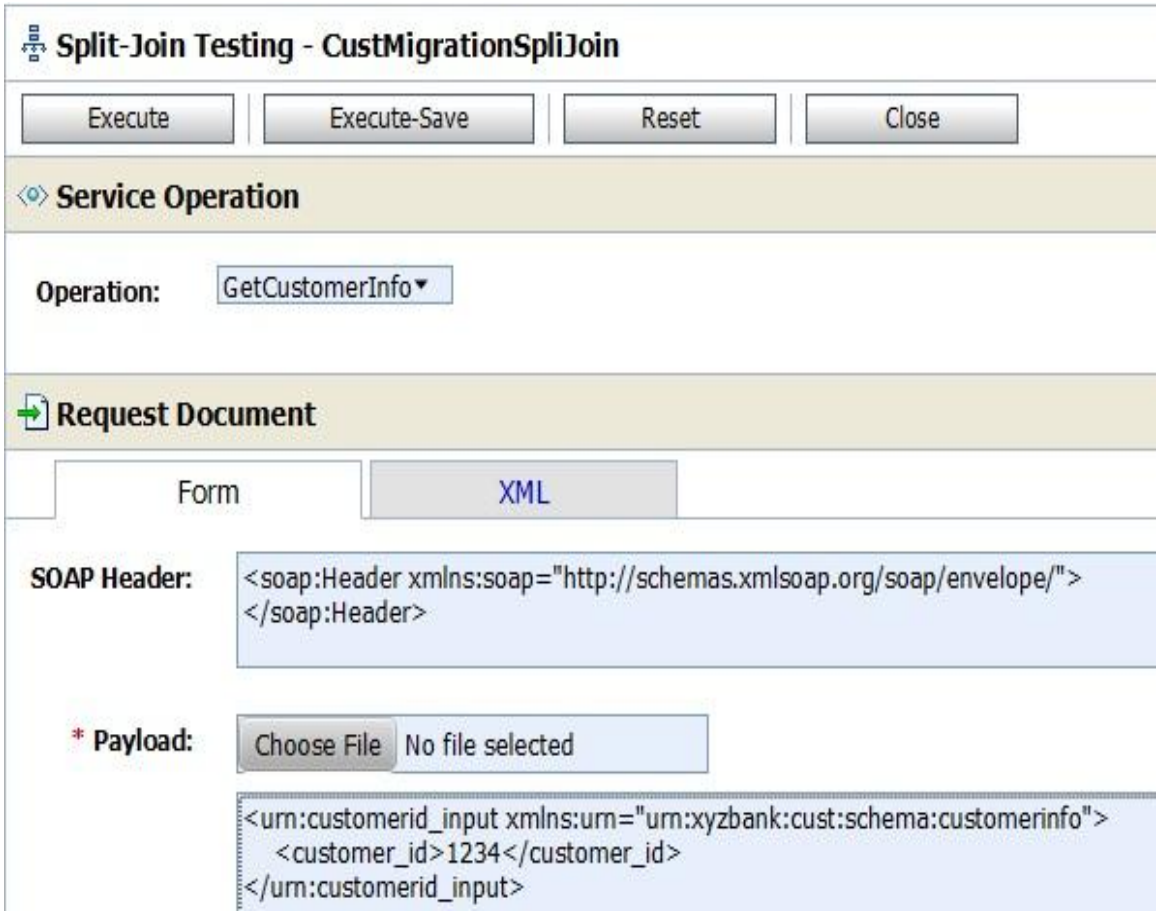
```
<custinfo:fault_msg xmlns:custinfo="urn:xyzbank:cust:schema:customerinfo">
        <custinfo:error_code>XYZ-0009</custinfo:error_code>
        <custinfo:error_message>Error in join operation.</custinfo:error_message>
</custinfo:fault_msg>
```



Test your Split join in similar to Proxy service/Pipeline and observe positive and negative cases.



Positive Case:



Response Document

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <soap-env:Body>
        <ns4:migrated_customer xmlns:ns4="urn:xyzbank:cust:schema:customerinfo">
            <customer_id>1234</customer_id>
            <full_name>FName.LName</full_name>
            <date_of_birth/>
            <email>abc@gmail.com</email>
            <phone>+911234567890</phone>
            <addr_line1>Madhapur</addr_line1>
            <addr_line2>Hyd</addr_line2>
            <addr_line3>Andhra Pradesh,India,500081</addr_line3>
            <customer_accounts>
                <account>
                    <account_id>ACT87878HKJHKJ</account_id>
                    <account_type>SAVINGS</account_type>
                    <account_name>svgonugu</account_name>
                    <description>desc</description>
                    <investor_type>Individual</investor_type>
                    <brokerage_code>FLEXI</brokerage_code>
                    <rebate_code>REB</rebate_code>
                    <trading_limit>15000</trading_limit>
                    <credit_limit>10000</credit_limit>
                    <account_status>ACTIVE</account_status>
                    <registration_details>
                        <market_id>BSE</market_id>
                        <instrument_code>CHECK</instrument_code>
                        <description>desc1</description>
                        <registration_details1>Madhapur</registration_details1>
                        <postal_code>500081</postal_code>
                        <country_code>IND</country_code>
                    </registration_details>
                </account>
            </customer_accounts>

            <customer_orders>
                <order>
                    <account_id>ACT87878HKJHKJ</account_id>
                    <instrument_code>CHECK</instrument_code>
                    <order_action>BUY</order_action>
                    <price>2345</price>
                    <currency_code>INR</currency_code>
                    <quantity>20</quantity>
                    <date_created/>
                    <brokerage_amount>70</brokerage_amount>
                    <brokerage_code>FLEXI</brokerage_code>
                    <brokerage_perc>10</brokerage_perc>
                </order>
            </customer_orders>
        </ns4:migrated_customer>
    </soap-env:Body>
</soap-env:Envelope>
```

Negative Case:

Request Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <urn:customerid_input xmlns:urn="urn:xyzbank:cust:schema:customerinfo">
            <customer_id>1234</customer_id>
        </urn:customerid_input>
    </soapenv:Body>
</soapenv:Envelope>
```

Response Document

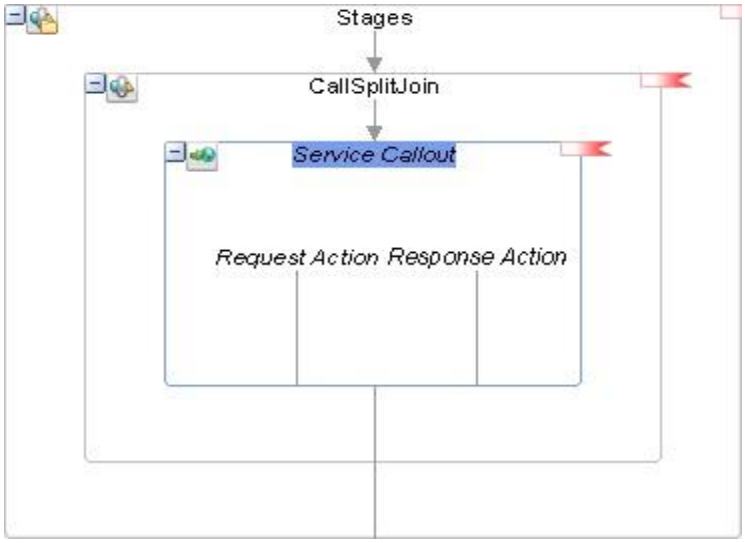
The invocation resulted in an error: [{http://xmlns.xyzbank.com/wsdl/Customer}FaultMsg] {http://xmlns.xyzbank.com/wsdl/Customer}FaultMsg <bind:GenericFault xmlns:bind="http://xmlns.xyzbank.com/wsdl/Customer">
<custinfo:fault_msg xmlns:custinfo="urn:xyzbank:cust:schema:customerinfo">
<custinfo:error_code>XYZ-0009</custinfo:error_code>
<custinfo:error_message>Error in join operation.</custinfo:error_message>
</custinfo:fault_msg>
</bind:GenericFault>.

