SAMPLE NAME : bpel-105-Headers

COMPONENT : BPEL

OVERVIEW

This sample contains instructions for how to create a BPEL process that can set standard header information,and create customer headers. It also shows how a BPEL process can retrieve both types of headers (standard and customized) for processing.

PROVIDED FILES

ReadMe.txt – A basic readme file that covers the general aspects of the sample.

Sample_Instructions.pdf – The file you are now reading. This is the main source of the information in this tutorial.

composite.xml – An SCA file that descries the composite application and its major components.

DetailHeaderReceivingProcess.bpel – This BPEL process shows how to read both standard and custom headers.

DetailHeaderReceivingProcess.componentType – A utility file generated by JDeveloper There is no need to modify this file directly.

DetailHeaderReceivingProcess.wsdl – A WSDL file created by JDeveloper when it created the associated BPEL process. You rarely need to modify WSDLs directly when using JDeveloper and SCA.

MasterHeaderProcess.bpel – This BPEL process shows how too set standard and customer header information.

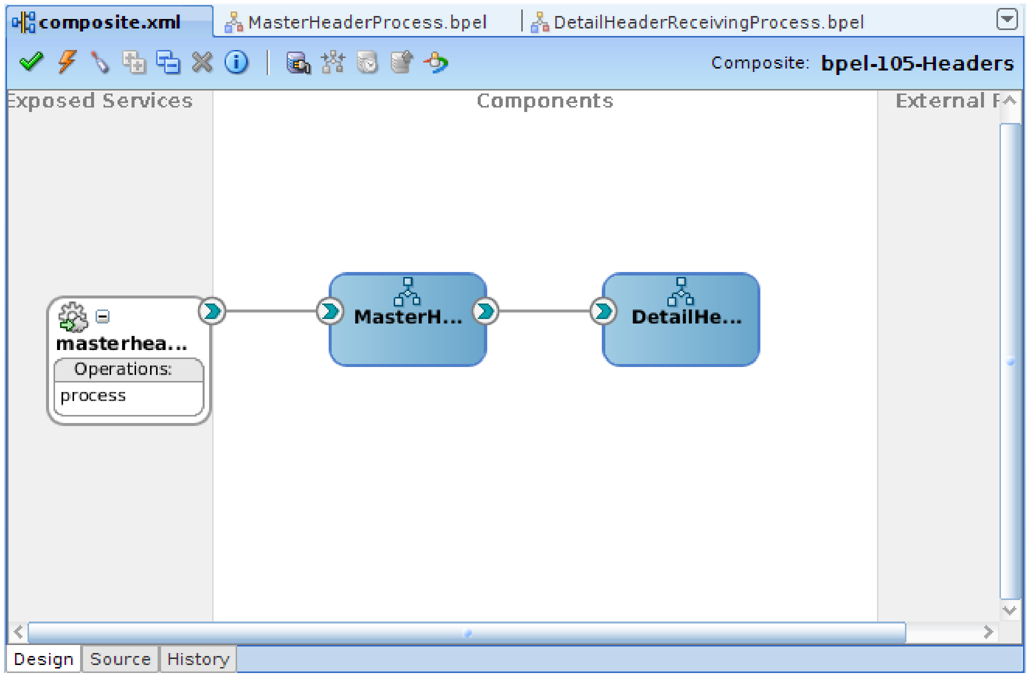
MasterHeaderProcess.componentType – A utility file generated by JDeveloper There is no need to modify this file directly.

MasterHeaderProcess.wsdl – A WSDL file created by JDeveloper when it created the associated BPEL process. You rarely need to modify WSDLs directly when using JDeveloper and SCA.

xsd/MasterHeaderProcess.xsd – The primary schema used for defining our data types.

SAMPLE OVERVIEW

Lets take a look at how this sample works. Open the application in JDeveloper by using Application → Open from the file menu and selecting the bpel-105-Headers.jws file in the project directory. Once the application loads into JDeveloper, open the composite.xml file. This file will give us an overview of the major pieces of the sample. Your composite.xml file should look like the following:



The three major pieces are:

