ORACLE UNIVERSITY

Oracle Service Bus 12*c*: Design & Integrate Services

Activity Guide
D88521GC10
Edition 1.0 | July 2015 | D92012

Learn more from Oracle University at **oracle.com/education/**

ORACLE

Author

Iris Li

Technical Contributors and Reviewers

Dimitri Laloue
Greg Fichtenholtz
Chad Schoettger
James Mills
Takyiu Liu
Matthew Slingsby
Armando Hernandez
Robert Wunderlich
Mike Muller

Editors

Malavika Jinka
Aishwarya Menon
Nikita Abraham

Graphic Designer

Kavya Bellur

Publishers

Michael Sebastian
Jayanthy Keshavamurthy
Jobi Varghese

Table of Contents

Course Practice Environment: Security Credentials

Course Practice Environment: Security Credentials

OS usernames and passwords can be obtained by:

- Asking your instructor or LVC producer for OS credential information, if you are attending a classroom-based or live virtual class.
- Referring communication received from Oracle University for this course, if you are using a self-study format.

For product-specific credentials used in this course, refer to the following table:

Product-Specific Credentials		
User Type	**Username**	**Password**
WebLogic Server administrator	`weblogic`	`welcome1`
WebLogic Server user that is authenticated by WLS default authenticator	`joe`	`welcome1`

`

Practices for Lesson 1: Introduction

Chapter 1

Practices for Lesson 1

Practices Overview

There are no practices for this lesson titled “Oracle Service Bus 12*c*: Design and Integrate Services–Course Introduction.”

General Notes

There are no notes.

Practices for Lesson 2: Introducing Oracle Service Bus

Chapter 2

Practices for Lesson 2

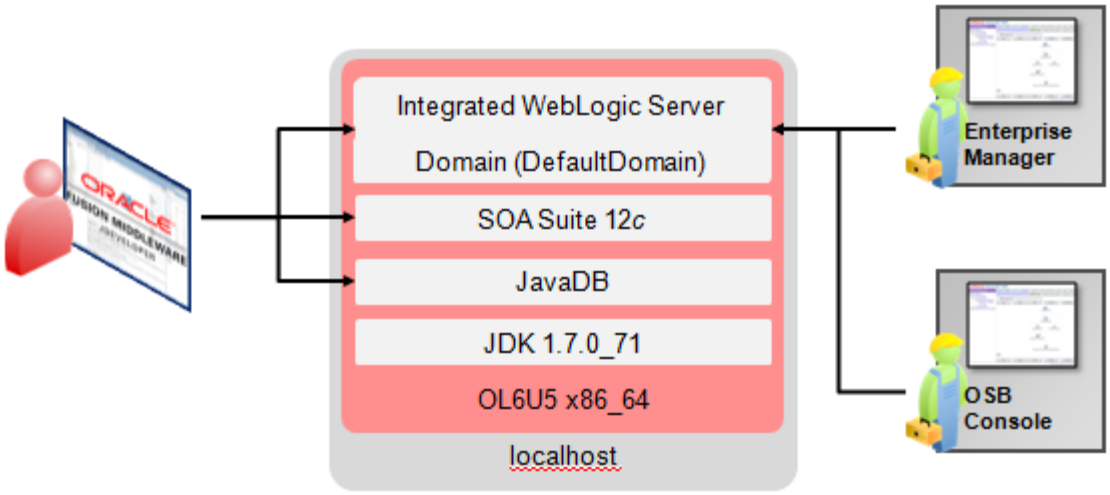
Practices Overview

The Oracle SOA Suite Quick Start distribution ensures quick installation of a development or an evaluation environment on a single-host computer. Quick Starts are installations meant only for development or evaluation.

In JDeveloper, the WebLogic Integrated Server is preconfigured with the preceding SOA Suite runtime components and JavaDB. JavaDB is a development database that allows you to start development with SOA Suite without the need to run the repository creation utility (RCU).

Your lab machine is preinstalled with the Quick Start distribution. In this lesson's practices, you will configure the Integrated WebLogic Server default domain.

Your environment looks like the following on the host server when you have finished:



Practice 2-1: Configuring the Integrated WebLogic Server's Default Domain in JDeveloper

Overview

By default, the Oracle SOA Suite Quick Start installation contains Oracle JDeveloper and an Integrated WebLogic Server.

In this practice, you will launch JDeveloper and configure the Integrated WebLogic Server's default domain in JDeveloper.

Tasks

1. Setup
   a. For Quick Start distribution, all software is installed in the single middleware home directory: `/u01/app/oracle/fmw_dev/12.1.3.0`

      An environment variable is set to this home directory. To verify, open a terminal window, and enter the following command:

      ```
      >echo $FMW_DEV_HOME
      ```

      You should see the above home directory in the output.

      Note: For this course, $FMW_DEV_HOME serves as $ORACLE_HOME, where your Oracle Fusion Middleware products are installed.

   b. Increase JDeveloper PermMemory size.
      1) In a terminal window, enter the following commands:

         ```
         cd $FMW_DEV_HOME/jdeveloper/jdev/bin/
         gedit jdev.conf
         ```

      2) In gedit editor, locate the following entry:

         ```
         AddVMOptionHotspot -XX:MaxPermSize=320M
         ```

      3) Modify the MaxPermSize value to 512M.

         ```
         AddVMOptionHotspot -XX:MaxPermSize=512M
         ```

      4) Save your edit and close gedit.

      Note: Increasing the memory size can help avoid “low memory” warning or “out of memory” error that you might encounter while developing and deploying projects.

   c. Set the `JDEV_USER_DIR` environment variable to define the location of the default domain. This will also determine where to save new applications created in Oracle JDeveloper.

      Open a terminal window, and enter the following commands:

      ```
      >export JDEV_USER_DIR=$HOME/domains/DefaultDomain
      ```

      Note: If you do not set JDEV_USER_DIR, the domain is created at the following default location:

      ```
      $HOME/.jdeveloper/system12.1.3.x.x.x.xxxx/DefaultDomain
      ```

2. Launch JDeveloper
   a. In the same terminal window that you set the `JDEV_USER_DIR` environment variable, enter the following command:

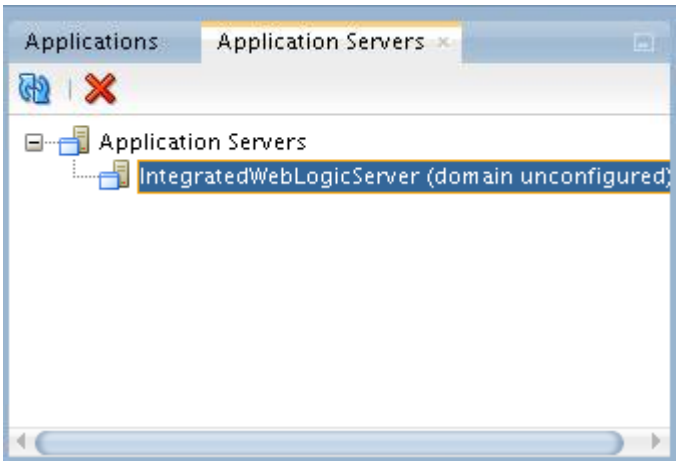
      ```
      $FMW_DEV_HOME/jdeveloper/jdev/bin/jdev
      ```

      As Oracle JDeveloper launches, you will get three prompts.
   b. When prompted for role, select **Studio Developer** as your role, deselect the “Always prompt for role selection on startup” option, and click **OK**.
   c. When prompted, say **No** to import preferences from a previous JDeveloper installation.
   d. When prompted about Oracle Usage Tracking, check whether or not you want Oracle JDeveloper to send automated usage reports to Oracle, and then press OK.

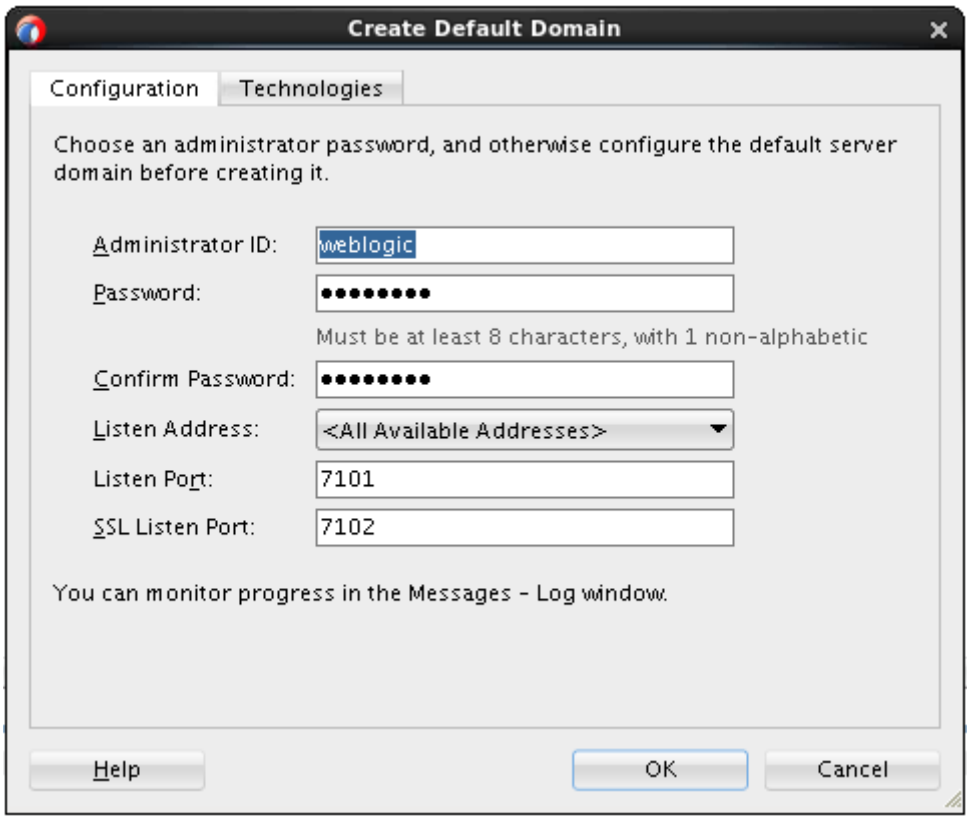
   JDeveloper has fully launched at this point.
3. Create a desktop shortcut for JDeveloper.
   a. Right click at a blank location on your desktop, and select **Create Launcher**.
   b. In the Create Launcher dialog box, select **Application** in Terminal as the Type.
   c. Enter **`JDeveloper`** as the name.

      Note that this name appears as the caption for the desktop icon.
   d. In the Command field text area, enter:

      ```
      gnome-terminal --title "Start JDev" -e
      "/home/oracle/labs_DI/resources/Scripts/startjdev.sh"
      ```
   e. In the Create Launcher dialog box, click OK.
      This should have created a "**JDeveloper**" icon on the desktop
4. Creating Default Domain and launching the Integrated WebLogic Server
   a. From the main menu under JDeveloper, select Window > Application Servers.
   b. In the Application Servers view, expand the Application Servers node, and you should see a domain unconfigured integrated WebLogic server, as shown in the following image:

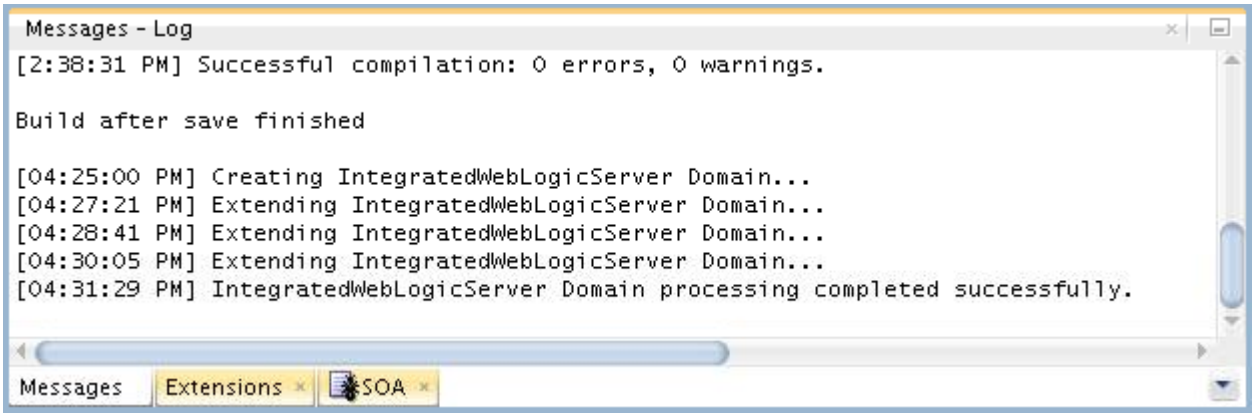
   c. Right-click the IntegratedWeblogicServer node, and from the context menu, select Create Default Domain.
   d. In the Create Default Domain wizard, on the Configuration tab page, retain the default Administrator ID, refer to “Course Practice Environment: Security Credentials” for the password, and accept the defaults for all other fields. Click **OK**.



e. The domain creation process starts and it takes several minutes.

You can monitor the progress in the Messages window pane. This window should automatically open at the bottom of the JDeveloper screen. If it is not there, you can open it by selecting Window > Log from the main menu.

f. When you see the following messages appear in the log, the default domain has been created successfully.



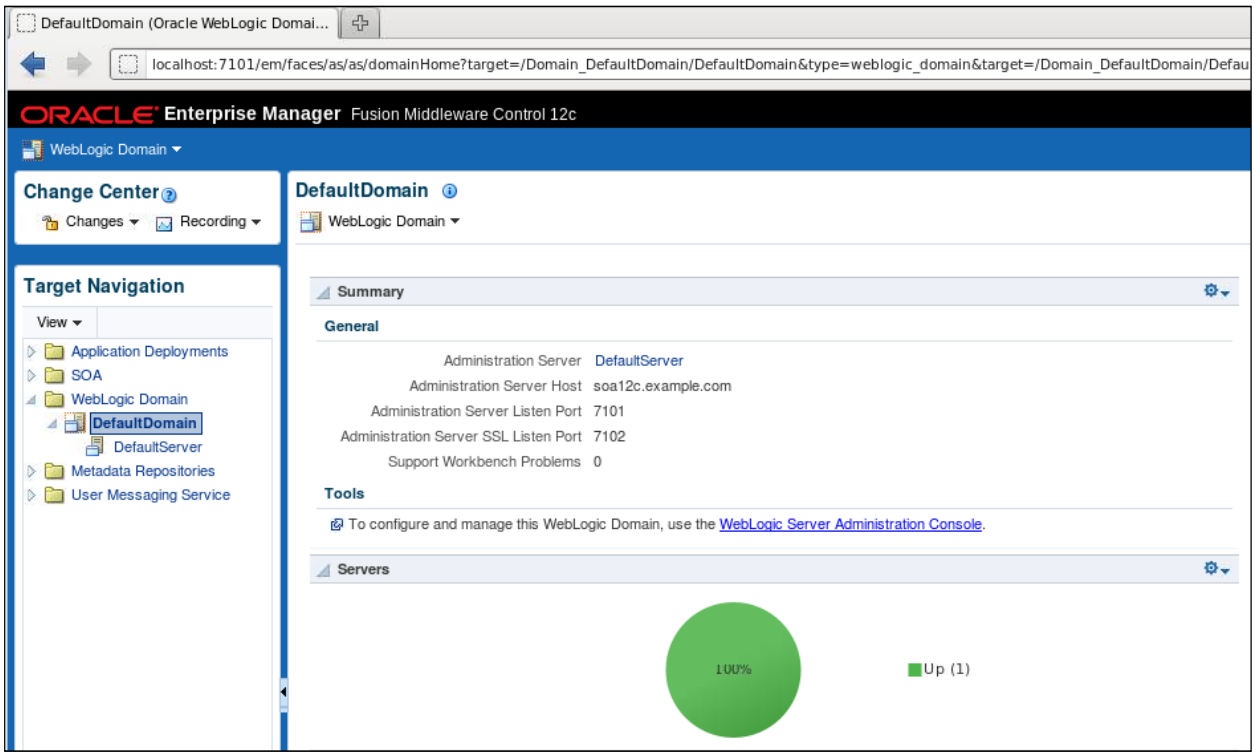
5. In the Application Servers view, right-click the IntegratedWeblogicServer node, and from the context menu, select Start Server Instance. This will take a few minutes. You should see an “SOA Platform is running and accepting requests” message in the Running: IntegratedWebLogicServer–Log window.

6. To verify that the default domain is configured properly and that the integrated server has started, perform the following steps:
   a. Open a Firefox browser, access Enterprise Manager Fusion Middleware Control with the following URL:

      `http://localhost:7101/em`

   b. Log in using the username and password that you specified while configuring the default domain. Refer to “Course Practice Environment: Security Credentials” if you don’t remember them.

      When you first log in, Fusion Middleware Control is displayed with the target navigation pane on the left and the domain home page on the right.



   c. You can expand SOA > service-bus and SOA > soa-infra > default. At this point, both folders are empty.

Practices for Lesson 3: Getting Started with Service Bus Applications

Chapter 3

Practices for Lesson 3: Overview

Practices Overview

Service Bus can protect consumers of the backend service from routine changes such as deployment location and implementation updates.

In these practices, you create a virtualized end-point for an existing SOAP style service. In addition, you learn how to build a team development environment by using MDS.

Practice 3-1: Deploying and Testing a SOA Composite

Overview

The composite validates credit card payments and returns the payment status. If the payment is denied, the order will not be processed.

In this practice, you deploy the ValidatePayment composite application to the integrated WebLogic server, and then initiate and test an instance in the Enterprise Manager Fusion Middleware Control (EM) console. This ValidatePayment will be used as the underlying service throughout the rest of the labs.

Assumption

The integrated WebLogic Server is up and running.

Tasks

High-level steps:

- Deploying the ValidatePayment composite application using a command
- Testing and examining the deployed composite

Deploying the composite

1. In a terminal window, enter the following commands:

```
>cd $HOME/labs_DI/lessons/lesson03
>ls -l
>$HOME/labs_DI/resources/Scripts/deploy.sh
sca_ValidatePayment_rev1.0.jar
```

2. The build and deployment information will be displayed in the console. You should see a message showing both build and deploying executed successfully.

Testing and examining the deployed composite

For ValidatePayment composite, all available credit cards are stored in the database, including payment type, card number, expiry date, card name, and daily limit.

The validation process includes three steps:

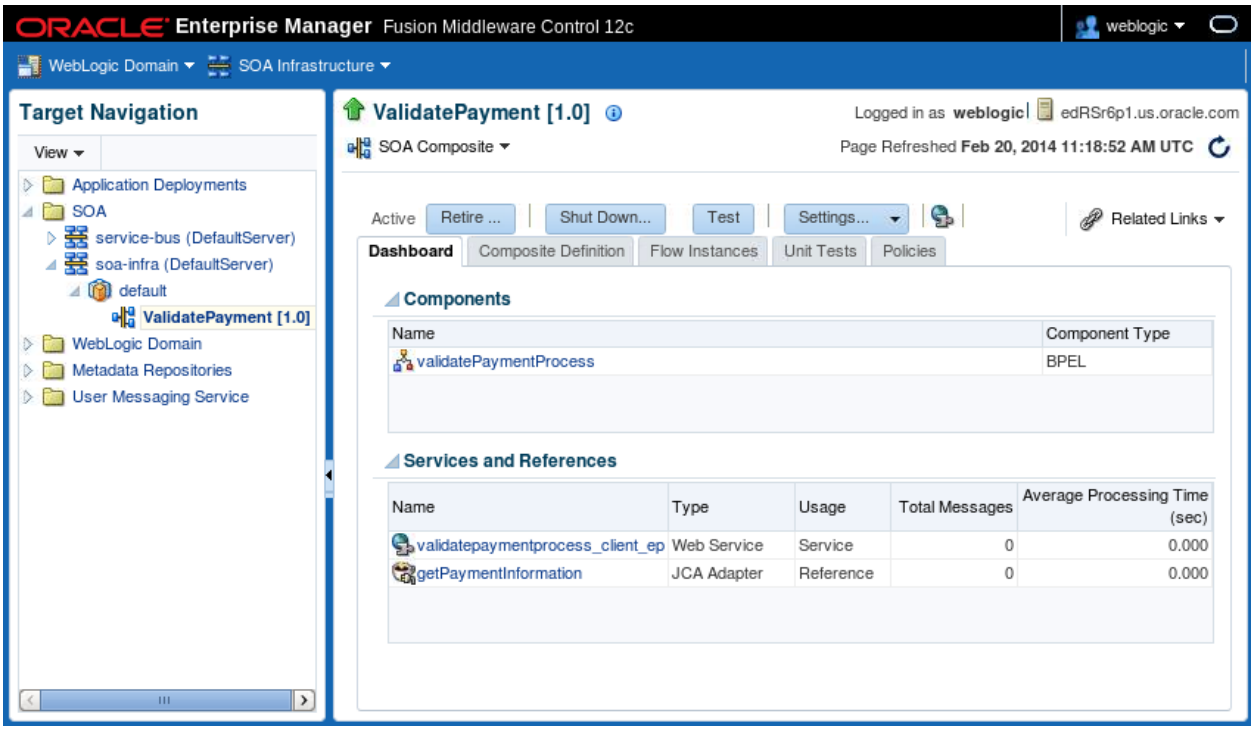
1) First, the payment information is retrieved from the database, using the credit card number quoted in the order message as the key. If there is no data available with this credit card number, payment is denied.
2) If data for the credit card number is available, the expiry date in the database record is compared to the expiry date listed in the order message. If they are not the same, the payment is also denied.
3) The last check compares whether the total order amount is less than the daily limit on the credit card in the database.

If all tests are successful, the payment is authorized. Otherwise it is denied.

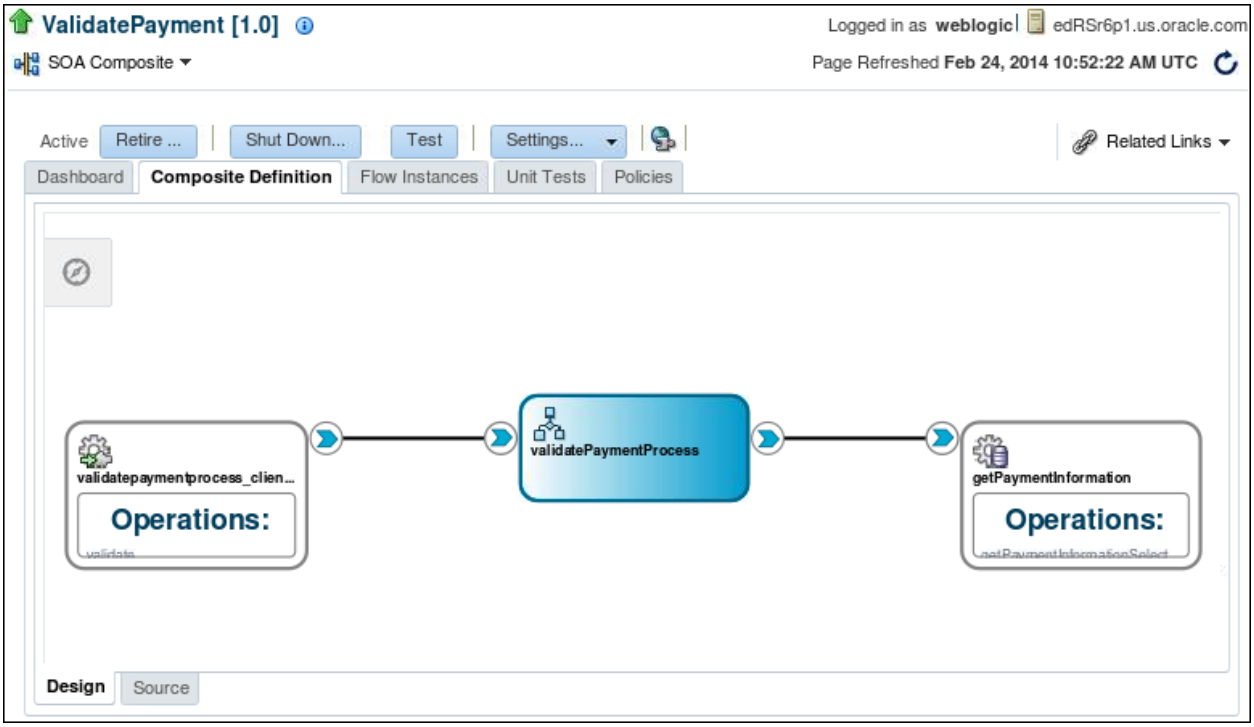
3. Examine the SOA composite application.
   a. Go to the Fusion Middleware Control console:
      http://localhost:7101/em
   b. In the Target Navigation panel, expand SOA > soa-infra > default. Now you should see the deployed ValidatePayment [1.0] composite application.

c. Click the ValidatePayment [1.0] link. The home page (Dashboard tab page) for the selected SOA composite application is displayed. You can view the service components and the service and reference binding components included in the composite.

Note: This page also enables you to perform composite management tasks such as retire, activate, shut down, start up, and test.

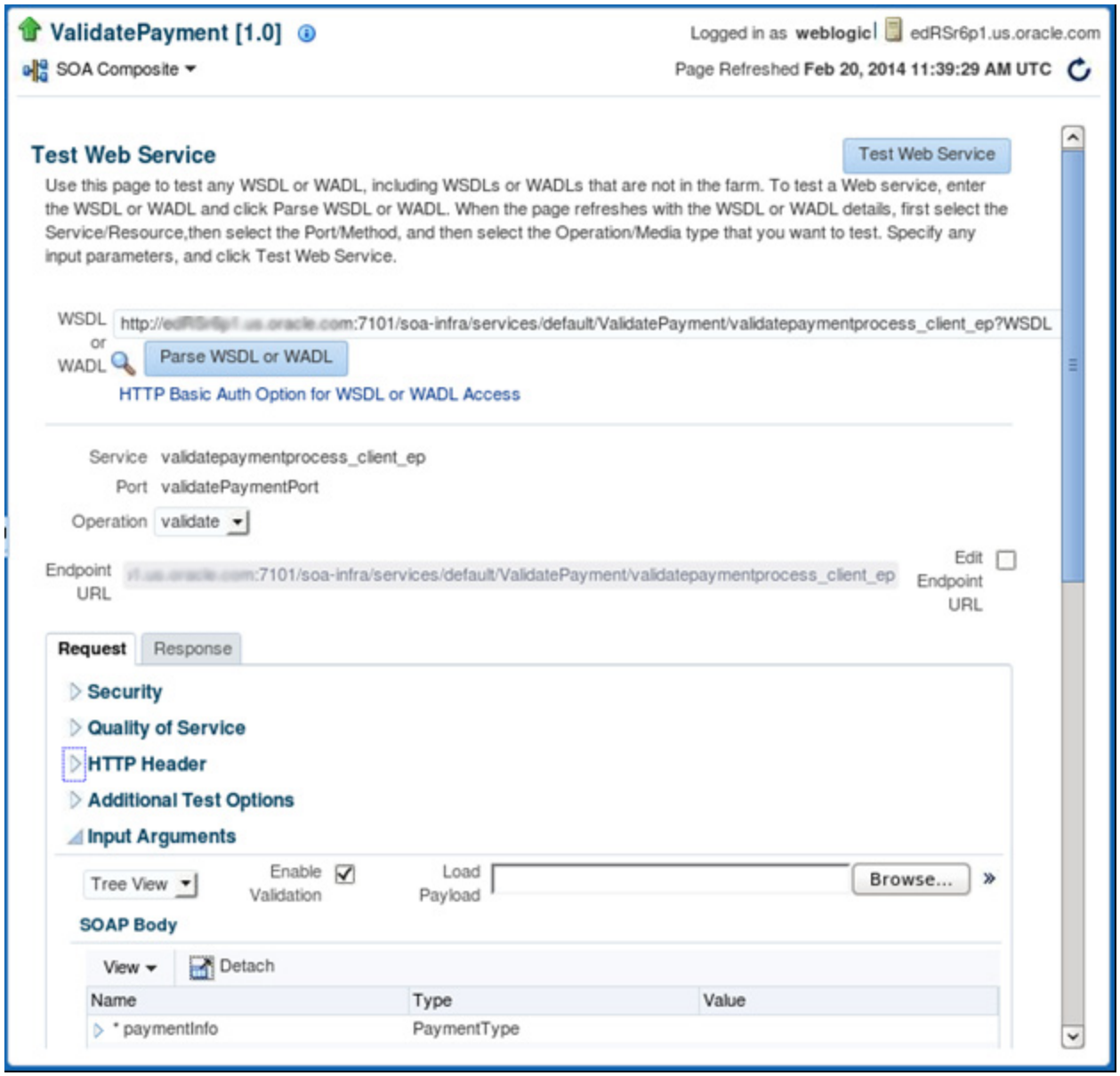


d. Click the Composite Definition tab. You should see a graphical display of the SOA composite applications designed in Oracle JDeveloper 12*c*. This view is similar to the display of the SOA composite application in Oracle JDeveloper.



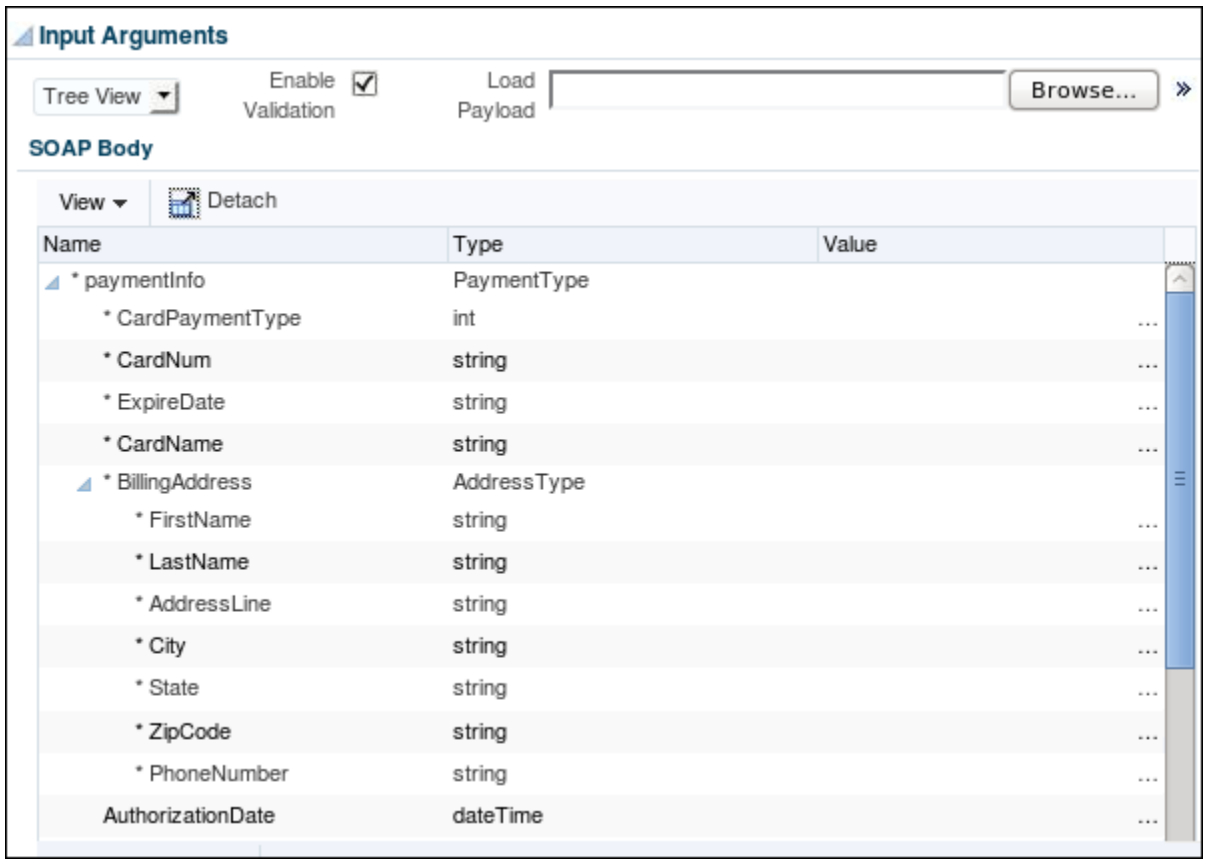
Here you can see that the implementation of this service uses a BPEL process to receive the incoming payment information, retrieve the credit card data from the database, and perform the tests outlined in step two.

4. Initiate and test the SOA composite application.
   a. On the ValidatePayment [1.0] home page, click Test.
      On the Test Web Service page, accept all default settings as shown below:



   b. Make sure that the Request tab is selected and scroll down to the **Input Arguments** section.
      To enter XML payload data, you have the option of selecting Tree View (default) or XML View.
      - Tree View: Displays a graphical interface of text fields in which to enter information.

- XML View: Displays the XML file format for inserting values. You can paste the raw XML payload of your message into this field.

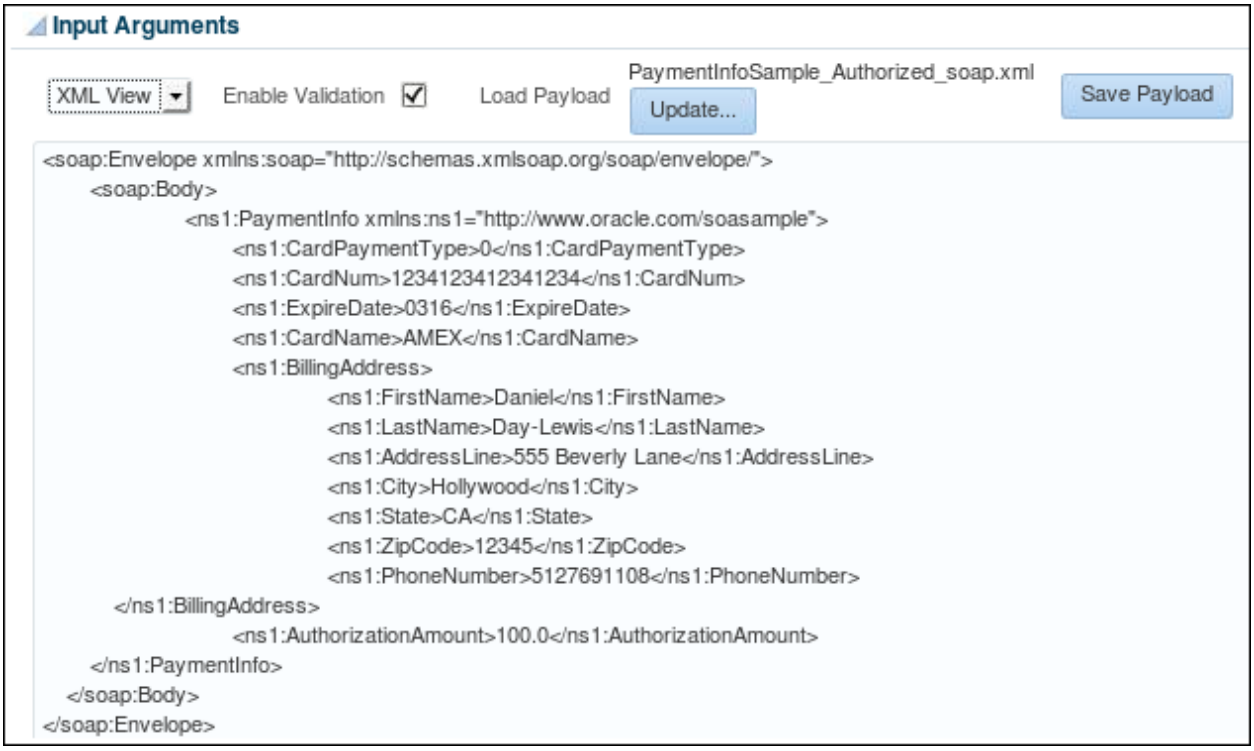


c. Select XML View. You should see the web service invocation payload in XML format.

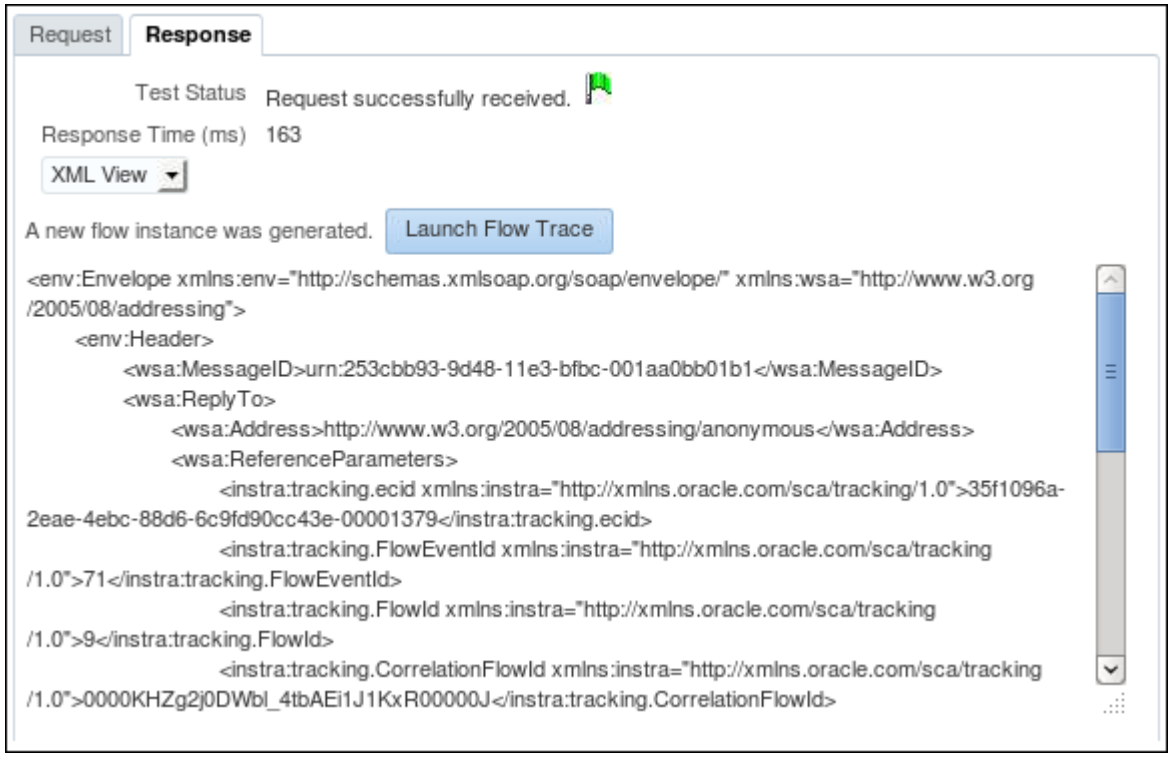
Tip: You can easily copy-and-paste your data into the XML View.

d. To load the Test message, click “Browse” next to the Load Payload field, and select the `/home/oracle/labs_DI/resources/sample_input/PaymentInfoSample_Authorized.xml` message.

This message already includes the SOAP envelope needed for the web service test.



e. Click Test Web Service. When the composite completes, the screen switches to the Response tab and the returned value is shown (similar to the screenshot below).



f. Review the response message, and you should see that the payment status is Authorized.

Note: You can also switch to Tree View to view the result.

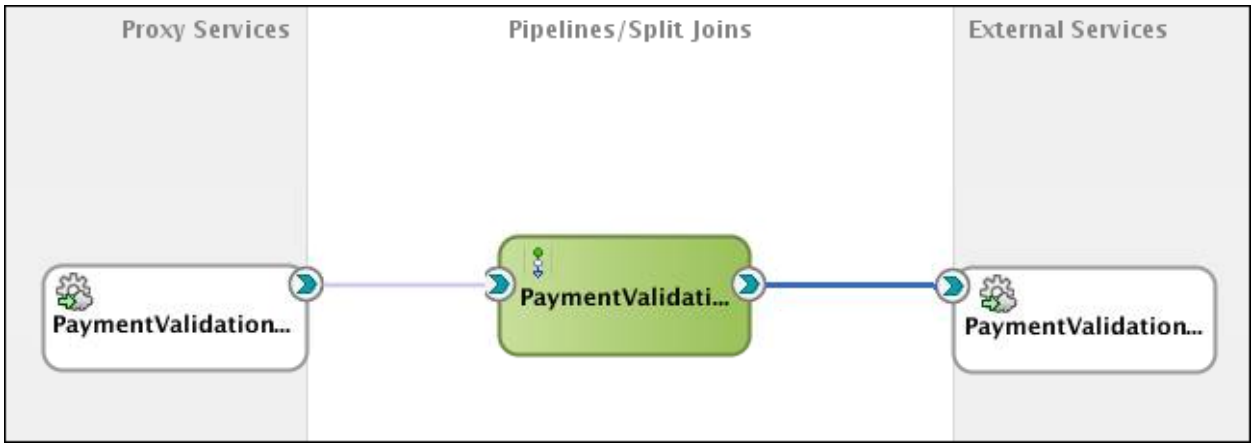
Practice 3-2: Virtualizing Service with Service Bus

Overview

In this practice, you register the `validatePayment` composite on Oracle Service Bus. Service Bus protects consumers of the `validatePayment` composite from routine changes such as changes of deployment location and implementation updates. Service Bus helps scale the service to handle higher volumes of requests and provide resiliency for the service, if it needs to be taken down for routine maintenance.

Start by creating a business service to register the composite URI. Then, add a simple pipeline and proxy. Pipelines can contain actions performed on the Service Bus; typically reporting, data transformation, and validation; before invoking the backend service. Consumers of the `validatePayment` service will invoke via a proxy rather than connect directly to the composite, allowing more agility and flexibility in managing change.

After completing this practice, the Service Bus application will look like the following screenshot:



High-Level Steps

In this lab, you accomplish the following tasks:

- Create a new Service Bus application and project.
- Define a folder structure for the Service Bus project.
- Create a business service to call the endpoint web service (SOAP-based), `ValidatePayment`, and review its properties.
- Generate a simple pass-through proxy service.
- Test and debug the end-to-end application.

Assumptions

`ValidatePayment [1.0]` has been deployed and is running on the SOA server.

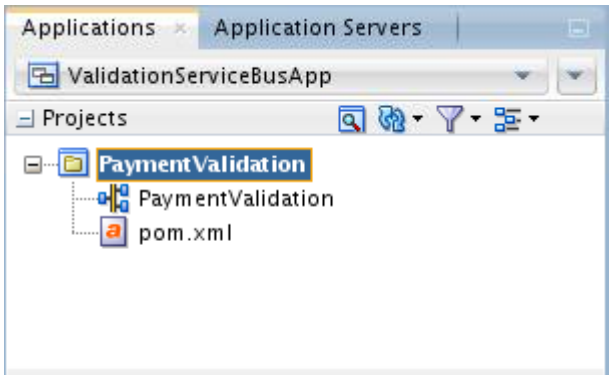
Tasks

Create a Service Bus application and project

The first steps in building a new application are to assign it a name and to specify the directory in which to save the source files. By creating an application using application templates provided by Oracle JDeveloper, you automatically get the organization of the workspace into projects, along with the project overview file.

1. Create a new Service Bus application.
   a. Open JDeveloper if it is not open.
   b. Select **File > New > From Gallery** from the menu.
      The New Gallery dialog box opens.
   c. Under Categories, select **Service Bus Tier**.
   d. Under Items, select **Service Bus Application with Service Bus Project**
   e. Click **OK**.
      The Create Service Bus Application with Service Bus Project dialog box opens.
   f. Specify the application name to `PaymentValidationServiceBusApp`.
   g. Enter `/home/oracle/labs_DI/lessons/lesson03/PaymentValidationServiceBusApp` for the directory.
   h. Click **Next**.
      You are prompted to create a new project.
   i. Set the project name to `PaymentValidation`
   j. Click **Finish**.

   Now your Project Explorer should look like the image below, showing Service Bus application and Service Bus project. The project folder contains two files: pom.xml and PaymentValidation. PaymentValidation, with the same name as your project, is called project overview file.



2. Double-click the project overview file, `PaymentValidation`.
   The Services Bus Overview Editor opens on the right.
   Note: The Overview Editor is a new view for Service Bus in 12*c*, and it is modeled on the SOA Composite Editor. This view allows construction of Service Bus projects using a top-down, drag-and-drop approach.
3. Take a few moments to browse the Applications pane on the left side of the window and the component palette on the right.
   - In the component palette, notice that the resources category contains pipeline and Split-Join icons. These are the components for a Service Bus application.
     Note: In 12*c*, the pipeline has been split from the proxy to allow it to be a re-usable component.
   - Other palette categories—Technology, Applications, and Advanced—contain adapters and transports for building business services (External References) and proxies (Exposed Services).

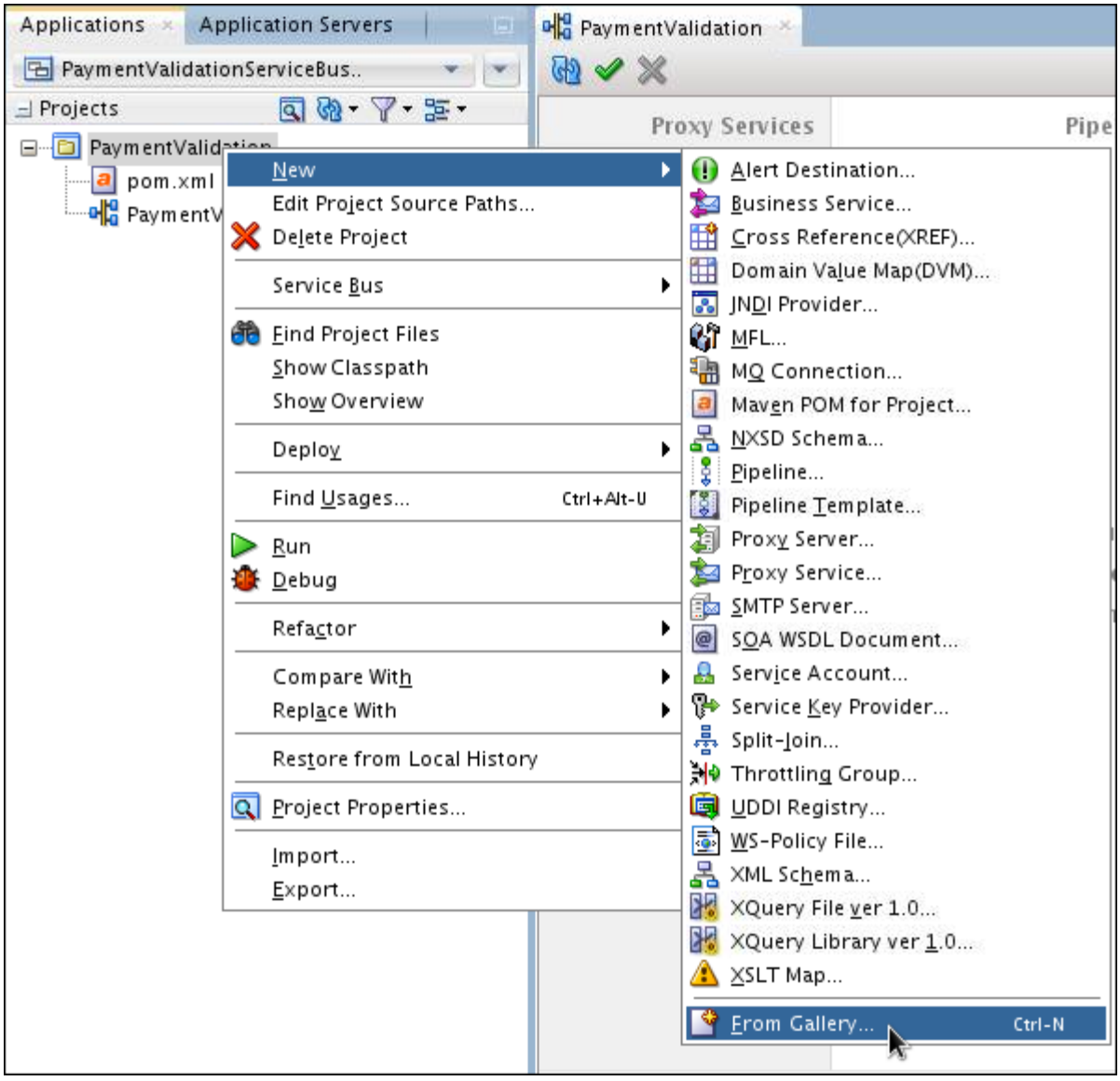
- If the Properties window is on the bottom right of the JDeveloper window, drag and position it to the bottom center. This will make editing properties of pipeline actions easier.

Define a folder structure and import artifacts

In Service Bus applications, folders are leveraged to organize artifacts within a project. For new applications, you are encouraged to create folders that align with the default folders in the composite application. Folders will not be automatically created in Service Bus applications, so that backward compatibility of Service Bus projects imported from previous releases is maintained.

For this practice, you keep the structure simple, because there are only a few artifacts to manage. As the projects grow, you can add folders for categorizing artifacts into business service, proxy, and templates.

4. Create two folders.
   a. Right-click the `PaymentValidation` project folder, and select **New > From Gallery**.



b. In the New Gallery window, select **General** (Categories) **> Folder** (items).

c. Click **OK**.

The Create Folder dialog box is displayed.

Note: This is only necessary the first time a folder is added. Thereafter, it will be listed on the menu by default.

d. Name the folder `Schemas`, and click **OK**.

e. Similarly, add another folder named `WSDLs`.

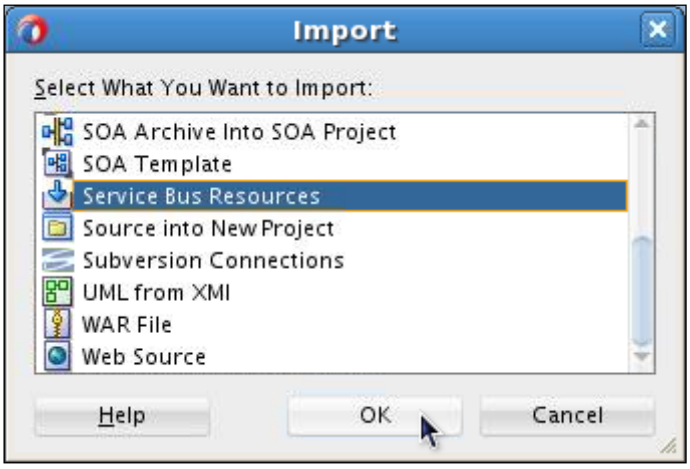
f. Verify your work.

5. Import Artifacts. This step is to register the WSDL for the ValidatePayment service that you will abstract with the business service.

a. Right-click the **WSDLs** folder in the Project Explorer, and select **Import** from the context menu.

The Import dialog box is displayed.

b. Select **Service Bus Resources**.

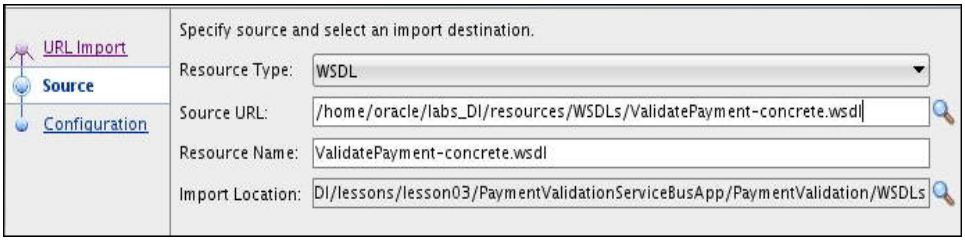


c. Click **OK**.

The Import Service Bus resources wizard is displayed. The title bar of the wizard dialog box shows the step number.

d. Complete the steps in the wizard by following the instructions in the table below:

Step	Page Description	Choices or Values
1)	Type	Select **resources from URL.** Click Next.
2)	Source	Click the Browse icon next to the Source URL field. In the Select WSDL dialog box, select File System in the top box, navigate to the `/home/oracle/labs_DI/resources/WSDLs` folder,

Step	Page Description	Choices or Values
		and select `ValidatePayment-concrete.wsdl`.  Click Next.
3)	Configuration	Accept the default. Make sure that you see the Schemas folder at the same level as the WSDLs folder containing CanonicalOrder.xsd. This directory structure is required to successfully import the WSDL. Click Finish.

The Applications pane resembles the following:

Service Bus supports the ability to import and export artifacts and projects at a fine-grain level. You may import artifacts individually, such as a WSDL or schema, or whole projects. You may control whether dependencies are included.

By default, when importing individual artifacts, Service Bus will attempt to import all dependencies declared in the artifact. For example, if a WSDL includes a schema, the schema will also be imported if the paths are relative.

Also note that you can control whether environmental settings, security policies, and credentials are preserved on import. For example, if you are bringing artifacts from a production environment for testing and editing, you may not want all the same policies applied in the development environment.

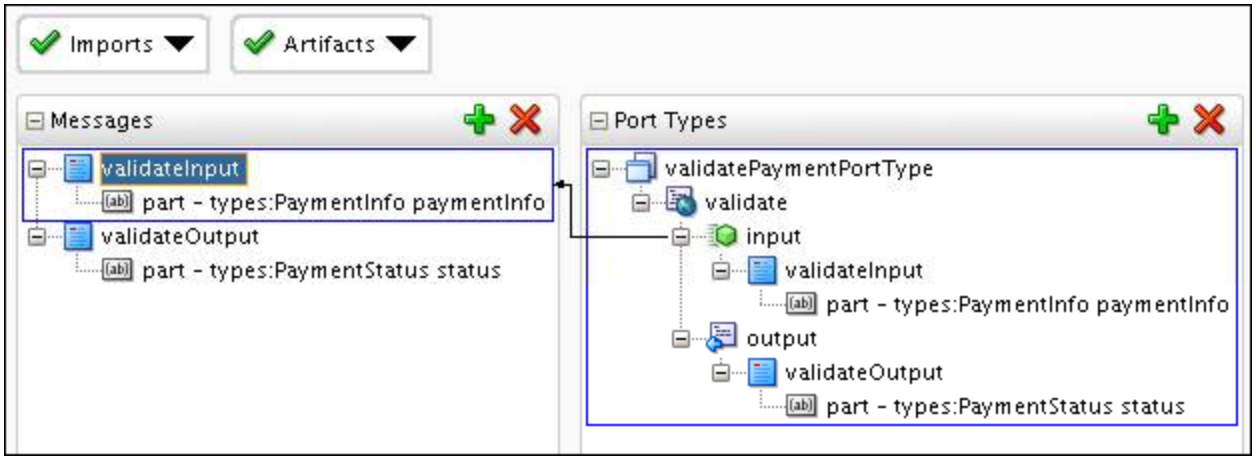
Inspecting WSDL and Schema files

6. Double-click the `ValidatePayment-concrete.wsdl` file in the WSDLs folder. The WSDL editor opens and presents the design view of the WSDL document.

   In this graphical overview of the WSDL file, you can explore the structure and relationship of the elements.

7. You should see the contents of Port Types, Bindings / Partner Link Types and Services. The WSDL document contains the portType element, a set of operations (in this case, an operation named `validate`), and the abstract messages (input and output) involved with those operations. Click the “Show Contents” icon on the Message tag.

8. A message can consist of multiple parts. Each part can be seen to represent a parameter in the operation request or response. Click validatePaymentPortType > validate> input element in the Port Types pane, and a link appears showing the relationship.



Message part can be seen to represent a parameter in the operation request or response. A message can consist of multiple parts.

Note: In general, it is recommended to stick with single-part messages because they are less complex and less likely to have you run into tool limitations.

9. Click the Source tab at the bottom of the WSDL editor. The source view of the WSDL document presents.

   Note that the WSDL has both binding and service definition.

10. Locate the `<types>` section. The type definitions can contain XSD-style elements or import one or more external XSD documents. In this case, the WSDL document imports message definitions using an external XSD document, `CanonicalOrder.xsd`.

```
<wsdl:definitions targetNamespace="http://www.oracle.com/ValidatePayment"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:tns="http://www.oracle.com/ValidatePayment"
                  xmlns:types="http://www.oracle.com/soasample">
  <wsdl:types>
    <xsd:schema>
      <xsd:import schemaLocation="../Schemas/CanonicalOrder.xsd" namespace="http://www.
    </xsd:schema>
  </wsdl:types>
```

Examine the XML Schema

Now you will open the supplied XML schema file using the XML schema (XSD) editor. This enables you to view the structure of a message payload.

When composing applications using services, it is important that the services use a common vocabulary or language. All services base their interface on one data model. The benefits of this are:

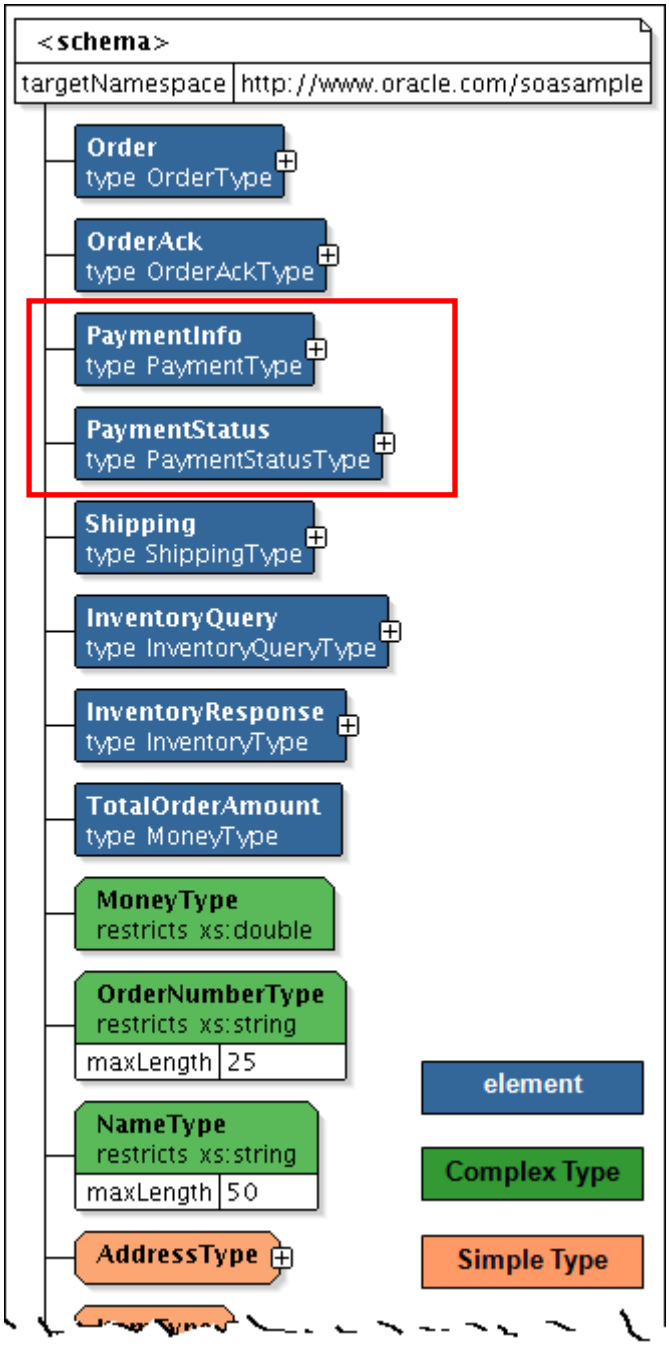
- Standardizing the service contracts so that consumers encounter the same data structures in all services
- Lowering the number of message formats that an application has to know about, so that the number of error-prone transformations can be reduced.

The `CanonicalOrder.xsd` defines the message structure of various services in the order management system.

11. You can view the XSD file by using one of the following two ways:
    a. In the design view of the WSDL editor, click the Import drop-down menu, and select the `CanonicalOrder.xsd` file.

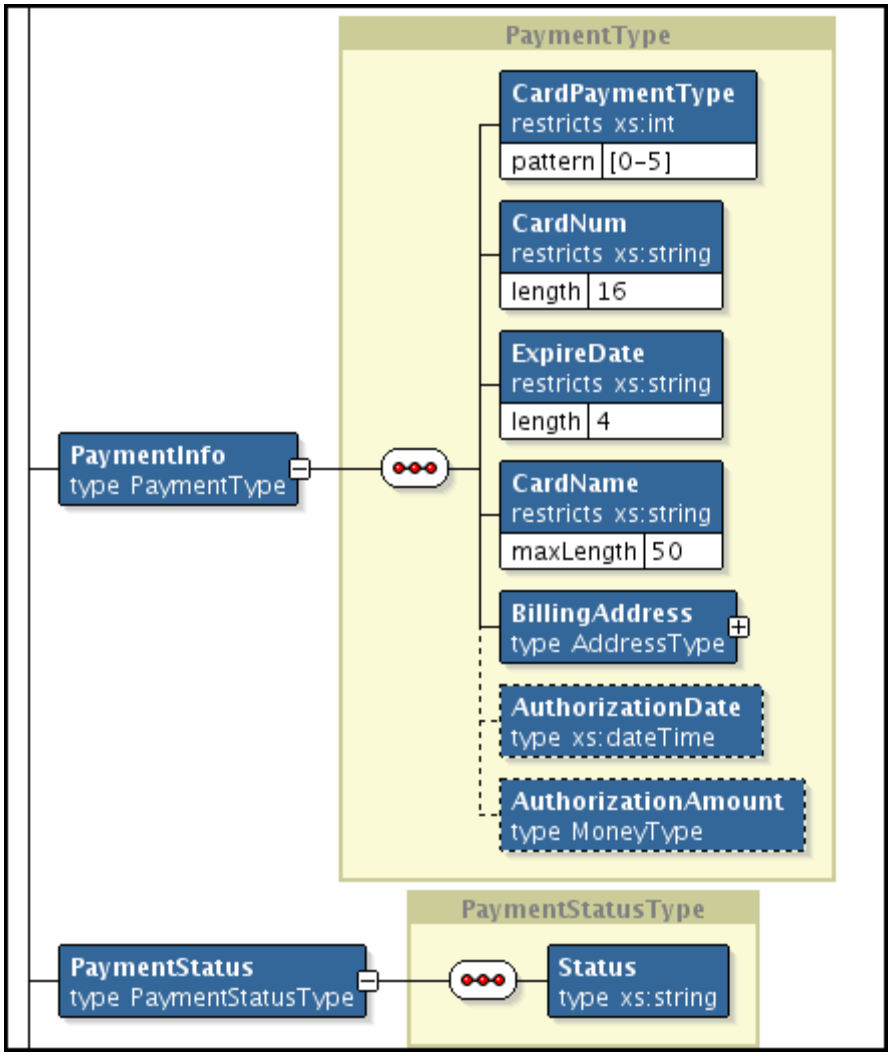
    Or

    b. Double-click the file in the project's Schemas folder:
12. The XSD editor opens and presents the design view of the XSD document. You can view the UML class model and entity relationship.



Note that the WSDLs of ValidatePayment composite only contain the elements `PaymentInfo` and `PaymentStatus` to define data structure.

13. Expand `PaymentInfo` and `PaymentStatus` elements to view their structure.



Note: In the previous practice, the test file (`PaymentInfoSample_Authorized_soap.xml`) contained one `PaymentInfo` element.

14. Click the Source tab at the bottom of the XSD editor to view the source code of the XSD document.

Create a business service

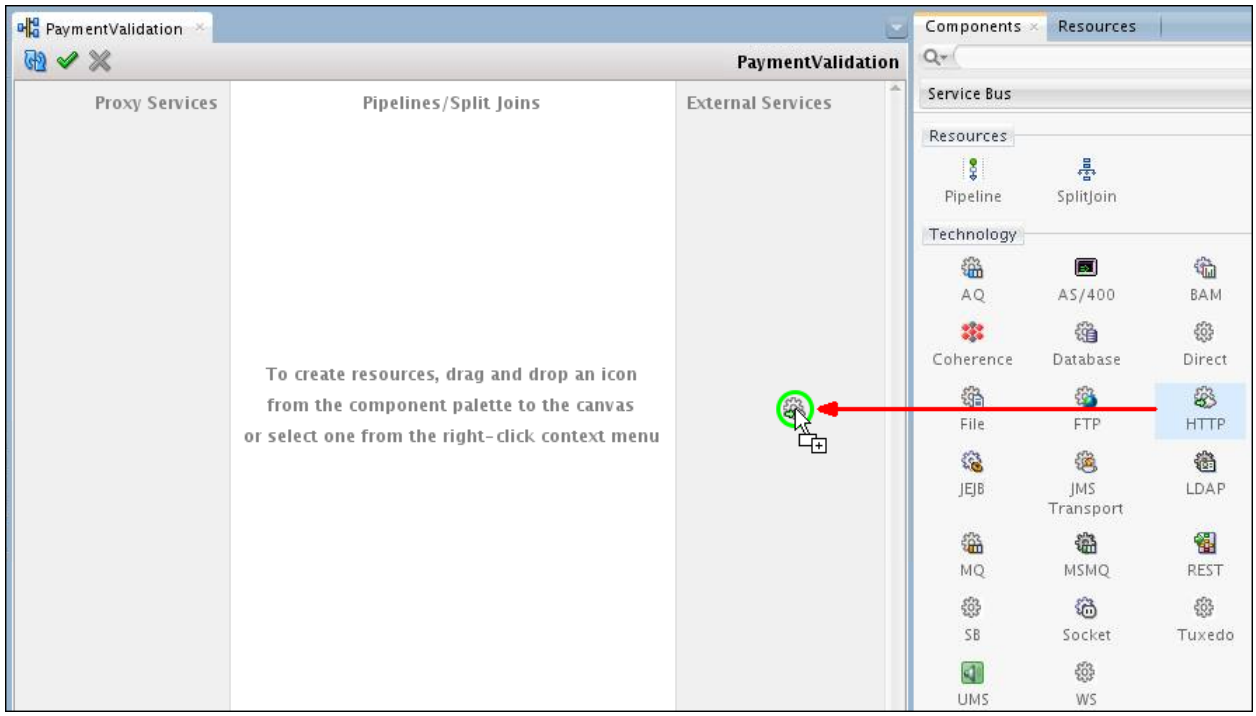
In Service Bus, business services provide an abstraction layer and take care of communication with service providers.

There are different ways to create artifacts in Service Bus in 12*c* JDeveloper:

- Use the right-click menu in the left Applications pane.
- Drag-and-drop icons from the component palette onto the Overview canvas.
- Right-click the overview canvas to insert artifacts.

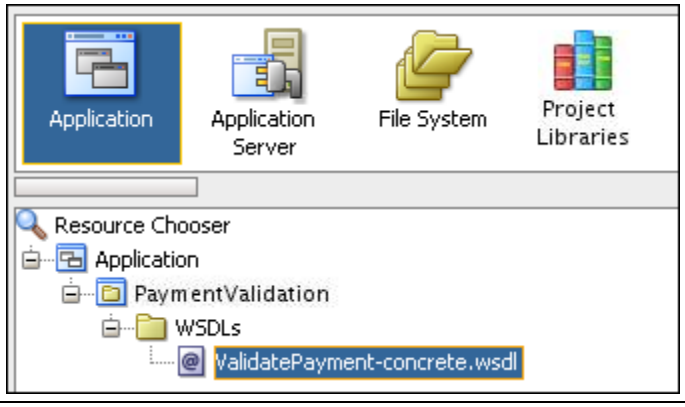
Drag-and-drop component palette to build the Service Bus project. However, feel free to experiment with other mechanisms.

15. Create a business service.
    a. In the PaymentValidation Overview Editor, drag the HTTP component (in the Technology section) from the Component palette and drop it onto the External Services lane.



    Upon dropping, the Create business service wizard pops up.

    b. Enter the required details for business by following the instructions in the table below:

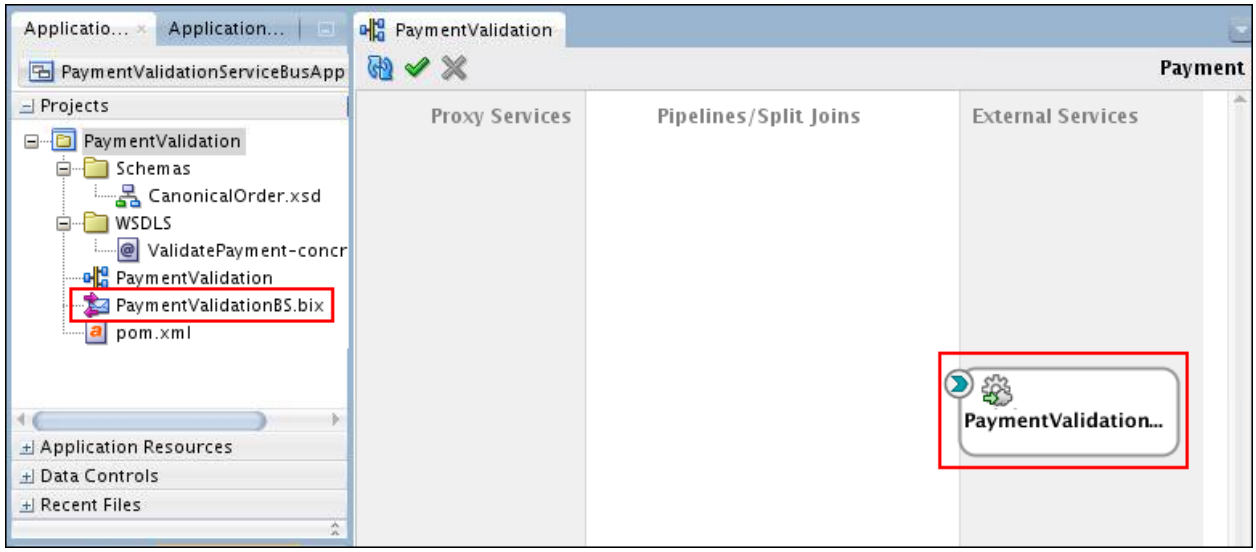
Step	Window/Page Description	Choices or Values
1)	Create Service	Name the service `PaymentValidationBS`. Click Next.
2)	Type	1) Select **WSDL** for Service Type. 2) Click the Browse WSDLs icon to the right of the WSDL choice. 3) In the Select WSDL dialog box, click **Application**, navigate and select **ValidatePayment-concrete.wsdl** as shown below: 

Step	Window/Page Description	Choices or Values
		4) Click **OK**. 5) Back in the Create Business Service window, you should see the WSDL and port fields populated with the information based on the WSDL you chose. 6) Click **Next**.
3)	Transport	Verify that http is selected in the Transport field. The Endpoint URI is populated as: http://localhost:7101/soa-infra/services/default/ValidatePayment/validatepaymentprocess_client_ep Make sure this URI is pointing to the deployed validatePayment service that you deployed to the server. Click **Finish**.

Note: There are several ways to check the deployed endpoint of a SOA service. One way is to visit the EM console (http://localhost:7101/em) and navigate to the composite. Click the Test icon near the top right portion of the window. This will bring up a Web Service test page that lists the deployed endpoint.

If needed, the endpoint URI can be updated as you review the business service properties on the Transport tab.

c. Business Service is created on the External Services lane and its file resides under the project folder.



Test the Business Service

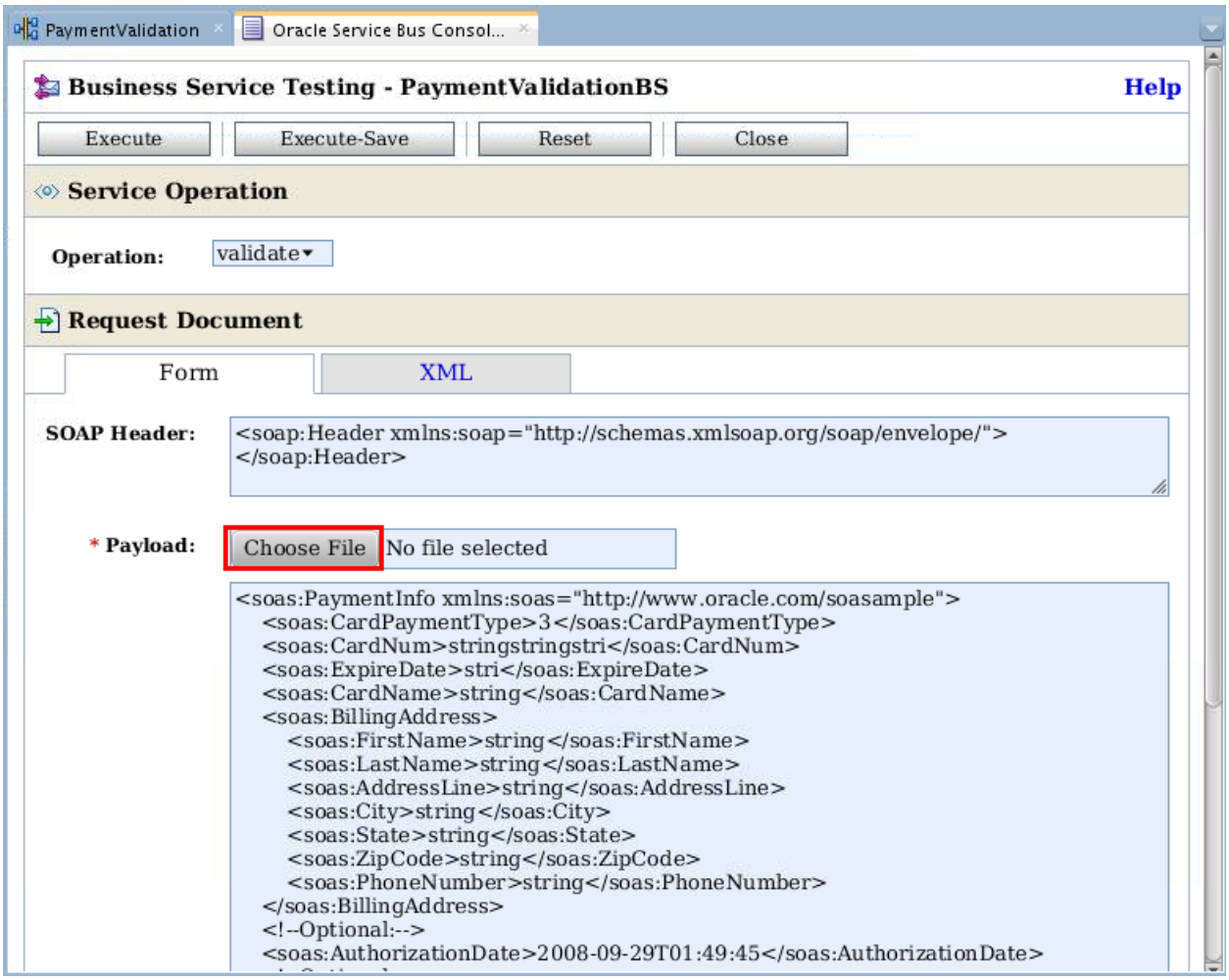
It is always recommended to run your Business/Proxy services to verify that you are able to send requests and receive responses as expected.

Tip: Typically, if the service provider is external to your organization, then you have to configure business service to use HTTP proxy server.

In 12*c*, you can use either Integrated or Standalone WLS to test business services.

- **Standalone:** Deploy your Service Bus project to standalone WLS from JDeveloper and test from sbconsole.
- **Integrated:** Right click your Business/Proxy service and select **Run**. In this course, we use integrated WLS for deployment and test.

16. Perform the following steps to test:
    a. Right-click the PaymentValidationBS in the External Services Lane. Select **Run**.
    The Test Console will activate as one of your tab windows in JDeveloper.



    By default, a sample payload will be generated for you; however, you will test it with a specific file.
    b. Click the **Choose File** button.
    c. Navigate to `/home/oracle/labs_DI/resources/sample_input/`, and select **`PaymentInfoSample_Authorized.xml`**
    d. Click **Open**.
    e. On the Test Console, click the **Execute** button.
    The next screen, in the Response Document section, indicates that the payment has been authorized.

```
<env:Body>
   <PaymentStatus xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/" xmlns:wsdl="http://schema
                  xmlns:ns2="http://www.oracle.com/ValidatePayment" xmlns:tns="http://xmlns.or
                  /ValidatePayment/getPaymentInformation" xmlns:soap="http://schemas.xmlsoap.c
                  xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/" xmlns:ns1 ="htt
                  xmlns="http://www.oracle.com/soasample">
      <Status>Authorized</Status>
   </PaymentStatus>
```

Create a simple pass-through proxy service

Now you create a proxy service and add a pipeline, which will allow a consumer to call your service on the OSB. The proxy will be the interface to the service from external consumers. The pipeline contains actions that will be performed before invoking the composite. Typical actions are data transform and validation, reporting with error handling.

When OSB needs to support the same web service interface as the backend service does, then creating a pass-through service is the quickest and easiest way.

17. Create Proxy Service and add pipeline.
    a. In Overview Editor, drag and drop the HTTP component from the Components palette to the Proxy Services lane.

       The Create Proxy Service wizard pops up.

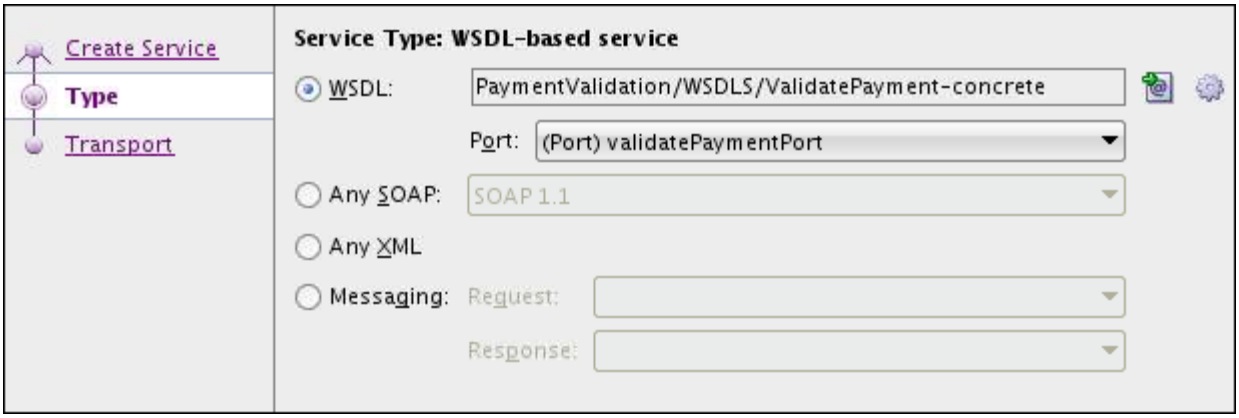
    b. In step one (Create Service), fill up the details for Proxy Service:
       - Service Name: **PaymentValidationPS**
       - Location: (keep as is)
       - Transport: http (keep as is)
       - Keep “Generate Pipeline” checked for creating pipeline along with Proxy Service
       - Change the pipeline name to **PaymentValidationPP**

       Click **Next**.

    c. In step two (Type), use the same WSDL of the business service for the proxy service:
       - Select service type **WSDL.**
       - Click the **Browse WSDLs** icon on the right.

         The Select WSDL dialog box pops up.
       - Navigate to Application > PaymentValidation > WSDLs directory, and select ValidatePayment-concrete.wsdl.
       - Click **OK**.

       Back in the wizard, you see the fields populated with the wsdl file and port.

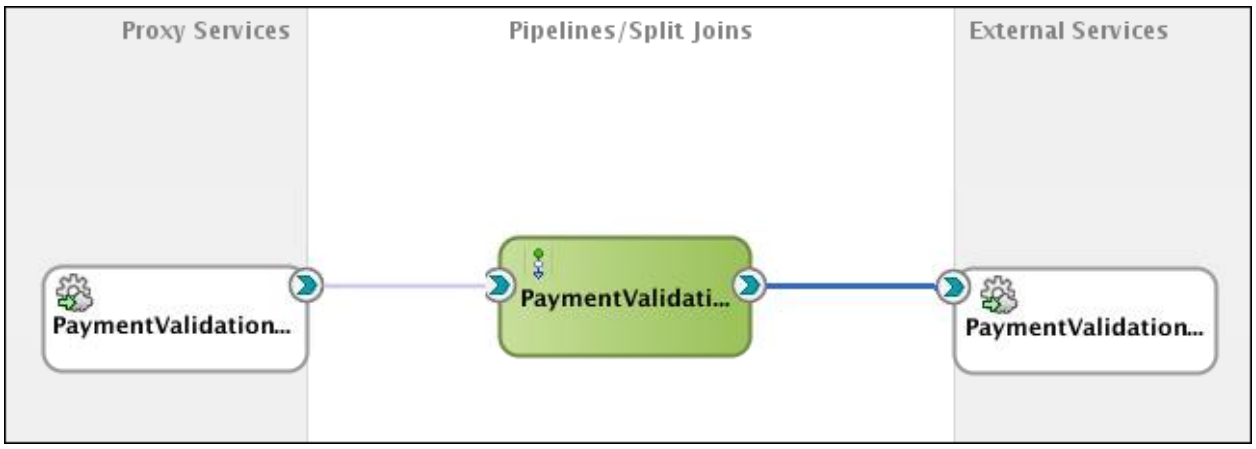


Click **Next**.

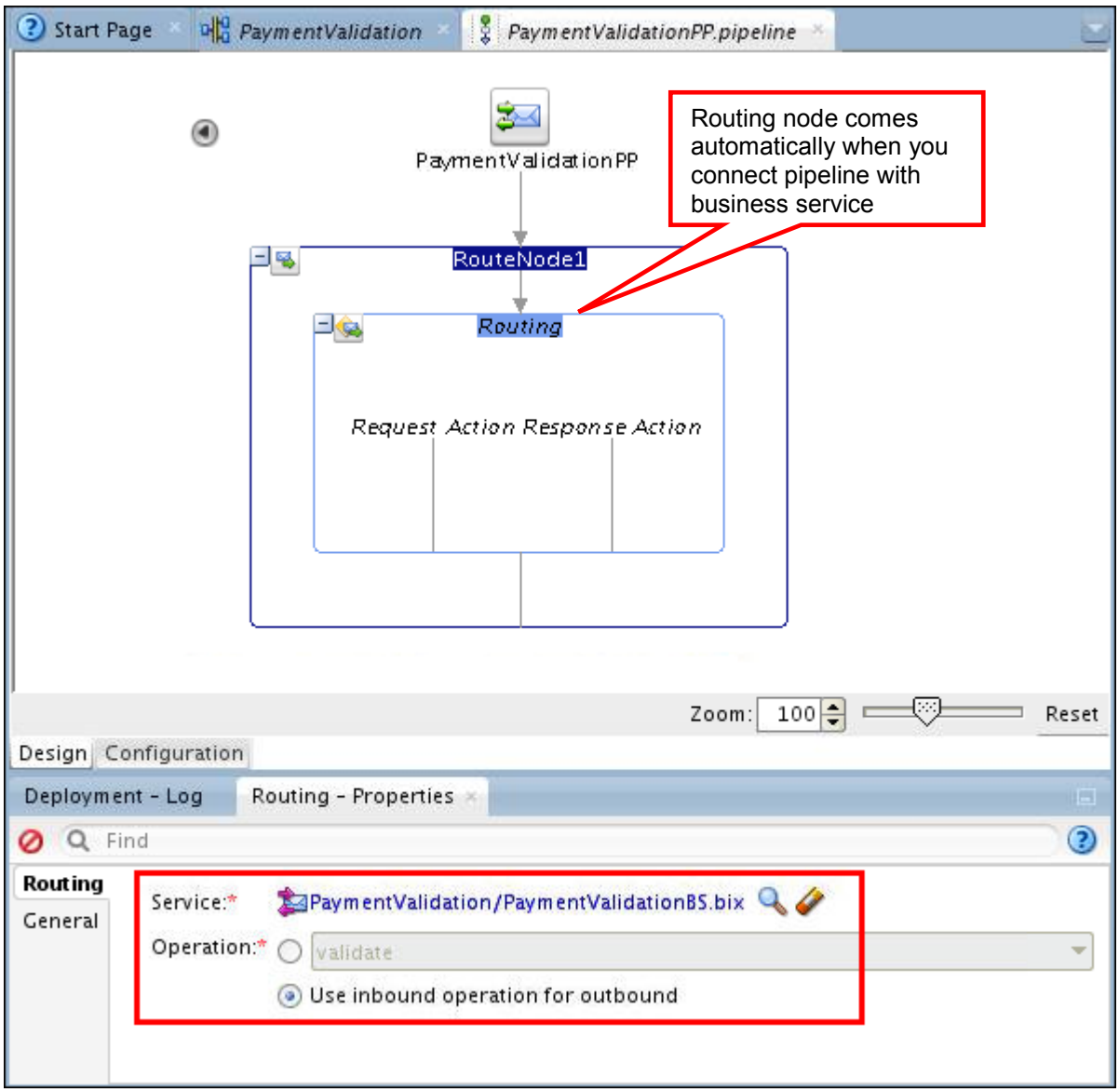
d. In step three (Transport), you should see the populated endpoint URI depending on your configuration in previous steps. Keep it as is, and click **Finish**.

Now you should see proxy and pipeline components in the Service Bus Overview Editor and their files (PaymentValidationPS.proxy, PaymentValidationPP.pipeline) added in the project folder.

18. To wire the pipeline to invoke the business service, select `PaymentValidationPP` and drag the arrow to `PaymentValidationBS`.



19. The Routing action is automatically configured in the pipeline. You can double-click the pipeline to open its editor:



You can add some actions to the pipeline, such as validation, transformation and so on. These will be covered in the subsequent labs.

20. Save your work.

Deploy and Test

Now you are ready to deploy and test end-to-end. This time you will use Service Bus console.

21. To deploy:
    a. In Application Navigator, right click PaymentValidation project and select Deploy as shown below:

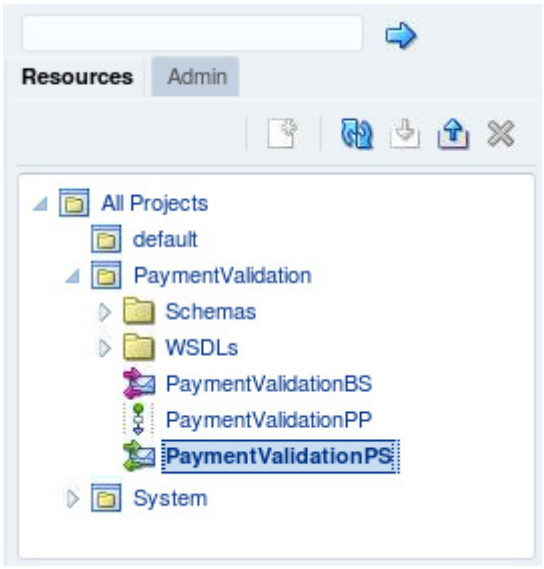
    The Deploy wizard window opens.
    b. For Deployment Action, select **Deploy to Service Bus Server**, and click **Next**.
    c. For Select Server, select **IntegratedWeblogicServer** and click **Next**.
    d. Review the summary and click **Finish**.

    In the Deployment log, at the bottom of JDeveloper, you should see the deployment finished without errors.

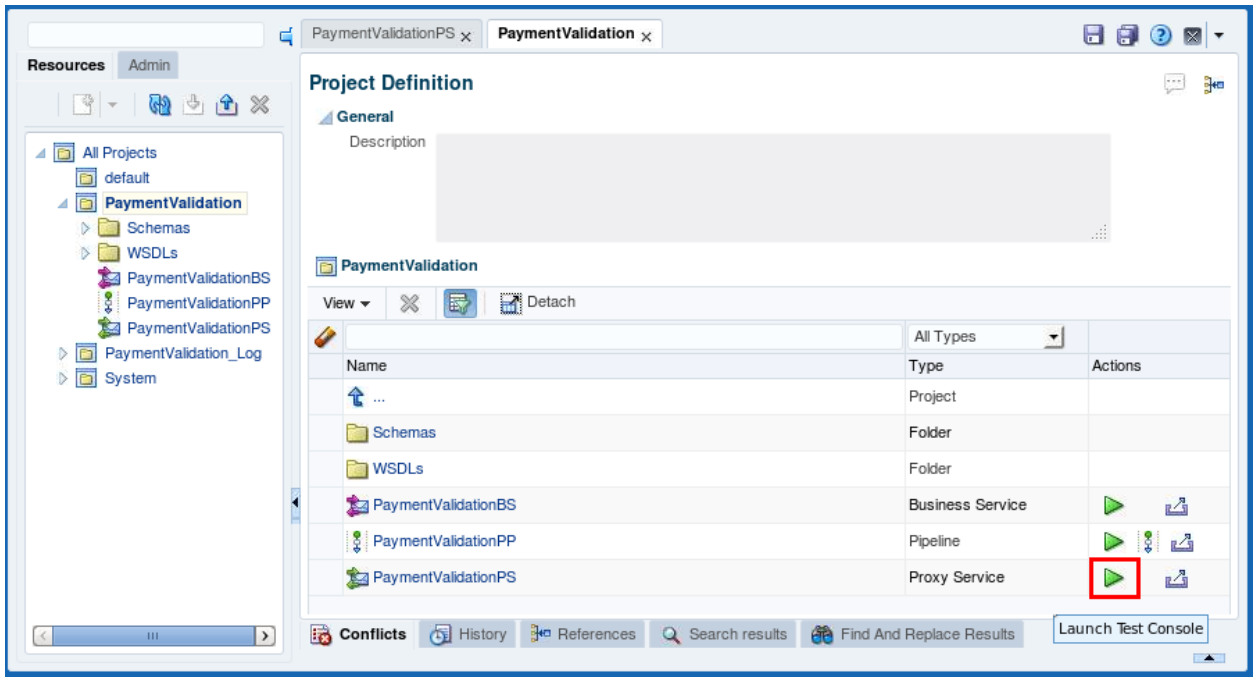
22. Test in Service Bus console
    a. In a Firefox browser, access Service Bus console using the URL:

    http://localhost:7101/sbconsole

    Once logged in, Service Bus Control is displayed with all projects, folders, and resources in the Project Navigator in a tree view on the left.
    b. Expand the PaymentValidation project, you should see a folder structure similar to the one in JDeveloper..



c. Select PaymentValidation to view its definition on the right.



d. Click the Launch Test Console button next to the proxy service.

The Test Console opens.

e. Click Browse button next to the Payload field.

The File Upload window opens.

f. Select File System, navigate to /home/oracle/labs_DI/resources/sample_input/, but this time, for payload, use the **`PaymentInfoSample_Denied.xml`** file in the folder.

g. On the Test Console, click the **Execute** button.

The next screen in the Response Document section indicates that the payment has been denied.

h. Close the test console window.

23. Verify the endpoint URI

a. In Service Bus console, select PaymentValidationPS, click Transport tab on the Proxy Service Definition page.

The Endpoint URI is what service consumers will use to access your PaymentValidation service.

b. To verify:

1) Copy Endpoint URI on the page.

2) Open another tab in Firefox browser, and type the following in the URL:

`http://localhost:7101/<paste_Endpoint_URI_here>?wsdl`

The wsdl of the PaymentValidationPS proxy service is displayed.

```
−<WL5G3N0:definitions targetNamespace="http://www.oracle.com/ValidatePayment">
  −<WL5G3N0:types>
    −<xsd:schema>
        <xsd:import namespace="http://www.oracle.com/soasample"/>
        <xsd:import="http://www.oracle.com/soasample" schemaLocation="http://localhost:7101
        /ValidatePayment/ValidatePS?SCHEMA%2FValidatePayment%2FSchemas%2FCanonicalOrder"/>
      </xsd:schema>
    </WL5G3N0:types>
  −<WL5G3N0:message name="validateInput">
      <WL5G3N0:part element="WL5G3N1:PaymentInfo" name="paymentInfo"/>
    </WL5G3N0:message>
  −<WL5G3N0:message name="validateOutput">
      <WL5G3N0:part element="WL5G3N1:PaymentStatus" name="status"/>
    </WL5G3N0:message>
  −<WL5G3N0:portType name="validatePaymentPortType">
    −<WL5G3N0:operation name="validate">
        <WL5G3N0:input message="WL5G3N2:validateInput"/>
        <WL5G3N0:output message="WL5G3N2:validateOutput"/>
      </WL5G3N0:operation>
    </WL5G3N0:portType>
  −<WL5G3N0:binding name="validatePaymentBinding" type="WL5G3N2:validatePaymentPortType">
      <WL5G3N3:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    −<WL5G3N0:operation name="validate">
        <WL5G3N3:operation soapAction="validate" style="document"/>
      −<WL5G3N0:input>
          <WL5G3N3:body use="literal"/>
        </WL5G3N0:input>
      −<WL5G3N0:output>
          <WL5G3N3:body use="literal"/>
```

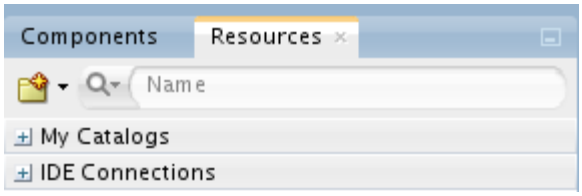
Practice 3-3: Populating MDS Repository with Source Data

Overview

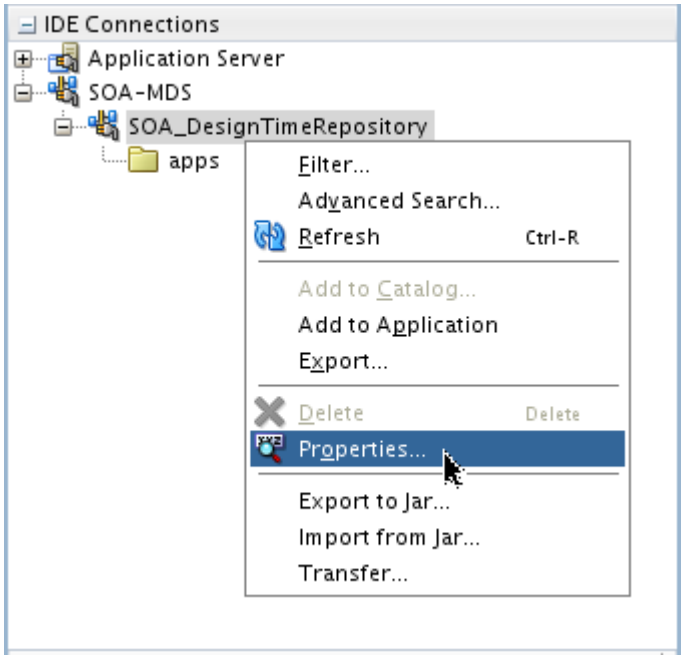
With practice, you learn how to share design time artifacts such as WSDL, XSD and XQuery files of PaymentValidation with other composite applications by using the MDS repository.

Viewing the default MDS repository configuration

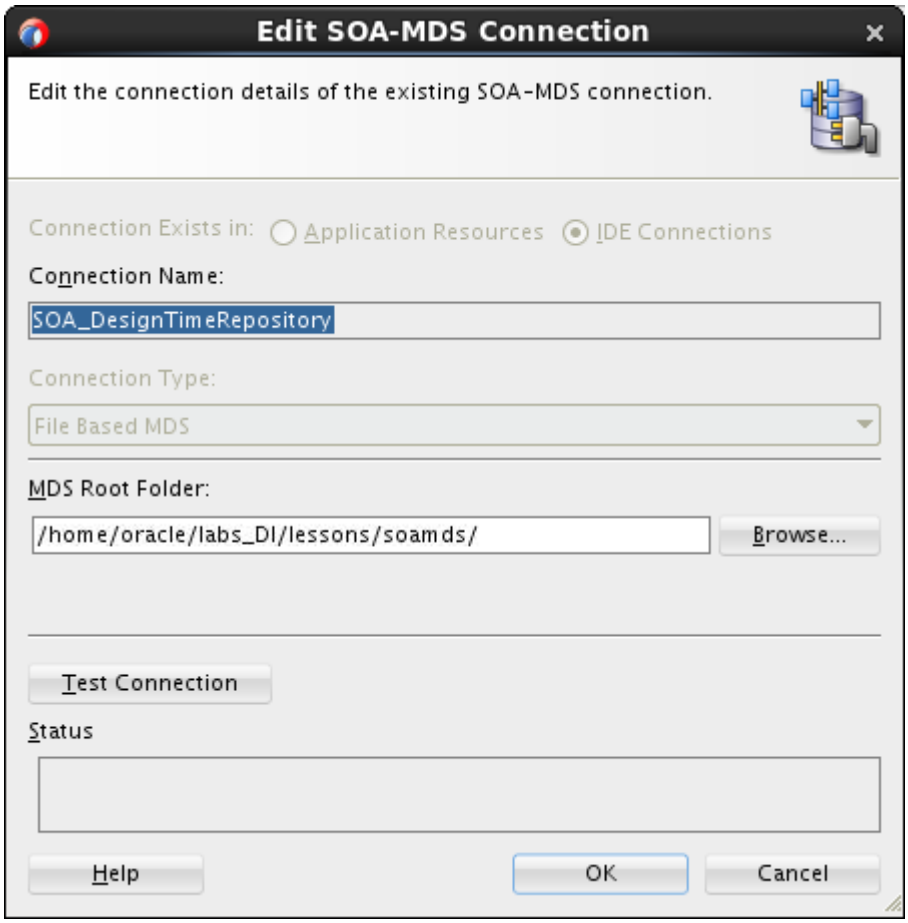
1. Go to JDeveloper and from the Window main menu, select **Resources**.
2. In the Resources window, expand IDE Connections.



