

Chapter 5

WS Security in GlassFish ESB

Michael.Czapski@sun.com, Revision 0.2.3, September 19, 2009

In this document I explore the effects of selected web services security policies on SOAP message exchange in the GlassFish ESB v2.1.

To provide early access I intend to release revisions of this document as significant new sections become available.

This is a work-in-progress document.

Rev 0.1: Content

- Assumptions and Notes
- Person Service XML Schema and WSDL Interface
- Common XML Project
- PersonSvc BPEL Module
- PersonCli BPEL Module
- JBI-based Person Service – Plain End-to-End
- JBI-based Person Service – SSL with Server-side Authentication

Rev 0.2: Additional Content

- JBI-based Person Service – SSL with Mutual Authentication (broken)
- EJB-based Person Service – No security
- EJB-based Person Service – SSL with Server-side Authentication
-

5.1 Chapter Content

GlassFish ESB v2.1 uses the Metro stack for web services standards support.

This chapter explores selected methods of applying security to the channel over which SOAP messages are exchanged and the SOAP messages themselves, using a basic BPEL 2.0-based invoker and provider set.

A pair of projects, an invoker and a provider, are used to provide the logic. Composite Applications are used to apply different variants of security policies. There will be one pair of Composite Applications for each security policy to demonstrate that security is a non-functional requirement and to show how security policy and application logic can be separated such that change in one does not require change to the other.

The following security policies are explored:

- None

- HTTP BC Channel Security - SSL / TLS with Server-side Authentication
- HTTP BC Channel Security - SSL / TLS with Mutual Authentication
- EJB Channel Security - SSL / TLS with Server-side Authentication
-
- Message Encryption
-

For each variant an end-to-end solution will be built and exercised. Server.log traces from both sides will be inspected and discussed as necessary to clarify what is happening during the process.

5.2 Assumptions and Notes

To explore different options for securing web services in GlassFish ESB v2.1 it is necessary to obtain and install the GlassFish ESB v2.1 software. Distribution, as at September 2009, can be downloaded from <https://open-esb.dev.java.net/Downloads.html>. Since installation of GlassFish ESB is adequately documents, see <http://wiki.open-esb.java.net/Wiki.jsp?page=UsingTheGlassFishESBInstallationGUI>, installation instructions are not repeated here. It is assumed that the GlassFish ESB v2.1 installation exists and is functional. This also assumes that the NetBeans 6.5.1 IDE, distributed as part of GlassFish ESB, is installed and operational. Issues have been reported with different version of the JDK 6. GlassFish ESB installation used for examples developed in this chapter uses JDK 1.6.0_16.

Exploration of channel security, while possible with a single installation of GlassFish ESB, will be easier if two instances of GlassFish ESB, on two different hosts, are available. If not, it will be hard for the reader to follow SSL Handshake log messages and figure out which are produced by the invoker and which are produced by the provider. Examples in this chapter will use two separate hosts for these projects. Typically, the server-side implementation will be deployed to a host whose fully qualified domain name (FQDN) is orad1.ssc and client-side implementation will be deployed to a host whose FQDN is mcz02.aus.sun.com.

It is assumed that the NetBeans IDE and one instance of the GlassFish Application Server are co-hosted on the same machine. Each time I use the expression “the local instance of the GlassFish” I mean the instance of the GlassFish which is resident on the same host as the NetBeans IDE used for development. If this is not the environment you use adjust as required.

5.3 Person Service XML Schema and WSDL Interface

A basic web service provider, which we will develop and use to explore web services security, will accept a request with a person identifier and will return a small set of person details for the selected person. This is a request/reply service. Since data returned by the service is not of importance we will not bother with details such as searching a database for person details. We will simply hardcode the response.

The request and response messages will conform to the XML Schema shown in Listing 5.3.1.