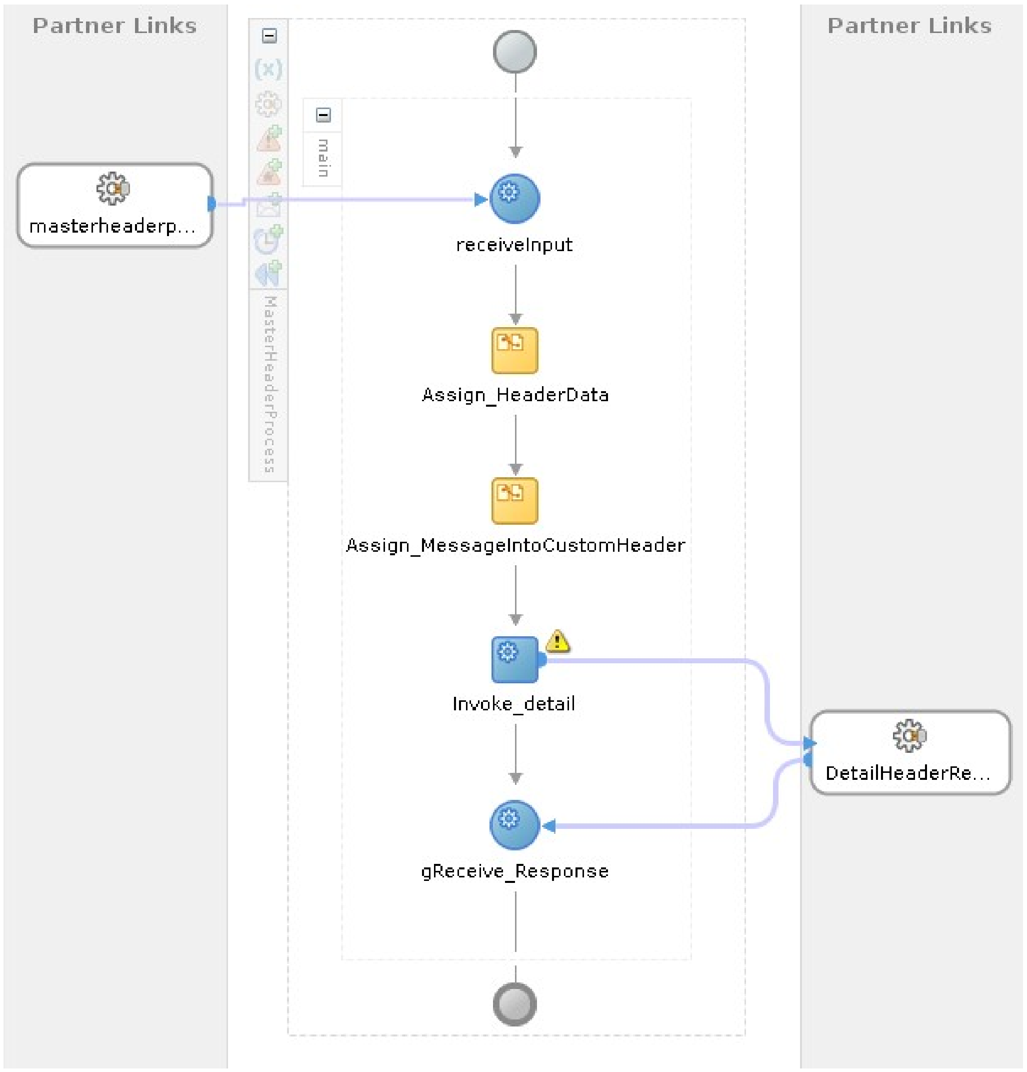
1) The masterheaderprocess_client_ep in the Exposed Services swim-lane. This is the interface that will be invoked.
2) MasterHeaderProcess.bpel. This is the BPEL process that demonstrates how to create custom and standard header values.
3) DetailheaderReceivingProcess.bpel – This is the BPEL process that shows how to read both standard and custom headers

The masterheaderprocess_client_ep was created by dragging and dropping the Web Service service adapter onto the Exposed Services swim-lane. It is based on the MasterHeaderProcess.wsdl file. The WSDL file is fairly striaght-forward. The more interesting file is the MasterHeaderProcess.xsd, located in the xsd folder of the project. This file defines the data-types for the WSDL. More importantly, it shows how you define headers in an XML Schema file. Takae special note of the “cuustomHeader” and “infraHeader” element definitions in the file (shown below):