3. In the IDE Connections panel, expand SOA-MDS. You should see the automatically created (when creating a SOA composite) connection named `SOA_DesignTimeRepository` and the `/apps` folder in the repository. At this point, the folder is empty.
4. Right-click the SOA_DesignTimeRepository connection, and select **Properties**.



The Edit SOA-MDS Connection dialog box is displayed. It resembles the following image:



The default SOA-MDS connection name is `SOA_DesignTimeRepository`. The default repository root folder is `/home/oracle/labs_DI/lessons/soamds/`.

5. Click **Browse**. The Choose Directory dialog box is displayed. You can see that there is an `apps` folder in the default repository root folder. This folder is automatically created.

   The connection name cannot be changed, but the default repository location can be modified to point to another folder or source control location.

   Leave this dialog box open; you will use it shortly.

Customizing the MDS repository folder

6. Use `/home/oracle/labs_DI` as the repository root folder, and add the `/apps` folder and a folder for PaymentValidation. Perform the following steps:

   a. Create the `apps` folder.

      Open a terminal window, and enter the following commands:

      ```
      >cd $HOME/labs_DI
      >mkdir apps
      ```

   b. Create a folder for PaymentValidation to share its artifacts.

      ```
      >cd apps
      >mkdir PaymentValidation
      ```

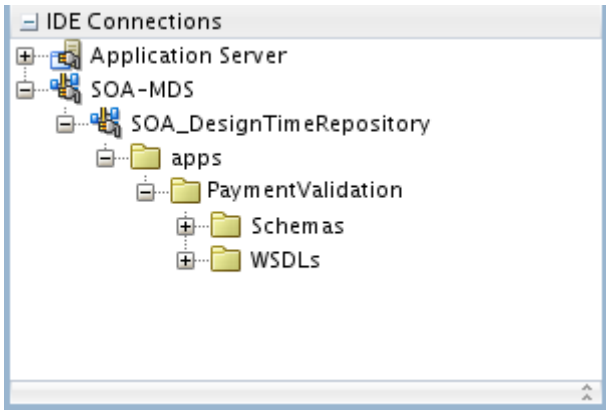
c. Copy the following two folders from `$HOME/labs_DI/resources/` into the share data folder: `/apps/PaymentValidation`:

- `Schemas`
- `WSDLs`

The copy command is:

```
>cd $HOME/labs_DI/resources/
>cp -r Schemas/ WSDLs/ $HOME/labs_DI/apps/PaymentValidation/
```

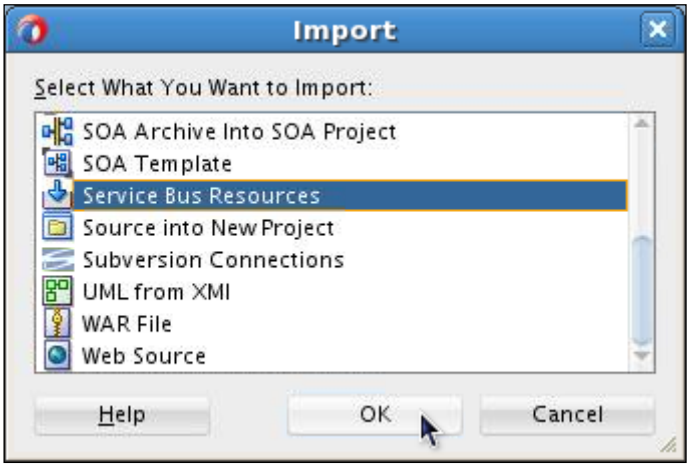
7. Go back to JDeveloper and, in the Choose Directory dialog box that you opened earlier, select `$HOME/labs_DI` folder, and click OK.
8. The Resources window is updated with the new MDS repository content and the artifacts from the resource folder.



9. To verify:
   a. In Application Navigator, right-click the WSDLs folder, and select **Import** from the context menu.

      The Import dialog box is displayed.

   b. Select **Service Bus Resources**.

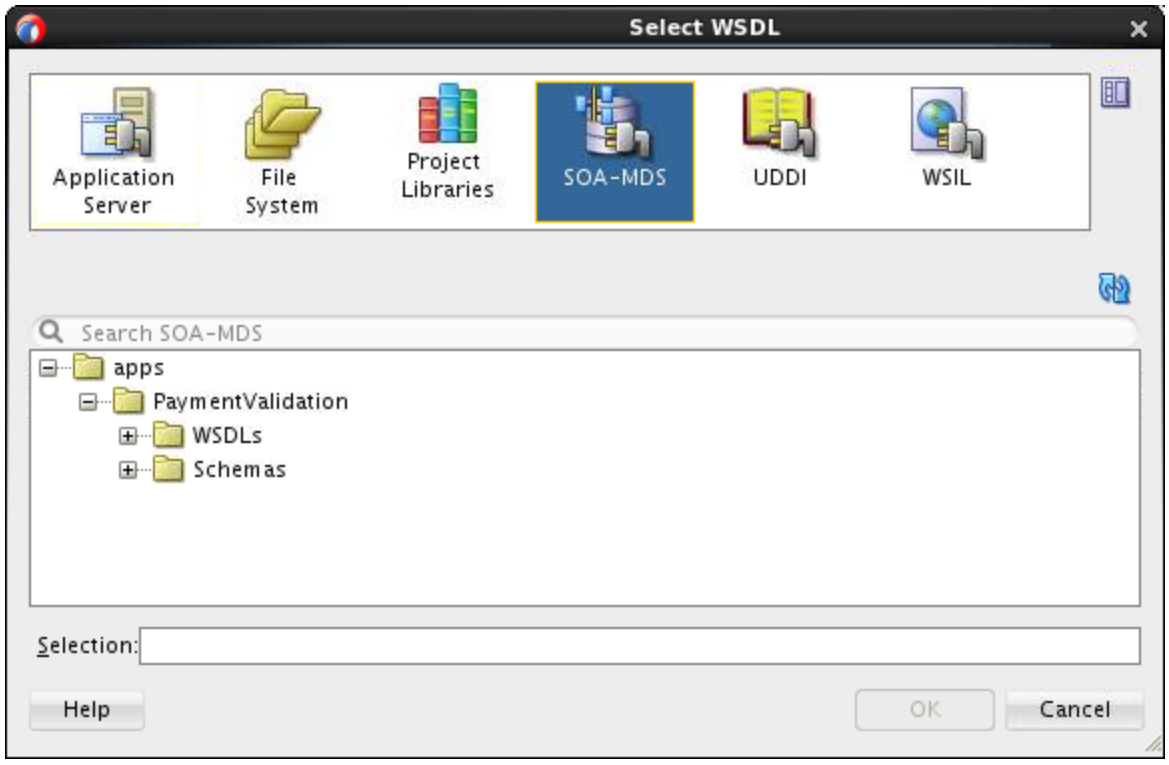


   c. Click **OK**.

      The Import Service Bus resources wizard is displayed. The title bar of the wizard dialog box shows the step number.

   d. For Type, select **Resources from URL**, and click **Next**.
   e. For Source, click the browse button next to the Source URL field.

f. In the Select WSDL window, select **SOA-MDS** and expand the apps folder.



You should see the artifacts that you copied into the MDS Repository.

g. Click Cancel for all the open windows as you already selected File System as the source of the WSDL files.

10. When you are done, close the application and all open windows of this project.

Practices for Lesson 4: Basics of Message Flow

Chapter 4

Practices for Lesson 4: Overview

Practices Overview

The pipeline template encapsulates all the repetitive tasks common with building a pipeline such as routing, data validation, reporting, error handling, and alerting under error conditions. This feature in Service Bus 12*c* enables you to template certain pieces of code and re-use them in other projects. In these practices, you will create a log pipeline template, and use it to implement a concrete pipeline.

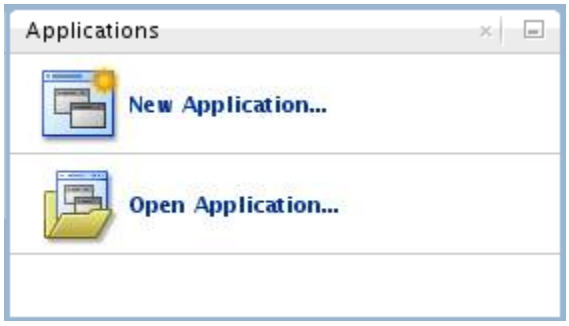
Practice 4-1: Creating a pipeline template for logging

Overview

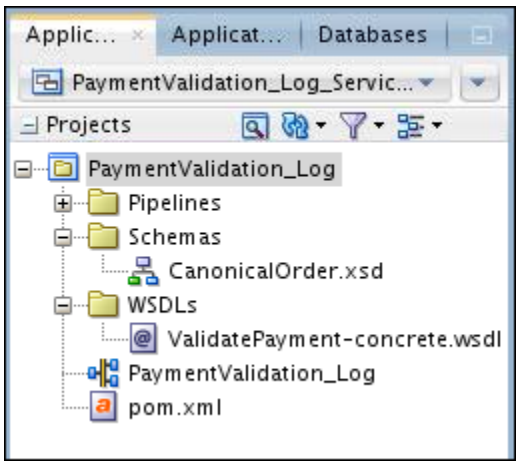
In this practice, you will create a pipeline template that defines the general shape of the logging message flow and then generate concrete message flow using this pipeline template.

Tasks

1. Open JDeveloper if it is not open. (You can use the JDeveloper shortcut on the desktop.)
2. In the Applications pane, click **Open Application** (or you can select **File** > **Open**).



3. In the Open Application(s) dialog box, navigate to the `~/labs_DI/lessons/lesson04/PaymentValidation_Log_ServiceBusApp/` directory, and open the `PaymentValidation_Log_ServiceBusApp.jws` file.
4. Expand the folders in Application Navigator, and you should see the folder structure resembling the image below:



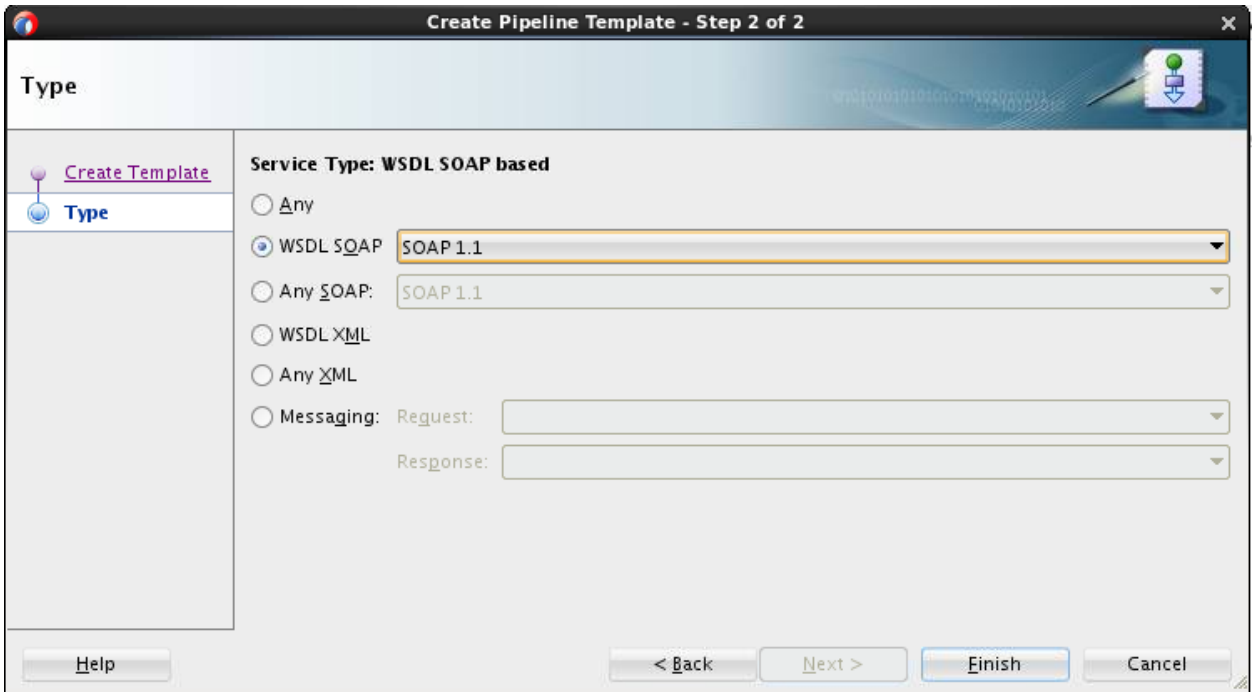
To save your time, the application and project are pre-created and artifacts are imported into the folders for you.

Create a pipeline template

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

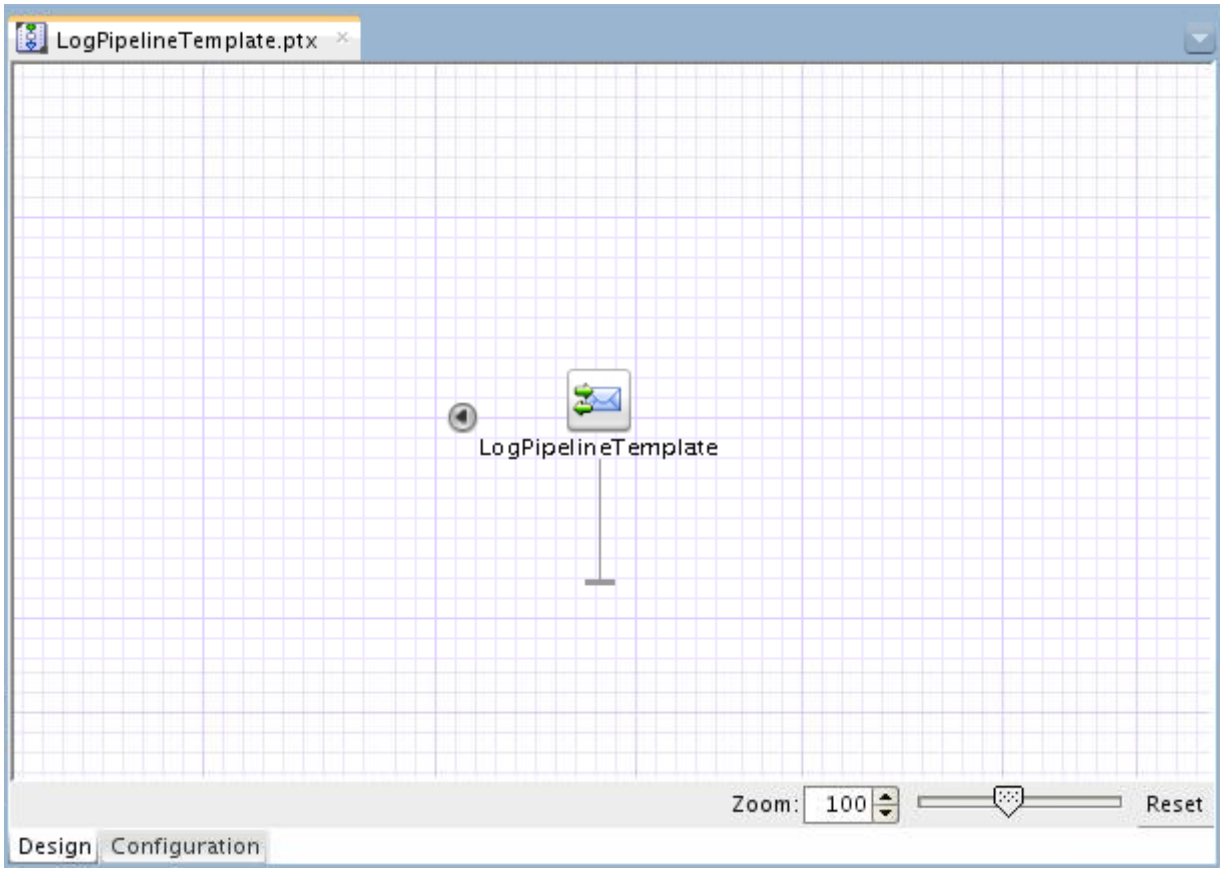
   The Create Pipeline Template dialog window opens.
6. In the Create Template screen, give the name as **LogPipelineTemplate**, and accept the default location.
7. Click **Next**.

8. In the Type screen, select WSDL SOAP as service type.



You have to choose this option as you are going to create all of your Proxy Services based on a WSDL.

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

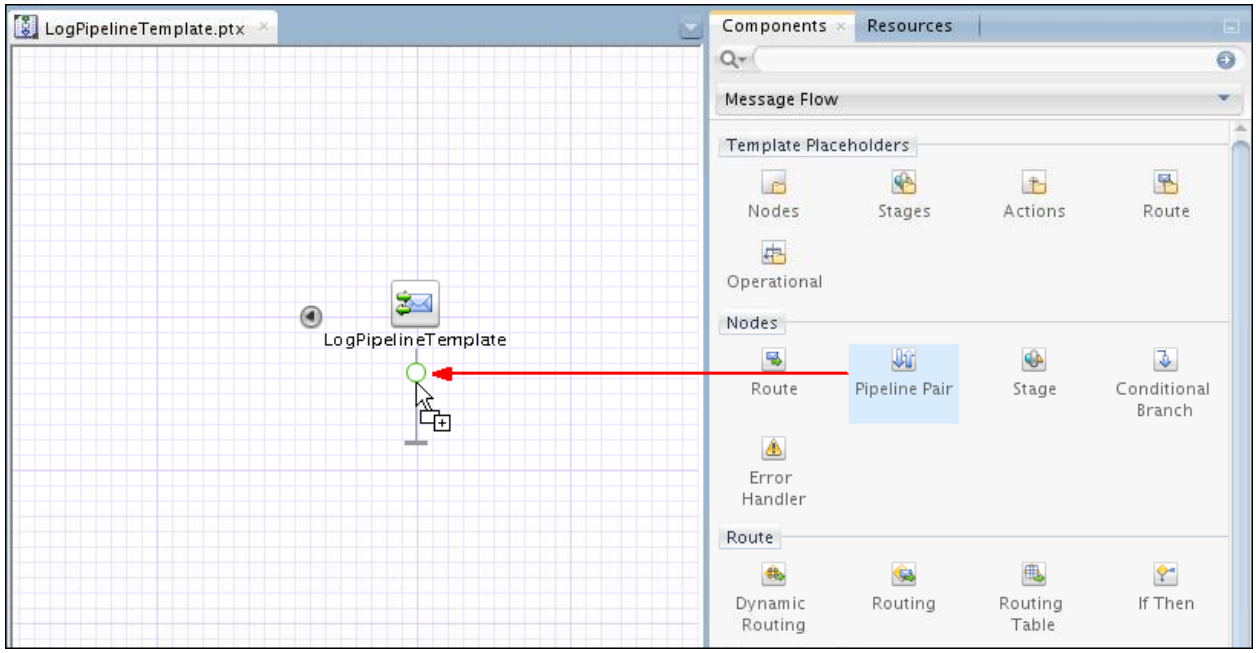


You should see a message flow and this message flow has only a start node. Next, you can add all required actions or nodes in the message flow that you want to have in the pipeline template.

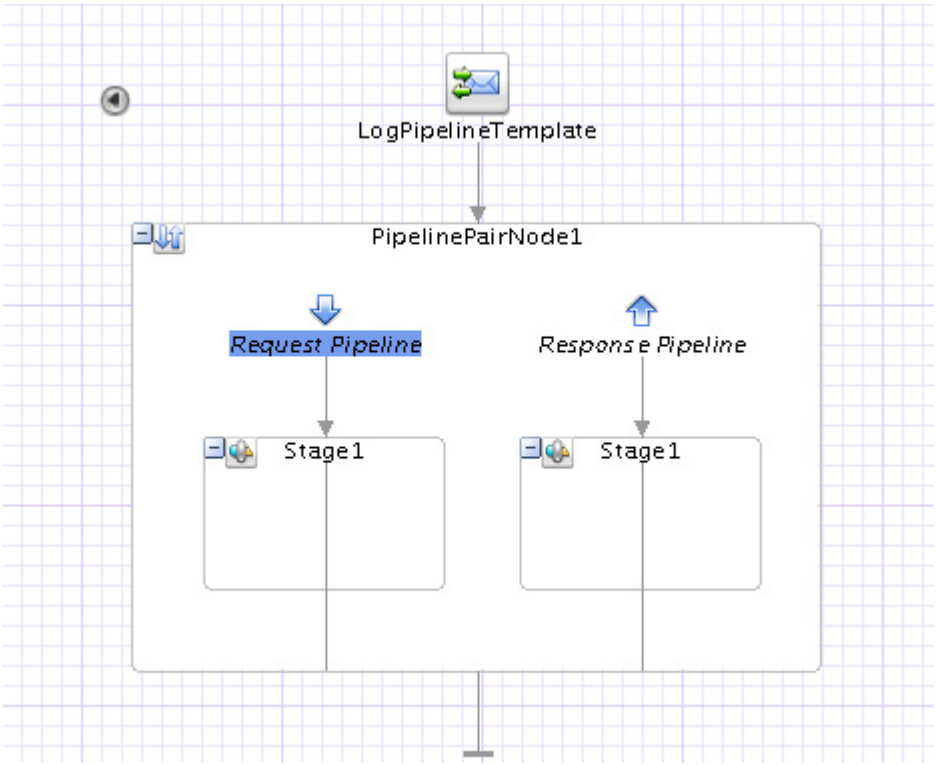
For this practice, you will log the incoming/request message payload in the log file, and then route to the target service. To do so:

10. Add the log action
    a. Drag Pipeline Pair from the Nodes section in the Components palette and drop it onto the yellow circle below the Start node in the template editor.

    Note: Yellow circles appear indicating valid places to drop the component in the pipeline.

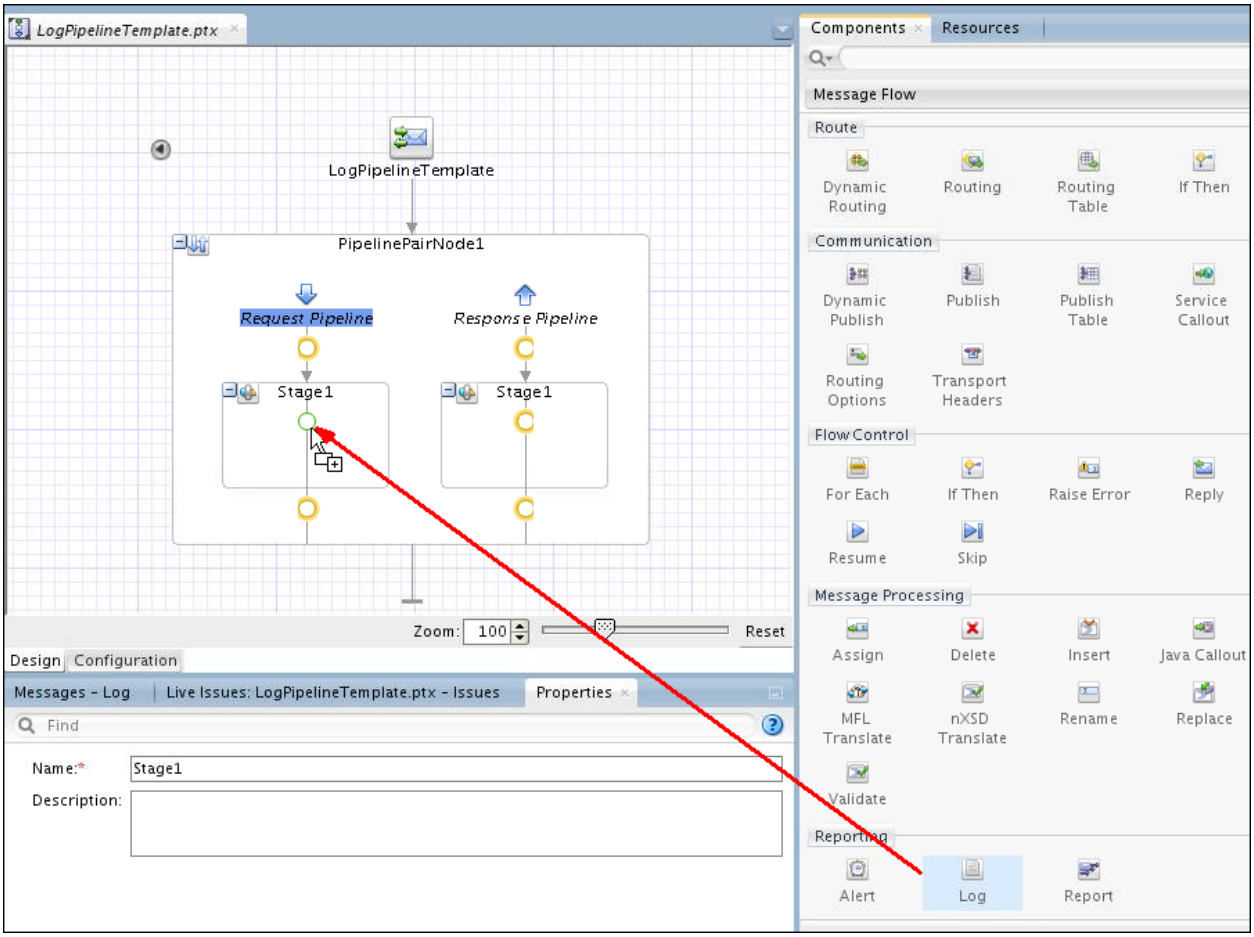


You should see the Pipeline Pair node being added to the start node as shown below. The Pipeline Pair comprises of a Request Pipeline and a Response Pipeline. The Request and Response pipelines have stages that can contain action nodes.

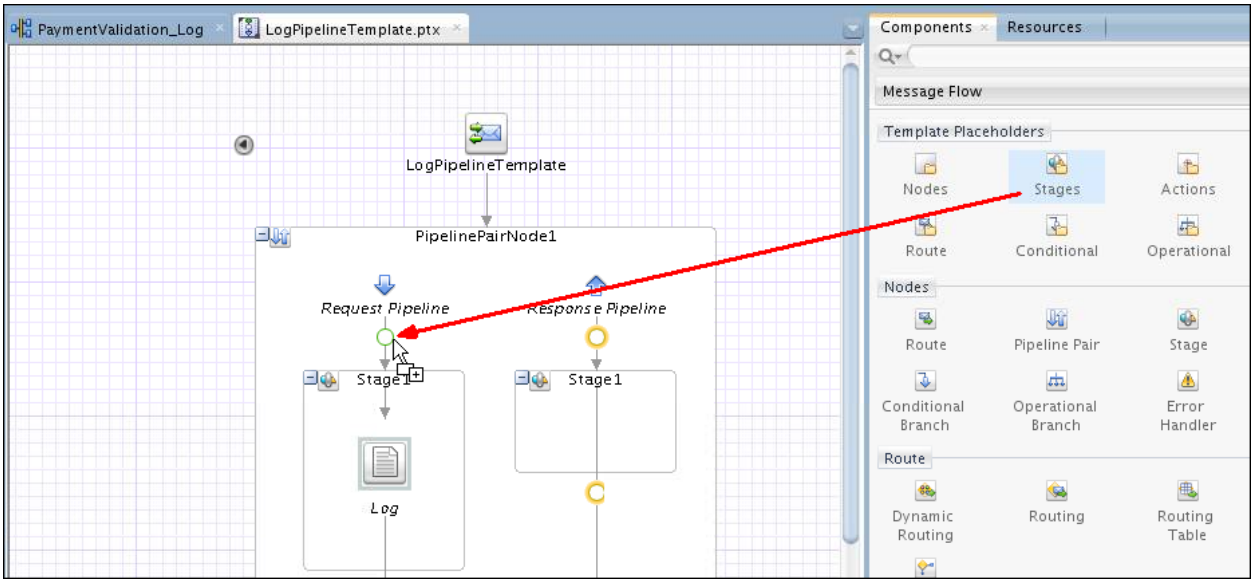


Note: When you add components to the editor, icons are displayed on the editor to represent the components. The relationships among the components are shown with lines and bounding boxes

b. Add the Log action by dragging Log activity from the Reporting section and dropping it into Stage 1 on the Request Pipeline.



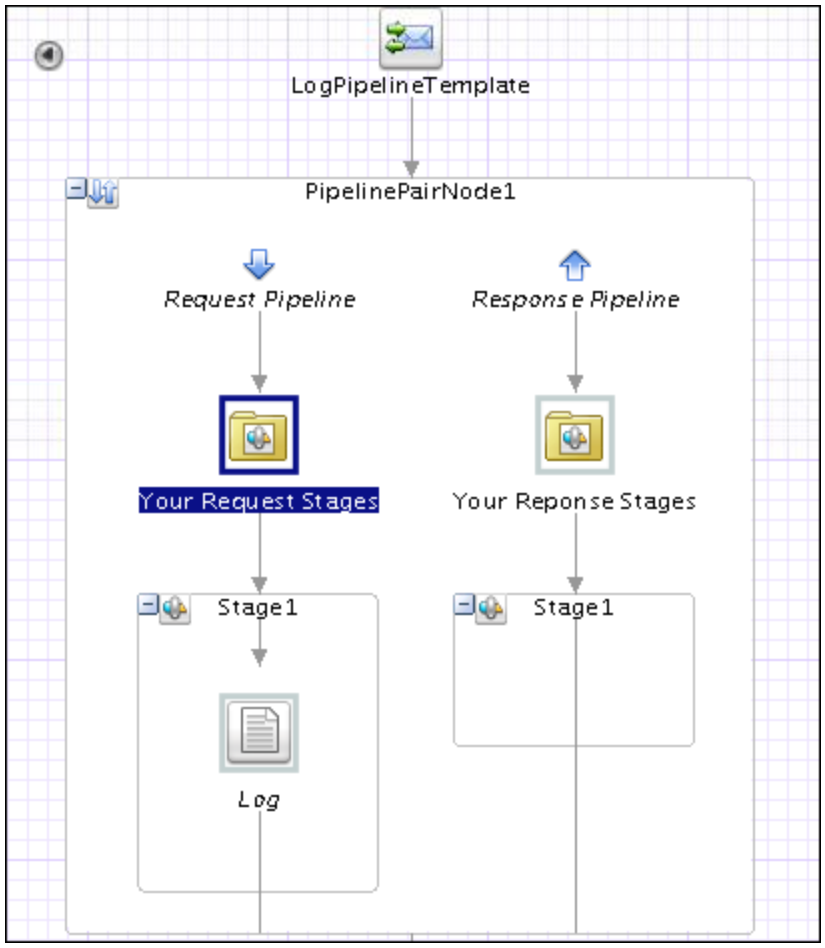
11. Add a Stages template placeholder. The placeholders allow people to customize the pipeline to do additional work such as transformations and message enrichment while creating the concrete pipeline.
    a. Drag Stages from the Template Placeholders section in the Components palette and drop it onto the yellow circle above the Stage 1 node in the Request Pipeline.



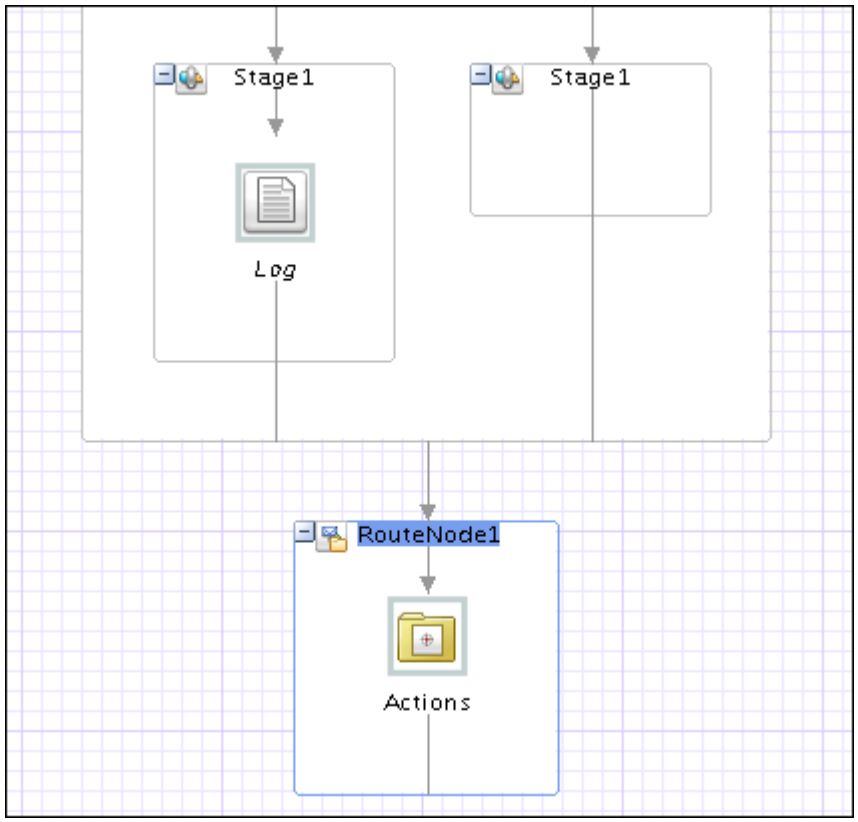
    b. Right-click the stage name and change it to **Your Request Stages**.

c. Follow the same steps, and add a Stages placeholder to the Response Pipeline, and name it **Your Response Stages**.

When you are done, your canvas should look like the image below:



12. Add a route node
    a. Drag-and-drop Route (from the Template Placeholders section) after the Pipeline Pair node.



    Routing node always depicts the end of the message flow and you cannot place any other activities after this node.
    b. Observe the Actions placeholder, which allows you to place any type of activity while designing message flow for your concrete pipelines.
13. Save your project.

    With this, you are done with pipeline template and can proceed with creation of pipelines using this template.

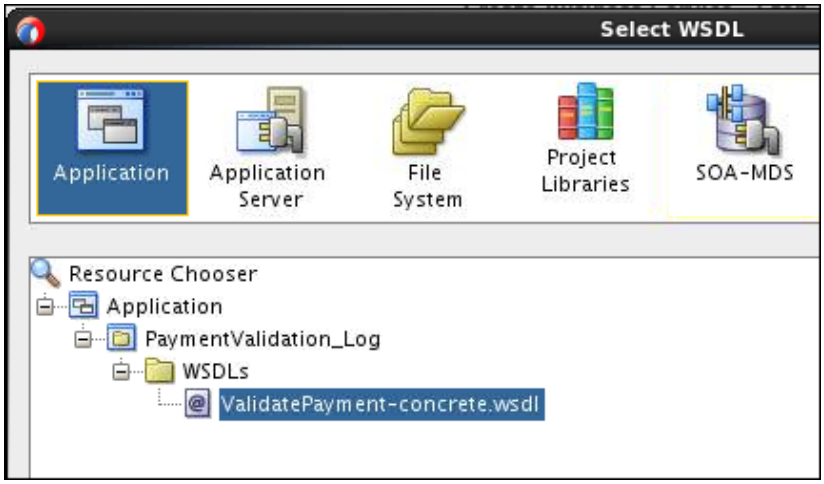
Practice 4-2: Creating a new pipeline using a pipeline template

Overview

In this practice, you create pipeline using the pipeline template and expose it as Proxy Service.

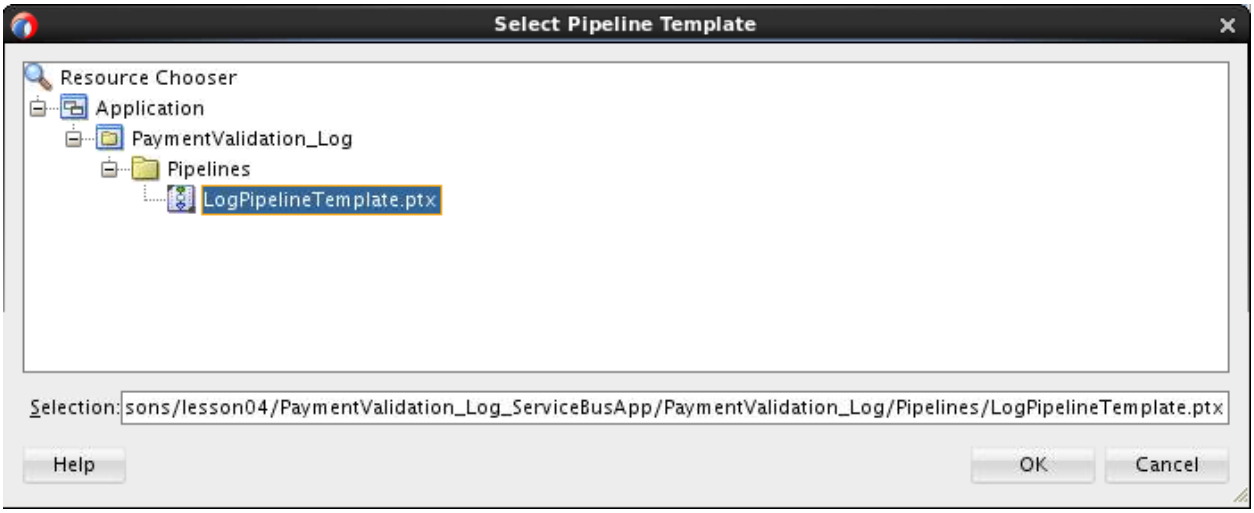
Tasks

1. Create the business service.
   a. In Application navigator, double-click PaymentValidation_Log to open it in Service Bus Overview editor.
   b. Drag the HTTP component from the Technology section in the Component palette and drop onto the External Services lane.
   c. Enter the required details for business by following the instructions in the table below:

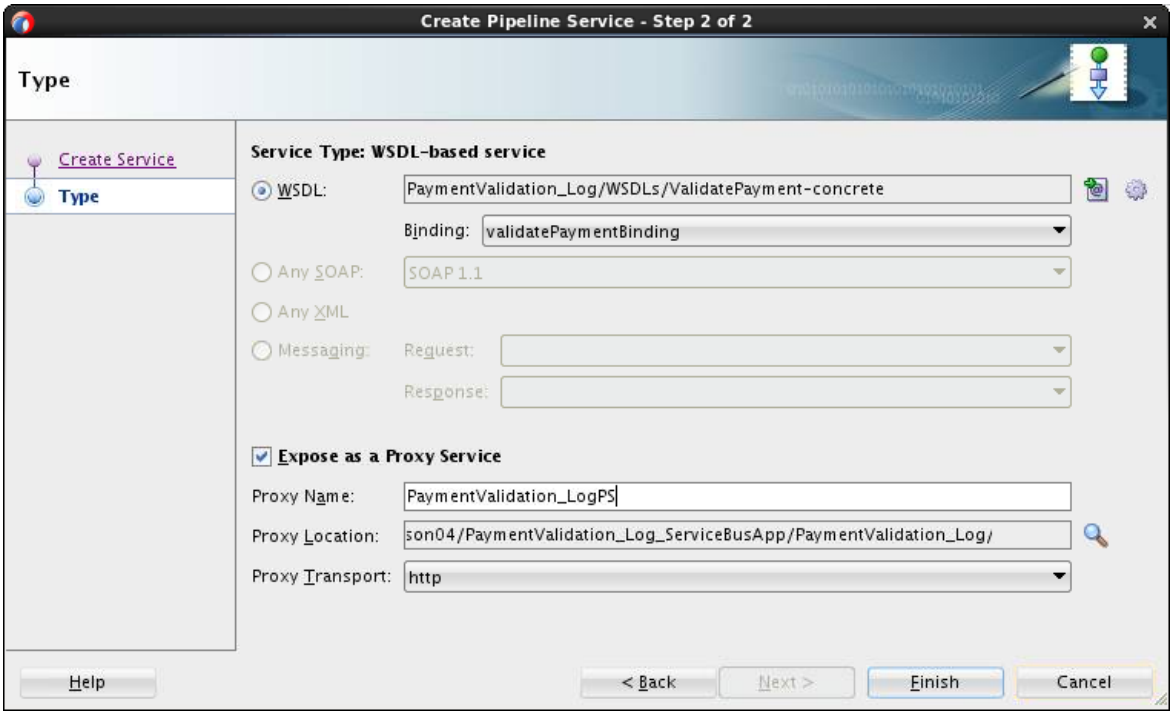
Step	Window/Page Description	Choices or Values
1)	Create Service	Name the service `PaymentValidation_LogBS`. Click Next.
2)	Type	• 1) Select **WSDL** for Service Type. • 2) Click the Browse WSDLs icon to the right of the WSDL choice. • 3) In the Select WSDL dialog box, click **Application**, navigate and select **ValidatePayment-concrete.wsdl**, as shown below:  • 4) Click **OK**. • 5) Back in the Create Business Service window, you should see the WSDL and port fields populated with the information based on the WSDL you chose. • 6) Click **Next**.
3)	Transport	Verify that http is selected in the Transport field.

Step	Window/Page Description	Choices or Values
		Confirm that Endpoint URI is pointing to the deployed validatePayment composite. Click **Finish**.

2. Create the pipeline using the template.
   a. In Application Navigator, right click the Pipelines folder and select **New -> Pipeline**.
      The Create Pipeline Service wizard opens.
   b. Provide the name **PaymentValidation_LogPP** and select the option **From Template**.
   c. Click the Search icon.
      The Select Pipeline Template dialog window opens.
   d. Select the pipeline template created in the previous practice.



   e. Click **OK** to go back to Create Pipeline Service window, and click **Next**.
   f. In Type screen, use the same WSDL of the business service for the pipeline service:
      1) Select service type **WSDL**.
      2) Click the **Browse WSDLs** icon on the right.
      3) The Select WSDL dialog box pops up.
      4) Navigate to **Application > PaymentValidation_Log >** WSDLs directory, and select ValidatePayment-concrete.wsdl.
      5) Click **OK**.
         Back in the wizard, you should see the fields populated with the WSDL file and port.
      6) Keep "Expose as a Proxy Service" checked for creating Proxy Service along with pipeline.
      7) Change the Proxy Service name to **PaymentValidation_LogPS**.
      8) Change the Proxy Location to **~/labs_DI/lessons/lesson04/PaymentValidation_Log_ServiceBusApp/PaymentValidation_Log**.



9) Click **Finish**.

The pipeline editor is opened in a separate tab window.

g. Verify both pipeline and Proxy Service are shown in Project Explorer.

Notice that the background is grey. This indicates that you are working with a pipeline derived from a template, which inherits all placeholders, names, and properties from the pipeline template. The Template Designer has made some items editable and others not.

Edit the pipeline

You will do the following to the message in the pipeline:

- Add an assign action to define a variable.
- Use the log action to log the request payload.
- Route the message to the business service.

3. Add an Assign action.

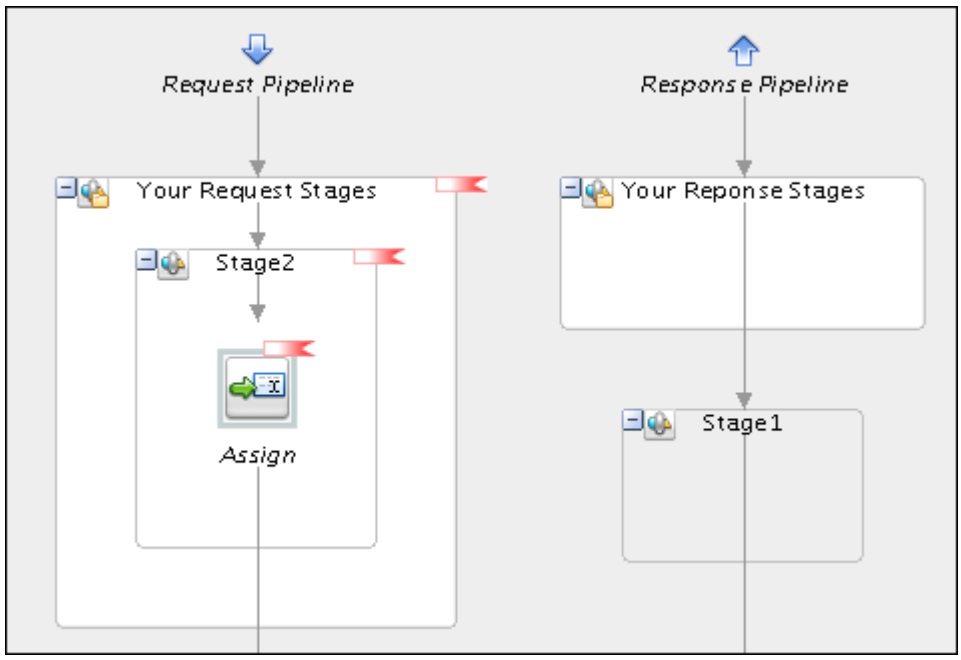
When a pipeline template defines a template placeholder of a particular type, a concrete pipeline can add zero or more nodes of the same type to the template placeholder. So here you can add stage node in this concrete pipeline because you have defined a couple of template placeholders for stages while creating the template. Each stage can, in turn, contain actions.

To add Assign action, you first need to add a stage to the stage placeholder:

a. Drag a Stage node and drop it onto the Your Request Stages placeholder.

You should see a stage node where the name Stage2 appears. This stage can contain actions.

b. Drag an Assign action (from the Message Processing section) and drop it onto Stage2 node.

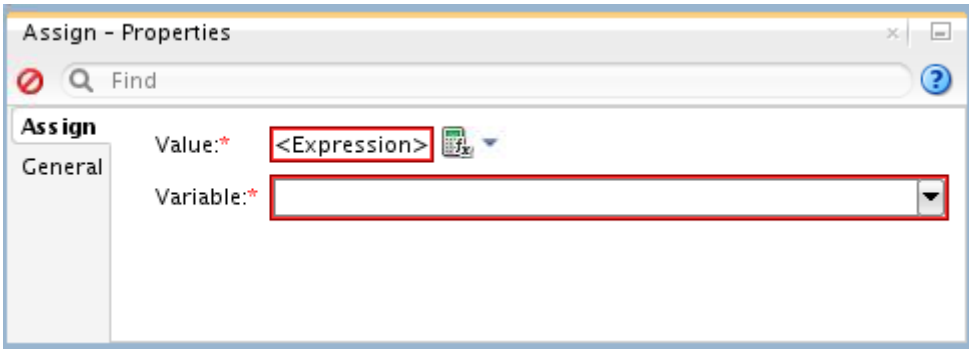


Note: Notice the red flags–these indicate areas that still need to be filled in to have a valid Pipeline. Service Bus has strong design-time validation. So it will help you keep track of when the Pipeline is completely filled out.

c. Assign the total order amount to a variable

1) Click the Assign action.

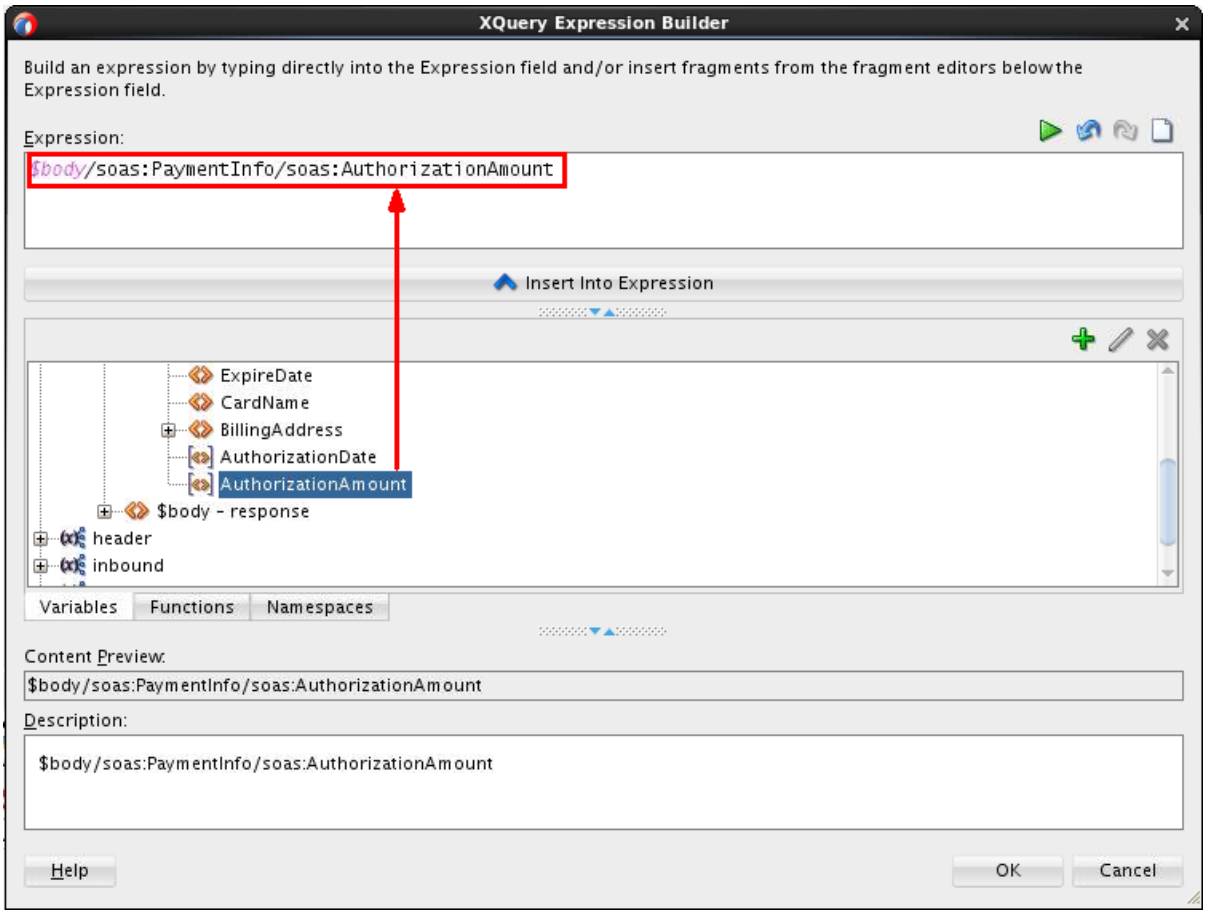
2) In the Assign-Properties window, click the XQuery Expression Builder icon to launch the XQuery Expression Builder window.



Note: If the Property inspector is not visible, you can make it appear by going to the main menu and selecting Windows->Properties. The window should appear, usually at the bottom right corner. Move to the bottom centre of your screen for easiest access.

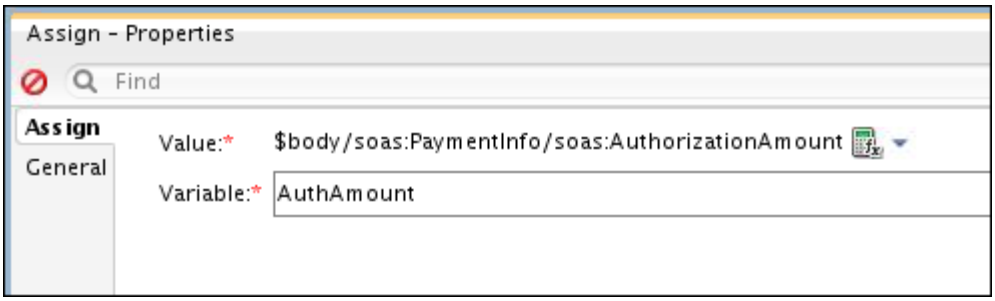
3) In the Variables tree, navigate to body > validate > $body – request > PaymentInfo, select **AuthorizationAmount**.

4) Click **Insert Into Expression** button to insert the element into the Expression field.



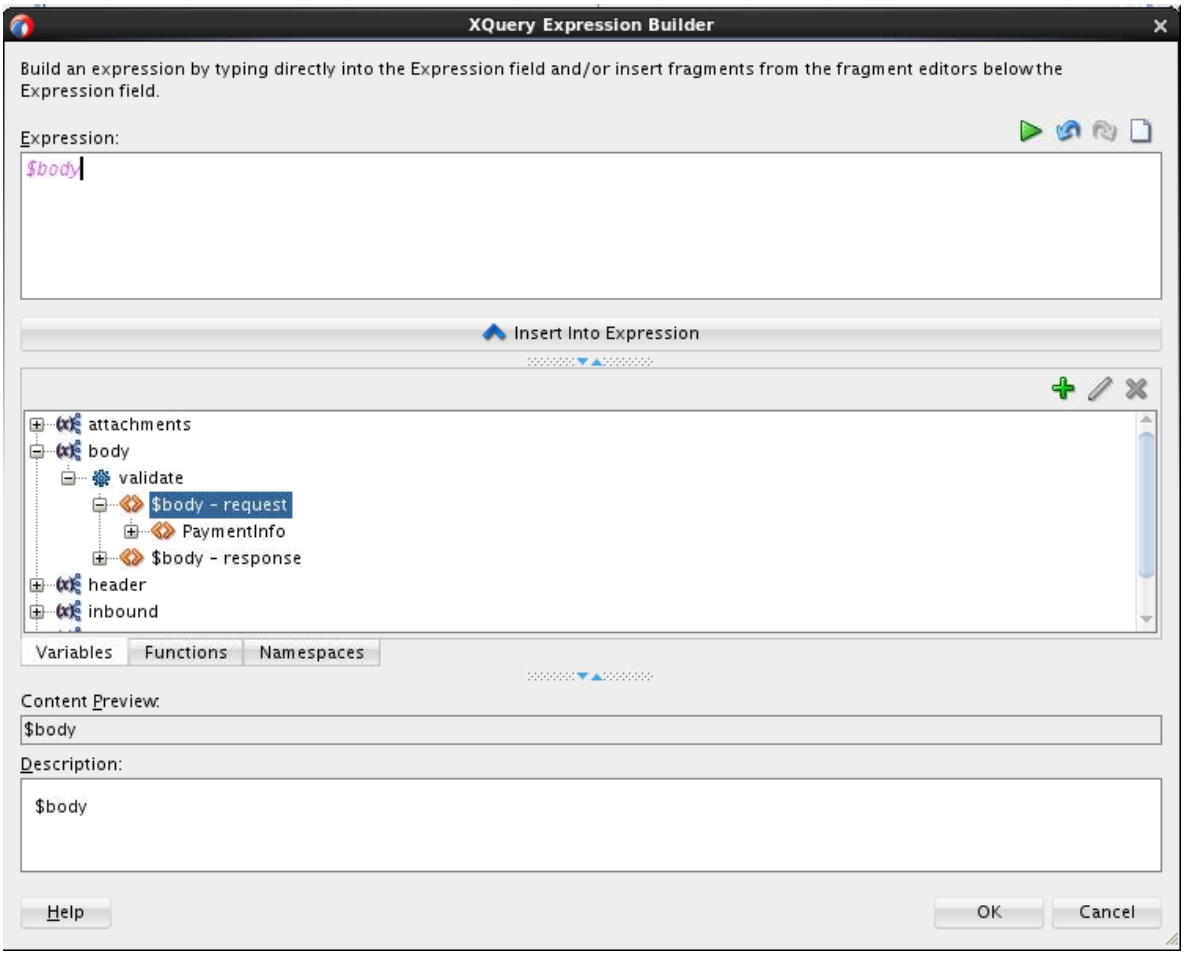
5) Click OK to close XQuery Expression Builder window

6) Back in the Assign - Properties window, enter AuthAmount as the variable name.



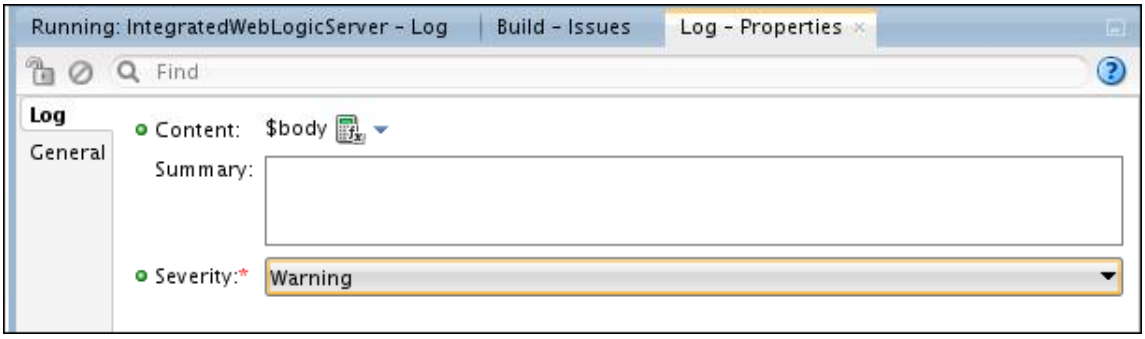
4. Define the log activity to log the request payload.
   a. Click the Log action in the Stage 1 node on the Request Pipeline.
   b. To use the incoming request payload as the log content, complete the following:
      1) Click the XQuery Expression Builder icon in the Log-Properties window to launch the XQuery Expression Builder window.
      2) Expand body > validate folder, select **$body – request**.

3) Click the **Insert Into Expression** button to insert the **$body v**ariable into the Expression field.

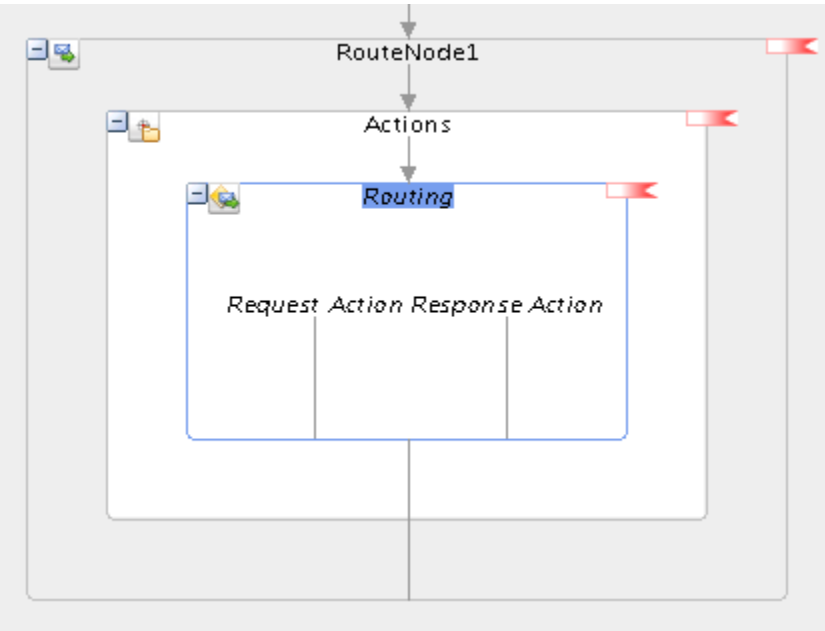


4) Click OK to close XQuery Expression Builder window.

5) Back in the Log-Properties window, select **Warning** for Severity.



5. Add the target service for the route.
   a. Drag Routing from Route section in the Component Palette and drop it onto Actions.



   b. Click Routing. In the Routing Properties window, select PaymentValidation_LogBS for service.

   c. Save the project.

Deploy and Test

6. To deploy:
   a. In Application Navigator, right click PaymentValidation_Log project and select Deploy as shown below:

   The Deploy wizard window opens.
   b. For Deployment Action, select **Deploy to Service Bus Server**, and click **Next**.

c. For Select Server, select **IntegratedWeblogicServer** and click **Next**.

d. Review the summary and click **Finish**.

In the Deployment log, at the bottom of JDeveloper, you should see the deployment finished without errors.

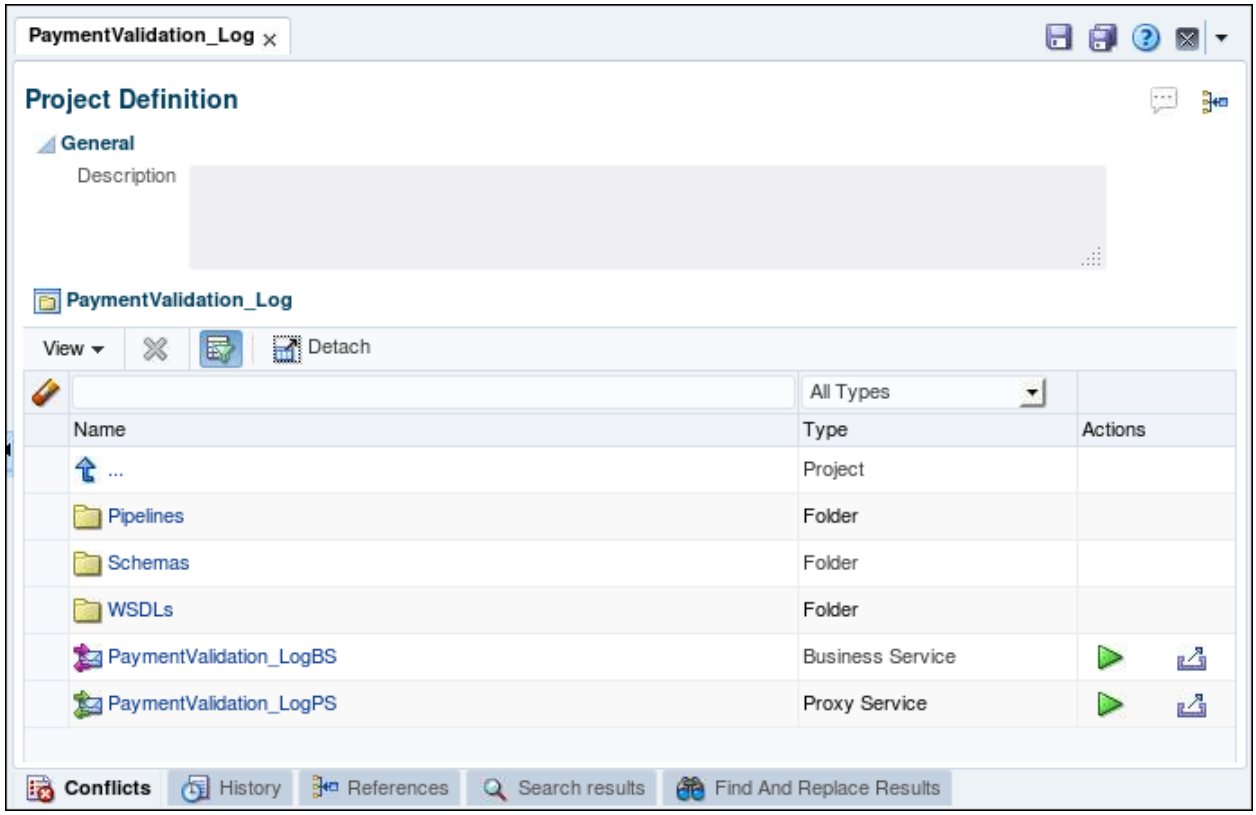
7. Test in Service Bus console

a. In a Firefox browser, access Service Bus console using the URL:

http://localhost:7101/sbconsole

Once logged in, Service Bus Control is displayed with all projects, folders, and resources in the Project Navigator in a tree view on the left.

b. In the Project Navigator, select PaymentValidation_Log to view the project definition on the right.



c. Click the Launch Test Console button (green arrow) next to the proxy service.

The Test Console opens.

d. Click Browse button next to the Payload field.

The File Upload window opens.

e. Select File System, navigate to `/home/oracle/labs_DI/resources/sample_input/`, and select **`PaymentInfoSample_Authorized.xml`**

f. Click **Open**.

g. To see the payload information written in the log file, open a terminal window, and navigate to the log file folder:

```
>cd
~/domains/DefaultDomain/system12.1.3.0.41.140521.1008/DefaultDom
ain/servers/DefaultServer/logs
```

```
>tail –f DefaultServer-diagnostic.log
```

Keep this window open.

h. Go back to Test Console, click the **Execute** button.

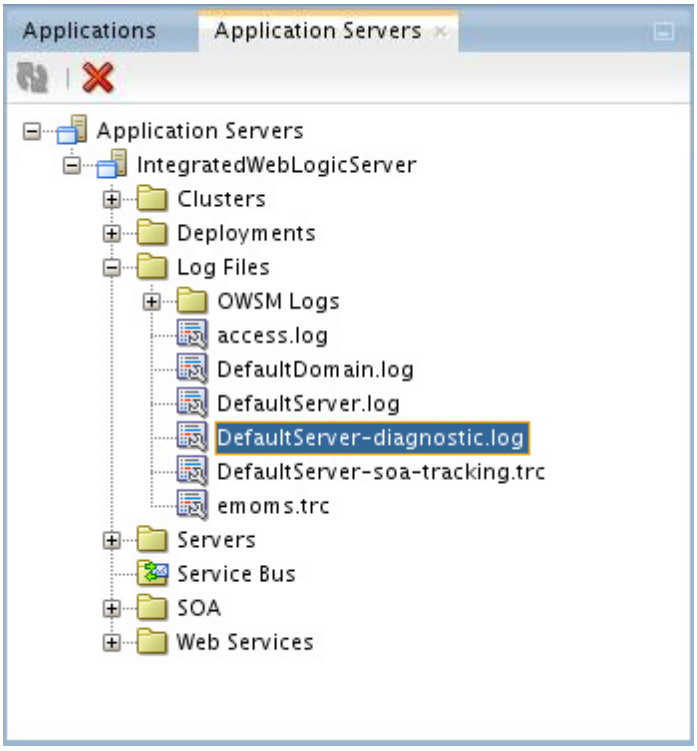
i. The next screen, in the Response Document section, indicates that the payment has been authorized.

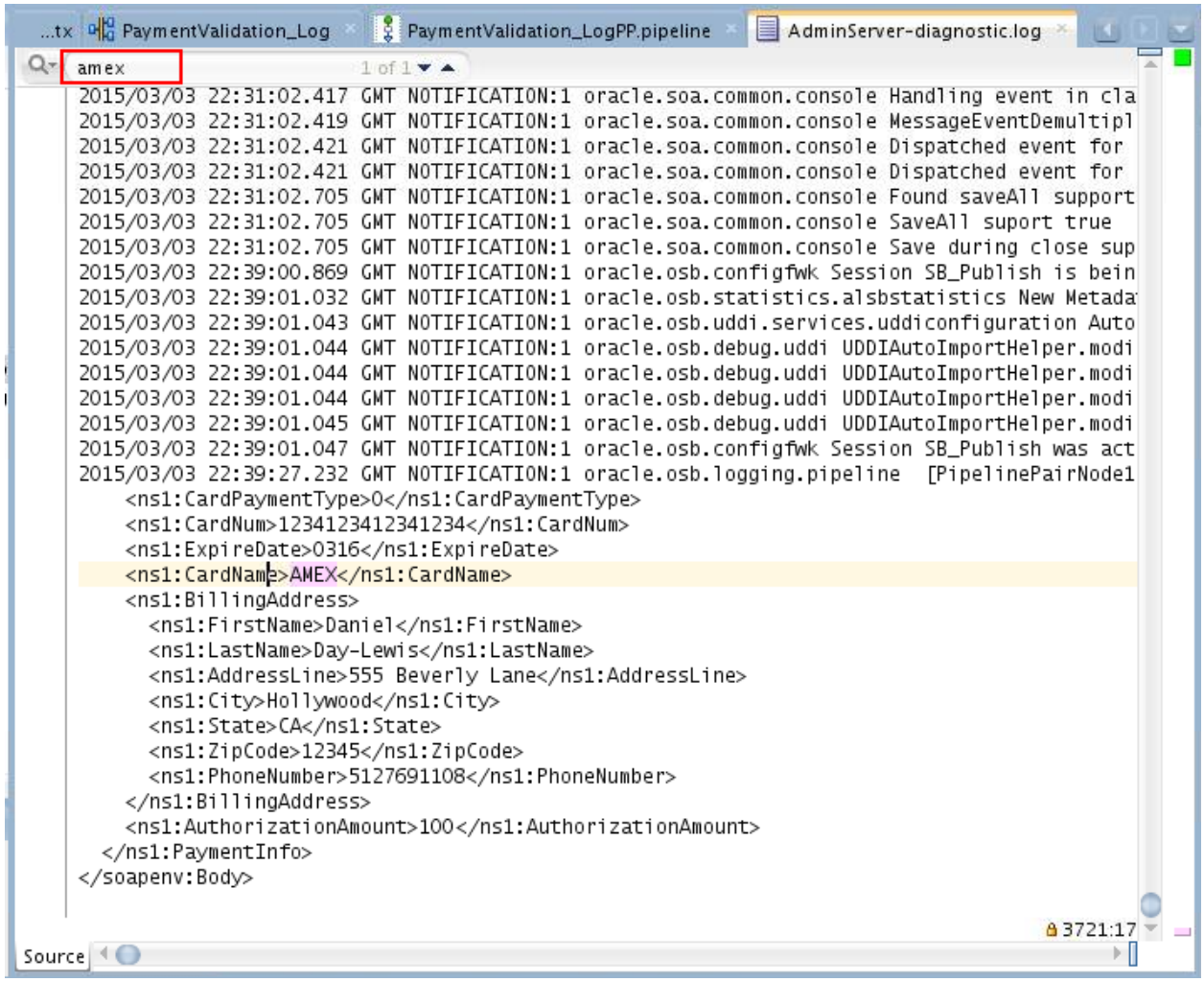
j. Check the terminal window with tail command running, you should see the payload displayed in the output as shown below:

k. Once you are done, stop the tail command (ctrl-c).

Alternatively, you can see the payload entry in the log file in JDeveloper by:

- Selecting Window > Application Server in the menu bar.
- Locate and open the DefaultServer-diagnostic.log file in the Log Files folder.





You should be able to see the payload in the log file.

Tip: Use a key word search to find the information quickly.

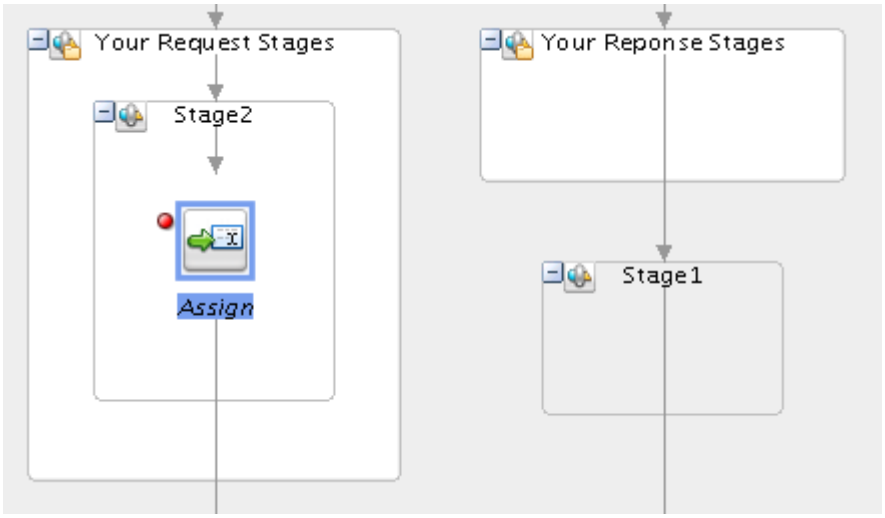
Practice 4-3: Debugging a Service Bus Application in JDeveloper

Overview

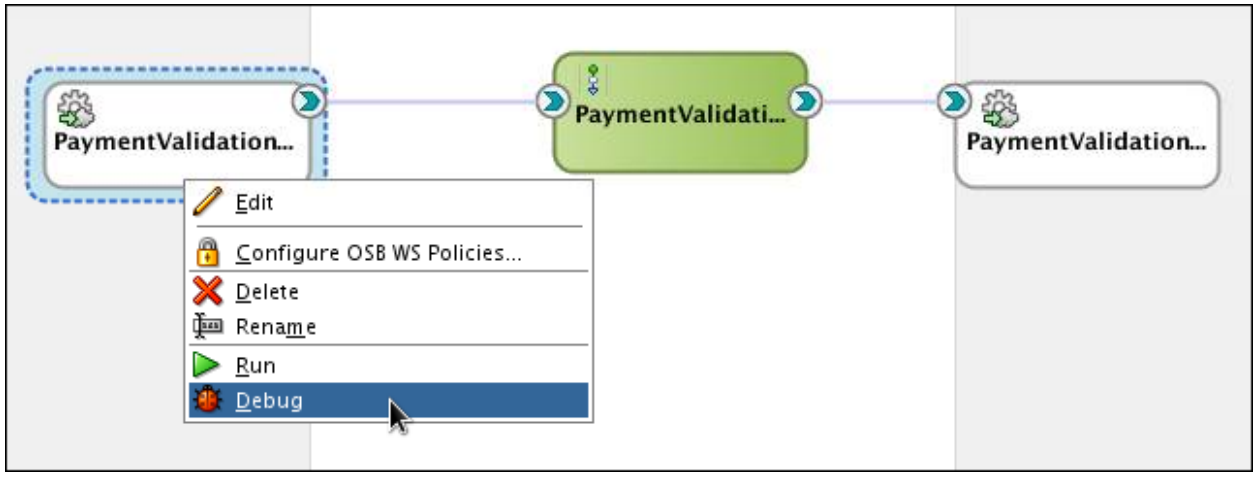
In this practice, you use the debugger to step through a running application and modify a variable value for further debugging.

Tasks

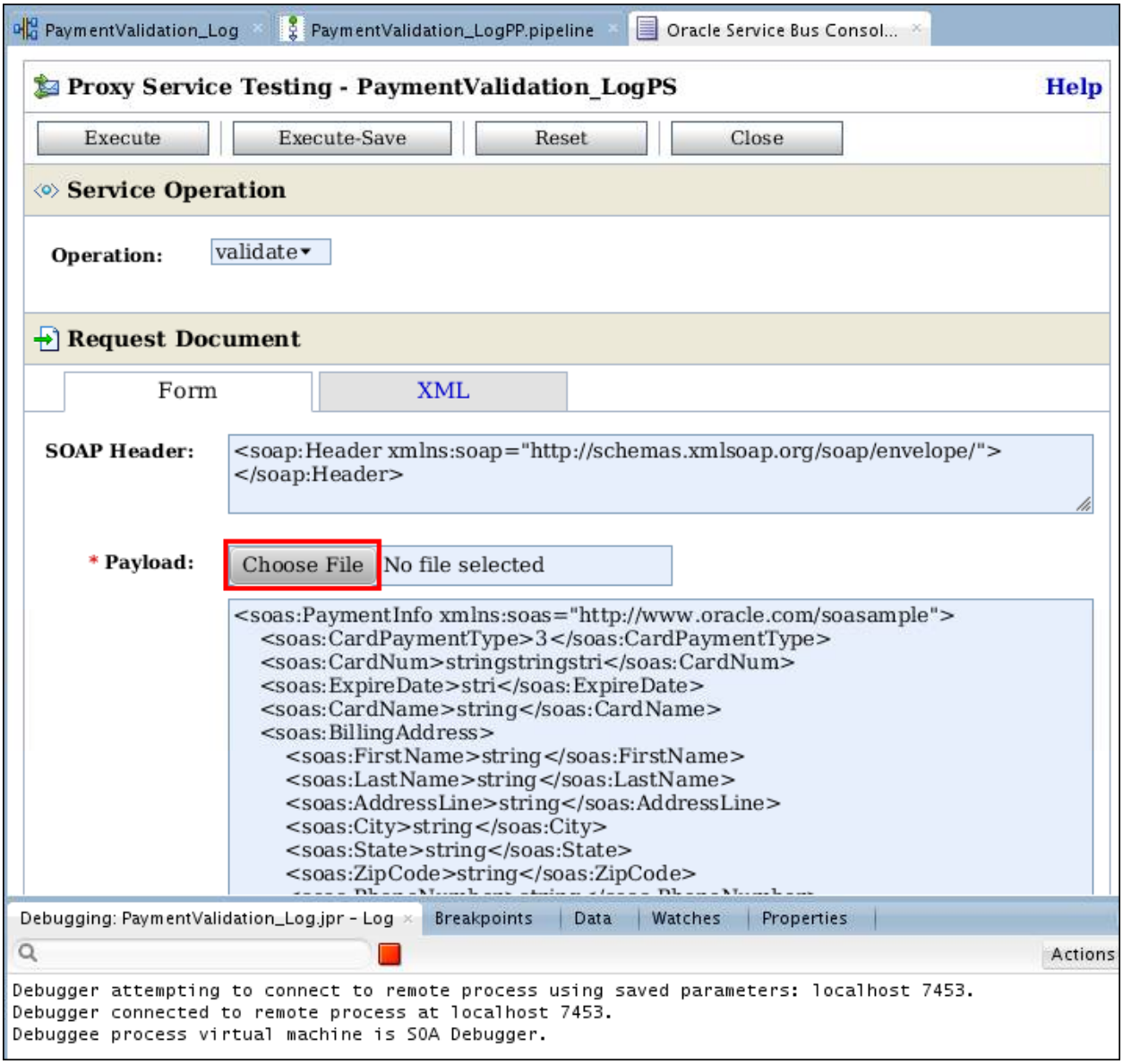
1. Start debugging.
   a. In the JDeveloper window, open the PaymentValidation_Log's pipeline in editor.
   b. Right-click the Assign action icon and select Toggle Breakpoint.



   c. In application Overview Editor, right-click the Proxy Service, and select Debug.



Test Console opens on a separate tab. Verify that the debugger is successfully connected (Checking the message in the Debugging Log window)



d. Click Choose File.

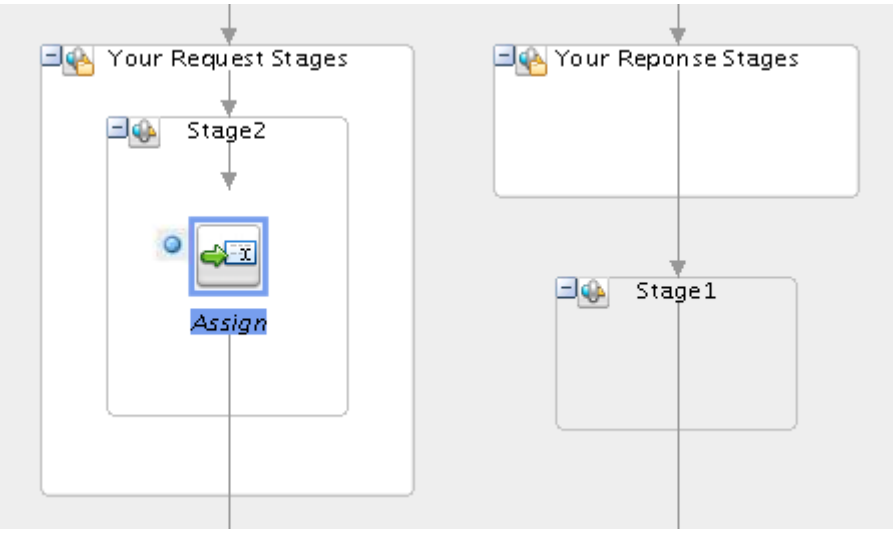
   The File Upload window opens.

e. Navigate to /home/oracle/labs_DI/resources/sample_input/, and select **PaymentInfoSample_Authorized.xml**

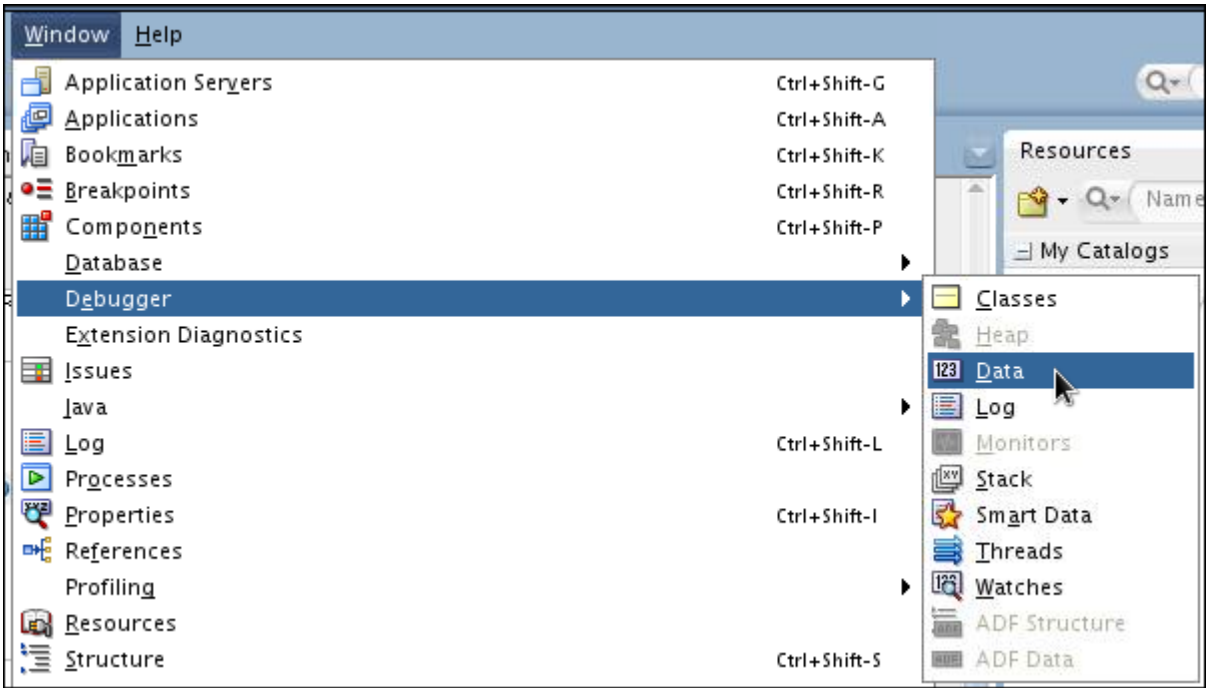
f. Click **Open**.

g. Click the **Execute** button

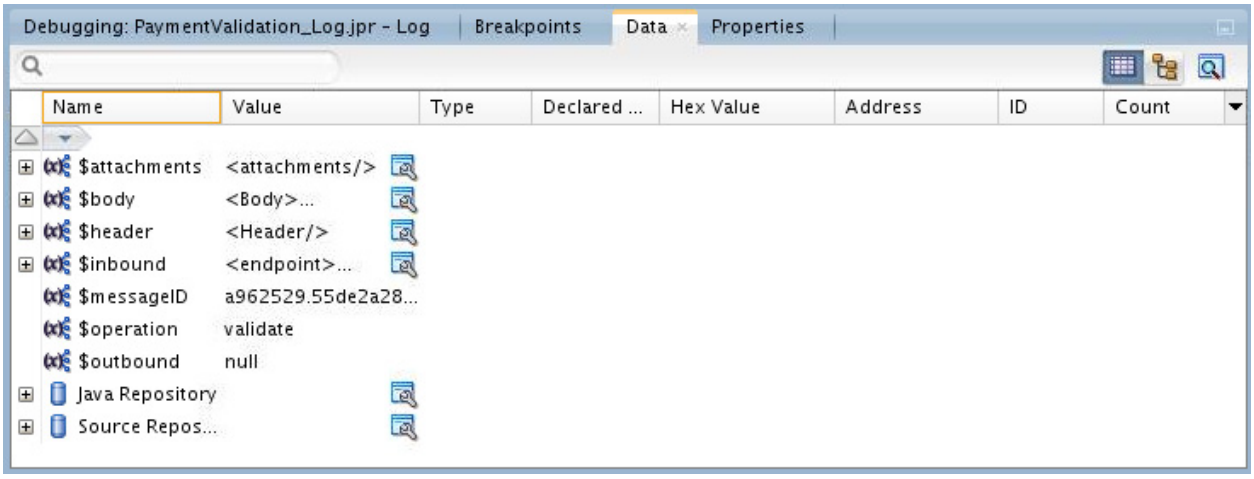
   The debugger stops at the breakpoint (which turns blue and starts pulsing).



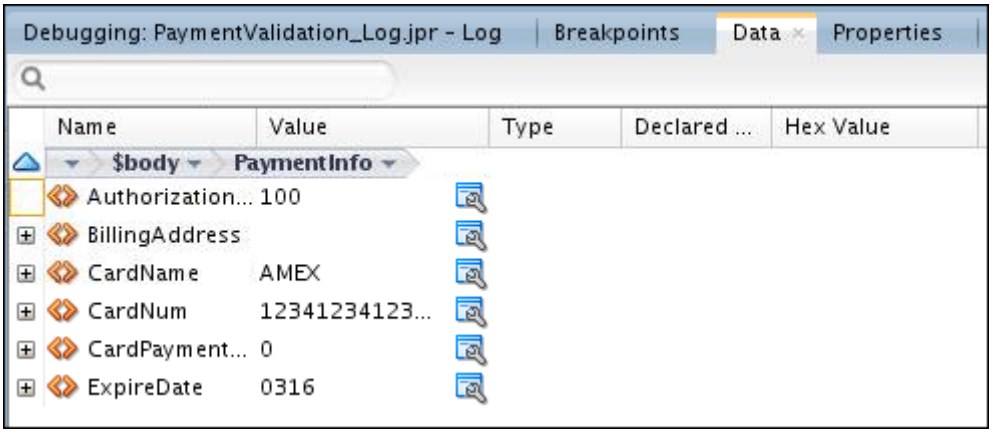
2. Step through the debugging session, examine and modify variables.
   a. From the main menu, select Window >Debugger > Data



The Data tab appears in the log window as shown below:



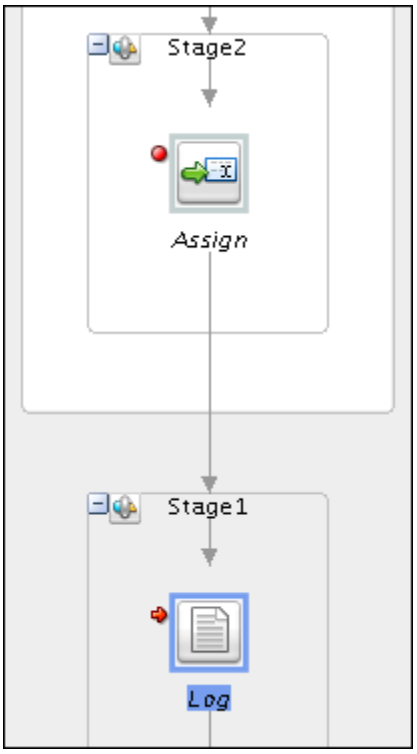
b. Expand $body > $PaymentInfo (by clicking the small plus icon in the Name column), examine the incoming values of the request.



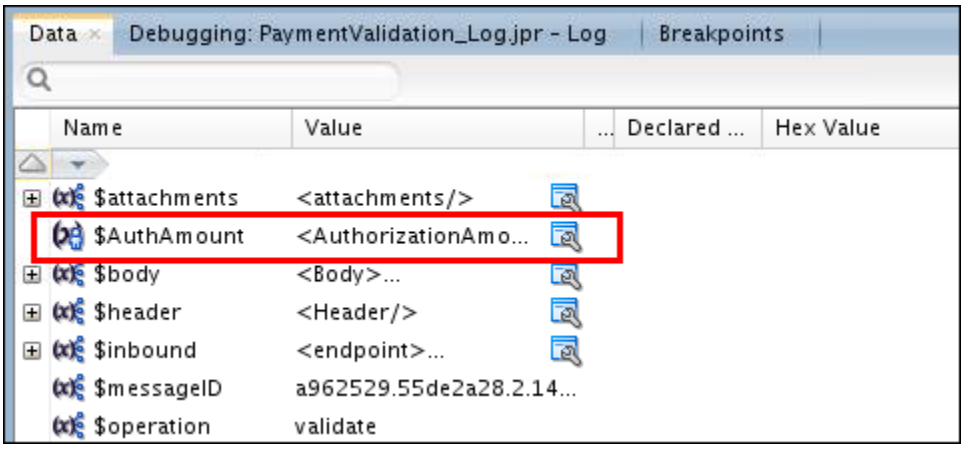
c. On the JDeveloper main menu toolbar, click Step Over or press F8.

Note that the breakpoint moves to Stage 1.

d. Click Step Over again, and the breakpoint moves to the Log action.



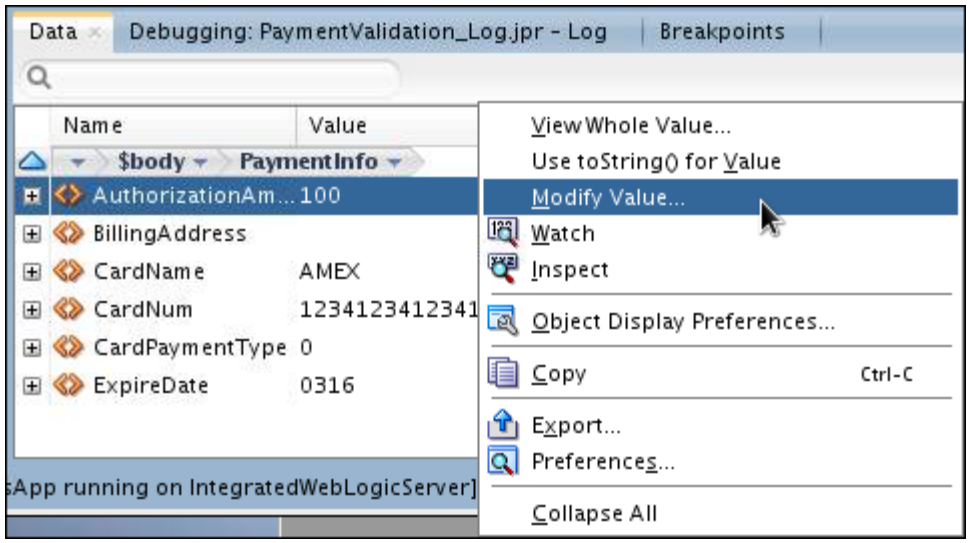
e. In the Data tab, navigate back to the root level. You see your defined variable $AuthAmount appear in the list.



3. Modify a variable value.

a. Expand $body > $PaymentInfo node.

b. Right-click AuthorizationAmount, and select **Modify value…**

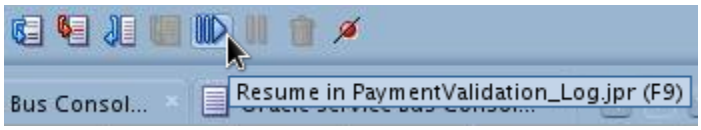


c. Enter `2000` in the Modify Value dialog that appears.

d. Click OK.

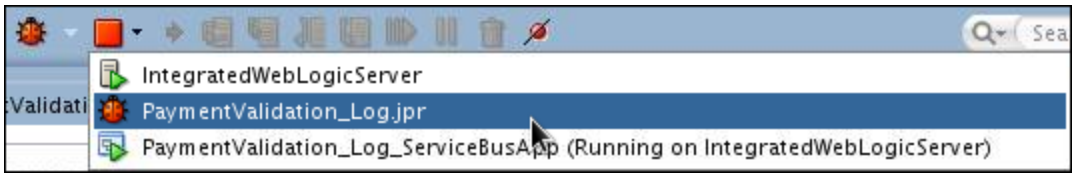
You see that the value of $AuthorizationAmount is updated to 2000.

4. Go directly to the end of message flow (by using F9) or click the Resume button on the main menu bar.

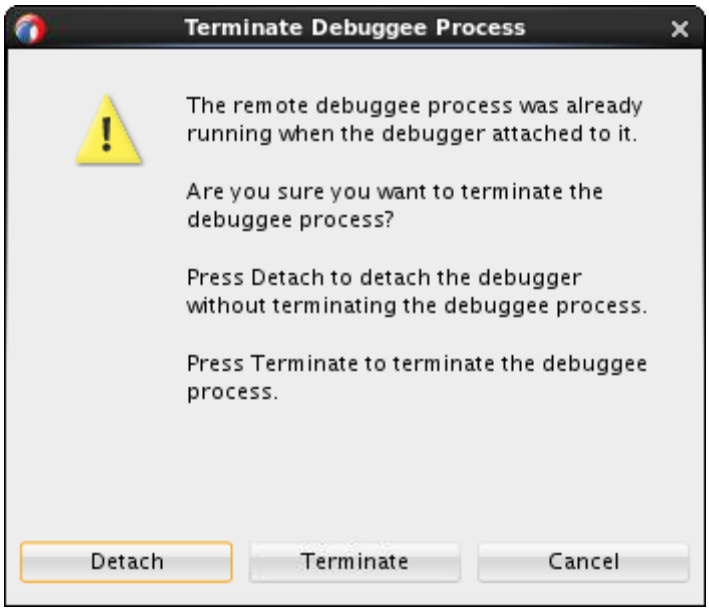


5. Go to Test Console, you should see that the status returned is Denied as opposed to Authorized.

6. On completion of debugging, on the JDeveloper menu toolbar, click the Terminate icon, and select the project's debugger process.



7. When prompted, select Terminate the Debuggee session.



8. When you are done, close the application and all open windows of the project in JDeveloper.

Practices for Lesson 5: Validating Messages and Error Handling

Chapter 5

Practices for Lesson 5: Overview

Practices Overview

In this practice, you validate messages against the schema, and explore some ways to handle the errors and monitor the deployed Service Bus application in EM.

Practice 5-1: Importing Template Resources and Creating a Business Service

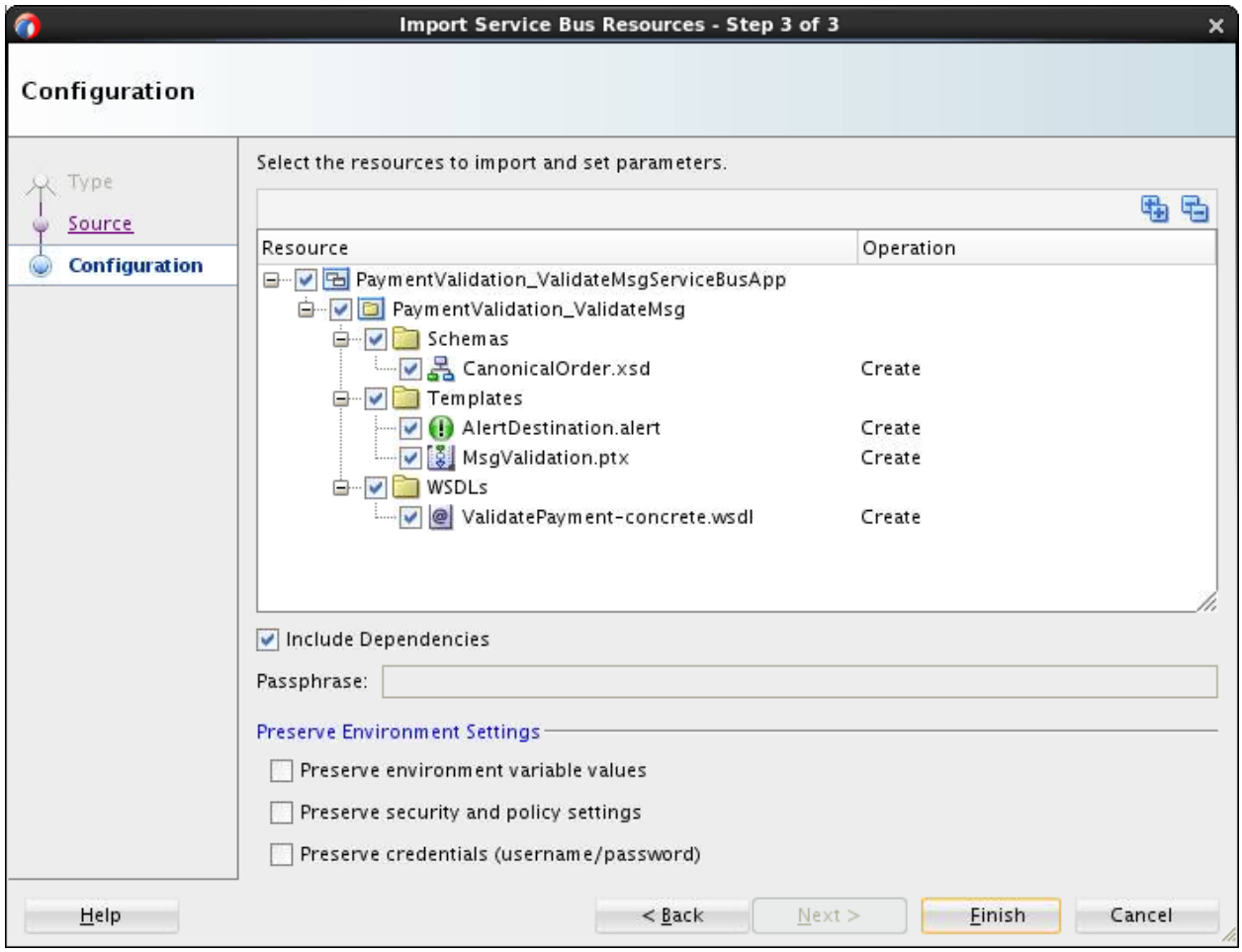
Overview

In this practice, you import the resources that were pre-created for this project and register the ValidatePayment SOA composite service to Business Bus.