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <soap-env:Body>
        <soap-env:Fault xmlns:bind="http://xmlns.xyzbank.com/wsdl/Customer">
            <faultcode>bind:FaultMsg</faultcode>
            <faultstring>
                {http://xmlns.xyzbank.com/wsdl/Customer}FaultMsg
            </faultstring>
            <detail>
                <custinfo:fault_msg xmlns:custinfo="urn:xyzbank:cust:schema:customerinfo">
                    <custinfo:error_code>XYZ-0009</custinfo:error_code>
                    <custinfo:error_message>Error in join operation.</custinfo:error_message>
                </custinfo:fault_msg>
            </detail>
        </soap-env:Fault>
    </soap-env:Body>
</soap-env:Envelope>
```

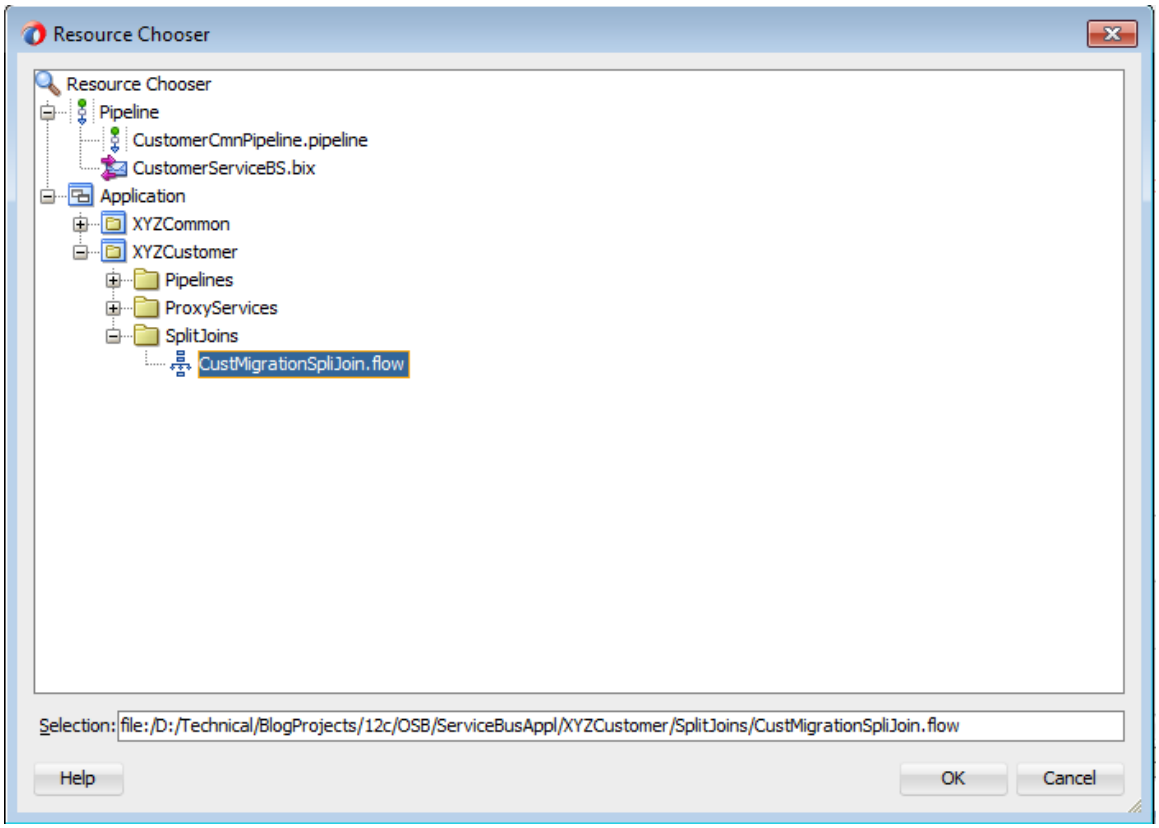
Invoking Split Join

In this section, you will invoke Split Join created in the previous section to fetch the Customer Information.

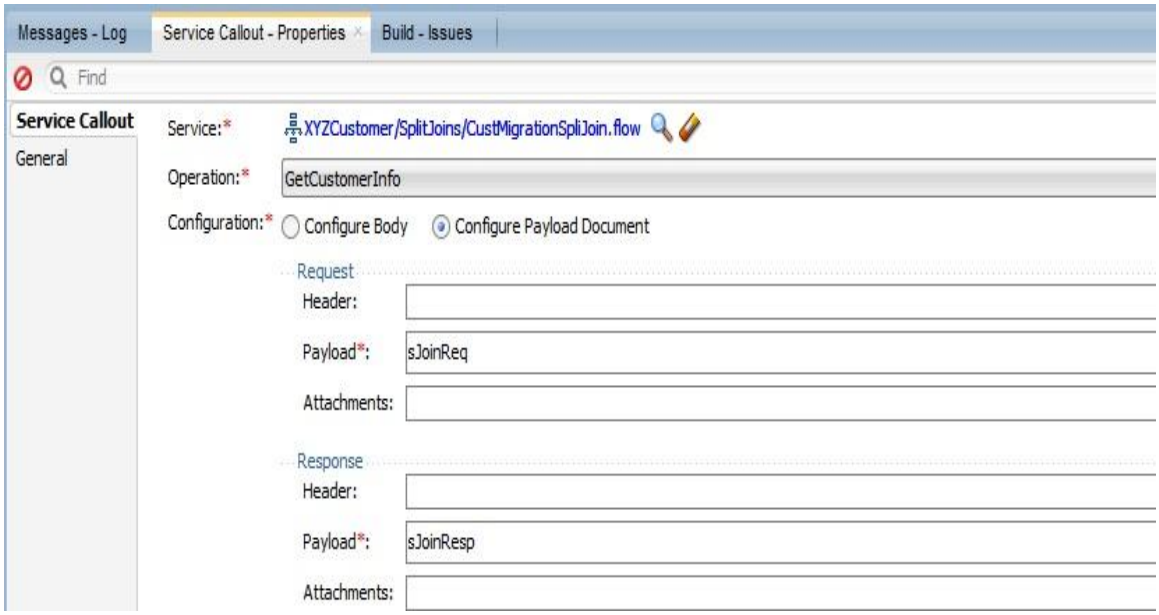
Drag **Stage** node into **Stages** placeholder from **Nodes** into **Request Pipeline** and set the name as **CallSplitJoin** and drag **Service Callout** into this stage from Communication shown below.



In **Properties** tab, browse and select **CustMigrationSplitJoin** for **Service** property.



Select **GetCustomerInfo** operation and set other properties of **Service Callout** as shown below.

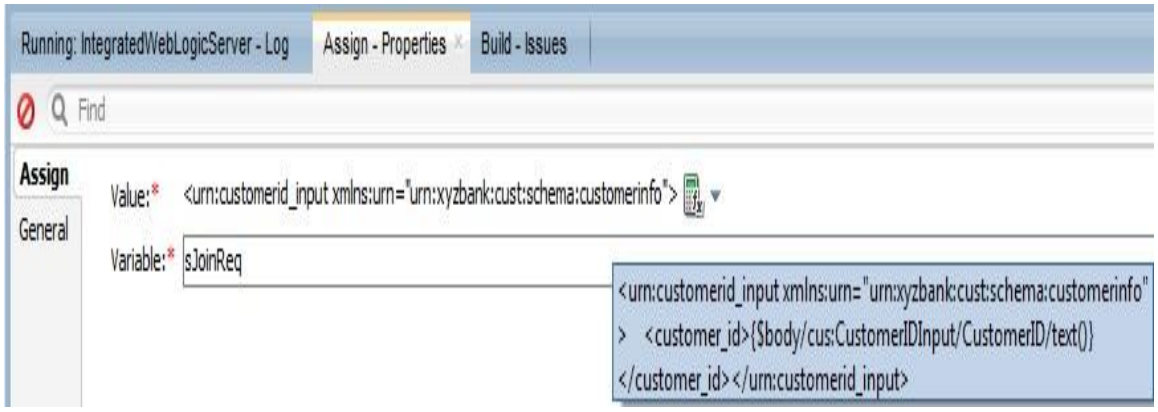


sJoinReq is variable to be populated with the payload for **GetCustomerInfo** operation and **sJoinResp** is variable that contains response returned by split join after service call. Since you are using '**Configure Payload Document**' option, you just need to populate the request variable with required payload.

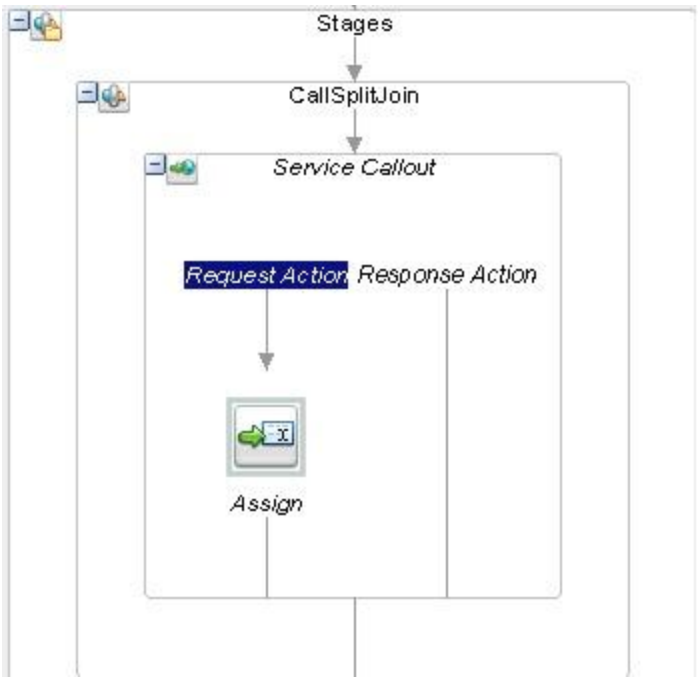
Drag **Assign** activity from **Message Processing** into **Request Action** of **Service Callout** and set properties as shown below.

Value:

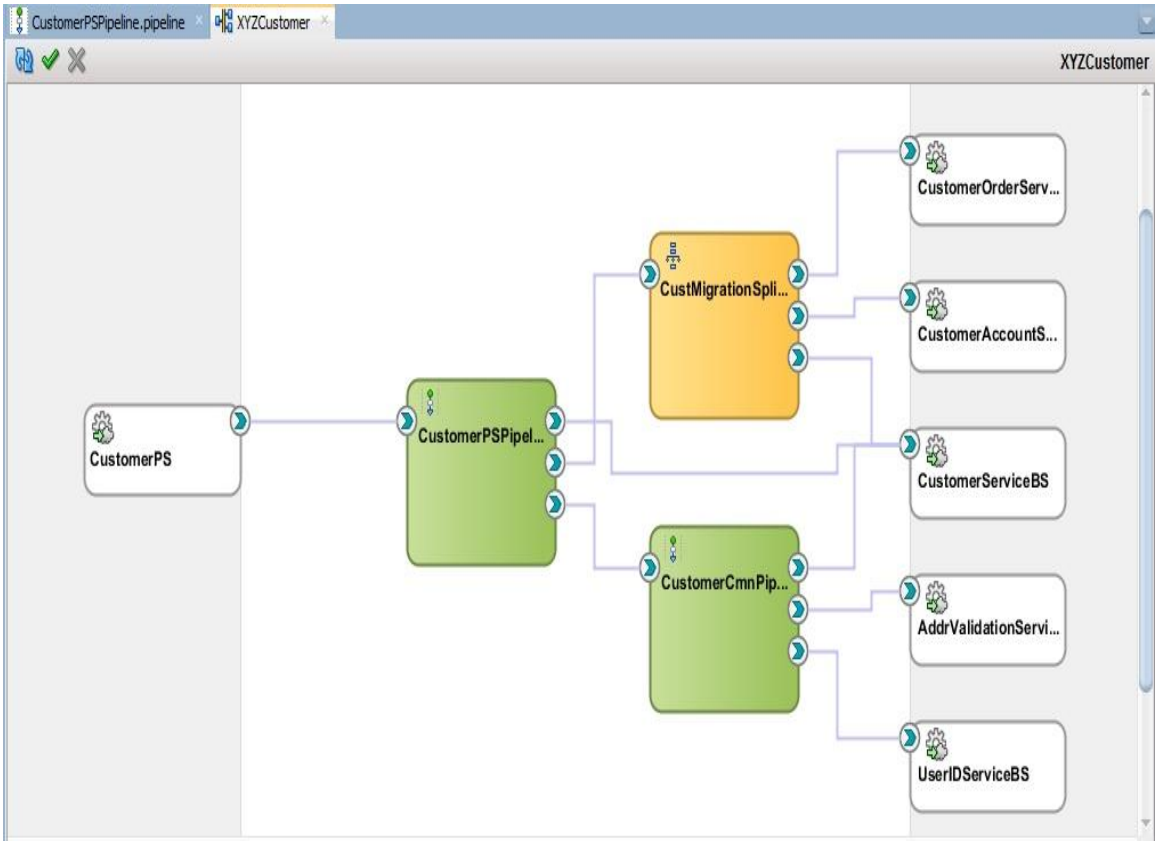
```
<urn:customerid_input xmlns:urn="urn:xyzbank:cust:schema:customerinfo">
        <customer_id>{$body/cus:CustomerIDInput/CustomerID/text()}</customer_id>
</urn:customerid_input>
```



Now your **CallSplitJoin** stage should look like below.



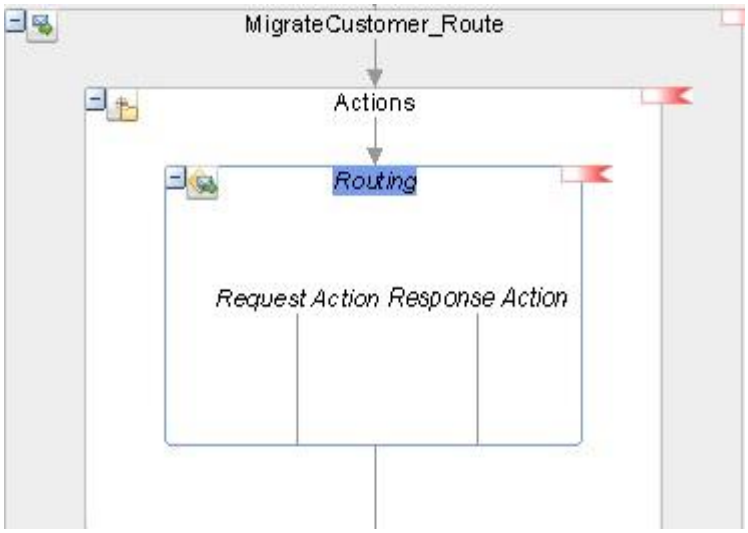
And your Service Bus Overview editor should look like below.



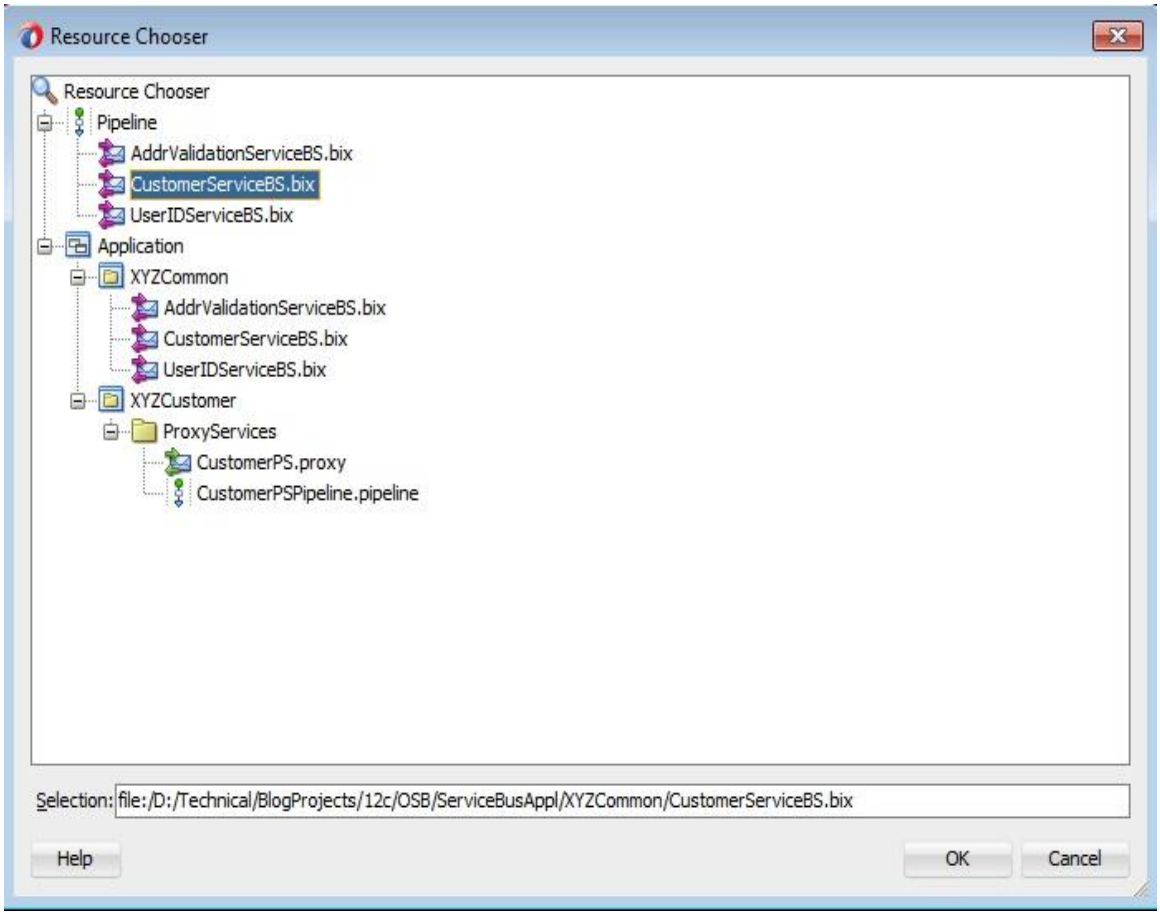
Routing