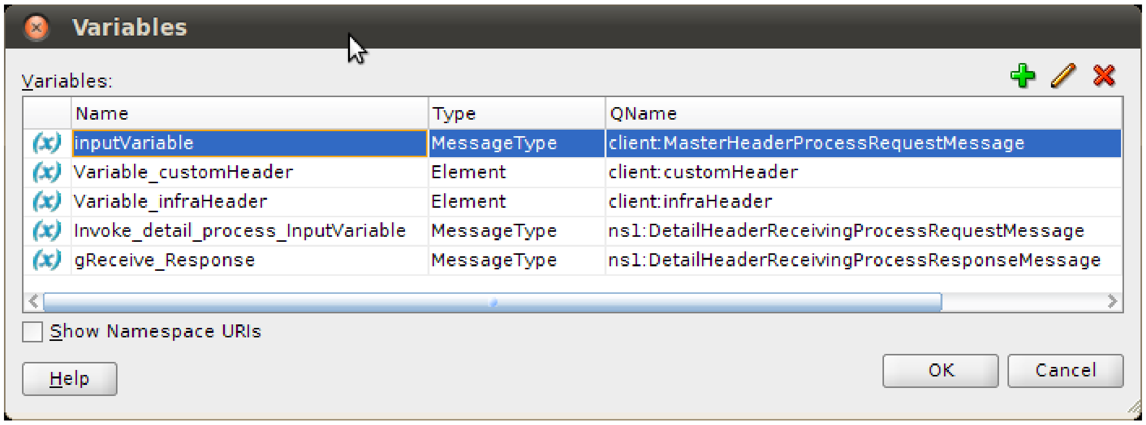
```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="unqualified"
   elementFormDefault="qualified"
   targetNamespace="http://example.com/bpel_105_Headers/MasterHeaderProcess"
   xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="process">
      <complexType>
         <sequence>
            <element name="input" type="string"/>
         </sequence>
```

```
        </complexType>
     </element>
     <element name="customHeader">
        <complexType>
           <sequence>
              <element name="message" type="string"/>
           </sequence>
        </complexType>
     </element>
     <element name="infraHeader">
        <complexType>
           <sequence>
              <element name="origin" type="string"/>
              <element name="sendDate" type="string"/>
              <element name="instanceId" type="string"/>
           </sequence>
        </complexType>
     </element>
     <element name="processResponse">
        <complexType>
           <sequence>
              <element name="result" type="string"/>
           </sequence>
        </complexType>
     </element>
  </schema>
```

Things get more interesting with the MasterHeaderProcess.bpel component. If you double-click on this component you will see the following activities in the process:



There are several points of interest in this BPEL process. The first is the definitions of the variables for this process. Take a look at the following screen-shot to see the variable that are defined.



Variables:

Name	Type	QName
inputVariable	MessageType	client:MasterHeaderProcessRequestMessage
Variable_customHeader	Element	client:customHeader
Variable_infraHeader	Element	client:infraHeader
Invoke_detail_process_InputVariable	MessageType	ns1:DetailHeaderReceivingProcessRequestMessage
gReceive_Response	MessageType	ns1:DetailHeaderReceivingProcessResponseMessage

The input variable is simply the incoming message as defined by our SOAP service. The `Variable_customHeader` and `Variable_infraHeader` correspond to the similarly named elements that we saw in the XSD.

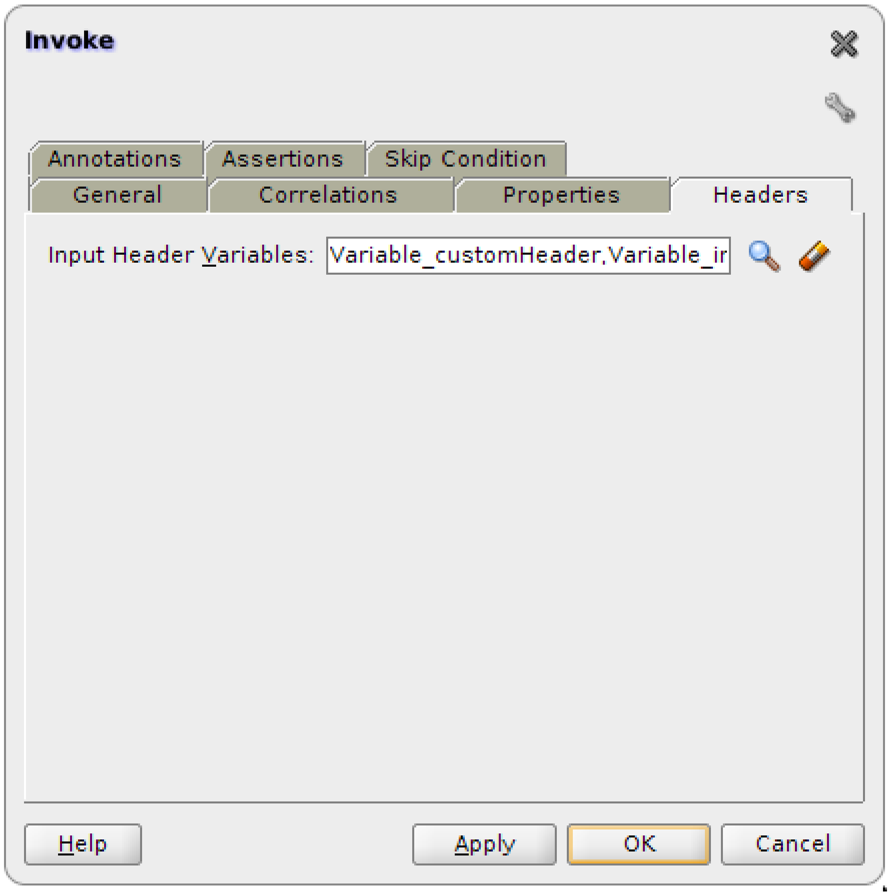
If you double-click on the `Assign_HeaderData` activity you will see that it contains a total of 3 Copy operations (see next image) that are used to assign values into 3 standard headers. Here is a summary of the 3 copy statements in this Assign activity:

From	To
Expression: ora:getComponentName()	**Variable**: /client:infraHeader/client:origin
Expression: ora:getCompositeInstanceId()	**Variable**: /client:infraHeader/client:instanceId
Expression: xpath20:current-dateTime()	**Variable**: /client:infraHeader/client:sendDate

Similarly the `Assign_MessageIntoCustomHeader` contains a single Copy operation that copes the incoming `input` message into the `message` element of the parent `customHeader` element.