Listing 5.3.1 *Person XML Schema*

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://xml.netbeans.org/schema/Person"
  xmlns:tns="http://xml.netbeans.org/schema/Person"
  elementFormDefault="qualified"
  >

  <xsd:element name="PersonReq">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="PersonID" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="PersonRes">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="PersonID" type="xsd:string"/>
        <xsd:element name="FamilyName" type="xsd:string"/>
        <xsd:element name="MiddleInitials"
          type="xsd:string" minOccurs="0"/>
        <xsd:element name="GivenName" type="xsd:string"/>
        <xsd:element name="Gender" type="xsd:string" minOccurs="0"/>
        <xsd:element name="AddressDetails" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="StreetAddress"
                type="xsd:string" minOccurs="0"/>
              <xsd:element name="CityTown"
                type="xsd:string" minOccurs="0"/>
              <xsd:element name="PostCode"
                type="xsd:string" minOccurs="0"/>
              <xsd:element name="StateProvince"
                type="xsd:string" minOccurs="0"/>
              <xsd:element name="Country"
                type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="CreditCardDetails" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="CardType"
                type="xsd:string" minOccurs="0"/>
              <xsd:element name="CardNumber"
                type="xsd:string" minOccurs="0"/>
              <xsd:element name="ExpiryDate"
                type="xsd:string" minOccurs="0"/>
              <xsd:element name="SecurityCode"
                type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```

    <xsd:element name="PersonFlt">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="PersonID" type="xsd:string"/>
          <xsd:element name="FaultDetail" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>

```

The service interface will conform to the WSDL interface document shown in Listing 5.3.2. This service uses messages defined in the Person XML Schema shown in Listing 5.3.1.

Listing 5.3.2 *PersonAbsSvc WSDL Interface Document*

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions
  name="PersonAbsSvc"
  targetNamespace="http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc"
  xmlns:ns="http://xml.netbeans.org/schema/Person"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
  <types>
    <xsd:schema
      targetNamespace="http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc">
      <xsd:import
        namespace="http://xml.netbeans.org/schema/Person"
        schemaLocation="Person.xsd"/>
    </xsd:schema>
  </types>
  <message name="getPersonDetailsRequest">
    <part name="msgPersonDetailsReq" element="ns:PersonReq"/>
  </message>
  <message name="getPersonDetailsResponse">
    <part name="msgPersonDetailsRes" element="ns:PersonRes"/>
  </message>
  <message name="getPersonDetailsFault">
    <part name="msgPersonDetailsFlt" element="ns:PersonFlt"/>
  </message>
  <portType name="PersonAbsSvcPortType">
    <operation name="getPersonDetails">
      <input name="input1" message="tns:getPersonDetailsRequest"/>
      <output name="output1" message="tns:getPersonDetailsResponse"/>
      <fault name="fault1" message="tns:getPersonDetailsFault"/>
    </operation>
  </portType>
  <plnk:partnerLinkType name="PersonAbsSvc">
    <plnk:role name="PersonAbsSvcPortTypeRole"
      portType="tns:PersonAbsSvcPortType"/>
  </plnk:partnerLinkType>
</definitions>

```

Note that this WSDL document only defines the Abstract part of the interface. We will add concrete part for each project variant we will explore. This WSDL definition will be named `PersonAbsSvc`.

To save the effort the client implementation will be exposed as a web service and will be triggered using a SoapUI web service testing project. The interface definition, TriggerCon, is shown in Listing 5.3.3.