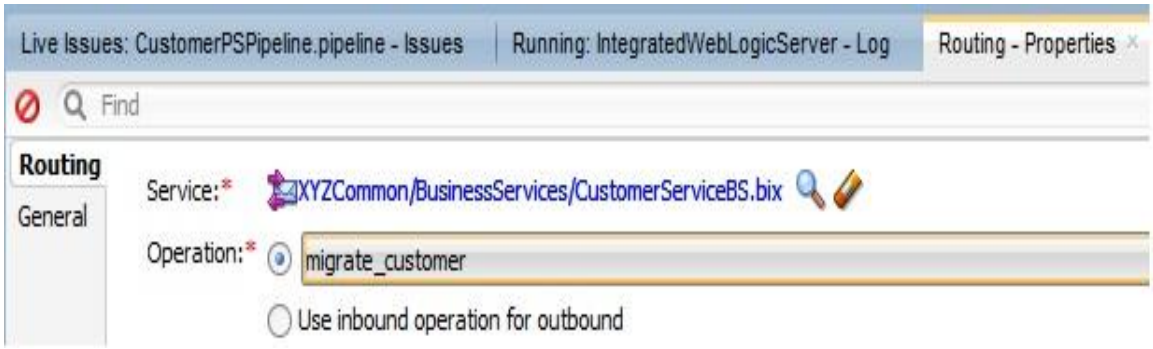
In this section, you will invoke **CustomerServiceBS** to finish message flow for **Migrate Customer**.

Drag **Routing** activity into **Actions** placeholder of **MigrateCustomer_Route**.

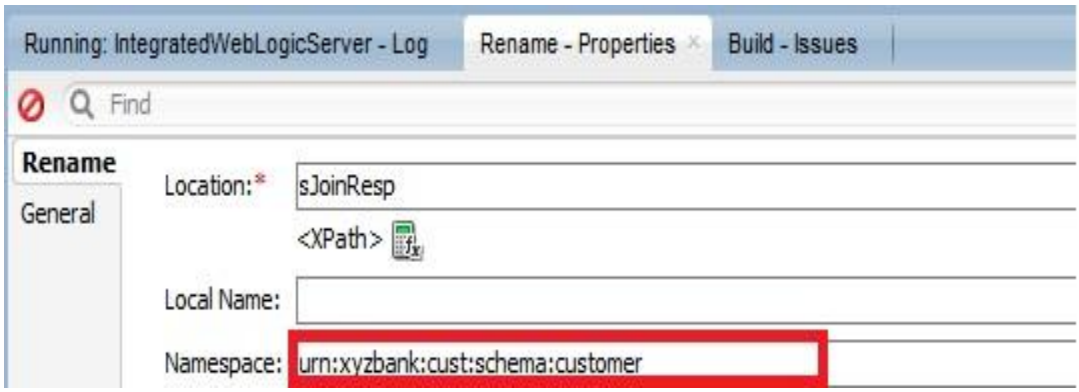


In **Properties** tab, browse and select **CustomerServiceBS** and **migrate_customer** operation.

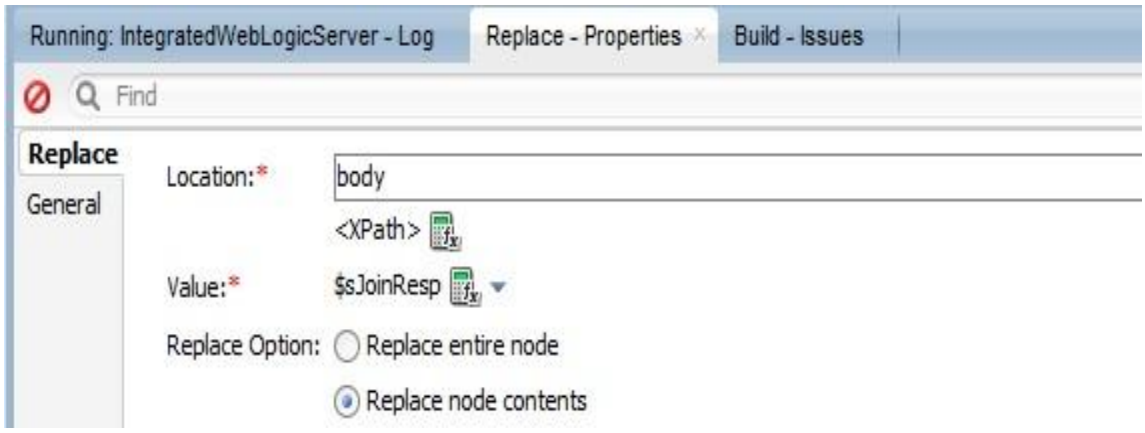




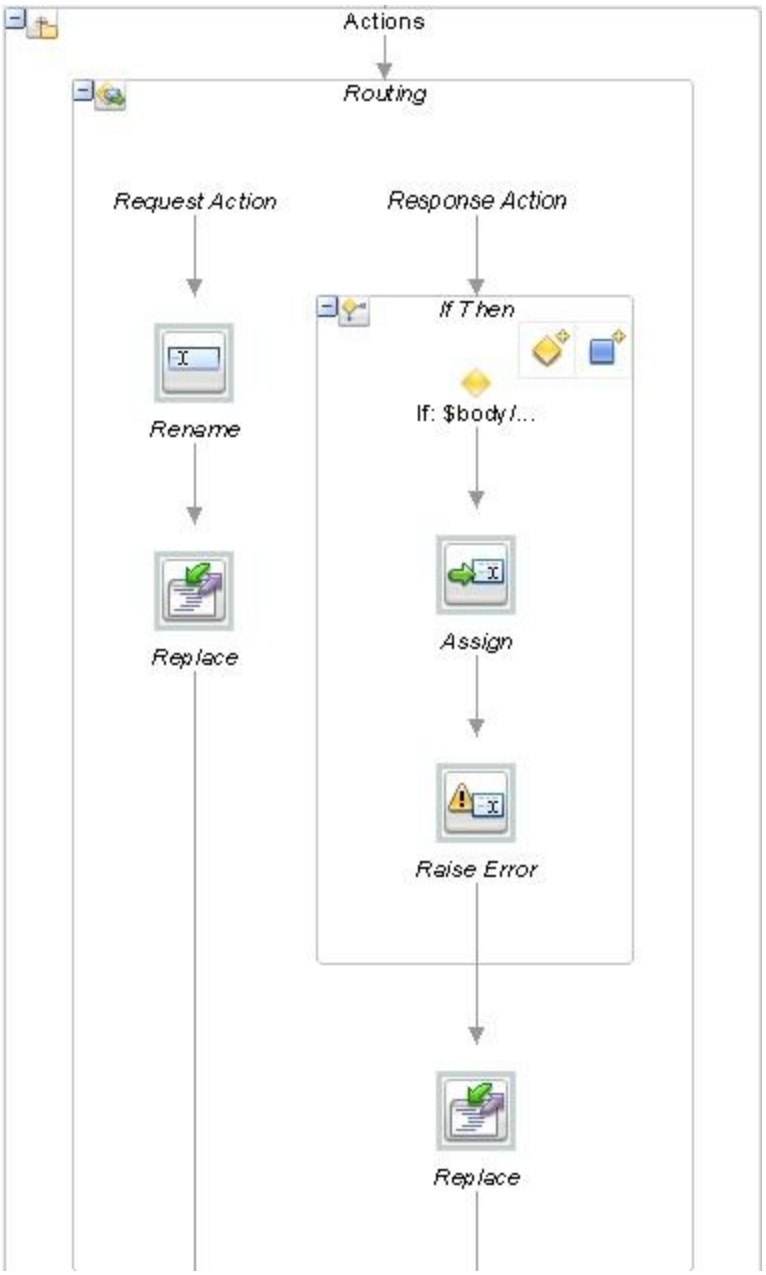
The response of Split Join is same as the payload required by **CustomerServiceBS** with different namespace so you could use **Rename** activity instead of another transformation. Drag **Rename** activity from **Message Processing** into **Request Action** of **Routing** and set properties as shown below.



Now you need to replace **$body** with this payload. So drag **Replace** activity after **Rename** from **Message Processing** and set properties as shown below.



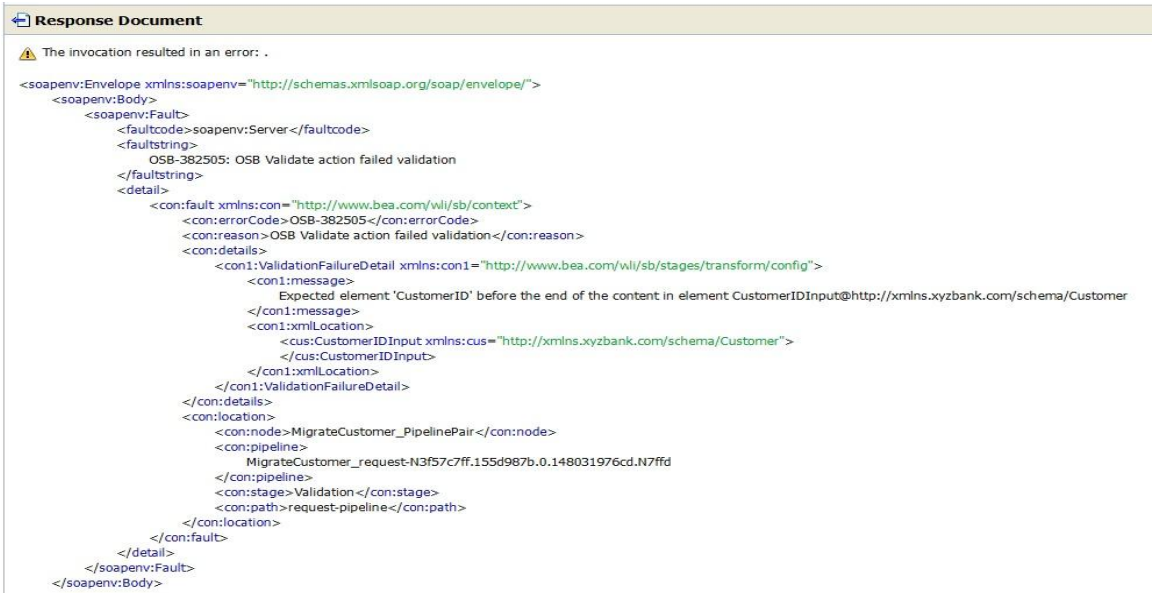
Finish **Response Action** in **Routing** similar to **CreateCustomer** as response structure of business service is same for all the operations. So you can reuse **CustStatusMsgToCustomerPSOutput** XQuery Map. Now your Routing node should look like below.


Actions
Routing
Request Action
Response Action
If Then
Rename
If: $body/...
Replace
Assign
Raise Error
Replace

Testing

Deploy both projects or run Pipeline directly as shown in **Deploying and Testing** section. You may want to frequently test your proxy service/pipeline during development. Run your pipeline with sample payloads given here and observe **Flow Trace and Variables** as shown below. You can also run Proxy Service but you will not observe any **Flow Trace**.

No CustomerID element:



No value for CustomerID:

Positive Case:

Invocation Trace

(receiving request)
CustomerOperation
MigrateCustomer_PipelinePair
Validation
CallSplitJoin
Invoked Services
Service Callout to: "CustMigrationSpliJoin"
$outbound:
$body (request):

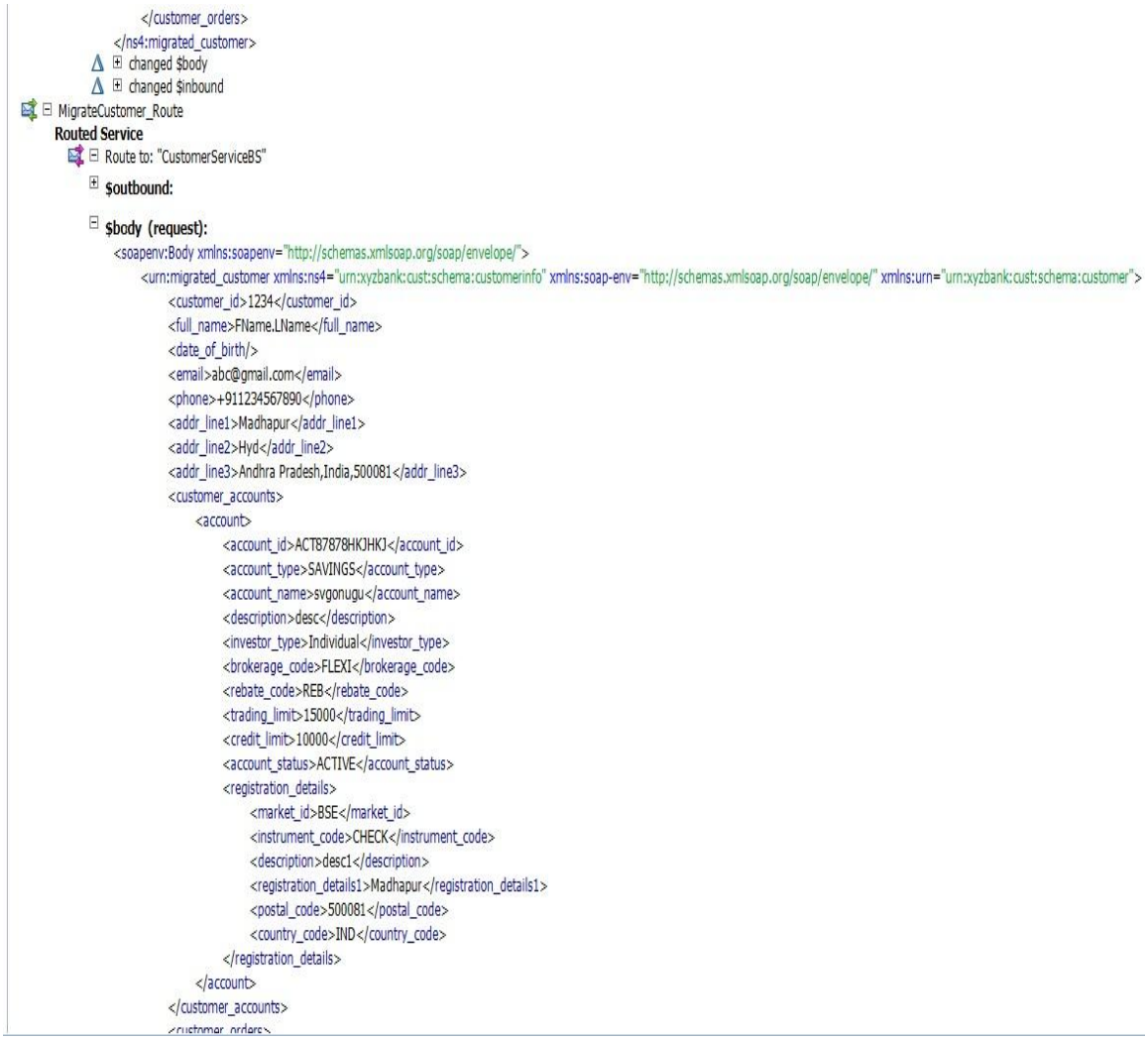
```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <urn:customerid_input xmlns:urn="urn:xyzbank:cust:schema:customerinfo">
        <customer_id>AHHHHH8787299</customer_id>
    </urn:customerid_input>
</soapenv:Body>
```

$header (request):
$body (response):
$header (response):
Message Context Changes
added $sJoinReq