From	To
Variable: /client:process/client:input	**Variable**: /client:customHeader/client:message

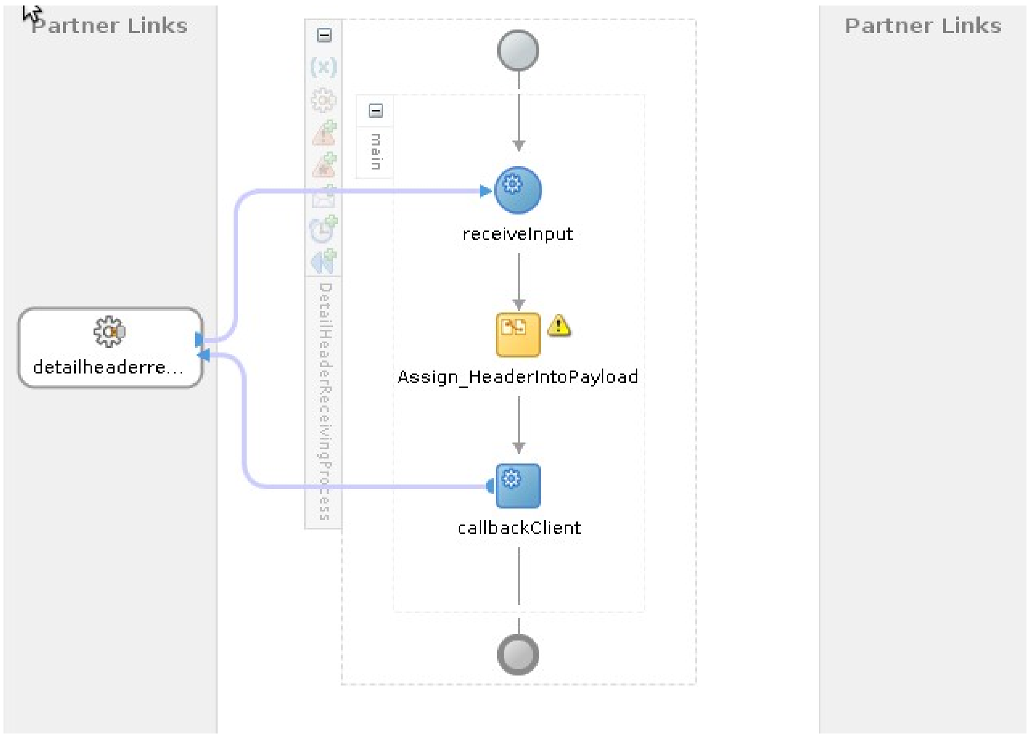
Right now we have defined our two header variables, `infraHeader` and `customHeader`. However, they are still just variables. We need to insert them into the Header list of the `Invoke_detail` activity. Double click on the `Invoke_detail` activity and then select the Headers tab to see how this is done. You will see the following:



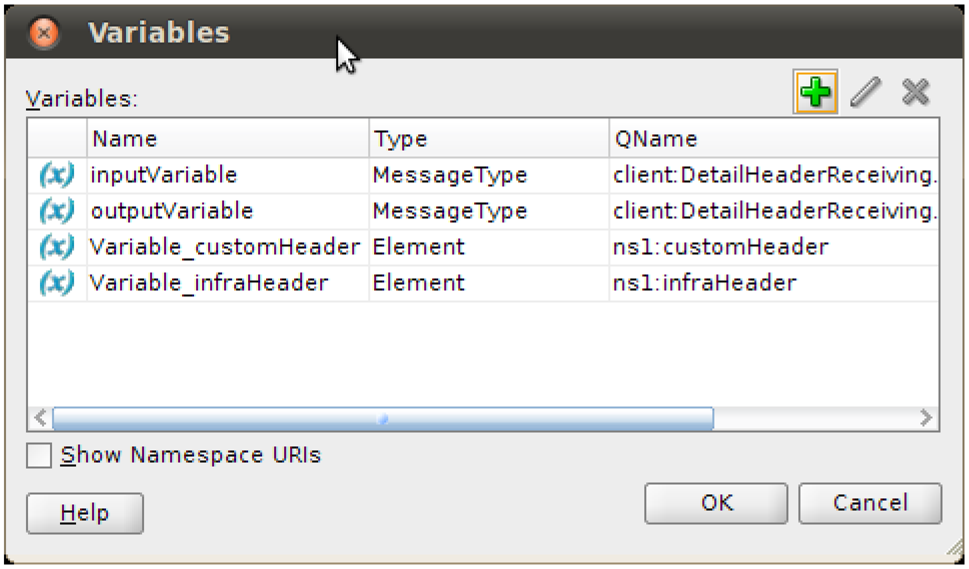
Input Header Variables: Variable_customHeader,Variable_ir