Import Template Resources into the Service Bus Application

1. Open JDeveloper if it is not open.
2. Create a new Service Bus application.
   a. Select **File > New > From Gallery** from the menu.
      The New Gallery dialog box opens.
   b. Under the General category, select **Service Bus Tier**.
   c. Under Items, select **Service Bus Application.**
   d. Click **OK**.
      The Create Service Bus Application dialog box opens.
   e. Specify the application name to `PaymentValidation_ValidateMsgServiceBusApp`.
   f. Enter `/home/oracle/labs_DI/lessons/lesson05/PaymentValidation_ValidateMsgServiceBusApp` for the directory.
   g. Click **Finish**.
      The new Service Bus application folder is created.
3. Import project resources
   a. Select **File > Import** from the menu.
   b. In the Import dialog, select **Service Bus Resources**.
   c. Click OK.
      The Import Service Bus Resources wizard is displayed.
   d. Select **Configuration Jar**.
   e. Click **Next**.
      The wizard advances to step two.
   f. Click the **Browse** icon to the right of Jar Source.
      The Configuration Jar chooser is displayed.
   g. Navigate to /home/oracle/labs_DI/lessons/lesson05/PaymentValidation_ValidateMsg_Resources.jar
   h. Click **Open**.
   i. Click **Next**.
      The wizard advances to step three.

j. Review exactly what is being imported into the application.



Service Bus supports the ability to import and export artifacts and projects at a fine-grain level. You may import artifacts individually, such as a WSDL or schema, or whole projects. You may control whether dependencies are included or not.

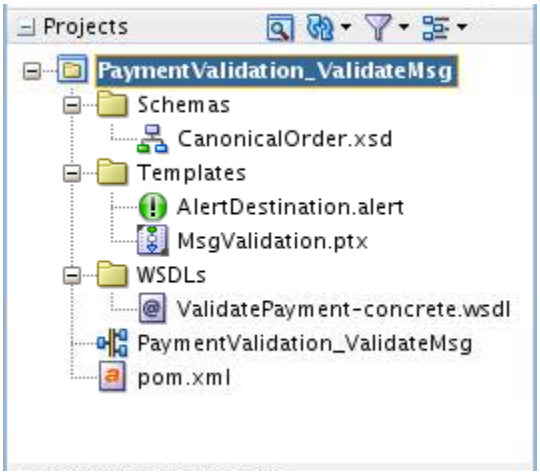
By default, when importing individual artifacts, Service Bus will attempt to import all dependencies declared in the artifact. For example, if a WSDL includes a schema, the schema will also be imported if the paths are relative.

Also note that you can control whether environmental settings, security policies, and credentials are preserved on import. If you are bringing artifacts from a production environment for testing and editing, you may not want all the same policies applied in the development environment, for example.

You will accept all defaults in this case.

k. Click **Finish**.

The Application Navigator now resembles the following:



Create business service

4. Create a business service.

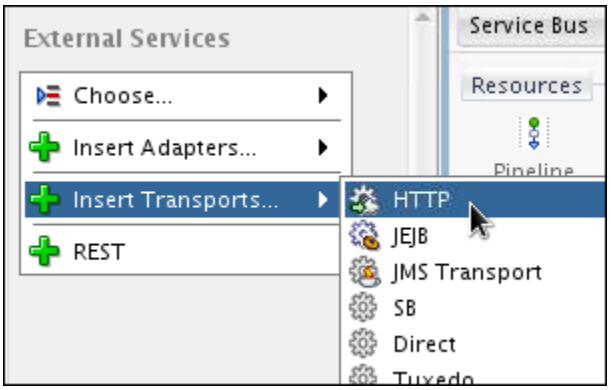
   Register the PaymentValidation service on Service Bus by creating a Business Service. This time, rather than dragging and dropping from the Component Palette, you will create using a menu-based Insert on the External References lane.

   a. Double-click the project overview file, `PaymentValidation_Validatemsg`.

      The Service Bus Overview Editor opens with a blank canvas.

   b. Right-click the **External Services** lane of the overview editor.

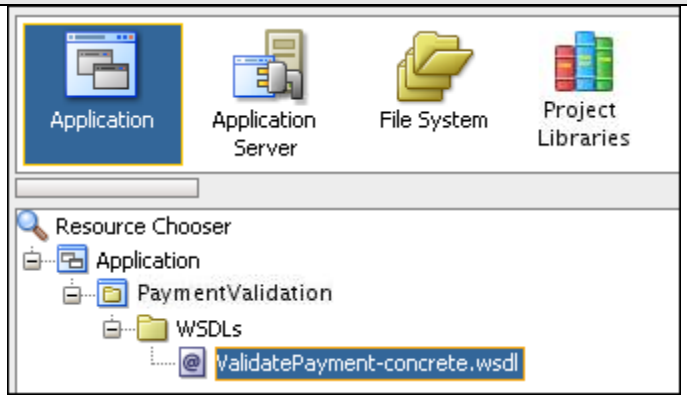
   c. Mouse-over **Insert Transports** and select **HTTP**.



   The Create Business Service dialog box is displayed.

   d. Enter the required details for business by following the instructions in the table below:

Step	Window/Page Description	Choices or Values
1)	Create Service	Name the service `PaymentValidation_ValidateMsgBS`. Click **Next**.
2)	Type	• 1) Select **WSDL** for Service Type. • 2) Click the Browse WSDLs icon to the right of the WSDL choice. • 3) In the Select WSDL dialog box, choose ValidatePayment-concrete.wsdl as shown below:

Step	Window/Page Description	Choices or Values
		 • 4) Click **OK**. • 5) Back in the Create Business Service window, you should see the WSDL and binding fields populated with the information based on the WSDL you chose. • 6) Click **Next**.
3)	Transport	Verify that http is selected in the Transport field. The Endpoint URI is populated as: http://localhost:7101/soa-infra/services/default/ValidatePayment/validatepaymentprocess_client_ep Click **Finish**.

Note: There are several ways to check the deployed endpoint of a Composite. One way is to visit the EM console (http://localhost:7101/em) and navigate to the composite. Click the Test icon near the top right of window. This will bring up a Web Service test page that lists the deployed endpoint.

If needed, the endpoint URI can be updated as you review the business service properties on the Transport tab.

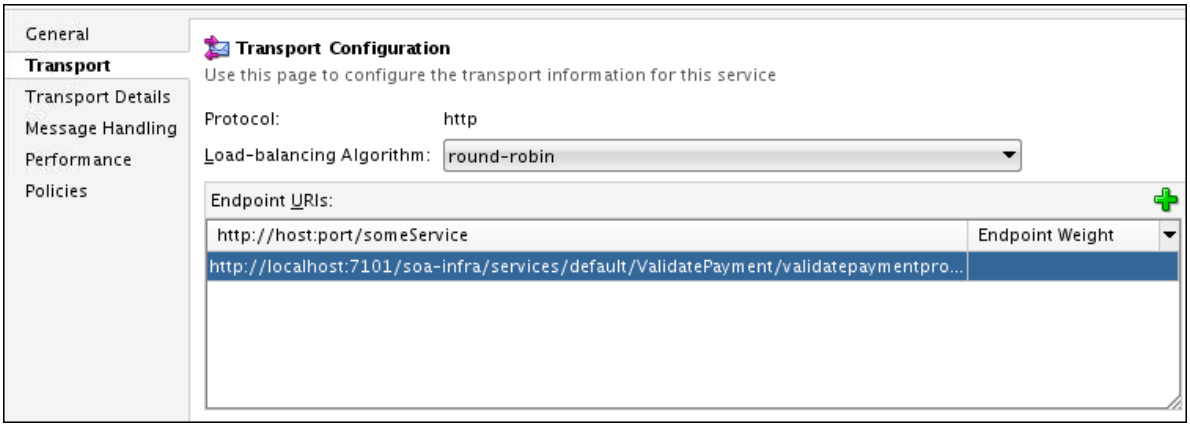
e. Business Service is created on the External Services lane and its file resides under the project folder.

f. Link the business service to the endpoint:

1) Double-clicking the PaymentValidation_ValidateMsgBS.bix file.

2) In the configuration window, select the **Transport** tab.

3) Double-click the Endpoint URIs field of the business service and replace it with the ValidatePayment service endpoint URL.



5. Deploy and test your business service using the Run command.
   You can refer to the instructions of Deploy and Test the Business Service in Practice 3-2.

Practice 5-2: Creating the pipeline that validates messages using the Pipeline Template

Overview

In this practice, you create the pipeline with capabilities of data validation, reporting, error handling and alerting under error conditions. Now, instead of creating the Pipeline for PaymentValidation from scratch, you leverage a Pipeline Template.

Tasks

Create new pipeline with proxy using a pipeline template

1. In JDeveloper, make sure your overview editor is active by double-clicking on the PaymentValidation_ValidateMsg in the Application Navigator.
2. Drag the pipeline from the Component palette onto the middle of your canvas.

   The Create Pipeline Service wizard opens.

   a. Specify name as **PaymentValidation_ValidateMsgPP**

   b. Select the option **From Template**, and click the Search icon.

   c. When the Pipeline Template chooser appears, navigate to **Application>PaymentValidation_ValidateMsg>Templates** and select **MsgValidation.ptx**

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

   e. In Type screen, use the same WSDL of the business service for the pipeline service:

      1) Select the service type **WSDL.**

      2) Click the **Browse WSDLs** icon on the right.

         The Select WSDL dialog box pops up.

      3) Navigate to Application > PaymentValidation_ ValidateMsg > WSDLs directory, and select ValidatePayment-concrete.wsdl.

      4) Click **OK**.

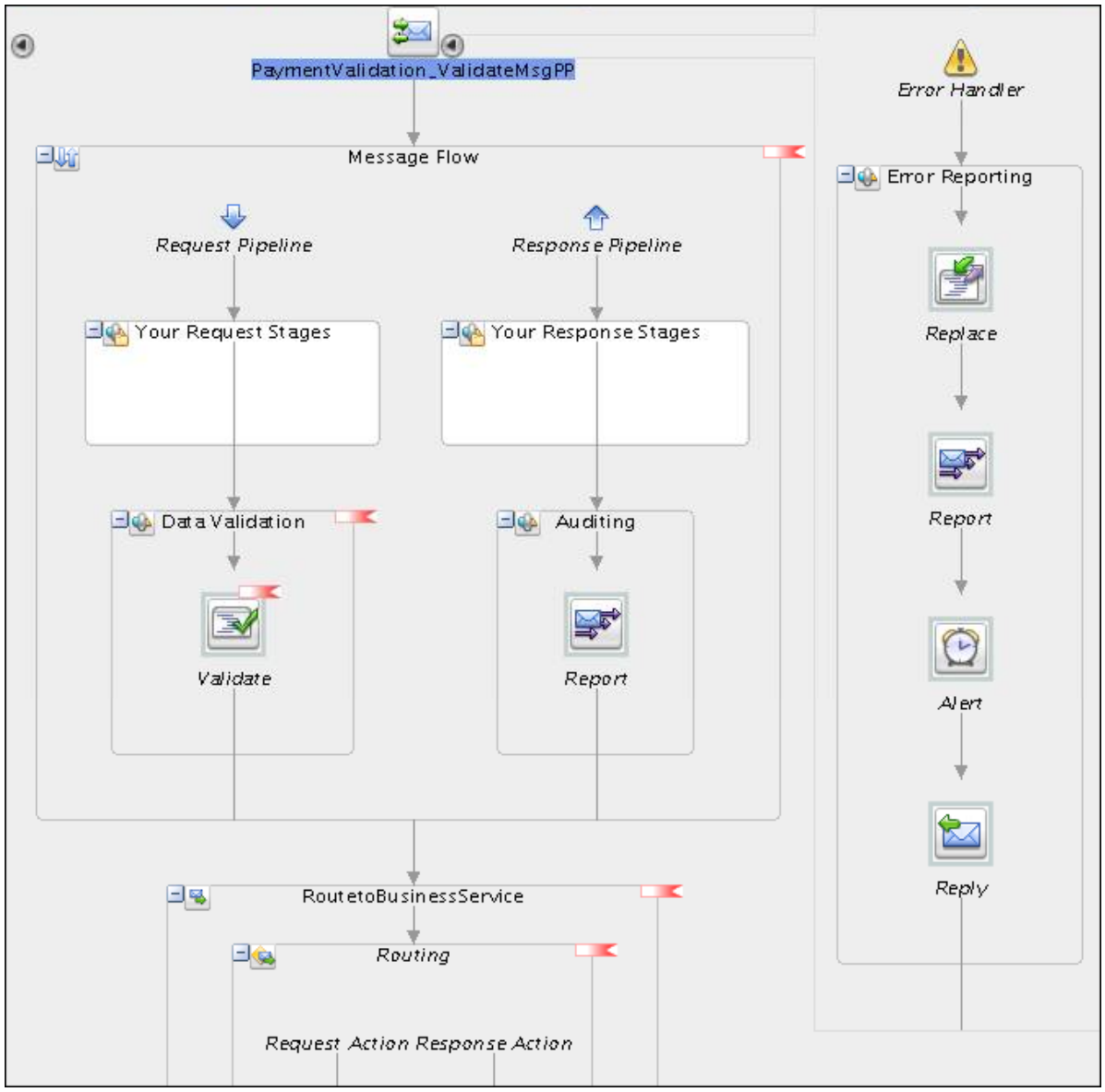
         Back in the wizard, you should see the fields populated with the wsdl file and binding.

      5) Keep "Expose as a Proxy Service" checked for creating Proxy Service along with pipeline.

      6) Change Proxy Service name to **PaymentValidation_ ValidateMsgPS**, and the default directory of the proxy service to the project folder: `/home/oracle/labs_DI/lessons/lesson05/PaymentValidation_ ValidateMsgServiceBusApp/PaymentValidation_ValidateMsg`. Leave the rest values to their defaults.

      7) Click Finish.

On your canvas, you will now see your new pipeline created from the template.



This Pipeline has all kinds of good stuff already built for you–Data Validation, Routing, Error Handling–all you need to do is fill in the blanks.

Notice the ‘Your Request Stages’. This is where the Template Designer has left a placeholder for you to customize the Pipeline to do any additional work like transformations, message enrichment before the data Validation action will be performed, likewise on the Response Pipeline.

Fill in the template and complete the pipeline

Notice the components marked with the red flag that indicate the areas that need to be filled in to have a valid pipeline.

3. Edit Validate action
   a. Click on the **Validate** action to open the Property Editor.

      You will validate the incoming payload against the canonical PaymentInfo element type that the ValidatePayment service is expecting. By validating in Service Bus, you save precious resources in the backend that are actually processing good payment authorization requests.

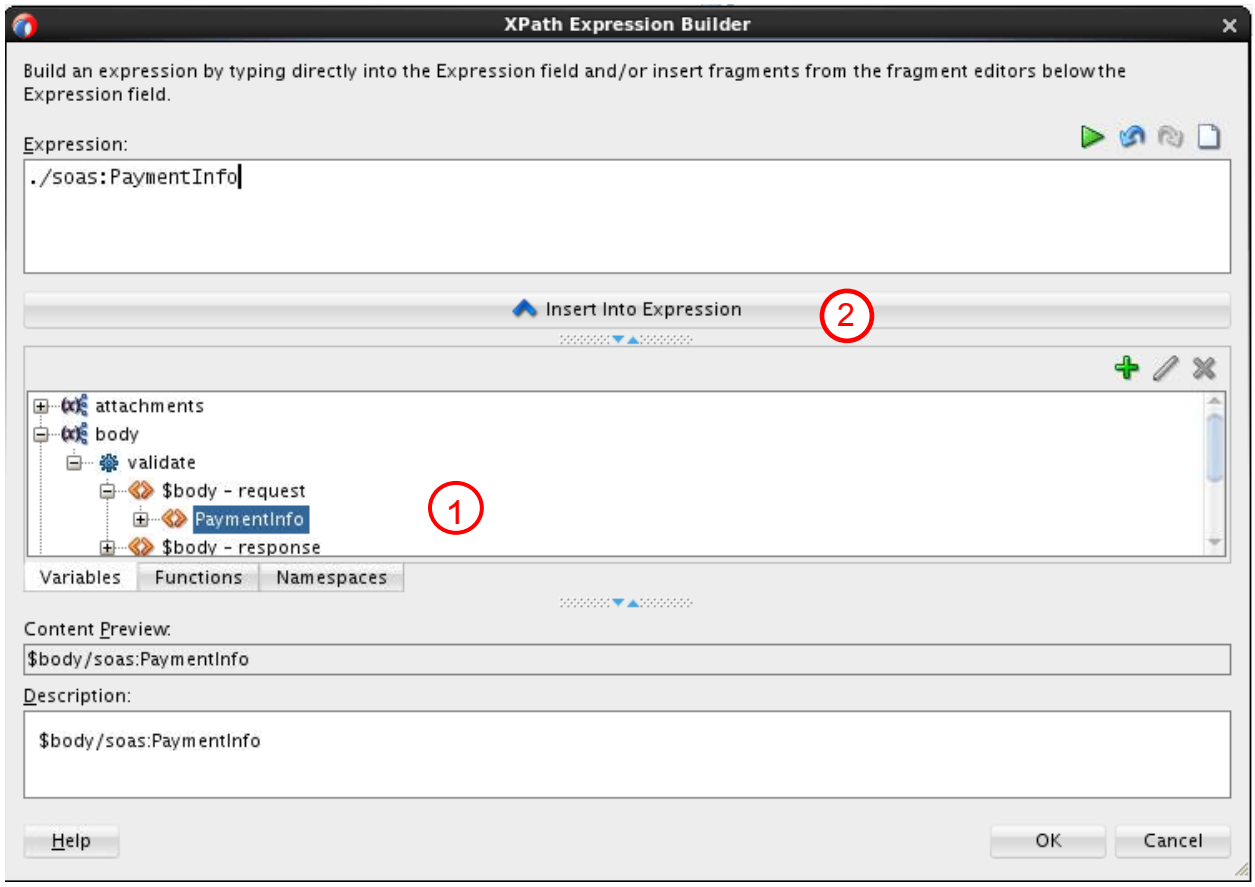
   b. The `body` location has already been provided by the template designer, you need to specify the element that need validation using XPath. Click  .

      The XPath Expression Builder opens.

   c. Navigate to body > validate > body – request, and select **PaymentInfo** element.

   d. Click **Insert Into Expression** button.

      You should see that the Expression field is populated with the XPath that you selected.

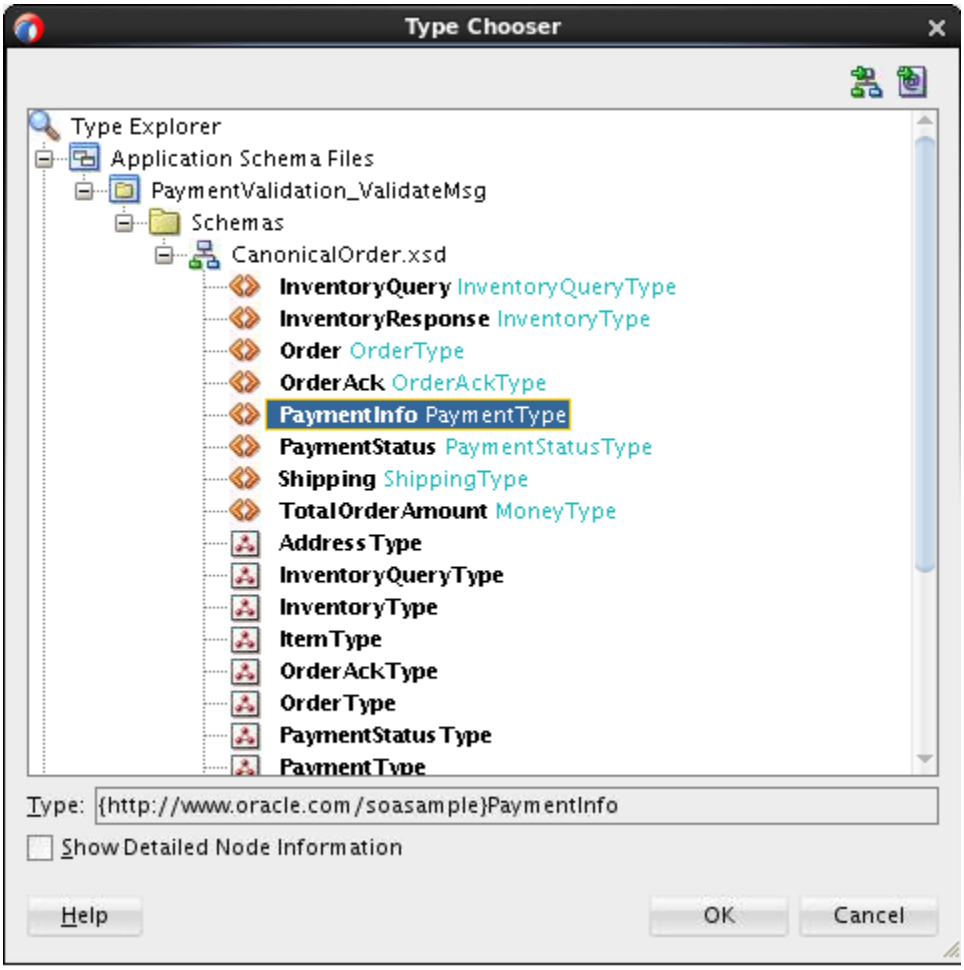


   e. Click **OK**.

   f. Select **Static** for Schema.

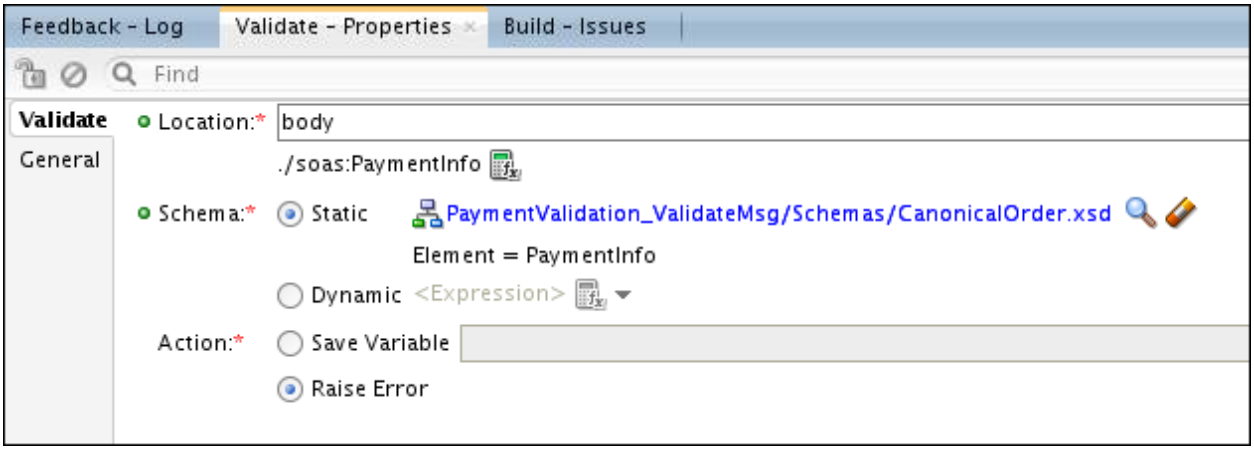
   g. Click the browse button on the right to bring up the XML Schema/WSDL chooser.

h. Navigate to Application Schema Files > PaymentValidation_ValidateMsg > Schemas > CanonicalOrder.xsd, and select **PaymentInfo.**



i. Click OK.

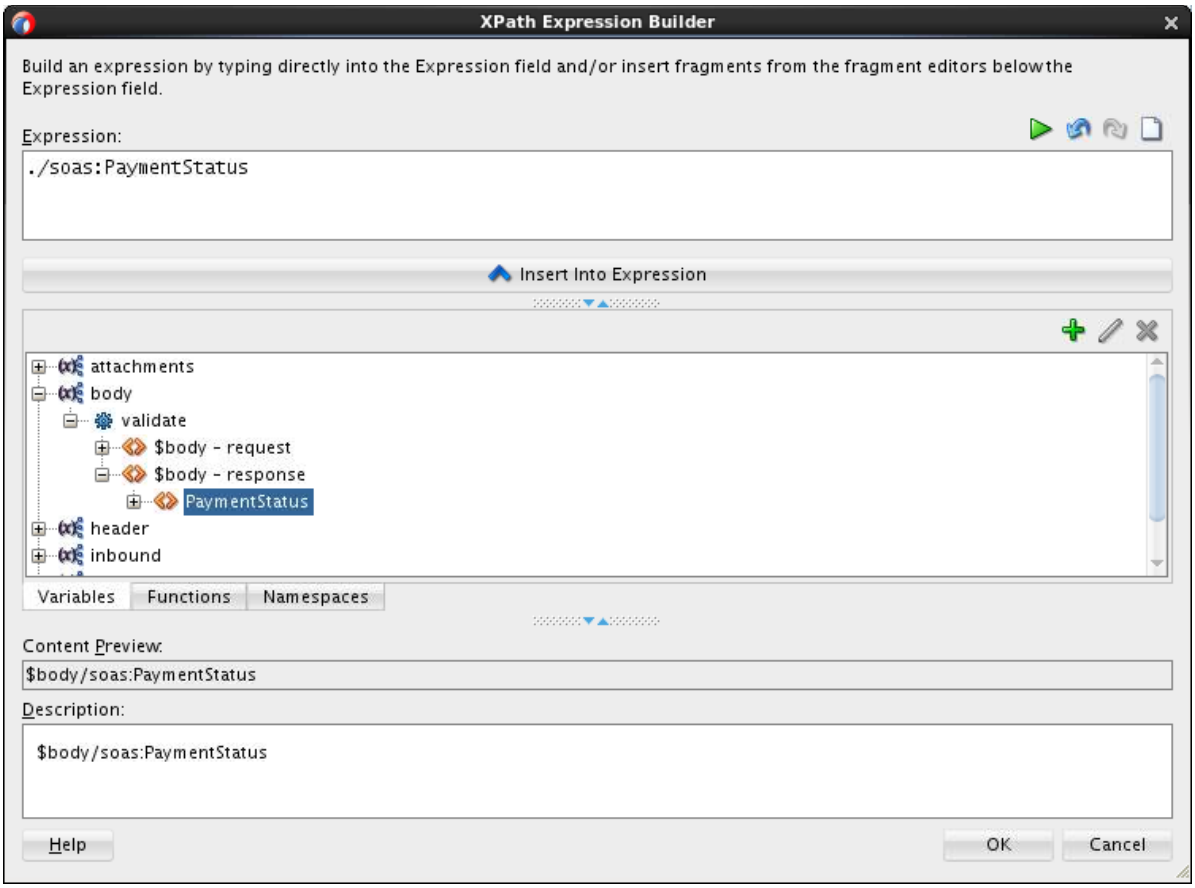
The Properties inspector is updated as shown below:



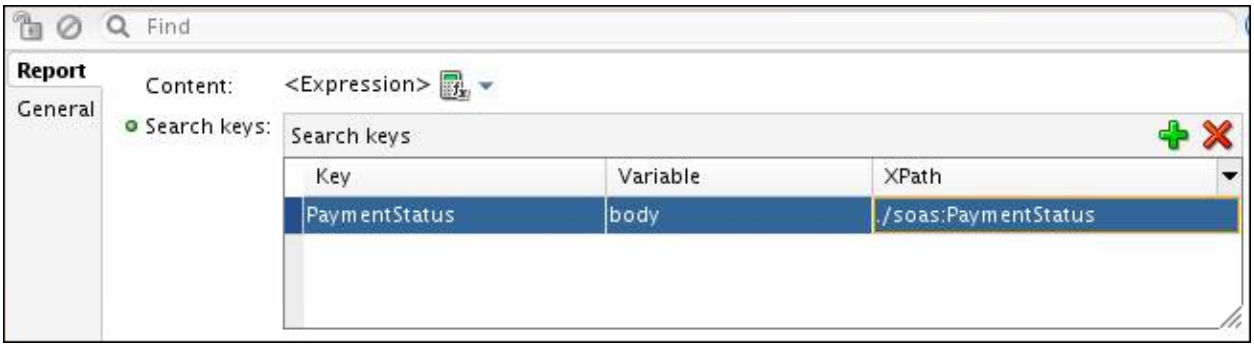
Notice that the red flag has gone.

Keep the default Raise Error for Action. The template defines an error handler to handle the errors generated from the failed validation.

4. Edit the Report action in the Auditing stage of the response flow. Let's customize to report on the payment status that is returned from the backend service.
   a. In the Report Properties inspector, click on the green + icon. Enter or set the following values:
      - Key: **`PaymentStatus`**
      - Variable: Select **body**
      - XPath: Click on the little icon to the right to pop-up the expression editor.
        In the XPath Expression Builder, set the expression as follow:



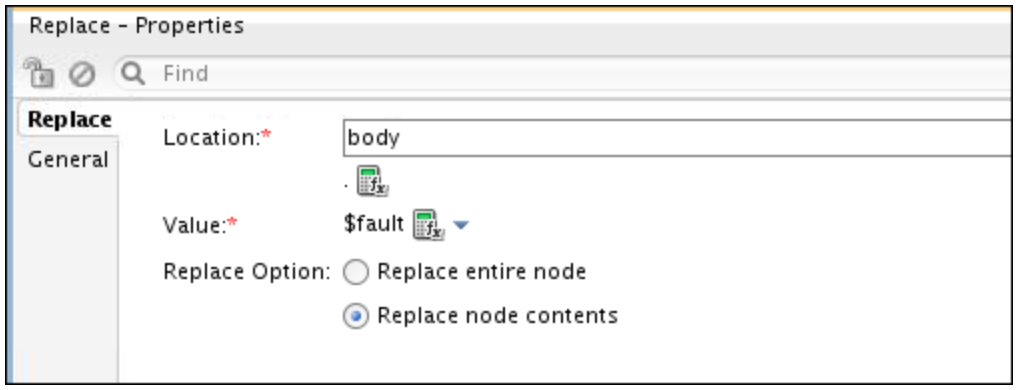
   b. Click OK.
      The Report Properties inspector is updated with the values that you specified.



5. Explore the Error Handler.

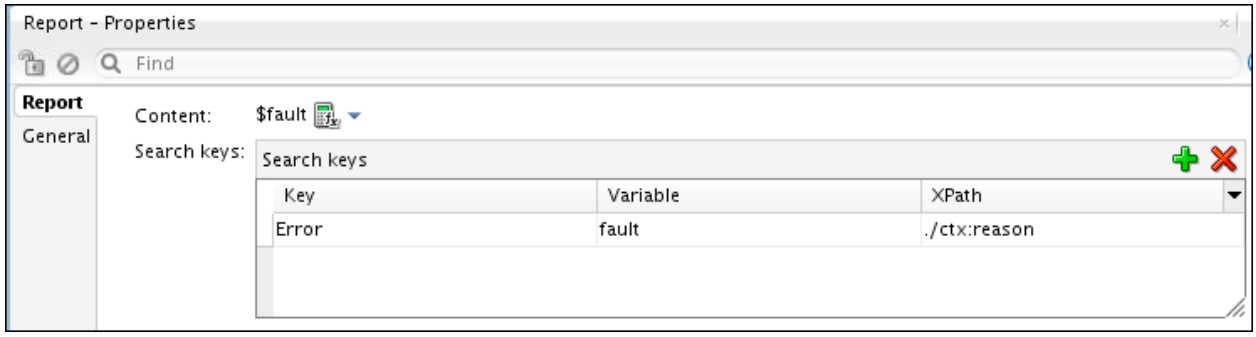
   Best practice is to always have an Error Handler defined for Pipelines. This pre-defined Error Handler will report the error back to the caller with details and also send up an Alert to the EM console that something is wrong with processing. It is likely that you may have invalid credit card data incoming.

   a. Click **Replace**.



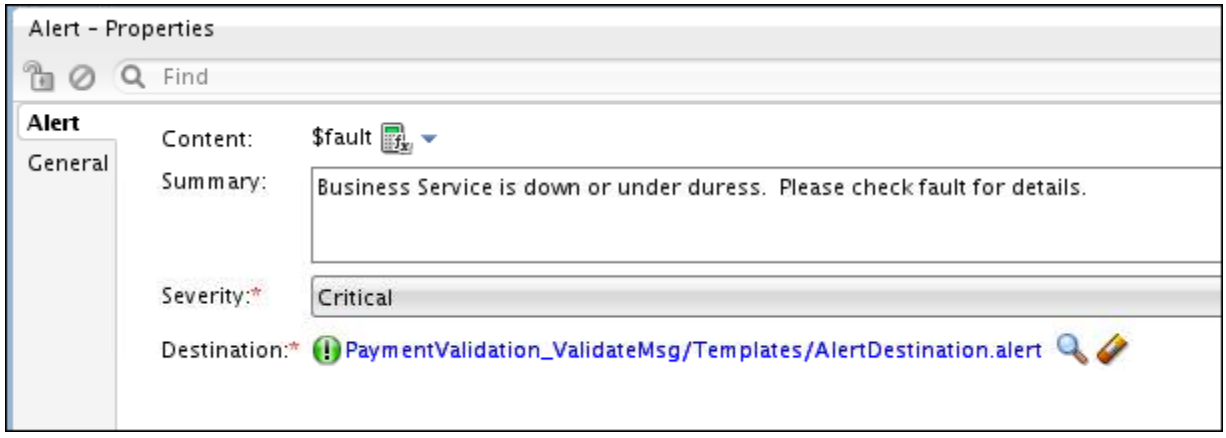
   When there is an error, the message body content will be replaced by fault value.

   b. Click Report action.



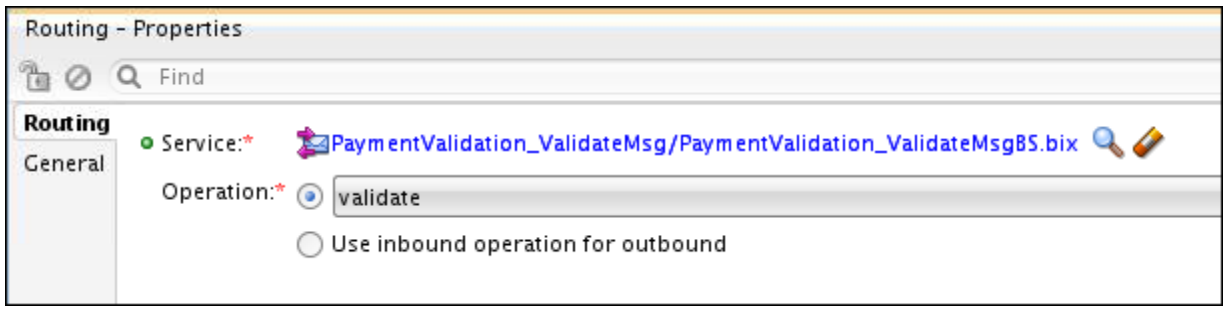
   The report action will provide the reason for the error.

   c. Click Alert action.



   When an error occurs in the pipeline, an alert will be sent to the alert logging (defined in the AlertDestination.alert file) with a severity level of Critical.

6. Routing to the business service
   a. Click the Routing node.
   b. In the Routing Property inspector, click the browse button, and select the PaymentValidation_ValidateMsgBS.bix as the target service.



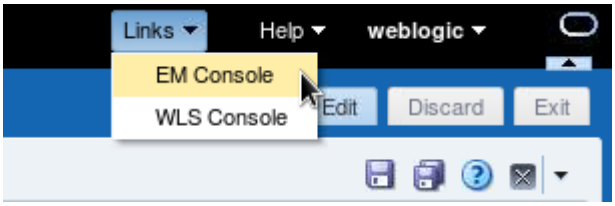
7. Save your project.

Deploy and Test

8. Deploy the project to the integrated WLS server.
   a. In Application Navigator, right click PaymentValidation_ValidateMsg project and select to deploy your application profile.
      The Deploy wizard window opens.
   b. For Deployment Action, select **Deploy to Service Bus Server**, and click **Next**.
   c. For Select Server, select **IntegratedWeblogicServer**, and click **Next**.
   d. Review the summary and click **Finish**.

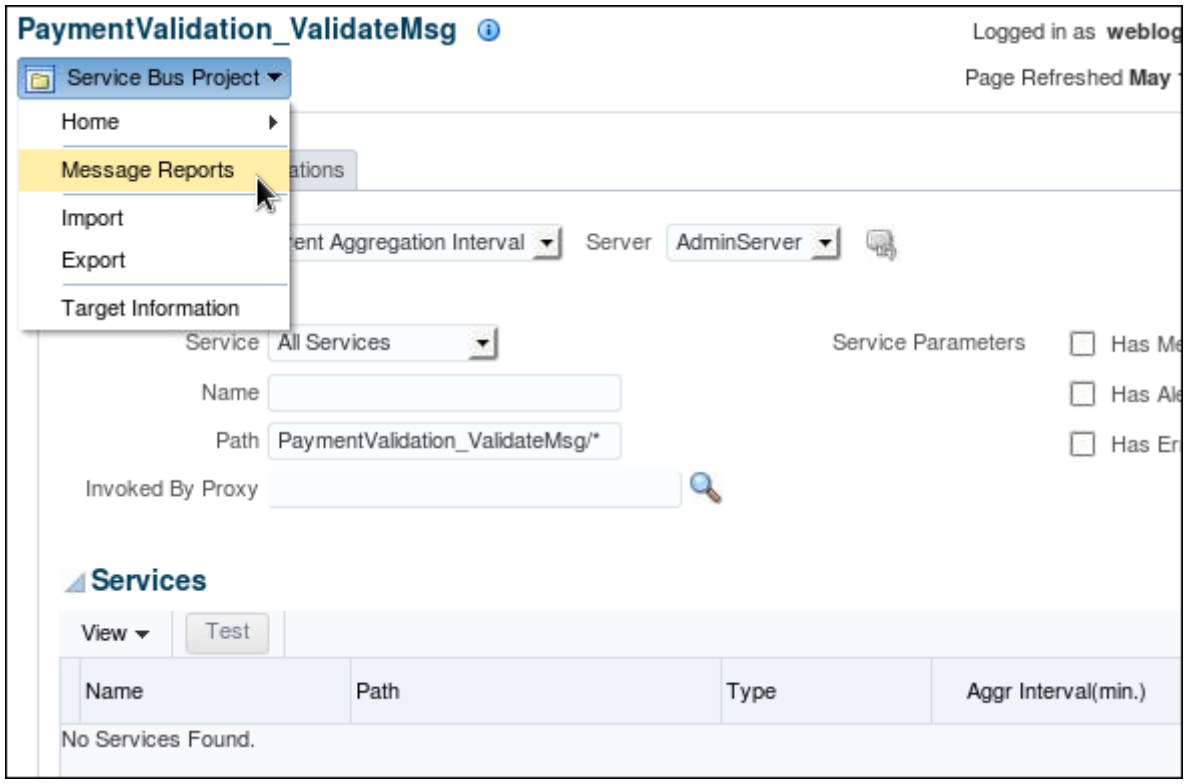
   In the Deployment log, at the bottom of JDeveloper, you should see the deployment finished without errors.
9. Launch Test Console
   a. In a Firefox browser, access Service Bus console using the URL:
      http://localhost: 7101/sbconsole
   b. In the Project Navigator, select PaymentValidation_ValidateMsg to view the project definition on the right.
   c. Click the Launch Test Console button next to the proxy service.
      The Test Console opens.
10. Test a “happy” path to view the result.
   a. Click the Browse button next to the Payload field.
      The File Upload window opens.
   b. Select File System, navigate to `/home/oracle/labs_DI/resources/sample_input/`, and select **`PaymentInfoSample_Authorized.xml`**
   c. Click **Open**.
   d. Click **Execute**.
      On the next screen, in the Response Document section, you see that the payment is authorized.

11. View the report in EM.
    a. Go back to Service Bus Console.
    b. Click Links on the top right corner of the window and select EM Console from the drop-down list.



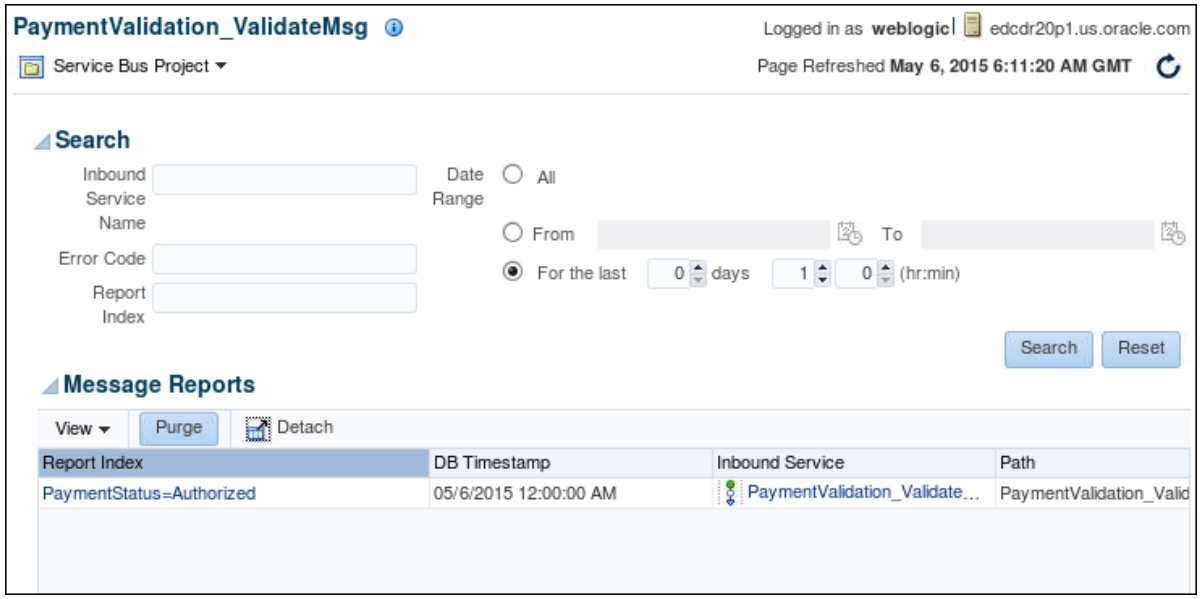
    The EM Console opens in a new tab window.
    c. In the navigator, navigate to SOA > service-bus, select **PaymentValidation_ValidateMsg**.

    PaymentValidation_ValidateMsg dashboard is displayed on the right.
    d. Click the Service Bus Project menu, and select Message Reports.



    e. In order to view message reports, you need to perform a search for the reports. To retrieve the most recent messages:
        1) Click the **For the last** radio button
        2) In the hr:mm fields, select 1 hour.
        3) Click Search.

A list of message reports matching your criteria appears at the lower portion of the window.

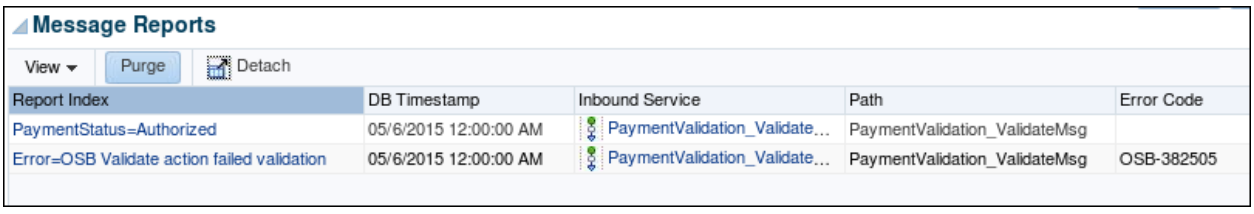


You can click the PaymentStatus=Authorized link to view the details.

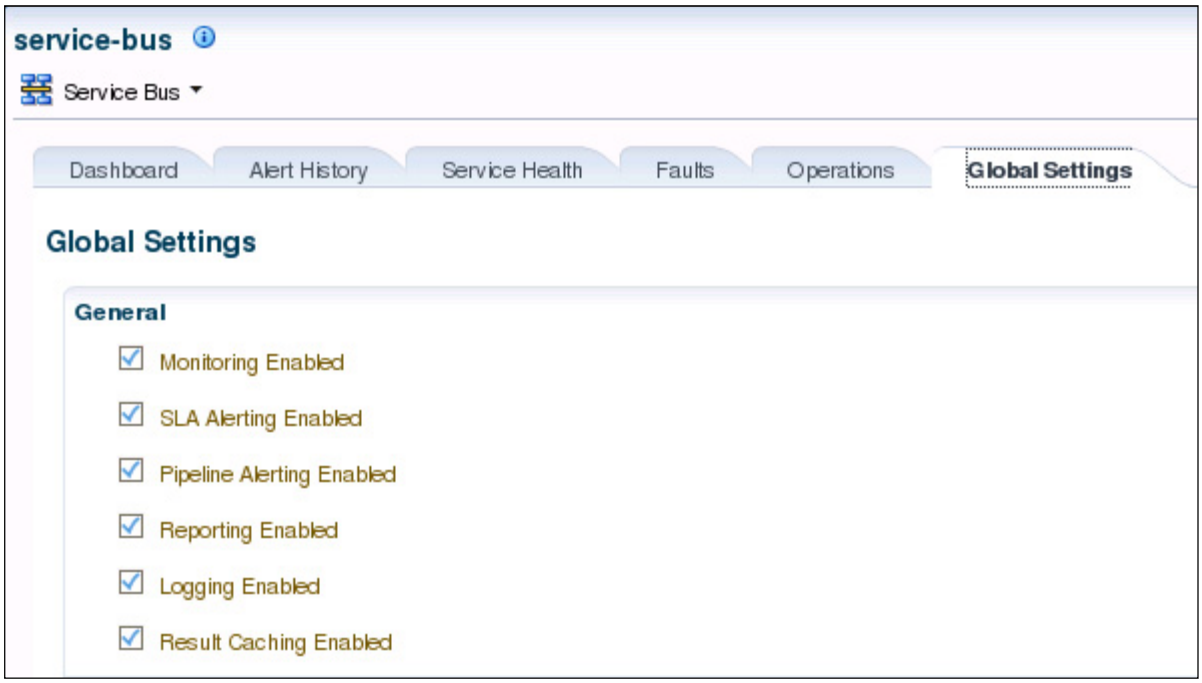
12. Now let's test a scenario with error to view the result.
    a. Go back to Test Console and make sure that you are at the Proxy Service Testing home page.
    b. Click the Browse button next to the Payload field.

       The File Upload window opens.
    c. Select File System, navigate to `/home/oracle/labs_DI/resources/sample_input/`, and select **`PaymentInfoSample_InvalidCardNum.xml`**
    d. Click **Open**.
    e. Click **Execute**.

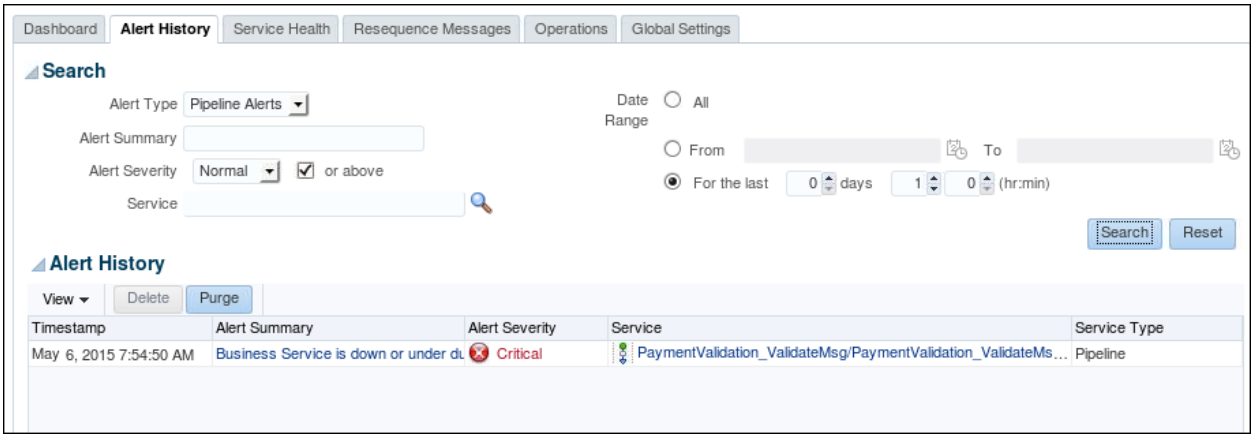
       On the next screen, in the Response Document section, you see that an error is raised and the response body shows the error details including errorCode, Reason, details, and Location.
13. View the error report in EM.
    a. Go back to EM Console.
    b. On the Message Reports page, click Search (using the same search criteria). You should see the new entry with error code added to the report.



14. View Alert information
    a. Click back on the top level of the service bus in the left-hand navigation tree.
       This will remove the search filter. Browse along the tabs: Global Settings, Operations (remember to hit Search!), Faults, Service Health, Alert History, and Dashboard.



    b. Click the **Alert History** tab.
    c. Select **Pipeline Alerts** for Alert Type.
    d. Set the search criteria to the last one hour, and click **Search**.
       You should see a critical alert that is thrown from the service.



15. When you are done, close the application and all open windows of the project in JDeveloper.

Practices for Lesson 6: Transforming Messages

Chapter 6

Practices for Lesson 6: Overview

Practices Overview

The order management system sometimes gets orders from their partners. These partners use different payment service APIs. In these practices, you define a transformation based on XQuery, to mediate the requests from partners, so that they can be processed by the ValidatePayment SOA service.

In addition, you add a new file order channel to the current order processing service. The new proxy handles incoming orders by file and translates common-delimited format with nXSD.

Practice 6-1: Transforming Data using XQuery Transformation

Overview

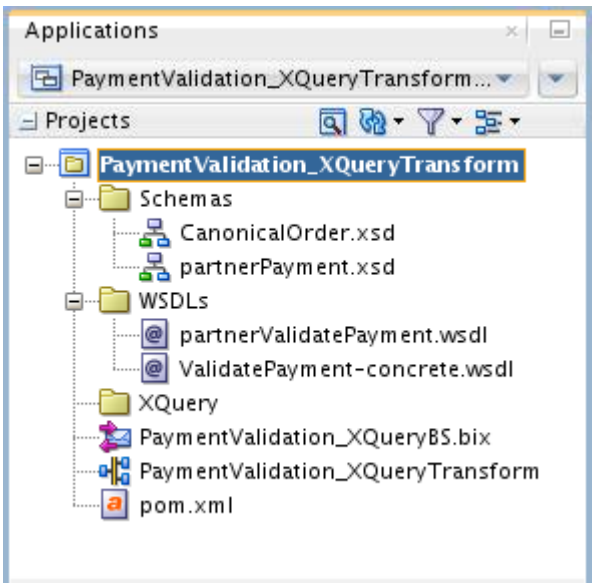
In this practice, you create an XQuery transformation to map data format from a partner's payment validation request to the format that conforms to the ValidatePayment service schema.

High-level steps:

- Examine the pre-created project artifacts.
- Create XQuery transformation map.
- Create proxy and pipeline and wire to the business service.
- Configure pipeline to convert the request using XQuery transformation and pass to the business service.
- Deploy and test.

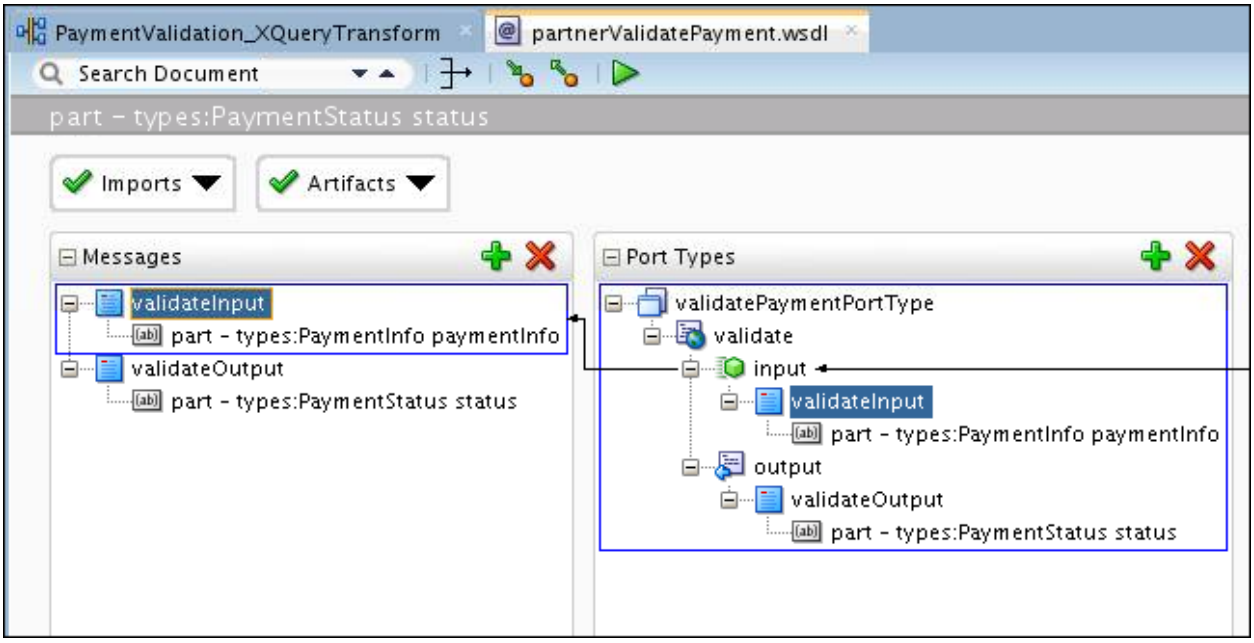
Tasks

1. Open Service Bus project
   a. In JDeveloper, select **File** > **Open** from the main menu.
   b. In the Open Application(s) dialog box, navigate to the `$HOME/labs_DI/lessons/lesson06/PaymentValidation_XQueryTransformServiceBusApp/` directory, and open the `PaymentValidation_XQueryTransformServiceBusApp.jws` file.
   c. Expand the folders in Application Navigator, and you should see the folder structure resembling the image below:



   To save your time, the project folder structure is pre-created and source WSDL and Schema files are imported into the project for you. In addition, the business service that invokes the backend ValidatePayment SOA service is also created. You can see it in the application Overview Editor.
2. Inspecting WSDL and Schema files of partner
   a. Double-click the `partnerValidatePayment.wsdl` file in the WSDLs folder. The WSDL editor opens and presents the design view of the WSDL document.

b. Examine the contents of Messages, Port Types, Bindings / Partner Link Types. In the Design view, they are the same as ValidatePayment-concrete.wsdl in terms of message type, operations, and bindings.



c. Click the Source tab at the bottom of the WSDL editor. The source view of the WSDL document presents.

Note that the namespace and referenced schema source file are different from ValidatePayment-concrete.wsdl.

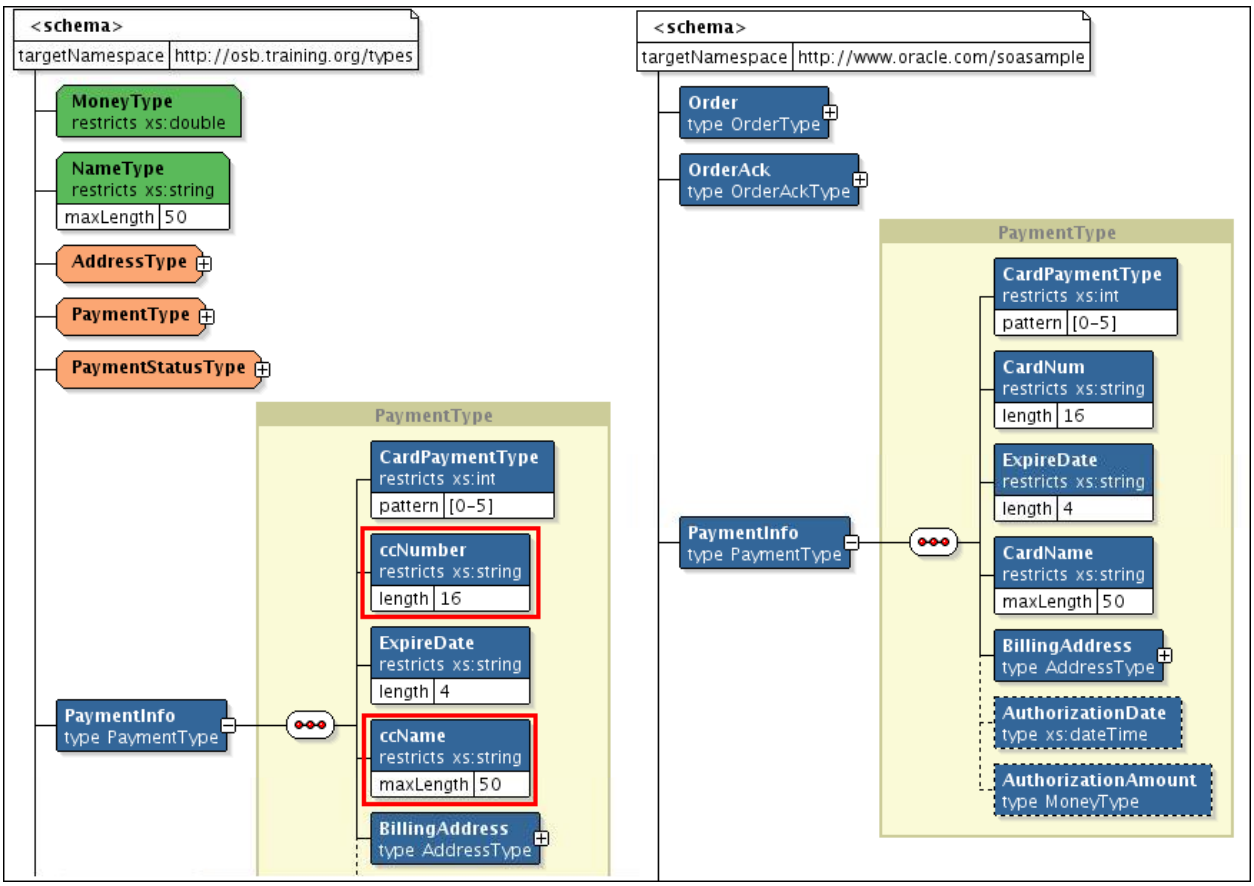
```
<wsdl:definitions targetNamespace="http://osb.training.org/partnerPaymentValidate"

  <wsdl:types>
    <xsd:schema>
      <xsd:import namespace="http://osb.training.org/types"/>
      <xsd:import schemaLocation="../Schemas/partnerPayment.xsd" namespace="http://
    </xsd:schema>
  </wsdl:types>
```

d. Double-click the partnerPayment.xsd file in the Schemas folder in Project Explorer.

The XSD editor opens and presents the design view of the XSD document.

e. Expand the PaymentInfo element and review the content.

f. You can compare it with the PaymentInfo definition in the CanonicalOrder.xsd file and tell the differences.

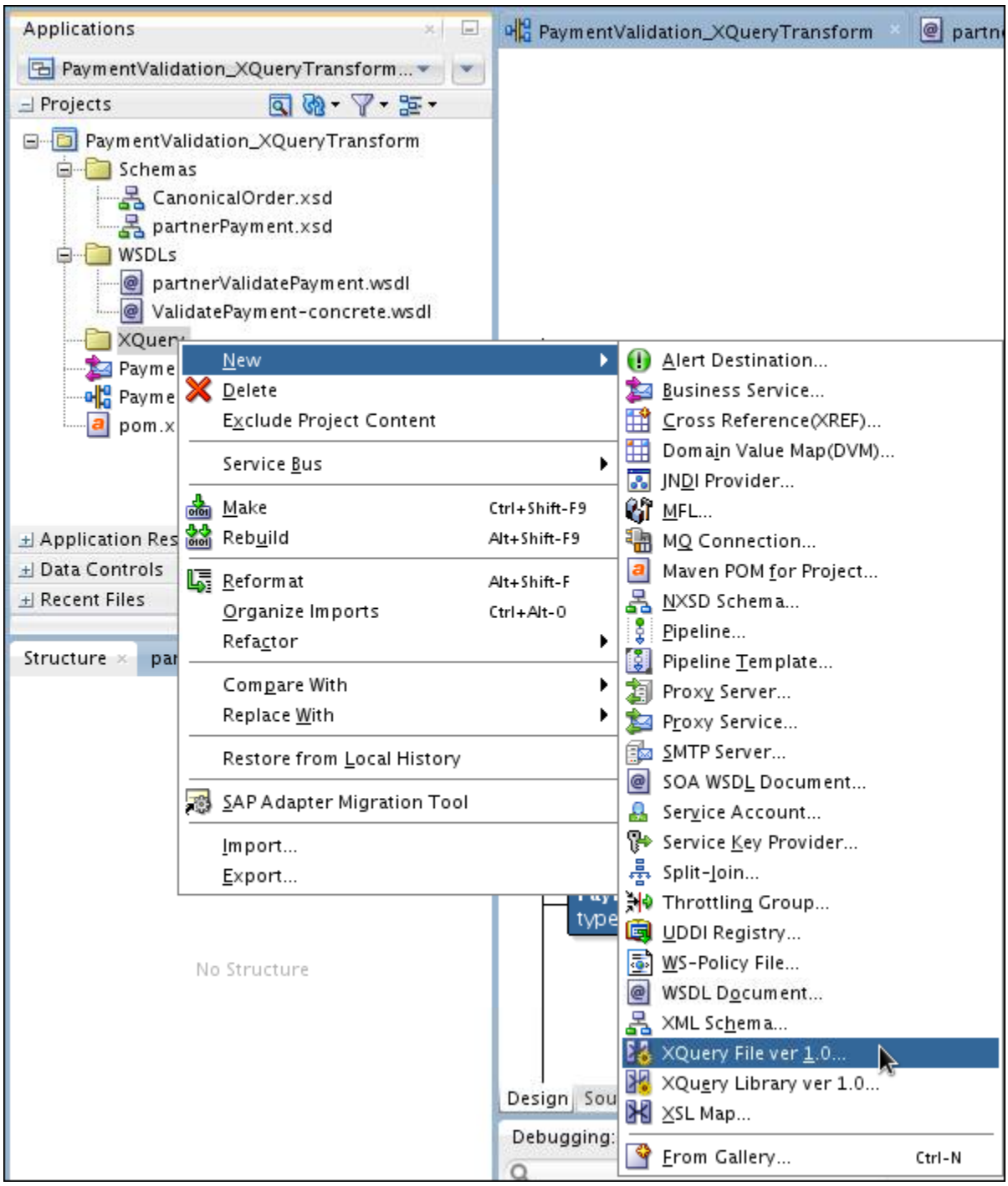


Note: This is the simplified use case for lab purpose. In the real world, the transformation could be more complicated.

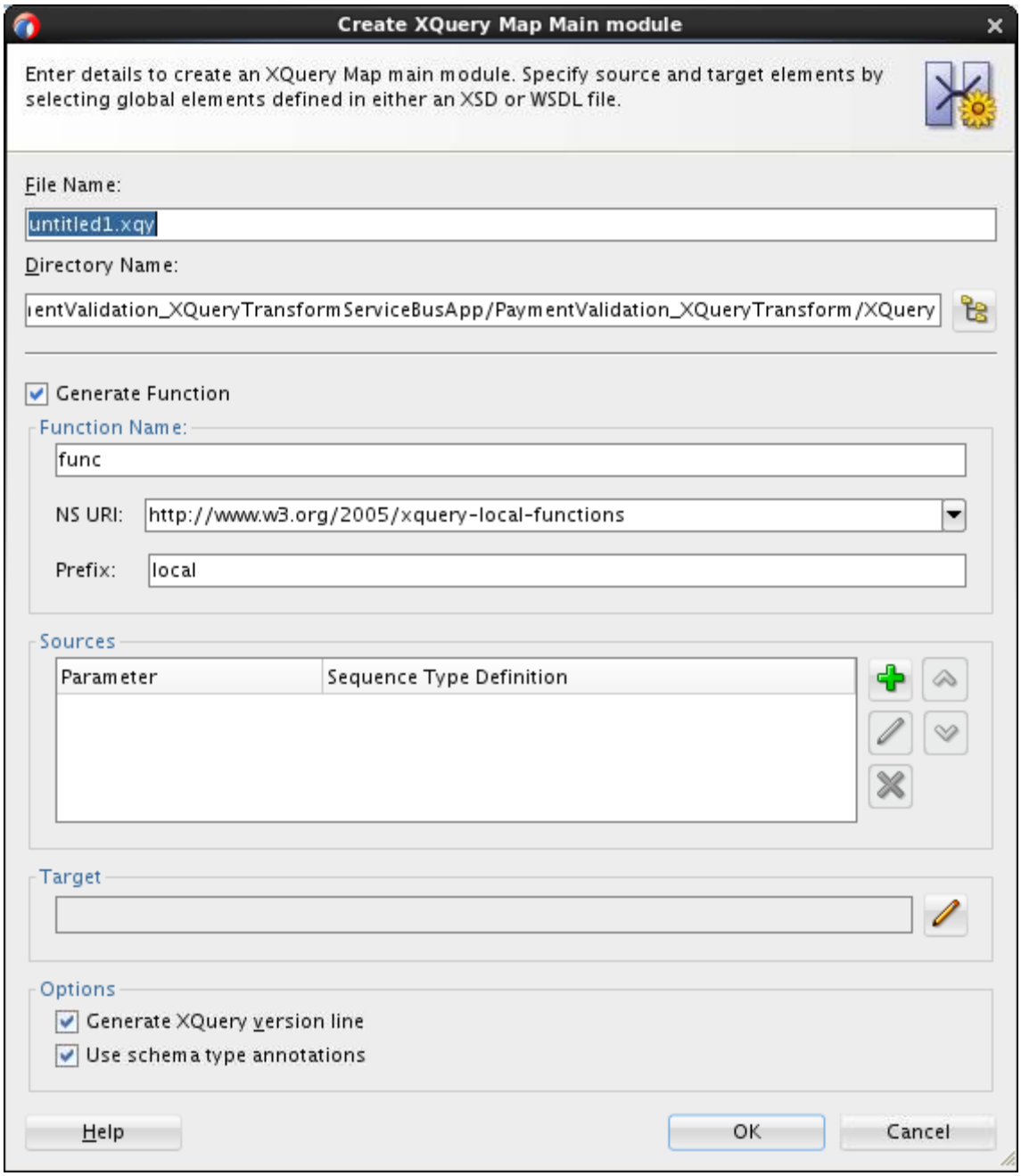
Create XQuery transformation map

You use XQuery Mapper to transform the data of payment validation request from the partner to the format that is accepted by the ValidatePayment endpoint.

3. In Project explorer, right-click the XQuery folder, select **New** > **XQueryFile ver 1.0…** from the context menu.



The Create XQuery Map Main Module dialog box appears.



4. Under File Name, enter **`partnerPayment2mainPayment.xqy`** for the XQuery map file. The file must have a `.xqy` extension.
5. Leave the default directory name as is.
6. Define an XQuery function
   a. Make sure that Generate Function is selected. This will create a function in the XQuery file.
   b. Under Function Name, enter **`transPaymentStatus`**.

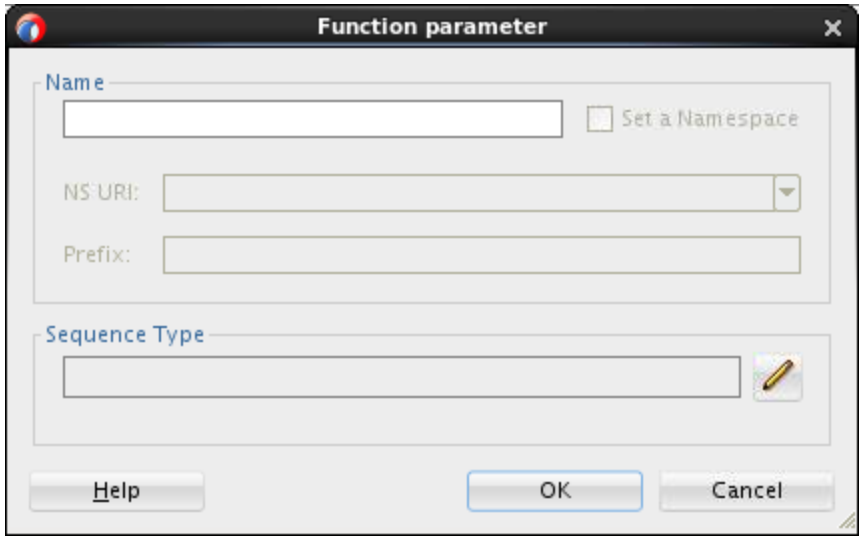
NS URI and Prefix are the additional fields for the function.

- NS URI specifies the namespace for the function.
- Prefix specifies the namespace prefix of the function.

c. Both fields are automatically populated. You can keep them as is.

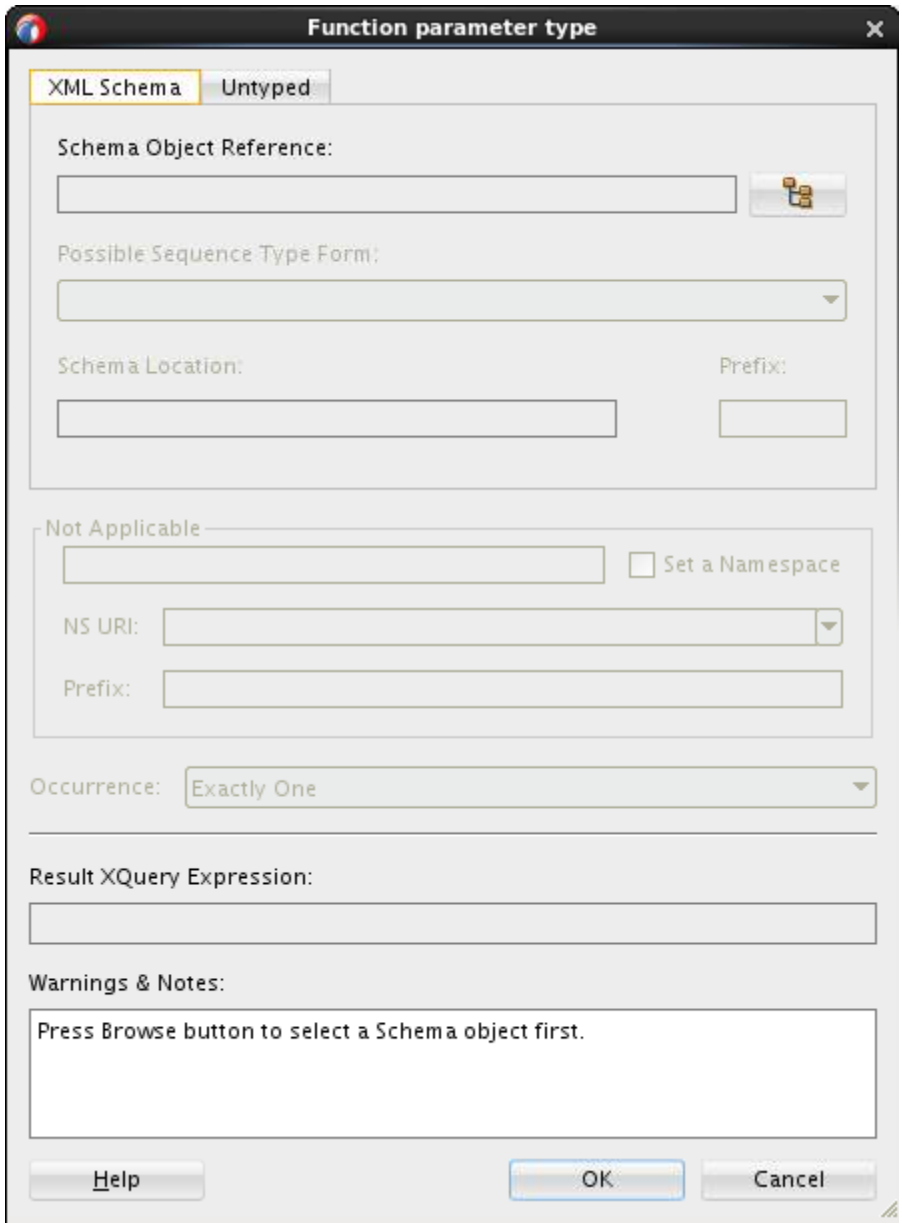
7. Add a parameter for the function's input sources.
   a. Under the Sources section, click the Add Source button identified by the green plus sign (+).

      The Function Parameter dialog box appears.



   b. Enter `partnerPayment` as the function parameter name.

c. Under Sequence Type, click the pencil icon to specify the data type for the parameter.
The Function Parameter Type dialog box appears.



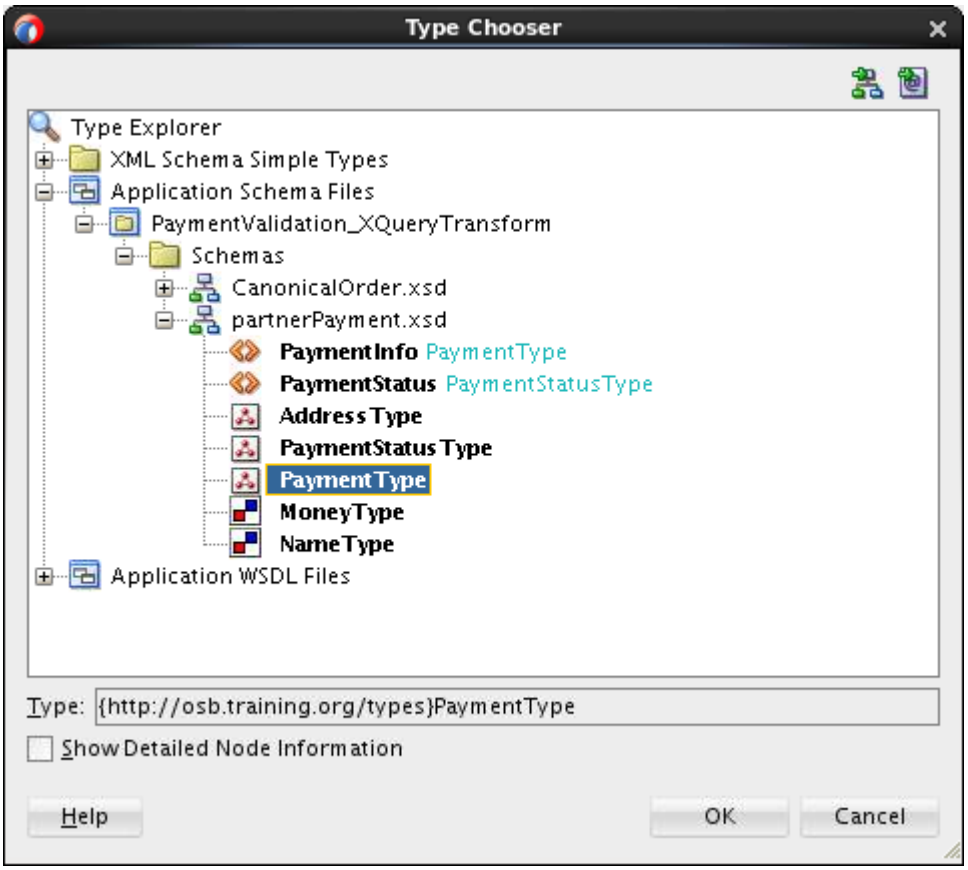
Use the XML Schema tab to specify an XML schema type as the data type for the function parameter. The Untyped tab can be used to specify an untyped (non-XML schema based) form of the parameter. You use an XML schema type in this lab.

1) Click the button to the right of Schema Object Reference to select a schema object as the data type.

   The Type Chooser dialog box appears.

   You can choose from Application Schema Files, XML Schema Simple Types, and schemas embedded in Application WSDL Files. You use Application Schema Files in this lab.

2) Navigate to partnerPayment.xsd, and select PaymentType as shown below:

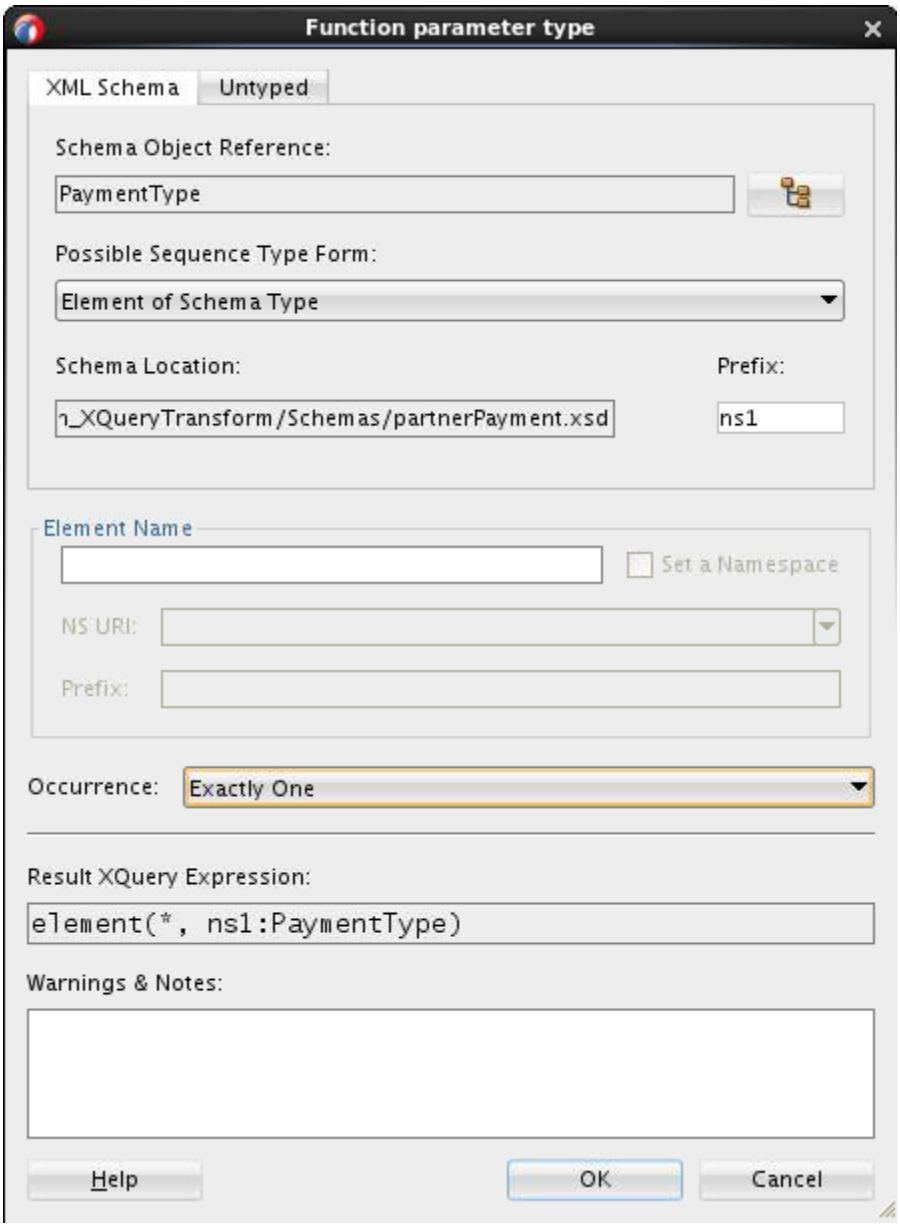


3) Click OK to close the Type Chooser dialog box.

   In the Function Parameter Type dialog, the Possible Sequence Type Form, Schema Location, and Prefix are automatically populated depending on your choice of Schema Object Reference.

4) Under Occurrence, keep the default value: Exactly One.

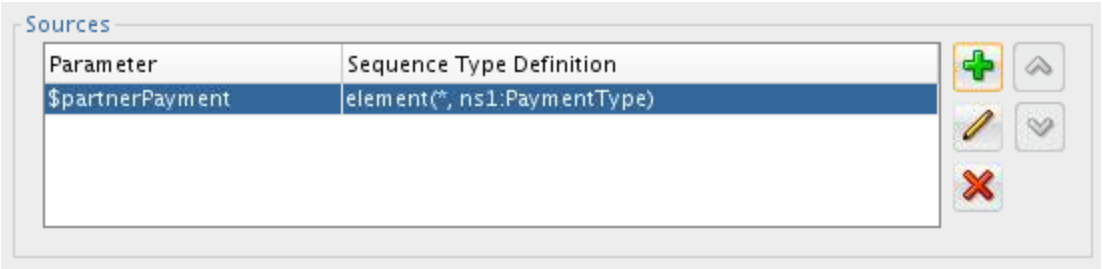
5) The resulting XQuery expression appears under Result XQuery Expression.



6) Click OK to close the Function Parameter Type dialog box.

d. Click OK to close the Function Parameter dialog box.

At this point, you finished defining the parameter for the function's input source.



8. In the Create XQuery Map Main Module dialog, specify the function's result data type.

   Click the button to the right of the Target field.

   The Function Result Type dialog box appears. This dialog box is identical to the Function Parameter Type dialog box.

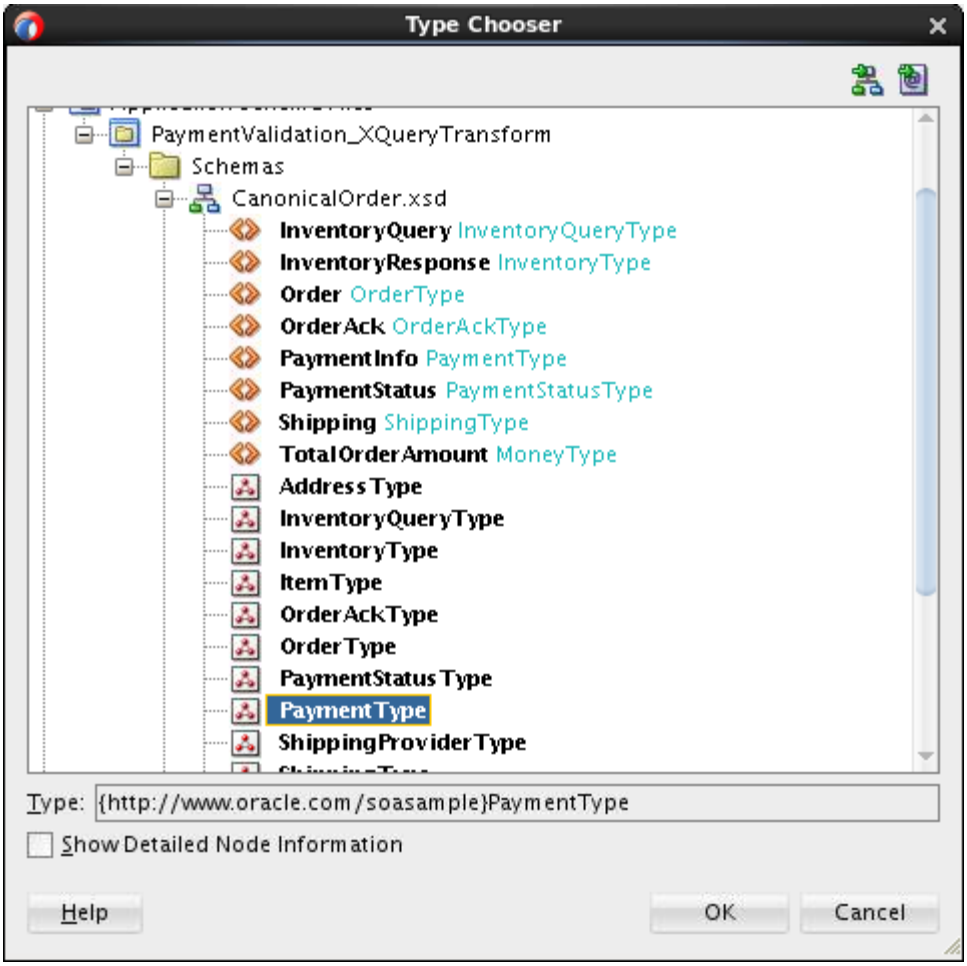
   You can use instructions under Step 7-c to specify the function's result data type.

   The steps are detailed below for your reference.

   a. Click the button to the right of Schema Object Reference to select a schema object as the data type.

      The Type Chooser dialog box appears.

   b. Select PaymentType in the CanonicalOrder.xsd file as shown below:



   c. Click OK to close the Type Chooser dialog box.

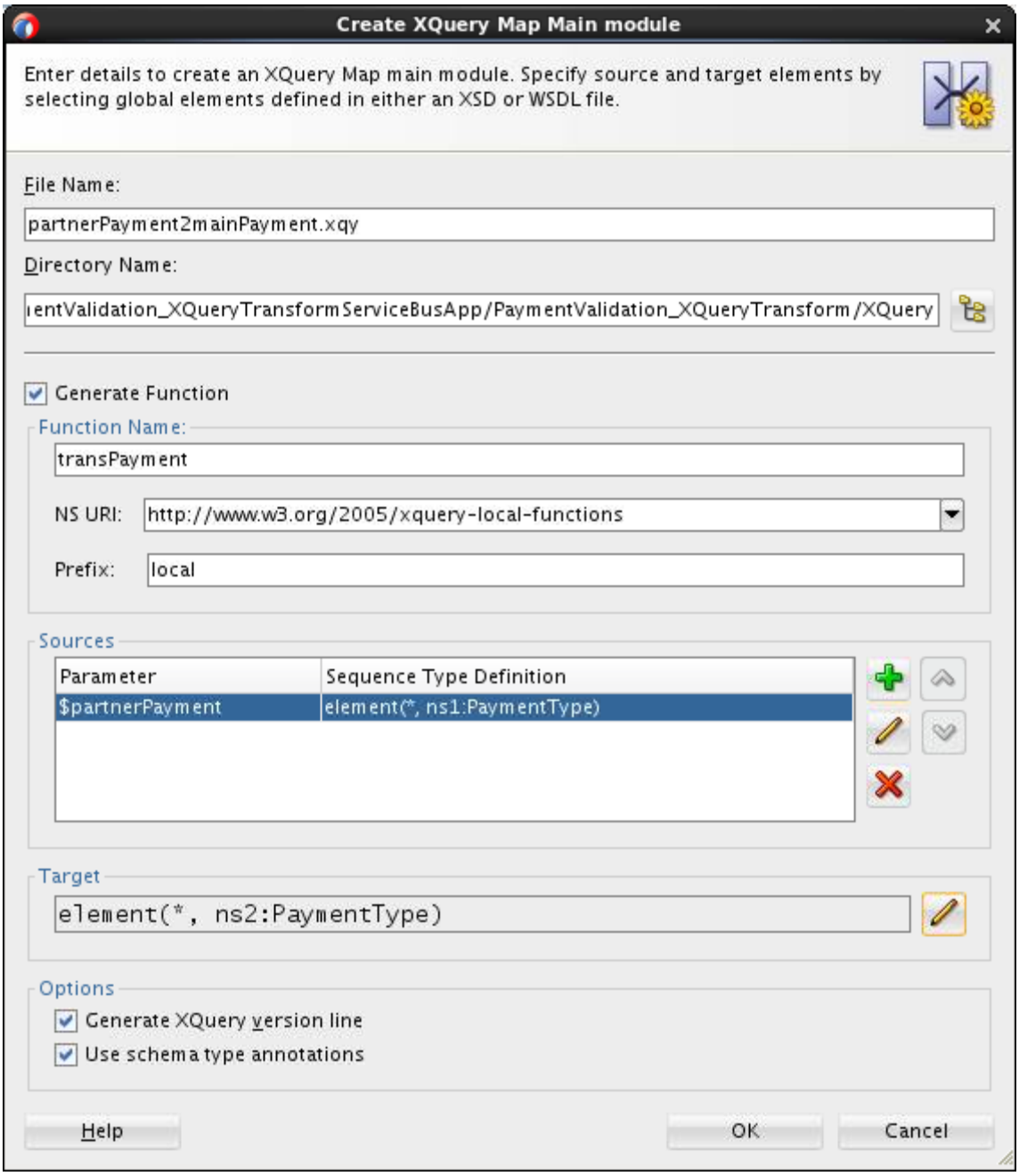
      In the Function Parameter Type dialog, the Possible Sequence Type Form, Schema Location, and Prefix are automatically populated depending on your choice of Schema Object Reference.

   d. Under Occurrence, keep the default value: Exactly One.

      The resultant XQuery expression appears under Result XQuery Expression.

e. Click OK to close the Function Result Type dialog box.

In the Create XQuery Map Main Module/Library Module dialog box, the Target field is populated with the type that you selected.

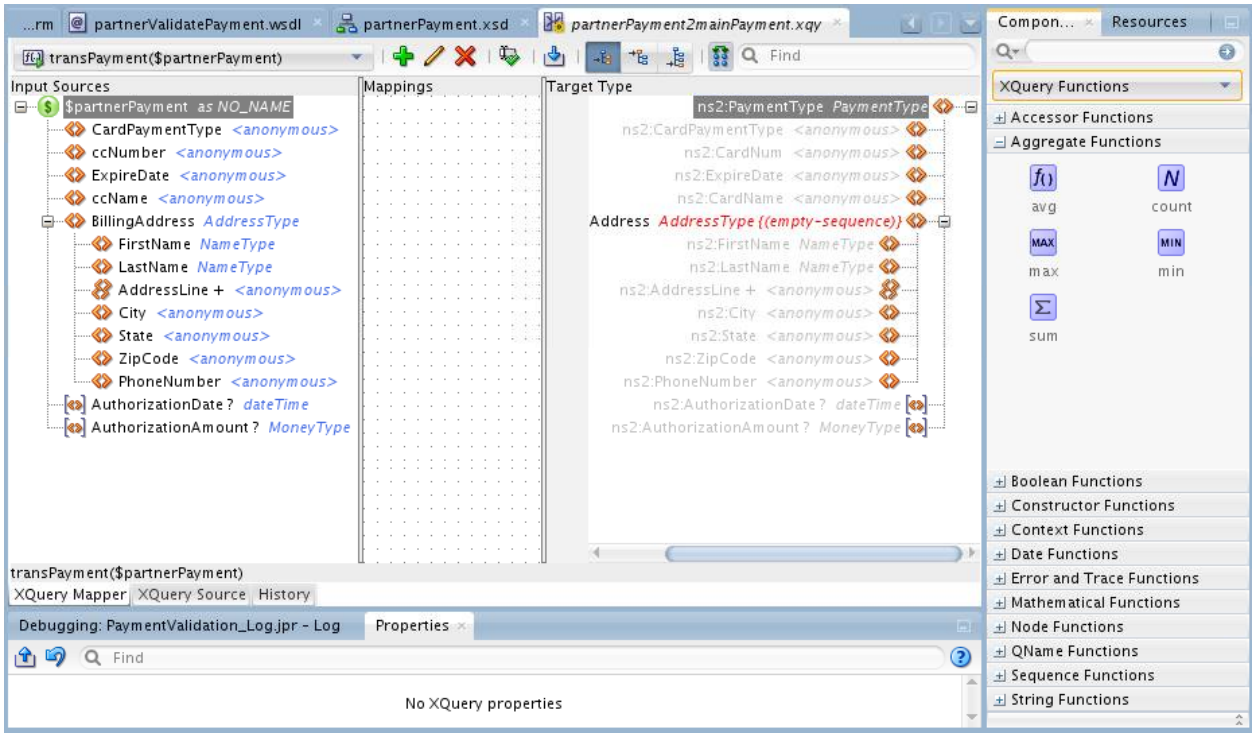


9. Under the Options section, make sure that the Generate XQuery version line and Use schema type annotations are selected.
10. Click OK.

    The newly created XQuery map opens up in the XQuery Mapper graphical view. You see the map file appear in the XQuery folder under Project Explorer.

Map source to target using XQuery Mapper

11. Expand the elements in both Input Sources and Target Type panes.

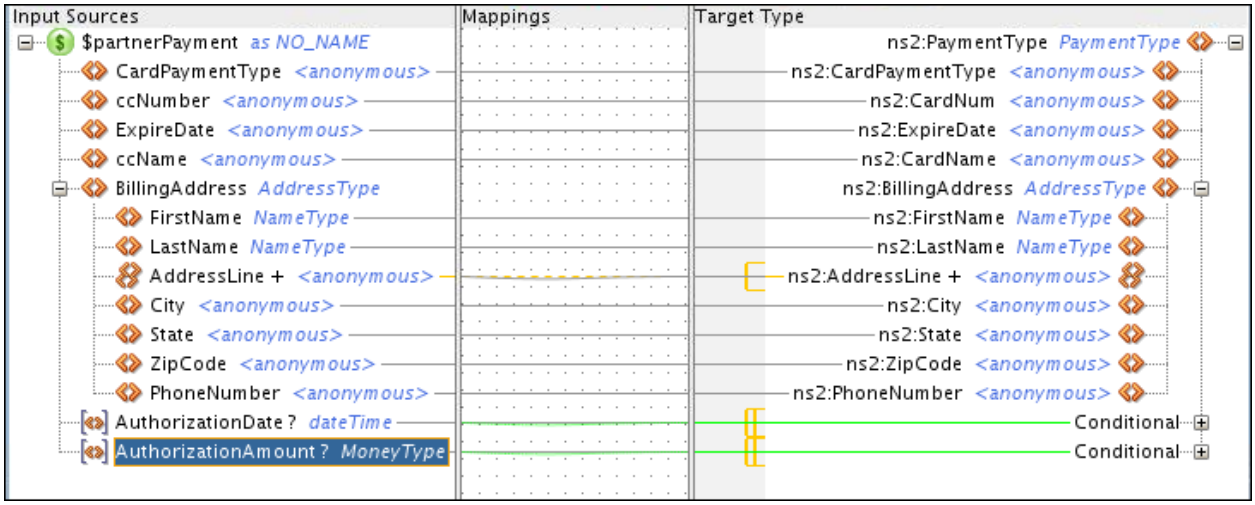


This is very simple, basic mapping, and there is no need to use XQuery Functions, XQuery Constructs, and XQuery Operators.

12. Manually map the leaf nodes (Drag the element in the source column and drop it onto the element in the target column). Elements in the table below are the ones that have different name in source and target:

Source	Target
ccNumber	cardNum
ccName	cardName

13. Verify and save your work.



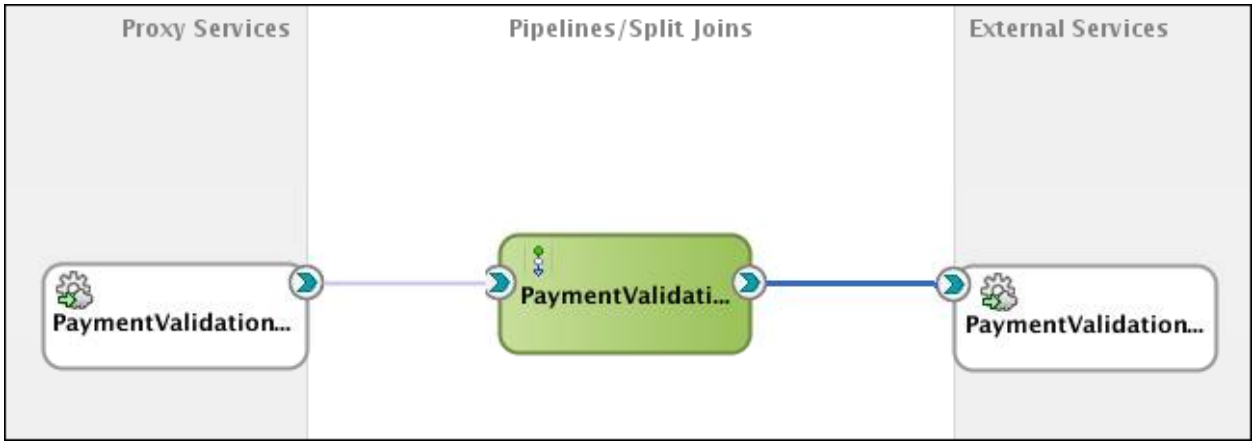
Create Proxy Service and Configure Pipeline

14. Create the proxy service
    a. In Overview Editor, drag the HTTP component from the Components palette to the Proxy Services lane.

       The Create Proxy Service wizard pops up.
    b. In step one (Create Service), fill up the details for Proxy Service:
       - Service Name: **PaymentValidation_XQueryPS**
       - Location: (keep as is)
       - Transport: http (keep as is)
       - Keep “Generate Pipeline” checked for creating pipeline along with Proxy Service
       - Change the pipeline name to **PaymentValidation_XQueryPP**

       Click **Next**.
    c. In step two (Type), use the same WSDL of the business service for the proxy service:
       - Select service type **WSDL**.
       - Click the **Browse WSDLs** icon on the right.