```
<urn:customerid_input xmlns:urn="urn:xyzbank:cust:schema:customerinfo">
    <customer_id>AHHHHH8787299</customer_id>
</urn:customerid_input>
```

added $sJoinResp

```
<ns4:migrated_customer xmlns:ns4="urn:xyzbank:cust:schema:customerinfo" xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <customer_id>1234</customer_id>
    <full_name>FName.LName</full_name>
    <date_of_birth/>
    <email>abc@gmail.com</email>
    <phone>+911234567890</phone>
    <addr_line1>Madhapur</addr_line1>
    <addr_line2>Hyd</addr_line2>
    <addr_line3>Andhra Pradesh,India,500081</addr_line3>
    <customer_accounts>
        <account>
            <account_id>ACT87878HKJHKJ</account_id>
            <account_type>SAVINGS</account_type>
            <account_name>svgonugu</account_name>
            <description>desc</description>
            <investor_type>Individual</investor_type>
            <brokerage_code>FLEXI</brokerage_code>
            <rebate_code>REB</rebate_code>
            <trading_limit>15000</trading_limit>
            <credit_limit>10000</credit_limit>
            <account_status>ACTIVE</account_status>
            <registration_details>
                <market_id>BSE</market_id>
                <instrument_code>CHECK</instrument_code>
                <description>desc1</description>
                <registration_details1>Madhapur</registration_details1>
                <postal_code>500081</postal_code>
                <country_code>IND</country_code>
            </registration_details>
        </account>
    </customer_accounts>
    <customer_orders>
        <order>
            <account_id>ACT87878HKJHKJ</account_id>
            <instrument_code>CHECK</instrument_code>
            <order_action>BUY</order_action>
            <price>2345</price>
            <currency_code>INR</currency_code>
            <quantity>20</quantity>
            <date_created/>
            <brokerage_amount>70</brokerage_amount>
```


</customer_orders>
</ns4:migrated_customer>
changed $body
changed $inbound
MigrateCustomer_Route
Routed Service
Route to: "CustomerServiceBS"
$outbound:
$body (request):
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<urn:migrated_customer xmlns:ns4="urn:xyzbank:cust:schema:customerinfo" xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:xyzbank:cust:schema:customer">
<customer_id>1234</customer_id>
<full_name>FName.LName</full_name>
<date_of_birth/>
<email>abc@gmail.com</email>
<phone>+911234567890</phone>
<addr_line1>Madhapur</addr_line1>
<addr_line2>Hyd</addr_line2>
<addr_line3>Andhra Pradesh,India,500081</addr_line3>
<customer_accounts>
<account>
<account_id>ACT87878HKJHKJ</account_id>
<account_type>SAVINGS</account_type>
<account_name>svgonugu</account_name>
<description>desc</description>
<investor_type>Individual</investor_type>
<brokerage_code>FLEXI</brokerage_code>
<rebate_code>REB</rebate_code>
<trading_limit>15000</trading_limit>
<credit_limit>10000</credit_limit>
<account_status>ACTIVE</account_status>
<registration_details>
<market_id>BSE</market_id>
<instrument_code>CHECK</instrument_code>
<description>desc1</description>
<registration_details1>Madhapur</registration_details1>
<postal_code>500081</postal_code>
<country_code>IND</country_code>
</registration_details>
</account>
</customer_accounts>
<customer_orders>


$header (request):
$attachments (request):
Message Context Changes
added $outbound
changed $body
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:xyzbank:cust:schema:customer">
<ns2:StatusMsg xmlns:ns2="http://xmlns.xyzbank.com/schema/Customer">
<CustomerID>1234</CustomerID>
<status>S</status>
</ns2:StatusMsg>
</soapenv:Body>
changed $sJoinResp
changed $attachments
changed $inbound
changed $header
MigrateCustomer_PipelinePair
CustomerOperation

Negative Case - Status E in business service response:

Invocation Trace

(receiving request)
CustomerOperation
MigrateCustomer_PipelinePair
Validation
CallSplitJoin
MigrateCustomer_Route
Routed Service
Route to: "CustomerServiceBS"
Message Context Changes
added $faultVar

```
<urn:status_msg xmlns:urn="urn:xyzbank:cust:schema:customer" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <customer_id>AHHHHH8787299</customer_id>
    <status>E</status>
    <error_message>Unexpected error in migrating user.</error_message>
</urn:status_msg>
```

added $outbound
changed $body
changed $sJoinResp
changed $attachments
changed $inbound
changed $header
Service Error Handler

$fault:

```
<con:fault xmlns:con="http://www.bea.com/wli/sb/context">
    <con:errorCode>XYZ-0006</con:errorCode>
    <con:reason>Error in Customer Migration.</con:reason>
    <con:location>
        <con:node>MigrateCustomer_Route</con:node>
        <con:path>response-pipeline</con:path>
    </con:location>
</con:fault>
```

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>XYZ-0006: Error in Customer Migration.</faultstring>
            <detail>
                <con:fault xmlns:con="http://www.bea.com/wli/sb/context">
                    <con:errorCode>XYZ-0006</con:errorCode>
                    <con:reason>Error in Customer Migration.</con:reason>
                    <con:location>
                        <con:node>MigrateCustomer_Route</con:node>
                        <con:path>response-pipeline</con:path>
                    </con:location>
                </con:fault>
            </detail>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Negative Case – Error in Split Join, Service Down:

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>
                OSB-382500: OSB Service Callout action received SOAP Fault response
            </faultstring>
            <detail>
                <con:fault xmlns:con="http://www.bea.com/wli/sb/context">
                    <con:errorCode>OSB-382500</con:errorCode>
                    <con:reason>
                        OSB Service Callout action received SOAP Fault response
                    </con:reason>
                    <con:details>
                        <con1:ReceivedFaultDetail xmlns:con1="http://www.bea.com/wli/sb/stages/transform/config">
                            <con1:faultcode xmlns:cus="http://xmlns.xyzbank.com/wsdl/Customer">cus:FaultMsg</con1:faultcode>
                            <con1:faultstring>
                                {http://xmlns.xyzbank.com/wsdl/Customer}FaultMsg
                            </con1:faultstring>
                            <con1:detail>
                                <custinfo:fault_msg xmlns:custinfo="urn:xyzbank:cust:schema:customerinfo" xmlns:bind="http://xmlns.xyzbank.com/wsdl/Customer" xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
                                    <custinfo:error_code>XYZ-0009</custinfo:error_code>
                                    <custinfo:error_message>Error in join operation.</custinfo:error_message>
                                </custinfo:fault_msg>
                            </con1:detail>
                        </con1:ReceivedFaultDetail>
                    </con:details>
                    <con:location>
                        <con:node>MigrateCustomer_PipelinePair</con:node>
                        <con:pipeline>
                            MigrateCustomer_request-N3f57c7ff.155d987b.0.148031976cd.N7ffd
                        </con:pipeline>
                        <con:stage>CallSplitJoin</con:stage>
                        <con:path>request-pipeline</con:path>
                    </con:location>
                </con:fault>
            </detail>
```

(receiving request)
CustomerOperation
MigrateCustomer_PipelinePair
Validation
CallSplitJoin
Invoked Services
Service Callout to: "CustMigrationSpliJoin"
$outbound:
$body (request):
$header (request):
Message Context Changes
added $sJoinReq
changed $body
changed $inbound
Service Error Handler

$fault:

```
<con:fault xmlns:con="http://www.bea.com/wli/sb/context">
    <con:errorCode>OSB-382500</con:errorCode>
    <con:reason>
        OSB Service Callout action received SOAP Fault response
    </con:reason>
    <con:details>
        <con1:ReceivedFaultDetail xmlns:con1="http://www.bea.com/wli/sb/stages/transform/config">
            <con1:faultcode xmlns:cus="http://xmlns.xyzbank.com/wsdl/Customer">cus:FaultMsg</con1:faultcode>
            <con1:faultstring>
                {http://xmlns.xyzbank.com/wsdl/Customer}FaultMsg
            </con1:faultstring>
            <con1:detail>
                <custinfo:fault_msg xmlns:custinfo="urn:xyzbank:cust:schema:customerinfo" xmlns:bind="http://xmlns.xyzbank.com/wsdl/Customer" xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
                    <custinfo:error_code>XYZ-0009</custinfo:error_code>
                    <custinfo:error_message>Error in join operation.</custinfo:error_message>
                </custinfo:fault_msg>
            </con1:detail>
        </con1:ReceivedFaultDetail>
    </con:details>
    <con:location>
        <con:node>MigrateCustomer_PipelinePair</con:node>
        <con:pipeline>
            MigrateCustomer_request-N3f57c7ff.155d987b.0.148031976cd.N7ffd
        </con:pipeline>
        <con:stage>CallSplitJoin</con:stage>
        <con:path>request-pipeline</con:path>
    </con:location>
```

Error Handling

In this section, you will learn how to do error handling in Pipeline. As shown in previous sections, the pipeline is not returning any fault message with fault structure defined in WSDL for both application/business and system errors.

As you know, service provider can send error to consumer in following ways:

- As a normal response, by populating fields like "**ErrorNumber**" and "**ErrorMesssage**" (assuming that these fields are defined in response message of operation in WSDL).
- As a SOAP fault

Typically when Service Bus is mediating, you might have to transform error or fault response to the fault structure defined in WSDL. So you need to understand the message context variables that can be used for this transformation.

As per WS-I BP, the service provider should send the HTTP response code as 200 when the error is sent as normal response and 500 should be sent when the error is sent as SOAP fault.

When HTTP response code 200 is received, Service Bus treats it as a normal response and proceeds further with message flow. When response code 500 is received, Service Bus runtime control goes to **Service Error Handler** if it exists or any other low level **Error Handlers** depending on where you received. That means Service Bus treats even Fault response as normal response when HTTP response code is 200. In message flow, **Error Handler** can be defined for **Stage** node, **Pipeline Pair** node (both Request Pipeline and Response Pipeline individually), **Routing** node and for entire **Pipeline** (called Service Error Handler).

Service Bus populates different message context variables with error/fault messages and is accessible in **Error Handler** depending on whether you used **Routing** or **Service Callout**. When **Routing** is used, the variable **$body** will have Fault response and when **Service Callout** is used, variable **$fault** will have Fault response. The following table summarizes this discussion:

Activity	Scenario	Context Variable
Routing	Raise Error activity	$fault
Routing	Fault Response from business service	$body
Routing	System fault while calling business service	$fault
Service Callout	Raise Error activity	$fault
Service Callout	Fault Response from business service	$fault
Service Callout	System fault while calling business service	$fault

With this background, let us go back to Pipeline to add required **Error Handlers**. In this case, the business service WSDL does not define any Fault structure and all errors will be sent as normal response with status **E**. But you should consider all possible cases for Error Handling. So there is a possibility of 3 variables having error/fault information and they are **$body**, **$fault** and **$faultVar** (populated in your message flow on receiving status E in **Routing**).