To add the headers to an invocation you simply add them in a comma-delimited list to the Input Header Variables field. It's just that simple. Now that the headers are set, we can invoke the `DisplayHeaderReceivingProcess`.

Close the `MasterHeaderProcess` in JDeveloper and double-click on the `DisplayHeaderReceivingProcess` in the composite.xml file. It will appear as follows:



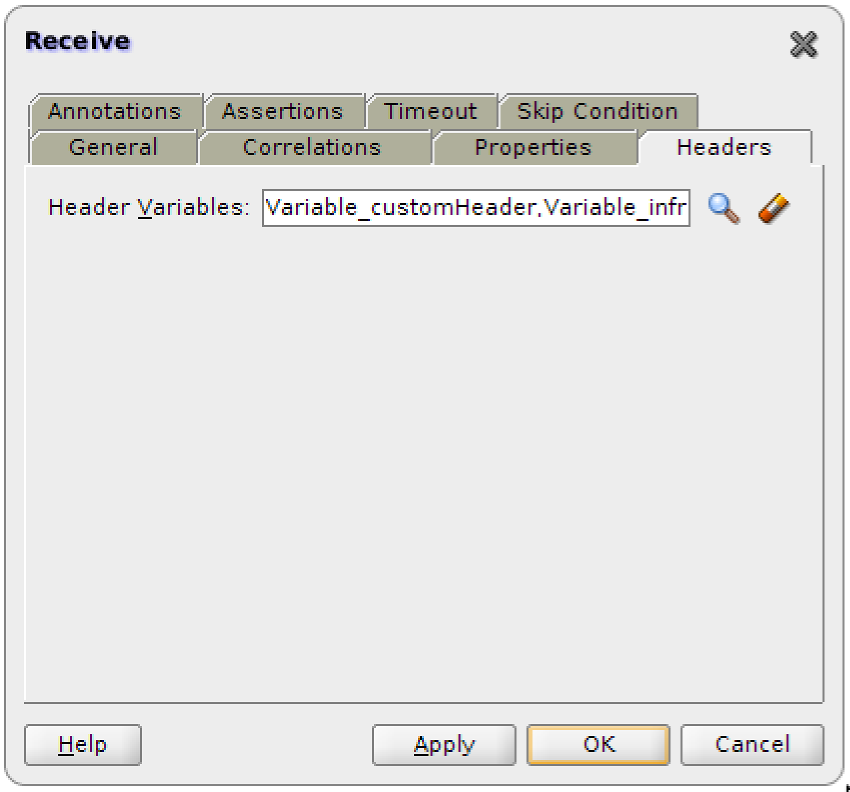
The BPEL process defines all of its local variables, including those for the headers, as shown next:



	Name	Type	QName
(x)	inputVariable	MessageType	client:DetailHeaderReceiving.
(x)	outputVariable	MessageType	client:DetailHeaderReceiving.
(x)	Variable_customHeader	Element	ns1:customHeader
(x)	Variable_infraHeader	Element	ns1:infraHeader

Next we need to inform the `receiveInput` node about the existence of these header variables so that it can retrieve them. The nice part is that when it does retrieve them, it automatically associates the data with our process variables so that we can use them without having to first copy them into local variables.

Double-click on the `receiveInput` node and then select the Headers tab. You will see that defining the input headers follows the same process as before.



Header Variables: Variable_customHeader,Variable_infr

Close the `receiveInput` dialog and then double-click on the `Assign_HeaderIntoPayload` activity. This activity contains a single Copy operation. Take a look at the expression in the following table and you'll see how the header information is accessed and used to create the response.