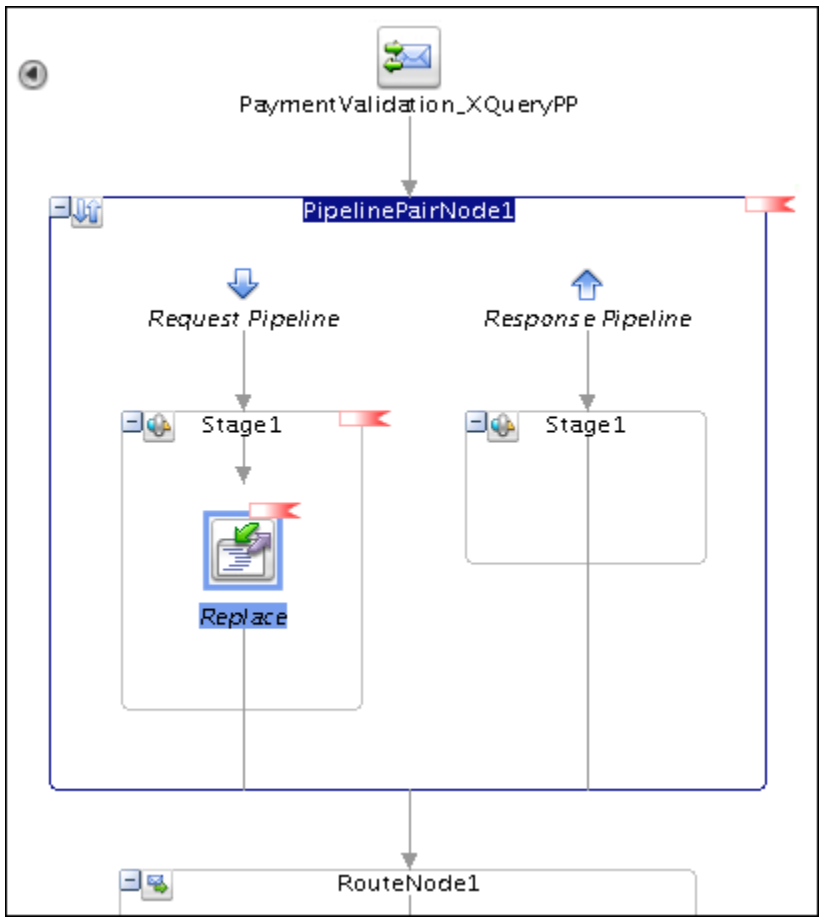
         The Select WSDL dialog box pops up.
       - Navigate to Application > PaymentValidation_XQueryTransform > WSDLs directory, and select **partnerValidatePayment.wsdl**.
       - Click **OK**.
       - Click **Next**.
    d. In step three (Transport), you should see the populated endpoint URI. Keep as is and click **Finish**.
15. In Overview Editor, wire the pipeline to the business service.



16. Configure the pipeline service
    a. Double-click the PaymentValidation_XQueryPP to open the pipeline editor.
    b. Drag Pipeline Pair from Nodes section in the Components palette and drop onto the yellow circle below the Start node.

       You should see the Pipeline Pair node, comprising a Request Pipeline and a Response Pipeline, being added to the start node.

c. Drag a Replace action (in Message Processing section) and drop onto Stage1 on the Request Pipeline.



d. To replace the request message payload to the transformed message, do as follows:

1) Select the source Xpath:

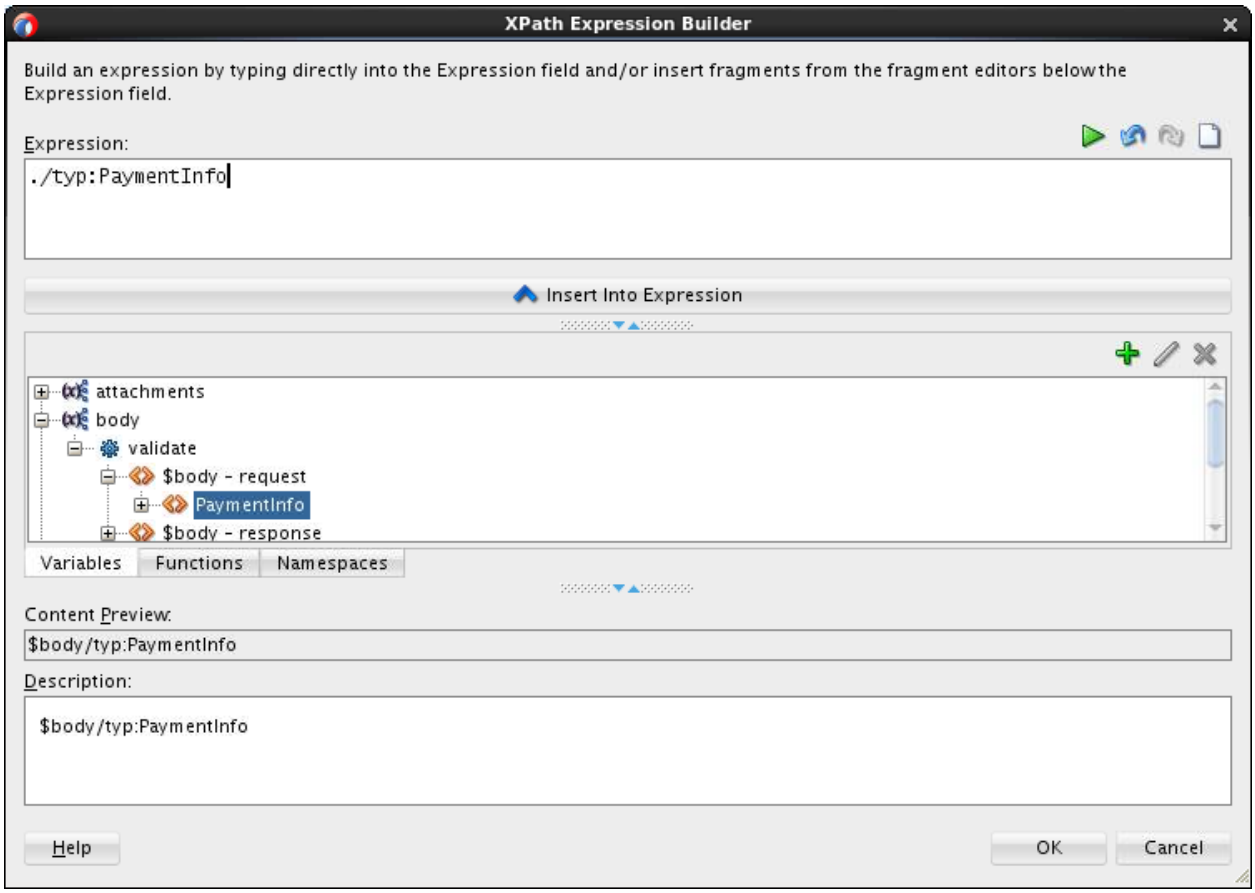
a) In the Replace Properties inspector, for Location, select **body**.

b) Click  next to <XPath> to open XPath Expression Builder.

c) Navigate to body > validate > body – request, select **PaymentInfo** element.

d) Click the **Insert Into Expression** button.

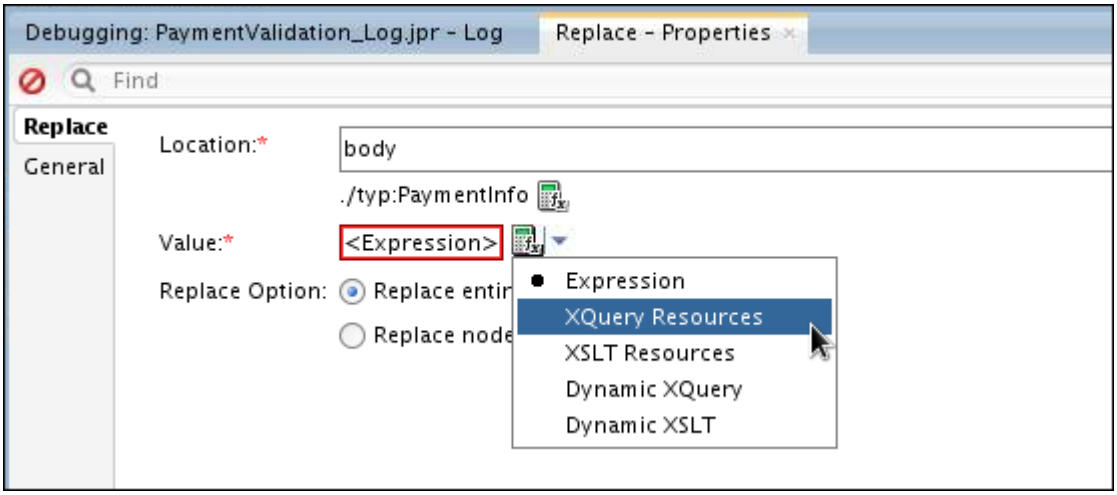
You see that the Expression field is populated with the XPath that you selected.



e) Click OK.

2) Select the replacement.

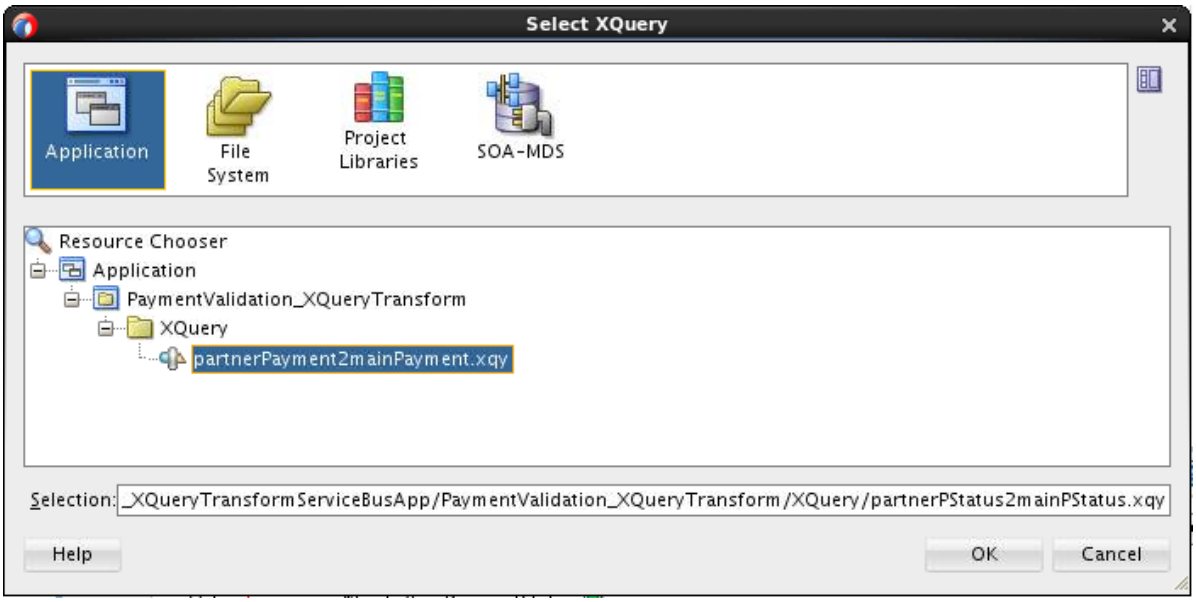
a) In the Replace Properties inspector, click the triangle shape next to <expression> to view the drop-down menu, and select Xquery Resources.



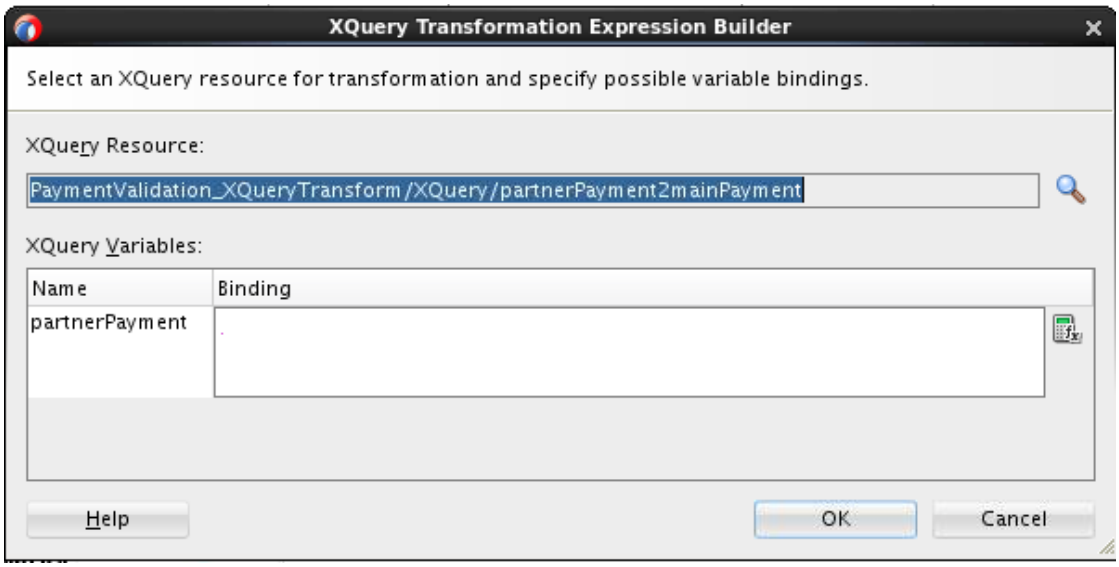
The XQuery Transformation Expression Builder window is displayed.

b) Click browse icon next to the XQuery Resource field.
The Select XQuery dialog box is displayed.

c) Select partnerPayment2mainPayment.xqy as the XQuery resource to use.



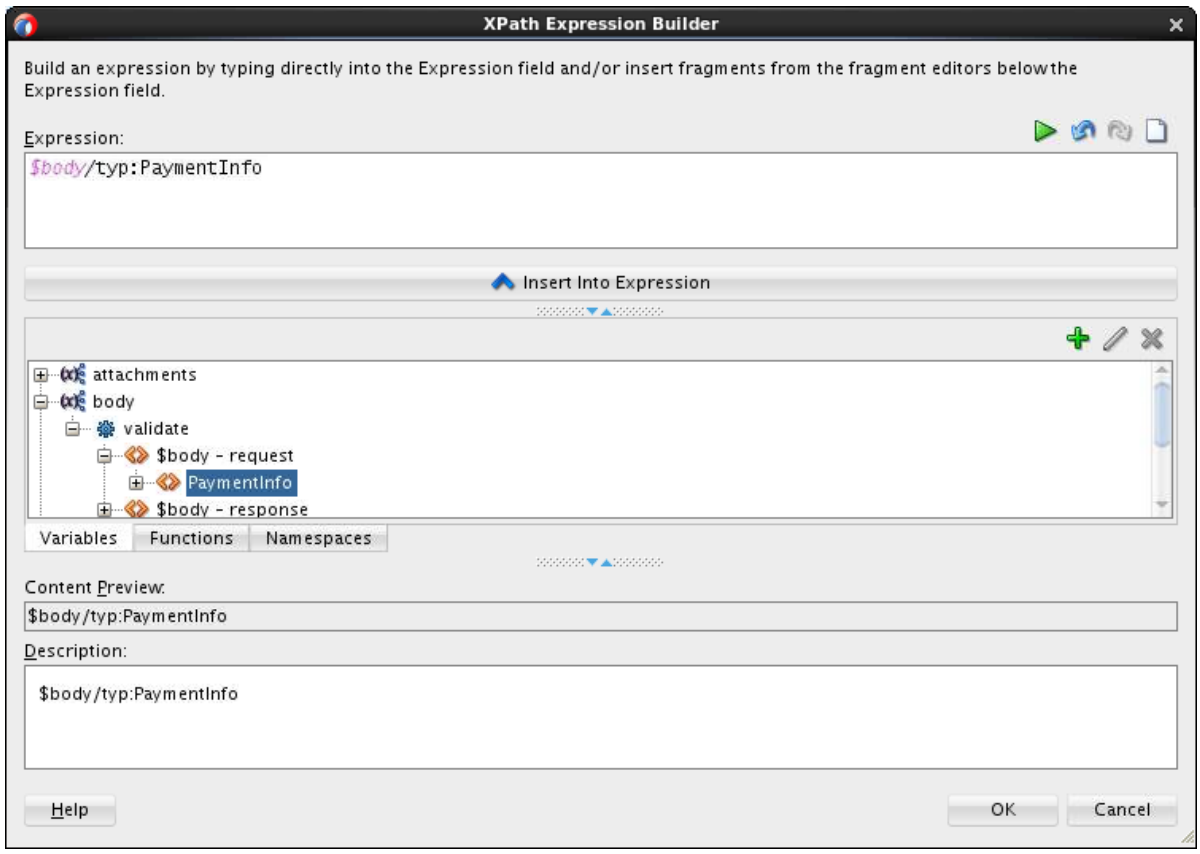
d) Click OK.
The XQuery Resource field is populated with your selection.



e) **Bind the input for the XQuery transformation:**

i) Under **XQuery Variables, click**  .
The XQuery Expression Builder window appears.

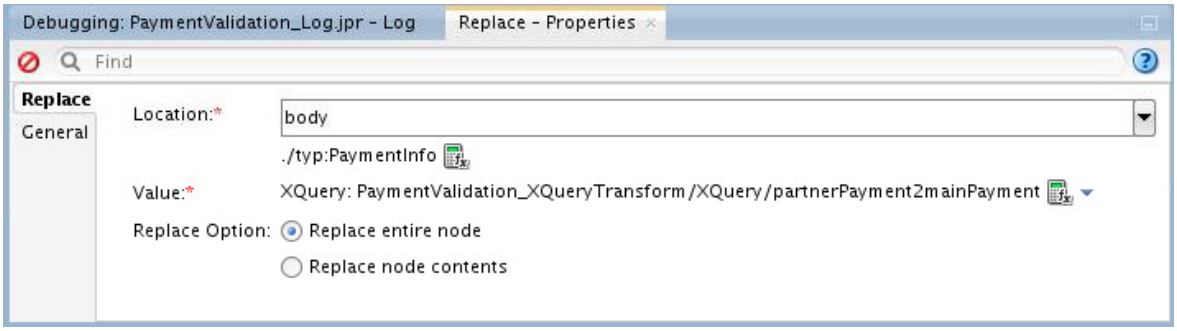
ii) Navigate to body > validate > body – request, select **PaymentInfo** element, and insert it into the Expression field.



This step is to specify PaymentInfo as the input document to give to the Query to perform the transformation upon.

iii) Click OK.

f) The Replace Properties window is updated with your selection.



g) Keep the Replace entire node option selected.

17. Save your project.

18. Deploy and test
    a. In application Overview Editor, right-click the PaymentValidation_XQueryPS proxy service. Select **Run**.

       The Test Console appears.
    b. Click the **Choose File** button.
    c. Navigate to `/home/oracle/labs_DI/resources/sample_input/`, and select **`partnerPaymentInfoSample.xml`**
    d. Click **Open**.
    e. On the Test Console, click the **Execute** button.

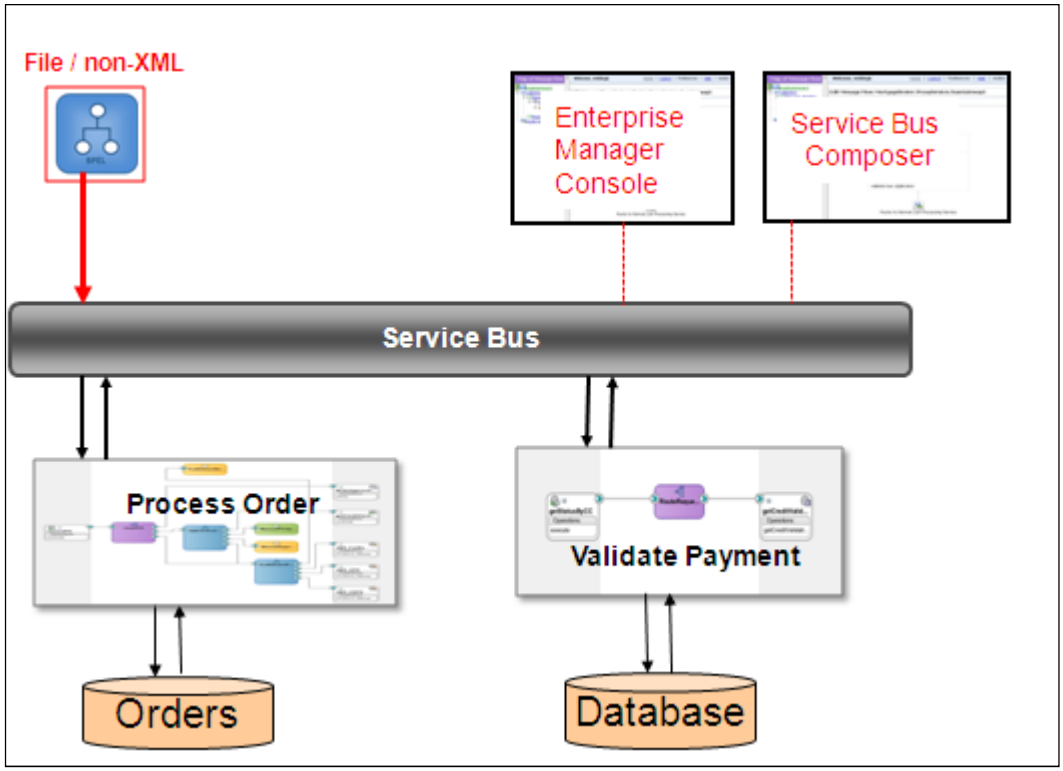
       The next screen in the Response Document section indicates that the payment has been Denied.
19. When you are done, close the application and all open windows of the project.

Practice 6-2: Configuring a File Adapter Proxy with nXSD Transformation

Overview

In this practice, you first deploy the ProcessOrder composite application. The composite has a web service interface that receives orders.

You can extend and add a new interface so that legacy systems may place orders without impacting the backend application. Service Bus can provide protocol and data translation to minimize any disruption to the backend business logic of ProcessOrder. To do that, you add a new file order channel for the ProcessOrder SOA service. The new proxy handles incoming orders by file and translates common-delimited format with nXSD.



Tasks

Setup

Before you create Service Bus application, you need to deploy and test the ProcessOrder composite.

Deploy the composite

1. In a terminal window, enter the following commands:

```
>cd $HOME/labs_DI/lessons/lesson06
>ls -l
>$HOME/labs_DI/resources/Scripts/deploy.sh
sca_ProcessOrder_rev1.0.jar
```

2. The build and deployment information will be displayed in the console. You should see a message showing both the build and the deploying executed successfully.

Test the deployed composite

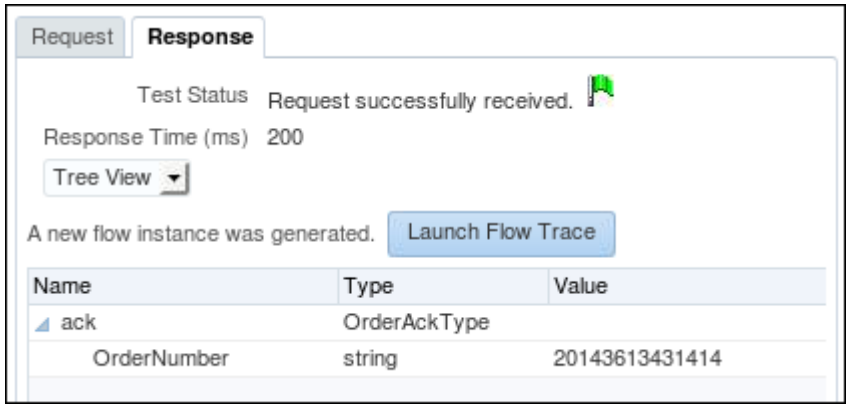
This ProcessOrder composite is implemented with the following functionalities:

- Receive an order through a web service call.
- Create an order number, set the order date to the current date, and set the order status to New.
- Calculate the total order amount.
- Save the order in the database with the status New.
- Return an acknowledgement to the client with the order number.

3. Test in EM.
   a. Open the EM console. In the Target Navigation panel, expand SOA > soa-infra > default. Now you should see the newly deployed ProcessOrder composite application in the folder.
   b. Select the ProcessOrder composite.
   c. Click the **Test** button at the top of the screen.

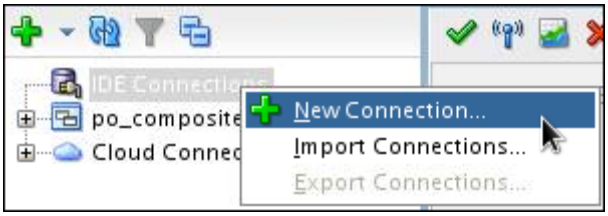
      The Test page opens.
   d. Scroll down to the **Input Arguments** section.
   e. Click **Browse**. In the File Upload window, select the `$HOME/labs_DI resources/sample_input/OrderSample.xml` file, and open it.

      The Order element in the SOAP Body area is populated with the order data.
   f. Click **Test Web Service**. When the composite completes, the screen switches to the Response tab showing an order number (similar to the following screenshot).



4. Verify the outcome of the test in the database.

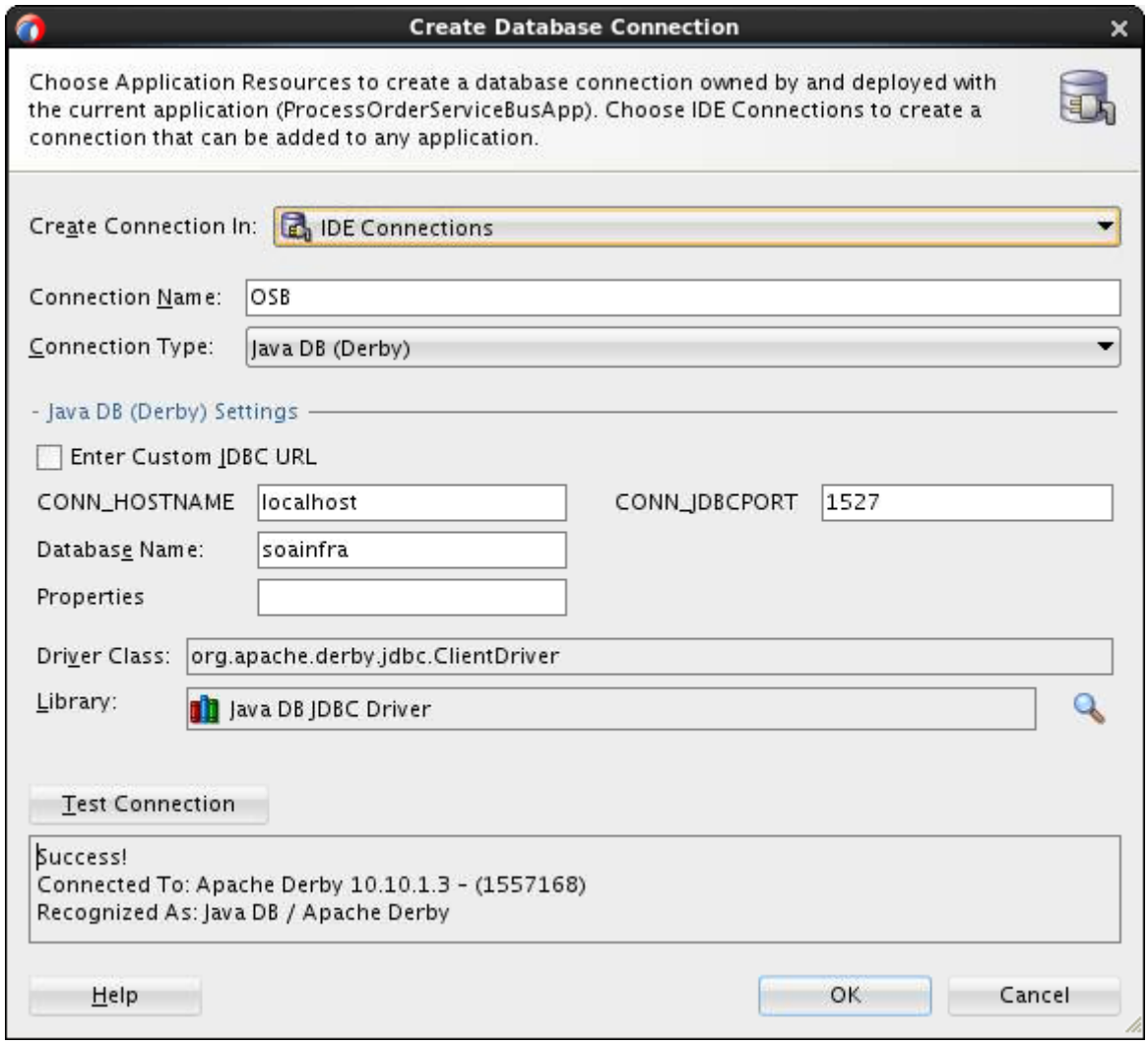
   To see whether the new order record has been created in the database, perform the following steps:
   a. Create a database connection in JDeveloper.
      1) From the JDeveloper main menu, select Window > Database >Databases. The Databases window is displayed.
      2) Right-click **IDE Connections**, and select **New Connection…** from the context menu.



3) In the Create Database Connection dialog, enter the following details:
   - Connection Name: **`OSB`**
   - Connection Type: **`Java DB (Derby)`**
4) Verify the default Java DB (Derby) settings match these:
   - Server Name: `localhost`
   - Port: `1527`
   - Database: `soainfra`
   - Driver Class: `org.apache.derby.jdbc.ClientDriver`
   - Library: `Java DB JDBC Driver`

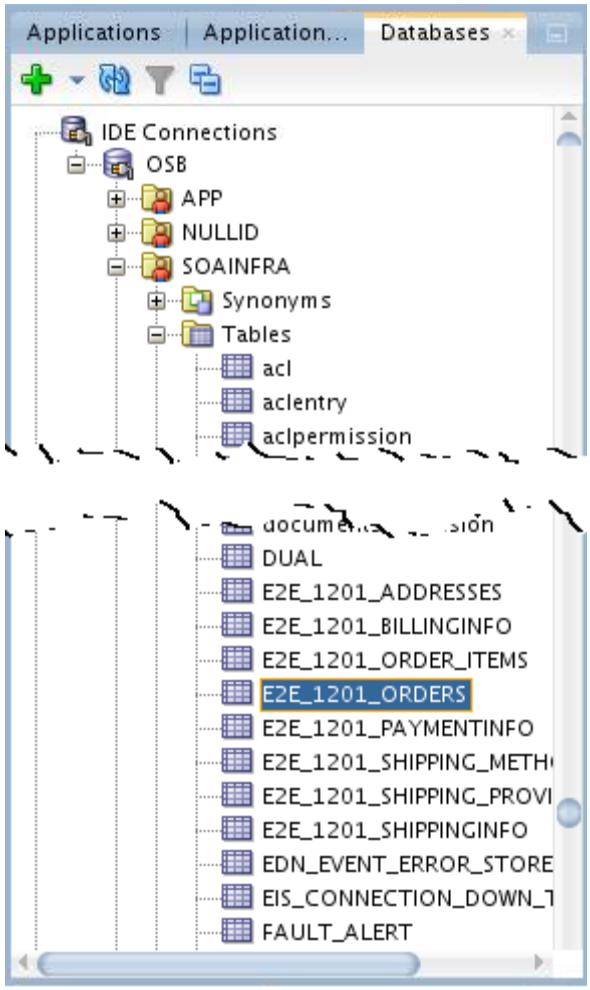
   Note: If the Library field is empty, click the Browse icon, locate the **Java DB JDBC Driver** library (under the Extension folder), and click OK.

5) Click the Test Connection button to verify that the connection works.
You see that Success! is displayed.

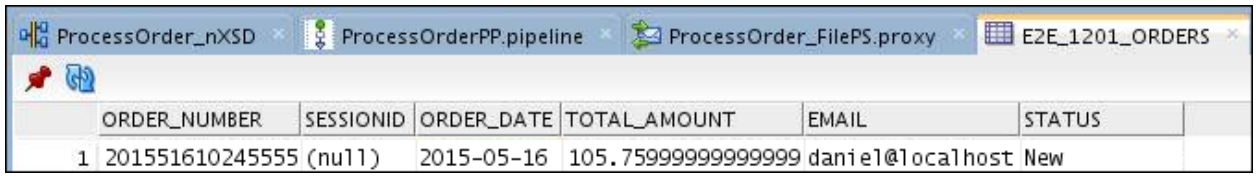


6) Click OK.
You see the new DB connection appear under the IDE folder.

b. Expand OSB folder, navigate to SOAINFRA > Tables, and locate the E2E_1201_ORDERS table in the list.



c. Right-click the table, and select **Open Object Viewer** from the context menu. You should see the new order record saved in the database.



Leave this tab window open. Revisit it later.

Importing Service Bus project

5. Create a new Service Bus application.
   a. Select **File > New > From Gallery** from the menu.
      The New Gallery dialog box opens.
   b. In Categories, select **Service Bus Tier**.
   c. Under Items, select **Service Bus Application**.
   d. Click **OK**.
      The Create Service Bus Application dialog box opens.
   e. Specify the application name to `ProcessOrder_nXSDServiceBusApp`.

f. Enter `/home/oracle/labs_DI/lessons/lesson06/ProcessOrder_nXSDServiceBusApp` for the directory.

g. Click **Finish**.

The new Service Bus application folder is created.

At this point, you created an empty Service Bus application- a placeholder. Next, you import a pre-built project with resources into this application.

6. Import project.

h. Select **File > Import** from the menu.

i. In the Import dialog, select **Service Bus Resources**.

j. Click OK.

The Import Service Bus Resources wizard is displayed.

k. Select **Configuration Jar**.

l. Click **Next**.

The wizard advances to step two.

m. Click the **Browse** icon to the right of Jar Source.

The Configuration Jar chooser is displayed.

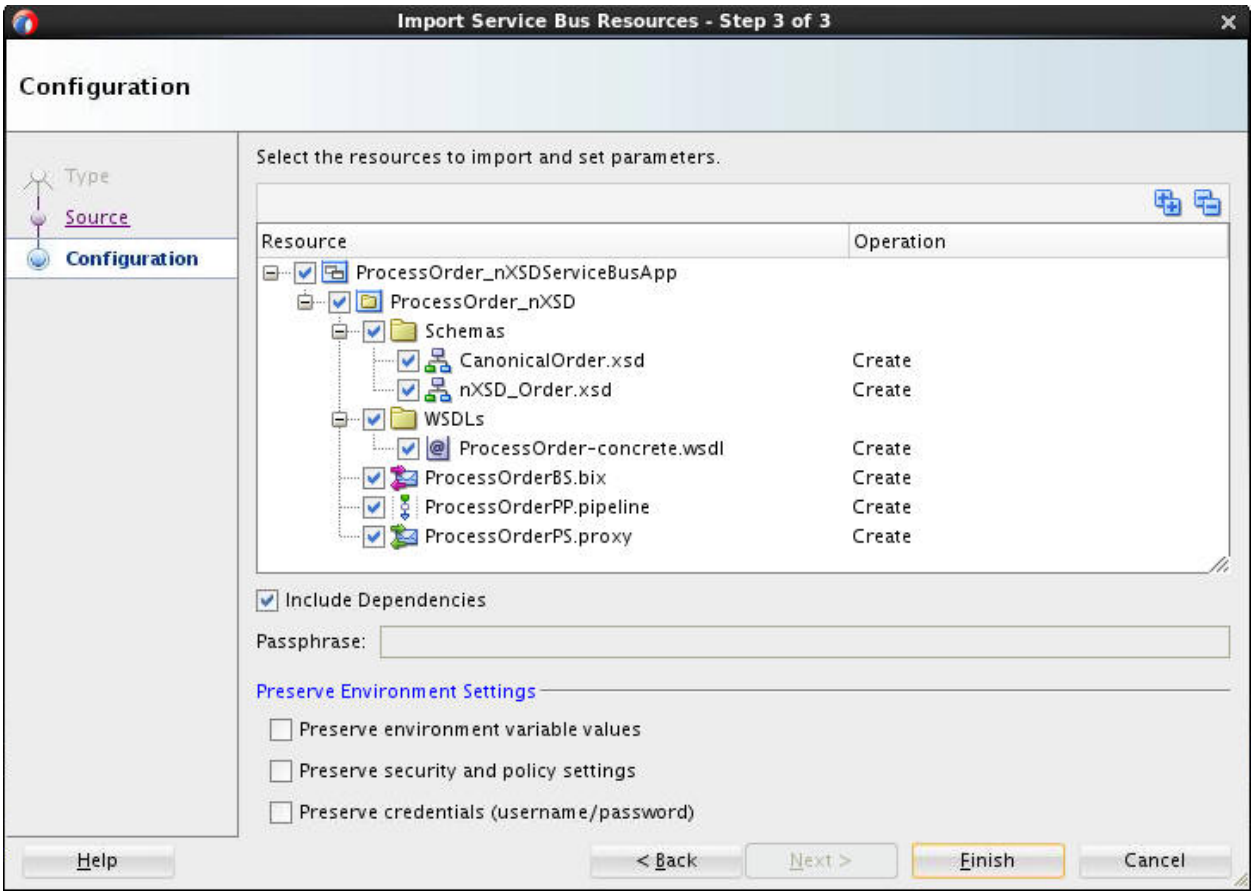
n. Navigate to /home/oracle/labs_DI/lessons/lesson06/, select **ProcessOrder_Resources.jar**.

o. Click **Open**.

p. Click **Next**.

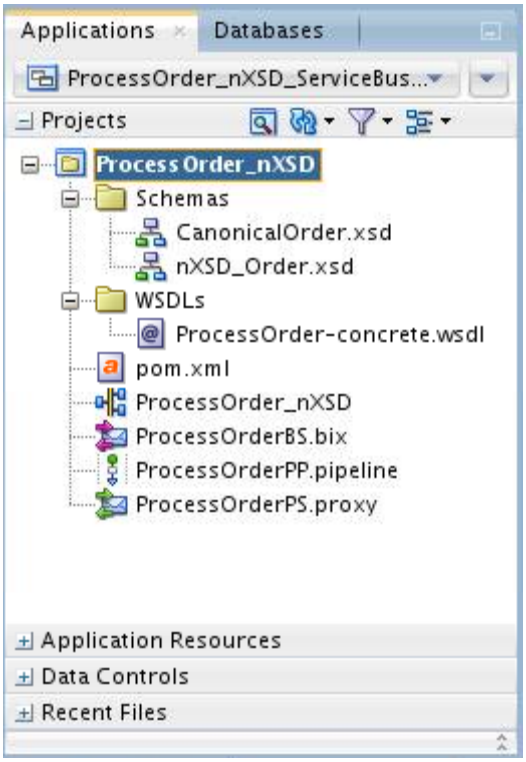
The wizard advances to step three.

q. Review what is imported into the application.



r. Click **Finish**.

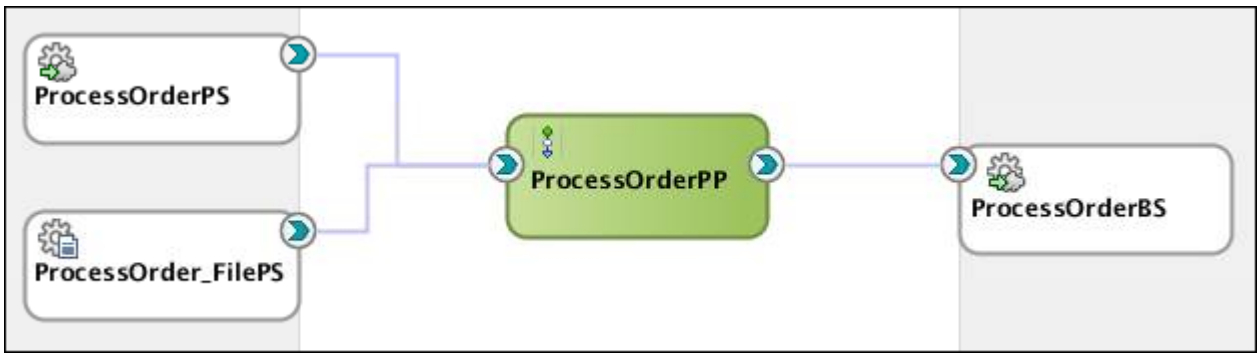
s. Your Application Navigator should now resemble the following:



Add file adapter proxy and wire to pipeline

You use the File Adapter wizard to add a new File-based proxy for ProcessOrder. The new File proxy will transform a comma-delimited file to a valid Order xml which the ProcessOrder pipeline service can use, validate it, and call the backend ProcessOrder SOA service.

At the end of this practice, your ProcessOrder_nXSD project should resemble the following:

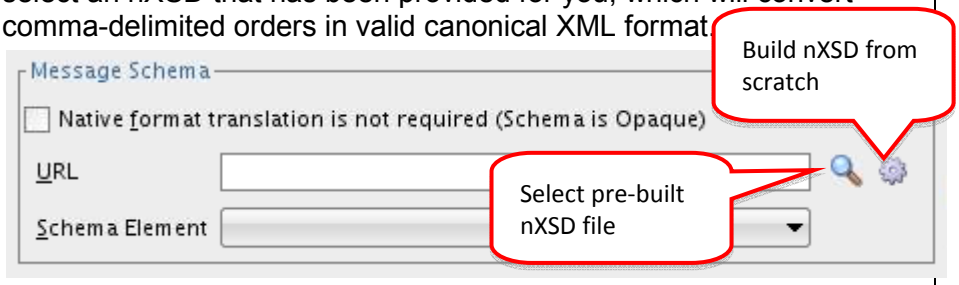


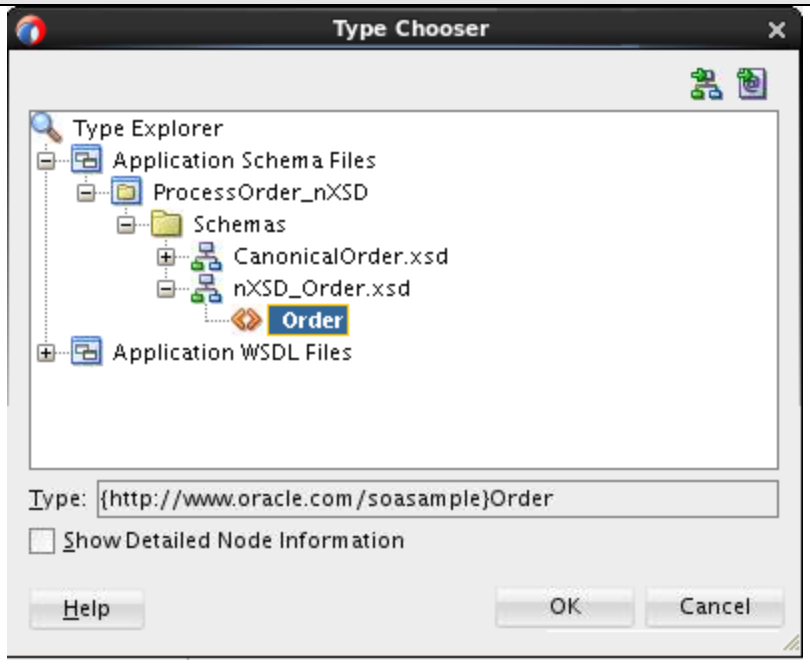
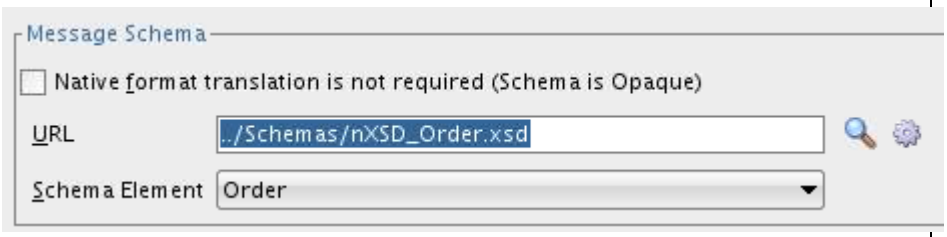
Note: In SOA Suite 12*c*, the Adapter wizards are shared between SOA and Service Bus.

7. Double-click the project overview file ProcessOrder_nXSD to open it in the Overview editor.
8. Locate the File Adapter icon in the Component Palette, and drag it onto the Service Bus Overview editor in the Proxy Services lane.

   File Adapter Configuration Wizard opens.

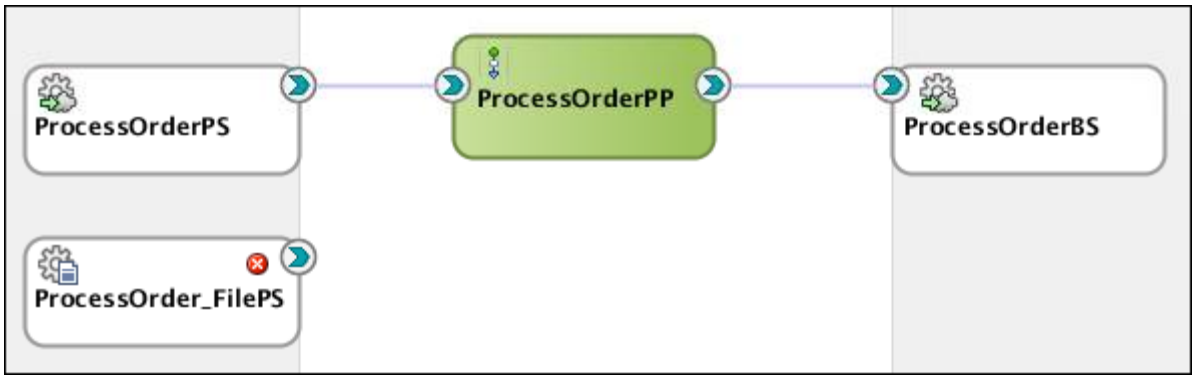
9. To configure the File adapter, use the instructions listed in the following table:

Step	Window Description	Choices or Values
a.	File Adapter Service	Name: **`ProcessOrder_FilePS`** Location: Leave as default Click Next.
b.	Adapter Interface	Accept the default value: Define from operation and schema (specified later). Click Next.
c.	File Server Connection	Accept the default value: eis/FileAdapter. Click Next.
d.	Operation Type	Select the Read File radio button. Click Next.
e.	File Directories	Use the Browse button to specify the Directory for Incoming Files: **`/home/oracle/labs_DI/tmp/input`** Accept the rest default configurations Click Next.
f.	File Filtering	Include Files with Name Pattern: **`*.csv`** Click Next.
g.	File Polling	Polling Frequency: **20 seconds** Click Next.
h.	Messages	In this step, you have a choice. You can proceed to Appendix–Build the nXSD translation to create the nXSD from scratch or you may select an nXSD that has been provided for you, which will convert comma-delimited orders in valid canonical XML format.  **Note:** The following instructions are for using pre-built nXSD. However, if you are running ahead of schedule, you should try to build from scratch. To select pre-built nXSD, click the browse button to the left. When the Type Chooser appears, navigate to Application Schema Files > ProcessOrder_nXSD > Schemas > nXSD_Order.xsd, select **Order**.

Step	Window Description	Choices or Values
		 Click OK. Notice that the URL field is populated with the schema file that you selected and the Schema Element is set to Order.  Click Next.
i.	Finish	Click Finish.

The Adapter Configuration wizard closes.

Your Overview editor should resemble the following:



10. Wire your new proxy service to the existing ProcessOrderPP pipeline (Click the small green arrow on your new proxy and drag over to the ProcessOrderPP pipeline.)

11. Save your project.

Deploy and Test

12. Deploy the project.
13. Test with file order.

    Use vi or gedit to open the sample test file, `FileOrderSample.csv`, located in the `~/labs_DI/resources/sample_input/` folder. The file appears as shown below:

```
0,4321432143214321,0316,VISA
Lawrence,Jennifer,9876 Beverly Lane,Hollywood,CA,12345,5127691108
Lawrence,Jennifer,9876 Beverly Lane,Hollywood,CA,12345,5127691108
Standard,jl@localhost
32779,5,4.72,Slacker,Water bottle,2008,Slacker Water Bottle
30421,16,10.72,Grand Alpine,Bicycle Tires,2005,Grand Alpine Bicycle
Tires
32861,11,60.72,Safe-T,Bicycle helmet,1829,Safe-T Helmet
```

    a. Open a terminal window. Navigate to the file input folder:

       `cd $HOME/labs_DI/tmp/input`

    b. Copy the test file to this folder:

       `cp $HOME/labs_DI/resources/sample_input/FileOrderSample.csv.`

       Or

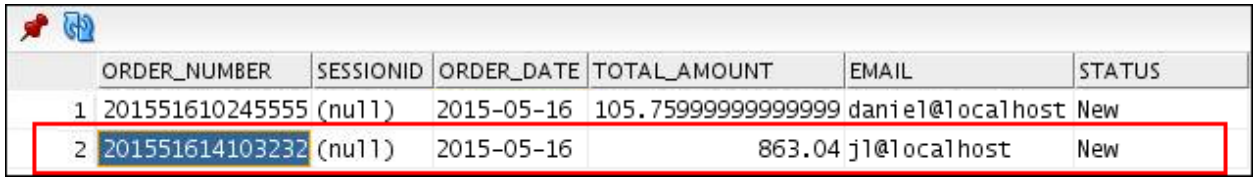
       You can open these two folders on the desktop and then do a copy-paste of the `FileOrderSample.csv` file.

    Warning: You *must copy (*do not *move)* the file because, the input file adapter removes the file from the input folder after it has been read (consumed) by the file adapter service.

14. In the `input` folder, after a few moments, the file is gone, meaning that it has been processed.
15. You can verify the result by checking the database to see whether the new order record has been created. Perform the following steps:

    a. In JDeveloper, go back to the E2E_1201_ORDERS table's Object Viewer tab window.

    b. Click the Refresh button. You should see the new order entry in the table as shown below:



	ORDER_NUMBER	SESSIONID	ORDER_DATE	TOTAL_AMOUNT	EMAIL	STATUS
1	201551610245555	(null)	2015-05-16	105.75999999999999	daniel@localhost	New
2	201551614103232	(null)	2015-05-16	863.04	jl@localhost	New

    This new order is received from the file adapter, transformed using nXSD, and then processed by the ProcessOrder SOA service.

16. When you are done, close the application and all open windows of the project in JDeveloper.

Appendix: Building the nXSD Transformation from the Beginning

Overview

When you click the Define Schema for Native Format button in the Messages page of the adapter Configuration Wizard shown in the figure below, the Native Format Builder wizard is displayed. The Messages page is the last page that is displayed in the adapter Configuration Wizard before the Finish page.

In this use case, the Native Format Builder uses `FileOrderSample.csv`, a complex file type which contains multiple records such as credit card information, billing address, shipping address and items. Also, using this use case, you can generate the nXSD and test it.

The data in a sample text file, ~/labs_DI/resources/sample_input/FileOrderSample.csv, appears as below:

```
0,4321432143214321,0316,VISA
Lawrence,Jennifer,9876 Beverly Lane,Hollywood,CA,12345,5127691108
Lawrence,Jennifer,9876 Beverly Lane,Hollywood,CA,12345,5127691108
Standard,jl@localhost
32779,5,4.72,Slacker,Water bottle,2008,Slacker Water Bottle
30421,16,10.72,Grand Alpine,Bicycle Tires,2005,Grand Alpine Bicycle
Tires
32861,11,60.72,Safe-T,Bicycle helmet,1829,Safe-T Helmet
```

High-Level Steps

- Invoke the Native Format Builder from the file adapter Configuration Wizard (Step 8 of 9).
- Build complex map.

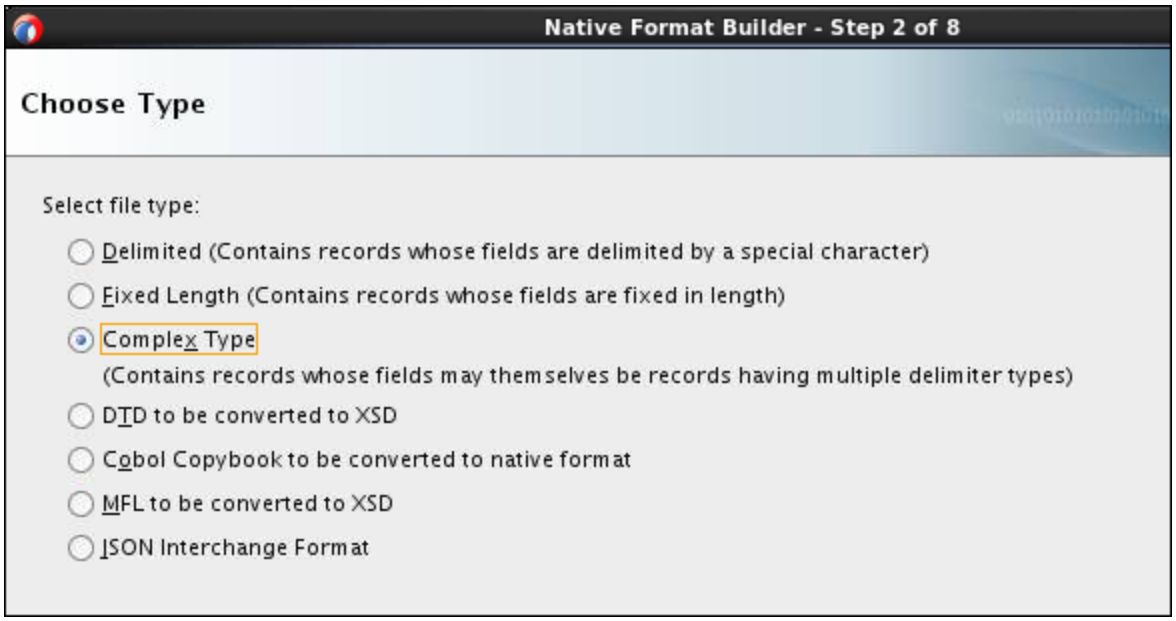
Configure the nXSD

1. File Adapter Configuration Wizard Step 8 of 9, click the “Define Schema for Native Format” (gear) icon

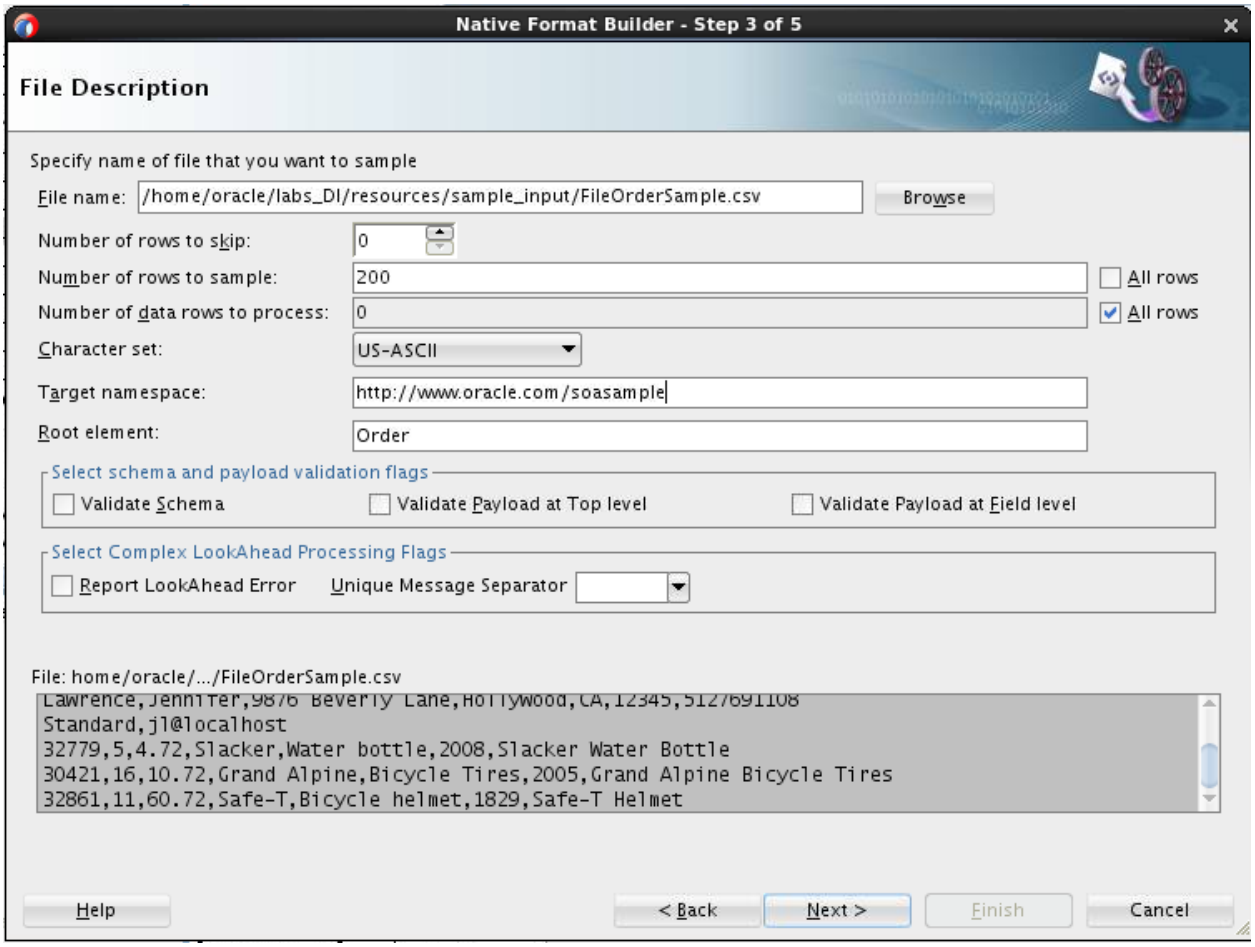
   The Native Format Builder Welcome page is displayed.
2. Click Next.
3. Name the file `order_nxsd.xsd`.
4. Click Next.

   The Choose Type page is displayed.

5. Select Complex Type.



6. Click Next.
   The File Description page is displayed.
7. Enter the following values:
   - Click Browse and select `FileOrderSample.csv`.
   - Root element: `Order`
   - Target namespace: `http://www.oracle.com/soasample`.



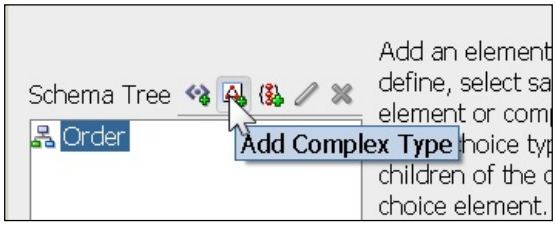
8. Click Next.

   The Design Schema screen is displayed. We will now create the complex types for data corresponding to address, billing, shipping, and item. You start by creating Complex type for address.

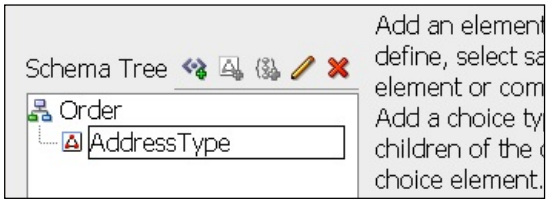
9. Add a new complex type.

   a. Select the Order element.

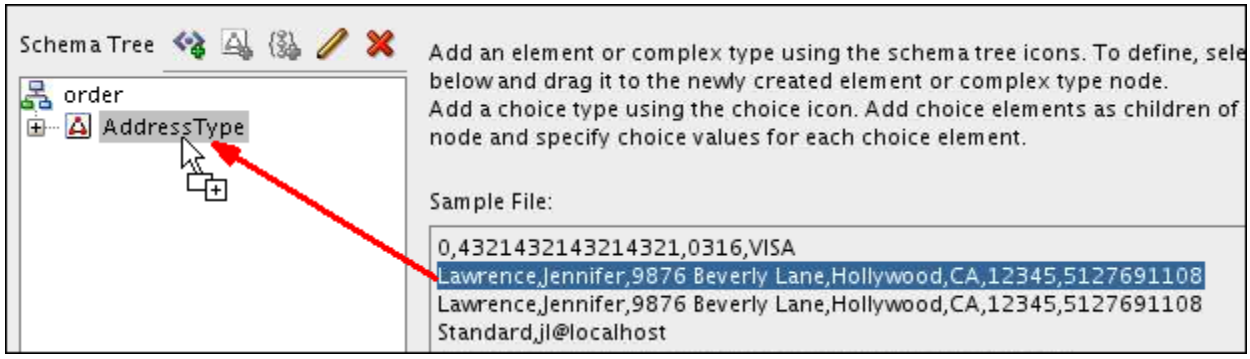
   b. Click the Add Complex Type icon.



   c. Rename <new_complex_type> `AddressType`.

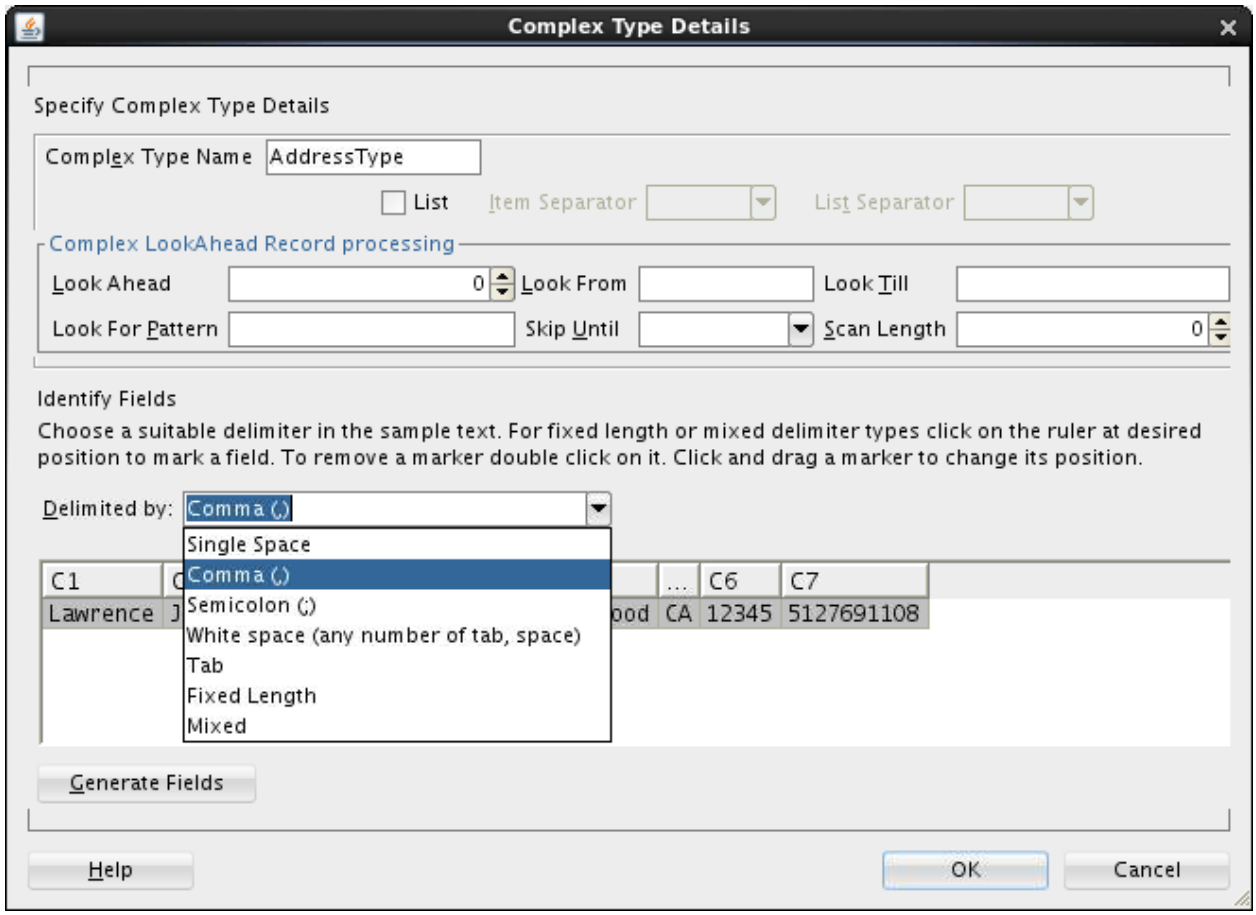


d. Drag the address line from the right pane for “Sample File” onto the `AddressType`.



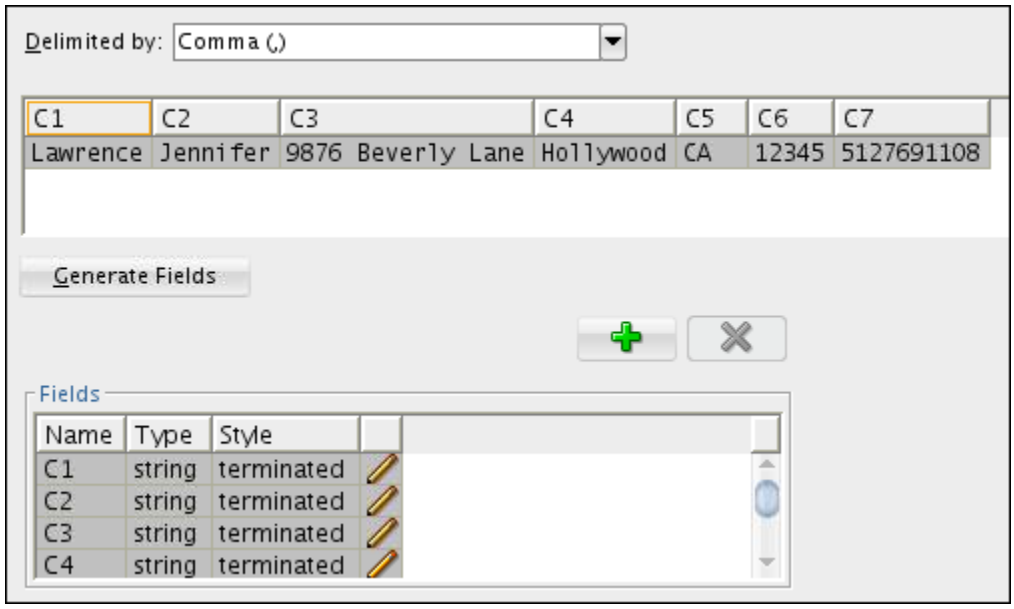
The Complex Type Detail screen is displayed.

e. Select Comma (,) from the Delimited by drop down.



f. Click the Generate Fields button.

Schema elements corresponding to the fields in the data file are generated.

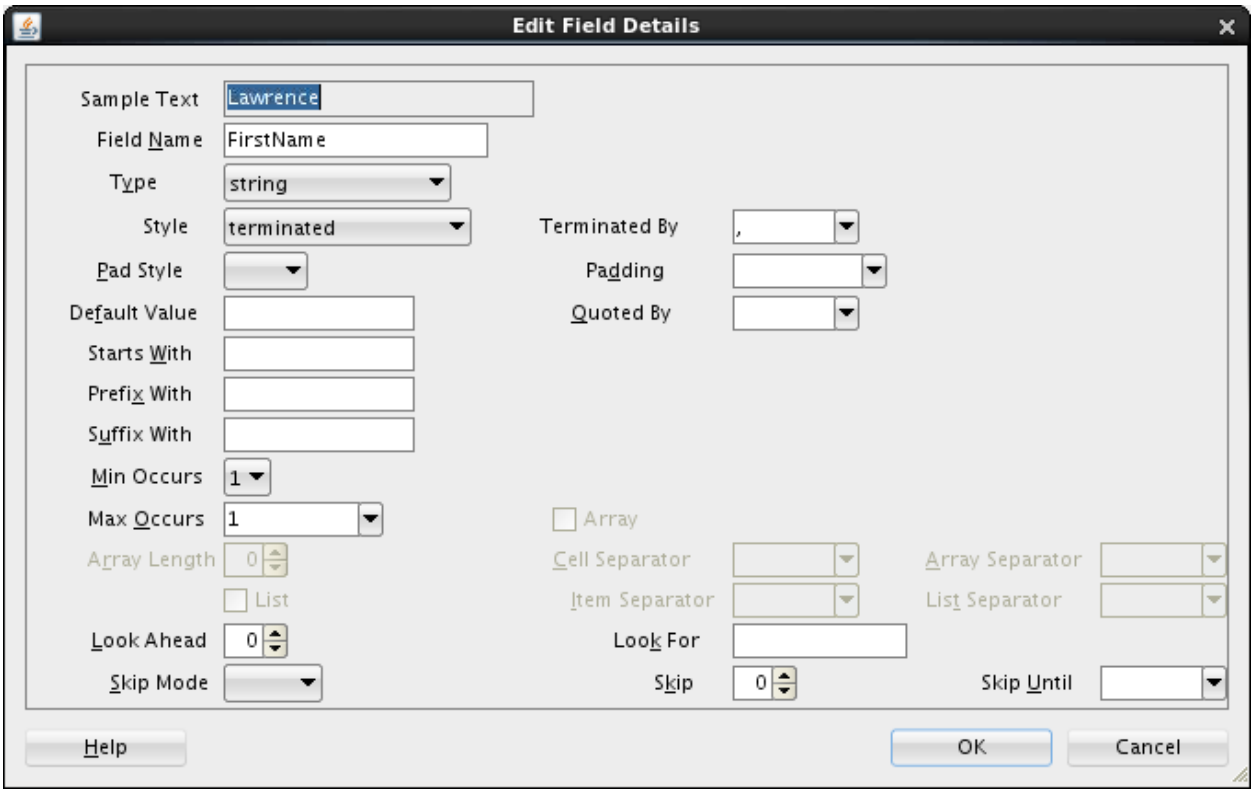


g. Rename the auto-generated elements C1,C2,…C7 to element names that are expected in the generated xml.

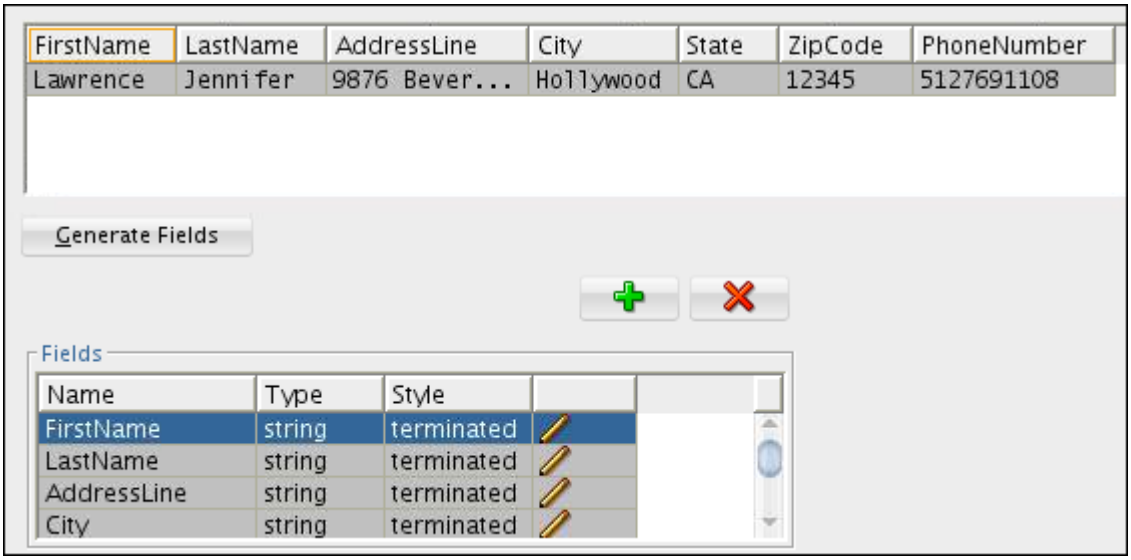
Select each of the elements and click the Pencil icon to open the Edit Field Details screen. Use the values in the table below to configure each field.

field	Field Name	Style	Terminated By
C1	FirstName	terminated	,
C2	LastName	terminated	,
C3	AddressLine	terminated	,
C4	City	terminated	,
C5	State	terminated	,
C6	ZipCode	terminated	,
C7	PhoneNumber	terminated	${eol}

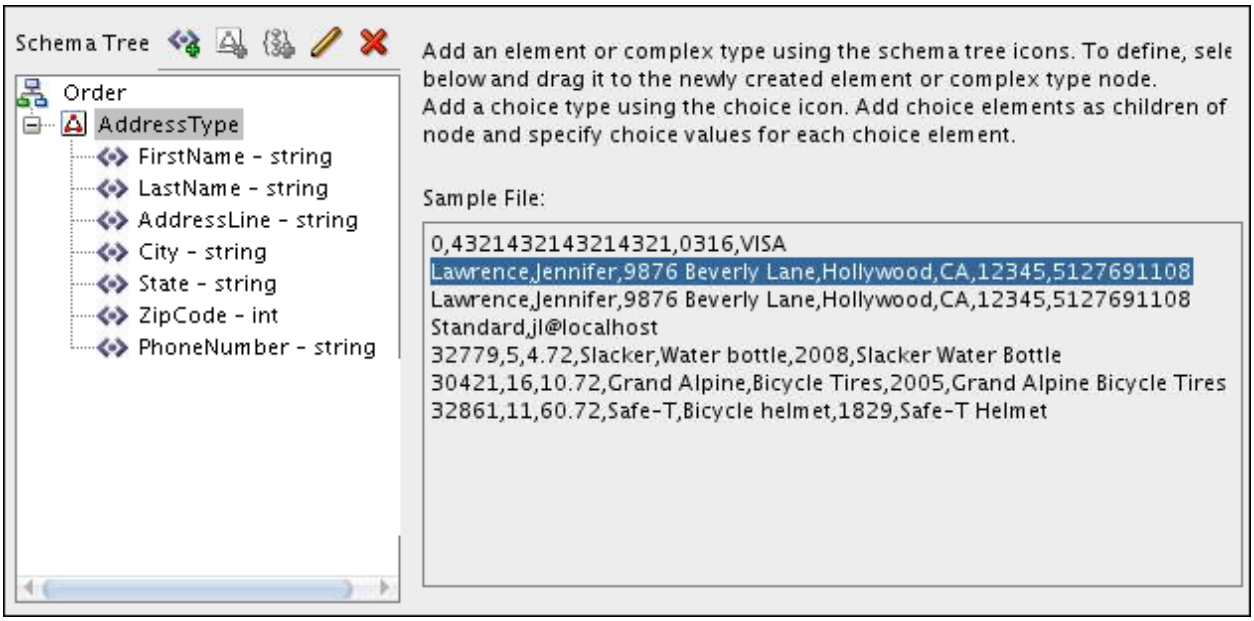
The first field is shown as an example.



h. Verify your work.



i. Click OK.



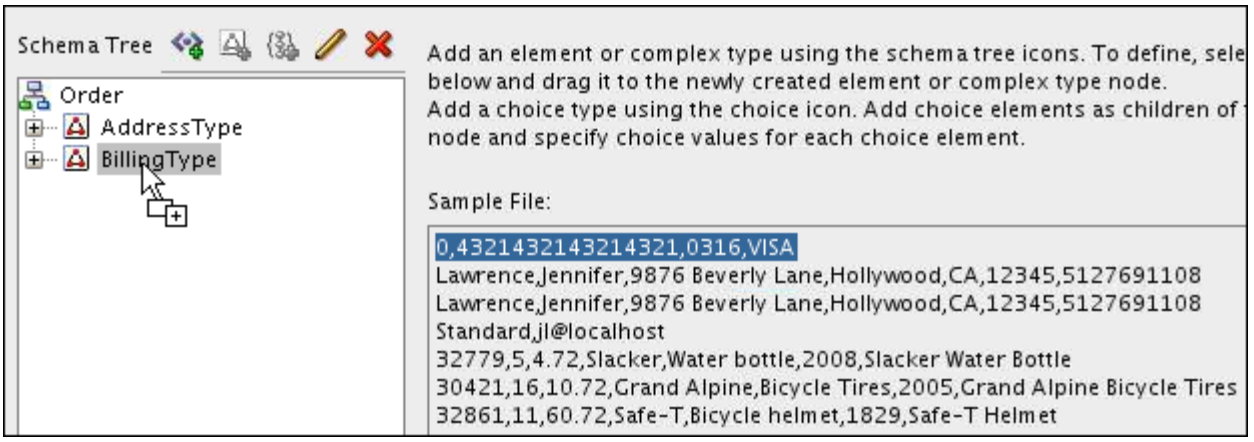
10. Create the complex type for `BillingType`.

    The `BillingType` consists of Credit Card Information and the Billing Address.

    a. Select the Order element in the left pane and click Add Complex Type.

    b. Name it `BillingType`.

    c. Select `BillingType` and drag the first line of data from the Sample File pane onto the Billing Type.



    The Complex Type Detail screen for `BillingType` is displayed.

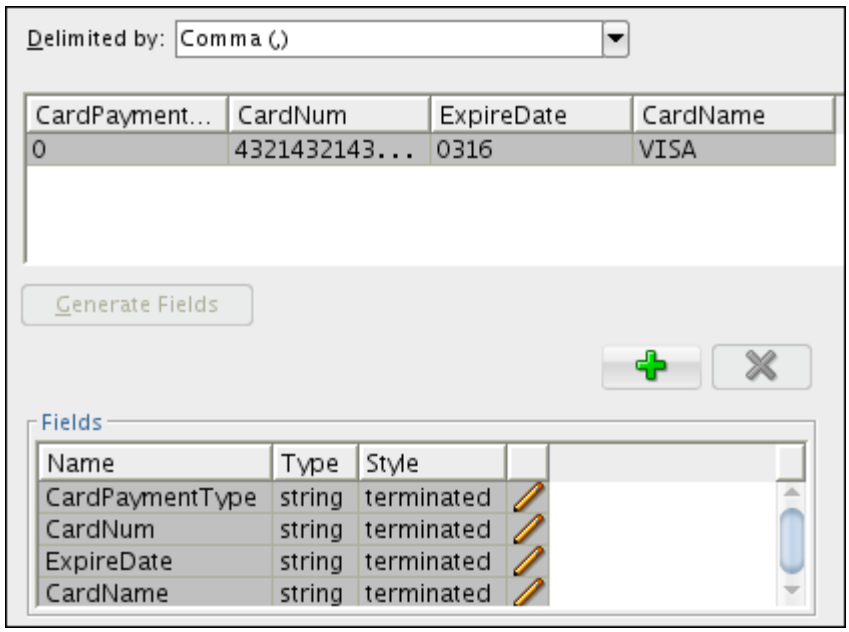
    d. Select Delimited by as Comma (,).

    e. Click Generate Fields.

f. Select each of the elements and click the “Pencil” icon to open the “Edit Field Details” screen in order to update each field. Use the values in the table below to configure each field.

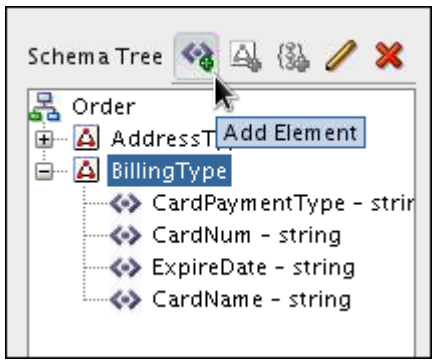
field	Field Name	Style	Terminated By
C1	`CardPaymentType`	terminated	,
C2	`CardNum`	terminated	,
C3	`ExpireDate`	terminated	,
C4	`CardName`	terminated	${eol}

g. Verify your work.



11. Add another field to this list.

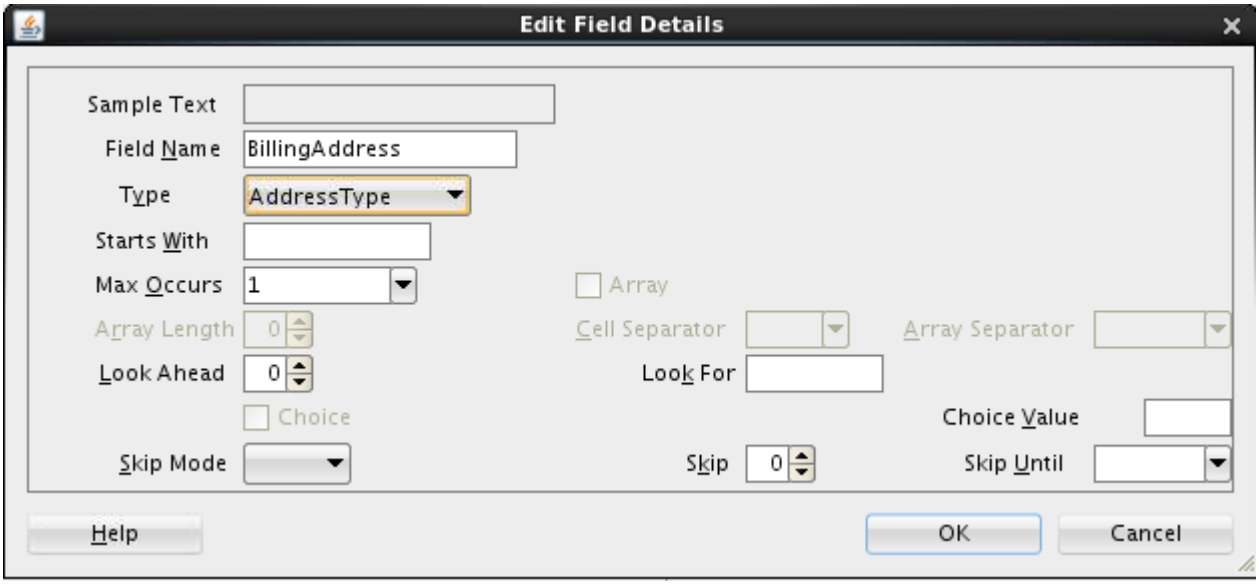
a. With BillingType selected, click the Green button (plus sign) to add a new element.



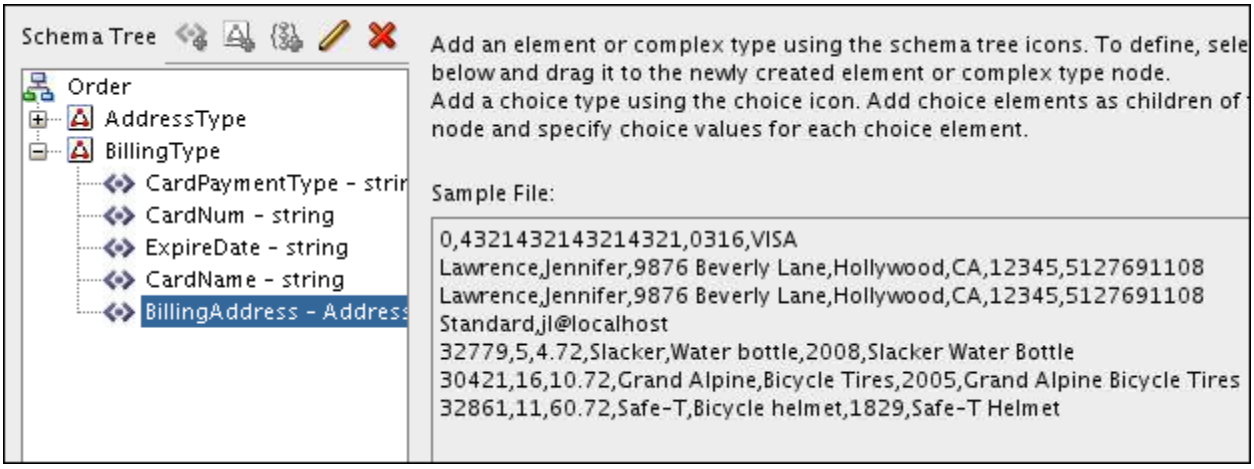
b. Name the element `BillingAddress`.

c. Click Edit Node (pencil sign).

d. In the Edit Filed Details dialog box, select `AddressType` as its type.

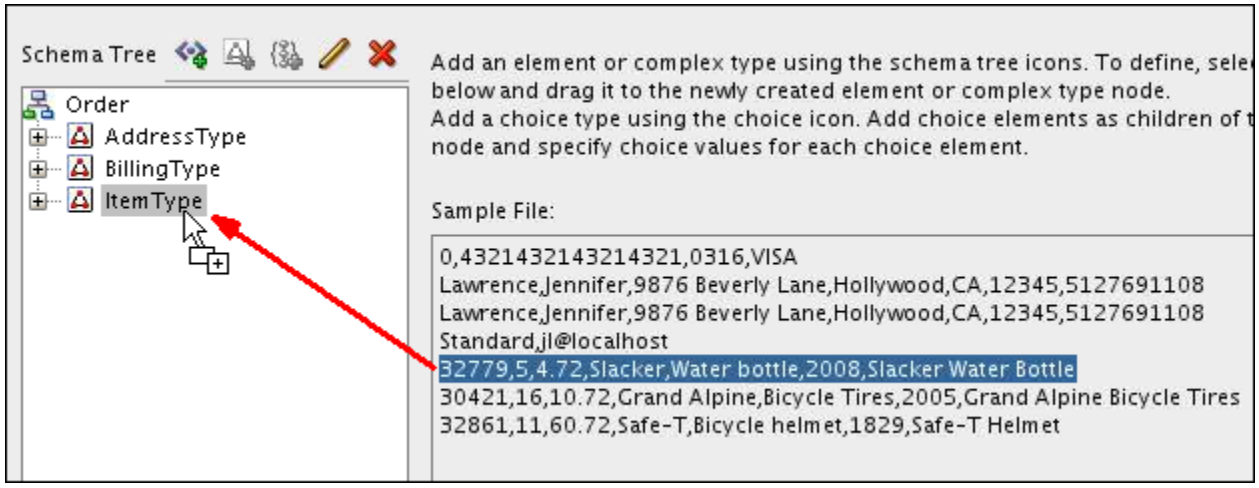


e. Click OK to close the Edit Filed Details dialog box.



12. Create the type information for the order line item.
    a. Select Order.
    b. Create a new complex type called `ItemType`.

c. Drag the first order line onto `ItemType`.



The Complex Type Detail window is displayed.

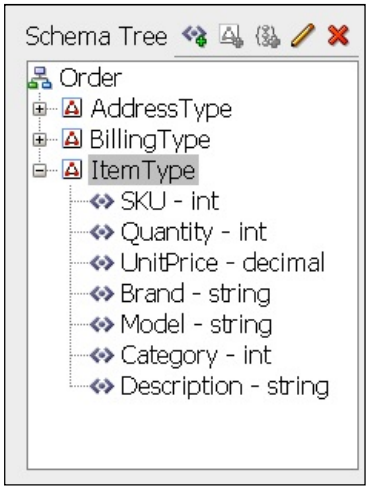
d. Select Delimited by as Comma (,) and click Generate Fields.

e. Use the data in the following table to configure newly-generated fields C1 through C7.

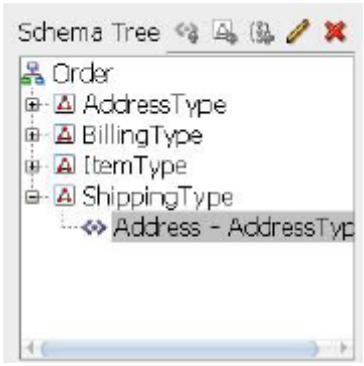
Field	Field Name	Style	Terminated By
C1	SKU	terminated	,
C2	Quantity	terminated	,
C3	UnitPrice	terminated	,
C4	Brand	terminated	,
C5	Model	terminated	,
C6	Category	terminated	,
C7	Description	terminated	${eol}

f. Click OK.

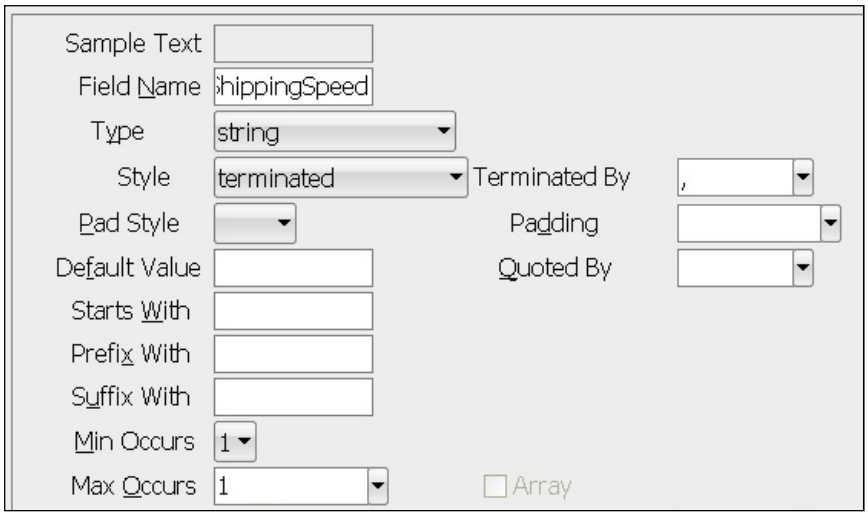
You return to the Design Schema dialog box.



13. Add a new complex type `ShippingType` under the `Order` element.
    a. Select the `ShippingType` element, and click the Green button (plus sign) to add a new element.
    b. Name the element `Address`.
    c. Drag `AddressType` onto `Address` element.

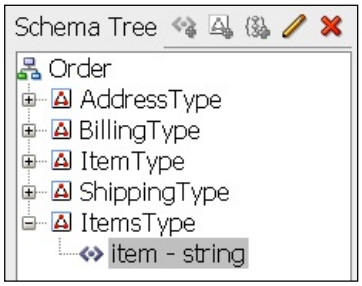


    d. Add another element, and name it `ShippingSpeed`.
    e. Select the `ShippingSpeed` element, and click the pencil icon on the top to edit the element.
    The Edit Field Details screen is displayed.
    f. Select terminated as the style.
    g. Add Terminated By as Comma (,) manually.



    h. Click OK.
    You return to the Design Schema screen.
14. Create another complex type with the name `ItemsType`.
    a. Select `Order` element.
    b. Select `ItemsType`.

c. Create a new element `item`.



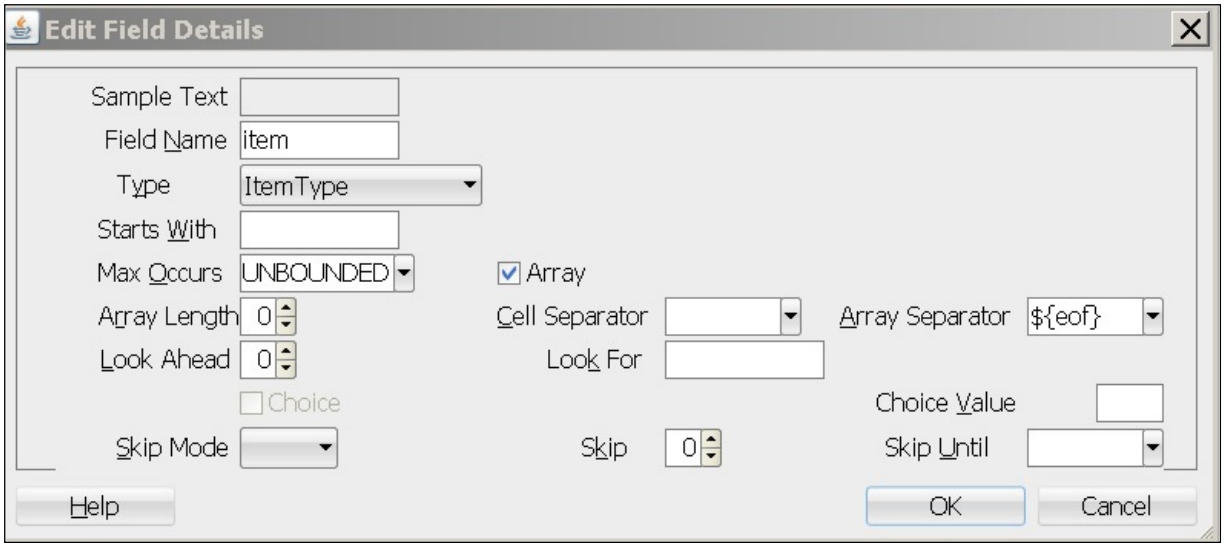
d. Select `item` element.

e. Click the pencil icon to edit the element.

f. Select type as `ItemType`, Max Occurs as UNBOUNDED.

g. Check the Array box.

h. Set the Array Separator to `${eof}`



i. Click OK.

You are returned to the Design Schema screen.

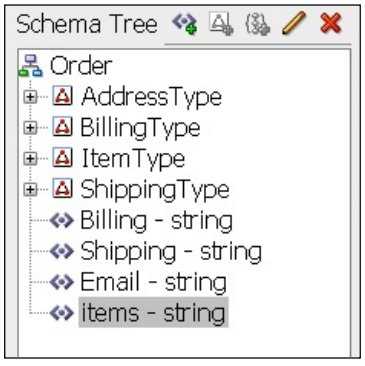
15. Create the elements.

a. Select the `Order` element.

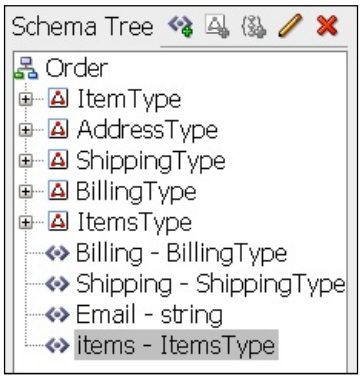
b. Click the Add Element icon.

c. Create four elements: `Billing`, `Shipping`, `Email`, and `items`.

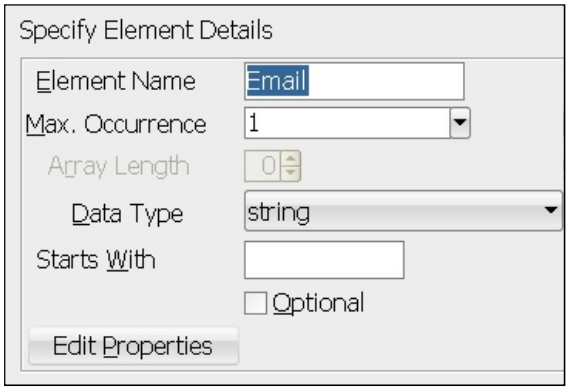
d. Verify your work.



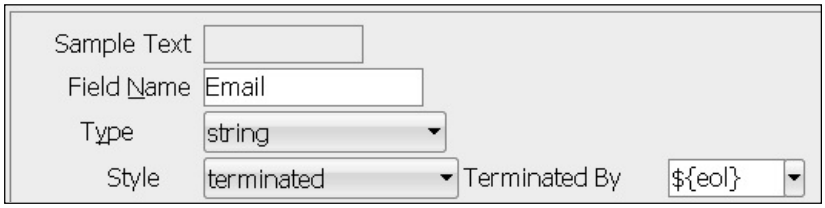
e. Drag `BillingType` onto `Billing`, `ShippingType` onto `Shipping`, and `ItemsType` onto `items`.



16. Configure the `Email` element.
    a. Select the `Email` element.
    b. Click the pencil icon to edit the information.
    The Element Details screen is displayed.

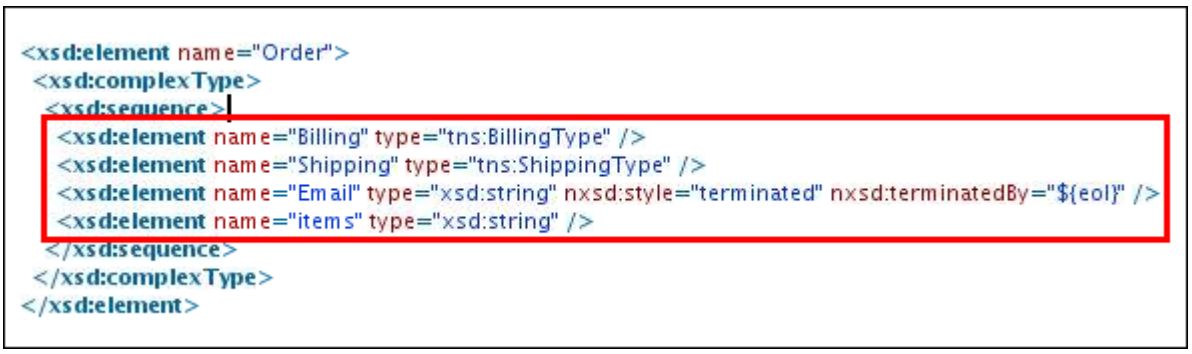


    c. Click Edit Properties.
    d. Set Type to `string`, Style to `terminated`, and Terminated By to `${eol}`.



    e. Click OK.
    f. Click OK again (in the Element Details window).
    You return to the Design Schema dialog box.
    g. Click Next.