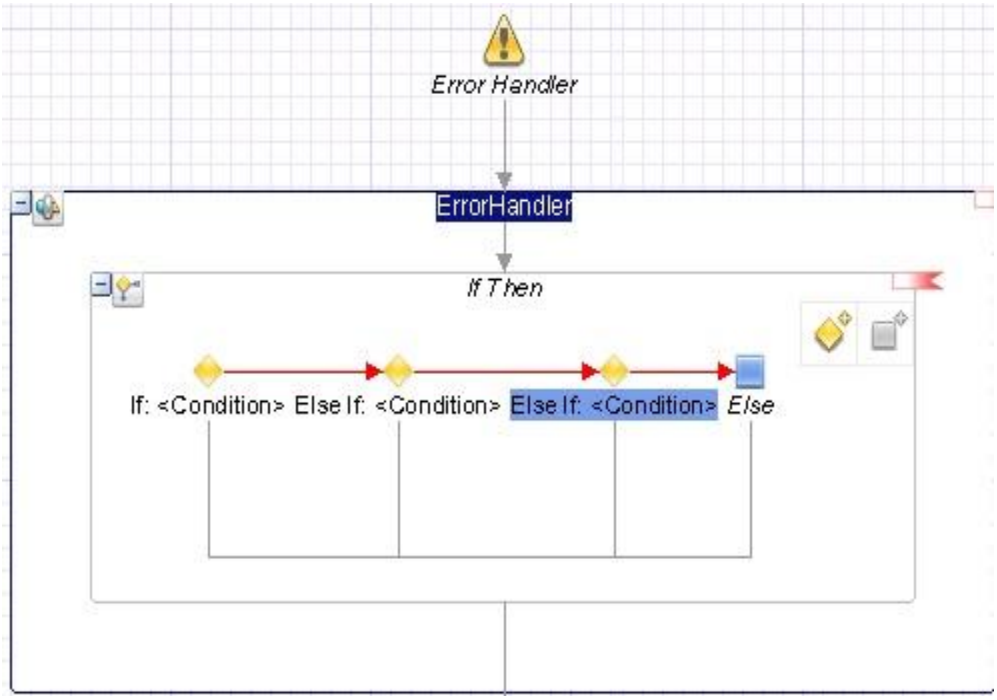
Your Pipeline uses both **Service Callout** and **Routing** in message flow so you can do the fault handling in following manner in pipeline template **CustomerPipelineTemplate** so that concrete pipelines would inherit the message flow.

- Add **Service Error Handler**. Note that you can add error handler for Routing node, Stage, Request Pipeline and Response Pipeline of Pipeline Pair nodes.
- Add conditional branches in error handler to verify **$body** is populated with SOAP Fault, or **faultVar** or **$fault** is populated. Based on this, you have to extract **Error Code** and **Error Message** from **$body, $faultVar** and **$fault** variables.
- Populate **$body** variable with fault structure as defined in WSDL.

CustomerPipelineTemplate

In this section, you will finish error handling in pipeline template.

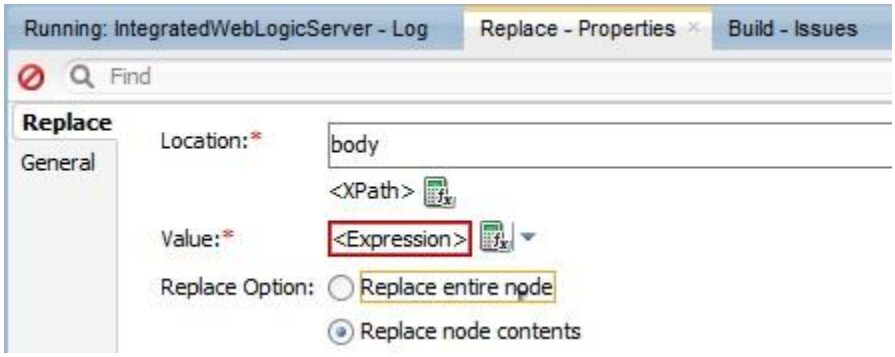
Drag **If-Then** activity into **ErrorHandler** stage from **Flow Control** and add 2 conditional branches (Else If branch) as shown below.



In **Properties** tab, set **Condition** property for all branches as shown below in expression builder. Remember adding the namespace **http://www.bea.com/wli/sb/stages/transform/config** with **con1** as alias in **Namespaces**.

Branch	Condition	Purpose
If	not(fn:empty($body/soap-env:Fault))	To handle fault received in Routing.
Else If	not(fn:empty($faultVar))	To handle error response received in Routing.
Else If	not(fn:empty($fault/ctx:details/con1:ReceivedFaultDetail))	To handle fault received in Service Callout or Raise Error cases.
Else	-NA-	All other possible cases.

Drag **Replace** activity into each of these branches from **Message Processing** and set properties as shown below.



Set expression in expression builder with the following SOAP Fault structure for **Replace** activity in **If** branch. Here you are extracting the error code and error message from received SOAP fault received in **Routing**, this way you are forwarding actual error from service provider to consumer of your proxy service.

```
<soap-env:Fault>
 <faultcode>env:Server</faultcode>
 <faultstring>{$body/soap-env:Fault/faultstring/text()}</faultstring>
 <detail>
    <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
       <ErrorCode>{$body/soap-env:Fault/faultcode/text()}</ErrorCode>
       <ErrorMsg>{$body/soap-env:Fault/faultstring/text()}</ErrorMsg>
    </cust:ErrorStatusMsg>
 </detail>
</soap-env:Fault>
```

Set expression in expression builder with the following SOAP Fault structure for **Replace** activity in **Else If** branch. Here you are extracting the error code and error message from error response received in **Routing**, this way you are forwarding actual error from service provider to consumer of your proxy service. Observe the usage of **$fault** variable to get error code given in **RaiseError** activity.

```
<soap-env:Fault>
 <faultcode>env:Server</faultcode>
 <faultstring>{$faultVar/error_message/text()}</faultstring>
 <detail>
    <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
       <ErrorCode>{$fault/ctx:errorCode/text()}</ErrorCode>
       <ErrorMsg>{$faultVar/error_message/text()}</ErrorMsg>
    </cust:ErrorStatusMsg>
 </detail>
</soap-env:Fault>
```

Set expression in expression builder with the following SOAP Fault structure for **Replace** activity in second **Else If** branch. Here you are extracting the error code and error message from SOAP Fault received in **Service Callout,** this way you are forwarding actual error from service provider to the consumer of proxy service.

```
<soap-env:Fault>
```

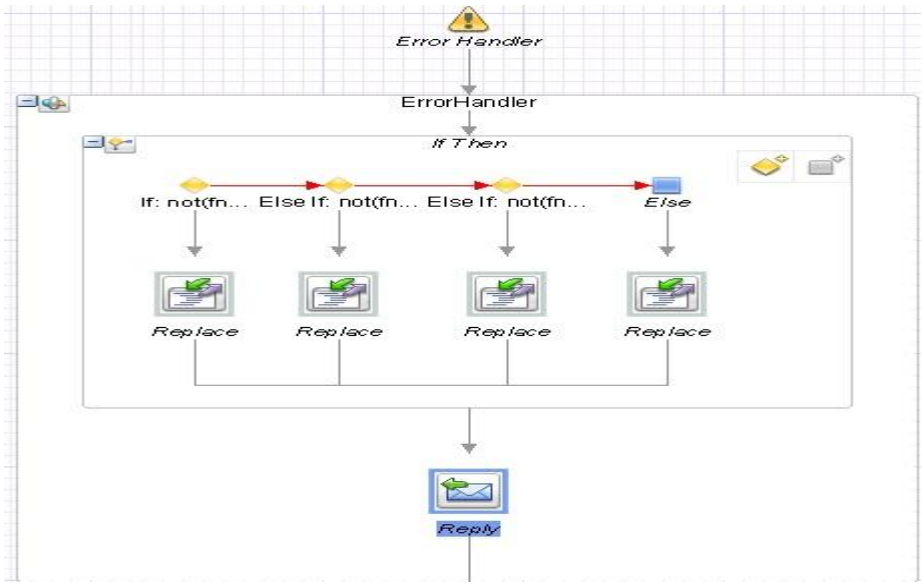
```
<faultcode>env:Server</faultcode>
<faultstring>{$fault/ctx:reason/text()}</faultstring>
<detail>
 <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
  <ErrorCode>{$fault/ctx:details/con1:ReceivedFaultDetail/con1:faultcode/text()}</ErrorCode>
  <ErrorMsg>{$fault/ctx:details/con1:ReceivedFaultDetail/con1:faultstring/text()}</ErrorMsg>
 </cust:ErrorStatusMsg>
</detail>
</soap-env:Fault>
```

Set expression in expression builder with the following SOAP fault structure for **Replace** activity in **Else** branch. Here you are extracting error code and error message from **$fault** to take care of other error scenarios.

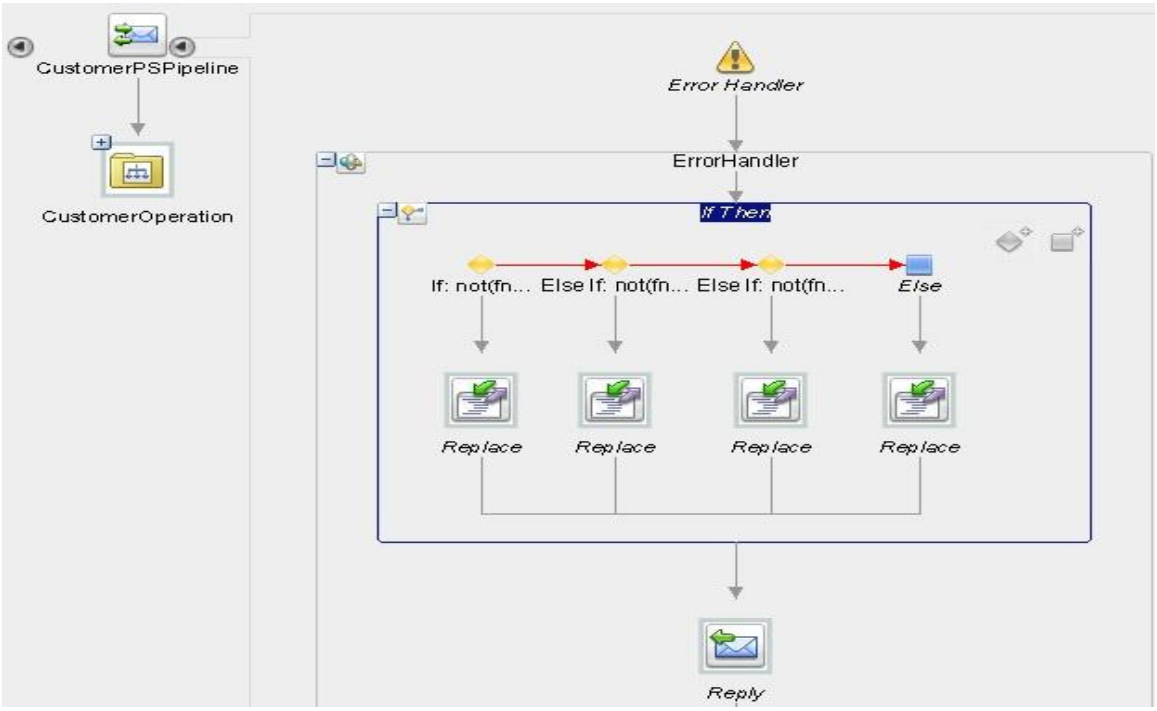
```
<soap-env:Fault>
 <faultcode>env:Server</faultcode>
 <faultstring>{$fault/ctx:reason/text()}</faultstring>
 <detail>
  <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
   <ErrorCode>{$fault/ctx:errorCode/text()}</ErrorCode>
   <ErrorMsg>{$fault/ctx:reason/text()}</ErrorMsg>
  </cust:ErrorStatusMsg>
 </detail>
</soap-env:Fault>
```