From	To
Expression: concat('origin: ', bpws:getVariableData('Variable_infraHeader','/ns1:infraHeader/ns1:origin'), ' message: ', bpws:getVariableData('Variable_customHeader','/ns1:customHeader/ns1:message'))	**Variable**: /ns1:processResponse/ns1:result

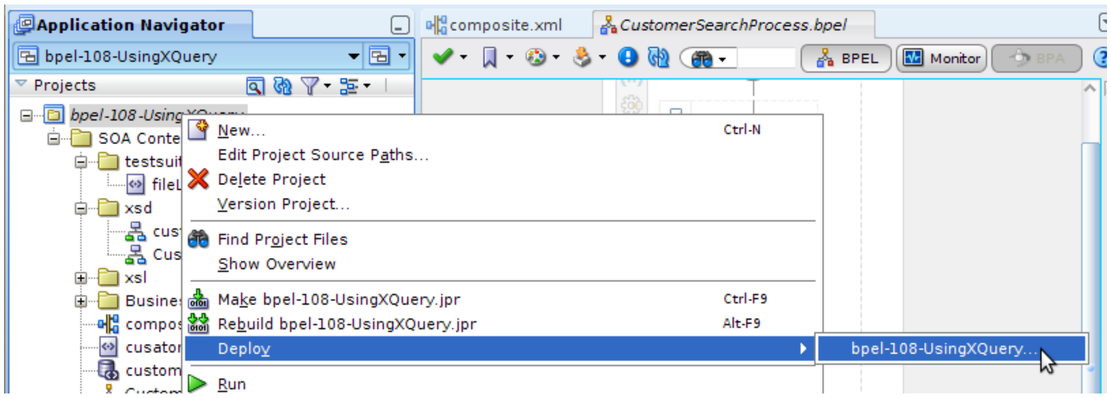
That's all there is to it.

Deploying the BPEL Process

Deploying any SCA composite application follows the same steps from JDeveloper. While the name of the project in the screen-shots will differ from the name if this particular project, the process is identical.

1. You must deploy the BPEL process to a running server before you can use it, even for testing. After you start your servers, you can deploy the project by right-

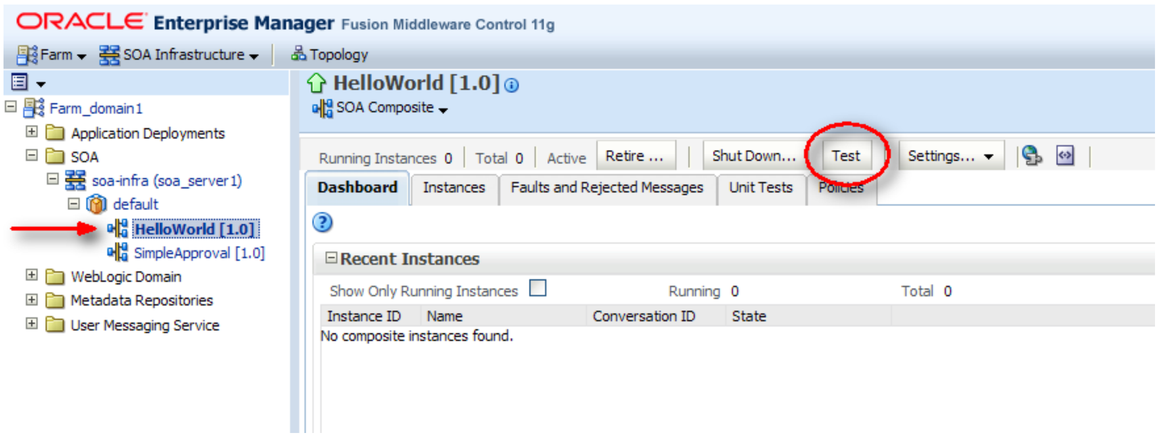
clicking on the project and selecting Deploy → bpel-105-Headers from the popup menu.