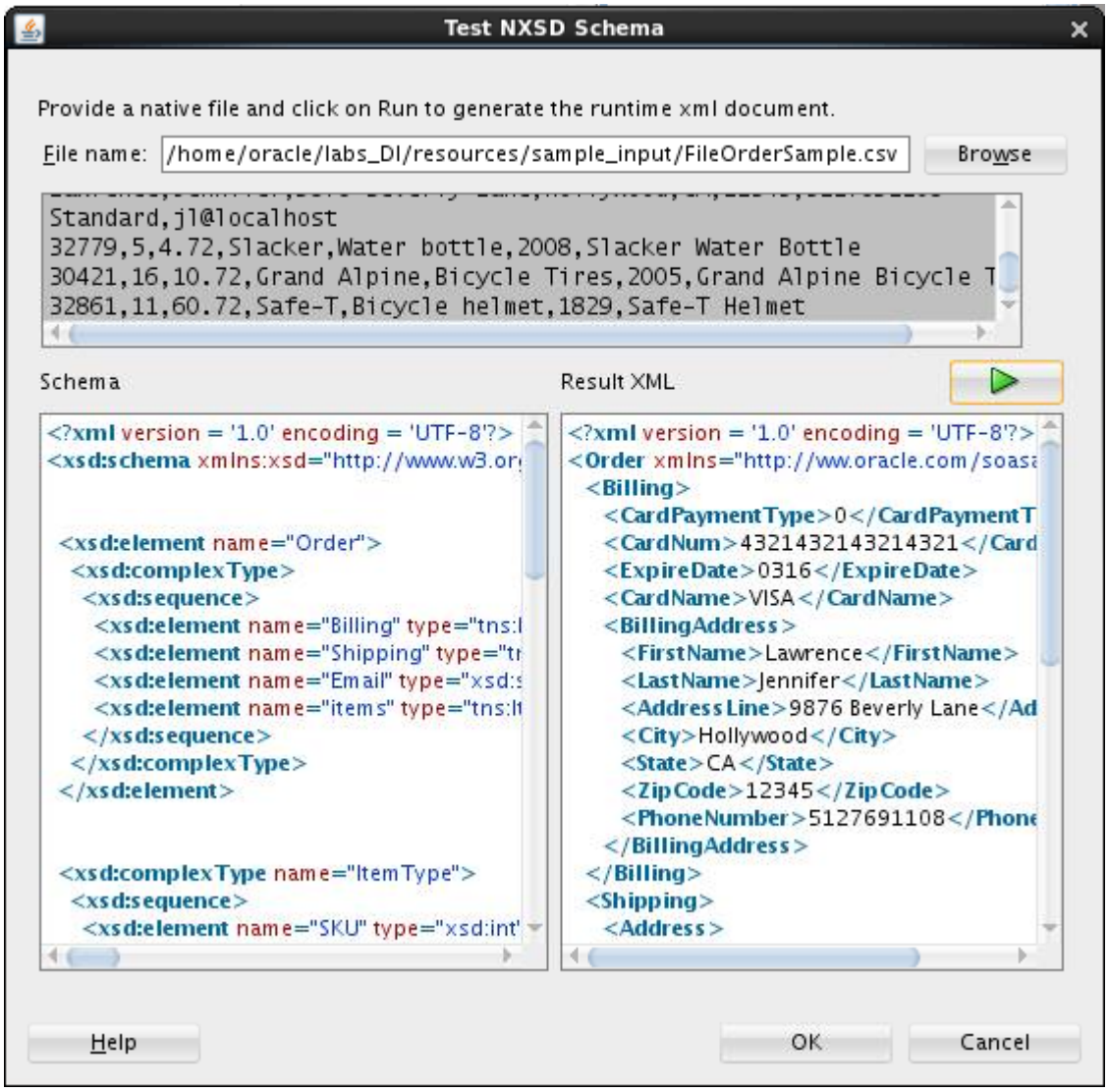
h. Verify that the top level `Order` elements appear exactly as shown below. Reorder the elements if they do not occur in the exact order shown.



Test the nXSD.

17. Click the Test button.
18. Click the Generate XML button.

The generated XML is displayed.



19. Save your project.

Practices for Lesson 7: Routing Messages

Chapter 7

Practices for Lesson 7: Overview

Practices Overview

Each credit card company can provide its own payment validation service. In these practices, you create applications that select the credit card service based on the card name in the request message. You first create a content-based routing by using conditional branching, then same credit card service selection requirement, but using dynamic routing.

Practice 7-1: Static Routing Messages using Conditional Branching

Overview

In this practice, you create a service bus application that takes requests and decides which credit card service to call.

OSB will route the incoming requests to 2 different credit card services: VISA and American Express based on the credit card name, element, and value sent in the payload.

Tasks

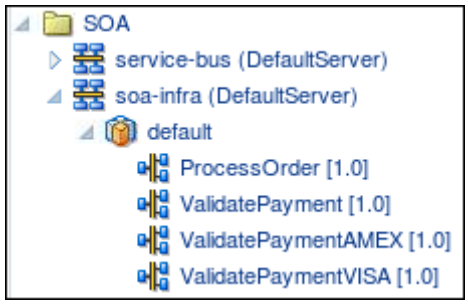
Setups

1. Deploy the two payment validation composites.
   a. In a terminal window, enter the following commands:

      ```
      >cd $HOME/labs_DI/lessons/lesson07
      >ls -l
      >$HOME/labs_DI/resources/Scripts/deploy.sh
      sca_ValidatePaymentVISA_rev1.0.jar
      >$HOME/labs_DI/resources/Scripts/deploy.sh
      sca_ValidatePaymentAMEX_rev1.0.jar
      ```

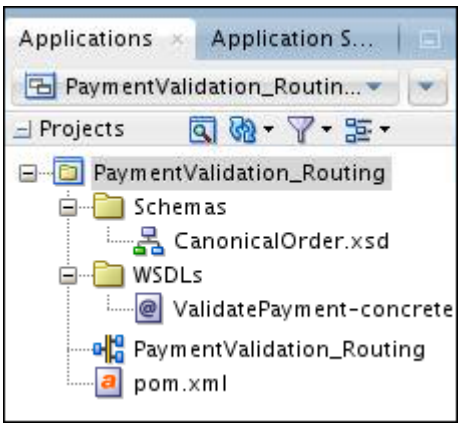
      Make sure you see that both builds are successful.

   b. Open EM console. In Target Navigation panel, expand SOA > soa-infra > default. Now you see the newly deployed two composite applications in the folder.

2. Open Service Bus project
   a. In JDeveloper, select **File** > **Open** from the main menu.
   b. In the Open Application(s) dialog box, navigate to the `$HOME/labs_DI/lessons/lesson07/PaymentValidation_RoutingServiceBusApp/` directory, and open the `PaymentValidation_RoutingServiceBusApp.jws` file.

c. Expand the folders in Application Navigator, and you should see the folder structure resembling the image below:



To save your time, the application and project folder structure is pre-created and source WSDL and Schema files are imported into the project for you.

3. Create the business services.

There are two credit card services (two endpoints), and you will create business services for each of them.

a. In Application navigator, double-click PaymentValidation_Routing to open it in Service Bus Overview editor.

b. Drag the HTTP component from the Technology section in the Component palette and drop it onto the External Services lane.

c. Enter the required details for business by following the instructions in the table below:

Step	Window/Page Description	Choices or Values
1)	Create Service	Name the service `AmexBS`. Click Next.
2)	Type	1) Select **WSDL** for Service Type. 2) Click the Browse WSDLs icon to the right of the WSDL choice. 3) In the Select WSDL dialog box, click **Application**, navigate to Application > PaymentValidation_Routing > WSDLs, and select **ValidatePayment-concrete.wsdl**. 4) Click **OK**. 5) Back in the Create Business Service window, you should see the WSDL and port fields populated with the information based on the WSDL that you chose. 6) Click **Next**.
3)	Transport	Verify that http is selected in the Transport field. Replace the Endpoint URI with the URL of validatePaymentAmex SOA service that you deployed earlier. It should look like the following: **http://localhost:7101/soa-infra/services/default/ValidatePaymentAMEX/validatepayment_client_amex** Click **Finish**.

d. Repeat the above steps to create another business service for Visa card service. Two things you need to do differently include:
   - Name the service `VisaBS`.
   - The Endpoint URI should point to validatePaymentVisa SOA service, which is **http://localhost:7101/soa-infra/services/default/ValidatePaymentVISA/validatepayment_client_visa**

4. Create Proxy Service and add pipeline.
   a. In Overview Editor, drag and drop the HTTP component from the Components palette to the Proxy Services lane.

      The Create Proxy Service wizard pops up.
   b. In step one (Create Service), fill up the details for Proxy Service:
      - Service Name: **PaymentValidation_RoutingPS**
      - Location: (keep as is)
      - Transport: http (keep as is)
      - Keep “Generate Pipeline” checked for creating pipeline along with Proxy Service
      - Change the pipeline name to **PaymentValidation_RoutingPP**

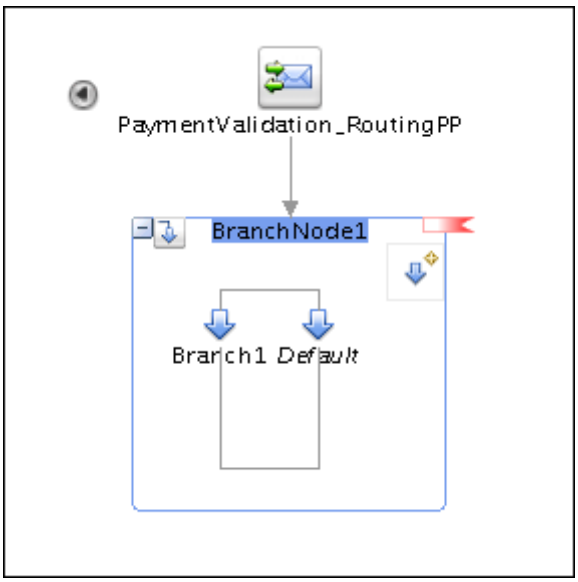
      Click **Next**.
   c. In step two (Type), use the same WSDL of the business service for the proxy service:
      - Select the service type **WSDL.**
      - Click the **Browse WSDLs** icon on the right.

        The Select WSDL dialog box pops up.
      - Navigate to Application > PaymentValidation_Routing > WSDLs directory, and select **ValidatePayment-concrete.wsdl**.
      - Click **OK**.

      Click **Next**.
   d. In step three (Transport), you should see the populated endpoint URI depending on your configuration in previous steps. Keep it as is and click **Finish**.

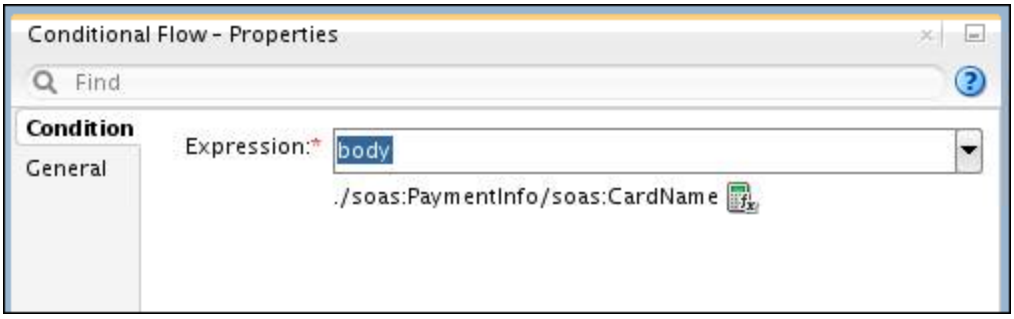
   Now you should see proxy and pipeline components in the Service Bus Overview Editor and their files (PaymentValidationPS.proxy, PaymentValidationPP.pipeline) added in the project folder.
5. Configure pipeline to add a conditional branching
   a. Open the pipeline in the Pipeline editor.

b. Drag the Conditional Branch node (from the Nodes section) and drop it below the Start node in the editor.



c. Select **BranchNode1**.

d. In Conditional Flow Property window, make the following selections:

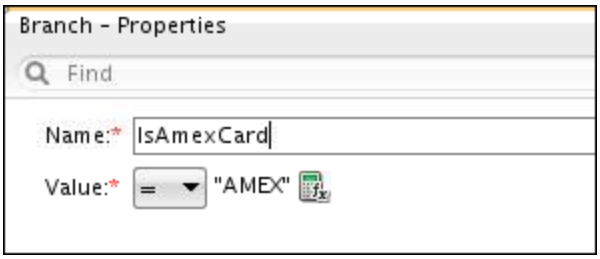


e. Select **Branch1**.

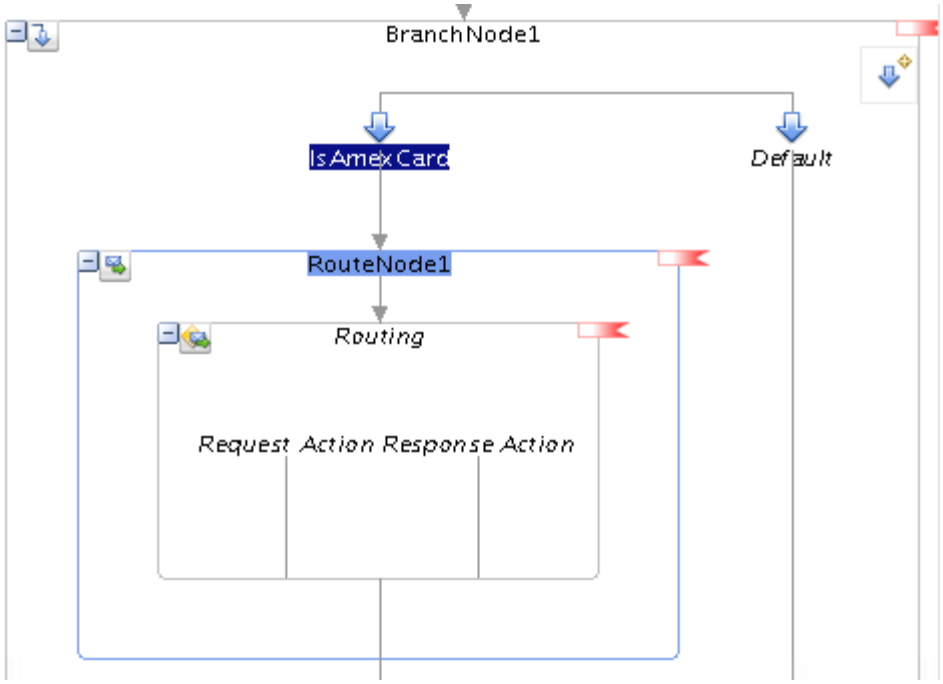
f. In Branch Property window, do the following:

- Enter **`IsAmexCard`** as name.
- Click the XQuery Expression Builder icon, type **`"AMEX"`** in the Expression field.
- Click OK.

Your Branch Property window should now resemble the following image:



g. Drag a Routing action (from the Route section in Component palette) and drop it onto the IsAmexCard branch.



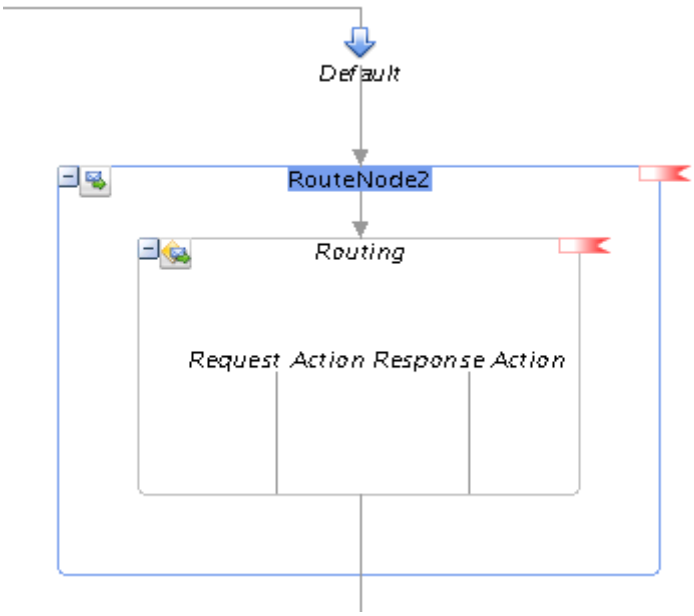
Note that this behavior automatically creates the route node, RouteNode1, along with the routing action.

h. Select **Routing**.

i. In the Routing Property window, select **AmexBS.bix** as the routing target business service.

Now you complete the configuration for the American Express card branch. Next, you do the same for the Visa card.

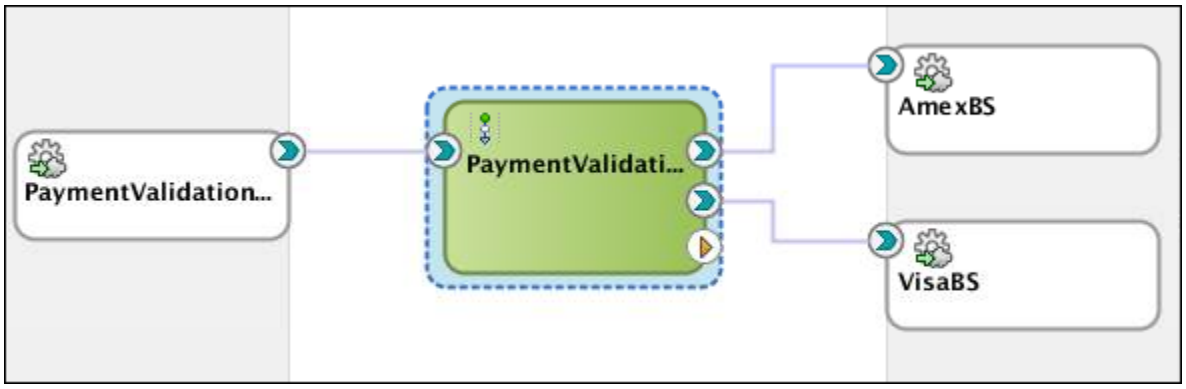
j. Add a routing action to the Default branch.



k. Select **Routing**.

l. In the Routing Property window, select **VisaBS.bix** as the routing target business service.

In the Service Bus Overview Editor, your project should look like the image below:

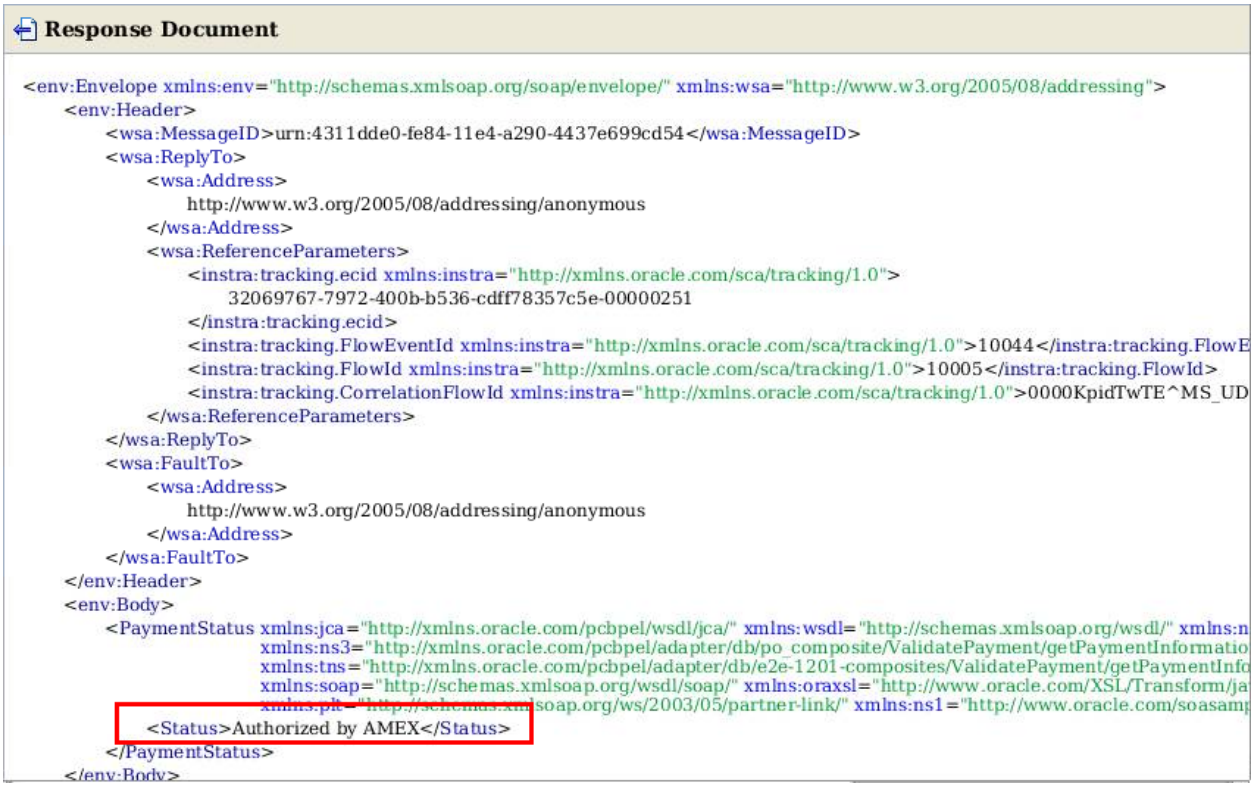


6. Save your project.

Deploy and Test

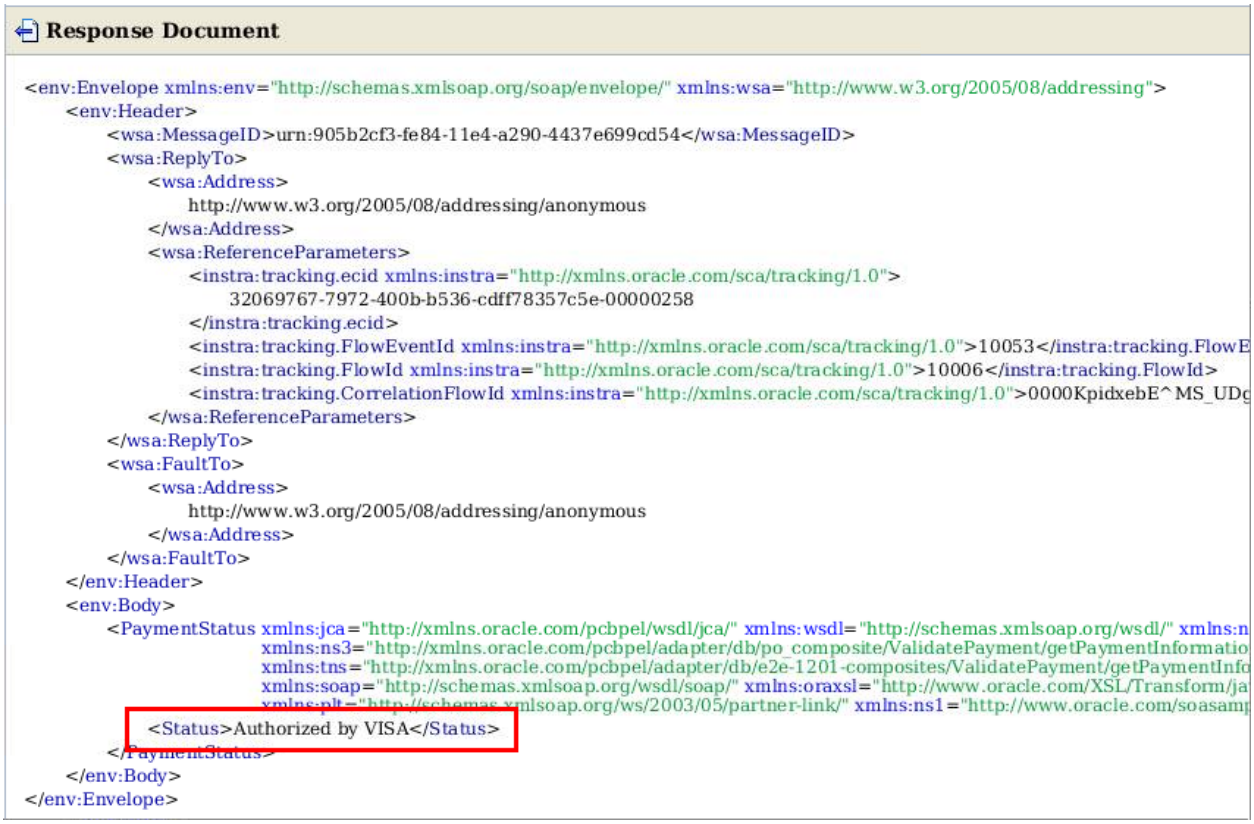
7. To test your Service Bus application:
   a. Right-click the proxy service, and select Run.
   b. In the Test console, test a payment authorization request with American Express card. Select /home/oracle/labs_DI/resources/sample_input/PaymentInfoSample_Amex.xml
   c. Click Execute button.

      You should see the status is “Authorized by Amex” in the response message.



   d. Similarly, select /home/oracle/labs_DI/resources/sample_input/PaymentInfoSample_Visa.xml for

testing a payment authorization request with Visa card, and you see “Authorized by VISA” in the response message, as shown in the image below:



Practice 7-2: Dynamic Routing Messages using XQuery-based Policy

Overview

Service Bus application can dynamically route to the correct payment validation service, based on the credit card name in the request message.

In the practices, you use the XQuery to provide a routing table that maps the logical names coming from an incoming request to the actual service. Then, you create a pipeline that performs this lookup and routes to the correct service.

In this case, we have two credit card companies, American Express and Visa. Here is the routing rule:

```
xquery  version "1.0" encoding "utf-8";
<Configuration>
    <CardName value ='AMEX'>
       <Service>PaymentValidation_Routing/AmexBS</Service>
    </CardName>

    <CardName value ='VISA'>
       <Service>PaymentValidation_Routing/VisaBS</Service>
    </CardName>
</Configuration>
```

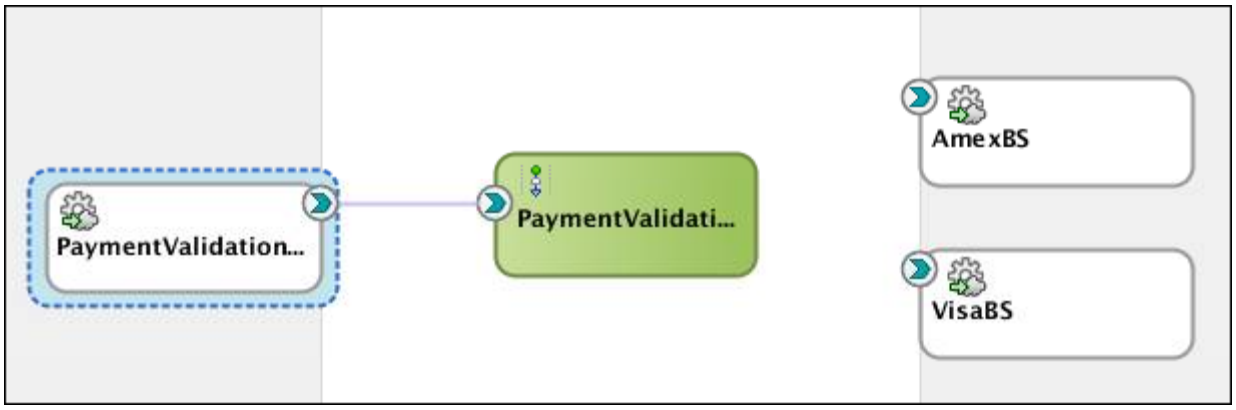
Here is the dynamic routing expression:

```
<ctx:route>
    <ctx:service
isProxy='false'>{$varConfig/CardName[@value=$varCardName]/Servic
e/text()}</ctx:service>
    <ctx:operation>{$operation}</ctx:operation>
</ctx:route>
```

The key tasks you perform are:

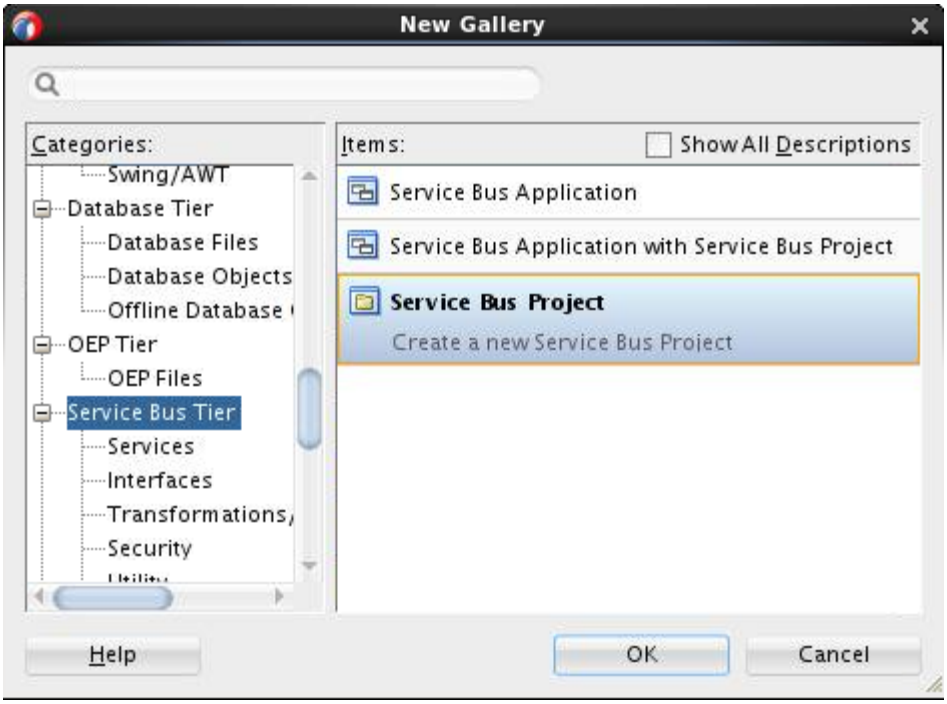
- Configure a pipeline with a proxy service that handles dynamic requests
- Determine where to route an incoming request

For this practice, you can reuse the business service resources created in the previous project. Once done, your project will look like this:



Tasks

1. Create a new Service Bus project.
   a. In JDeveloper, select **File > New > Project** from the main menu.
   b. Select **Service Bus Project**.



   c. Click **OK**.
   d. Name the project `PaymentValidation_DynamicRouting`, and click **Finish**. You see that a new project folder is added in Application Navigator.
   e. In the project folder, add a new folder, and name it **XQuery**.

2. Create XQuery resource.

   You use an XQuery resource to store the information about the business services that can be used at runtime.

   You have two ways to create the XQuery resource:

   - Importing from the XQuery resource file into the project
   - Creating from scratch by using the XQuery editor (Right-click XQuery folder, and select New > XQuery File ver 1.0…)

   To import from the file system, do as follows:

   a. Right-click the **XQuery** folder in the Project Explorer, and select Import from the context menu.

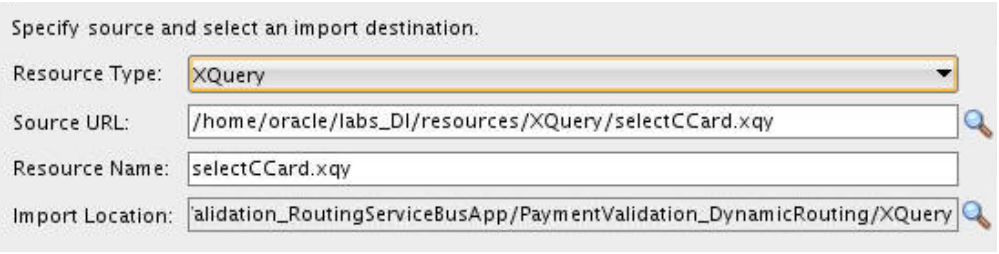
      The Import dialog box is displayed.

   b. Select **Service Bus Resources**.

   c. Click **OK**.

      The Import Service Bus resources wizard is displayed.

   d. Complete the steps in the wizard by following the instructions in the table below:

Step	Page Description	Choices or Values
1)	Type	Select **resources from URL.** Click **Next**.
2)	Source	Select **XQuery** for Resource Type. Click the Browse icon next to the Source URL field. In the Select XQuery dialog box, select File System in the top box, navigate to the `/home/oracle/labs_DI/resources/XQuery` folder, and select **`selectCCard.xqy`**. Click **OK**.  Click **Next**.
3)	Configuration	Accept the default. Click **Finish**.

   You see that the selectCCard.xqy file is added in the XQuery folder.

   e. Right-click selectCCard.xqy and select Open to open it in the editor.

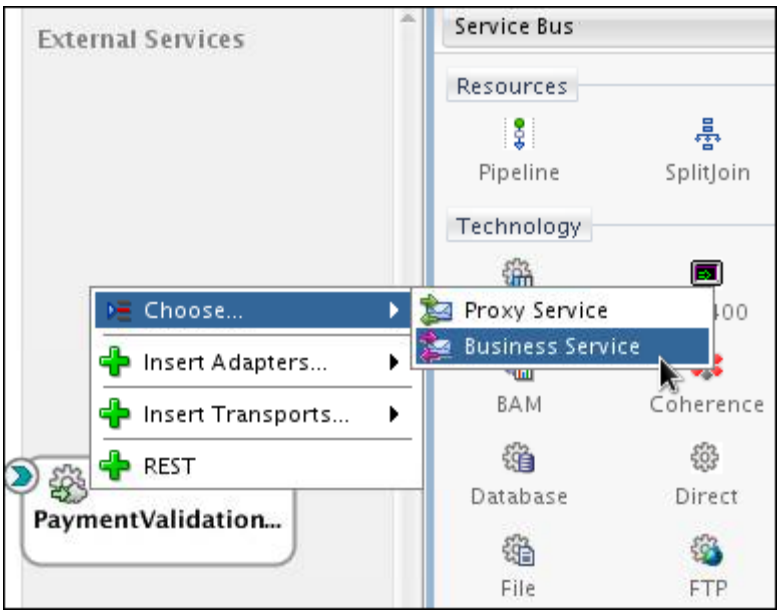
f. Click XQuery Source tab to view the source code.

```
xquery  version "1.0" encoding "utf-8";
<Configuration>
    <CardName value ='AMEX'>
       <Service>PaymentValidation_Routing/AmexBS</Service>
    </CardName>

    <CardName value ='VISA'>
       <Service>PaymentValidation_Routing/VisaBS</Service>
    </CardName>
</Configuration>
```

Note that dynamic routing uses the physical identifier of the Business service to route the request. The physical identifier is expressed as:
`<project_name>/<business_service>`

The project and business services are pointing to those created in the previous practice as you reuse them for this dynamic routing project.

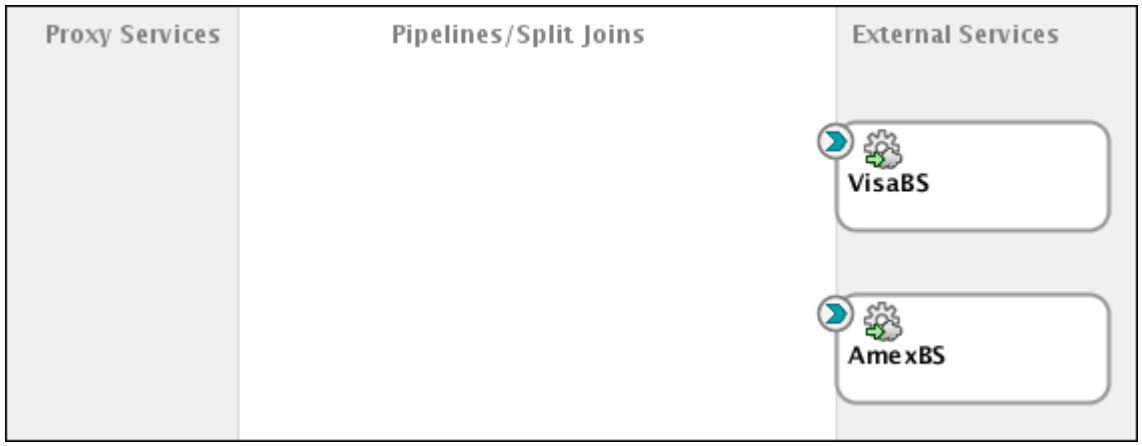
3. Reuse the business services created in the previous project:
   a. Right-click in the External Services lane, and select **Choose** > **Business Service** from the drop-down menu.



   b. Under the Resource Chooser, navigate to Application > PaymentValidation_Routing, and select **AmexBS.bix**.
   c. Click OK.

d. Use the same steps to reuse the business service created for the Visa card.

When you are done, your Service Bus Overview editor should resemble the image below:



Create and configure the pipeline to implement dynamic routing

4. To create the pipeline and proxy service, do as follows:
   a. In the PaymentValidation_DynamicRouting Overview Editor, drag and drop the Pipeline component from the Components palette to the Pipeline /Split Joins lane.

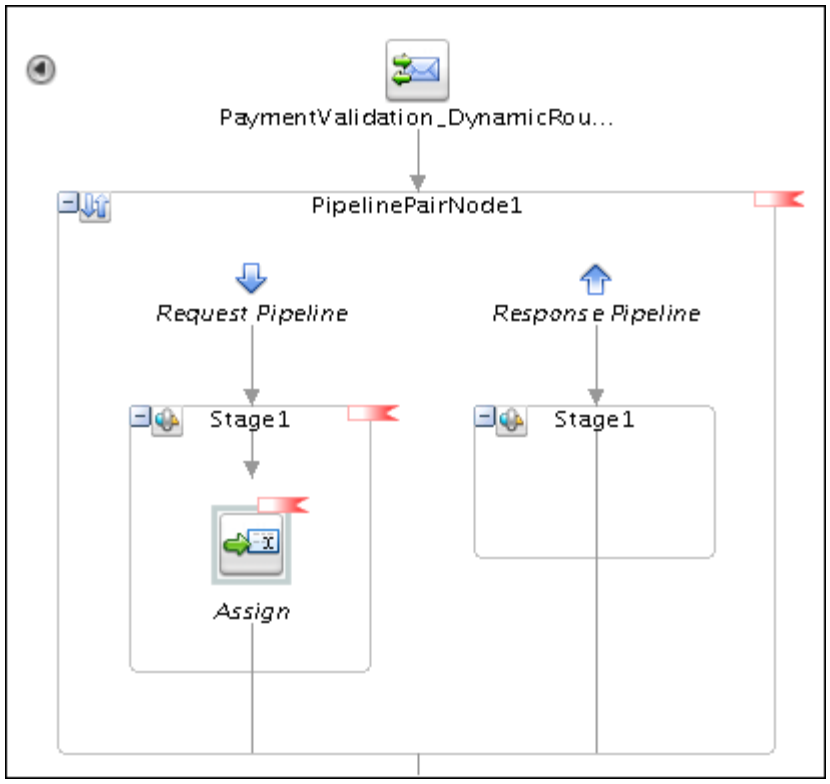
      The Create Pipeline Service wizard pops up.
   b. In step one (Create Service), name the Pipeline Service **PaymentValidation_DynamicRoutingPP**.
   c. Click **Next**.
   d. In step two (Type), use the same WSDL of the business service for the pipeline service:
      1) Select the service type **WSDL.**
      2) Click the **Browse WSDLs** icon on the right.
      3) The Select WSDL dialog box pops up.
      4) Navigate to Application > PaymentValidation_Routing > WSDLs directory, and select **ValidatePayment-concrete.wsdl**.
      5) Click **OK**.

         Back in the wizard, you should see the fields populated with the wsdl file and port.
      6) Keep "Expose as a Proxy Service" checked for creating Proxy Service along with pipeline.
      7) Change the Proxy Service name to **PaymentValidation_DynamicRoutingPS**. Leave rest of the values to their defaults.
   e. Click **Finish**.

      You see that both the pipeline and the proxy service are added in Overview Editor.
5. Configure the pipeline.
   a. Double-click the pipeline to open its editor.
   b. Drag Pipeline Pair from the Nodes section in the Components palette and drop it below the Start node.

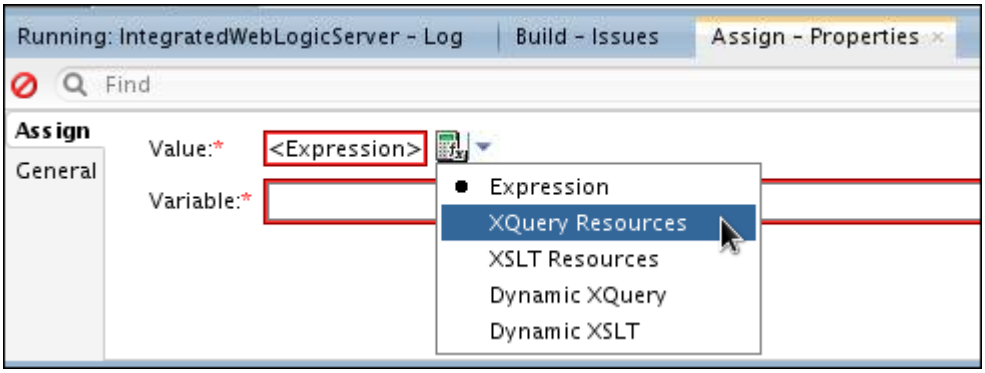
      You see the Pipeline Pair node, comprising a Request Pipeline and a Response Pipeline, being added to the start node.

c. Add an Assign action to Stage1 on the Request Pipeline.



You use this Assign activity to load the XQuery resource and store the content in a variable.

1) In the Assign Properties inspector, click the triangle shape to view the drop down menu, and select XQuery Resources.

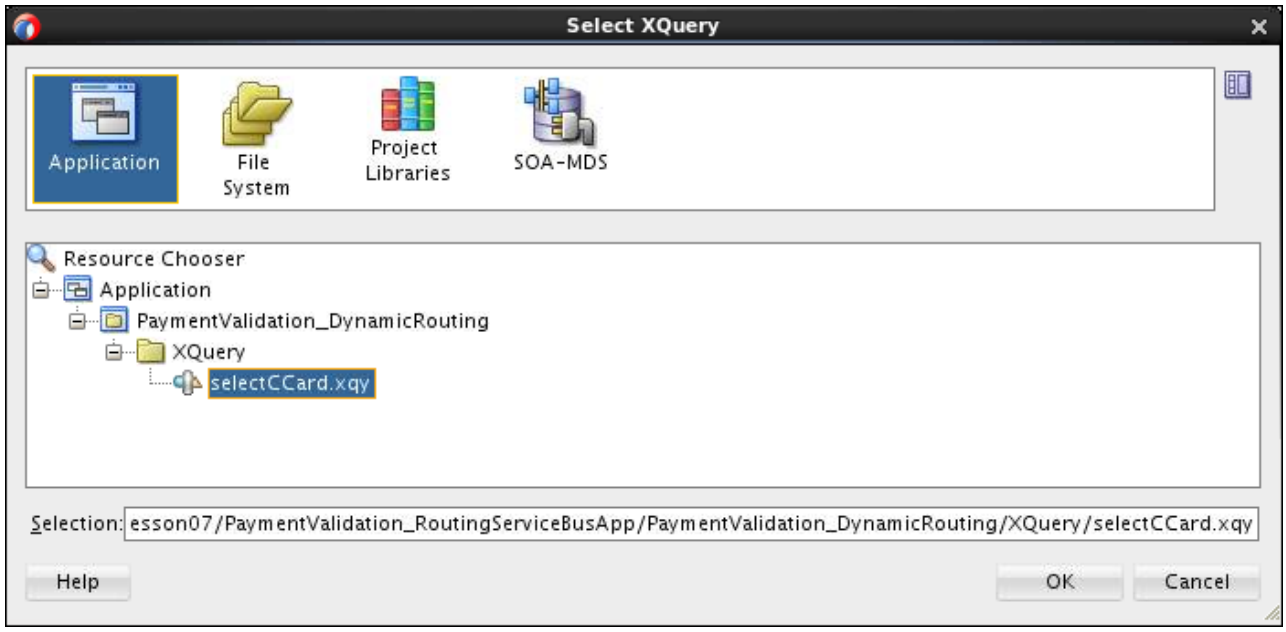


The XQuery Transformation Expression Builder window is displayed.

2) Click browse icon next to the XQuery Resource field.

The Select XQuery dialog box is displayed.

3) Select selectCCard.xqy imported into your application.



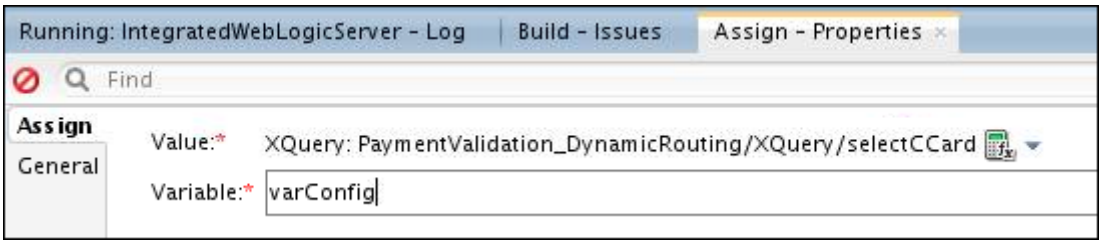
4) Click OK.

The XQuery Resource field is populated with your selection.

5) Click OK.

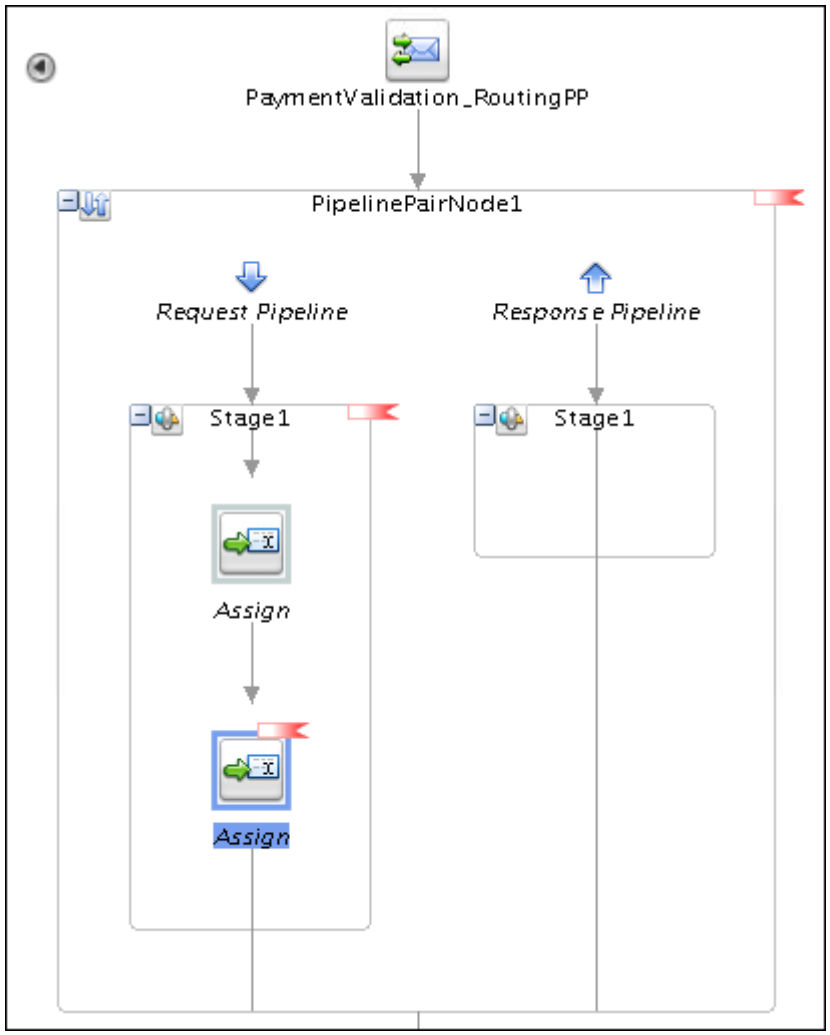
Back in the Assign Properties inspector, you see that the Value field is updated too.

6) In the Variable field, enter **`varConfig`** as the name of the variable.



This assigns the XQuery resource to this variable.

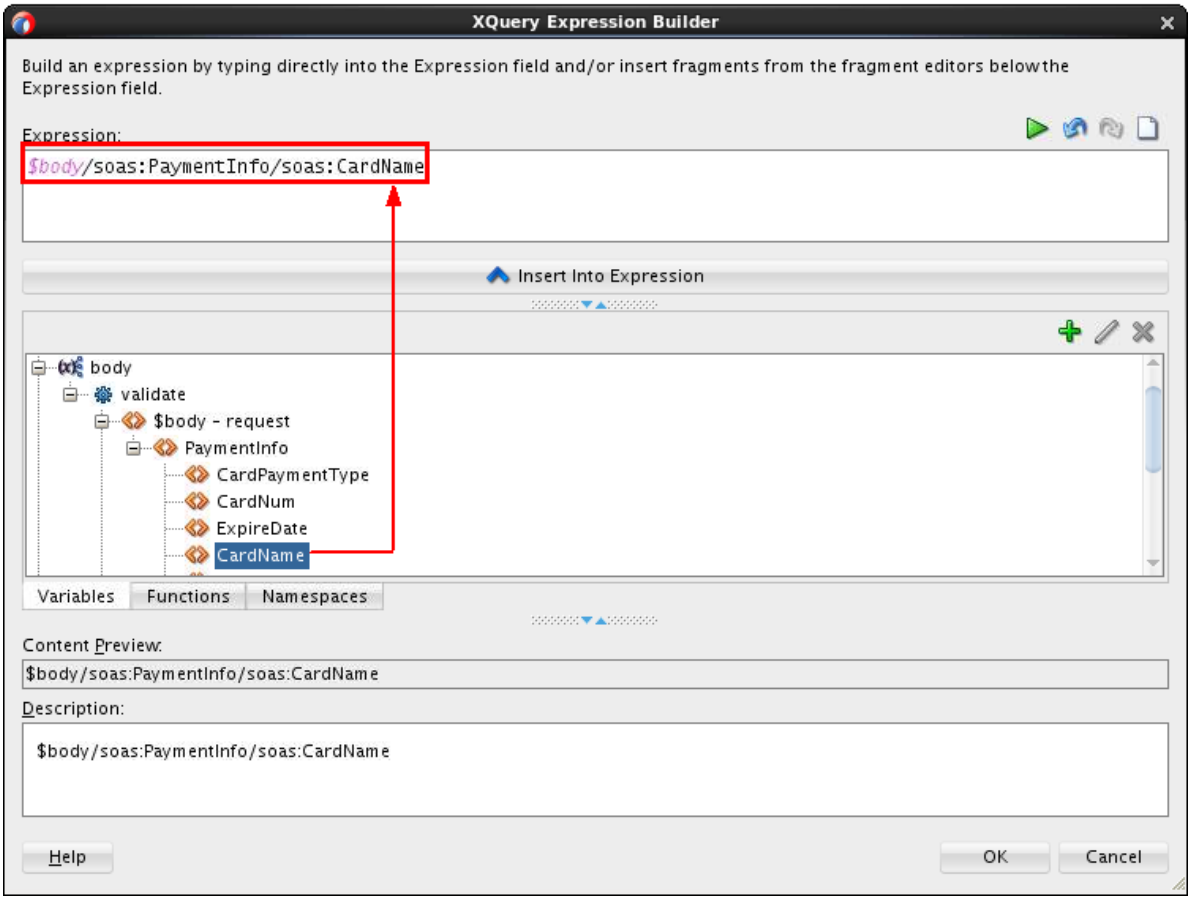
d. Add another Assign action right below the first Assign action.



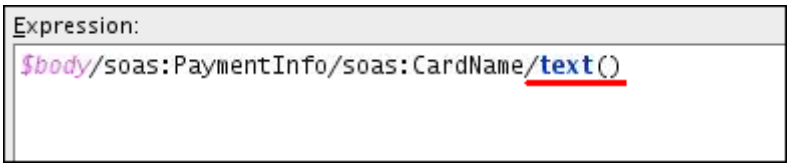
You use this Assign activity to store the value of the `CardName` element in the payload in a variable.

1) In the Assign Properties inspector, click the XQuery Expression Builder icon  . The XQuery Expression Builder window opens.

2) Locate the CardName element in $body, and insert it into the Expression field.



3) Append /text() to the end of the expression as shown below:

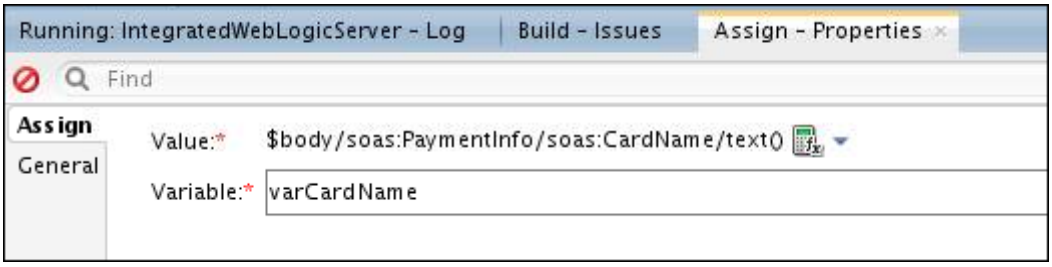


This way you make sure to get the string from the context element.

4) Click OK.

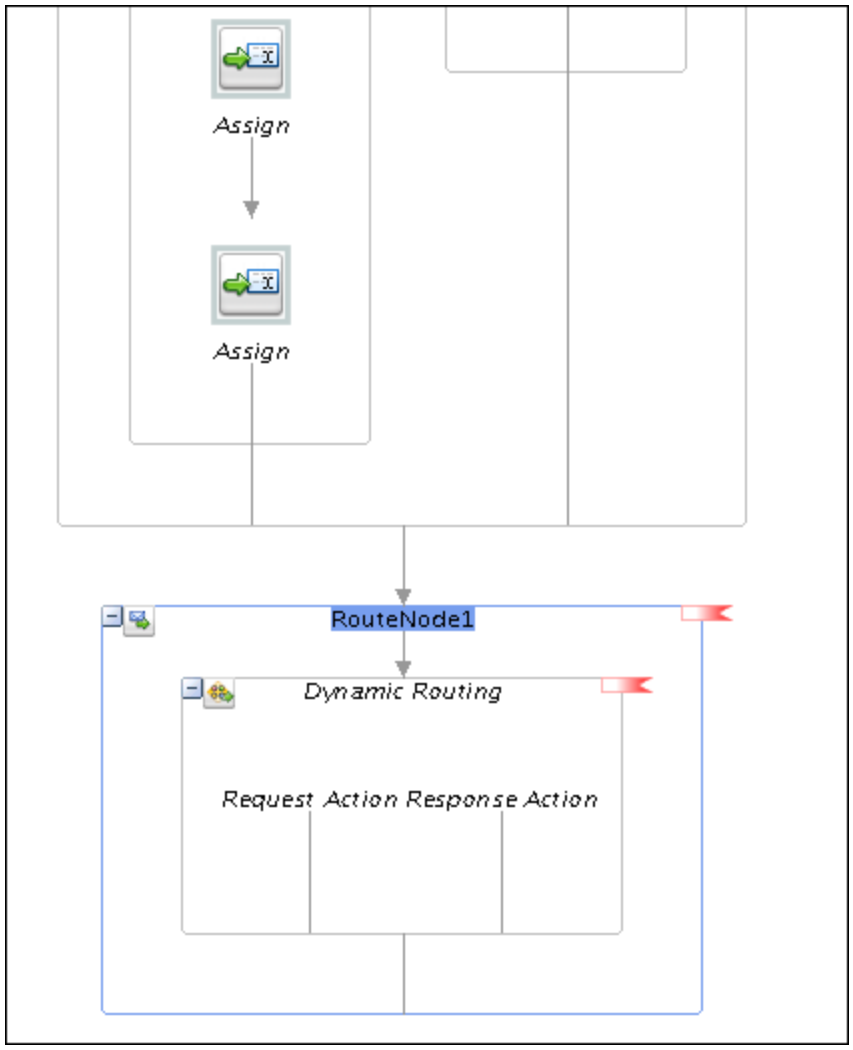
Back in the Assign Properties inspector, you see that the Value field is populated with value that you defined.

5) In the Variable field, enter **`varCardName`** as the name of the variable.



e. The last step of pipeline configuration is adding the dynamic routing node.

1) Drag the Dynamic Routing action from the Route section in the Component Palette and drop it below the Pipeline Pair node.



2) Click **Dynamic Routing**.

3) In Dynamic Routing Properties inspector, click the XQuery Expression Builder icon.

The XQuery Expression Builder window opens.

4) In the Expression field, type the dynamic routing expression as follows:

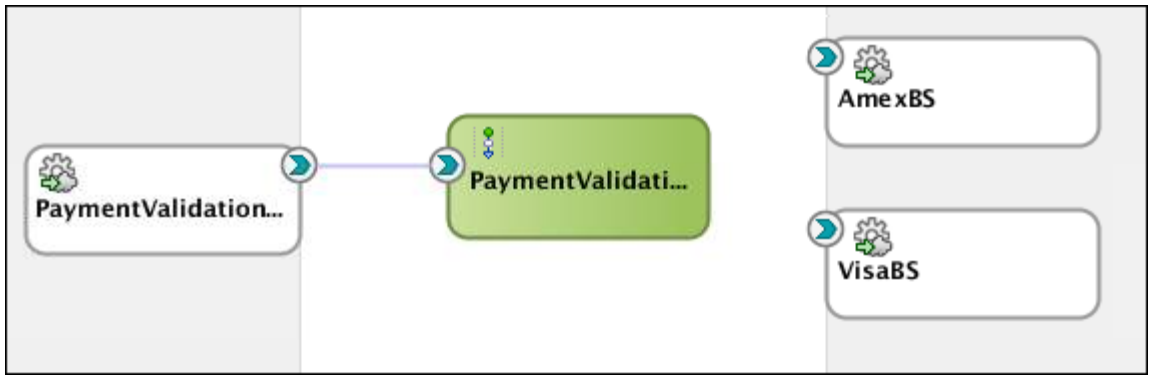
Expression:

```
<ctx:route>
    <ctx:service isProxy='false'>{$varConfig/CardName[@value=$varCardName]/Service/text()}</ctx:service>
    <ctx:operation>{$operation}</ctx:operation>
</ctx:route>
```

Note: You can find the code in the DynamicRoutingExpression.txt file in the ~/labs_DI/lessons/lesson07 directory.

The attribute is Proxy has to be set to ‘true’ if you are routing the request to proxy service or ‘false’ if you are routing the request to business service. Operation name is optional as business service might not always be WSDL service.

Now your project Overview Editor should resemble the following image:



f. Save your project.

Deploy and Test

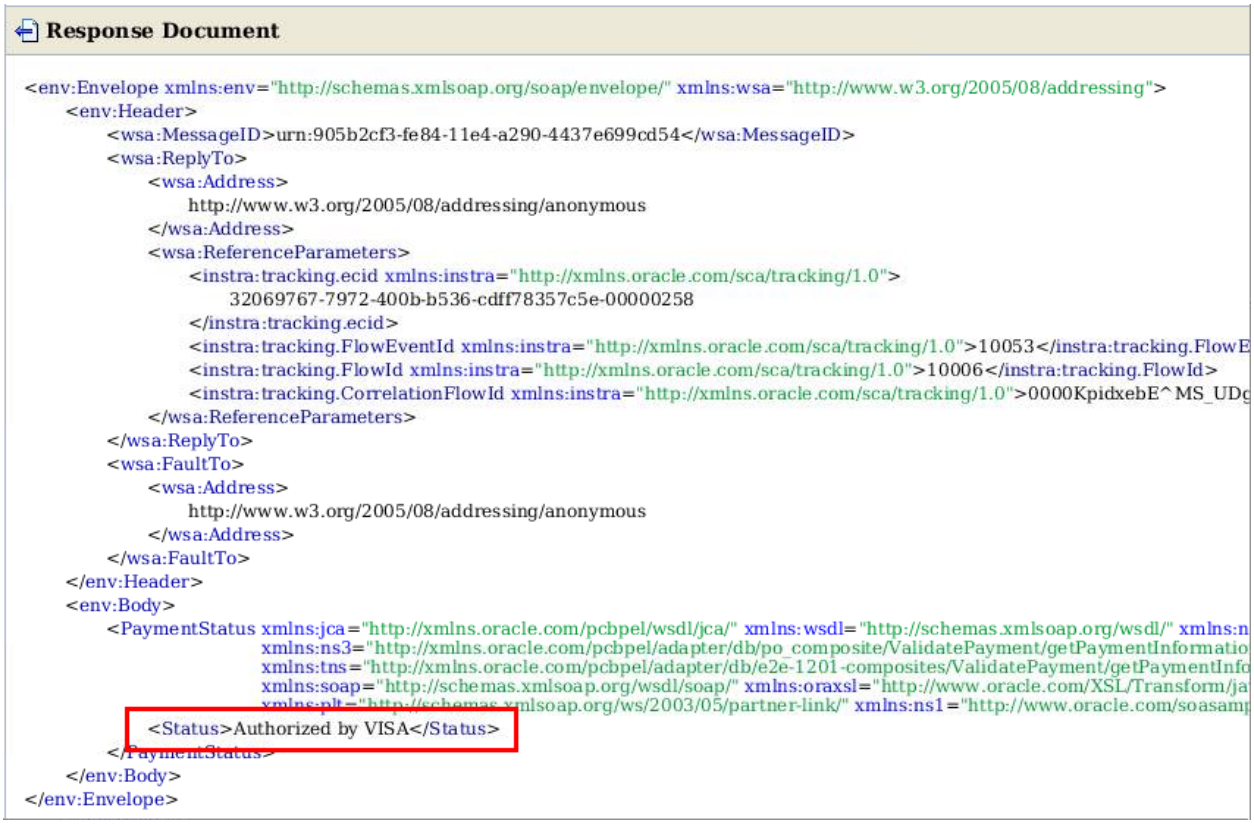
6. To test your Service Bus application, do as follows:
   a. Right-click the proxy service, and select Run.
   b. In the Test console, test a payment authorization request with American Express card. Select /home/oracle/labs_DI/resources/sample_input/PaymentInfoSample_Amex.xml
   c. Click the Execute button.

   You should see that the status is “Authorized by Amex” in the response message.

Response Document

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <env:Header>
        <wsa:MessageID>urn:4311dde0-fe84-11e4-a290-4437e699cd54</wsa:MessageID>
        <wsa:ReplyTo>
            <wsa:Address>
                http://www.w3.org/2005/08/addressing/anonymous
            </wsa:Address>
            <wsa:ReferenceParameters>
                <instra:tracking.ecid xmlns:instra="http://xmlns.oracle.com/sca/tracking/1.0">
                    32069767-7972-400b-b536-cdff78357c5e-00000251
                </instra:tracking.ecid>
                <instra:tracking.FlowEventId xmlns:instra="http://xmlns.oracle.com/sca/tracking/1.0">10044</instra:tracking.FlowE
                <instra:tracking.FlowId xmlns:instra="http://xmlns.oracle.com/sca/tracking/1.0">10005</instra:tracking.FlowId>
                <instra:tracking.CorrelationFlowId xmlns:instra="http://xmlns.oracle.com/sca/tracking/1.0">0000KpidTwTE^MS_UD
            </wsa:ReferenceParameters>
        </wsa:ReplyTo>
        <wsa:FaultTo>
            <wsa:Address>
                http://www.w3.org/2005/08/addressing/anonymous
            </wsa:Address>
        </wsa:FaultTo>
    </env:Header>
    <env:Body>
        <PaymentStatus xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:n
                       xmlns:ns3="http://xmlns.oracle.com/pcbpel/adapter/db/po_composite/ValidatePayment/getPaymentInformatio
                       xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/db/e2e-1201-composites/ValidatePayment/getPaymentInfo
                       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:oraxsl="http://www.oracle.com/XSL/Transform/ja
                       xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/" xmlns:ns1="http://www.oracle.com/soasamp
            <Status>Authorized by AMEX</Status>
        </PaymentStatus>
    </env:Body>
```

d. Similarly, select /home/oracle/labs_DI/resources/sample_input/PaymentInfoSample_Visa.xml for testing a payment authorization request with Visa card, and you should see "Authorized by VISA" in the response message as shown in the image below:



7. When you are done, close the application and all open windows of the project.

Practices for Lesson 8: Enriching Messages

Chapter 8

Practices for Lesson 8: Overview

Practices Overview

In the first practice, you use service callout to invoke a pipeline service to get a session ID, and then insert it in the database.

To pass the Session ID between the calling and callee services, you have two options:

- Use request/response messages.
- Shared variable

You use shared variable in this practice.

In the second practice, you use service callout to call a external service via the business service, and then pass the response to the next service.

Practice 8-1: Enriching the Message by Calling a Pipeline Service (Session ID Generation)

A process order can have a session ID, which can be used for testing or auditing across multiple services.

In this practice, you use service callout to invoke another service to get a session ID, and then insert it in the database.

To pass the Session ID between the calling and callee services, you can use either:

- User-defined variables passing via request/response payload of Service Callout

Or

- Shared variable

You use shared variable in this practice.

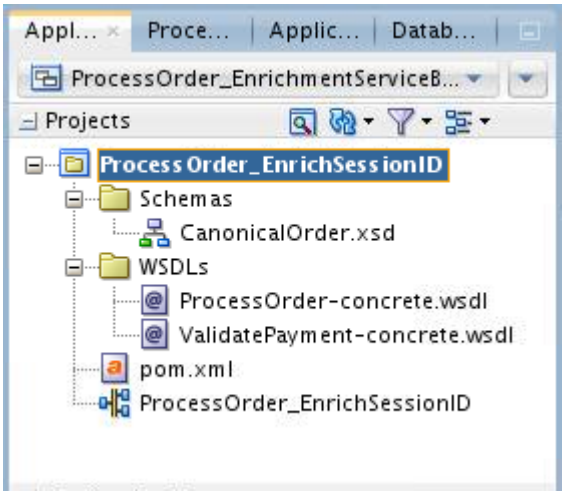
Overview

The high-level steps of this practice are:

- Create the business service, proxy service, and pipeline.
- Create the pipeline to generate the session ID.
- In the main pipeline, add the service callout to invoke the pipeline that generates the session ID, and route it to the business service.

Tasks

1. Open Service Bus project.
   a. In JDeveloper, select **File** > **Open** from the menu bar.
   b. In the Open Application(s) dialog box, navigate to the `$HOME/labs_DI/lessons/lesson08/ProcessOrder_EnrichmentServiceBusApp/` directory, and open the `ProcessOrder_EnrichmentServiceBusApp.jws` file.
   c. Expand the folders in Application Navigator, and you see the folder structure resembling the image below:



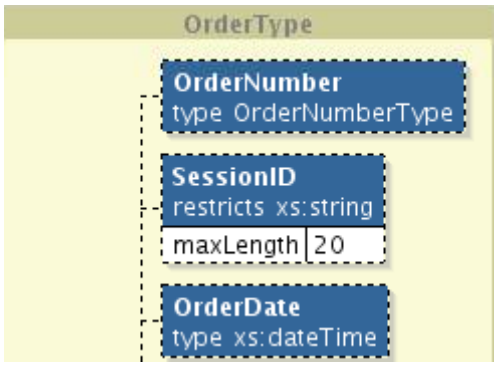
2. Double-click ProcessOrder_EnrichSessionID to open Service Bus Overview Editor.

3. Create the business service to invoke the ProcessOrder SOA service. Using the following information for the creation:
   - Service name: **ProcessOrder_EBS**
   - Transport: **HTTP**
   - WSDL file: **ProcessOrder-concrete.wsdl**
4. Create the proxy service with the pipeline using the following information:
   - Proxy service name: **ProcessOrder_sessionIDPS**
   - Transport: **HTTP**
   - Generate pipeline with name: **ProcessOrder_sessionIDPP**
   - WSDL file: **ProcessOrder-concrete.wsdl**

   After you are done, the service bus Overview Editor should resemble the image below:

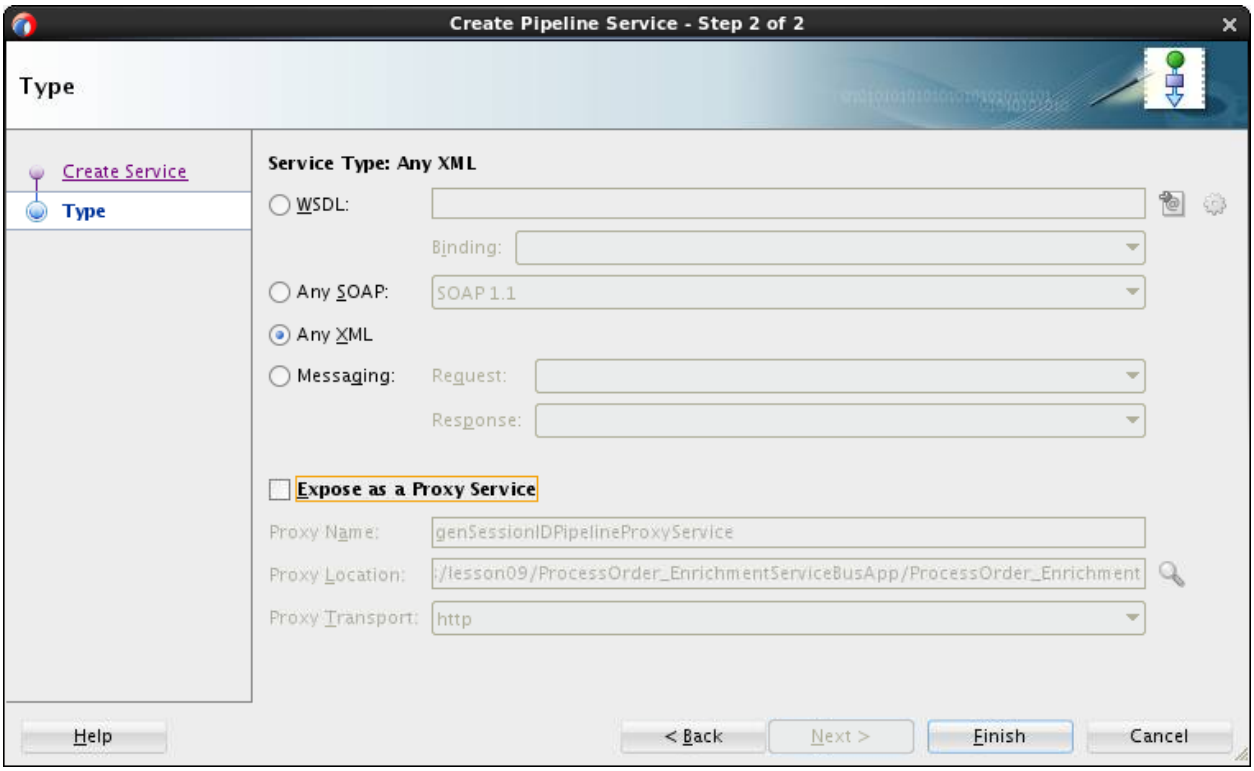
5. Create a new pipeline that generates session ID.

   You will use function `fn-bea:uuid()`, a Service Bus function extension, to generate a universally unique identifier. The generated UUID is a string with over 20 characters, but the length of the sessionID, defined in the schemas, should be no more than 20. So you will use another function `fn:substring` to truncate the UUID to 20 characters or less. The image below (for your reference) shows the sessionID element definition in XSD:



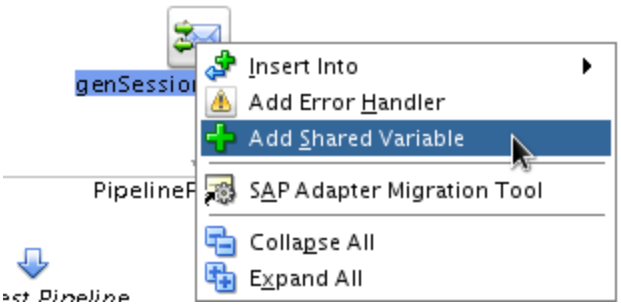
   a. Drag-and-drop the Pipeline component in the center lane of the Overview Editor.
   b. Name the pipeline **`genSessionID`**.
   c. For Service Type, use the default value Any XML.

d. Do not expose the pipeline as a proxy service.



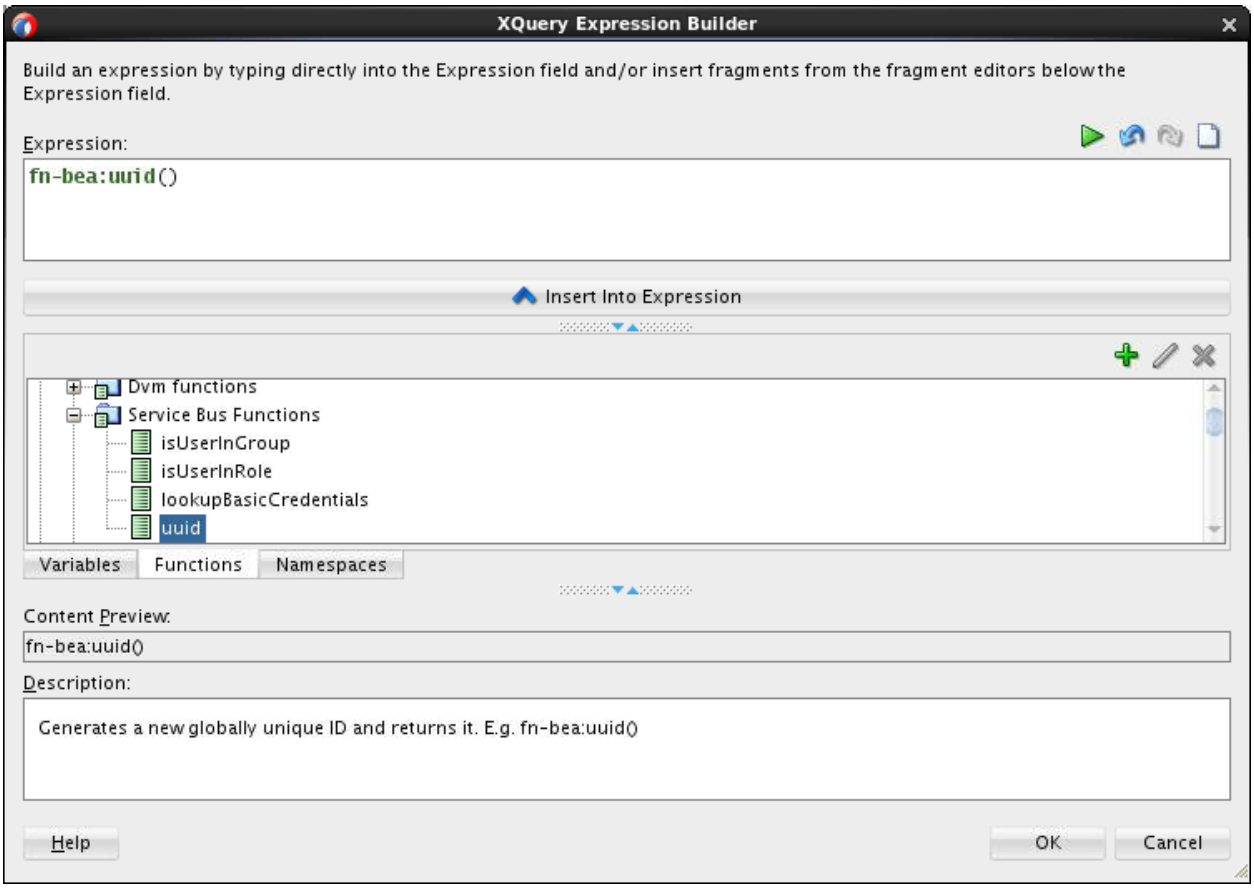
e. Click Finish.

6. Add two XQuery functions to the pipeline.
   a. Open genSessionID in the editor.
   b. Add a shared variable.
      1) Right-click the genSessionID start node, and select Add Shared Variable.



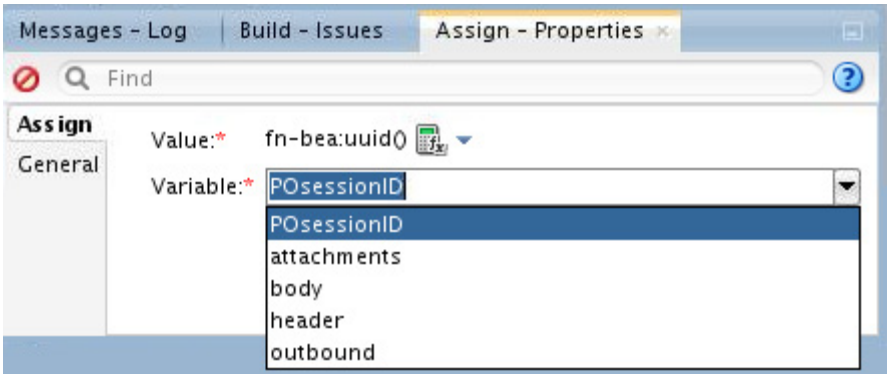
2) Specify the name as **POsessionID**.
3) Click the arrow next to the start node, and you should see the newly added shared variable as shown below:

c. Add a Pipeline Pair node after the genSessionID start node.
d. In the Request pipeline, add an Assign activity in Stage 1.
e. In the Assign Properties window, set the value to the shared variable:
   1) Click  to open XQuery Expression Builder.
   2) Click Functions tab, navigate to Custom XPath Functions > Service Bus Functions, and select **uuid**.
   3) Insert the uuid function in the Expression field.

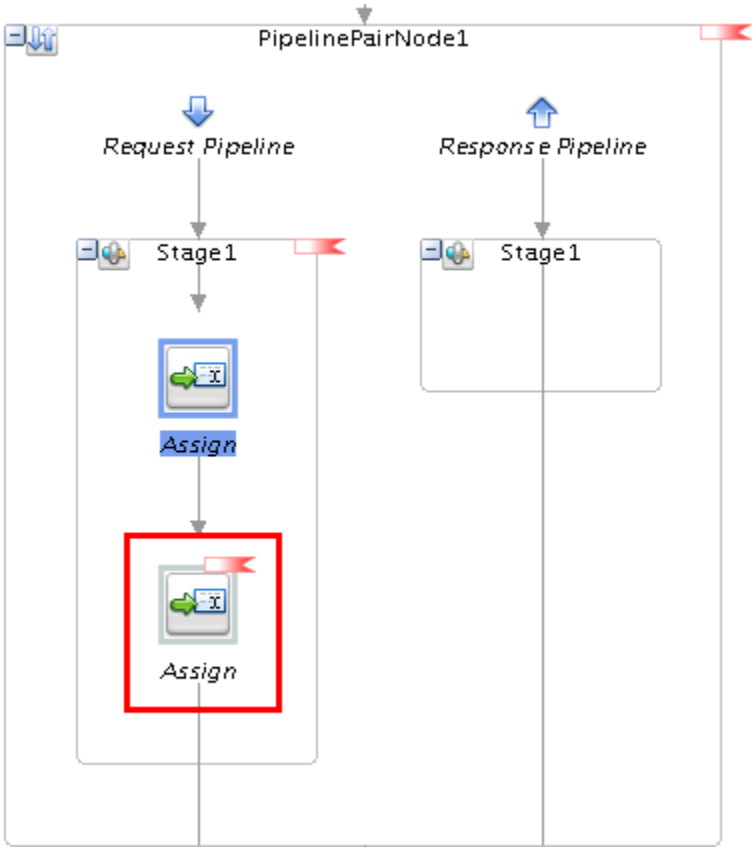


   4) Click OK to close XQuery Expression Builder.

5) Back in the Assign Properties window, select POsessionID for Variable.



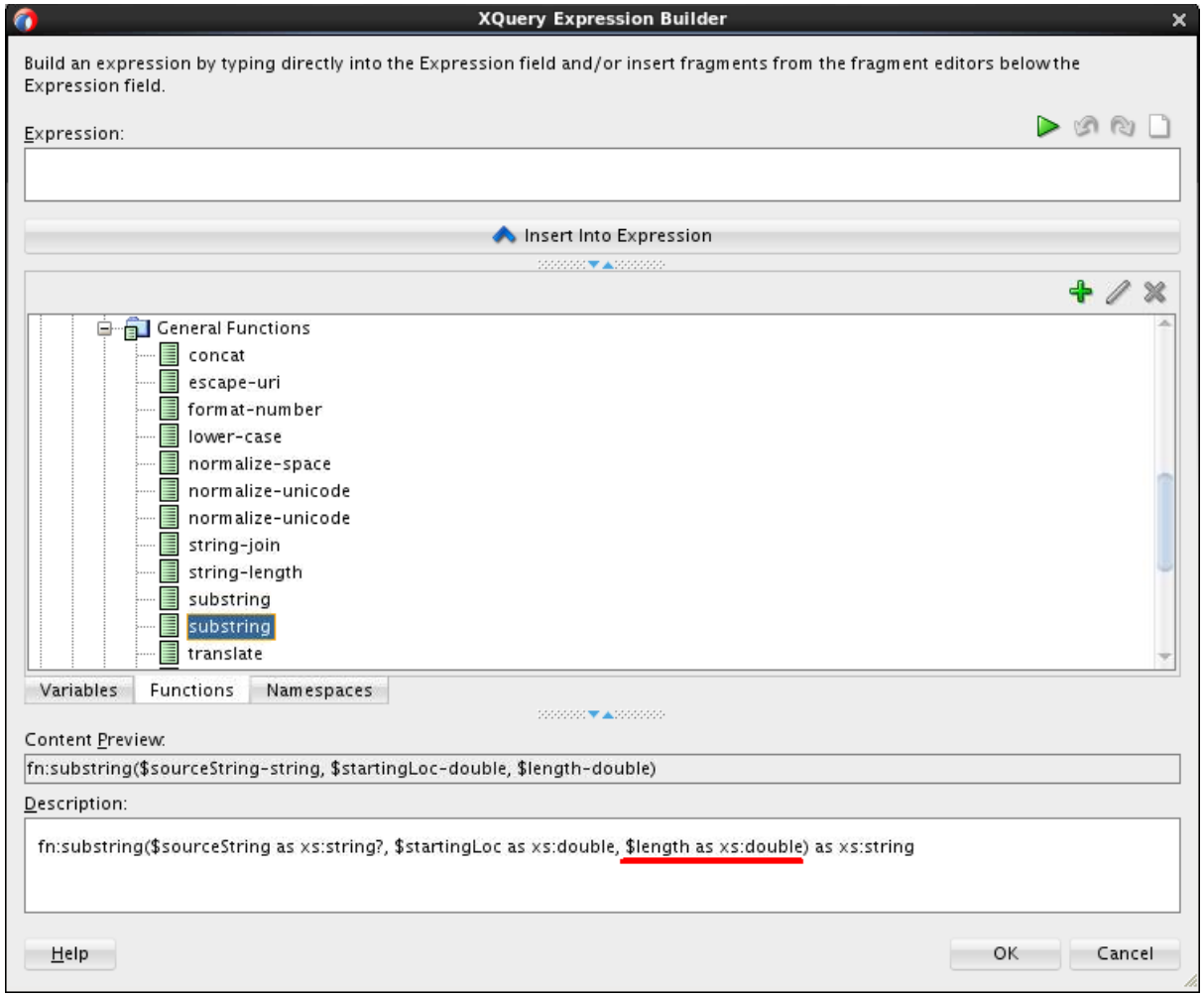
f. Add another Assign activity in Stage1 after the current one.



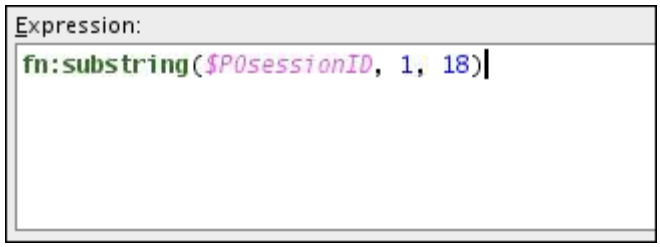
g. In the Assign Properties window,

1) Click  to open XQuery Expression Builder.

2) Click Functions tab, navigate to XQuery 1.0 Functions > Strings > General Functions, and select the second **substring**, the one with $length argument.

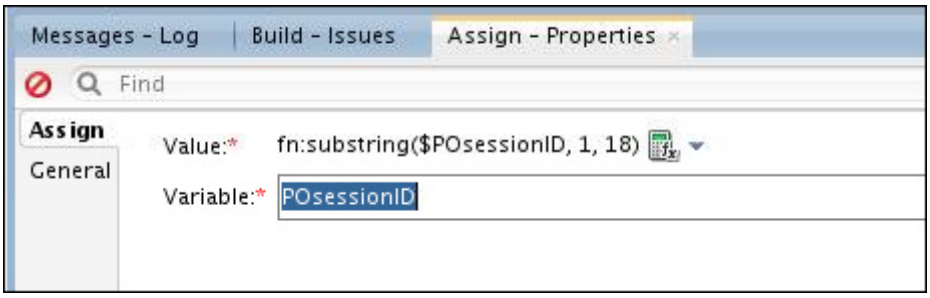


3) Insert the function in the Expression field, and set the value as shown below:



4) Click OK to close XQuery Expression Builder.

5) Back in the Assign Properties window, select POsessionID for Variable.



Note: You can use just one Assign action to get a substring of uuid. The XQuery expression should be:

```
fn:substring(fn-bea:uuid(), 1, 18)
```

7. Save your work.
8. Configure the main pipeline, ProcessOrder_sessionIDPP.

   In this pipeline, you will add a Service Callout action to call the genSessionIDPipeline service. On getting the sessionID value, you need to insert the <SessionID> element into $body as it is not included in the original request payload.

   a. Open the ProcessOrder_sessionIDPP editor

   b. Add the shared variable using the same steps for genSessionID:

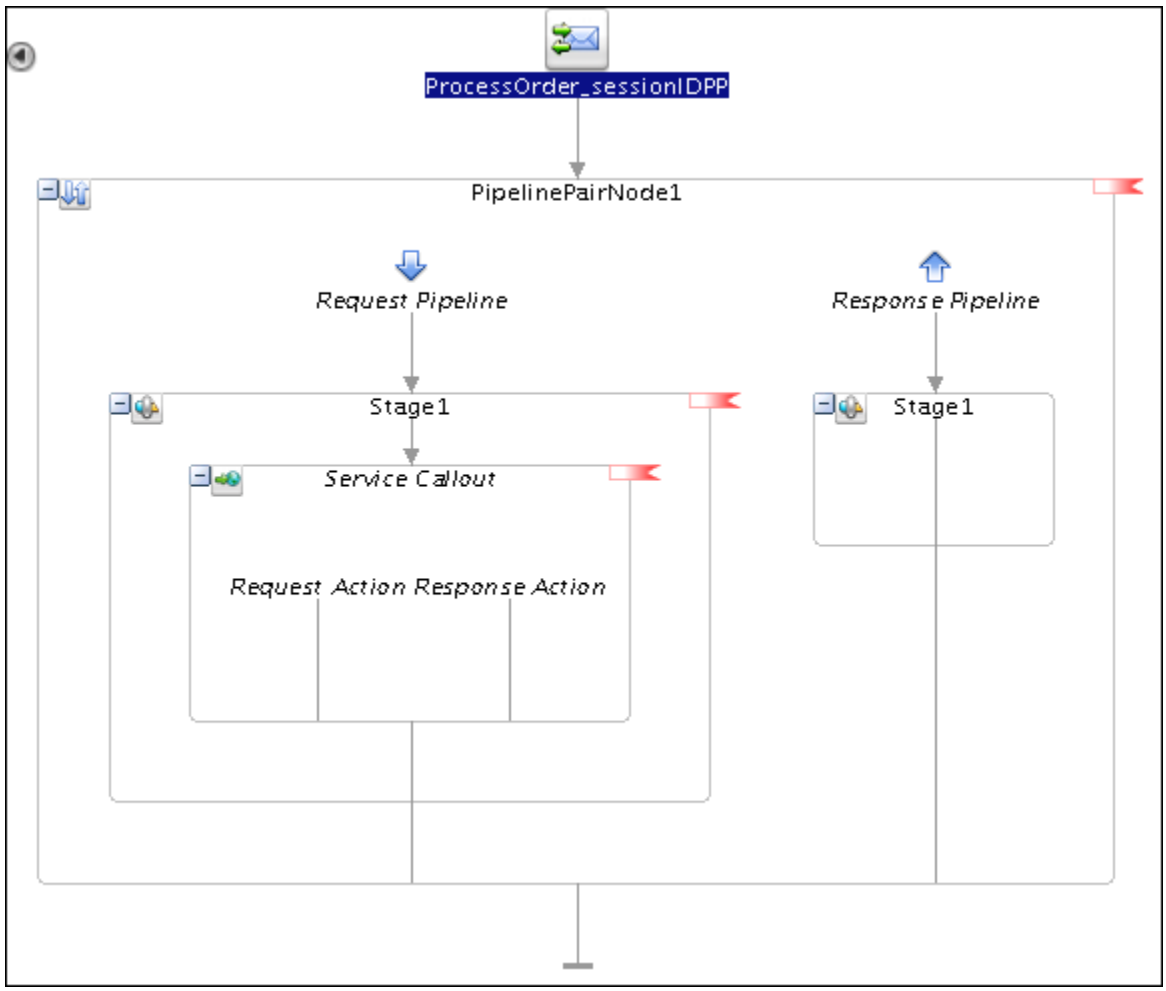
      1) Right-click the ProcessOrder_EPP start node, and select Add Shared Variable.

      2) Specify the name as **POsessionID**.

      3) Click the arrow next to the start node, and you should see the newly added shared variable as shown below:

   c. Add a Pipeline Pair node after the ProcessOrder_sessionIDPP start node.

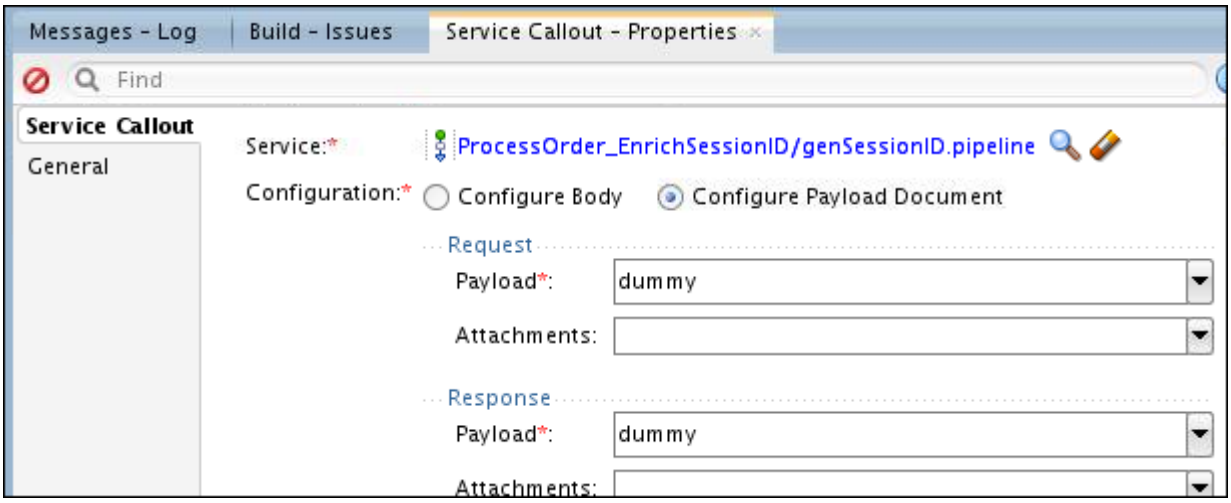
d. In the Request pipeline, drag and drop a Service Callout action (from the Communication section in the Components palette) in Stage One.



e. In the Service Callout Properties window, click Browse, and select **genSessionID** under Resource Chooser for Service.

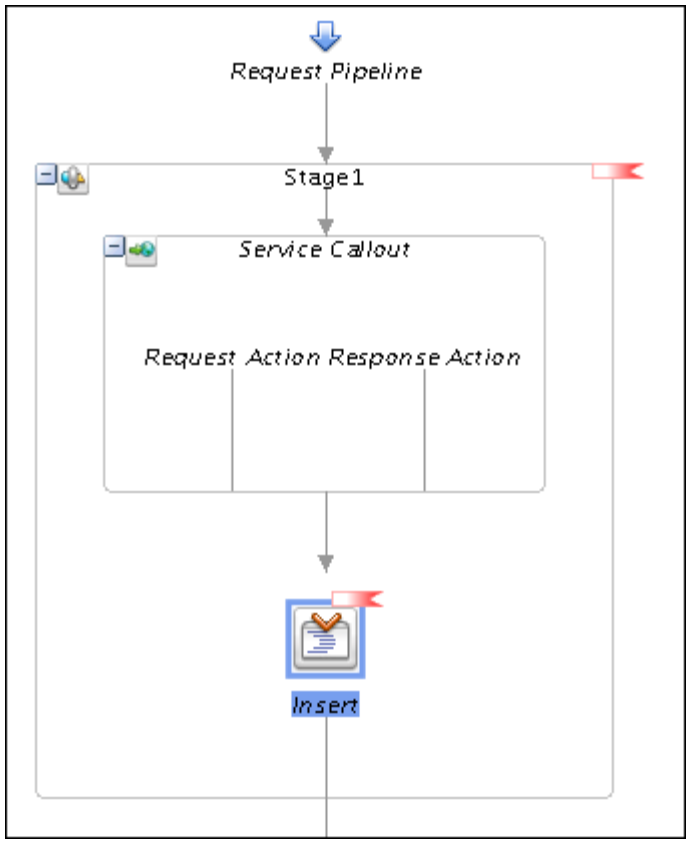
f. The property window is updated with the service that you selected and variables to be configured.

g. Type **dummy** in both Request Payload and Response Payload fields.



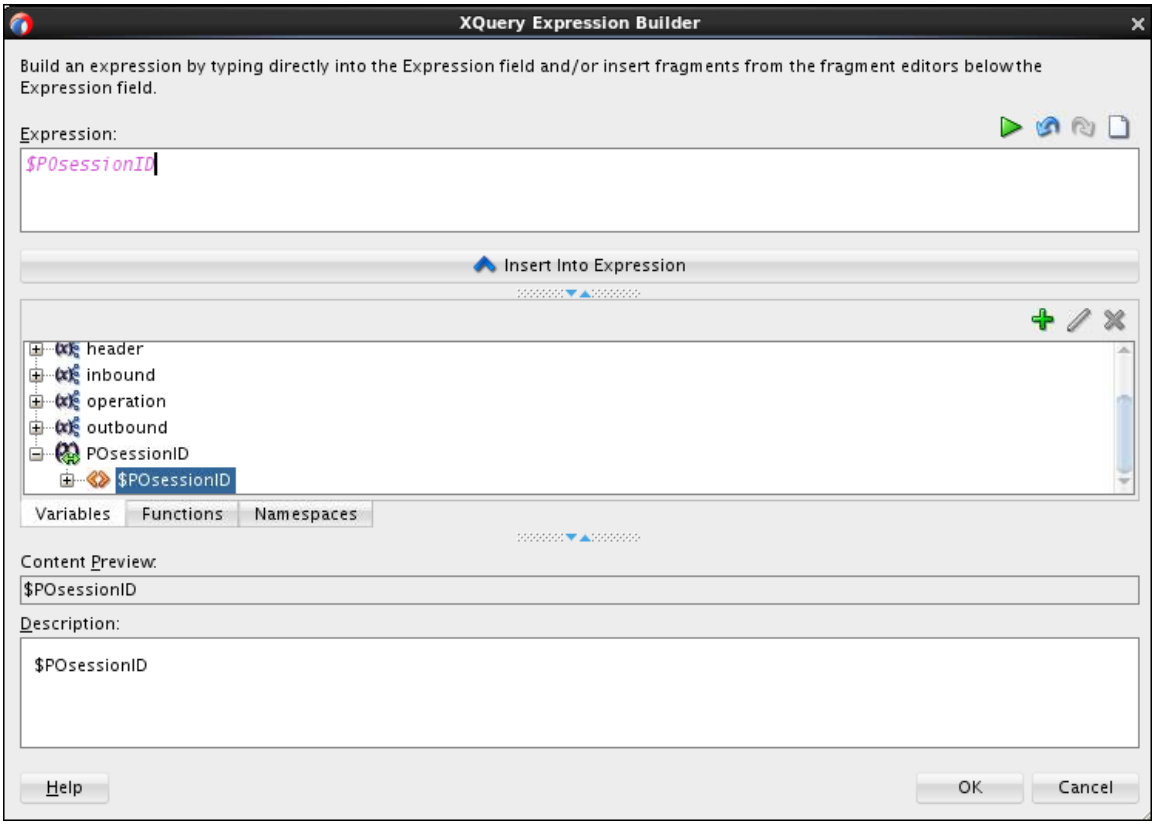
Note: Because you do not use Request and Response payload for passing sessiondID information, you can type any string in both Request Payload and Response Payload fields.

h. Add an Insert action after Service Callout in Stage1.

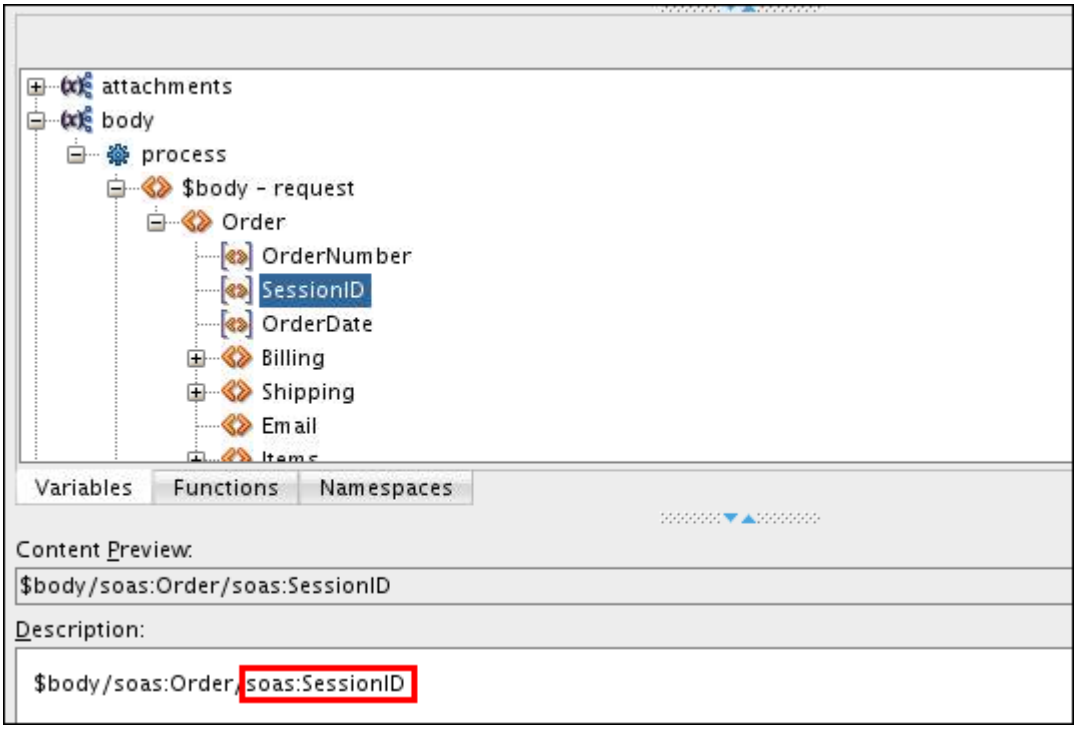


i. In the Insert Properties window, set the values as follows:

1) Click  next to the Value field to open XQuery Expression Builder.
2) Under the Variables tab, select POsessionID and insert it into the Expression field.

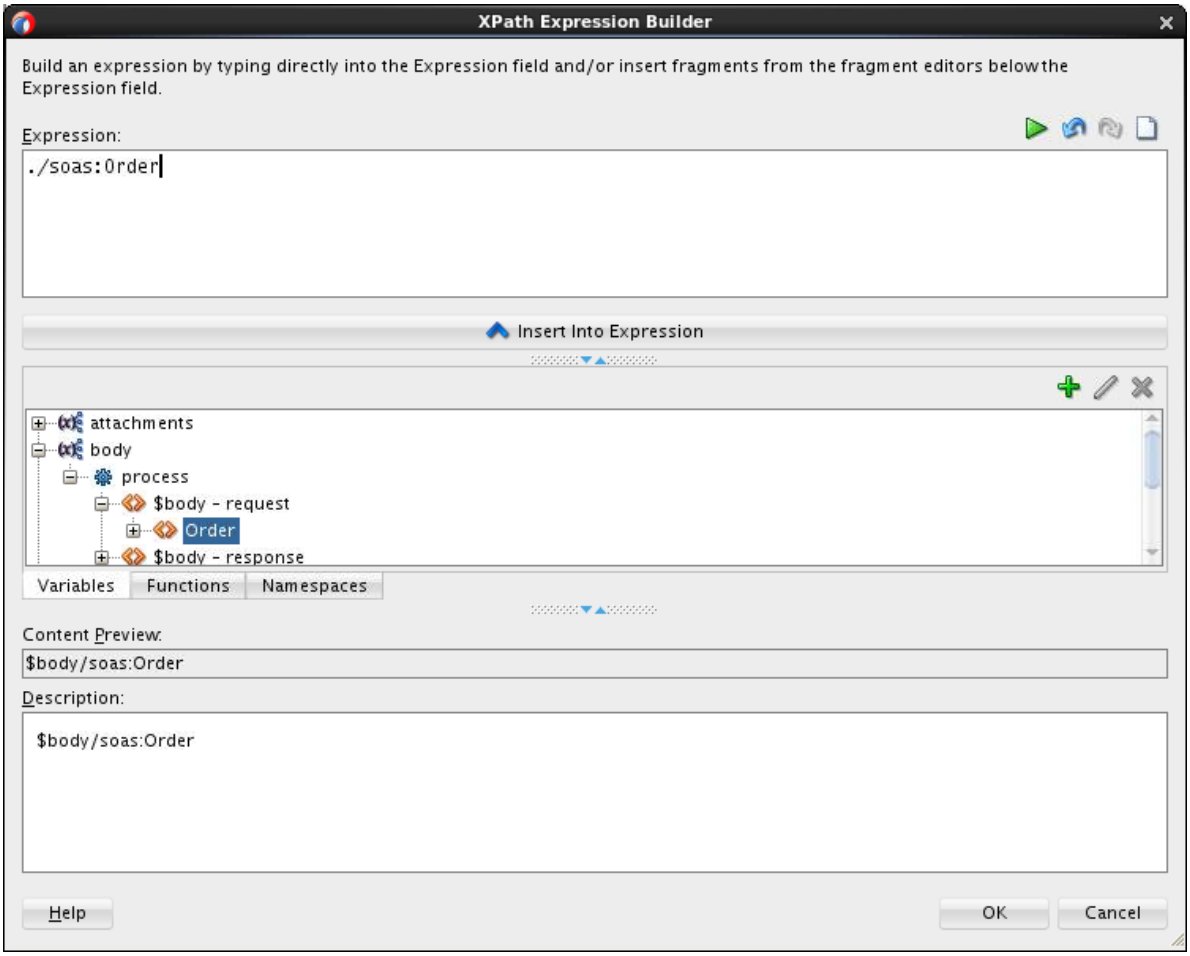


3) To add element tag for $POsessionID, you need to know the namespace prefix and element name. You can find it by navigating to body > process > $body - request > Order > SessionID.



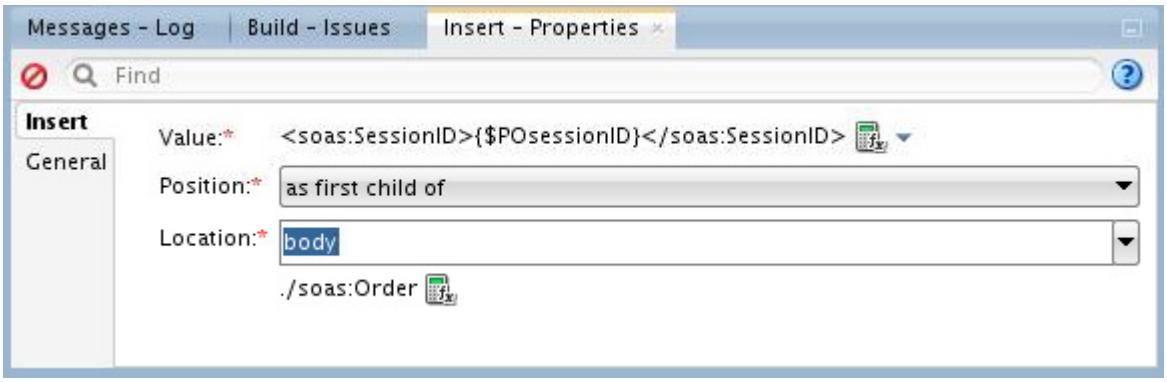
4) With this information, update the Expression field as follow:

5) Click OK to close XQuery Expression Builder.
6) Retain “as first child of” value for Position.
7) In the Location field, select **body** from the drop-down menu.
8) Click  to open XPath Expression Builder.
9) Navigate to body > process > $body - request > **Order**, and insert it to the Expression field.

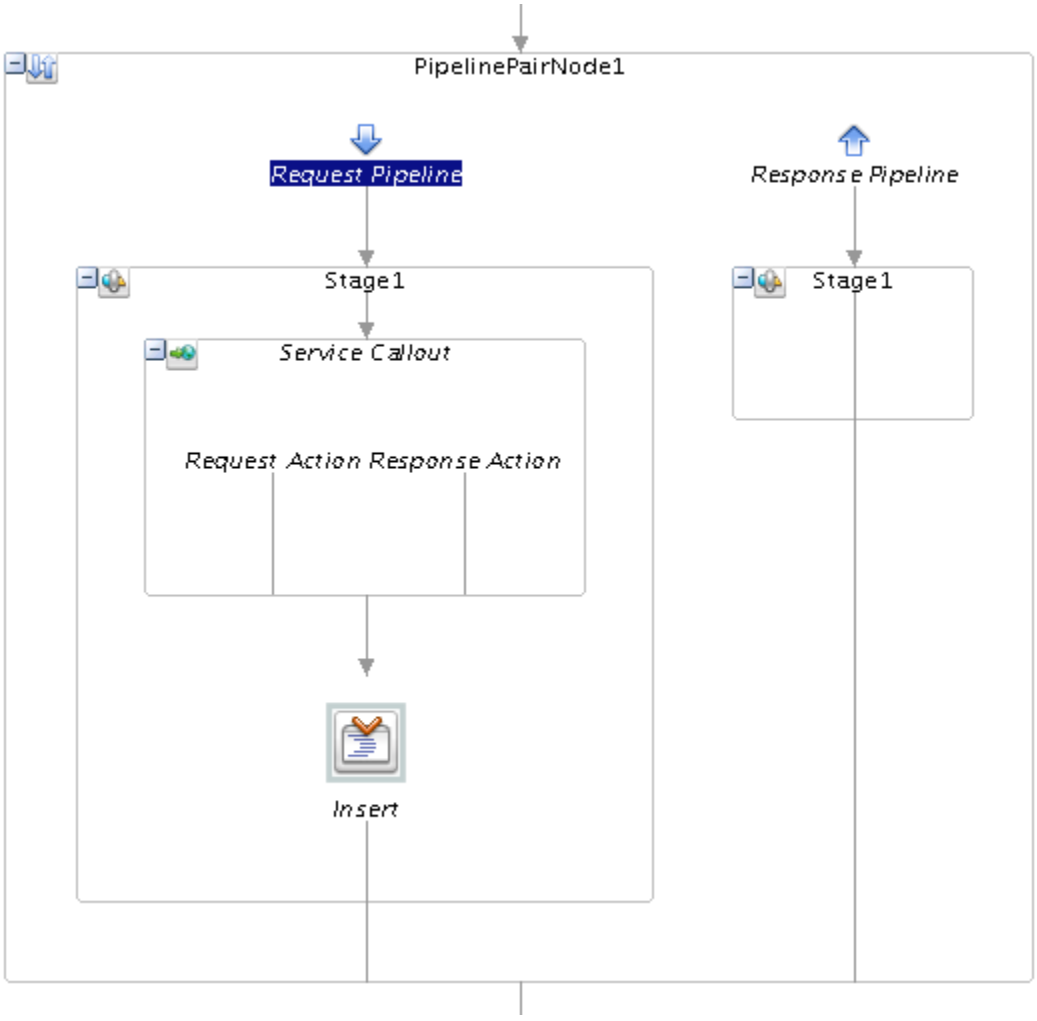


10) Click OK.

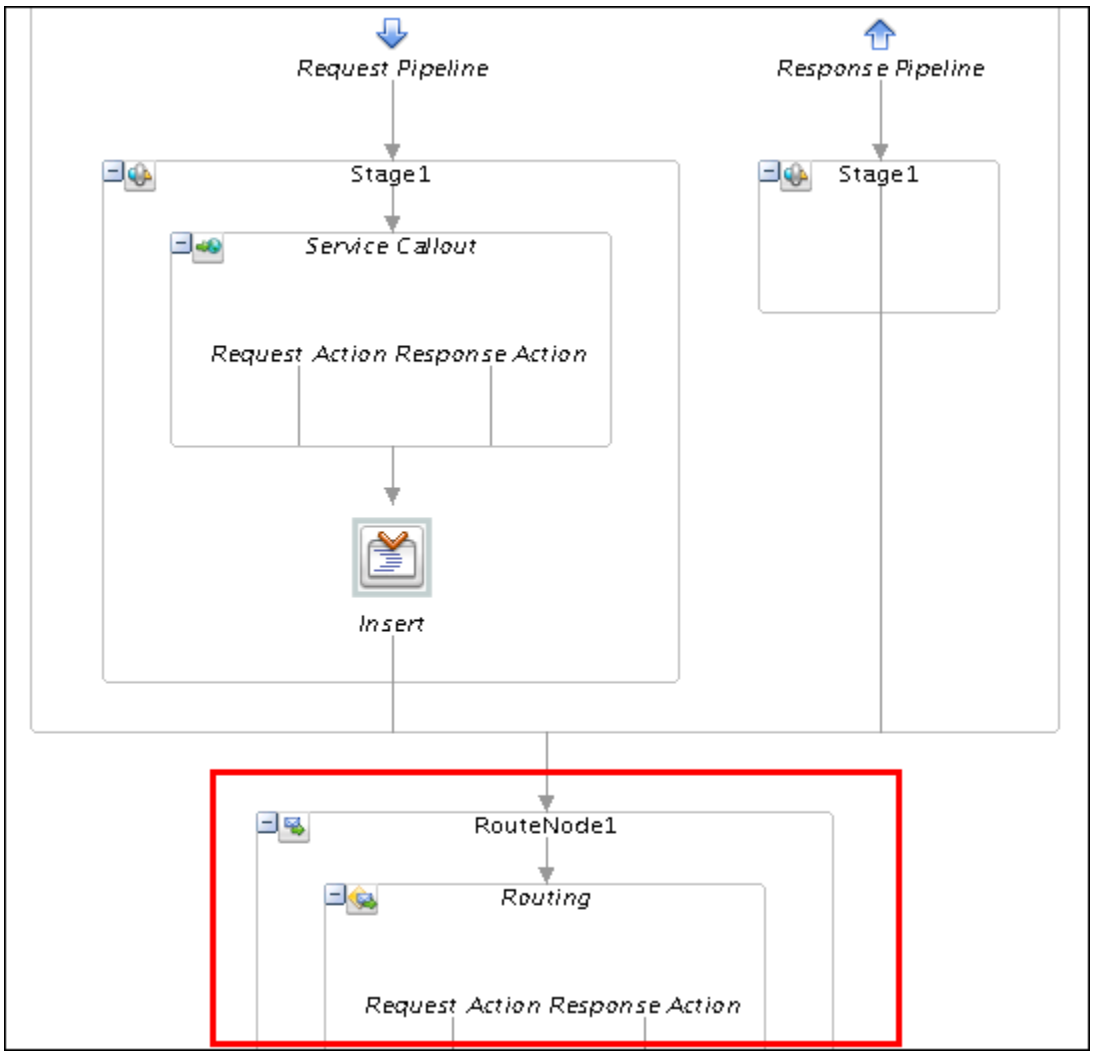
Now your Insert Property window should look like the image below:



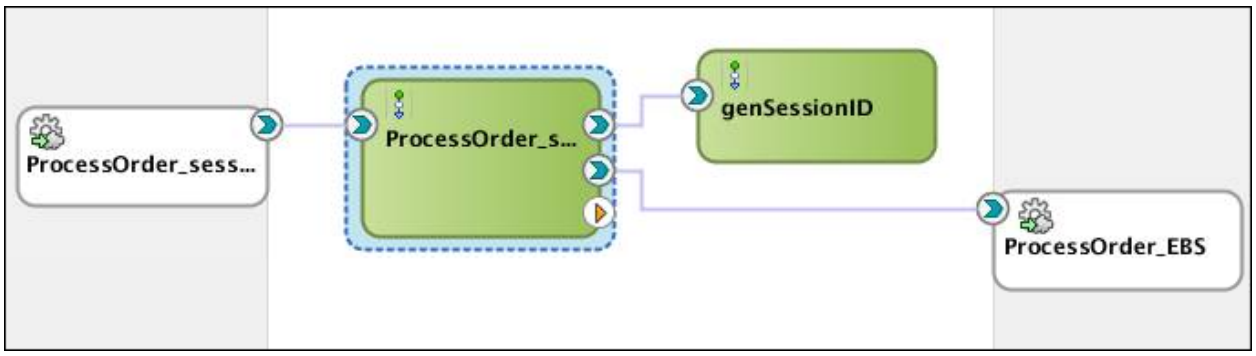
And your request pipeline should resemble the following image:



9. Add a Routing action after PipelinePairnode1, and select ProcessOrder_EBS business service as the destination service.



Your service bus project should look like the image below in the Overview Editor:



10. Save your project.

Deploy and Test

11. To test your Service Bus application:
    a. In the project Overview Editor, right-click the proxy service, and select Run.
    b. In the Test console, select **~/labs_DI/resources/sample_input/OrderSample.xml**

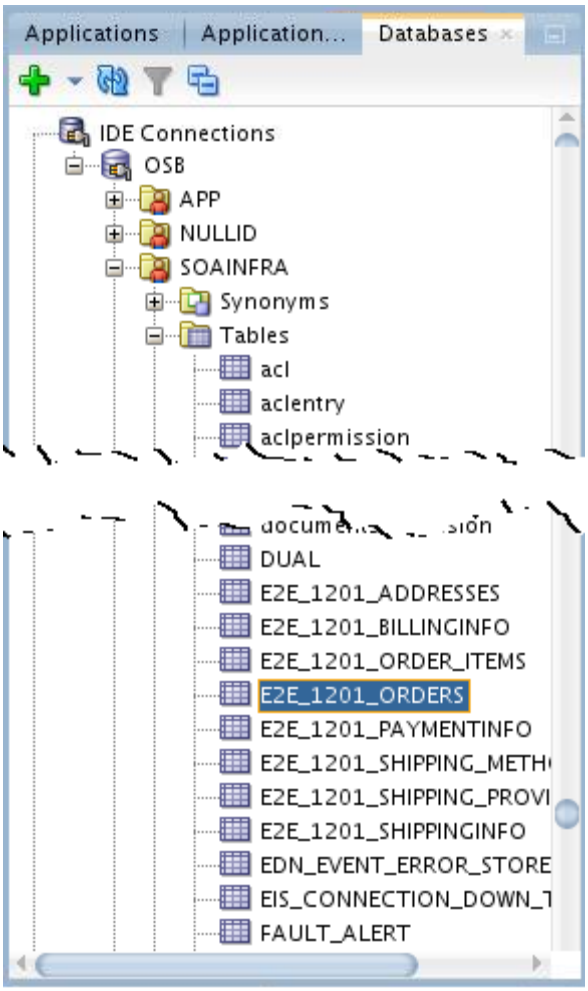
c. Click the Execute button.

You should see the order number in the response message.

d. Verify the sessionID inserted in the database.

1) From the JDeveloper main menu, select Window > Database >Databases. The Databases window is displayed.

2) Expand IDE Connections folder, navigate to OSB > SOAINFRA > Tables, and locate the E2E_1201_ORDERS table in the list.



3) Right-click the table, and select **Open Object Viewer** from the context menu. You see that the latest order entry in the database includes a sessionID.

24	201552912515353	f936ad04-9077-4139	2015-05-29	105.75999999999999	daniel@localhost	New

Practice 8-2: Enriching the Message by Calling a Business Service (ValidateCreditCard)

Overview

In this practice, you use service callout to call a service and pass the response from the service to the next service.

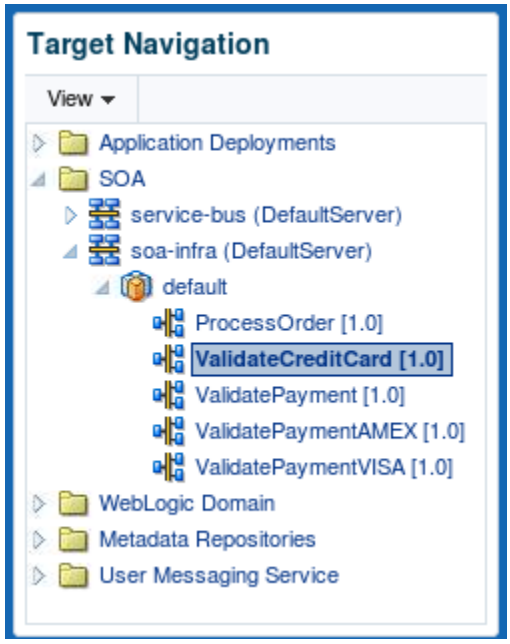
Tasks

Setups

1. Deploy the credit card validation composite service.
   a. In a terminal window, enter the following commands:

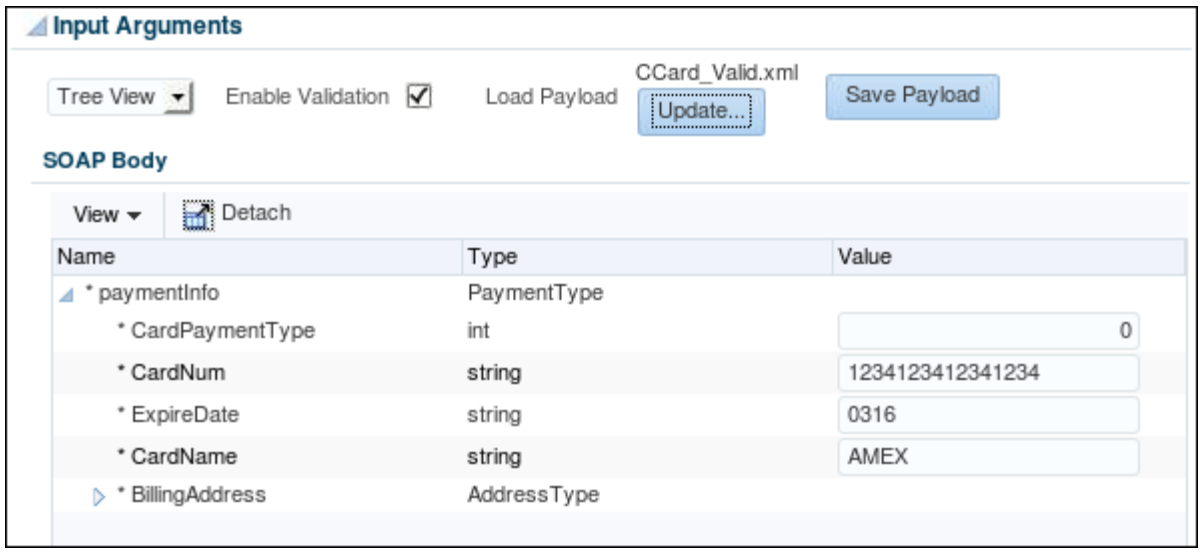
      ```
      >cd $HOME/labs_DI/lessons/lesson08
      >ls -l
      >$HOME/labs_DI/resources/Scripts/deploy.sh
      sca_ValidateCCard_rev1.0.jar
      ```

      Make sure you see that the build is successful.
   b. Open EM console. In the Target Navigation panel, expand SOA > soa-infra > default. Now you should see the newly-deployed two composite applications in the folder.

      This service only checks the credit card number and expiration date. It won't compare the total order amount with the daily limit.
2. You can test this service as follows:
   a. On the ValidateCreditCard [1.0] home page, click Test.
   b. On the Test Web Service page, accept all default settings. Note down the URL of the service. You will use it later in this practice.
   c. On the Request tab, click Browse, and select `/home/oracle/labs_DI/resources/sample_input/CCard_Valid.xml` message.

d. Review the message body and note that it does not include the total order amount (AuthorizationAmount).



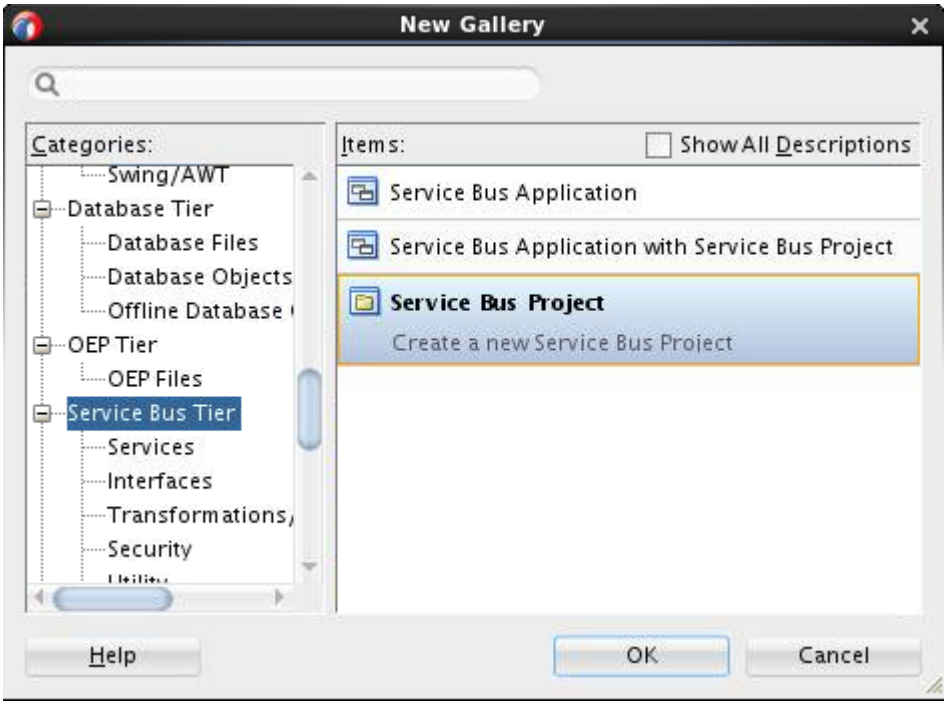
e. Click Test Web Service.

f. Review the response message, and you should see that the payment status is “Valid”.

3. Create a new Service Bus project.

a. In JDeveloper, select **File** > **New > Project** from the main menu.

b. Select **Service Bus Tier** > **Service Bus Project**.



c. Click **OK**.

d. Name the project: **ProcessOrder_EnrichStatusUpdate**.

e. Click **Finish**.

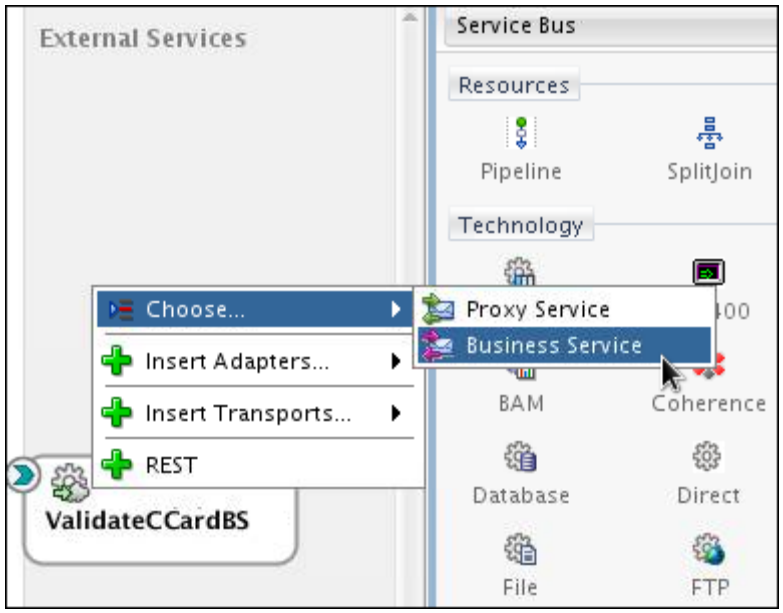
You see that a new project folder is added in Application Navigator.

4. Double-click ProcessOrder_EnrichStatusUpdate to open Service Bus Overview Editor.

5. Create the business service to invoke the ValidateCreditCard SOA service that you just deployed.
   a. Drag the HTTP transport from the Technology section in the Component palette and drop onto the External Services lane.
   b. Enter the required details for business service by following the instructions in the table below:

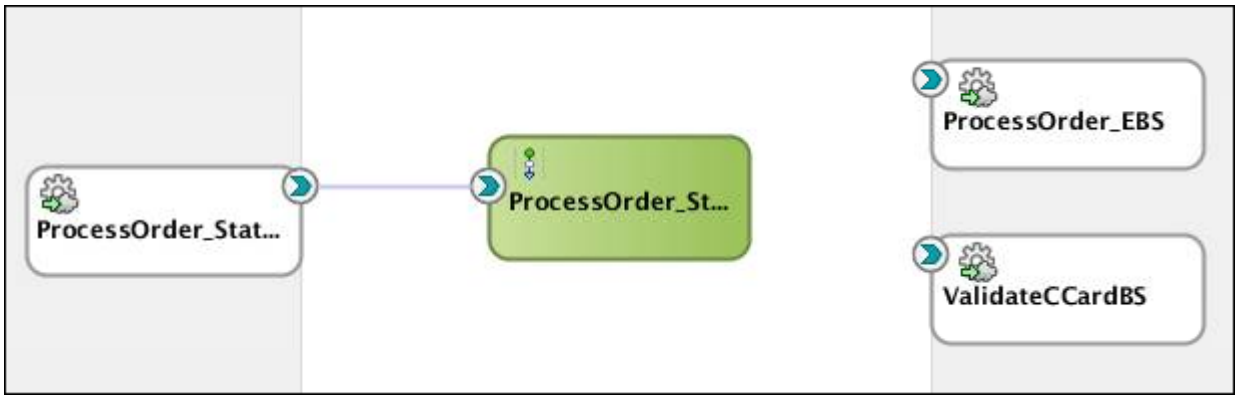
Step	Window/Page Description	Choices or Values
a)	Create Service	Name the service `ValidateCCardBS`. Click Next.
b)	Type	• 1) Select **WSDL** for Service Type. • 2) Click the Browse WSDLs icon to the right of the WSDL choice. • 3) In the Select WSDL dialog box, click **Application**, navigate and select **ValidatePayment-concrete.wsdl.** • 4) Click **OK**. • 5) Back in the Create Business Service window, you should see the WSDL and port fields populated with the information based on the WSDL that you chose. • 6) Click **Next**.
c)	Transport	Verify that http is selected in the Transport field. Change the Endpoint URI to point to the ValidateCreditCard composite: `http://localhost:7101/soa-infra/services/default/ValidateCreditCard/validateCCard_client_ep` Click **Finish**.

6. Create the business service to invoke the ProcessOrder SOA service, which is your target service. This means that the message is routed to this service. Here, you can reuse the business service created in the previous project:
   a. Right-click in the External Services lane, and select **Choose** > **Business Service** from the drop-down menu.



   b. Under the Resource Chooser, navigate to Application > ProcessOrder_EnrichSessionID, and select **ProcessOrder_EBS.bix**.
   c. Click OK.
7. Create the proxy service with the pipeline using the following information:
   - Proxy service name: **ProcessOrder_StatusUpdatePS**
   - Transport: **HTTP**
   - Generate pipeline with name: **ProcessOrder_StatusUpdatePP**
   - WSDL file: **ProcessOrder-concrete.wsdl**

   After you are done, the service bus Overview Editor should resemble the image below:

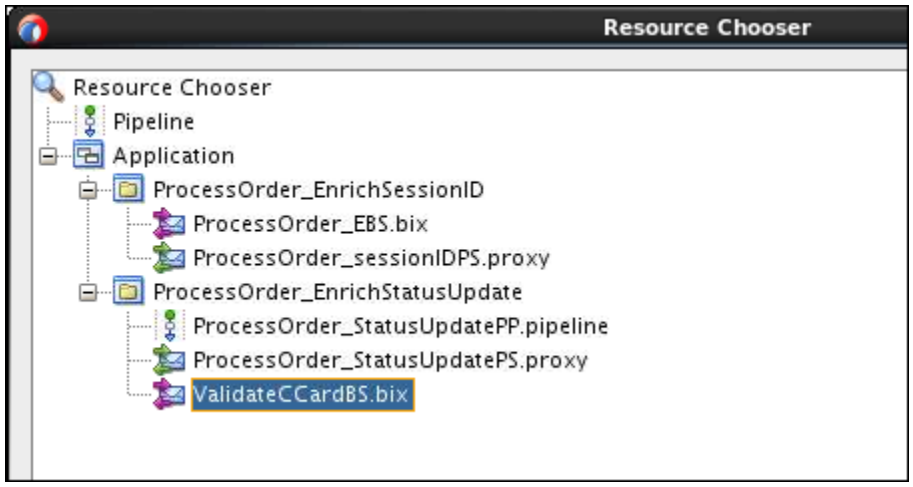


8. Configure the pipeline by adding a Service Callout action to invoke the ValidateCCardBS business service and pass the Service Callout response to the ProcessOrder business service.
   a. Double-click the pipeline to open its editor.

b. Drag Pipeline Pair from the Nodes section in the Components palette and drop it below the Start node.

c. In the Request pipeline, drag and drop a Service Callout action (from the Communication section in the Components palette) in Stage One.

d. In the Service Callout Properties window, click Browse, and select **ValidateCCardBS** under Resource Chooser for Service.

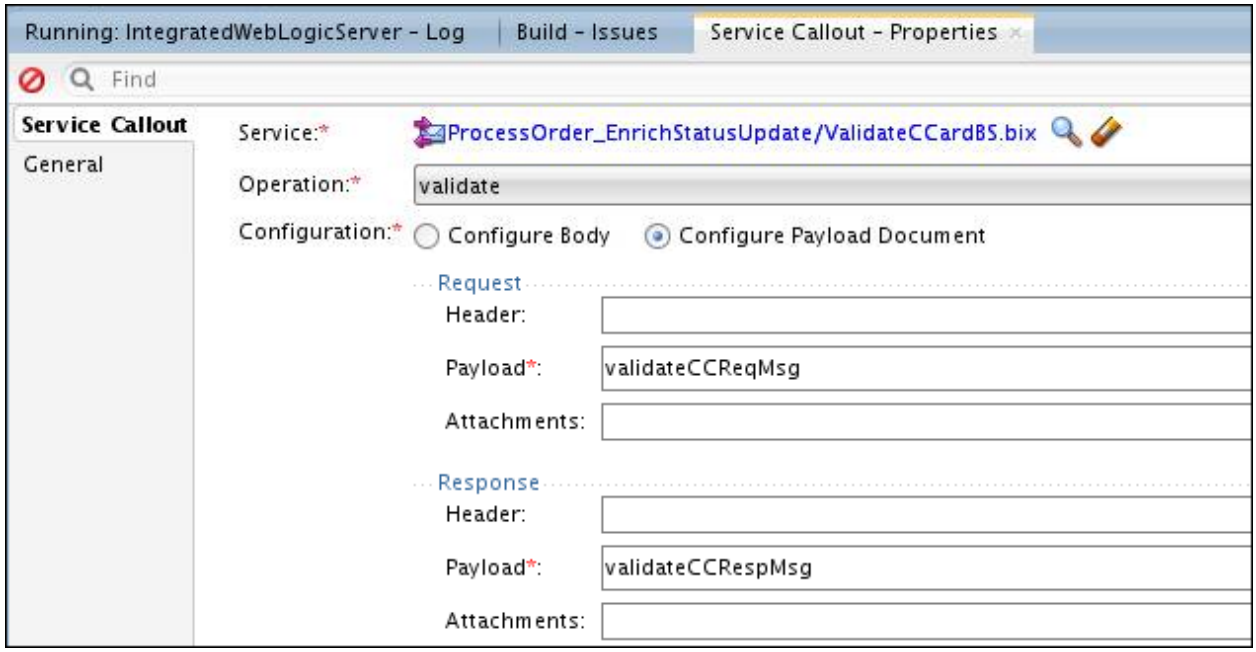


The property window will be updated with the service that you selected and the variables to be specified.

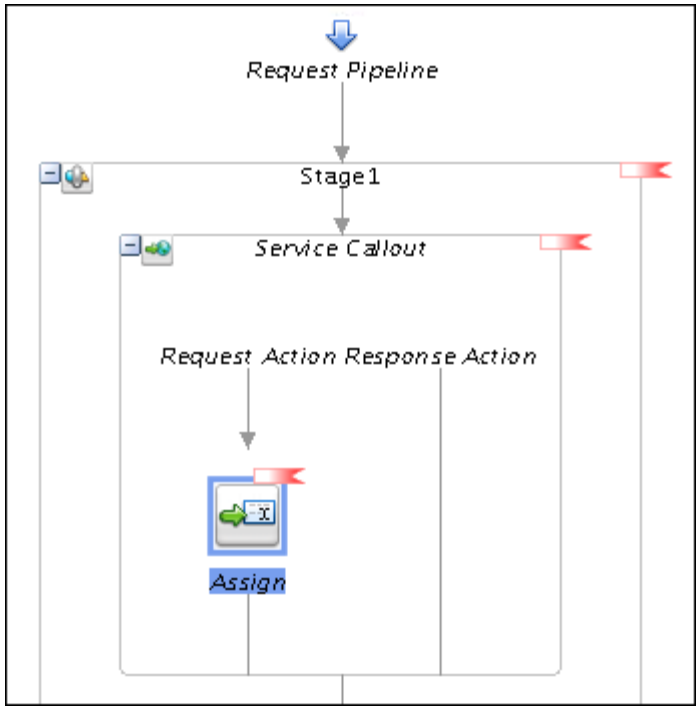
e. Create two variables inside the message flow.

1) Click the Request Payload field, and type **`validateCCReqMsg`**.

2) Click the Response Payload field, and type **`validateCCRespMsg.`**

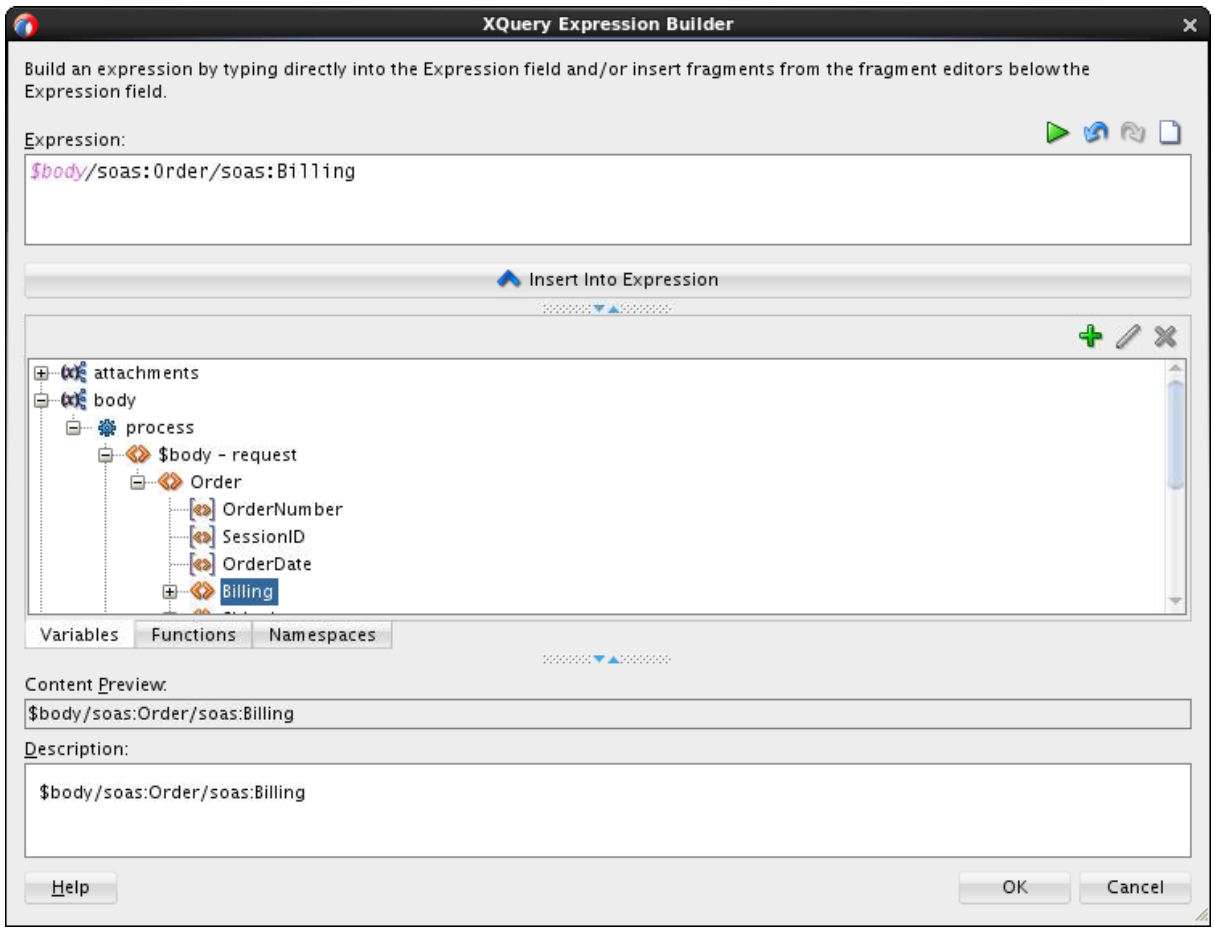


9. Add an Assign action to pass the payment information from Proxy request message to the `validateCCReqMsg` variable.
   a. Drag an Assign action and drop it below the Request Action of Service Callout. Use the following image as a guide:

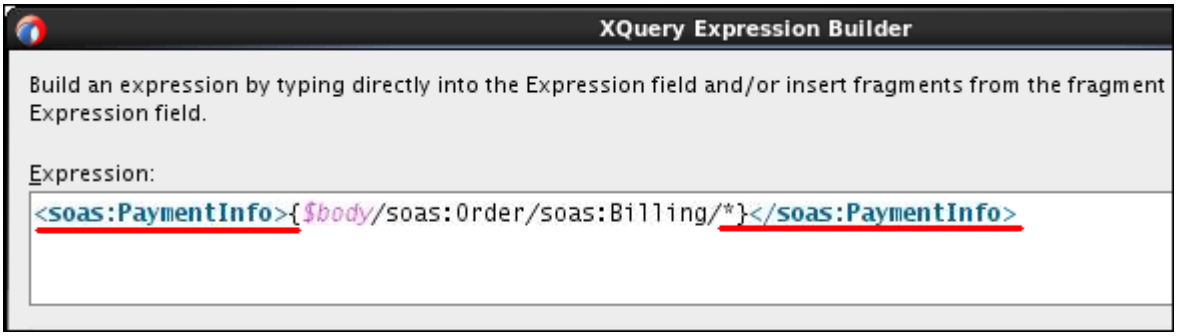


   b. In the Assign Properties window, set the value to the validateCCReqMsg variable:
      1) Click  to open XQuery Expression Builder.

2) On the Variables tab, navigate to body > process > $body – request > Order, select **Billing,** and insert it into the Expression field.



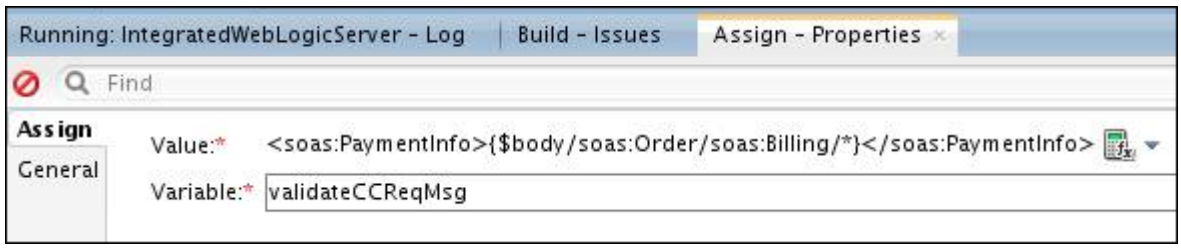
3) However, ValidateCreditCard SOA service expects the `PaymentInfo` element in the request payload, so you need to wrap the content of `Billing` element in the `PaymentInfo` element. To do so, add the following code(underlined in red) in the Expression field:



Note: You can also create an XQuery/XSLT transformation to convert from the Billing element to the PaymentInfo element.

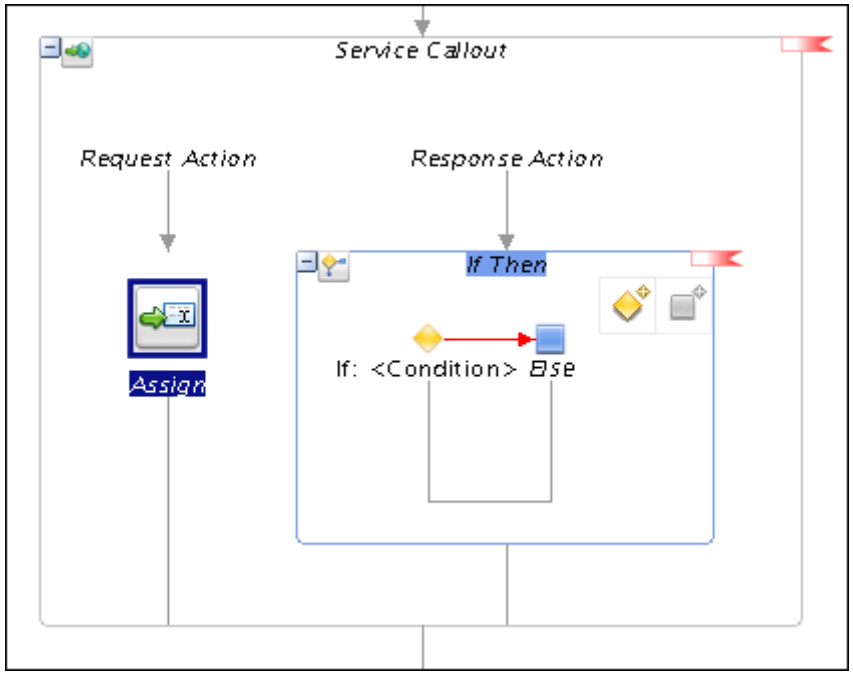
4) Click OK to close XQuery Expression Builder.

5) Back in the Assign Properties window, type `validateCCReqMsg` in the Variable field.



10. Use Service callout response message, add it to pipeline request message, and pass it to the target service.

a. Drag an **If Then** action from the Flow Control seciton in Components palette and drop it below Response Action of Service Callout. Use the following image as a guide:



b. Select **If: <Condition>.**

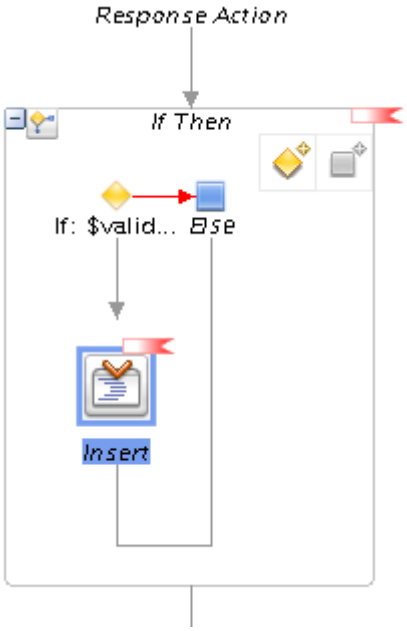
c. In the If Properties window, Click  to open XQuery Expression Builder.

1) Type `$validateCCRespMsg="Invalid"` in the Expression field.

2) Click OK.

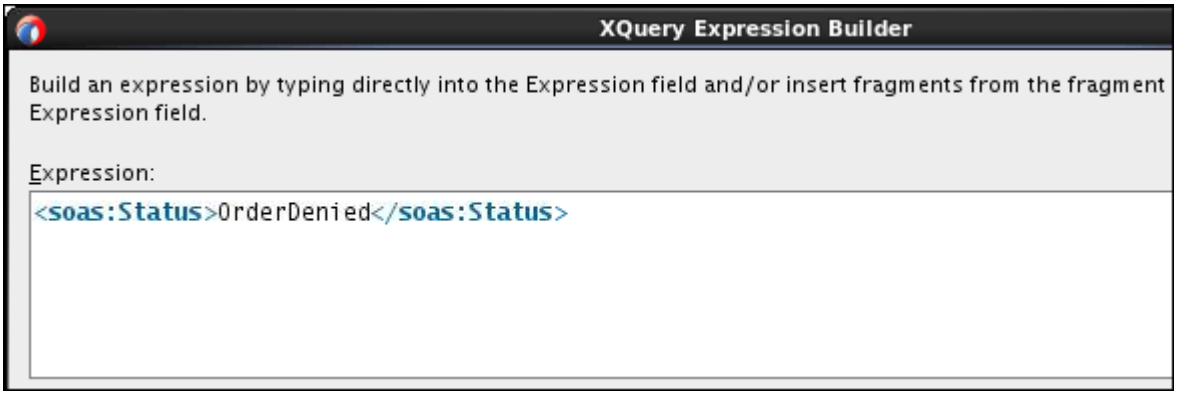
The If Properties window is updated as shown in this image:

d. Drag an Insert action and drop it on to an If condition branch.



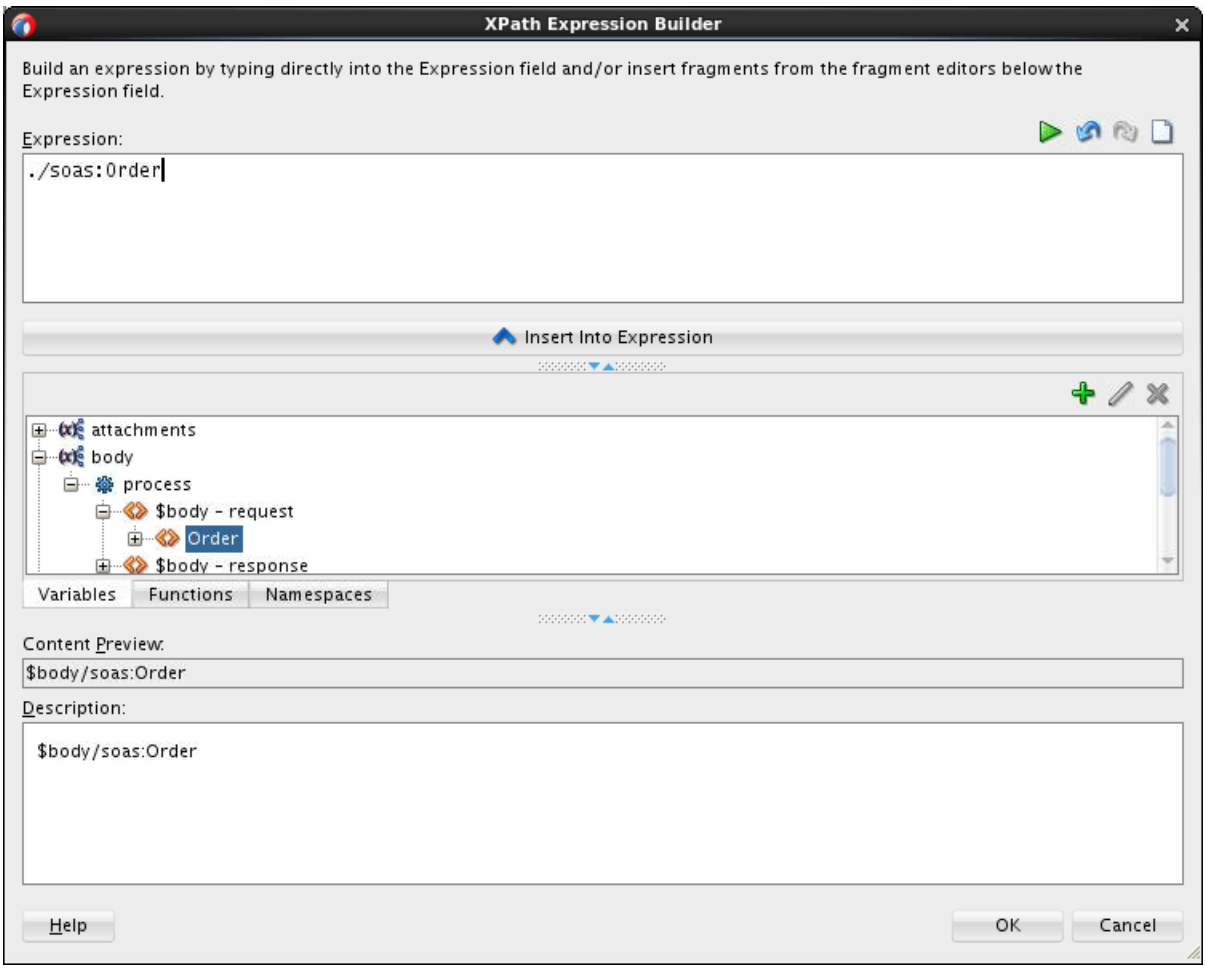
e. In the Insert Properties window, set the values as follow:

1) Click  next to the Value field to open XQuery Expression Builder.
2) Type the following code in the Expression field.



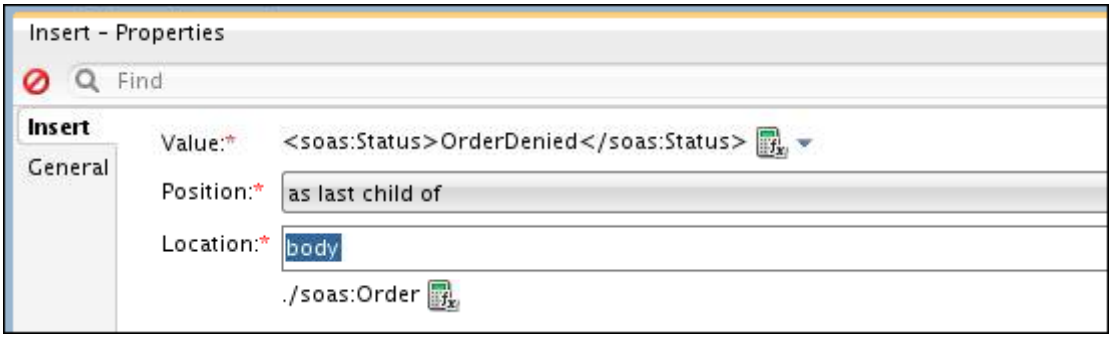
3) Click OK to close XQuery Expression Builder.
4) In the Position field, select "**as last child of**" from the drop-down menu.
5) In the Location field, select **body** from the drop-down menu.
6) Click  to open XPath Expression Builder.

7) Navigate to body > process > $body - request > **Order,** and insert it to the Expression field.



8) Click OK.

Now your Insert Property window should look like the image below:

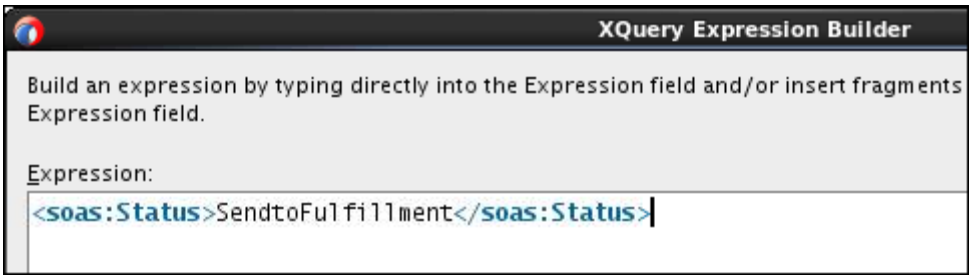


f. Similarly, add an Insert action in the ELSE condition branch.

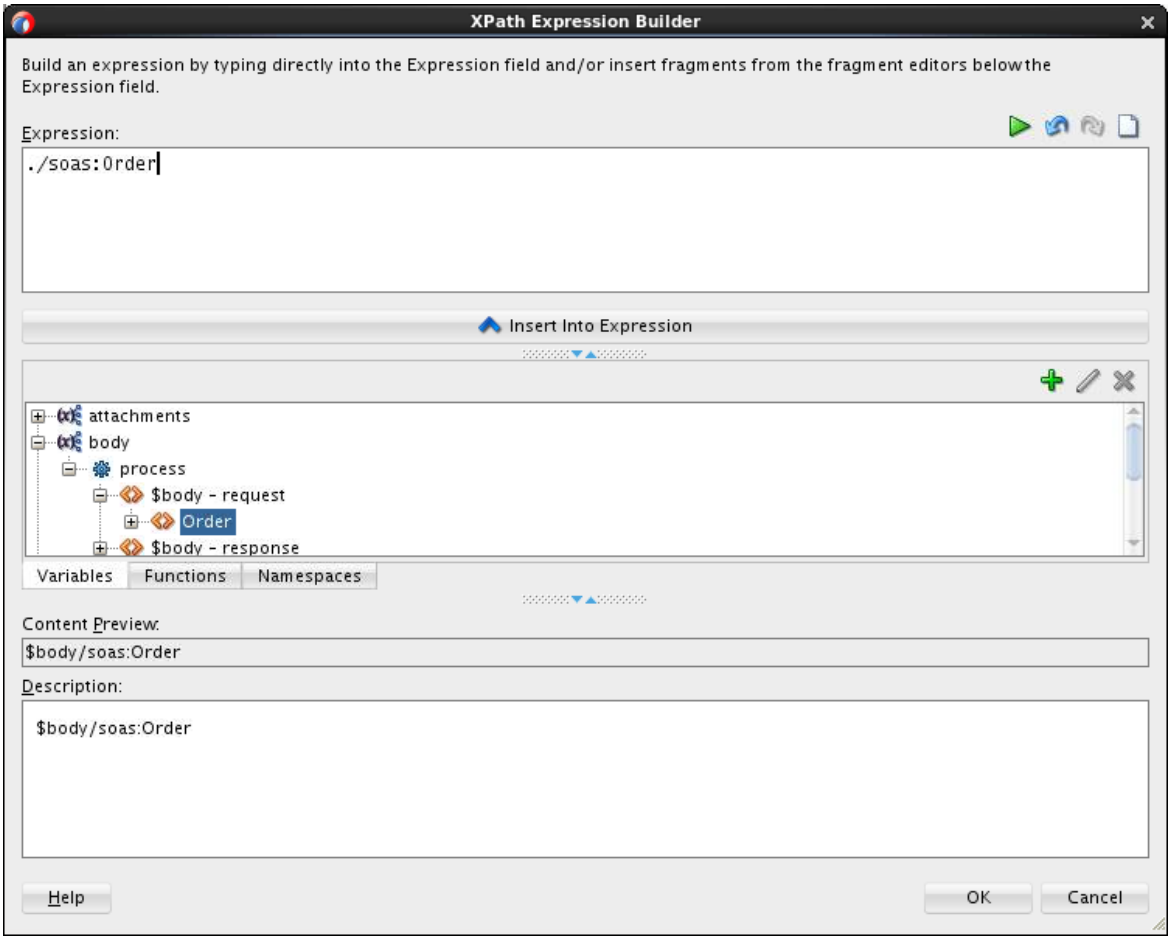
g. In the Insert Properties window, set the values as follow:

1) Click  next to the Value field to open XQuery Expression Builder.

2) Type the following code in the Expression field.

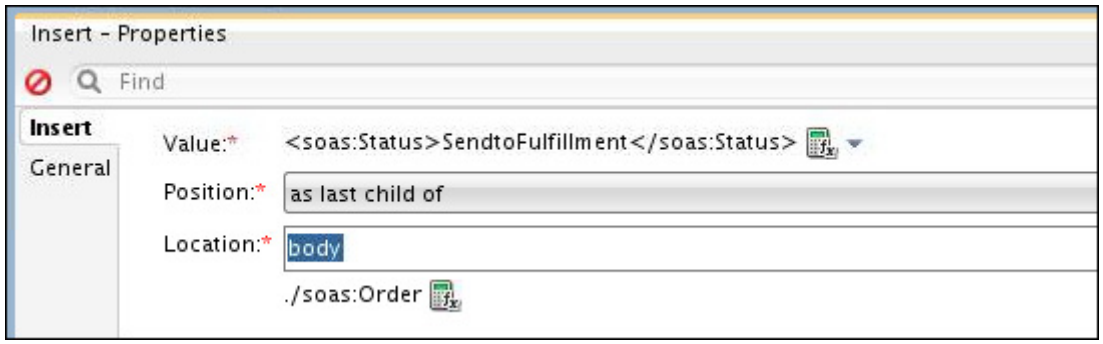


3) Click OK to close XQuery Expression Builder.
4) In the Position field, select "**as last child of**" from the drop-down menu.
5) In the Location field, select **body** from the drop-down menu.
6) Click  to open XPath Expression Builder.
7) Navigate to body > process > $body - request > **Order**, and insert it to the Expression field.

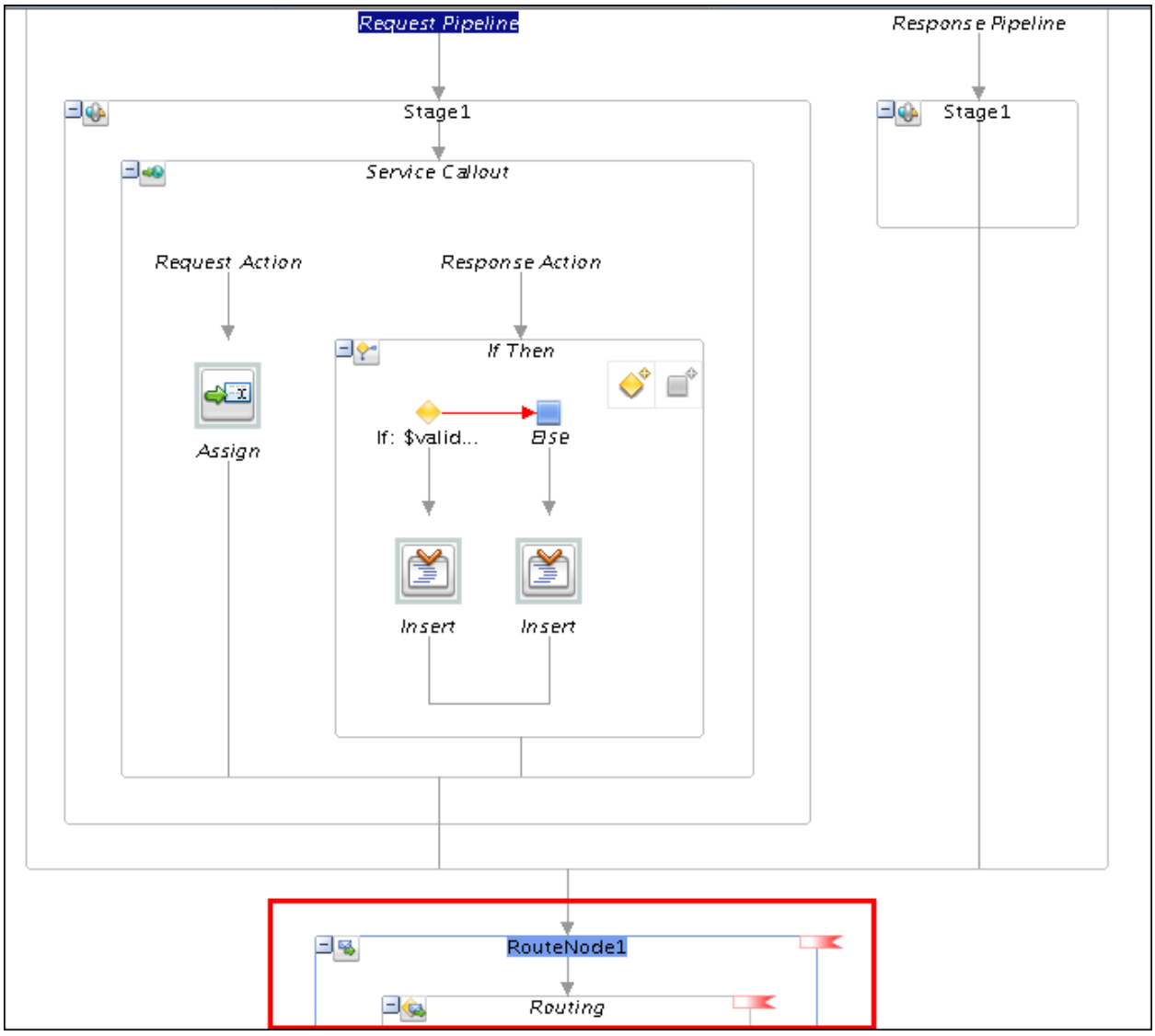


8) Click OK.

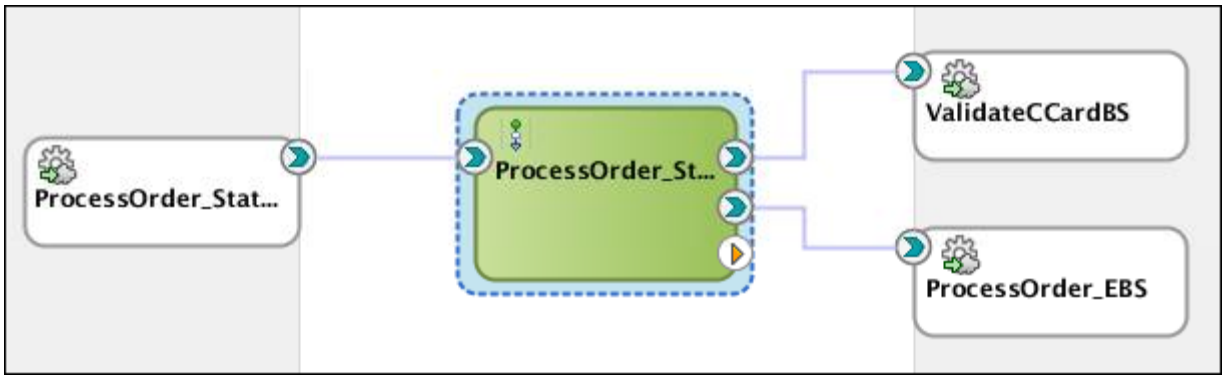
Now your Insert Property window should look like the image below:



11. Add a Routing action below PipelinePairnode1, and select ProcessOrder_EBS.bix business service as the destination service.



Your Service Bus project should look like the image below in the Overview Editor:

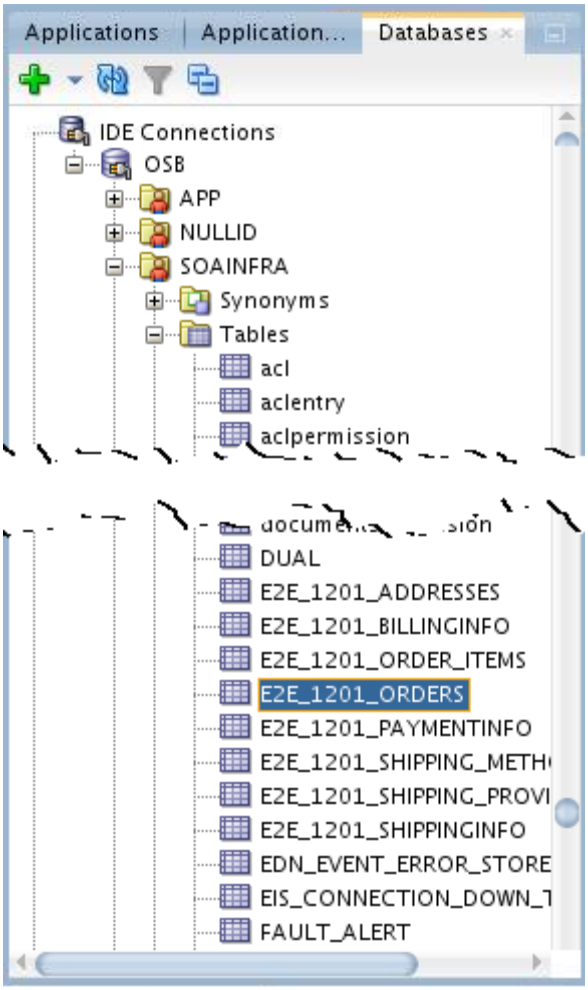


12. Save your project.

Deploy and Test

13. To test your Service Bus application, do as follows:
    a. In the project Overview Editor, right-click the proxy service, and select Run.
    b. In the Test console, select **~/labs_DI/resources/sample_input/OrderSample.xml**
    c. Click Execute button.
       You should see the order number in the response message.
    d. Verify the status update in the database.
       1) From the JDeveloper main menu, select Window > Database >Databases. The Databases window is displayed.

2) Expand IDE Connections folder, navigate to OSB > SOAINFRA > Tables, and locate the E2E_1201_ORDERS table in the list.



3) Right-click the table, and select **Open Object Viewer** from the context menu. You should see the latest order entry in the database with a status of SendtoFulfillment.

201561119521010	(null)	2015-06-11	105.75999999999999	daniel@localhost	SendtoFulfillment

You can use the BadOrderSample.xml file to test an invalid credit card scenario, and view the result.

14. When you are done, close the application and all open windows of the projects.

Practices for Lesson 9: Processing Messages with Concurrent Calls

Chapter 9

Practices for Lesson 9: Overview

Practices Overview

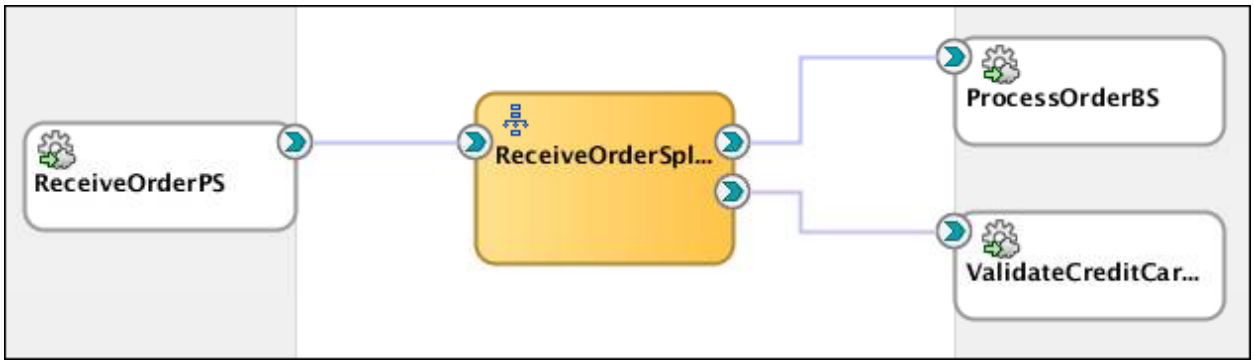
When the order management system receives a customer order, it calls the order processing service to create an order number and save the order in the database with the status New. At the same time, it will invoke the ValidateCreditCard service to validate credit card information before the order is processed.

Practice 9-1: Processing Messages with Static Split-Join

Overview

In this practice, you use split-join to implement the above scenario; in the split-join you add two branches which execute in parallel, one to save the order in the database and the other to validate the credit card.

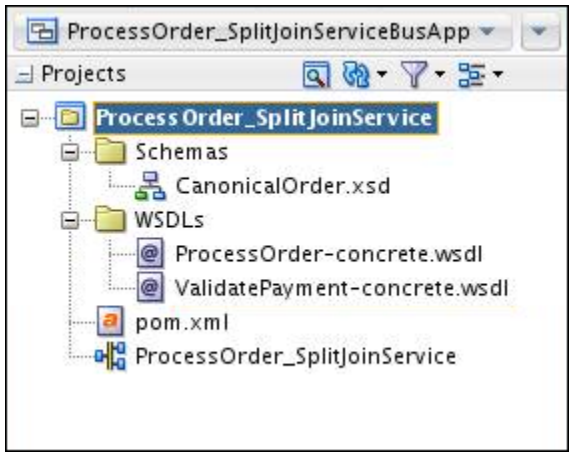
When completed, the project should look like this:



Tasks

Setups

1. Open Service Bus project.
   a. In JDeveloper, select **File** > **Open** from the menu bar.
   b. In the Open Application(s) dialog box, navigate to the `$HOME/labs_DI/lessons/lesson09/ProcessOrder_SplitJoinServiceBusApp/` directory, and open the `ProcessOrder_SplitJoinServiceBusApp.jws` file.
   c. Expand the folders in the Application Navigator, and you see that the folder structure resembles the image below:



To save your time, the application and project folder structure is pre-created and the source WSDL and Schema files are imported into the project for you.

2. Create the business services

   You will create two business services, one to invoke the ProcessOrder SOA service, and one for the ValidateCreditCard SOA service.

   a. In Application navigator, double-click ProcessOrder_SplitJoinService to open it in Service Bus Overview editor.

   b. Drag the HTTP component from the Technology section in the Component palette and drop it onto the External Services lane.

   c. Enter the required details for business by following the instructions in the table below:

Step	Window/Page Description	Choices or Values
a)	Create Service	Name the service `ValidateCreditCardBS`. Click Next.
b)	Type	1) Select **WSDL** for Service Type. 2) Click the Browse WSDLs icon placed to the right of the WSDL choice. 3) In the Select WSDL dialog box, click **Application**, navigate and select **ValidatePayment-concrete.wsdl.** 4) Click **OK**. 5) Back in the Create Business Service window, you should see the WSDL and port fields populated with the information based on the WSDL you chose. 6) Click **Next**.
c)	Transport	Verify that http is selected in the Transport field. Change the Endpoint URI to point to the ValidateCreditCard composite: `http://localhost:7101/soa-infra/services/default/ValidateCreditCard/validateCCard_client_ep` Click **Finish**.

   d. Repeat the above mentioned steps to create another business service for ProcessOrder SOA service. Two things you need to do differently include:

   - Name the service `ProcessOrderBS`**.**
   - Select **ProcessOrder-concrete.wsdl** for WSDL file**.**
   - The Endpoint URI should point to the ProcessOrder composite, which is:
     `http://localhost:7101/soa-infra/services/default/ProcessOrder/receiveorder_client_ep`

   Now you should see two business services created in the External Services lane in Overview Editor.

Add Split-Join

3. To create a Split-Join and proxy service:

   a. In the project Overview Editor, drag Split-Join into the middle swim lane from the Resources section in the Components palette.

   b. In step one (Create Service), name the Split-Join service **ReceiveOrderSplitJoin.**

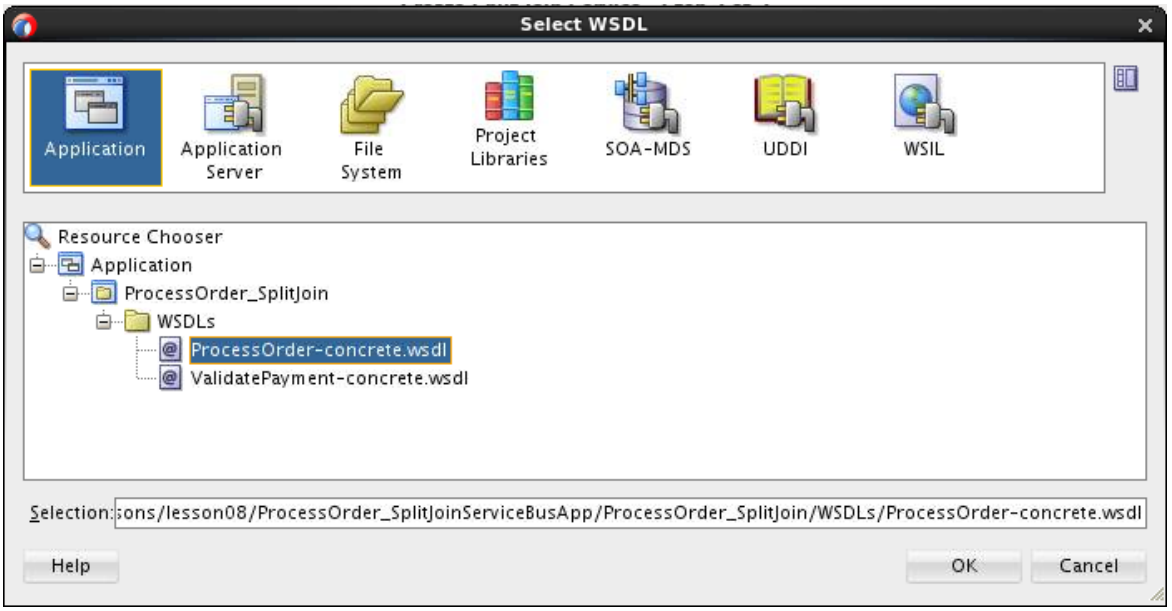
c. Click **Next**.

d. In step two (Type), use the WSDL of the ProcessOrder business service for the Split-Join service:

1) Click the **Browse WSDLs** icon to the right of the WSDL field.

The Select WSDL dialog box pops up.

2) Locate **ProcessOrder-concrete.wsdl** under Application.



3) Click **OK**.

Back in the wizard, you should see the fields populated with the wsdl file.

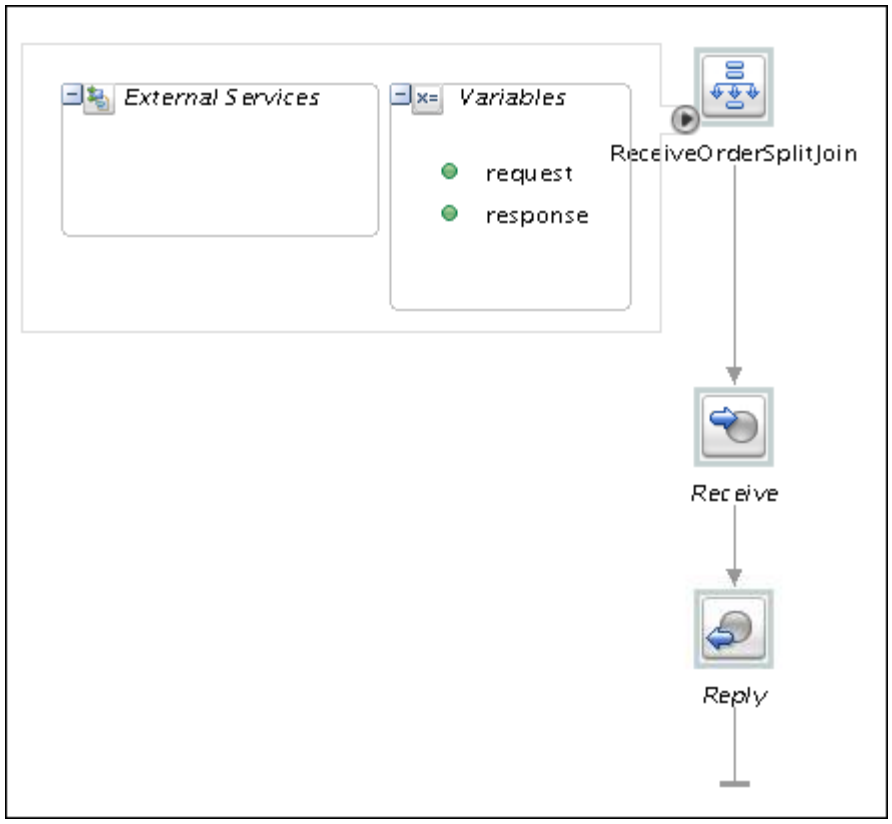
4) Keep "Expose as a Proxy Service" checked for creating Proxy Service along with the pipeline.

5) Change the Proxy Service name to **ReceiveOrderPS**. Leave the rest of the values to their defaults.

e. Click **Finish**.

You see that both Split-Join and proxy service are added in Overview Editor.

4. Configure Split-Join
   a. Double-click ReceiveOrderSplitJoin to open it in the editor.



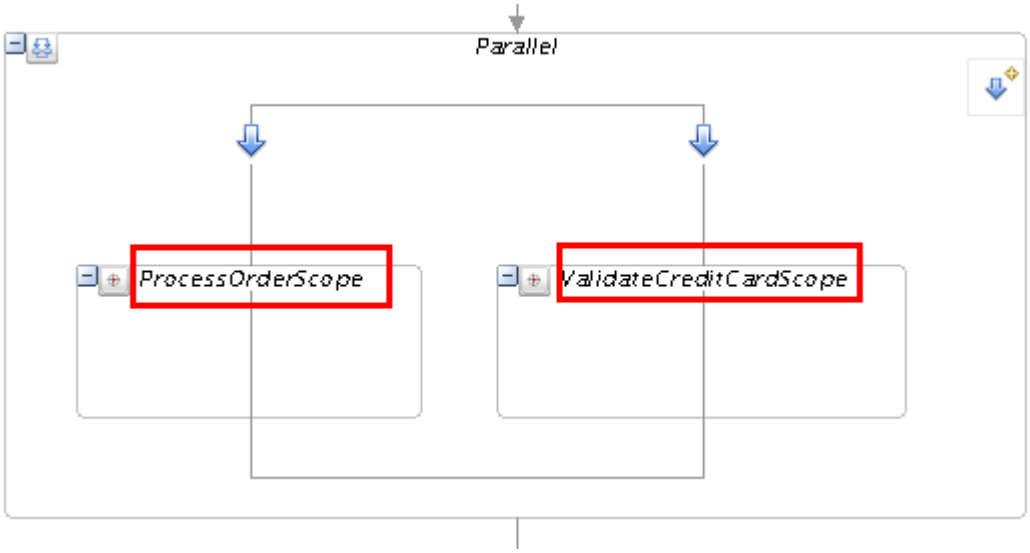
   You see that the primary elements constructing a message flow in a split-join:
   - A start node, which contains the request and response variables introspected from the WSDL operation
   - A receive node, to receive incoming request messages
   - A reply node, to send response messages

   Note that request and response are global variables.

   b. Add Parallel flow control activity after Receive activity in split-join. The Parallel activity allows you to invoke different services concurrently.

   To invoke two services from this split-join, you need only two branches in parallel flow.

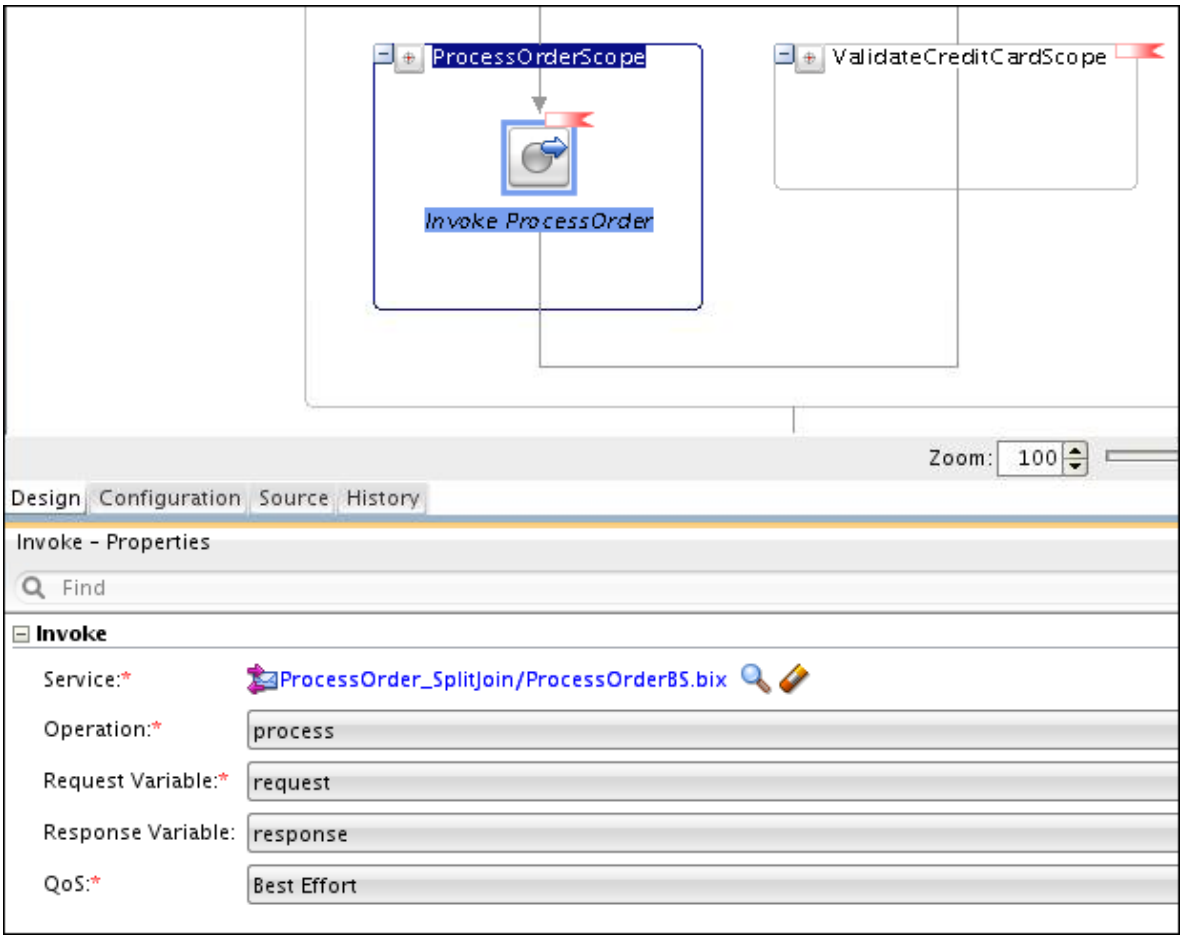
c. Modify Name for both Scope activities(right-click) as shown below:



d. Add an Invoke Service activity in the ProcessOrder branch. Invoke activity is used to call business service.

e. Configure the Invoke Service activity to invoke ProcessOrder business service.
   1) Modify the name of the invoke activity to **InvokeProcessOrder**.
   2) In Invoke Properties inspector, select ProcessOrderBS as the target service.

3) Keep the rest of the fields unchanged.



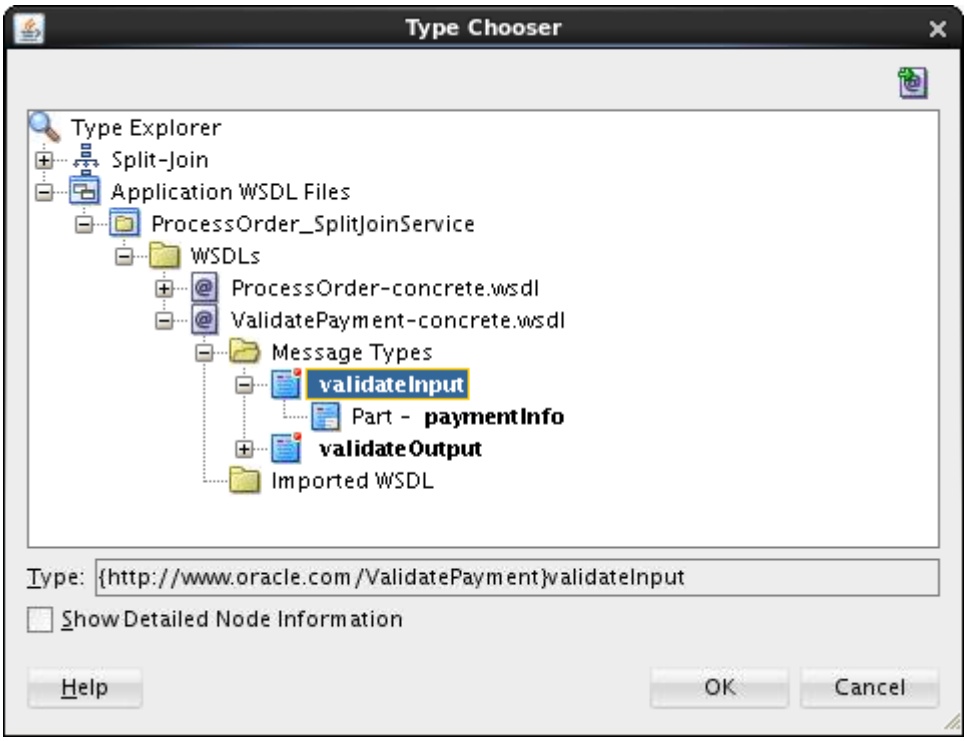
Create variables

You create two variables, one local variable in ValidateCreditCardScope and one global variable:

- The local variable will store the content of the Billing element from the order, which will be used as the request for ValidateCreditCard service.
- The global variable will store the response from the ValidateCreditCard service.

5. To create the local variable:
   a. Right-click the ValidateCreditCardScope, and then select **Create Variable**.
      The Create Variable Alias dialog appears.
   b. Enter `CreditCard` as the name for the variable.
   c. Select the WSDL Message as variable type.
   d. Click Browse next to the variable type.
      The Type chooser dialog appears.

e. Navigate, and select **validateInput**. Use the following image as a guide:



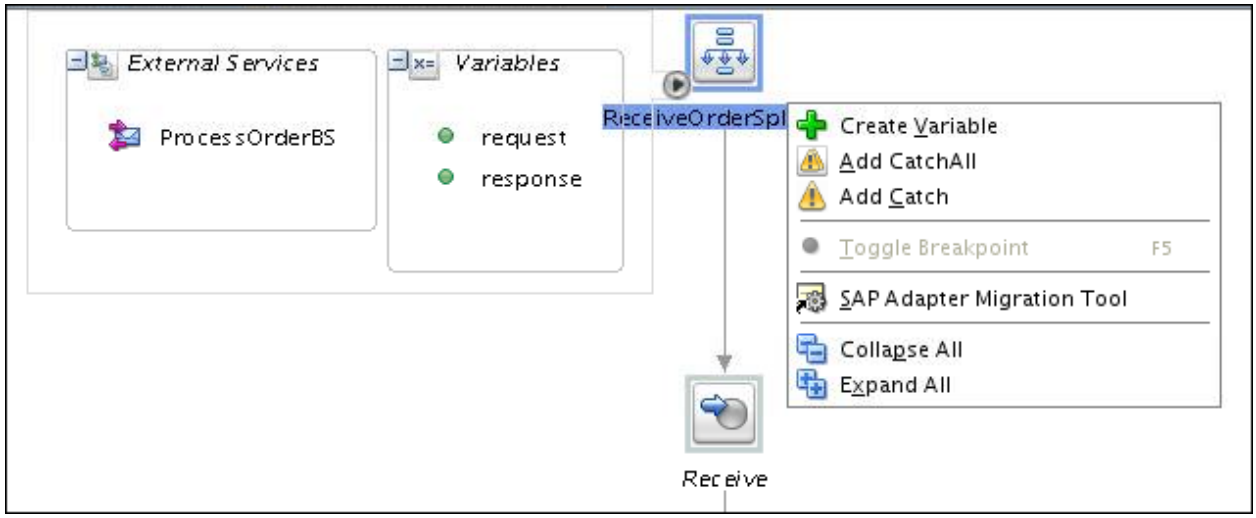
f. Click OK to close Type Chooser.

g. Click OK to close Create Variable.

The new variable appears in the list of variables in the Variables box.

6. To create the global variable:

a. Right-click the Start Node (ReceiveOrderSplitJoin), and then select **Create Variable**.



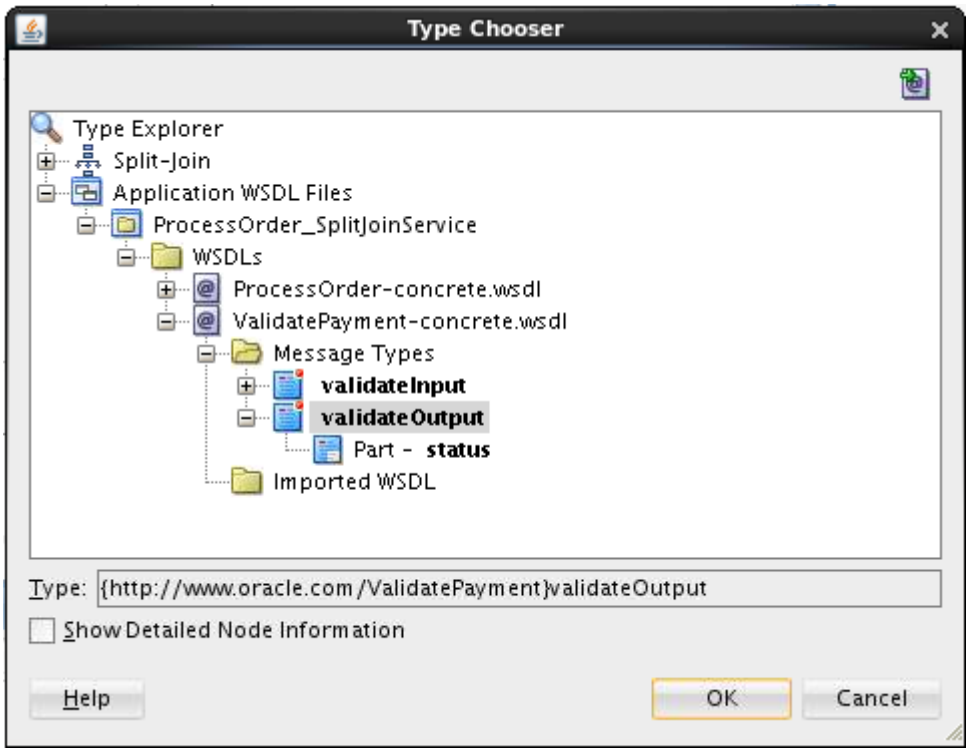
b. Enter `ValidationStatus` as the name for the variable.

c. Select the WSDL Message as variable type.

d. Click Browse next to the variable type.

The Type chooser dialog box appears.

e. Navigate and select **validateOutput.**



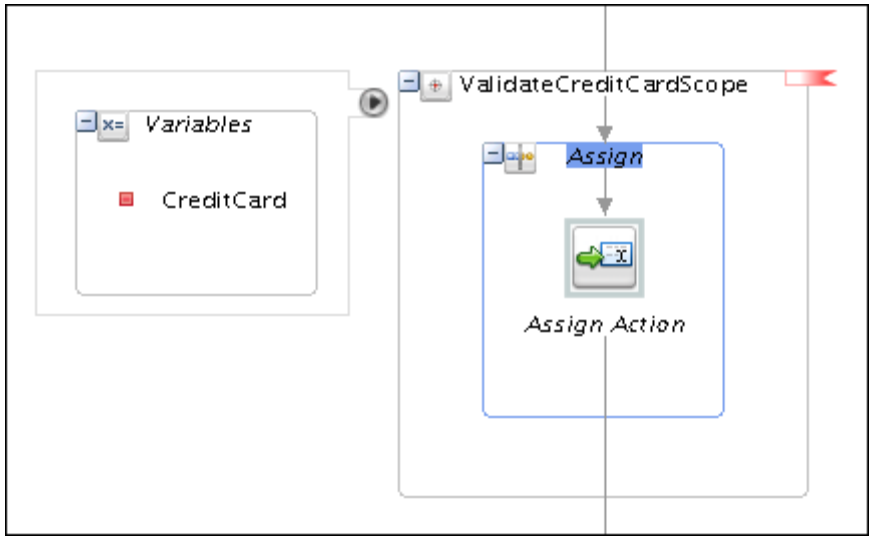
f. Click OK to close Type Chooser.

g. Click OK to close Create Variable.

The new variable appears in the list of variables in the Variables box.

7. To assign Billing content to the local variable:

a. Add an Assign activity in the ValidateCreditCardScope.



b. Click Assign Action.

c. Modify the name to **Assign CreditCard**.

d. In Assign Properties inspector, click XQuery Expression Builder icon.

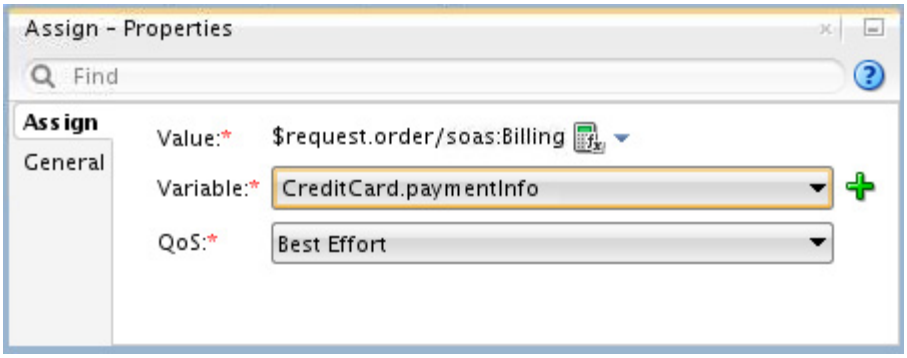
The XQuery Expression Builder window opens.

e. Locate the Billing element in $request, and insert it into the Expression field.

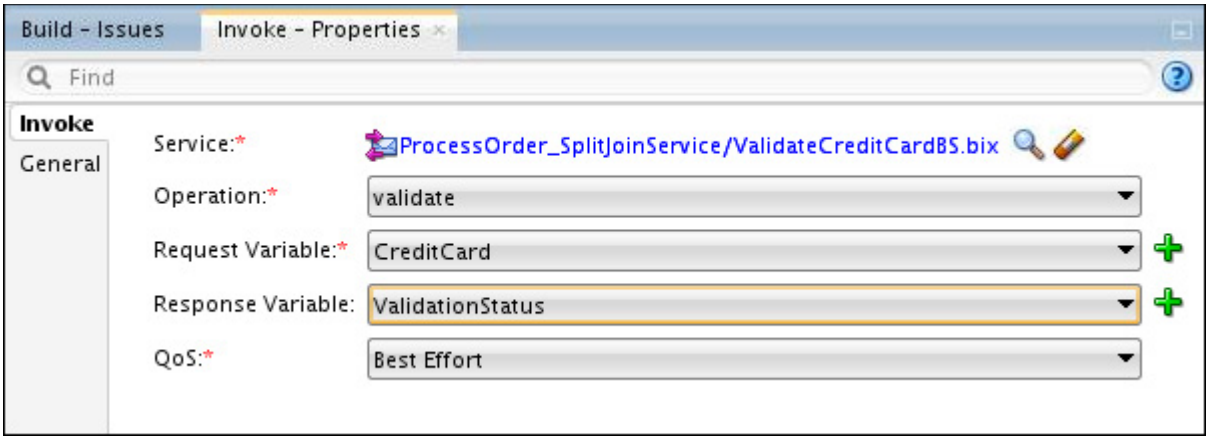
f. Click OK.

Back in the Assign Properties inspector, you see that the Value field is populated with value you defined.

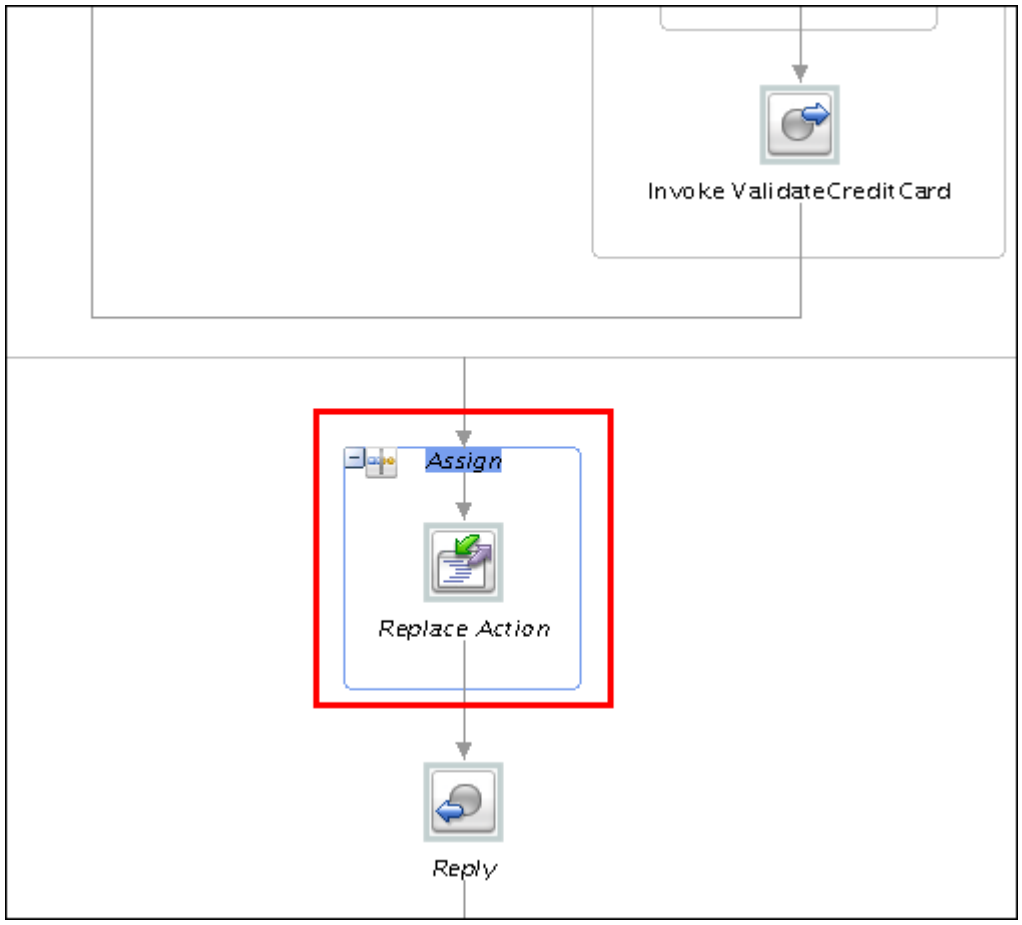
g. In the Variable field, select **CreditCard.paymentinfo** from the drop-down list. Now your Assign Properties inspector should look like the image below:



8. Add another Invoke activity in the ValidateCreditCard branch to invoke the ValidateCreditCard business service.
   a. Drag-and-drop Invoke Service after Assign CreditCard activity.
   b. Modify the name of the invoke activity to **InvokeValidateCreditCard**.
   c. In Invoke Properties inspector,
      1) Select **ValidateCreditCardBS** as the target service.
      2) Select **CreditCard** as Request Variable.
      3) Select **ValidationStatus** for Response Variable.

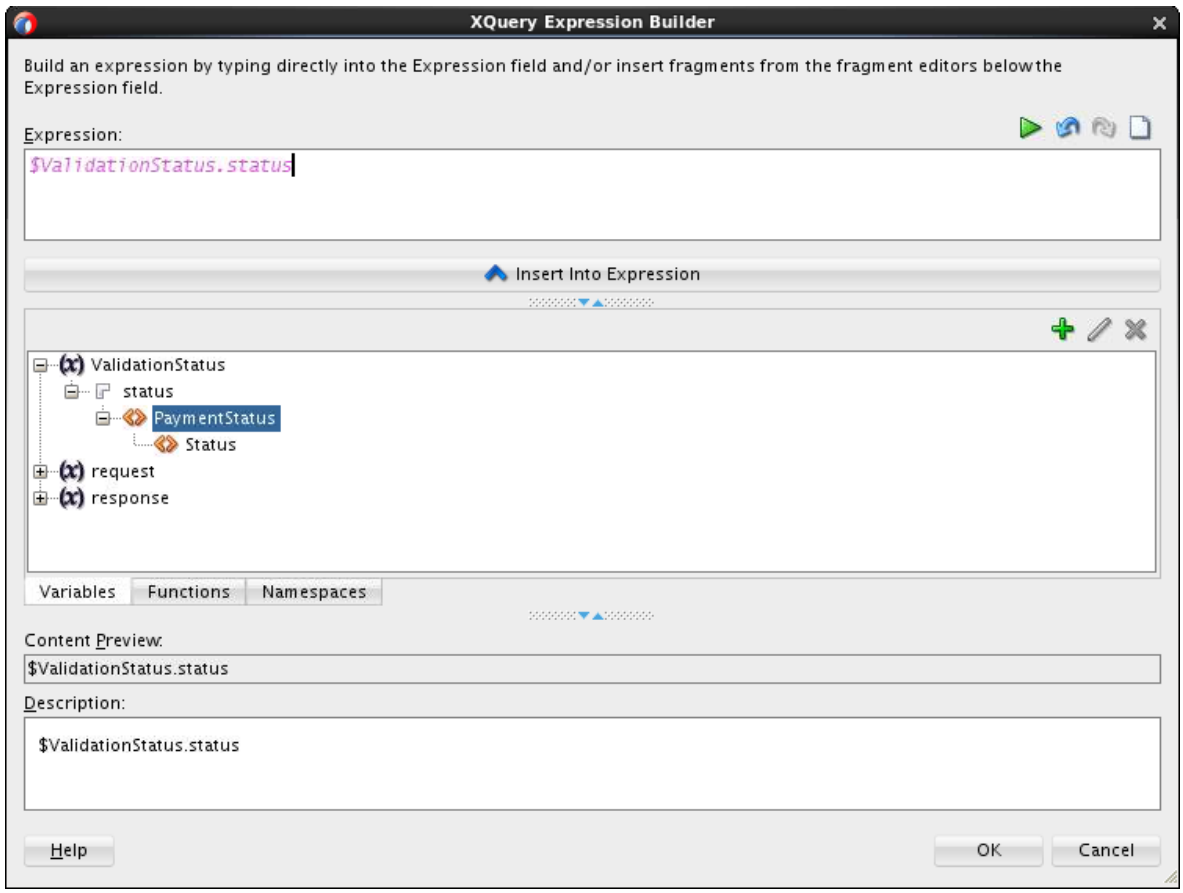


9. Add Replace action to replace the order Number in the response message with the credit card validation status.
   a. Drag-and-drop Replace action before Reply activity.



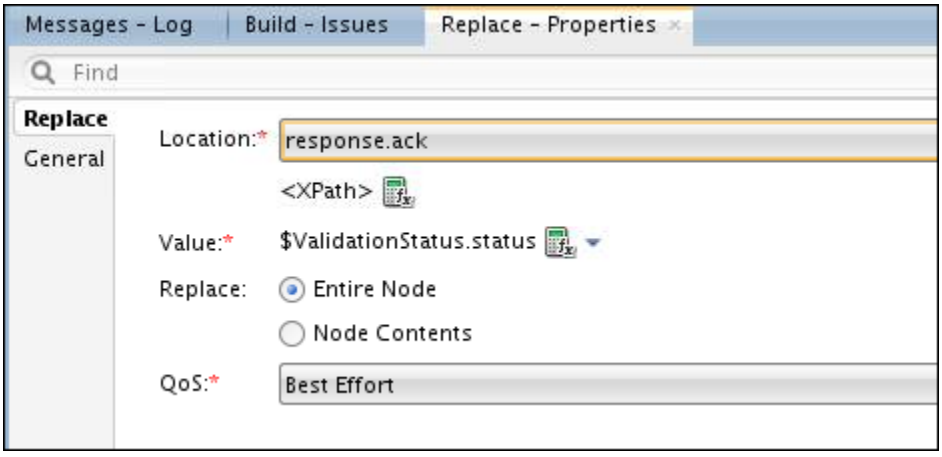
   b. Modify the name of Replace to **Replace Response**.
   c. In the Replace Properties inspector, do as follows:
      1) Select **respone.ack** as Location.
      2) Click XQuery Expression Builder icon next to the Value field.

3) Select ValidationStatus > status > PaymentStatus element, and insert it into the Expression field.

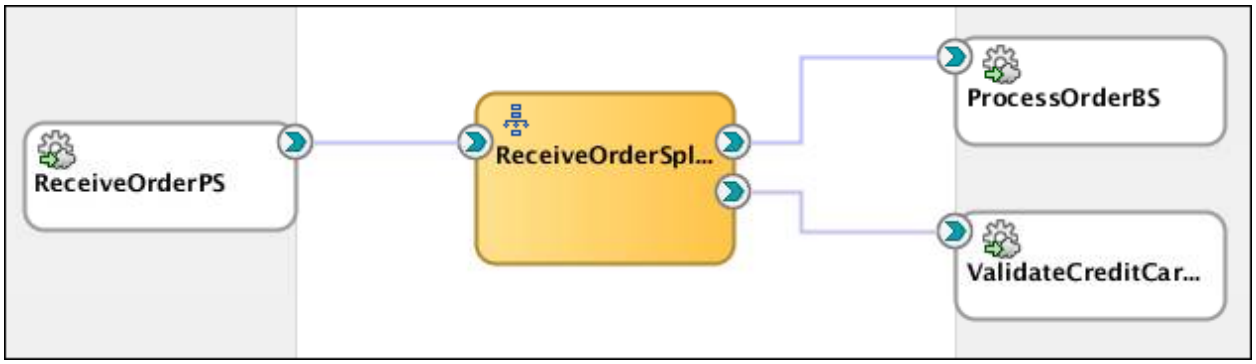


4) Click OK.

5) Keep the rest of the field values unchanged.



Now your project Overview Editor should resemble the following image:



10. Save your project.

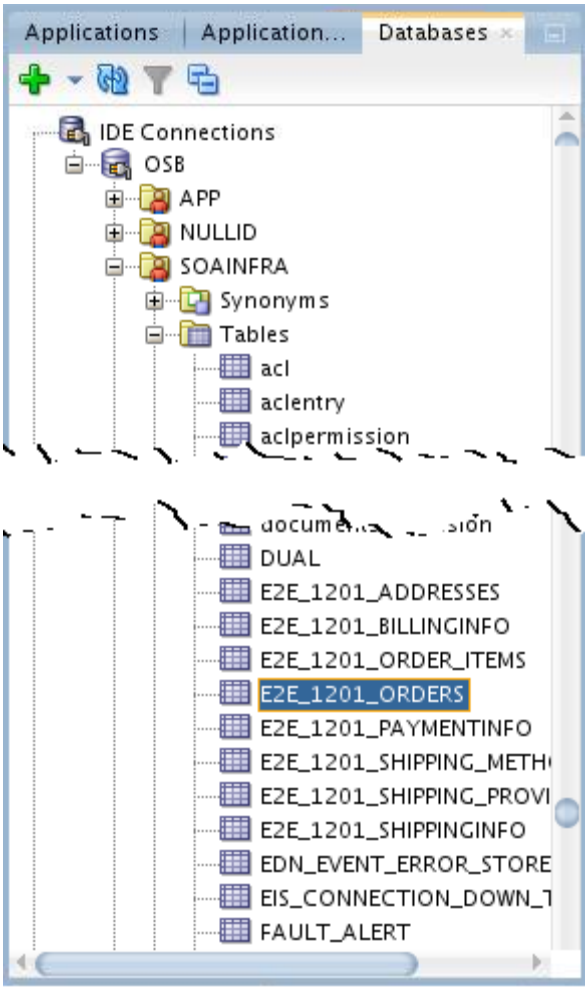
Deploy and Test

11. To test your Service Bus application, do as follows:
    a. In the project Overview Editor, right-click the proxy service, and select Run.
    b. In the Test console, select **~/labs_DI/resources/sample_input/OrderSample.xml**
    c. Click the Execute button.
    You see that the status is “Valid” in the response message.

    This is the return from the ValidateCreditCard branch.

12. Verify the outcome of the test from the ProcessOrder branch, which is adding a new order in the database.
    a. From the JDeveloper main menu, select Window > Database >Databases. The Databases window is displayed.

b. Expand IDE Connections folder, navigate to OSB > SOAINFRA > Tables, and locate the E2E_1201_ORDERS table in the list.



c. Right-click the table, and select **Open Object Viewer** from the context menu. You should see the new order entry added in the database.

13. When you are done, close the application and all open windows of the project.

Practices for Lesson 10: Adapters and Transports

Chapter 10

Practices for Lesson 10: Overview

Practices Overview

In these practices, you learn how to:

- Invoke a synchronous SOA service from Oracle Service Bus using SOA-Direct protocol
- Provide a REST interface to SOAP service, so it handles requests from mobile devices as well as future systems that use REST/XML and REST/JSON

Practice 10-1: Calling a SOA Service using SOA-Direct Protocol

Overview

In this practice, you deploy the ValidatePayment composite application, which adds a synchronous Direct Binding service interface. Then, you create the service bus application to invoke the direct binding interface through the SOA-DIRECT transport.

Tasks

Deploy a composite application with direct binding

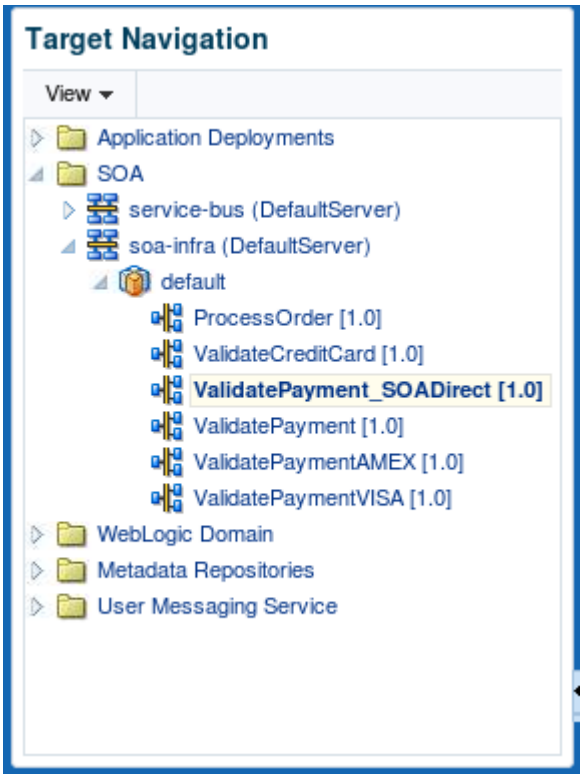
To use the SOA-Direct protocol in OSB, first you need to have an SOA service that provides direct binding.

1. In a terminal window, enter the following commands:

```
>cd ~/labs_DI/lessons/lesson10
>ls -l
>$HOME/labs_DI/resources/Scripts/deploy.sh
sca_ValidatePaymentSOADirect_rev1.0.jar
```

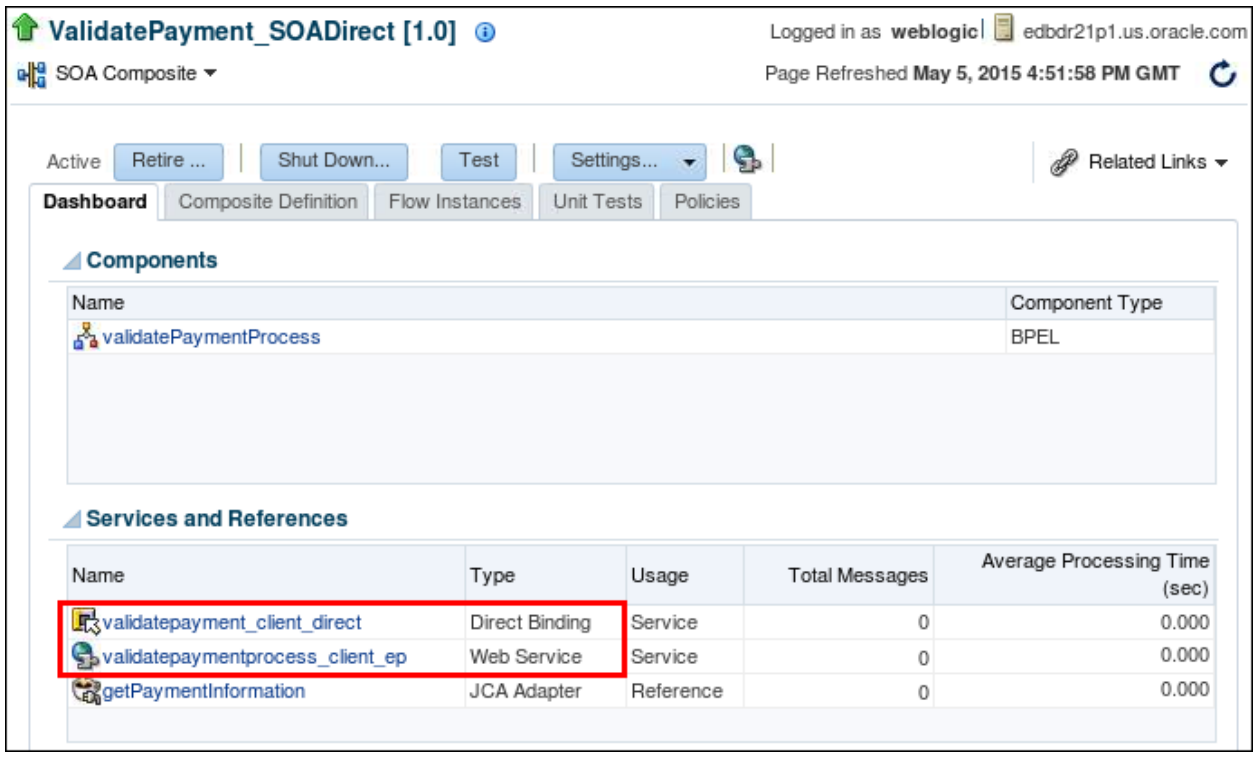
   The build and deployment information will be displayed in the console. You should see a message showing both build and deploying executed successfully.

2. Verify and examine the deployed service in EM.
   a. Open the EM console. In the Target Navigation panel, expand SOA > soa-infra > default. Now you should see the deployed composite application ValidatePayment_SOADirect[1.0] in the folder.



   b. Select the composite.

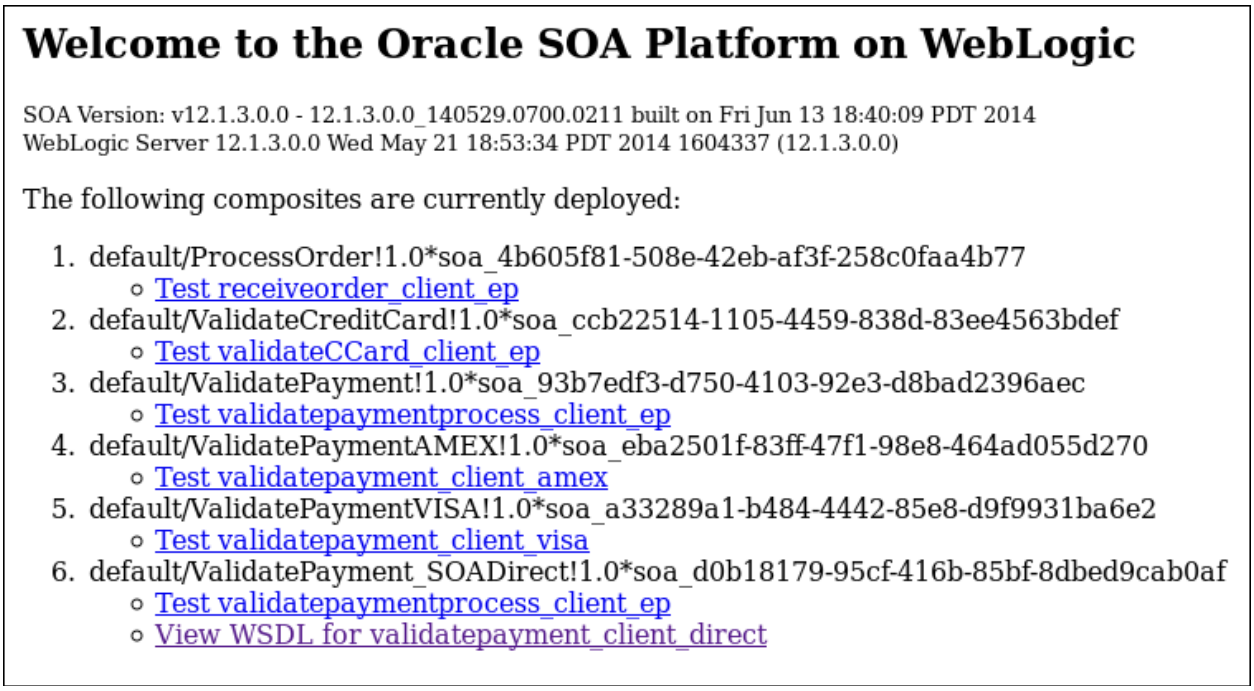
The home page (Dashboard tab page) for the selected composite application is displayed to the right pane.



Note that the composite provides two interfaces, one is Direct Binding, and the other is the SOAP web service.

3. You cannot access direct binding from the EM console, so do as follows to get the SOA-DIRECT Endpoint URI:
   a. Open a new browser tab, and entr the URL below:
      `http://localhost:7101/soa-infra`

b. Log in with WebLogic Server administrator's username and password.



c. Click View WSDL for the validatepayment_client_direct link.

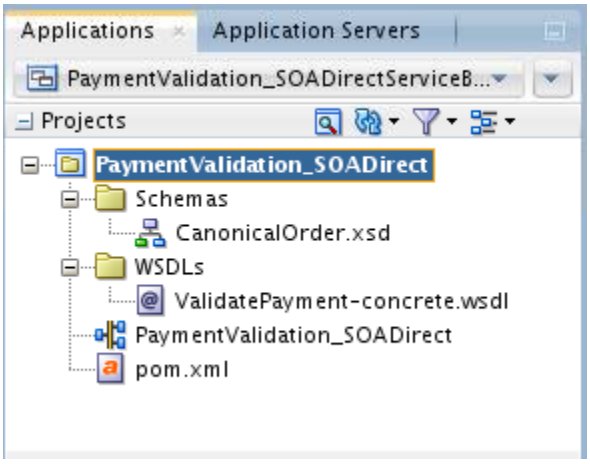
d. Note down or copy the service endpoint location which you will use in later stage.

4. Open the Service Bus project

a. In JDeveloper, select **File** > **Open** from the main menu.

b. In the Open Application(s) dialog box, navigate to the `$HOME/labs_DI/lessons/lesson10/PaymentValidation_SOADirectServiceBusApp/` directory, and open the `PaymentValidation_SOADirectServiceBusApp.jws` file.

c. Expand the folders in Application Navigator, and you should see the folder structure resembling the image below:



5. Create a business service.

   This business service will invoke the ValidatePayment_SOADirect SOA service using the SOA-direct protocol.

   a. Double-click the project overview file, `PaymentValidation_SOADirect`, in Application Navigator.

      The Services Bus Overview Editor opens on the right.

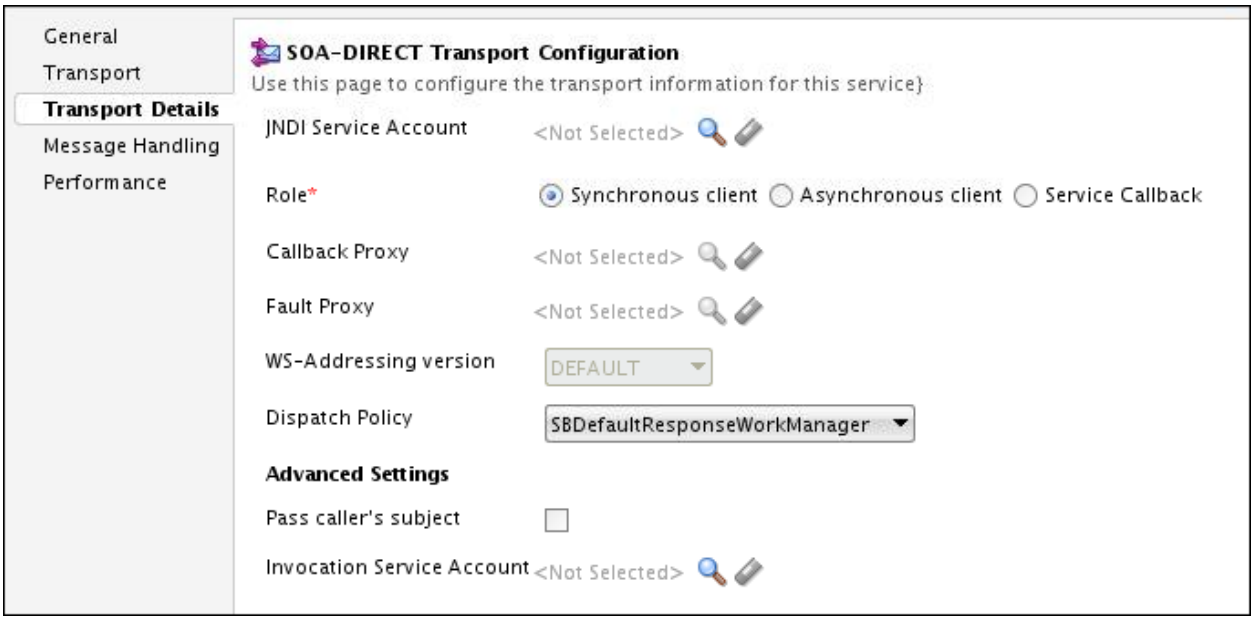
   b. In the Overview Editor, drag the Direct component (in the Technology section) from the Component palette and drop it onto the External Services lane.

      Upon dropping, the Create business service wizard pops up.

   c. Enter the required details for the business service by following the instructions given in the table below:

Step	Window/ Page Description	Choices or Values
1)	Create Service	Name the service `PaymentValidation_SOADirectBS`. Click Next.
2)	Type	a) Click the Browse WSDLs icon to the right of the WSDL field. b) In the Select WSDL dialog box, click **Application**, navigate and select **ValidatePayment-concrete.wsdl**. c) Click **OK**. d) Back in the Create Business Service window, you should see the WSDL and port fields populated with the information based on the WSDL that you chose. e) Click **Next**.
3)	Transport	Verify that soa-direct is selected in the Transport field. Set the Endpoint URI to the one you copied earlier. It should resemble the following: `t3://<yourhostname>:7101/default/ValidatePayment_SOADirect!1.0/validatepayment_client_direct` Click **Finish**.

d. Right-click the business service and select **Edit** in the Overview Editor.
   The business service editor opens in a new tab window.
e. Click the **Transport Details** tab, and review the configuration.



Note that the business service is set with synchronous client role in a synchronous invocation of an SOA composite.

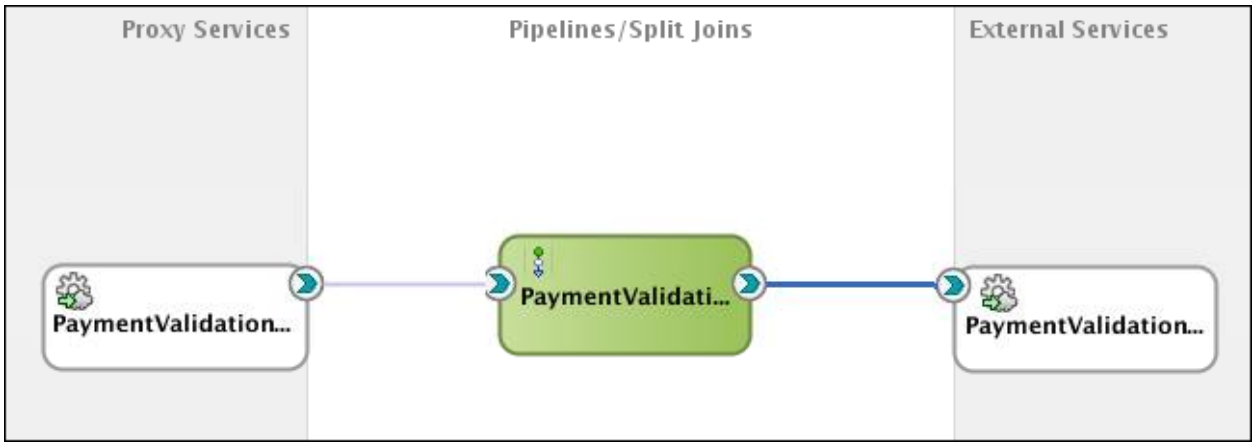
Create a proxy service to invoke the business service

6. Create Proxy Service.
   a. In Overview Editor, drag and drop the HTTP component from the Components palette to the Proxy Services lane.
      The Create Proxy Service wizard pops up.
   b. In step one (Create Service), fill up the details for Proxy Service:
      - Service Name: **PaymentValidation_SOADirectPS**
      - Keep “Generate Pipeline” checked to create pipeline along with Proxy Service.
      - Change the pipeline name to **PaymentValidation_SOADirectPP.**

      Click **Next**.
   c. In step two (Type), use the same WSDL of the business service for the proxy service:
      - Select service type **WSDL.**
      - Click the **Browse WSDLs** icon on the right.
        The Select WSDL dialog box pops up.
      - Navigate to Application > PaymentValidation_SOADirect > WSDLs directory, and select ValidatePayment-concrete.wsdl.
      - Click **OK**.

      Back in the wizard, you should see the fields populated with the wsdl file and binding information.
      Click **Next**.
   d. In step three (Transport), you should see the populated endpoint URI depending on your configuration in previous steps. Keep it unchanged and click **Finish**.

7. Wire the pipeline to invoke the PaymentValidation_SOADirectBS business service.



8. Save your work.

Deploy and Test

9. To run the test, do as follows:
   a. Right-click the PaymentValidation_SOADirectPS in the Service Bus Overview Editor. Select **Run**.

      The Test Console will activate.
   b. Click the **Choose File** button.
   c. Navigate to `/home/oracle/labs_DI/resources/sample_input/`, and select **`PaymentInfoSample_Authorized.xml`**
   d. Click **Open**.
   e. On the Test Console, click the **Execute** button.

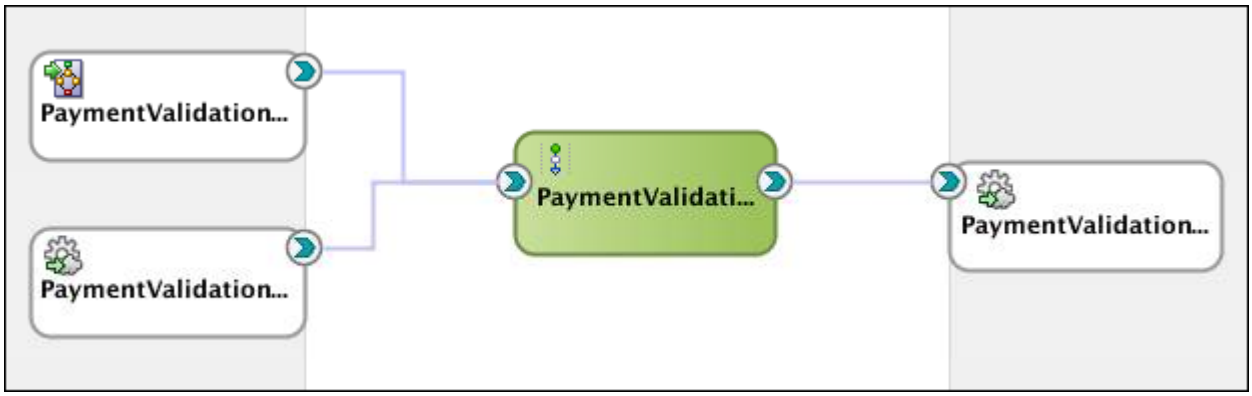
      You should see the response message with the status “Authorized” right away.
10. Close the application and all open windows of the project.

Practice 10-2: Exposing a SOAP Service as a REST Service

Overview

In this practice, you add a REST interface to the existing PaymentValidation service bus application, so it can support authorizations from mobile devices as well as future systems that use REST/XML and REST/JSON.

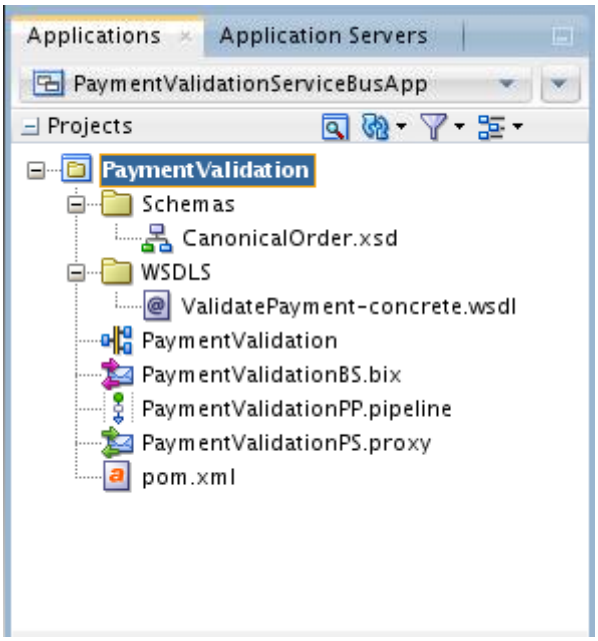
Once done, your project should resemble the following:



Assumptions

You completed Practice 2 - Virtualizing Service with Service Bus of Lesson 3.

1. Open the ValidatePayment Service Bus project
   a. In JDeveloper, select **File** > **Open** from the main menu.
   b. In the Open Application(s) dialog box, navigate to the `$HOME/labs_DI/lessons/lesson03/PaymentValidationServiceBusApp/` directory, and open the `PaymentValidationServiceBusApp.jws` file.
   c. Expand the folders in Application Navigator, and you should see the folder structure resembling the image below:

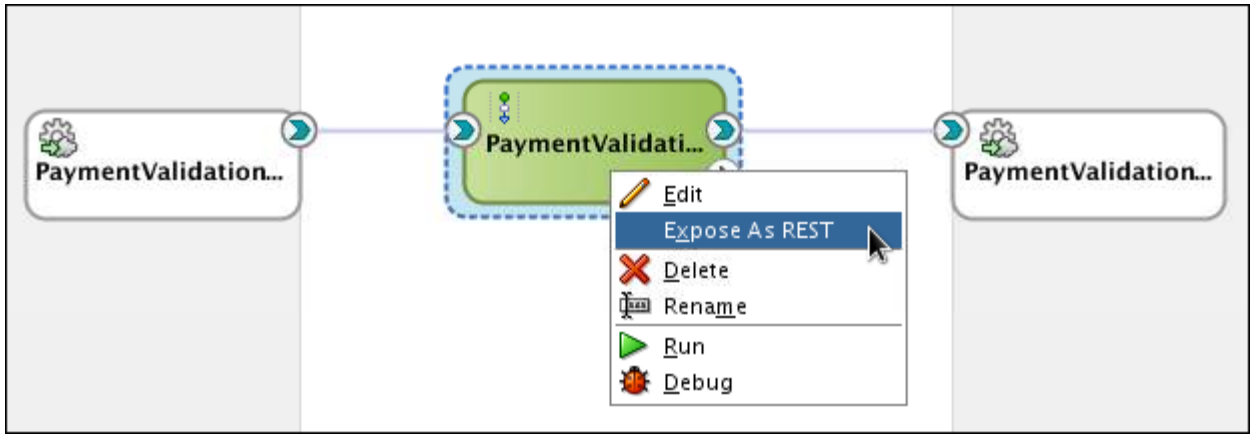


   d. Open PaymentValidation Overview.

2. Expose the Pipeline as REST

   There are multiple ways to create a REST binding, on either the proxy service side or the external services/business services side. In this case, you will choose to expose the PaymentValidationPP pipeline.

   a. Right-click the pipeline and select **Expose as REST**.



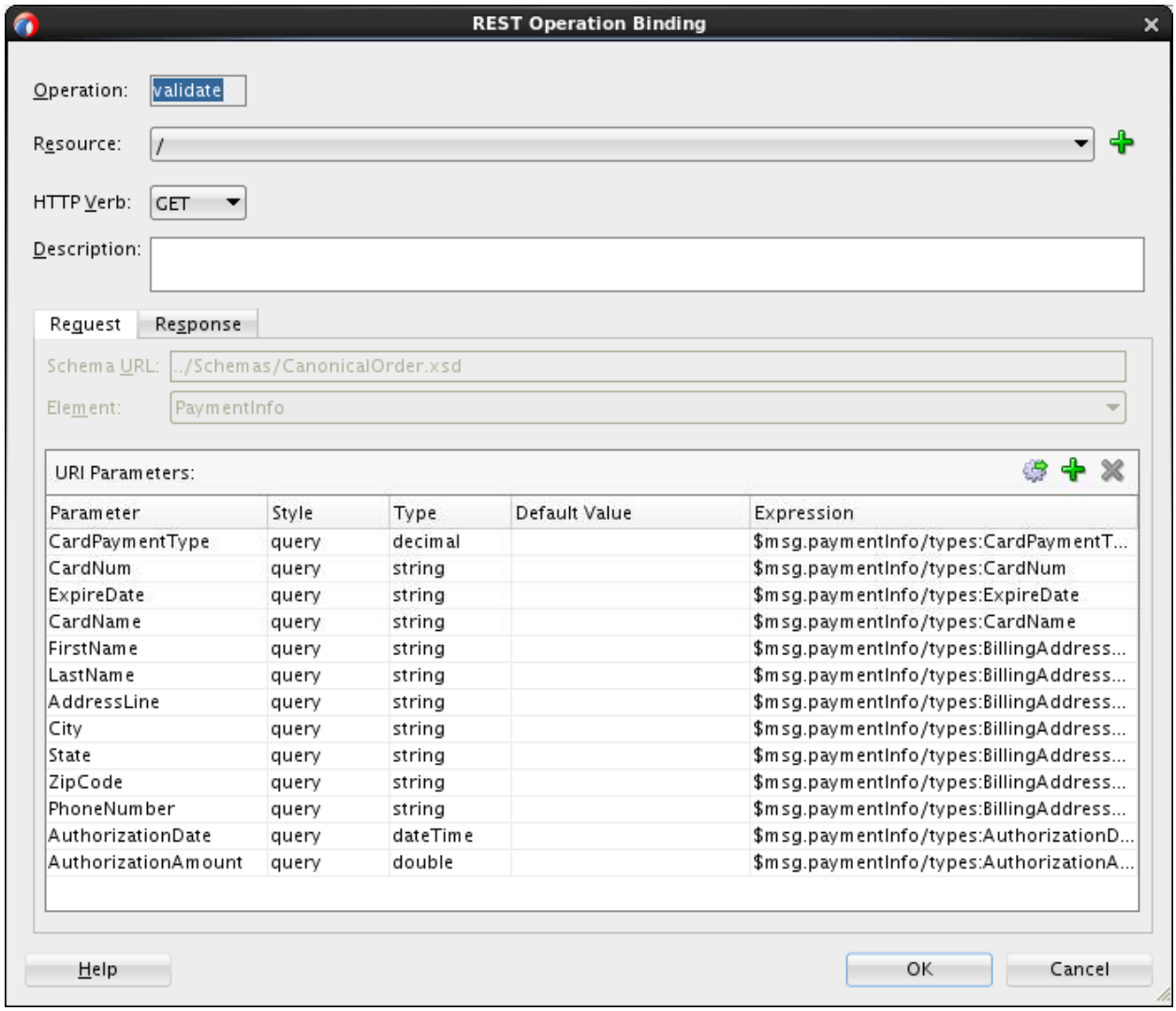
   The Create REST binding wizard appears.

   b. Name the service **`PaymentValidationRS`**.

   c. Add a description, for example “RestService to ValidatePayment SOA service.”

   d. Select the validate operation and click the edit  icon.

e. The REST Operation Binding window is displayed.



The pipeline you expose is WSDL based and has an underlying “CanonicalOrder” schema as its structure. Your binding automatically takes this shape.

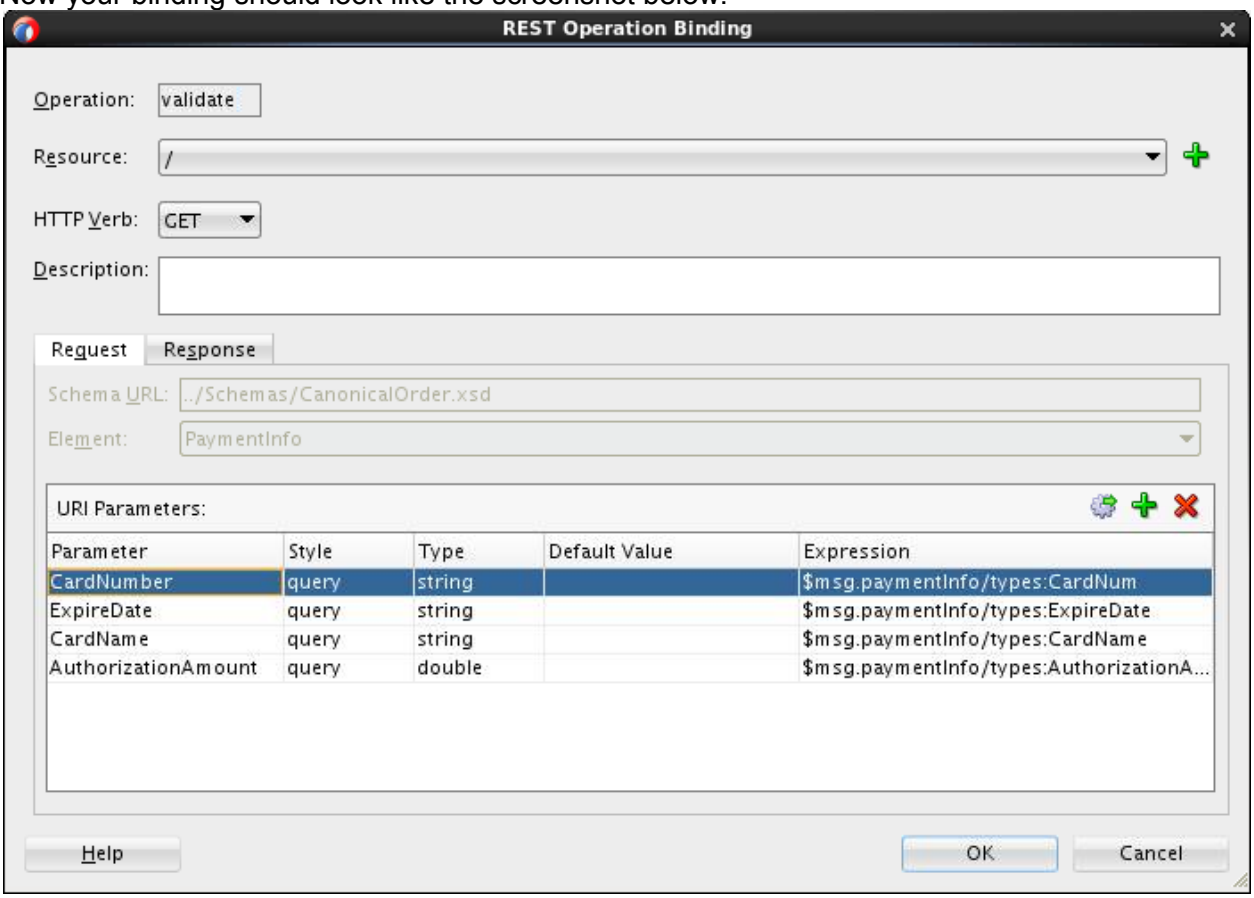
f. Click the HTTP Verb drop-down list. Note that you can create bindings for all of the standard verbs. Retain GET HTTP verb for this lab.

g. Remove some of the auto-created parameters for the purposes of this lab. You will not use all of them.

h. Remove all parameters besides CardNum, CardName, ExpireDate, and AuthorizationAmount. (You can hold down the CTRL key and click the items in the list to choose them)

i. Double-click the CardNum and rename to **`CardNumber`**.

Now your binding should look like the screenshot below:



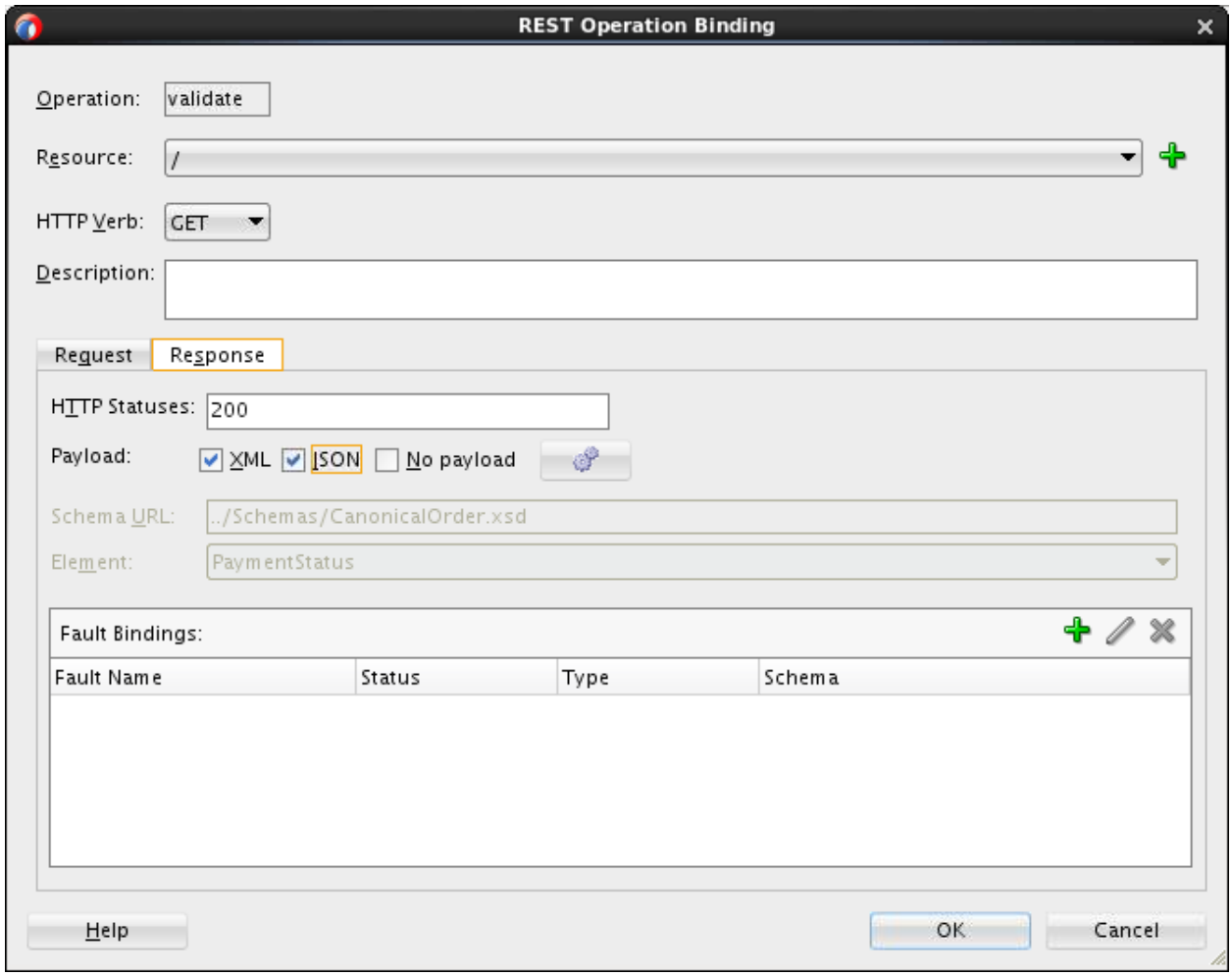
Even with the renamed CardNumber parameter, it still maps to the same underlying expression.

j. Click the **Response** tab.

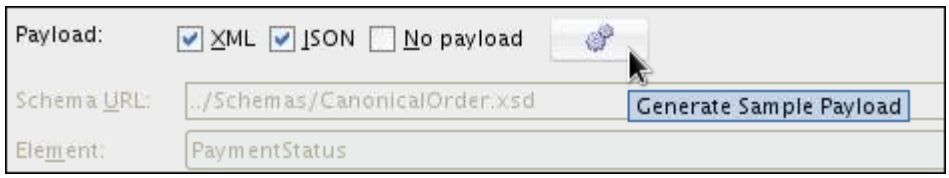
The response is also based on the same underlying schema.

k. Retain the status code of 200 for HTTP status code.

l. Check both XML and JSON so that your new interface will support both data formats.

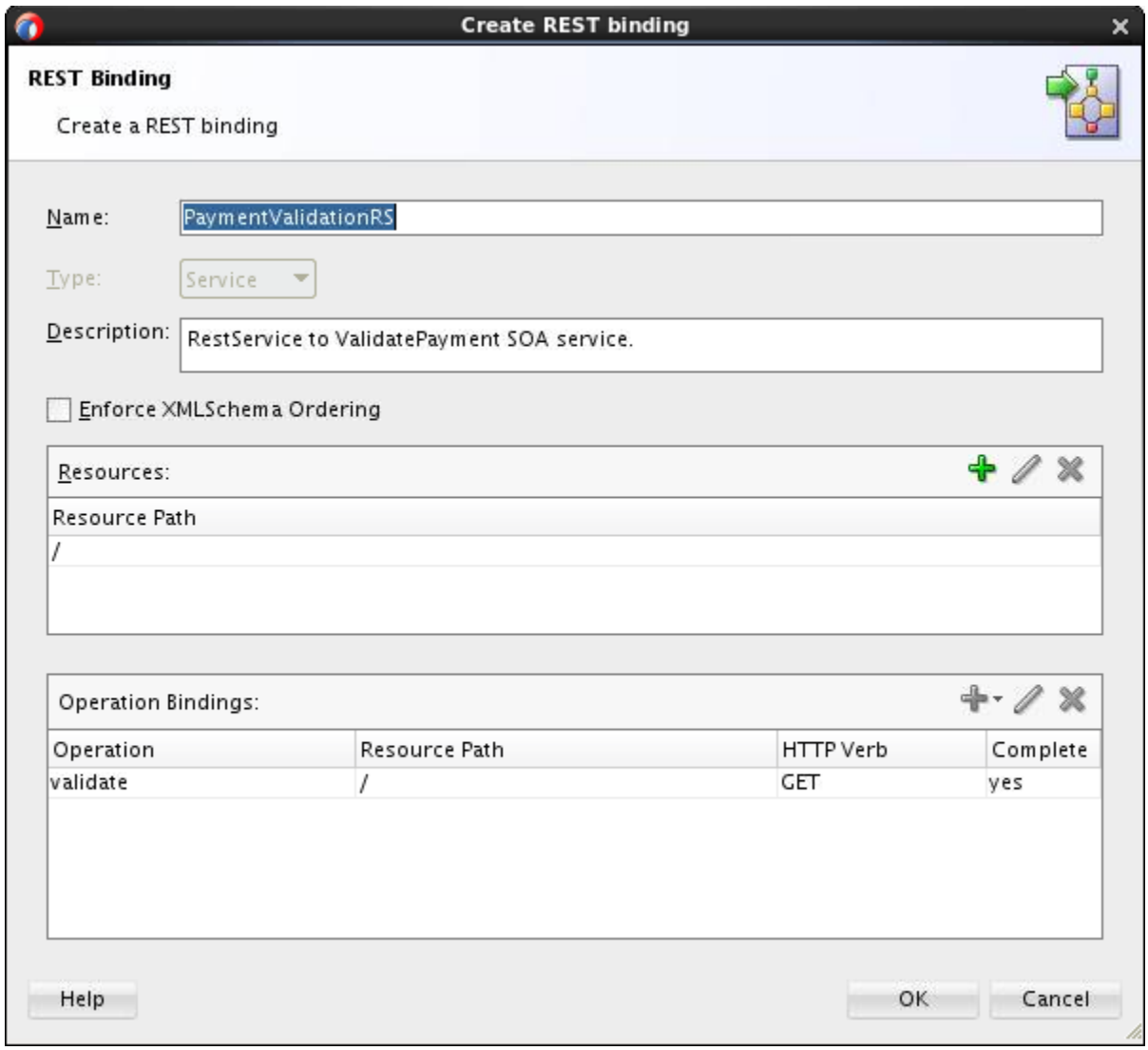


m. You can click Generate Sample Payload to view a sample of the selected response payload.



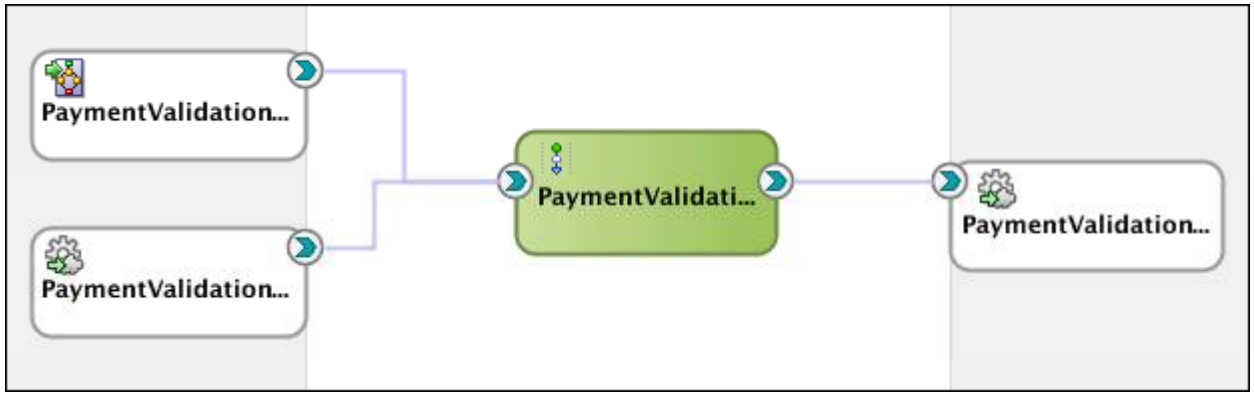
Once you are satisfied with your Operation binding, click **OK.**

With the Operation Binding defined, your REST Binding should look like the screenshot below:

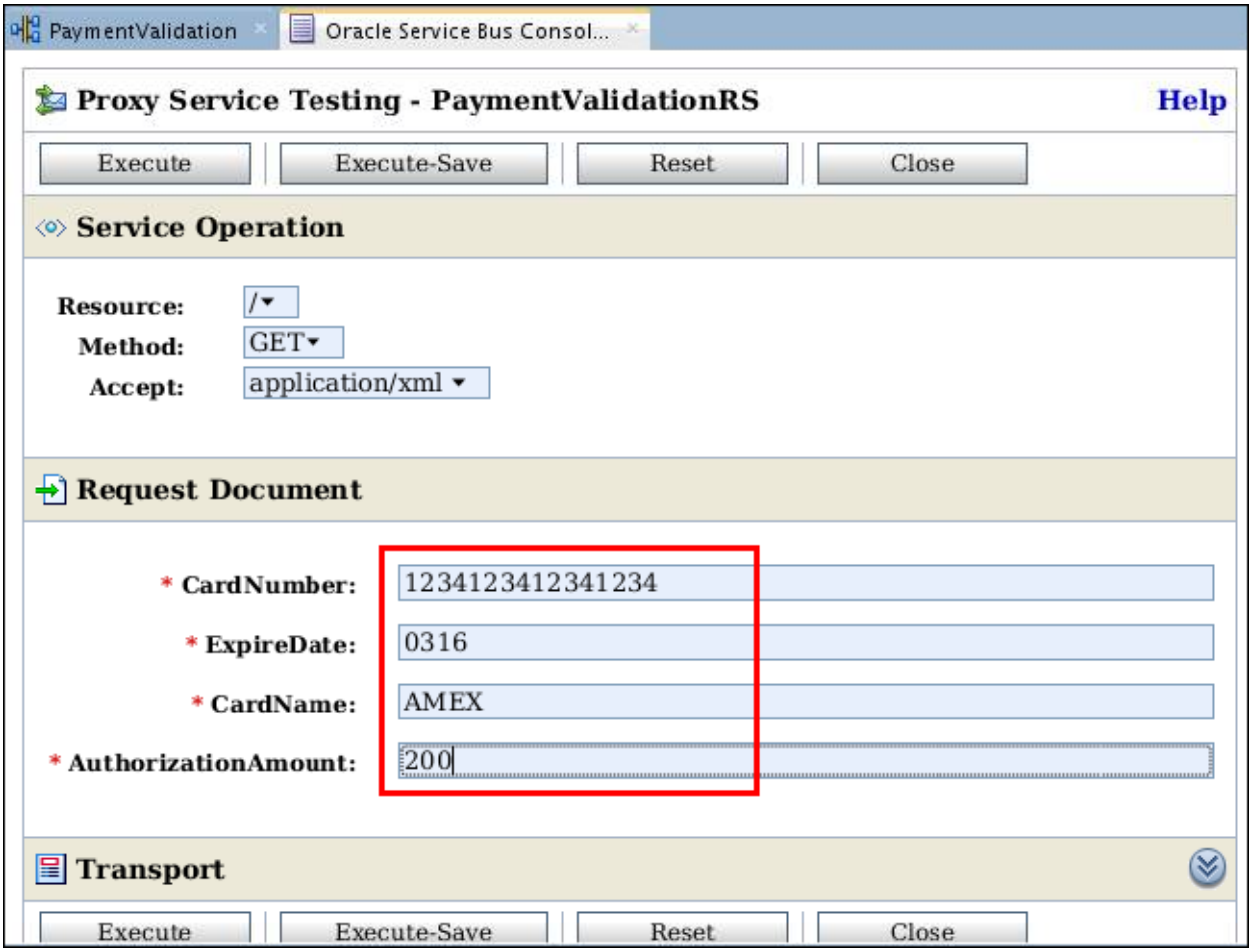


n. Click OK.

Now you have two proxy services for the same pipeline to support SOAP/XML, REST/XML, and REST/JSON using the same backend service.

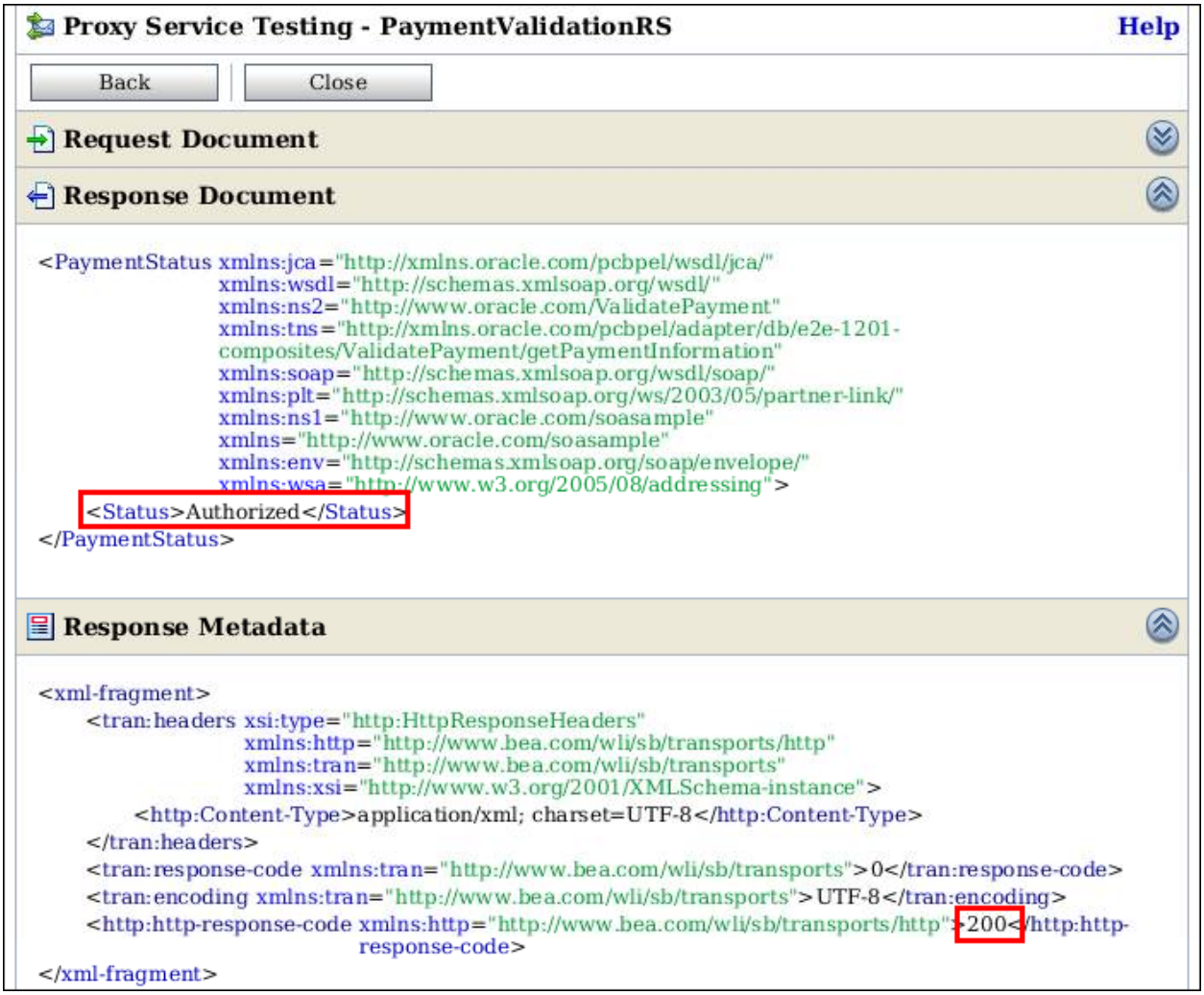


3. Test the new PaymentValidationRS proxy service.
   a. Right-click the PaymentValidationRS proxy service and select **Run**.
   b. The Test Console is launched.
   c. Enter test data to test the service as shown below:



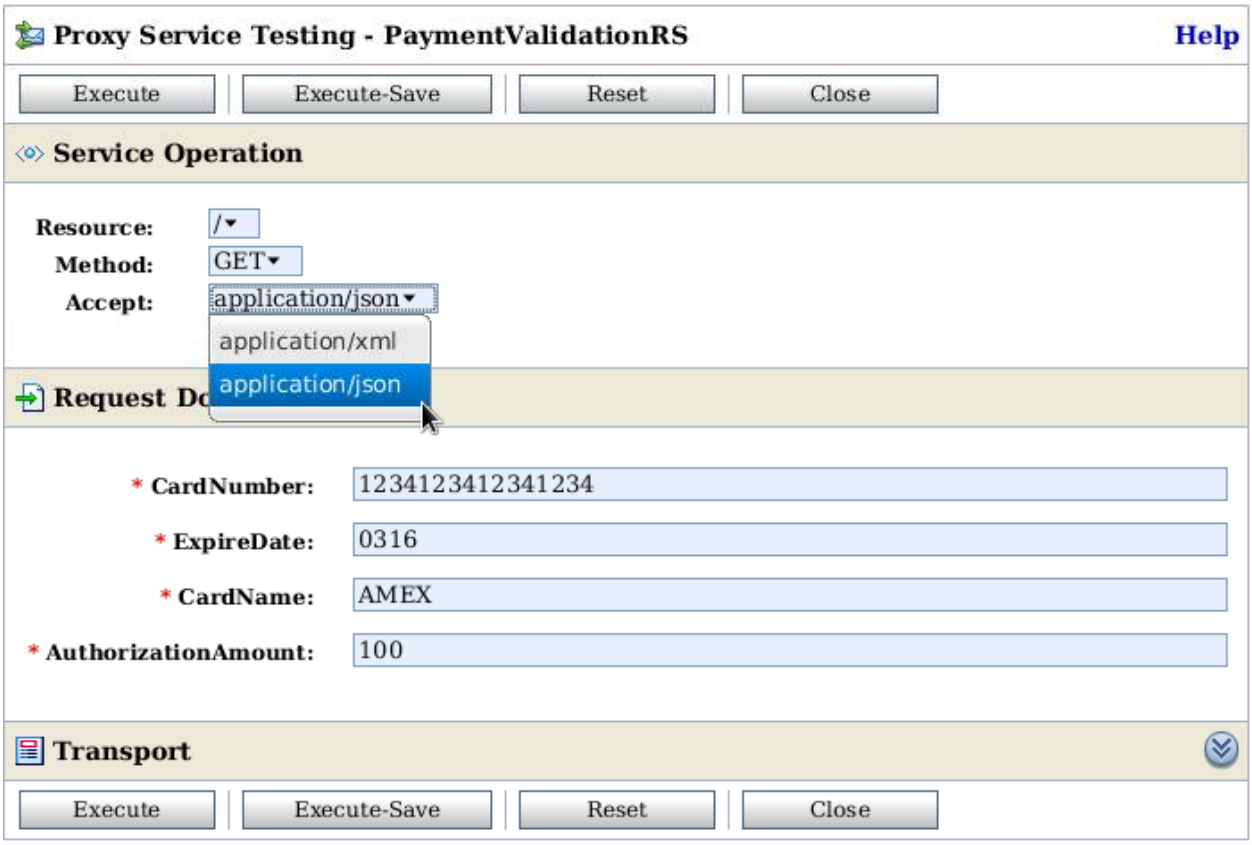
   d. Click **Execute**.

The test console will show the request and response payload. Look for the status returned by the service. Notice that it is in a REST/XML form.



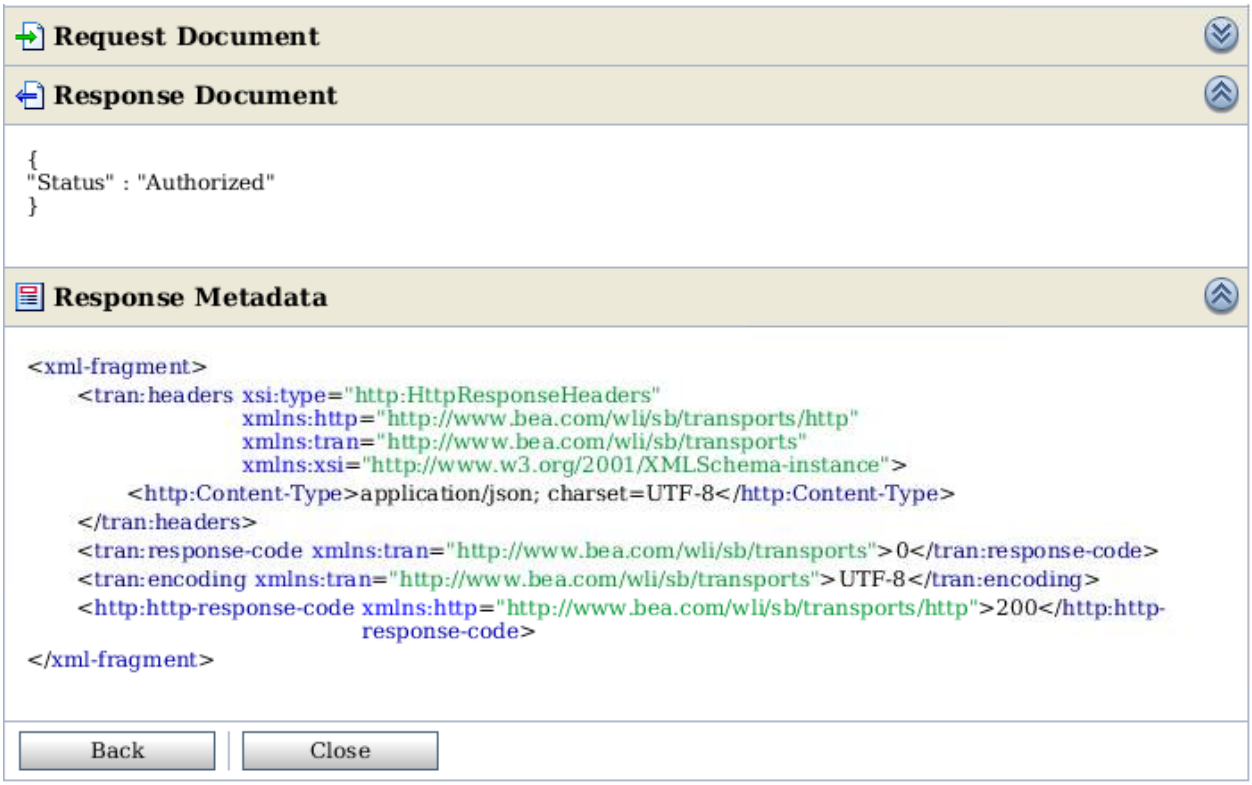
e. Click **Back**.

f. Re-run the test, but this time selecting application/json as the Accept parameter.



g. Click **Execute**.

The test console will show the request and response payloads, but this time the response is in the JSON format.



4. When you are done, close the application and all open windows of this project.

Practices for Lesson 11: Reliable Messaging

Chapter 11

Practices for Lesson 11

Practices Overview

There are no practices for this lesson titled “Oracle Service Bus 12*c*: Design and Integrate Services–Reliable Messaging.”

General Notes

There are no notes.

Practices for Lesson 12: Service Bus Security

Chapter 12

Practices for Lesson 12: Overview

Practices Overview

In the practices for this lesson, you perform the following:

- Configure and set up Oracle WSM and Oracle Service Bus security environment by creating the identities.
- Configure and invoke a proxy service requiring User Name Token authentication.
- Propagate identity of the authenticated user from Oracle Service Bus to the web service application by specifying an SAML Oracle WSM Service Policy

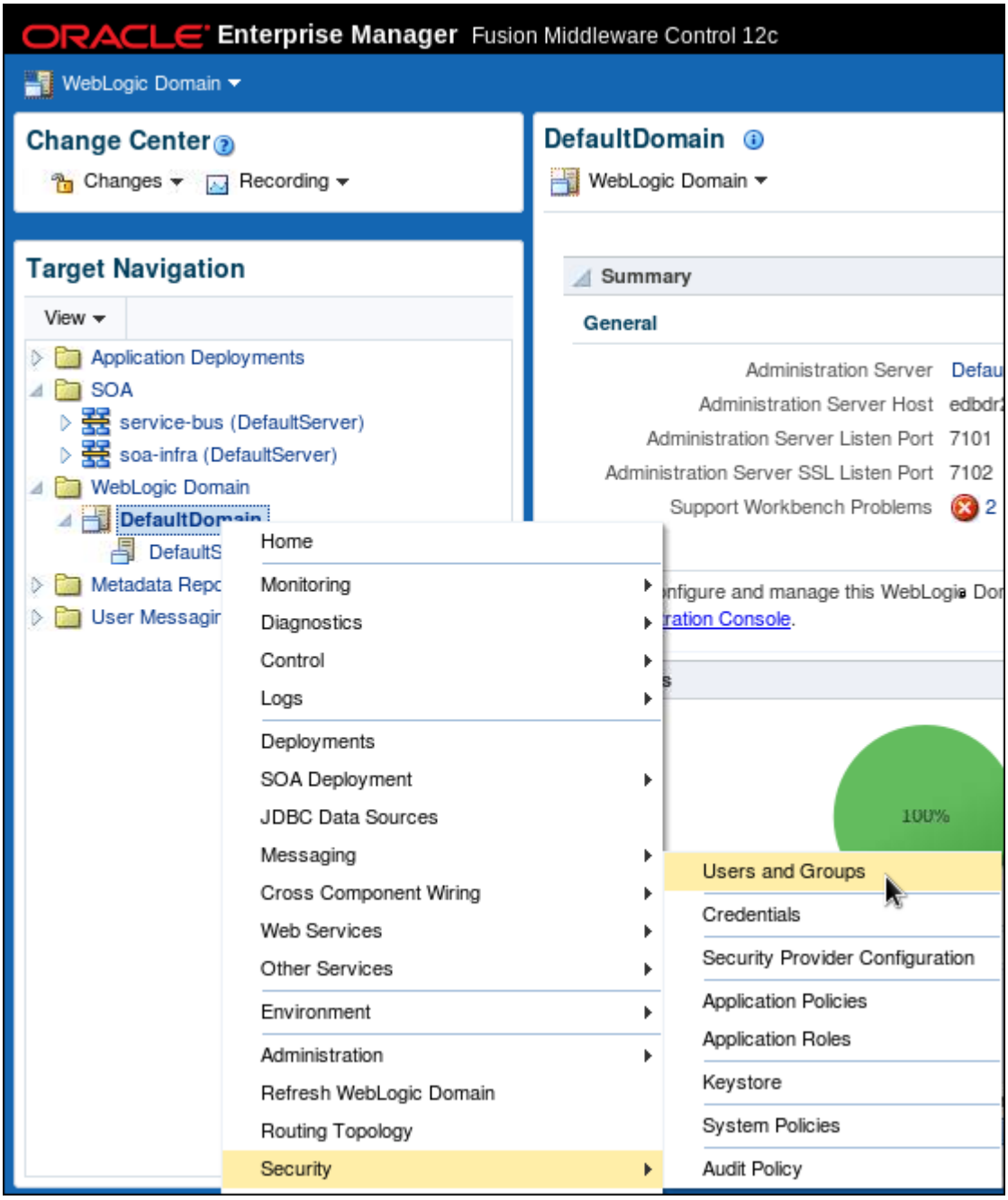
Practice 12-1: Configuring the Security Environment

Overview

In this practice, you configure Oracle WSM security by configuring the identities.

Tasks

1. Create a user in WebLogic Server (WLS) by using Enterprise Manager. The Service Bus proxy uses the WLS default authenticator to authenticate the username and password in the WS-Security SOAP Headers received from the client. The user created using Enterprise Manager is available to the WLS default authenticator.
   a. In the Enterprise Manager console, right-click **DefaultDomain** and select **Security** > **Users and Groups**.

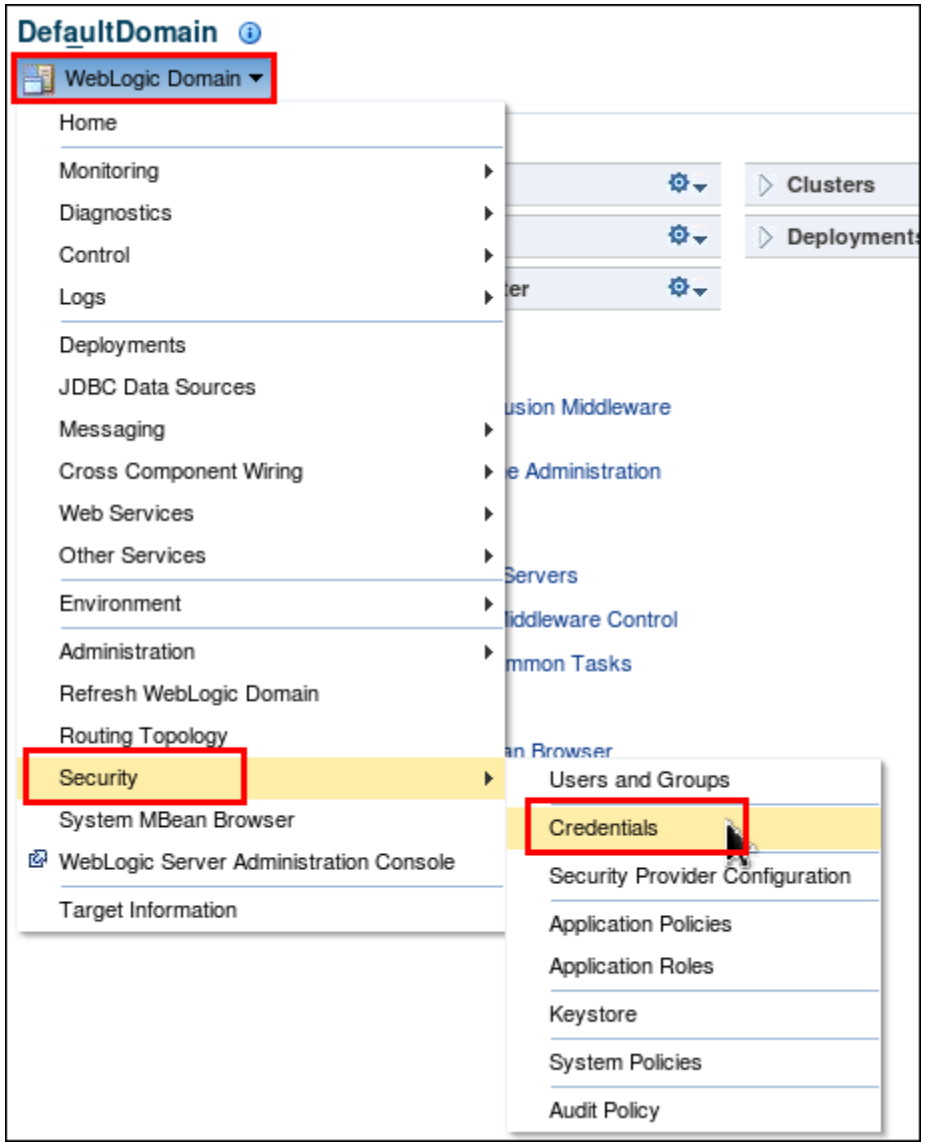


b. In the Users and Groups page, click **Create**.

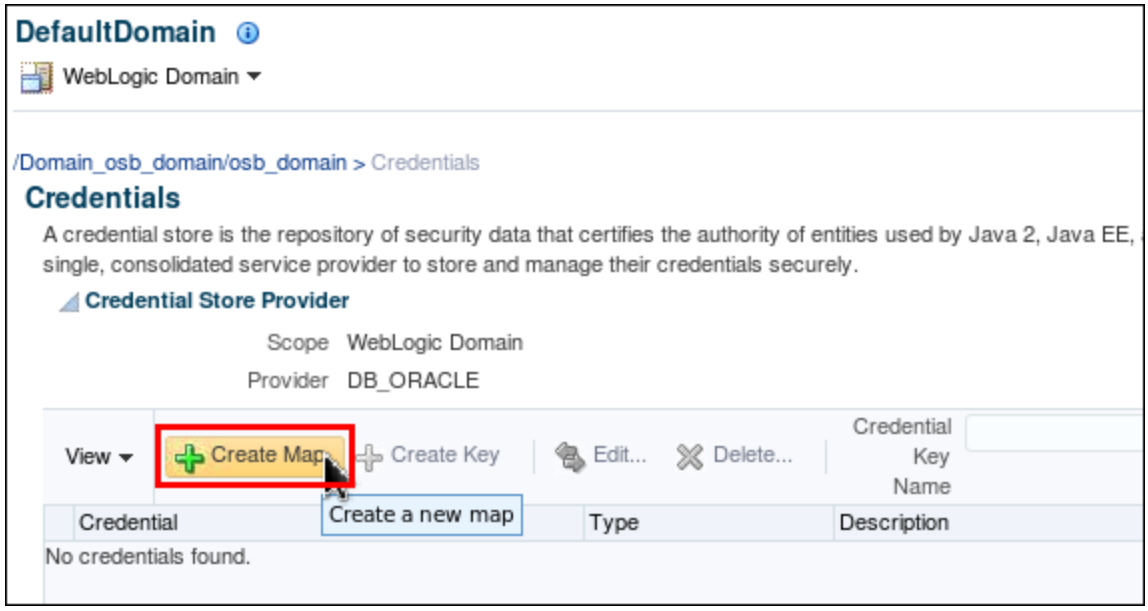
c. Create a user by using the information specified in “Course Practice Environment: Security Credentials”, and then click **Create**:

You see a user-created confirmation message.

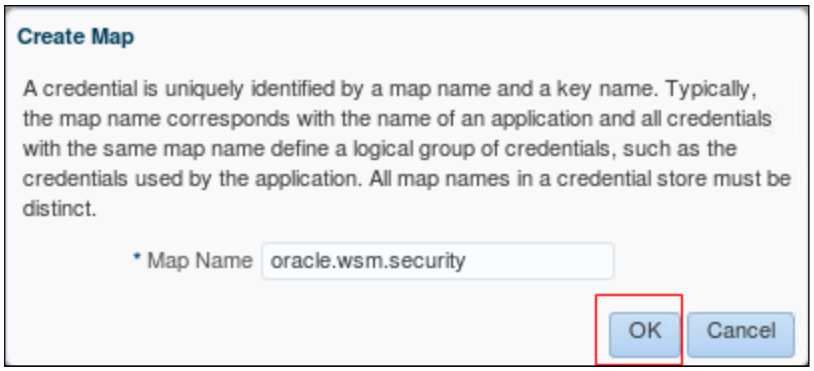
2. Add a csf-key for the user joe in Enterprise Manager. This step is required for the Service Bus Test Console to look up the username and password using the csf-key.
   a. In Enterprise Manager, expand **WebLogic Domain**, and click **DefaultDomain**.
   b. In the DefaultDomain page, from the **WebLogic Domain** drop down menu, click **Security** > **Credentials**.



c. Click **Create Map.**



d. Add the following name: **`oracle.wsm.security`**, and click **OK**.



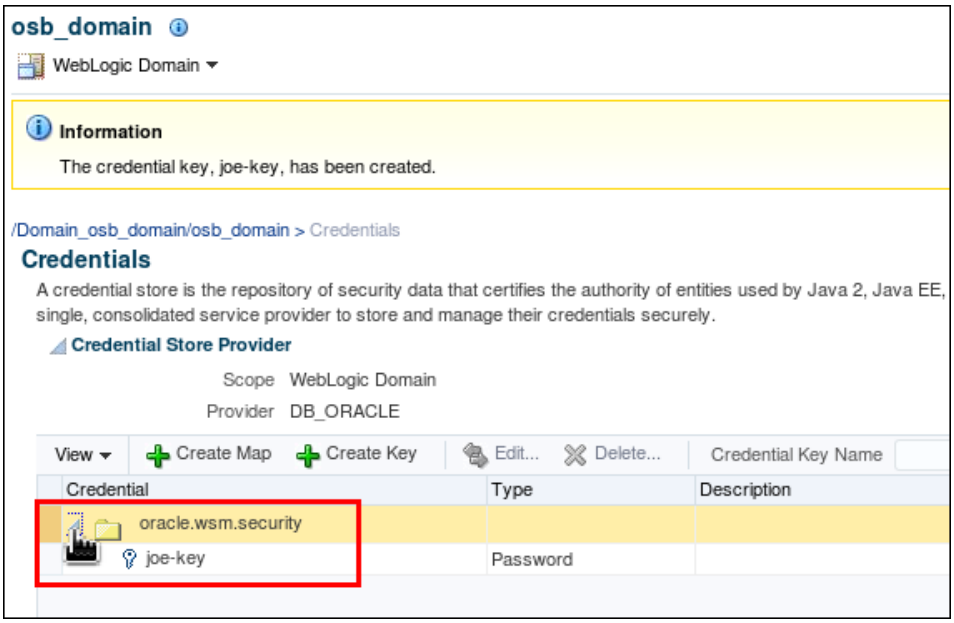
e. Select the `oracle.wsm.security` map and click **Create Key.**

f. On the Create Key page, specify the following options and then click **OK**.
Map: **`oracle.wsm.security`**

Key: **`joe-key`**
Type: **Password**
User Name: **`joe`** (same as entered in Service Bus Console)
Password: Refer to “Course Practice Environment: Security Credentials”

You should see the message: “**The credential key, joe-key, has been created**.”

g. Expand **oracle.wsm.security** and you should see joe-key.



Practice 12-2: Securing Back-end Web Service and Attaching Security Policy to Business Service

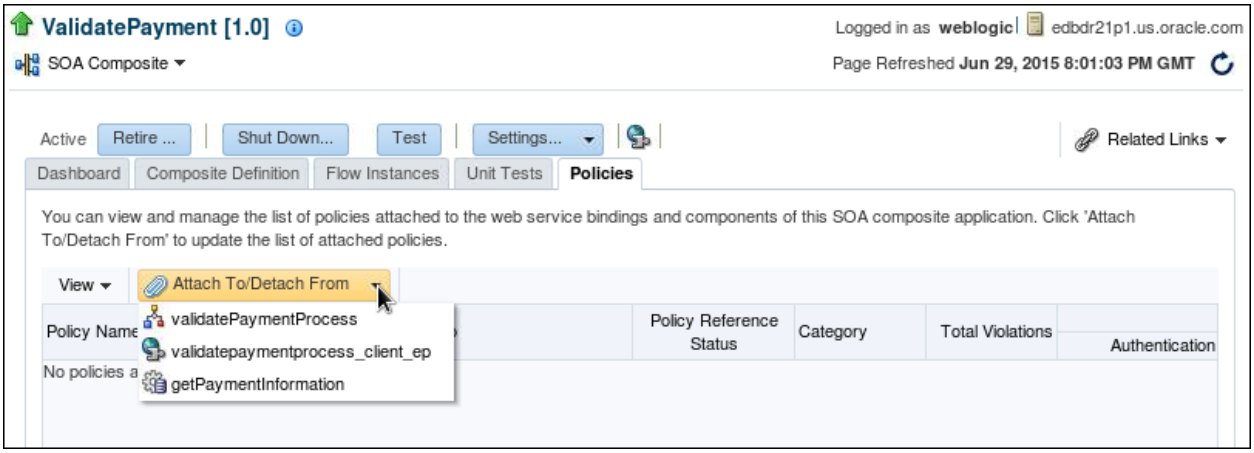
Overview

In this practice and the next practice, you propagate the identity of the user authenticated in Oracle Service Bus to the backend web service. As part of this practice, you perform the following steps:

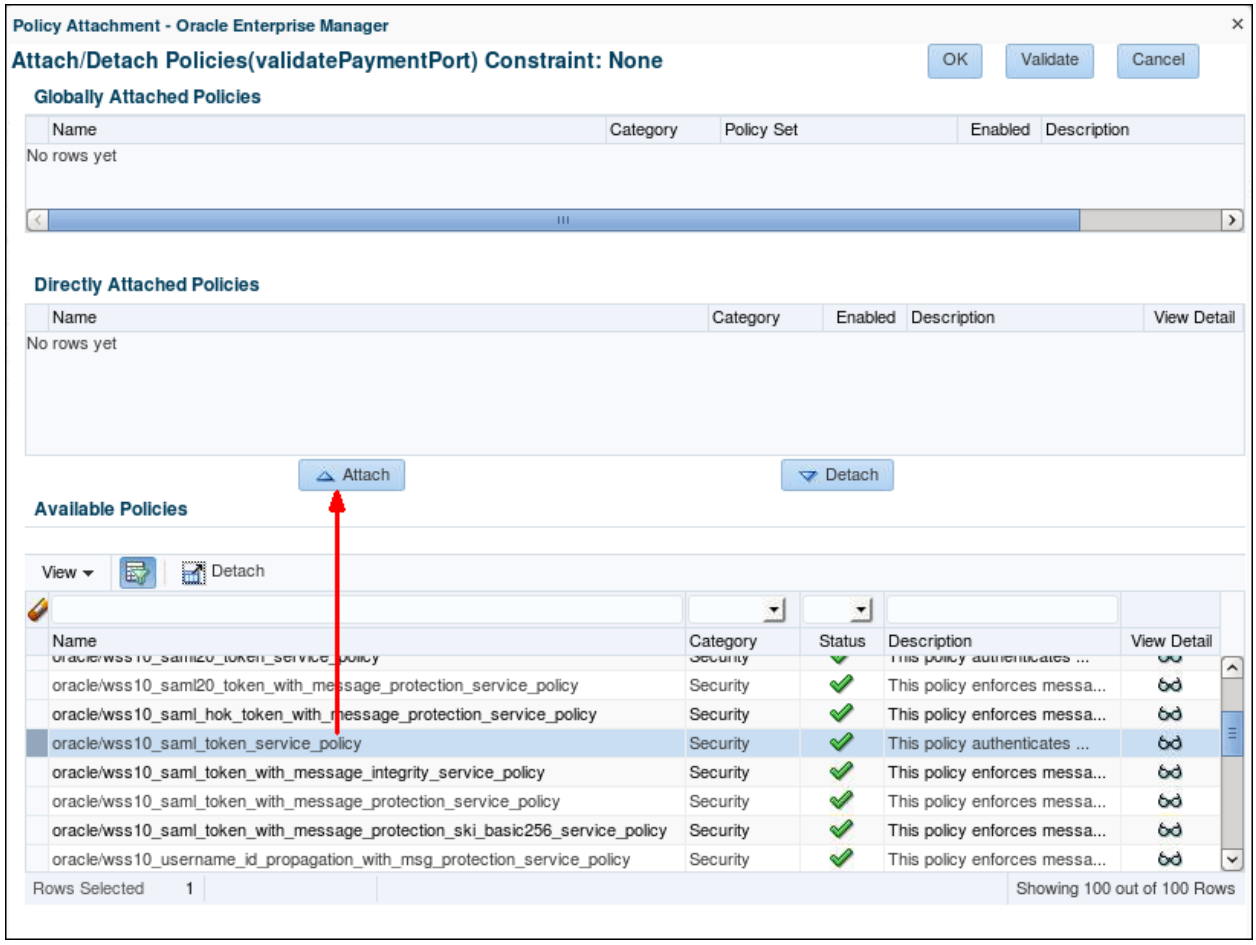
- Protect the ValidatePayment web service by using the oracle/wss10_saml_token_service_policy service Oracle WSM policy.
- Update and attach the business service in Oracle Service Bus with the oracle/wss10_saml_token_client_policy client Oracle WSM policy.

Tasks

1. Add an SAML service Oracle WSM policy to the ValidatePayment web service.
   a. In the Target Navigation panel of EM console, expand **SOA** > **soa-infra** > **default**, and select **ValidatePayment [1.0]**.
   b. On the ValidatePayment home page, select **Policies** tab.
   c. Click the **Attach To/Detach From** drop-down menu, and select **validatePaymentprocess_client_ep**.



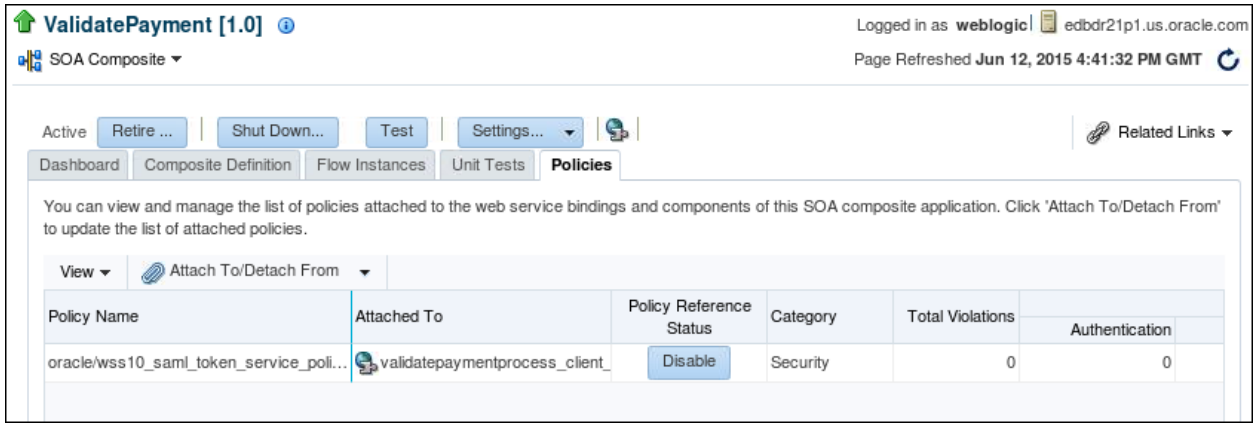
   d. In the Attach/Detach Policies window, locate and select **`oracle/wss10_saml_token_service_policy`** in the Available Policies panel, and then click **Attach**.



You should see the selected policy displayed in the Directly Attached Policies panel.

e. Click **OK.**

Verify that the policy is attached to the service.

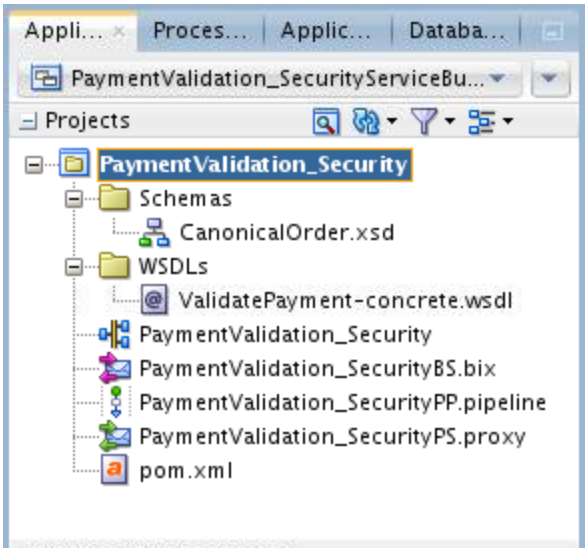


2. Add an SAML client Oracle WSM policy to the `PaymentValidation_Security` business service.

   a. In JDeveloper, select **File** > **Open** from the menu bar.

   b. In the Open Application(s) dialog box, navigate to the `$HOME/labs_DI/lessons/lesson12/PaymentValidation_SecurityServiceBusApp/` directory, and open the `PaymentValidation_SecurityServiceBusApp.jws` file.

c. Expand the folders in Application Navigator, and you should see the folder structure resembling the image below:

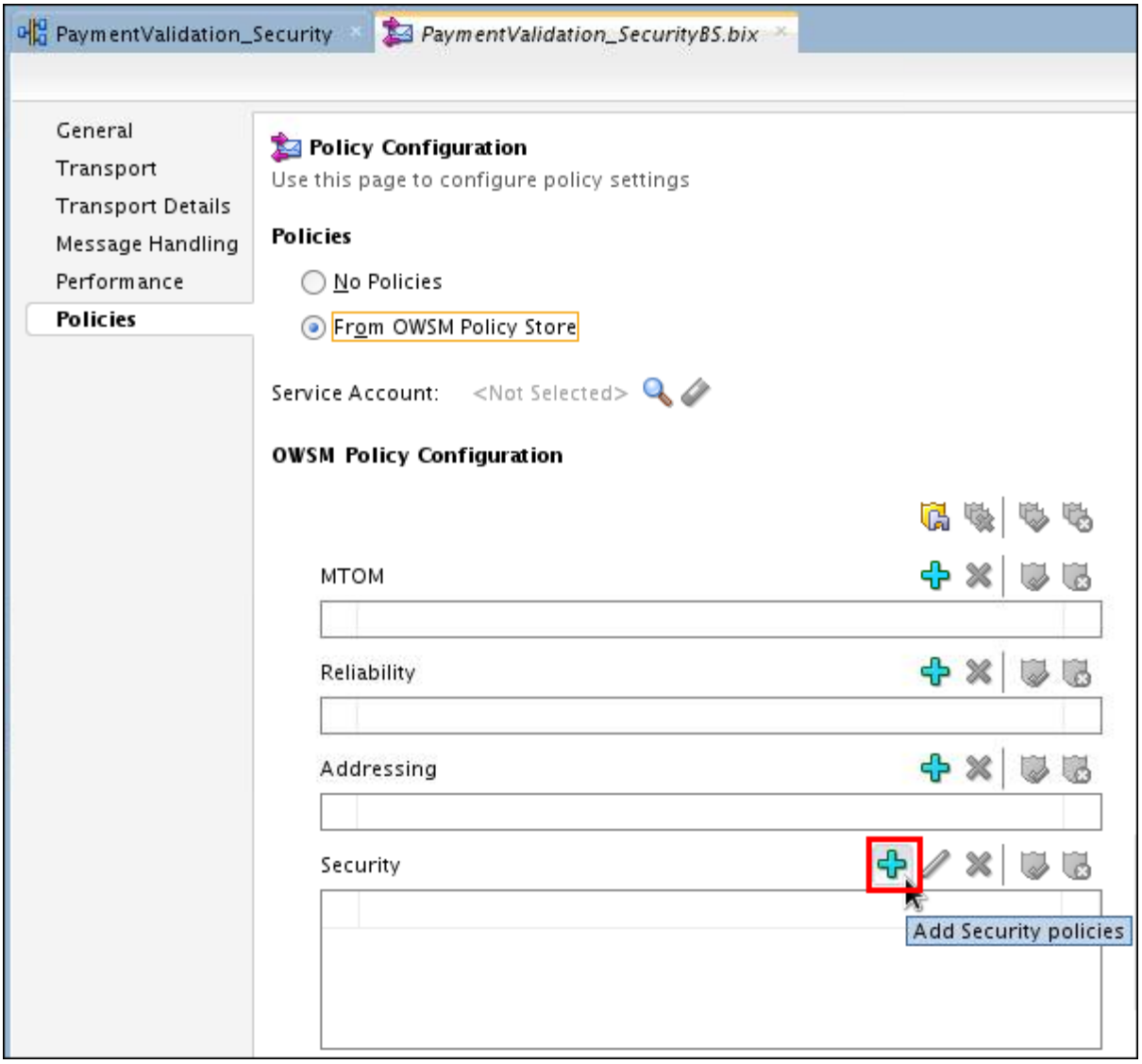


d. In the Application navigator, double-click PaymentValidation_Security to open it in Service Bus Overview editor.

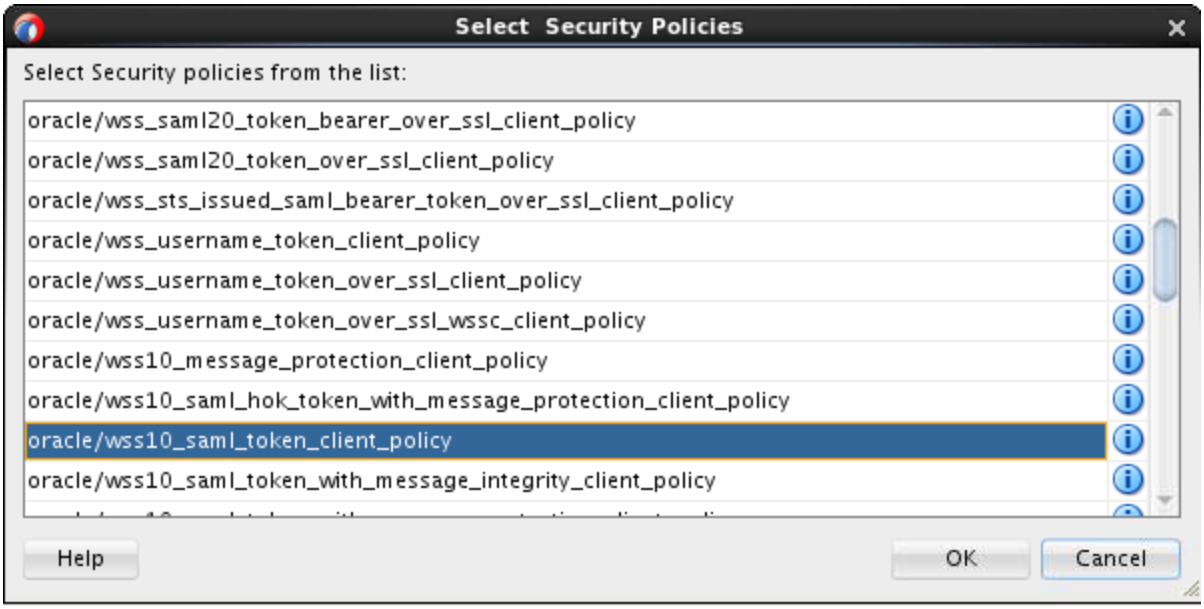
e. Double_click the business service to open its configuration editor.

f. Click the Policies tab.

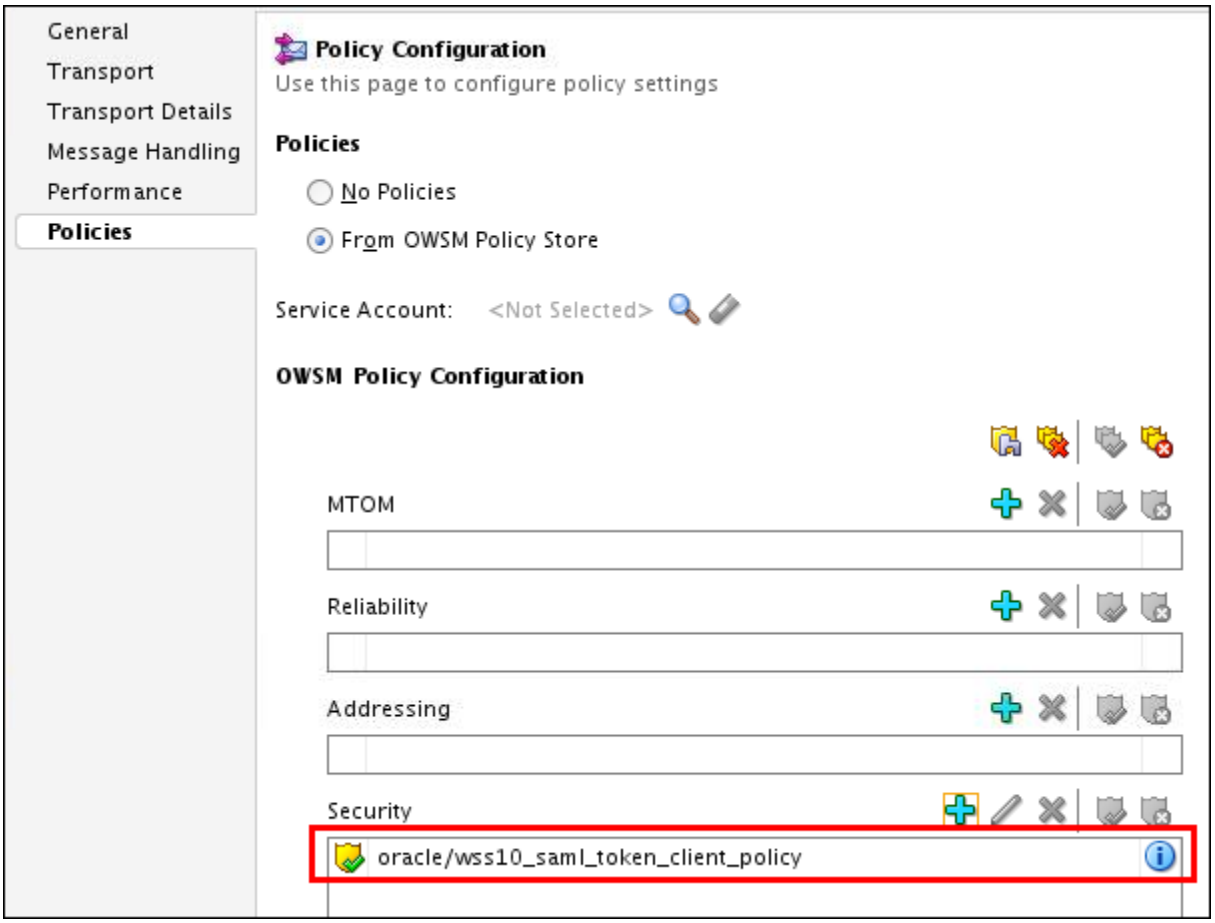
g. Under the Policies tab, select **From OWSM Policy Store,** and then click **Add Security policies** (the plus icon).



h. In the Select Security Policies dialog box, locate and select **`oracle/wss10_saml_token_client_policy`**, and then click OK.



i. Verify that the policy is attached.



j. Save your project.

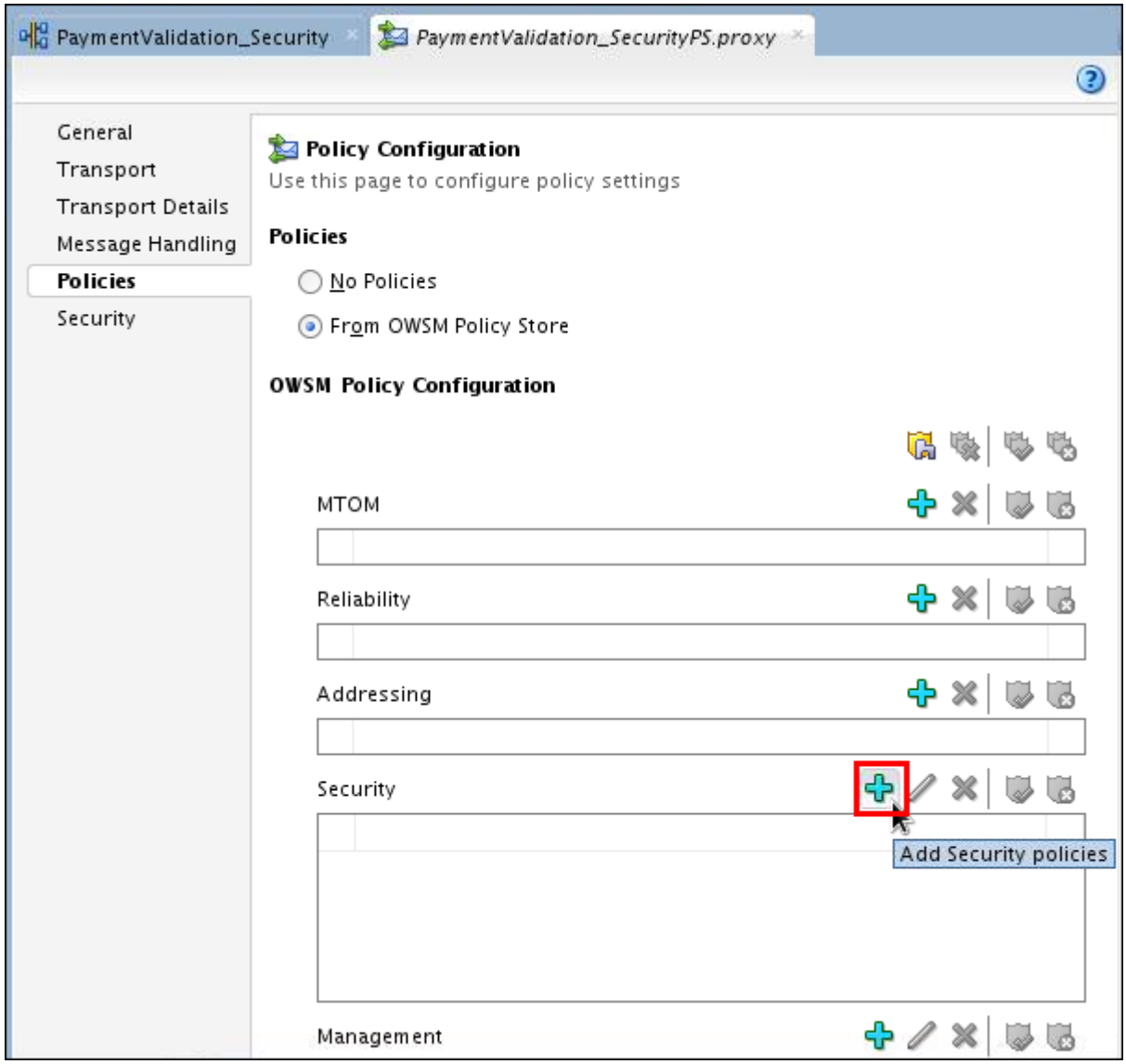
Practice 12-3: Applying a Security Policy to Proxy Service

Overview

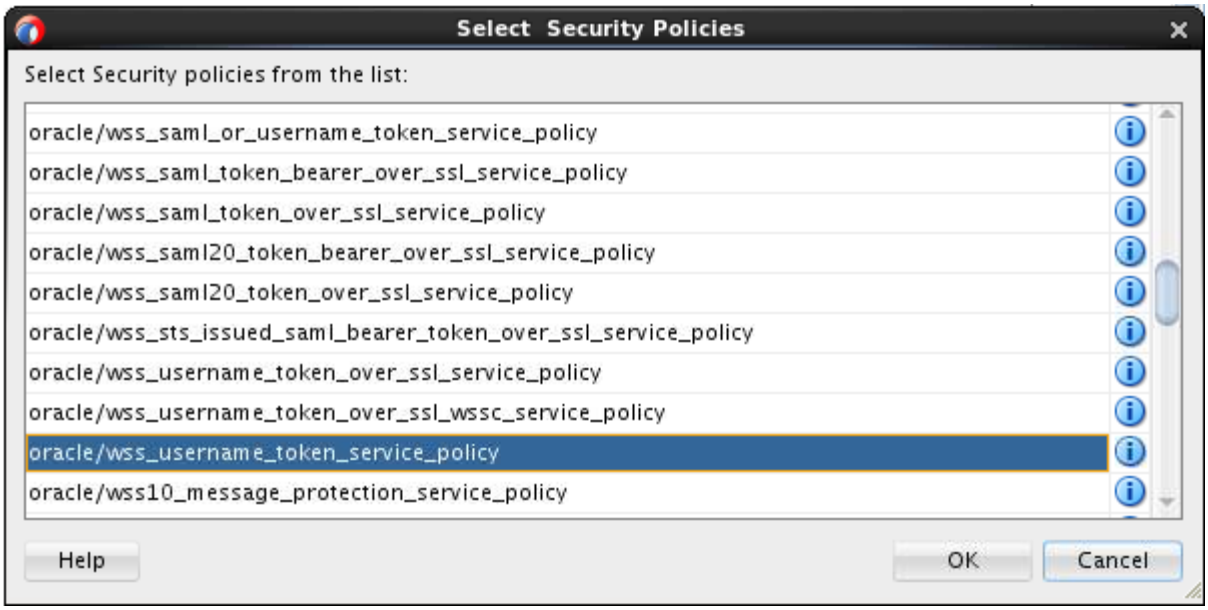
In this practice, you add Oracle/wss_username_token_service_policy Oracle WSM policy at design time to the proxy service in JDeveloper, and then test your security implementation end-to-end.

Tasks

1. Add User Name Token Service Oracle WSM policy to the proxy service.
   a. In the PaymentValidation_Security Service Bus Overview editor, double click the proxy service to open its configuration editor.
   b. Click the Policies tab.
   c. Under the Policies tab, select **From OWSM Policy Store,** and then click **Add Security policies** (the plus icon).



d. In the Select Security Policies dialog box, locate and select **`oracle/wss_username_token_service_policy`**, and then click OK.



e. Verify that the policy is attached.

f. Save your project.

Test

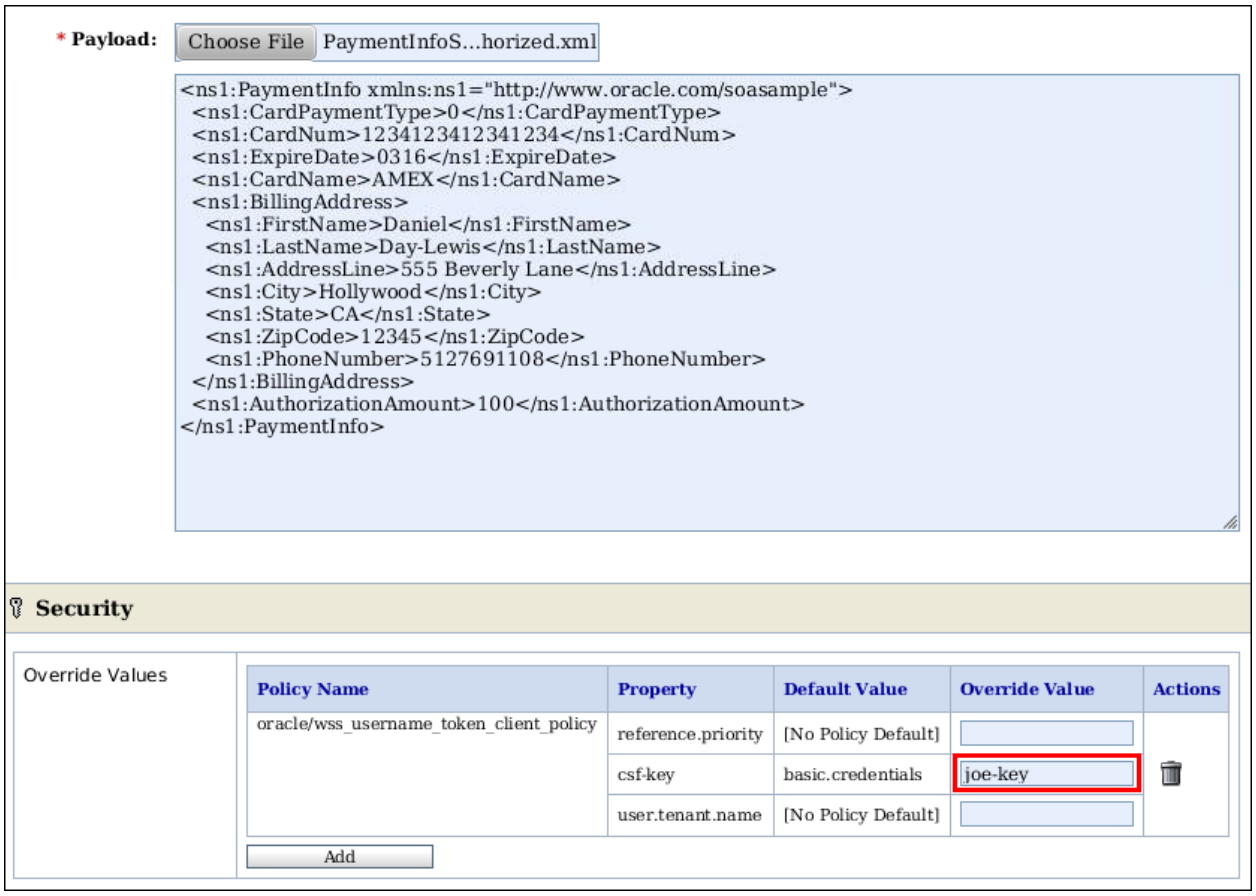
1 Perform a positive test and a negative test to confirm the Oracle WSM security (User Name Token policy) implementation on the proxy service.

2. In the Service Bus Overview Editor, right-click the proxy service, and select Run.

3. In the Test Console, choose **/home/oracle/labs_DI/resources/sample_input/ PaymentInfoSample_Authorized.xml** as the test file.

   Since the CreditCardService_Proxy proxy service is attached with the User Name Token Oracle WSM policy, you notice a Security section in the Test Console with an oracle/wss_username_token_client_policy policy name and with a csf-key property.

4. Specify **joe-key** as the override value for the csf-key property.

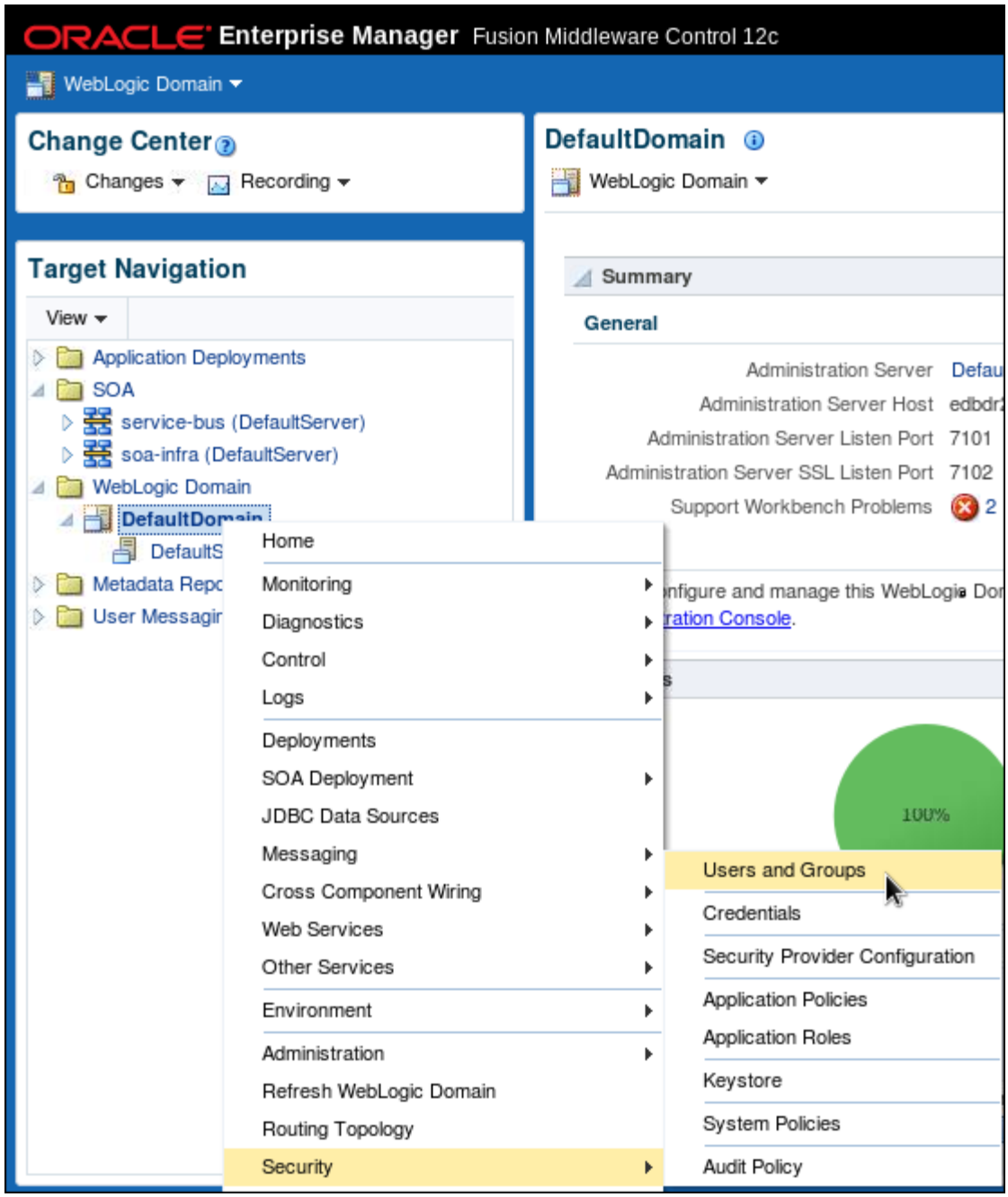


Note that the mapping joe-key csf-key with the credentials `joe/welcome1` is already created by using EM console in the previous practice.

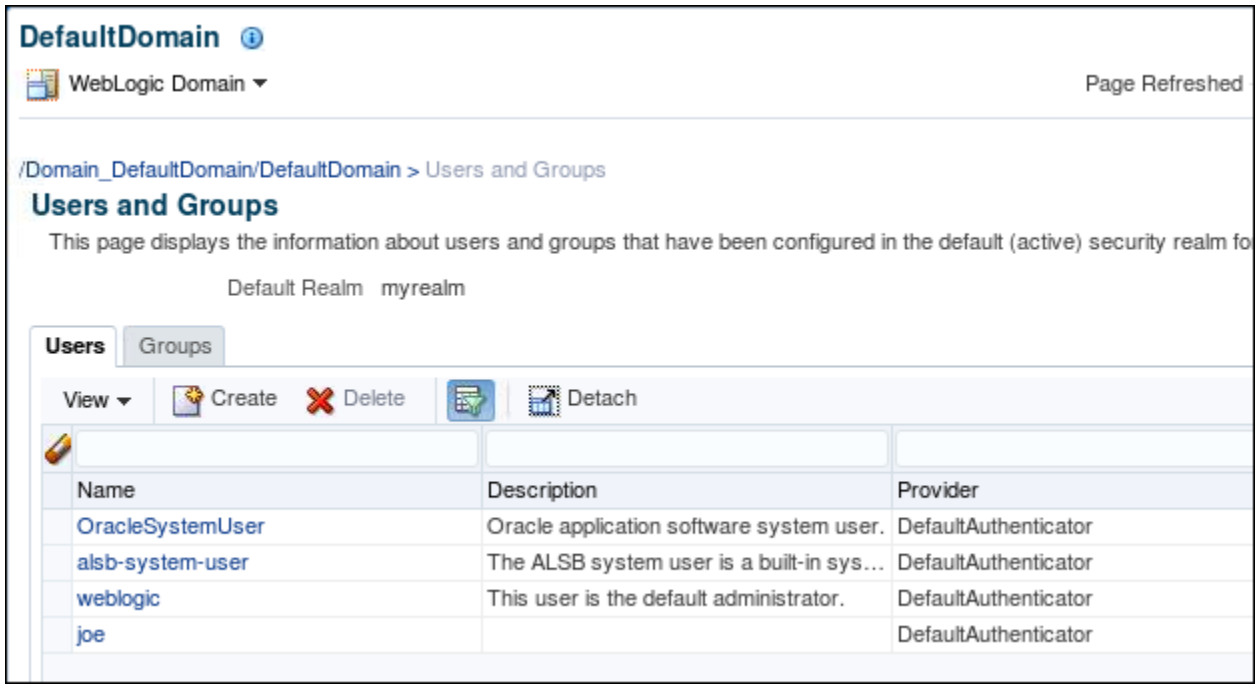
5. Click **Execute** to test the proxy service.

   You should see that the payment status is “Authorized” in the response message.

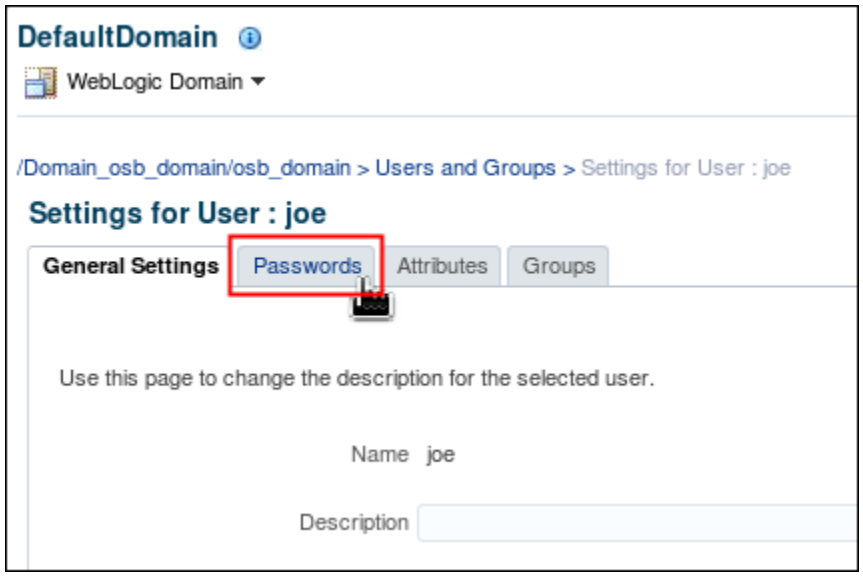
6. Perform a negative test by modifying the password of user `joe` to a different value, and then invoking the proxy service.
   a. In EM console, right-click **DefaultDomain** and select **Security** > **Users and Groups**.



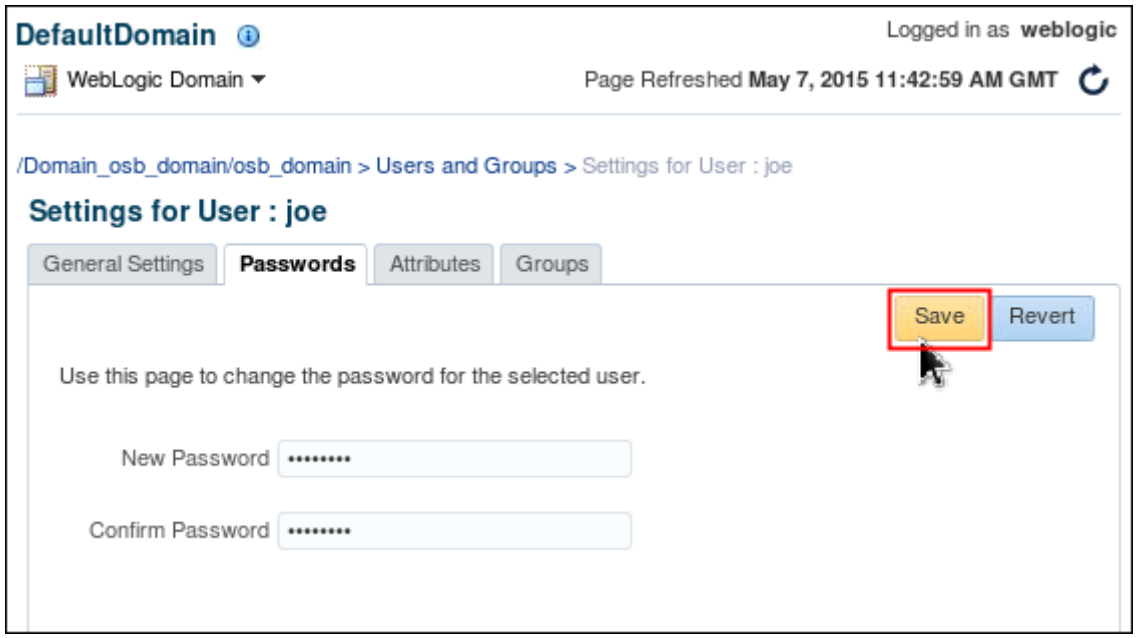
b. In the Users and Groups page, under the Users tab, click **joe**.



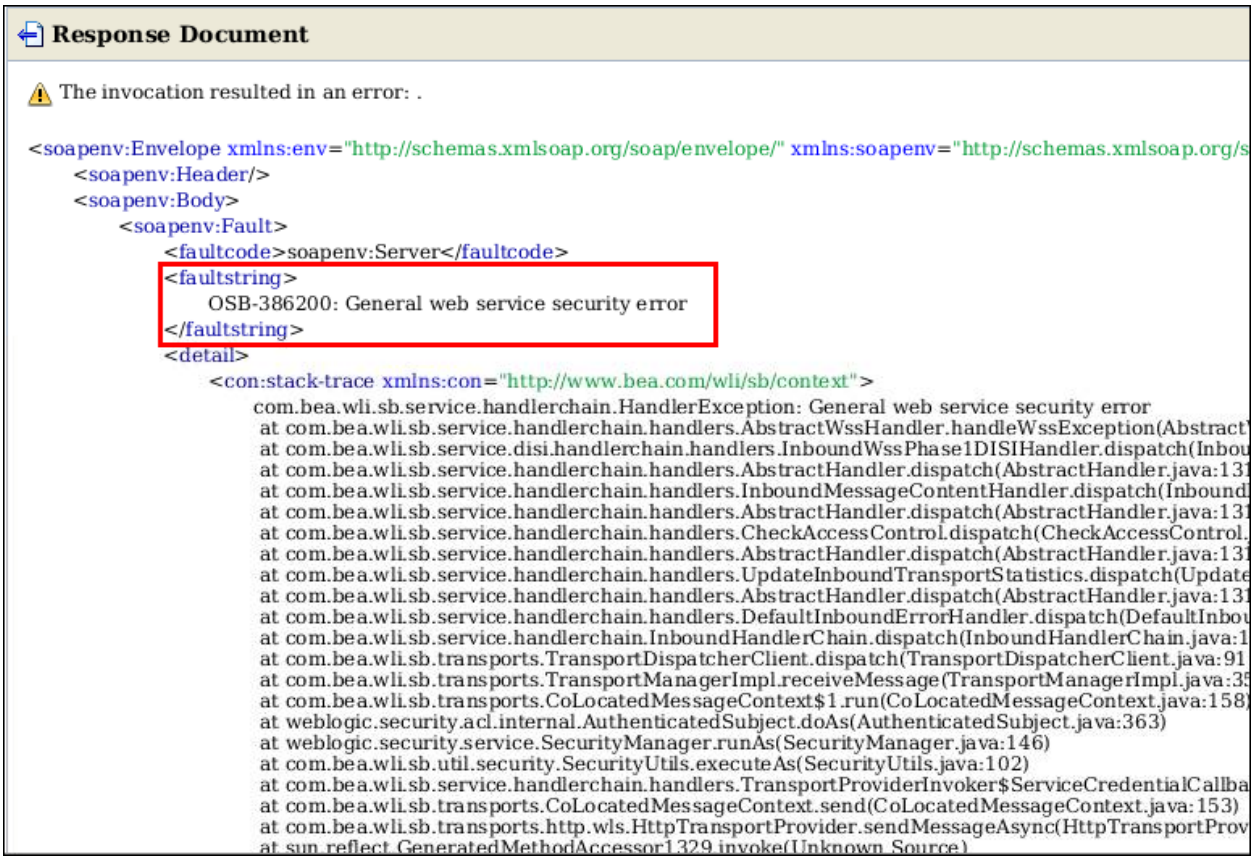
c. On the “Settings for User: joe” page, click the **Passwords** tab.



d. Modify the password to anything but the real password, and then click **Save**.



e. Go back to the Service Bus Test Console in JDeveloper, and execute the same test case (the one you performed for the positive test). You should see the request fail with the following response, as joe's password does not match the value used in the Credential Map.



7. Reset the password of user joe back to the original one.

8. Disable the security policy for the ValidatePayment web service.
   a. In the Target Navigation panel of EM console, expand **SOA** > **soa-infra** > **default,** and select **ValidatePayment [1.0]**.
   b. On the ValidatePayment home page, select **Policies** tab.
   c. Click the Disable button of the oracle/wss10_saml_token_service_policy policy.

9. When you are done, close the application and all open windows of the project.

Practices for Lesson 13: Advanced Topics

Chapter 13

Practices for Lesson 13: Overview

Practices Overview

In these practices, you explore the ways to improve your service's performance, and capture and raise awareness to an error condition.

Practice 13-1: Configuring Result Caching to Improve Service Performance

Overview

Service Result caching is used when the external service, which your business service connects to, returns a relatively static response. By using Service Result Caching, you do not invoke external service for the same request. Instead, it takes the response from cache which improves OSB performance.

In this practice, you use ValidatePayment as the back-end service, but with an added wait activity of 30 seconds. So when the business service is invoked for the first time, it should take over 30 seconds to return the response, and when the business service gets called the next time with the same request, it should not take more than 1-5 seconds as it would get the response from the cache.

Assumptions

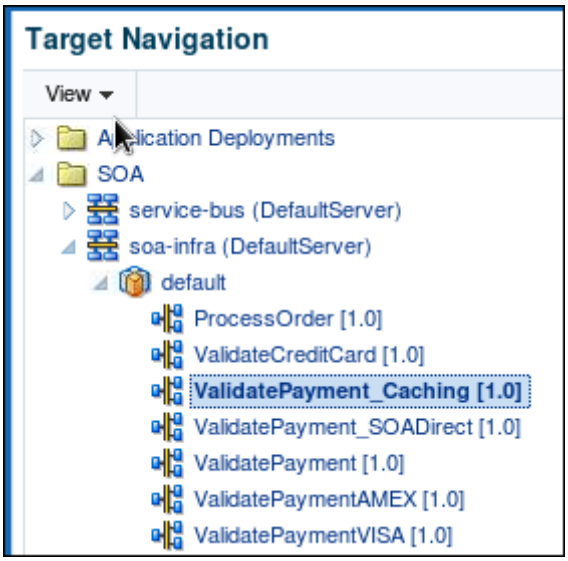
Tasks

1. In a terminal window, enter the following commands:

```
>cd ~/labs_DI/lessons/lesson13
>ls -l
>$HOME/labs_DI/resources/Scripts/deploy.sh
sca_ValidatePaymentCaching_rev1.0.jar
```

   The build and deployment information will be displayed in the console. You should see a message showing both build and deploying executed successfully.

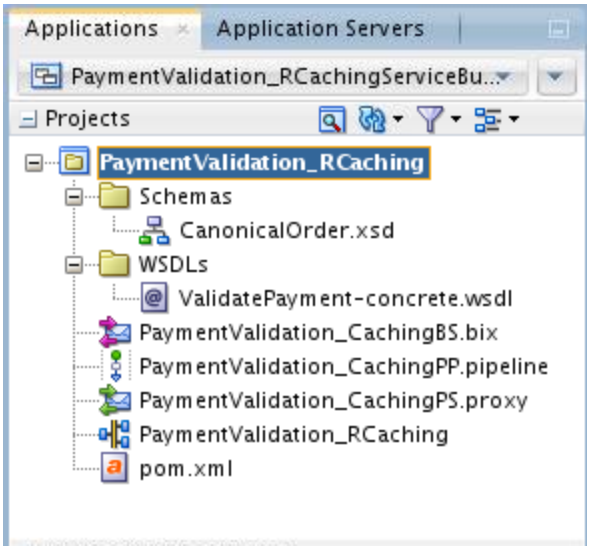
2. Test the deployed service in EM
   a. Open the EM console. In Target Navigator, expand SOA > soa-infra > default. Now you should see the deployed composite application ValidatePayment_Cachingt[1.0] in the folder.



   b. Select the composite and test it using the PaymentInfoSample_Authorized.xml file as the request. The PaymentInfoSample_Authorized.xml file is located in the /home/oracle/labs_DI/resources/sample_input/ folder.

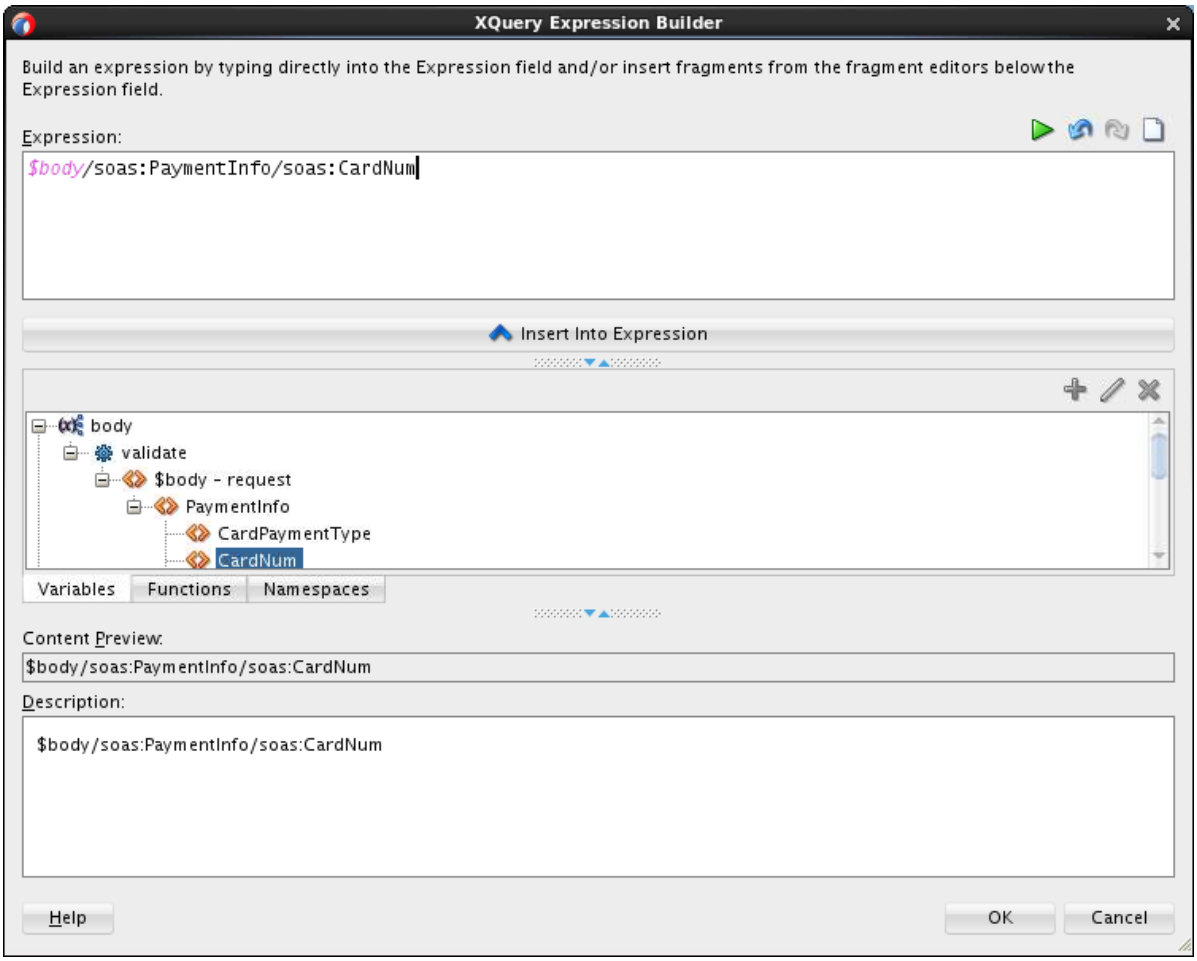
      Note that you need to wait more than 30 seconds to see the response.

3. Open Service Bus project
   a. In JDeveloper, select **File** > **Open** from the main menu.
   b. In the Open Application(s) dialog box, navigate to the `$HOME/labs_DI/lessons/lesson13/PaymentValidation_RCachingServiceBusApp/` directory, and open the `PaymentValidation_RCachingServiceBusApp.jws` file.
   c. Expand the folders in Application Navigator, and you should see the folder structure resembling the image below:



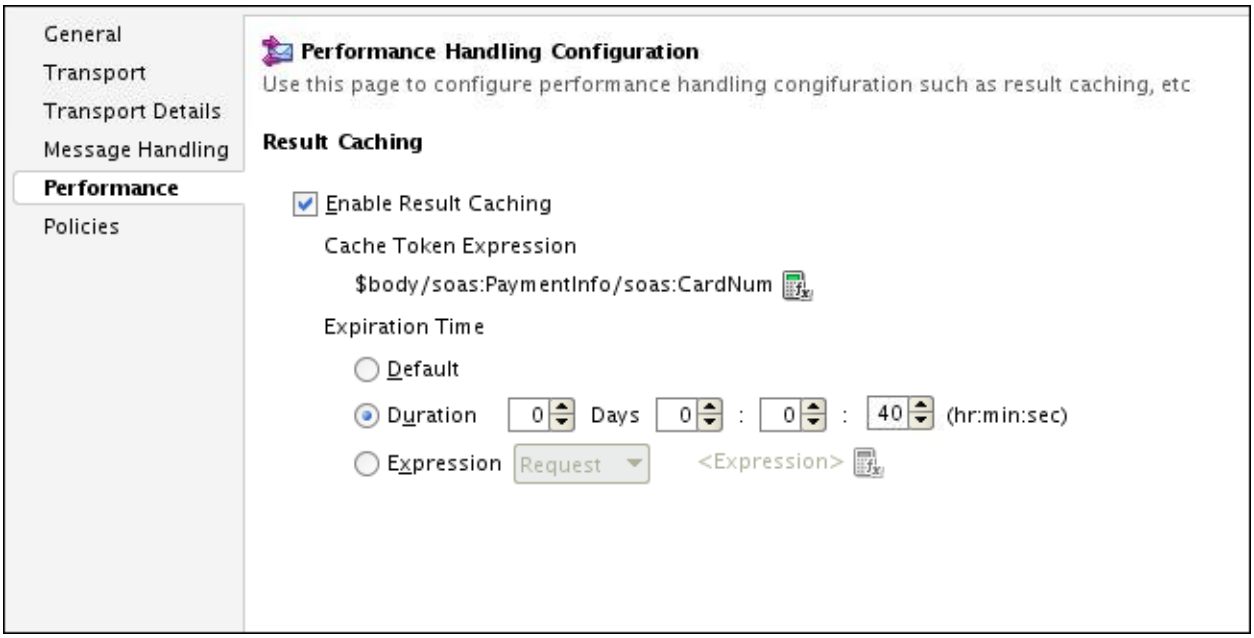
4. Configure the business service
   a. Double-click PaymentValidation_RCaching to open it in Overview Editor.
   b. Right-click the business service and select Edit from the context menu.
   c. In the business service editor, select Transport tab.
      Note that the endpoint URIs are still pointing to the ValidatePayment SOA service.
   d. Change the URIs to the endpoint URLs of ValidatePayment_Caching service that you just deployed:
      `http://<yourhostname>:7101/soa-infra/services/default/ValidatePayment_Caching/validatepaymentprocess_client_ep`
   e. Click the Performance tab.
   f. Check the box next to Enable Result Caching.
   g. Specify Cache Token.
      1) Open XQuery Expression Editor by clicking  next to Cache Token Expression.
      2) For variables, navigate to body > validate > $body – request >PaymentInfo, and select **CardNum.**

3) Insert CardNum into the expression field.



4) Click OK.

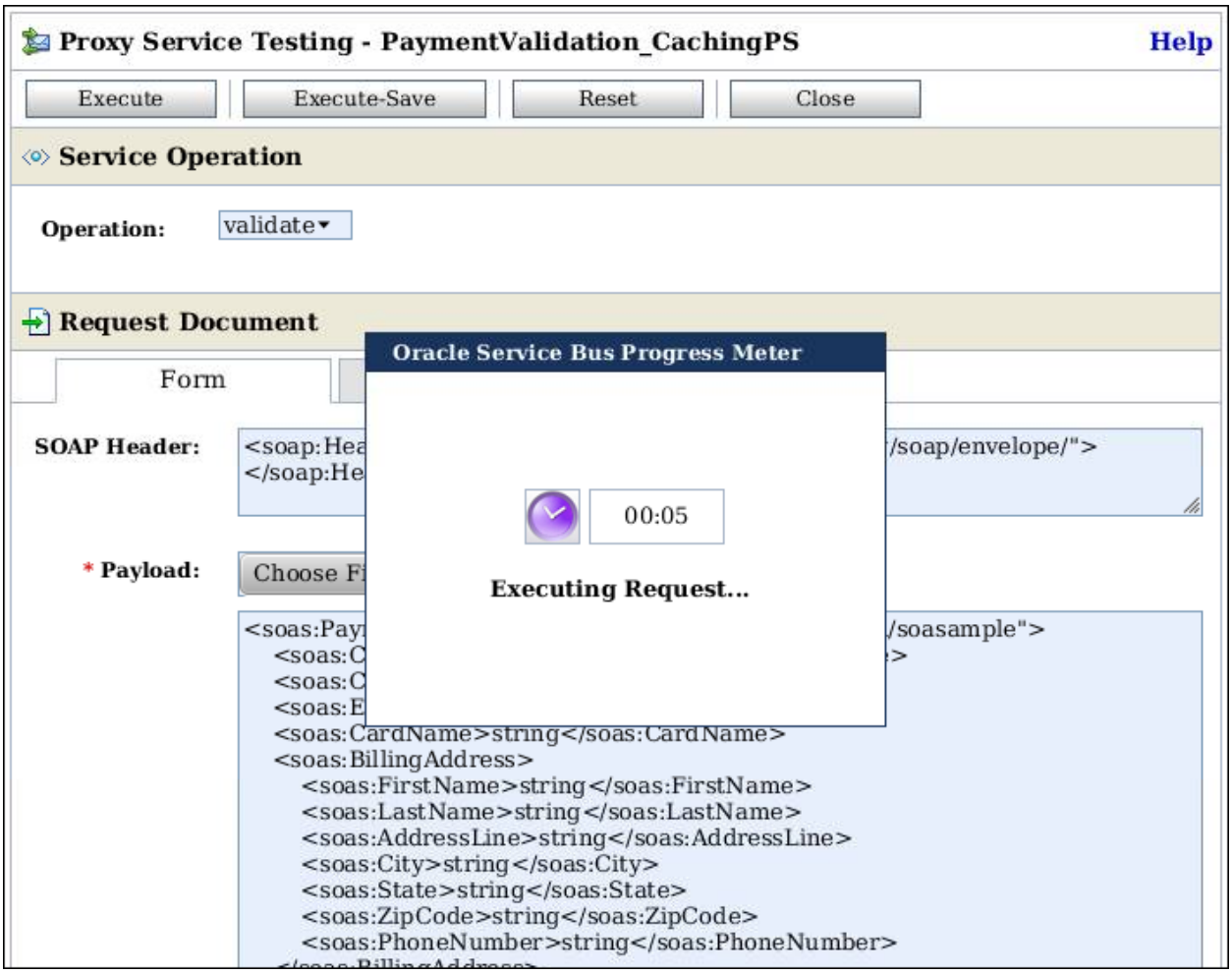
5) Under Expiration Time, set the duration to **40** seconds.



5. To test your Service Bus application, do as follows:
   a. Right-click the proxy service, and select Run.
   b. In the Test console, select ~/labs_DI/resources/sample_input/PaymentInfoSample_Authorized.xml for the request payload.

c. Click the Execute button.

This is the first-time request, so business service will invoke the external service and get the response. You should receive the response after 30 seconds because of the 30-second wait activity in the external service.



d. After the first test, click Back.

e. Test you proxy with the same request. Now you should get the response immediately as the business service takes the response from the cache this time.

6. When you are done, close the application and all open windows of the project.

Practice 13-2: Adding a Service Level Agreement Alert

Overview

Unlike a Pipeline alert which detects a condition typically based on the business logic or a handled exception, an SLA alert provides insight into other health metrics, such as response time, errors, or even load.

In this practice, you add an SLA alert when the status of the endpoint changes from up to down.

High-level steps:

- Add an SLA for Errors > 0 to the PaymentValidationBS Business Service.
- Stop the PaymentValidationBS Composite.
- Test the PaymentValidationRS proxy service.
- Observe SLA in the dashboard.

Assumptions

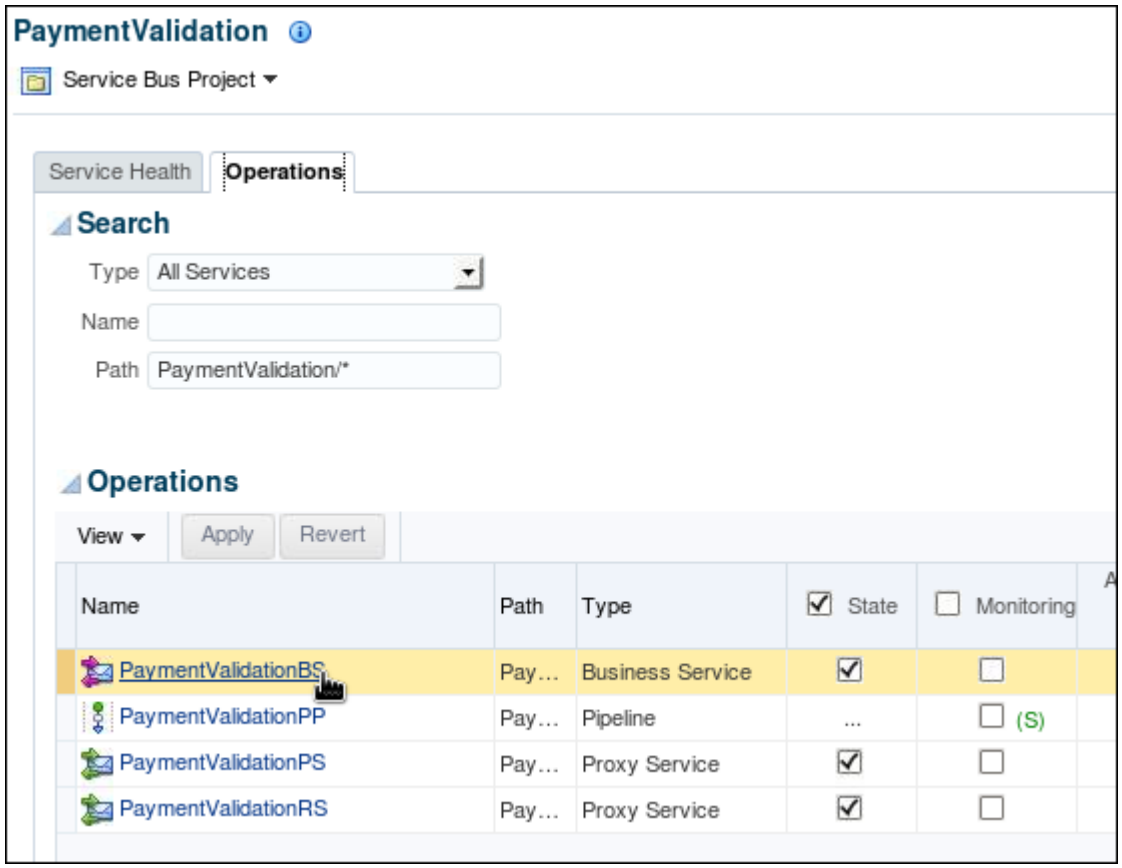
The PaymentValidation service bus application is deployed and running.

Tasks

1. Enable monitoring and SLA for PaymentValidationBS business service.
   a. Open EM Console in a browser.
   b. In Target Navigator, navigate to SOA > service-bus folder, and select PaymentValidation.

      The Service Health page for the project appears.
   c. Click the Operations tab.

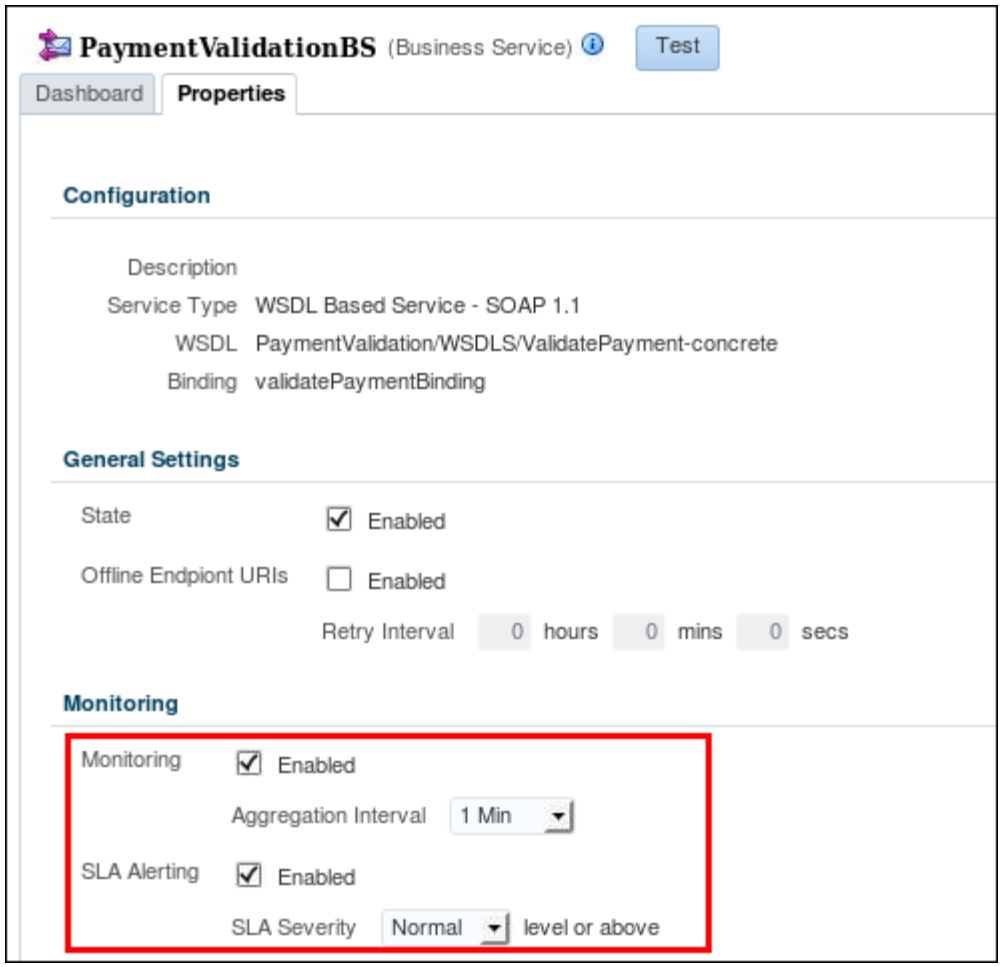
d. Click PaymentValidationBS link.



e. On the Properties tab of the business service, do as follows:

1) Enable Monitoring.
2) Set Aggregation Interval to 1 minute.

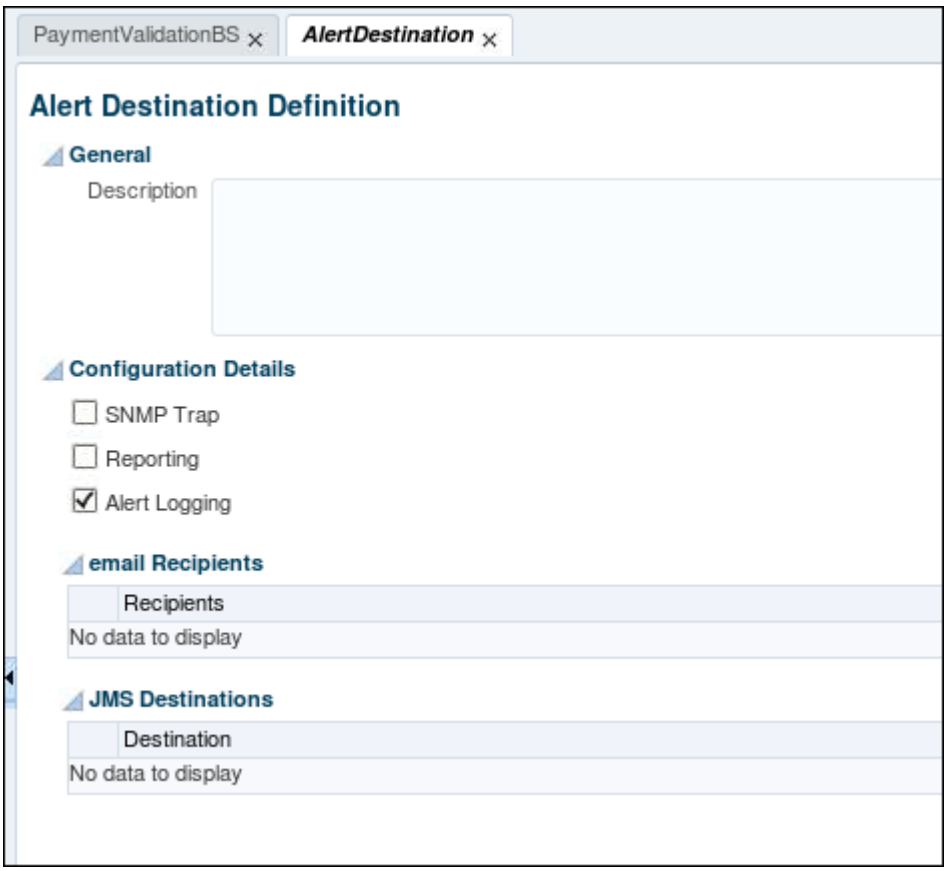
3) Verify that SLA Alerting is enabled.



f. Click **Apply** to apply your changes.

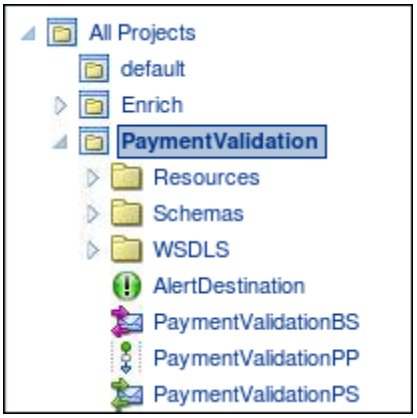
2. Define SLA Alerts for the PaymentValidationBS business service.
   a. Open Service Bus Console.
   b. Create a session by clicking the **Create** button (on the top-right of the page).
      Whenever you want to make changes, you need to create a session which contains a transaction of all of our changes.
   c. Before defining SLA Alert, create an alert destination.
      1) In the navigator, right-click PaymentValidation and select **Create** > **Alert Destination.**
      2) In the Create Alert Destination dialog, enter `AlertDestination` for Resource Name.
      3) Click **Create**.

4) On the Alert Destination Definition page, make sure Alert Logging is selected and deselect the SNMP Trap option.



5) Click **Save**.

You see that the file is added to the project folder.



d. Select PaymentValidationBS in the project folder.

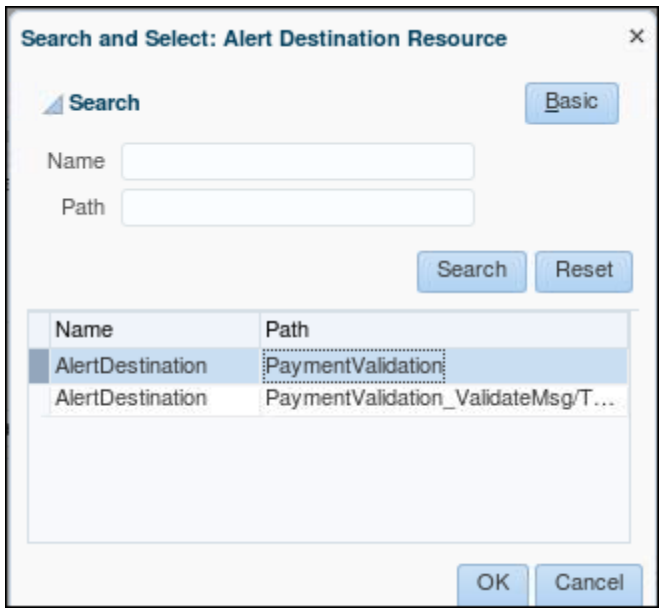
e. Click the **SLA Alert Rules** tab in the Business Service Definition page.

f. Under the Summary of SLA Alert Rules, click the add button to add a rule.

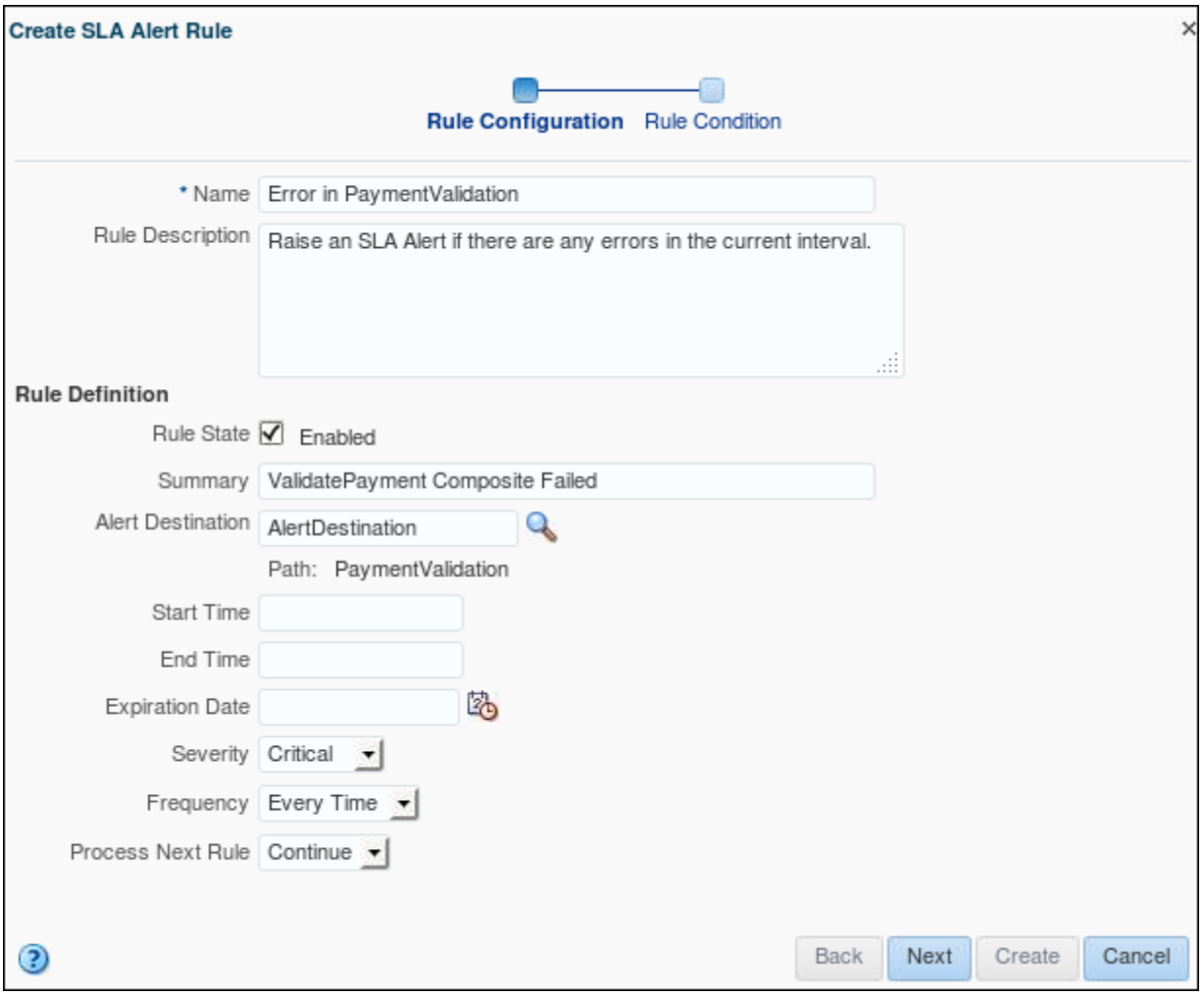
g. Add a rule with a Name, Description, and Summary:
   - Name: **`Error in PaymentValidation`**
   - Description: **`Raise an SLA Alert if there are any errors in the current interval`**.
   - Summary: **`ValidatePayment Composite Failed`**

h. Click Search next to the Alert Destination field.
   The Search and Select dialog box appears.

i. Simply click Search, and select the destination that you just created.



j. Set the severity level to Critical.
Now your rule configuration should look like this:

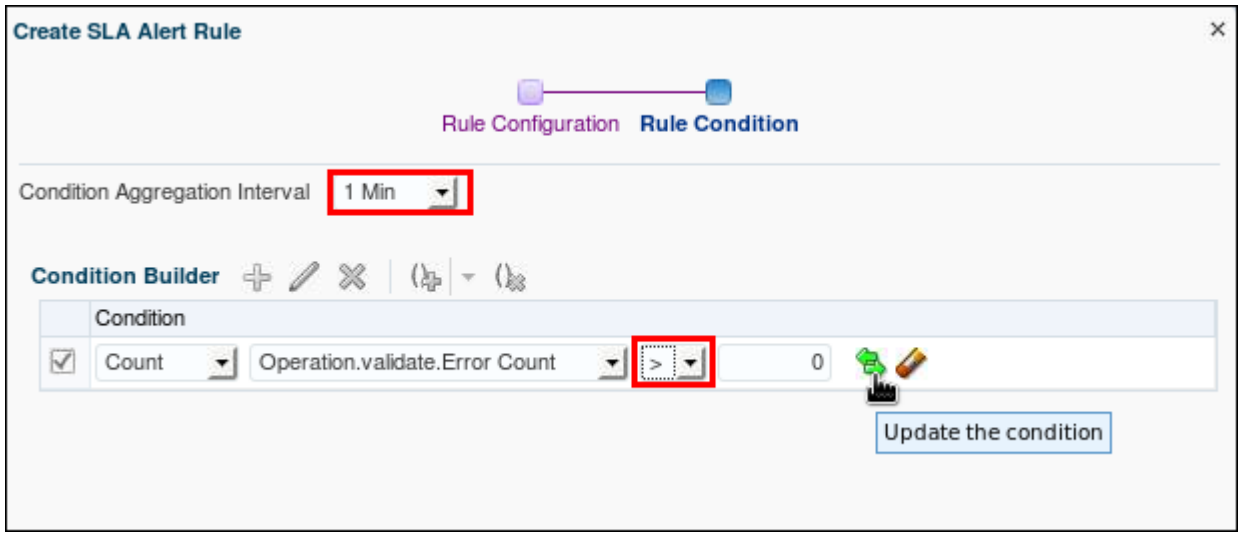


k. Click Next.

l. On the Rule Condition page, set the condition aggregation interval to 1 minute.

m. Click the Add button to add a condition.

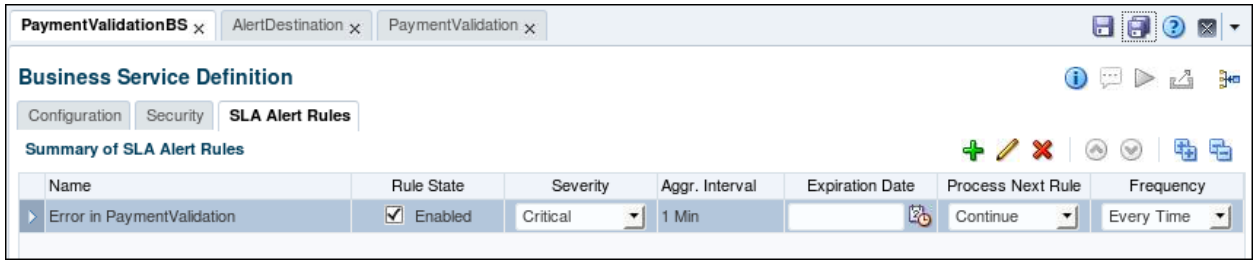
n. Set the Condition to Error Count > 0



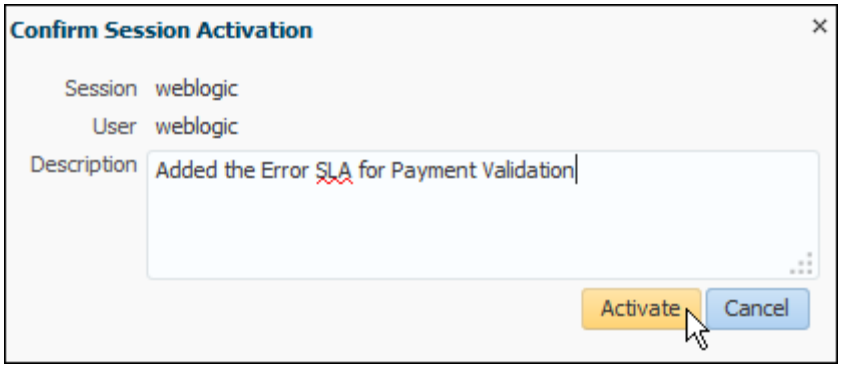
o. Click the update rule condition button.

p. With the rule updated, click the **Create** button.

q. Save your work.



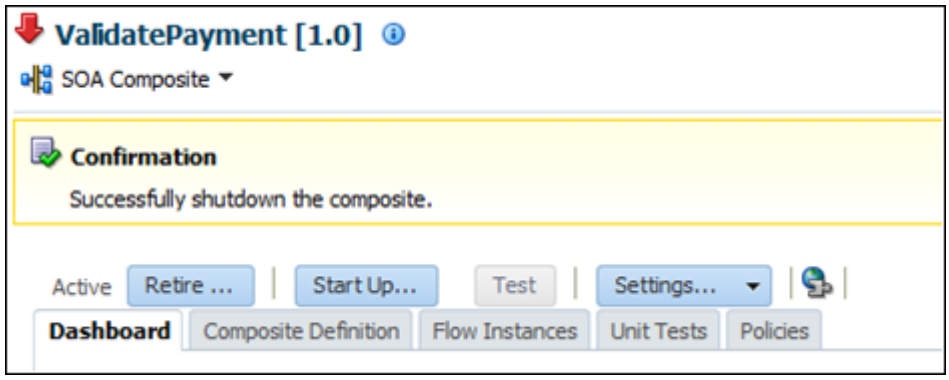
3. Activate the session and include a description.



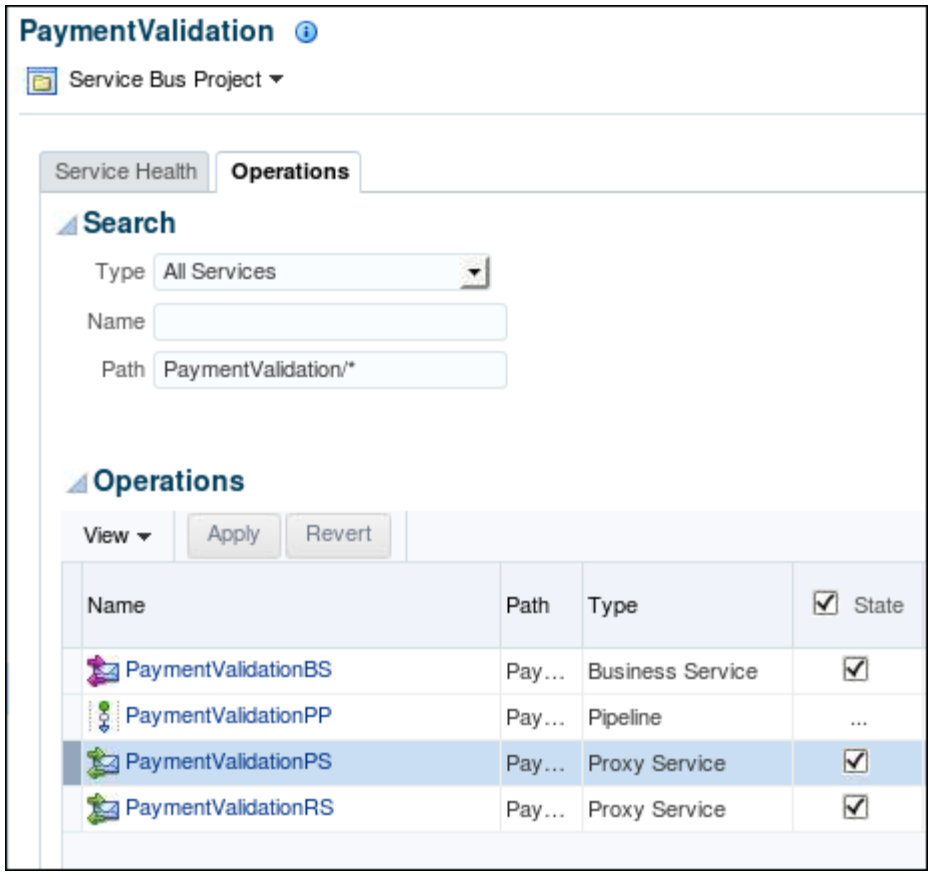
4. Now return to Enterprise Manager to test the SLA Alert. To set off an alert, you need to shutdown the ValidatePayment composite.

a. Select the composite in Enterprise Manager under SOA->soa-infra->default->ValidatePayment

b. In the composite Dashboard home page, click **Shut Down** and confirm. Make sure you see the following:



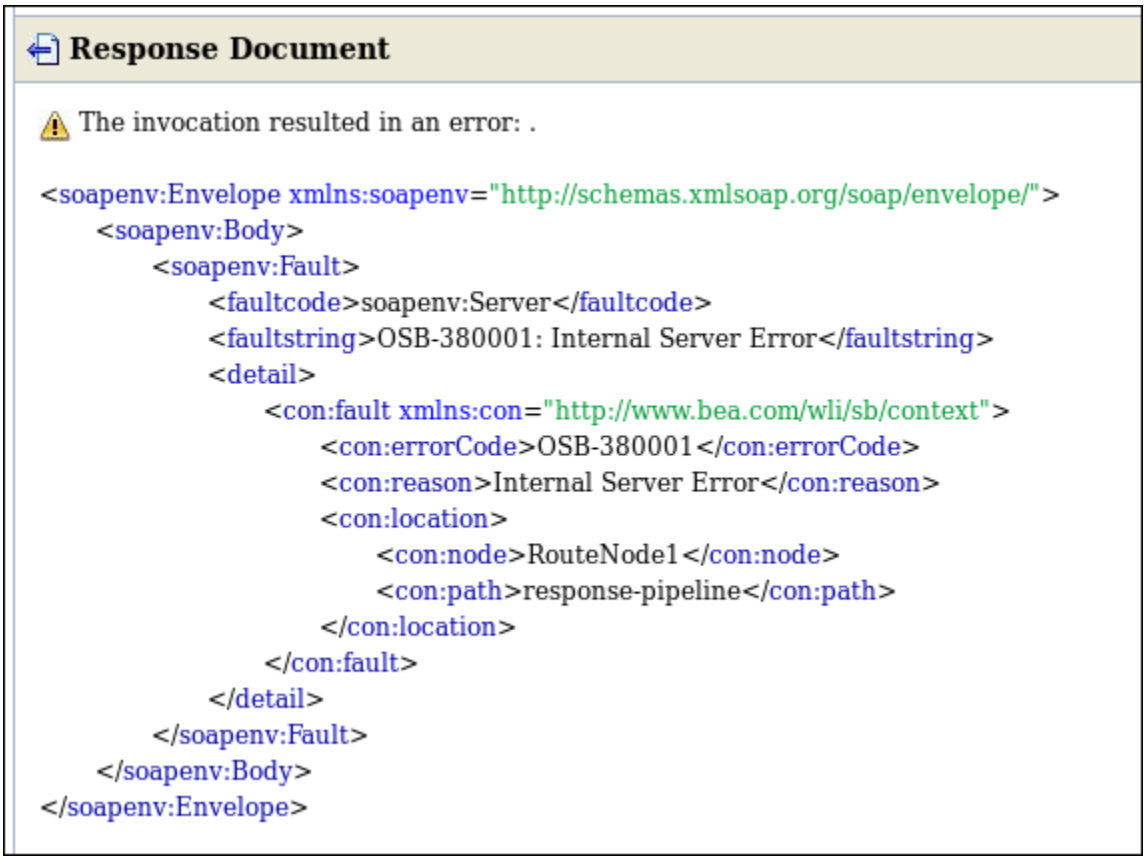
5. Test the PaymentValidationRS proxy service.
   a. Select the **PaymentValidation** project under SOA->service-bus.
   b. Click the **Operations** tab and select the **PaymentValidationPS** proxy service:



   The PaymentValidationPS proxy service page is displayed.
   c. Click **Test**.
   d. In the Test Console, select **~/labs_DI/resources/sample_input/PaymentInfoSample_Authorized.xml** for the request payload.

e. Click **Execute**.

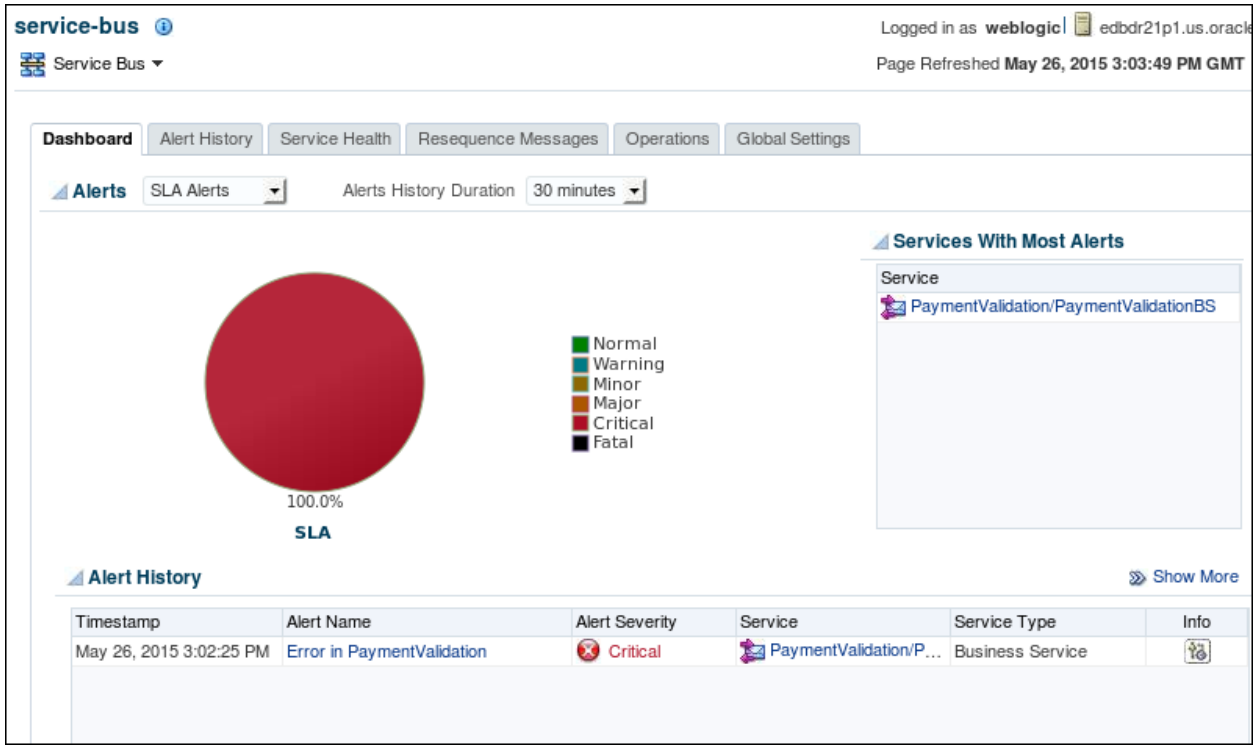
You should receive an invocation error because the composite is down. This is expected.



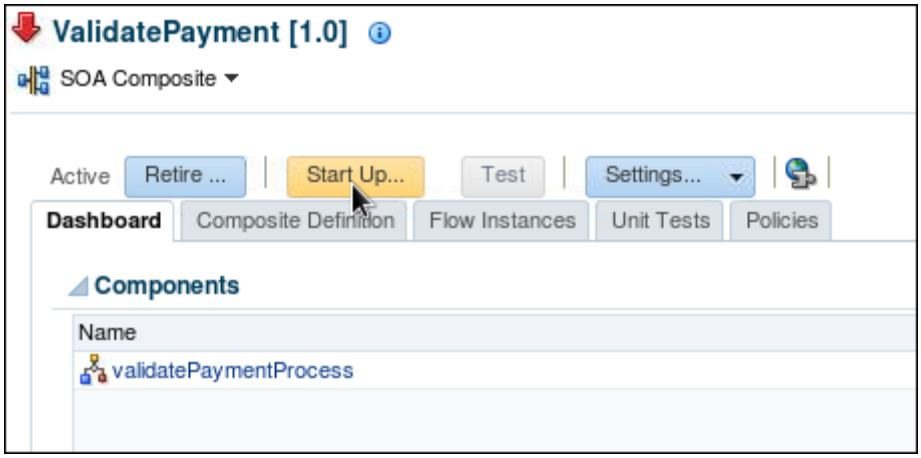
f. Back in EM console, click the SOA ->service bus (DefaultServer) folder.

In the service-bus home page, make sure that **SLA Alerts** is selected. At first, you may not see any results because the interval has not passed and the stats are not collected yet. This is normal. Just click the refresh button on the upper right-hand side of the dashboard every 30 seconds or so.

Once the stats have completed the refresh task, you should see results appear as shown below:



6. Once you finish the test, start the ValidatePayment composite.



Practice 13-3: Building and Deploying Service Bus Projects with Maven (Optional)

In this practice, you build your service bus projects with Maven. With 12.1.3 release of Oracle Service Bus and Oracle SOA Suite, you can do it natively without calling a utility like configjar or ANT from Maven.

Tasks

Install Maven sync plugin and populate the local repositories.

1. In a terminal window, enter the following commands:
   a. List the files and directories of .m2 folder:
      ```
      >cd /home/oracle/.m2
      >ls –l
      ```
      You should see a folder named repository. This is your Maven local repository.
   b. Make a back up of this folder:
      ```
      >cp –r repository repository_bak
      ```
   c. Extract the archived repository file:
      ```
      >tar xzvf /home/oracle/labs_DI/lessons/lesson13/m2-repository-thin.tgz
      ```
      Now your local repository contains all the library files (dependencies, plug-ins, etc.) that are needed to install and configure Oracle Service Bus Maven plug-In. These files are pre-downloaded from Maven's central repository. You can compare the current repository with the one that you just backed up to see the differences.
2. Set up environment variables using the following commands:
   ```
   >export M2_HOME=$FMW_DEV_HOME/oracle_common/modules/org.apache.maven_3.0.5
   >export PATH=$M2_HOME/bin:$PATH
   ```
3. Check if maven works:
   ```
   >mvn –v
   ```
   You should see an output similar to the following:
   ```
   Apache Maven 3.0.5 (r01de14724cdef164cd33c7c8c2fe155faf9602da; 2013-02-19 13:51:28+0000)
   Maven home: /u01/app/oracle/fmw_dev/12.1.3.0.0/oracle_common/modules/org.apache.maven_3.0.5
   Java version: 1.7.0_71, vendor: Oracle Corporation
   Java home: /usr/java/jdk1.7.0_71/jre
   Default locale: en_US, platform encoding: UTF-8
   OS name: "linux", version: "3.8.13-16.2.1.el6uek.x86_64", arch: "amd64", family: "unix"
   ```

4. Install the Maven sync plugin to the local repository:
   a. Go to the folder where the plug-ins are located:

      ```
      >cd
      $FMW_DEV_HOME/oracle_common/plugins/maven/com/oracle/maven/oracl
      e-maven-sync/12.1.3
      ```

      ```
      >ls -l
      ```

      You should see two files: oracle-maven-sync-12.1.3.jar and oracle-maven-sync-12.1.3.pom listed.

      Note: The Maven commands used in the subsequent steps can be found in the ~/labs_DI/lessons/lesson13/MavenCommands.txt file.

   b. Install oracle-maven-sync-12.1.3.jar and oracle-maven-sync-12.1.3.pom files to your local repository:

      ```
      >mvn install:install-file -DpomFile=oracle-maven-sync-12.1.3.pom
      -Dfile=oracle-maven-sync-12.1.3.jar -DoracleHome=$FMW_DEV_HOME
      ```

      You should see BUILD SUCCESS in the output message.
5. Deploy all the libraries to the local mvn repository:

   ```
   >mvn com.oracle.maven:oracle-maven-sync:push -
   DoracleHome=$FMW_DEV_HOME
   ```
6. Validate:

   ```
   >mvn help:describe -DgroupId=com.oracle.servicebus.plugin -
   DartifactId=oracle-servicebus-plugin -Dversion=12.1.3-0-0
   ```

   You should see Service Bus plug-in has 2 goals: deploy and package.

   Tip: If you want to learn the details of each goal, use Maven's Online Help. For example, the command for `package` details:

   ```
   >mvn help:describe -Ddetail=true -
   Dcmd=com.oracle.servicebus.plugin:oracle-servicebus-plugin:package
   ```
7. Update local archetype catalog:

   ```
   >mvn archetype:crawl -Dcatalog=$HOME/.m2/archetype-catalog.xml
   ```

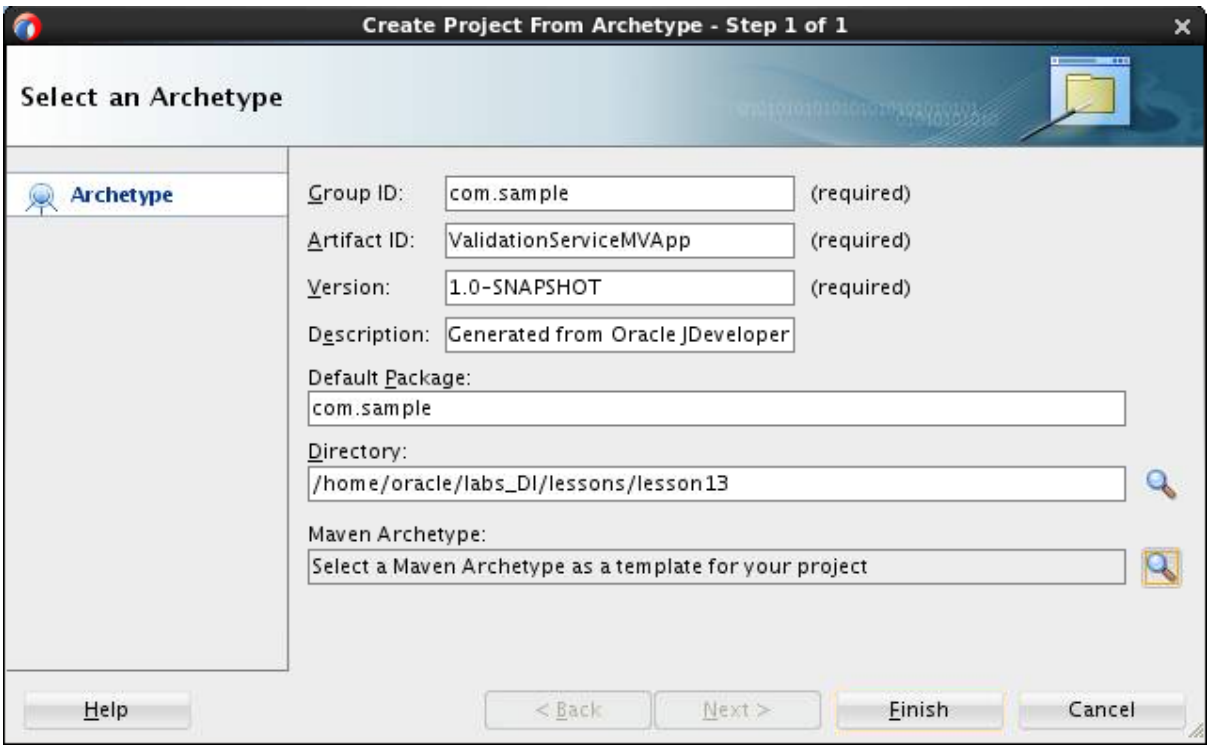
   Besides the Build Success message, you should also see the archetype-catalog.xml file created in the $HOME/.m2 folder.

   Now you are able to create a new Service Bus project from a Maven archetype.

Create a new Service Bus project with Maven

8. Create a new Service Bus application.
   a. Go back to JDeveloper. Select **File > New > From Gallery** from the menu.
      The New Gallery dialog box opens.
   b. Under the General category, select **Maven**.
   c. Under Items, select **Generate from Archetype**.
   d. Click **OK**.
      The Create Project from Archetype wizard opens.
   e. Provide all the maven details:
      - Group ID: **`com.sample`**
      - Artifact ID: **`ValidationServiceMVApp`**
      - version: (use the default value)

- Default Package: **`com.sample`**
- Directory: **`/home/oracle/labs_DI/lessons/lesson13`**



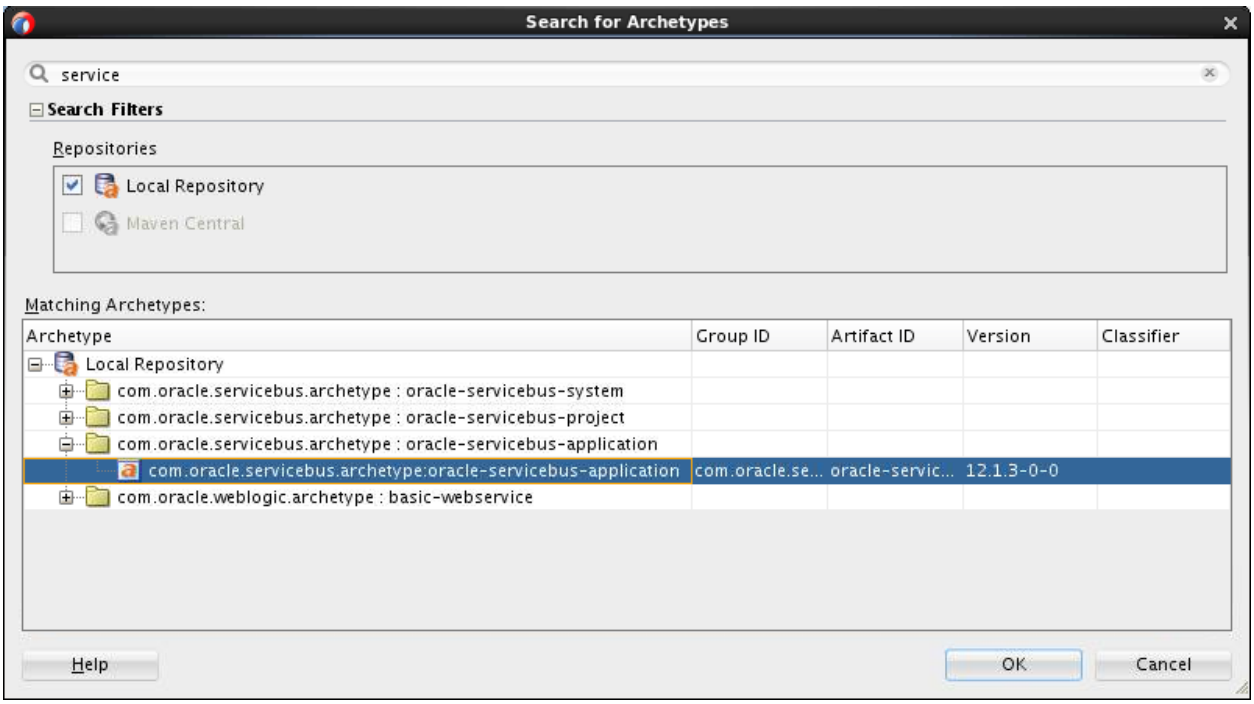
f. Click the Browse button next to the Maven Archetype field.

   The Search for Archetype dialog box opens. Because there is no internet connection, you can only use local repository.

g. Type **`service`** in the search field.

   The Matching Archetypes panel is updated with the search result.

h. Expand the Local Repository folder and select **com.oracle.servicebus.archetype : oracle-servicebus-application** as shown below:



i. Click **OK**.

Back in the Create Project from Archetype wizard, the Maven Archetype field is populated with the archetype you selected.

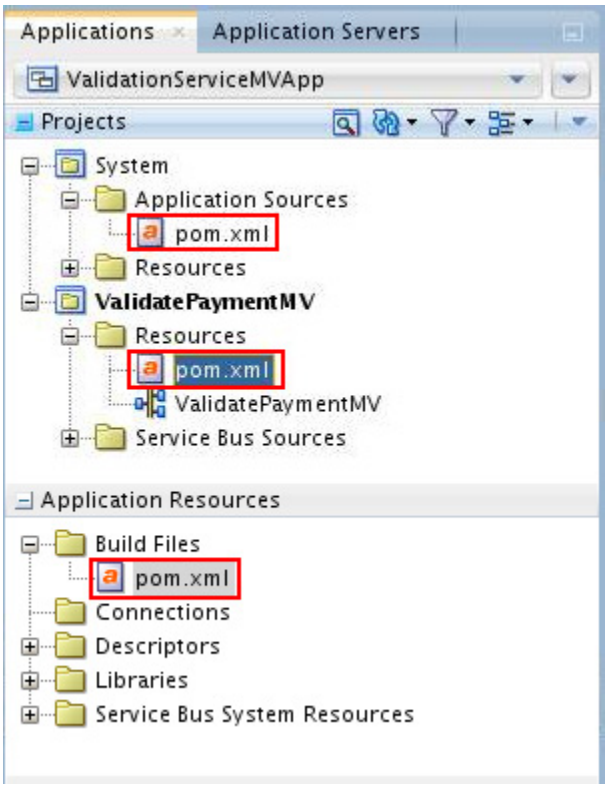
j. Click **Next**.

k. Provide the required projectName of this archetype: **ValidatePaymentMV**.

The Create Application dialog box is displayed.

l. Accept the default values for all the fields, and click OK.

Now you got a Service Bus application with 3 poms: one project pom with sbar as the package type, System project pom, and a parent (application) pom with these 2 projects.

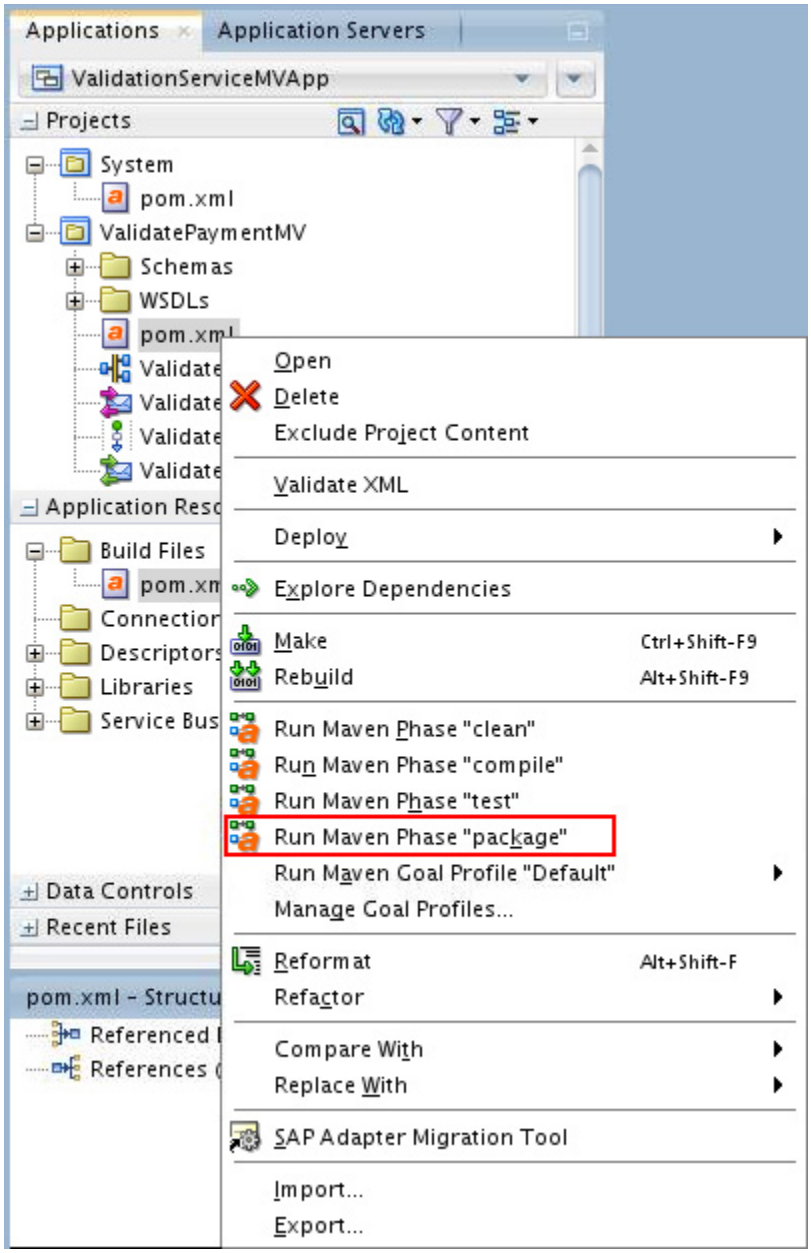


You can open each pom file to view its details.

9. Now you can build the ValidatePaymentMV Service Bus project (under the ValidatePaymentMV folder). The high level steps are:
   a. Create folders and import artifacts: `ValidatePayment-concrete.wsdl` and `CanonicalOrder.xsd`.
   b. Create the business service.
   c. Create the simple pass-through proxy service.

   Note: If you still need detailed instructions, please refer to Practice 3-2: Step 4 through Step 20.

10. After you are done, right-click the pom.xml file in the PaymentValidationMV folder to run the package phase:



You see that the project is packaged using Maven plug-in, and observe a Build Success message at the end.

```
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building ccvalidate 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- oracle-servicebus-plugin:12.1.3-0-0:package (default-package) @ ccvalidate ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3:16.879s
[INFO] Finished at: Sun Apr 26 18:41:01 GMT 2015
[INFO] Final Memory: 13M/216M
[INFO] ------------------------------------------------------------------------
Process exited with exit code 0.
```

The Maven Package phase creates a SBAR file, **sbconfig.sbar** from your project, and places it in the following directory:
`<project_folder>/.data/maven/`

In your case, it is:

`~/labs_DI/lessons/lesson03/ValidationServiceMVApp/ValidatePaymentMV/.data/maven/`

11. When you are done, close the application and all open windows of this project.