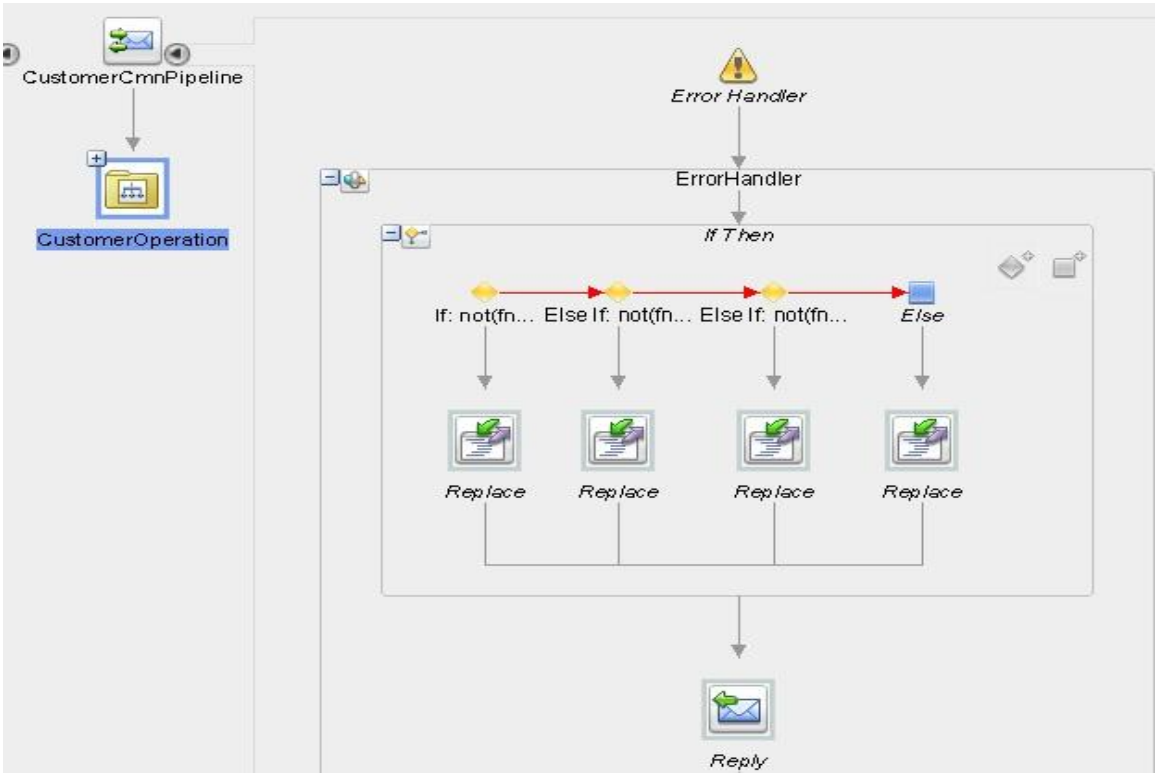
Now drag **Reply** activity into **ErrorHandler** stage from **Flow Control** after **If-then**. In **Properties** tab, select property as **With Failure.** This would send HTTP response code as 500 along with SOAP fault. Selecting **With Success** will send HTTP response code as 200.

Alternatively, you can come up with XQuery map accepting these parameters to take care of all these scenarios and return SOAP fault. Now your **ErrorHandler** stage should look like below.



Observe that **CustomerPSPipeline** and **CustomerCmnPipleline** had inherited ErrorHandler stage as shown below.

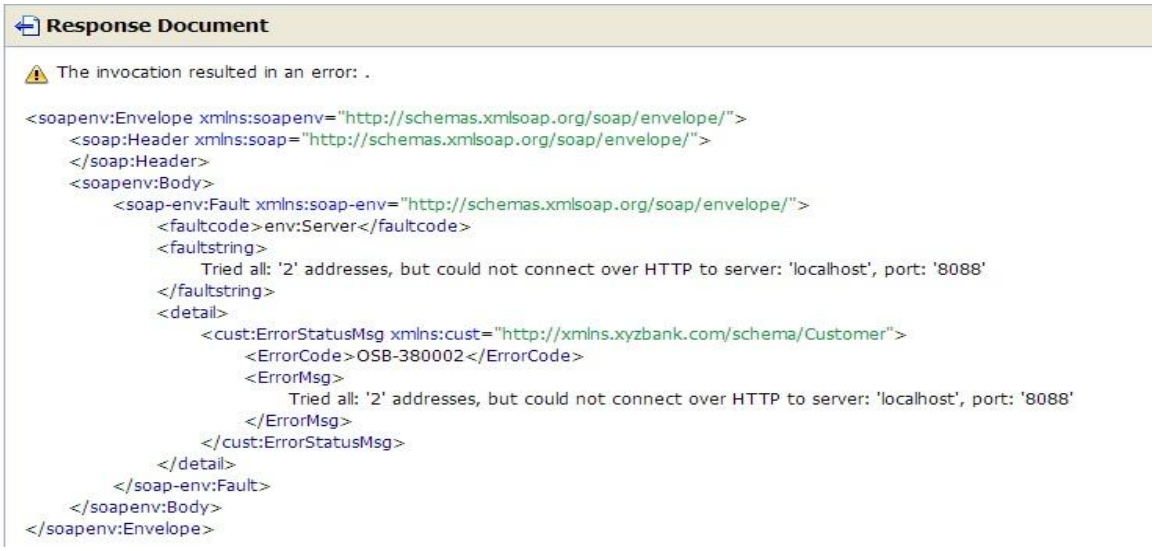




Testing

Deploy both projects or run **Pipeline** directly as shown in **Deploying and Testing** section. You may want to frequently test your proxy service/pipeline during development. Run your pipeline with sample payloads given in previous sections and observe **Flow Trace and Variables** as shown below. Observe the error response from your pipeline/proxy service and always contains **SOAP Fault** containing error message. Following screenshots show response of **CustomerCmnPipeline** for different error scenarios.

Routing – System Error (Update Customer):

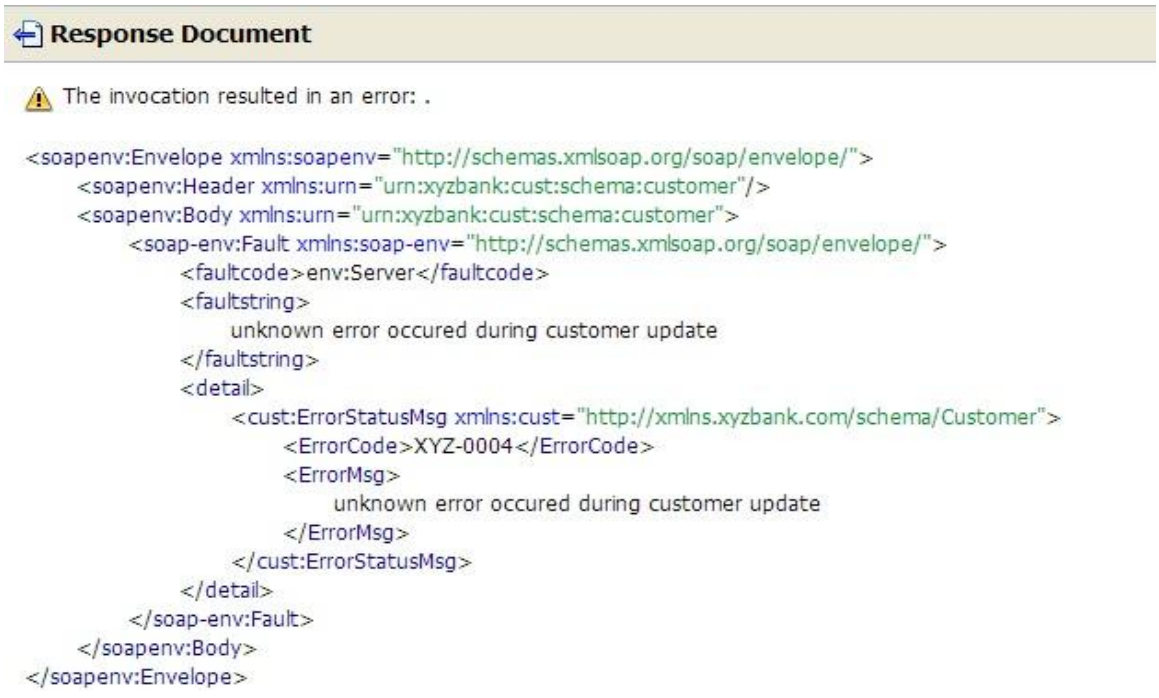


Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <soap-env:Fault xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
            <faultcode>env:Server</faultcode>
            <faultstring>
                Tried all: '2' addresses, but could not connect over HTTP to server: 'localhost', port: '8088'
            </faultstring>
            <detail>
                <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
                    <ErrorCode>OSB-380002</ErrorCode>
                    <ErrorMsg>
                        Tried all: '2' addresses, but could not connect over HTTP to server: 'localhost', port: '8088'
                    </ErrorMsg>
                </cust:ErrorStatusMsg>
            </detail>
        </soap-env:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Routing – Error Response (Update Customer):



Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header xmlns:urn="urn:xyzbank:cust:schema:customer"/>
    <soapenv:Body xmlns:urn="urn:xyzbank:cust:schema:customer">
        <soap-env:Fault xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
            <faultcode>env:Server</faultcode>
            <faultstring>
                unknown error occured during customer update
            </faultstring>
            <detail>
                <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
                    <ErrorCode>XYZ-0004</ErrorCode>
                    <ErrorMsg>
                        unknown error occured during customer update
                    </ErrorMsg>
                </cust:ErrorStatusMsg>
            </detail>
        </soap-env:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Routing – Fault Response (Create Customer):

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header xmlns:urn="urn:xyzbank:cust:schema:customer"/>
    <soapenv:Body xmlns:urn="urn:xyzbank:cust:schema:customer">
        <soap-env:Fault xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
            <faultcode>env:Server</faultcode>
            <faultstring>Error in Create Customer</faultstring>
            <detail>
                <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
                    <ErrorCode>soapenv:Server</ErrorCode>
                    <ErrorMsg>Error in Create Customer</ErrorMsg>
                </cust:ErrorStatusMsg>
            </detail>
        </soap-env:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Invalid CustomerType Validation Error (Create Customer):

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <soap-env:Fault xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
            <faultcode>env:Server</faultcode>
            <faultstring>OSB Validate action failed validation</faultstring>
            <detail>
                <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
                    <ErrorCode>OSB-382505</ErrorCode>
                    <ErrorMsg>OSB Validate action failed validation</ErrorMsg>
                </cust:ErrorStatusMsg>
            </detail>
        </soap-env:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Email Validation Error (Update Customer):

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <soap-env:Fault xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
            <faultcode>env:Server</faultcode>
            <faultstring>Email is not valid.</faultstring>
            <detail>
                <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
                    <ErrorCode>XYZ-0001</ErrorCode>
                    <ErrorMsg>Email is not valid.</ErrorMsg>
                </cust:ErrorStatusMsg>
            </detail>
        </soap-env:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Service Callout – System Error (CreateCustomer):

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <soap-env:Fault xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
            <faultcode>env:Server</faultcode>
            <faultstring>
                Tried all: '2' addresses, but could not connect over HTTP to server: 'localhost', port: '8088'
            </faultstring>
            <detail>
                <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
                    <ErrorCode>OSB-380002</ErrorCode>
                    <ErrorMsg>
                        Tried all: '2' addresses, but could not connect over HTTP to server: 'localhost', port: '8088'
                    </ErrorMsg>
                </cust:ErrorStatusMsg>
            </detail>
        </soap-env:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Service Callout – Error Response (CreateCustomer):

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <soap-env:Fault xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
            <faultcode>env:Server</faultcode>
            <faultstring>Address is invalid.</faultstring>
            <detail>
                <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
                    <ErrorCode>XYZ-0002</ErrorCode>
                    <ErrorMsg>Address is invalid.</ErrorMsg>
                </cust:ErrorStatusMsg>
            </detail>
        </soap-env:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Service Callout – Fault Response (CreateCustomer):

Response Document

The invocation resulted in an error: .

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>env:Server</faultcode>
            <faultstring>
                OSB Service Callout action received SOAP Fault response
            </faultstring>
            <detail>
                <cust:ErrorStatusMsg xmlns:cust="http://xmlns.xyzbank.com/schema/Customer">
                    <ErrorCode>XYZLEGACY-789</ErrorCode>
                    <ErrorMsg>Error in performing operation</ErrorMsg>
                </cust:ErrorStatusMsg>
            </detail>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

CustomerPSPipeline

Similarly, run this pipeline with sample payloads given in previous sections and observe that you always receive fault message as the response.

Securing Proxy Service

Security is one of the main aspects while developing any service and is no different from our regular web applications. Also these services are reusable and can be invoked by either internal or external customers, so you should consider securing your Service Bus Proxy Services so that only valid users will be able to invoke.

Service Bus is completely integrated with **Oracle Webservices Manager** (OWSM) which provides several out of the box security policies. You can use these any of these OWSM policies to secure your proxy services based on your service security requirements. In this tutorial, you will use the policy **oracle/wss_username_token_service_policy** to secure your Proxy Service.