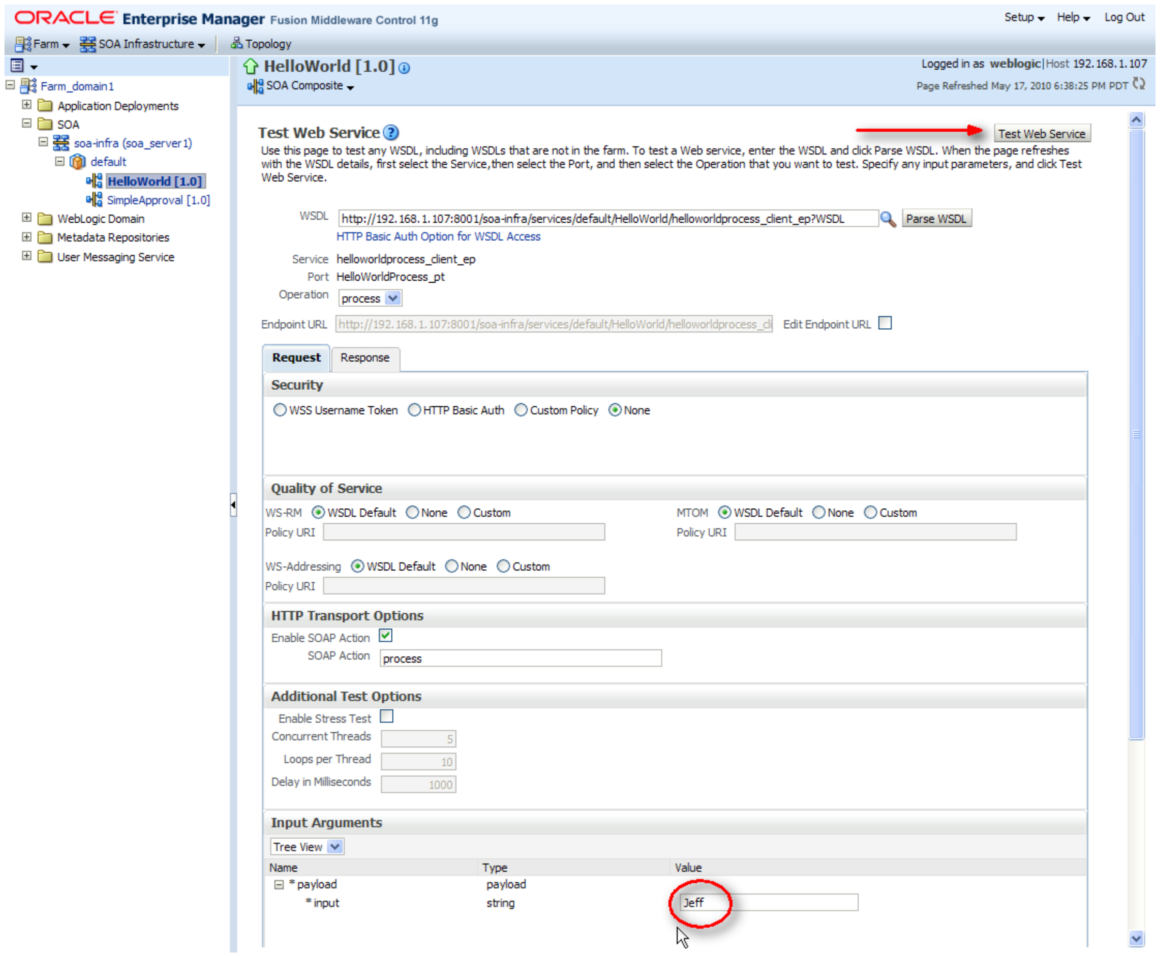
2. Follow the instructions in the wizard to deploy the application. You must already have an application server connection defined in JDeveloper before you can deploy the project successfully.

Testing the BPEL Process

To test your BPEL process you need to use the Oracle Enterprise Manager (EM) web interface. Open your browser to http://localhhost:7001/em and login as the admin user. Using the navigation bar on the left side of the EM interface, navigate to the project and select it. When the page loads, entey any informartion that is required in the Request tab and thhn press the Test button.



Be sure too enter your name in the *input* field at the bottom of the test page. Then press the *Test Web Service* button.



ORACLE Enterprise Manager Fusion Middleware Control 11g — Setup | Help | Log Out

Farm | SOA Infrastructure | Topology

- Farm_domain1
  - Application Deployments
  - SOA
    - soa-infra (soa_server1)
      - default
        - HelloWorld [1.0]
        - SimpleApproval [1.0]
  - WebLogic Domain
  - Metadata Repositories
  - User Messaging Service

HelloWorld [1.0]
SOA Composite

Logged in as weblogic | Host 192.168.1.107
Page Refreshed May 17, 2010 6:38:25 PM PDT

Test Web Service — Test Web Service

Use this page to test any WSDL, including WSDLs that are not in the farm. To test a Web service, enter the WSDL and click Parse WSDL. When the page refreshes with the WSDL details, first select the Service, then select the Port, and then select the Operation that you want to test. Specify any input parameters, and click Test Web Service.

WSDL	http://192.168.1.107:8001/soa-infra/services/default/HelloWorld/helloworldprocess_client_ep?WSDL — Parse WSDL
	HTTP Basic Auth Option for WSDL Access
Service	helloworldprocess_client_ep
Port	HelloWorldProcess_pt
Operation	process
Endpoint URL	http://192.168.1.107:8001/soa-infra/services/default/HelloWorld/helloworldprocess_cl — Edit Endpoint URL ☐

Request | Response

Security

WSS Username Token | HTTP Basic Auth | Custom Policy | [✓ None]

Quality of Service

WS-RM [✓ WSDL Default] None Custom — Policy URI
MTOM [✓ WSDL Default] None Custom — Policy URI
WS-Addressing [✓ WSDL Default] None Custom — Policy URI

HTTP Transport Options

Enable SOAP Action ✓
SOAP Action: process

Additional Test Options

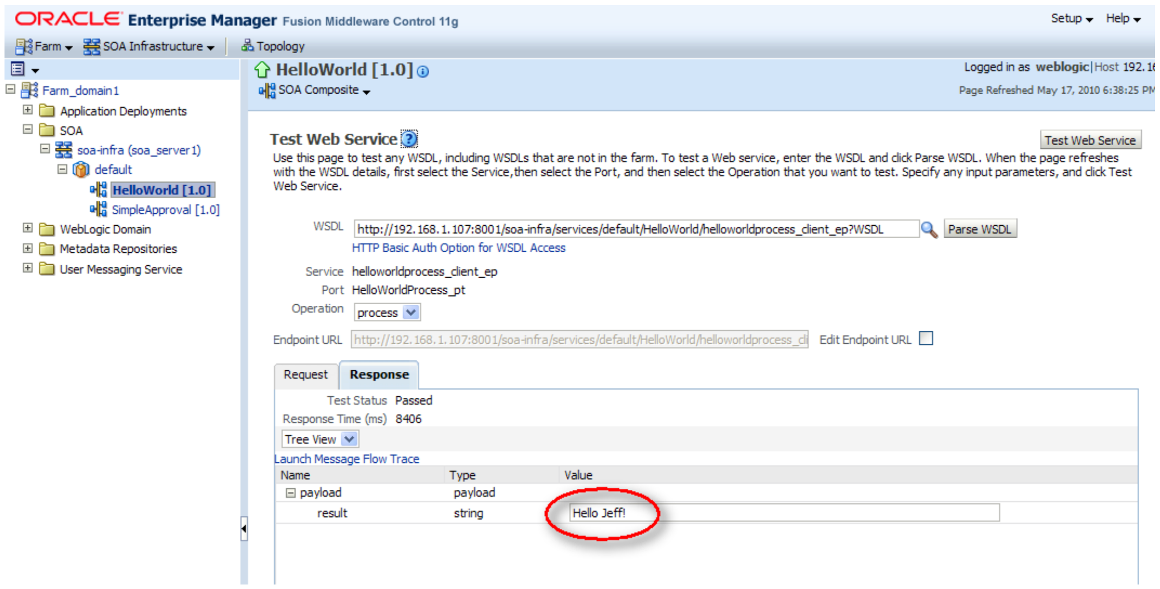
Enable Stress Test	☐
Concurrent Threads	5
Loops per Thread	10
Delay in Milliseconds	1000

Input Arguments

Tree View

Name	Type	Value
* payload	payload	
* input	string	Jeff

The Response tab will automatically be selected when the web service call returns. Examine the result string at the bottom of the page to see the response message from the web service.



ORACLE Enterprise Manager Fusion Middleware Control 11g — Setup | Help

Farm | SOA Infrastructure | Topology

- Farm_domain1
  - Application Deployments
  - SOA
    - soa-infra (soa_server1)
      - default
        - HelloWorld [1.0]
        - SimpleApproval [1.0]
  - WebLogic Domain
  - Metadata Repositories
  - User Messaging Service

HelloWorld [1.0]
SOA Composite

Logged in as weblogic | Host 192.1
Page Refreshed May 17, 2010 6:38:25 PM

Test Web Service — Test Web Service

Use this page to test any WSDL, including WSDLs that are not in the farm. To test a Web service, enter the WSDL and click Parse WSDL. When the page refreshes with the WSDL details, first select the Service, then select the Port, and then select the Operation that you want to test. Specify any input parameters, and click Test Web Service.

WSDL	http://192.168.1.107:8001/soa-infra/services/default/HelloWorld/helloworldprocess_client_ep?WSDL — Parse WSDL
	HTTP Basic Auth Option for WSDL Access
Service	helloworldprocess_client_ep
Port	HelloWorldProcess_pt
Operation	process
Endpoint URL	http://192.168.1.107:8001/soa-infra/services/default/HelloWorld/helloworldprocess_cl — Edit Endpoint URL ☐

Request | **Response**

Test Status	Passed
Response Time (ms)	8406

Tree View

Launch Message Flow Trace

Name	Type	Value
payload	payload	
result	string	Hello Jeff!

The first time you call the web service it may actually take a couple of seconds to respond. That is because the WebLogic Server that is hosting the web service needs to initialize a few things in the background on the very first call. Click on the *Request* tab in the EM Test Web Service page and enter a different name in the *input* field and run the test again. You will see the second time it responds instantly!

Importing the Sample Code

If you want to see the working code provided with this sample you will need to open the application in your JDeveloper environment.. First you need to unzip the *bpel-105-Headers.zip* file that you downloaded from OTN. Next , select Application → Open from the main menu in JDeveloper and navigate to and select the *bpel-105-Headers.jws* file.

Next Steps

.Check out the other samples on the OTN page for SOA Suite 11g. You'll find all of the samples located at `https://soasamples.samplecode.oracle.com/`

SUMMARY

In this sample we have covered the basics of handling headers in BPEL processes. Creating and using headers can be a great way to add information to existing messages without modifying the contents of those messages. Headers are also often used to pass security credentials around between systems. There are many different uses for headers and now you have the knowledge that you need to make them work for you.