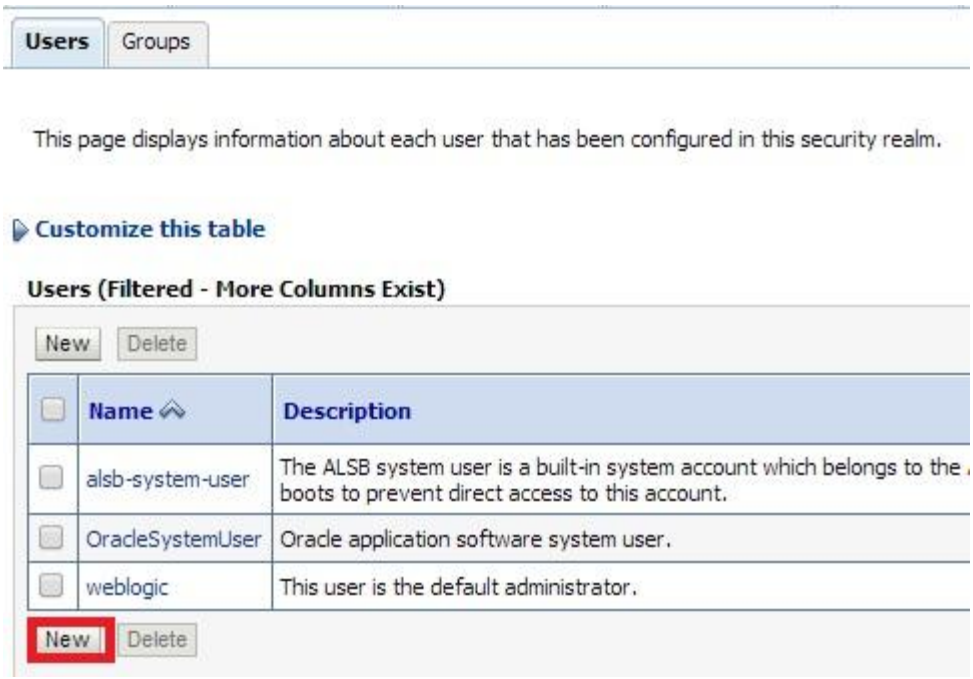
First step is creating users. Here you will see how to create users using **sbconsole**. But this is not the recommended way and typically an Organization will have a list of valid users in some LDAP directory and you have to configure one of the Authenticators in Weblogic to access that LDAP for authentication purpose. This configuration will not be discussed here in this tutorial.

Creating User

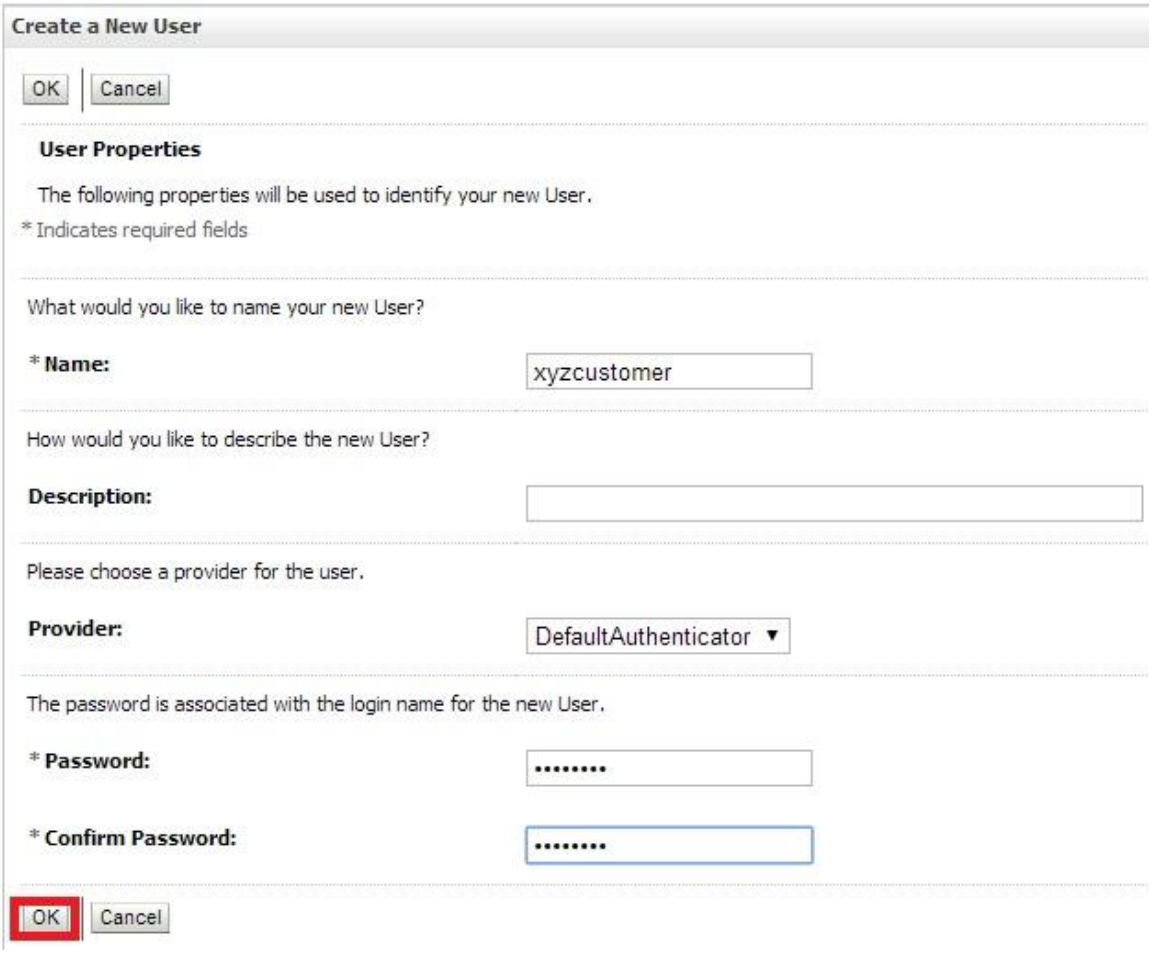
Login to **Admin Console** and click **Security Realms** in Domain Structure.

Click **myrealm** as shown below.

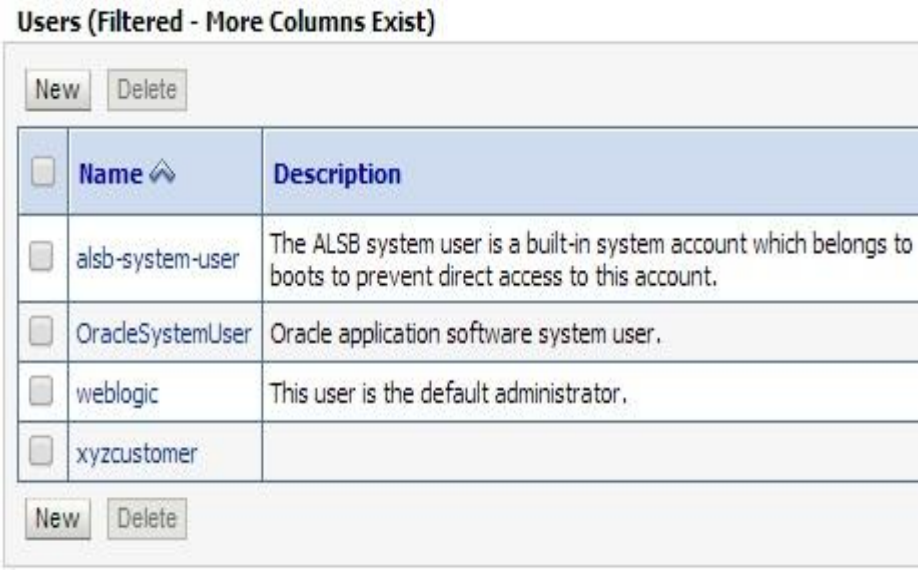
Navigate to Users by clicking on **Users and Groups** tab.



Click **New** and enter credentials as shown below.

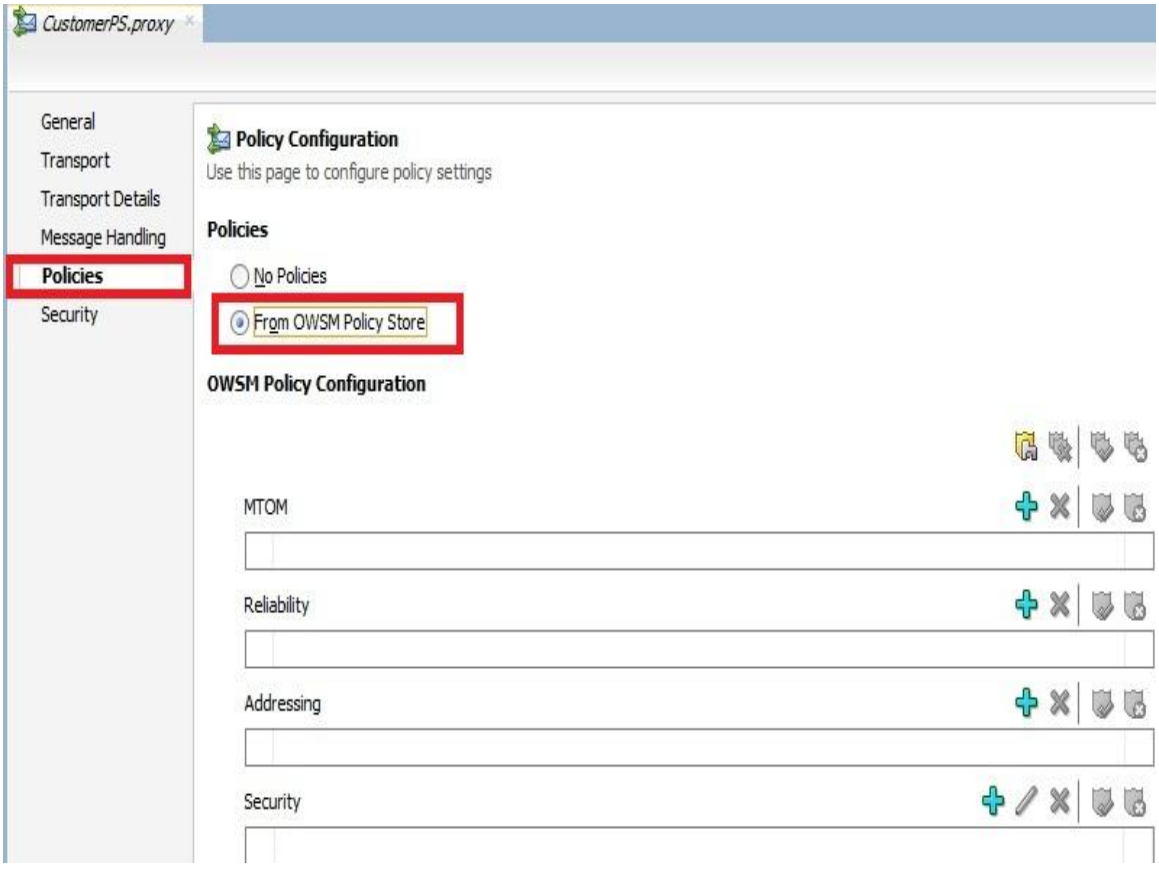


Click **OK** and observe new user got created. You can use this user to invoke your Proxy Services.

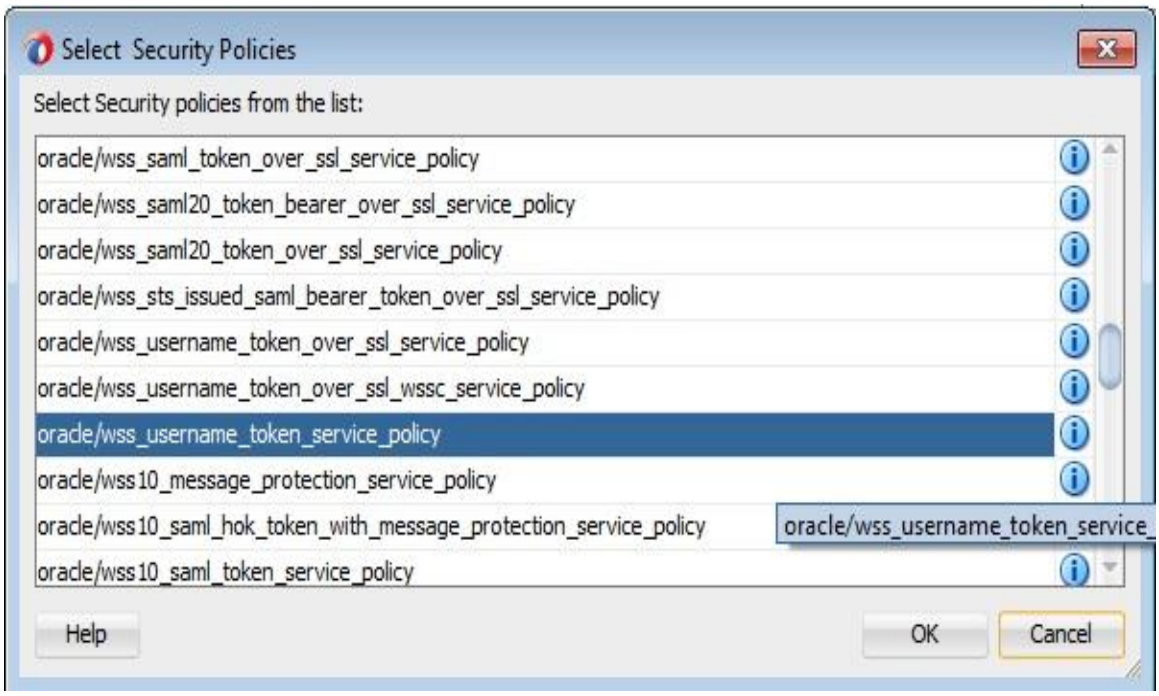


Attaching OWSM Policy

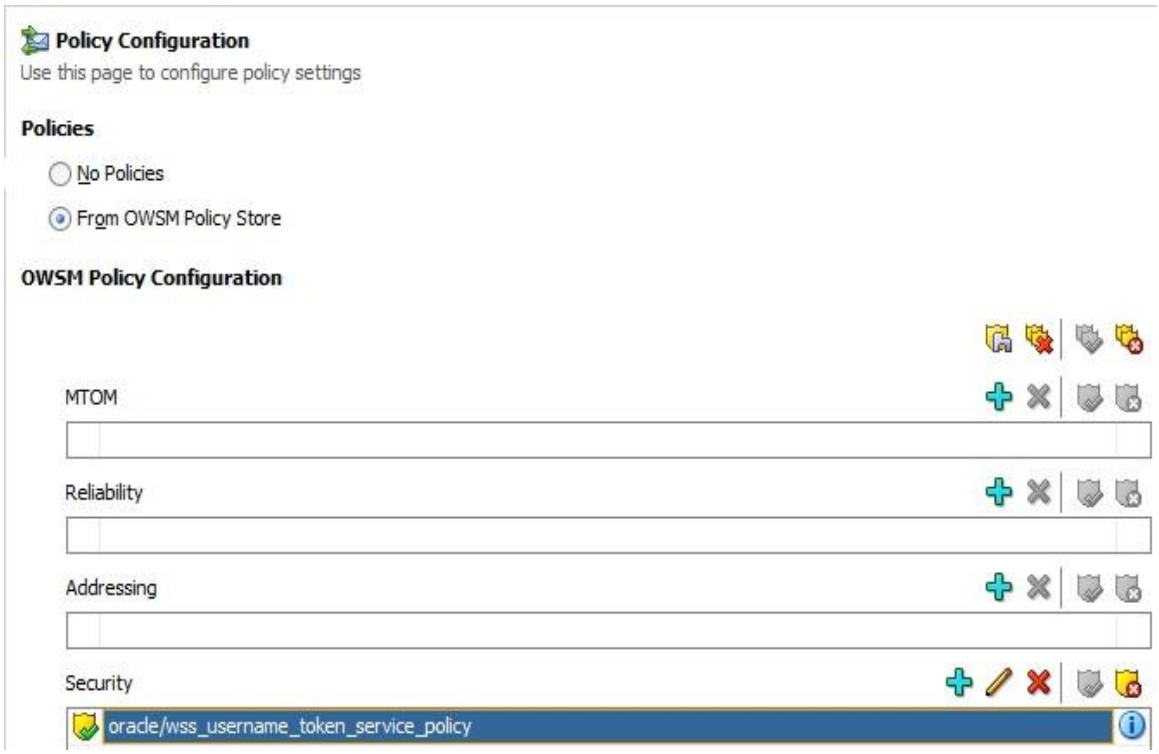
Open **CustomerPS**. Navigate to **Policies** tab and select option as shown below.



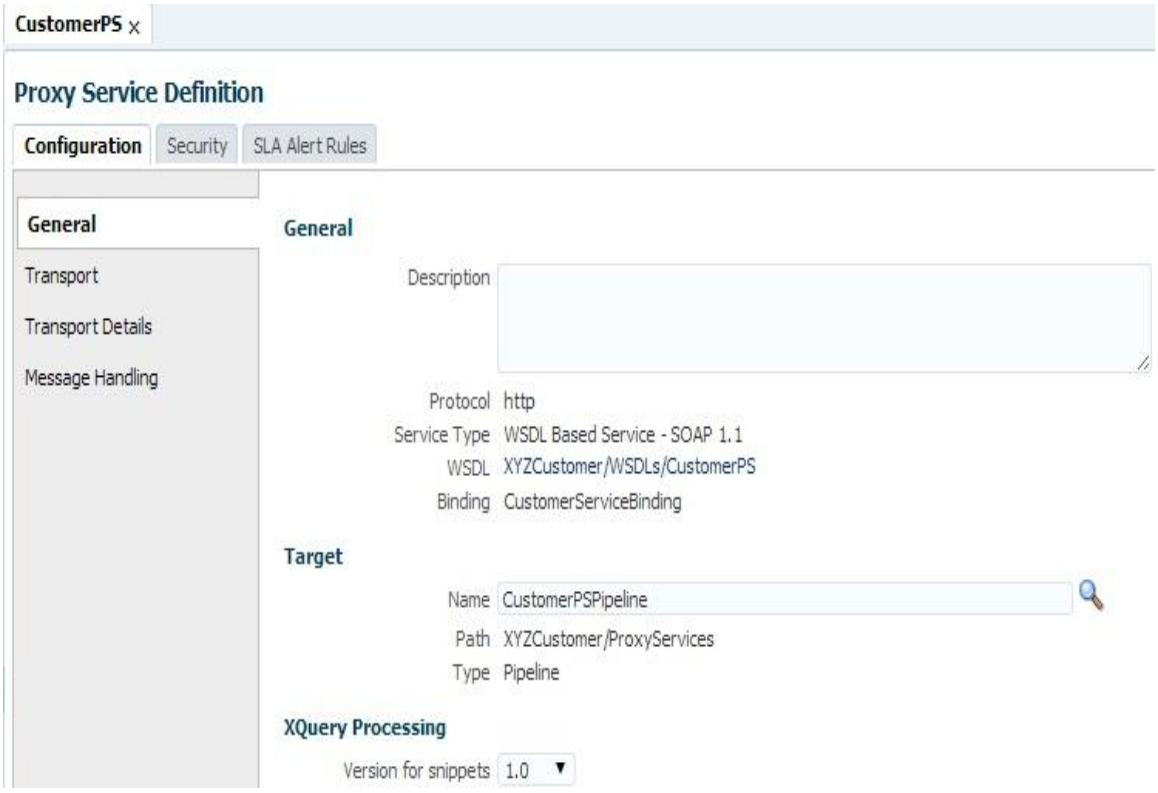
Click **+ icon** in **Security** section and select policy **oracle/wss_username_token_service_policy**.



Click **OK** and observe that selected policy is shown up in **Security** section as shown below.

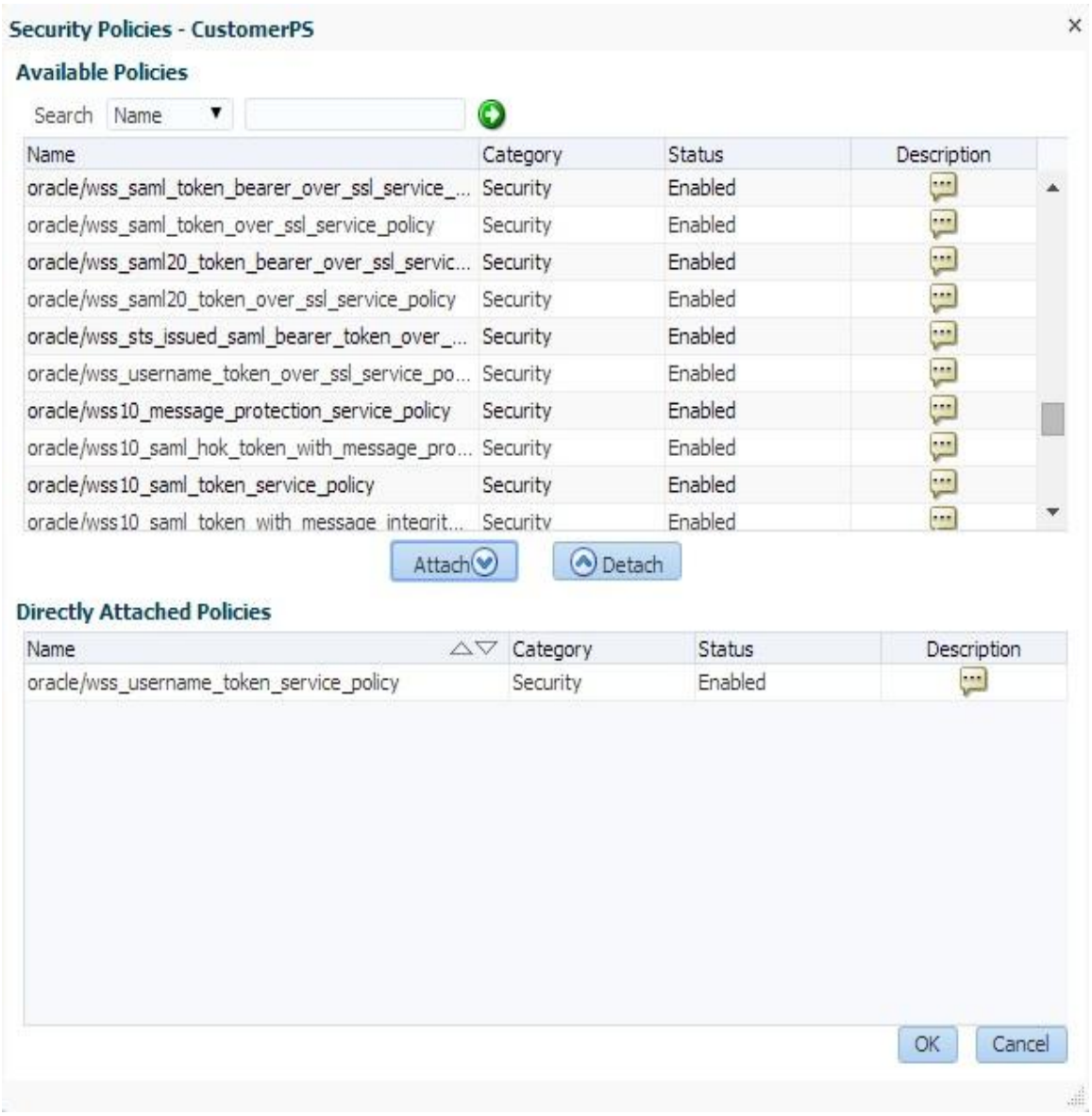


You can also attach OWSM policies to Proxy Service in **sbconsole**. Launch **sbconsole** and create a new session. Navigate to **All Projects -> XYZ Customer -> ProxyServices** and open **CustomerPS** which brings up a new tab as shown below.



Click **Security** and choose the option as shown below.

Click **Attach Policies** icon (highlighted above). Select required policy as shown below and click **Attach.**



Click **OK** and observe that selected policy is shown up and enabled as shown below.

Save your changes in current tab and activate the session.

Testing

You can use **SOAP UI** for testing. Create a test suite in SOAP UI using your proxy service WSDL. Refer to http://soapui.org for any additional help.

Open request editor for any operation and paste the following as one of the SOAP headers. This represents **WS-Security** header and is required by OWSM policy attached to your Proxy Service above. You can also observe **username** and **password** fields.

```
<wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
  <wsse:UsernameToken>
    <wsse:Username>UNAME1</wsse:Username>
    <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">Welcome1</wsse:Password>
  </wsse:UsernameToken>
</wsse:Security>
```

Test using wrong credentials and observe the output showing security error.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <soapenv:Fault>
      <faultcode>soapenv:Server</faultcode>
      <faultstring>OSB-386200: General web service security error</faultstring>
      <detail>
        <con:stack-trace xmlns:con="http://www.bea.com/wli/sb/context">com.bea.wli.sb.service.handlerchain.HandlerException: General web service security error
at com.bea.wli.sb.service.handlerchain.handlers.AbstractWssHandler.handleWssException(AbstractWssHandler.java:97)
at com.bea.wli.sb.service.disi.handlerchain.handlers.InboundWssPhase1DISIHandler.dispatch(InboundWssPhase1DISIHandler.java:113)
at com.bea.wli.sb.service.handlerchain.handlers.AbstractHandler.dispatch(AbstractHandler.java:131)
at com.bea.wli.sb.service.handlerchain.handlers.InboundMessageContentHandler.dispatch(InboundMessageContentHandler.java:169)
at com.bea.wli.sb.service.handlerchain.handlers.AbstractHandler.dispatch(AbstractHandler.java:131)
at com.bea.wli.sb.service.handlerchain.handlers.CheckAccessControl.dispatch(CheckAccessControl.java:65)
at com.bea.wli.sb.service.handlerchain.handlers.AbstractHandler.dispatch(AbstractHandler.java:131)
```

Test using credentials created in previous section and observe the output.


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header xmlns:urn="urn:xyzbank:cust:schema:customer"/>
   <soapenv:Body xmlns:urn="urn:xyzbank:cust:schema:customer">
      <cus:StatusMsg xmlns:cus="http://xmlns.xyzbank.com/schema/Customer">
         <CustomerID>12</CustomerID>
         <status>S</status>
      </cus:StatusMsg>
   </soapenv:Body>
</soapenv:Envelope>
```

Response