Listing 5.3.3 *TriggerCon WSDL Interface Document*

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
  name="TriggerCon"
  targetNamespace="http://j2ee.netbeans.org/wsdl/CommonXML/TriggerCon"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://j2ee.netbeans.org/wsdl/CommonXML/TriggerCon"
  xmlns:ns="http://xml.netbeans.org/schema/Person"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <xsd:schema
      targetNamespace="http://j2ee.netbeans.org/wsdl/CommonXML/TriggerCon">
      <xsd:import
        namespace="http://xml.netbeans.org/schema/Person"
        schemaLocation="Person.xsd"/>
    </xsd:schema>
  </types>
  <message name="triggerPersonRequest">
    <part name="msgPersonDetailsReq" element="ns:PersonReq"/>
  </message>
  <message name="triggerPersonResponse">
    <part name="msgPersonDetailsRes" element="ns:PersonRes"/>
  </message>
  <message name="triggerPersonFault">
    <part name="msgPersonDetailsFlt" element="ns:PersonFlt"/>
  </message>
  <portType name="TriggerConPortType">
    <operation name="triggerPerson">
      <input name="input1" message="tns:triggerPersonRequest"/>
      <output name="output1" message="tns:triggerPersonResponse"/>
      <fault name="fault1" message="tns:triggerPersonFault"/>
    </operation>
  </portType>
  <binding name="TriggerConBinding" type="tns:TriggerConPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="triggerPerson">
      <soap:operation/>
      <input name="input1">
        <soap:body use="literal"/>
      </input>
      <output name="output1">
        <soap:body use="literal"/>
      </output>
      <fault name="fault1">
        <soap:fault use="literal" name="fault1"/>
      </fault>
    </operation>
  </binding>
  <service name="TriggerConService">
    <port name="TriggerConPort" binding="tns:TriggerConBinding">
      <soap:address location="http://localhost:${HttpDefaultPort}/TriggerConService/TriggerConPort"/>
    </port>
  </service>
  <plnk:partnerLinkType name="TriggerCon">
```

```
<plnk:role name="TriggerConPortTypeRole" portType="tns:TriggerConPortType"/>
</plnk:partnerLinkType>
</definitions>
```

This is a concrete interface. Remember to change the FQDN of the host in the WSDL to that of your host.

5.4 Common XML Project

Let's create a Project Group to contain projects developed in this chapter. This project group will be called WSPolicyExploration and will contain the common XML artifacts, the Person XML Schema, the PersonAbsSvc WSDL and the TriggerCon WSDL.

Let's create a New Project ... -> SOA -> BPEL Module, named CommonXML. Figures 5.4.1 and 5.4.2 show important steps in the process. This BPEL project is just a convenient location for the XML documents we will be creating.

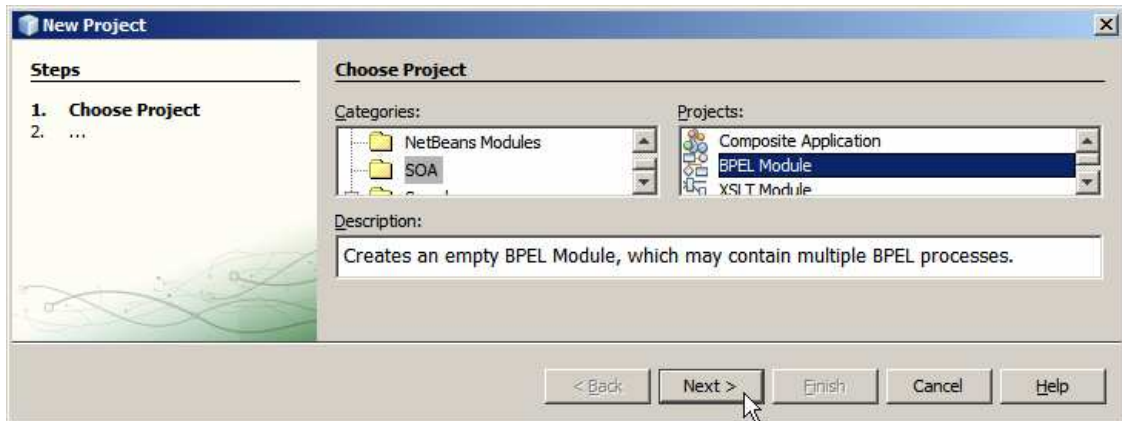


Figure 5.4.1 Create BPEL Module

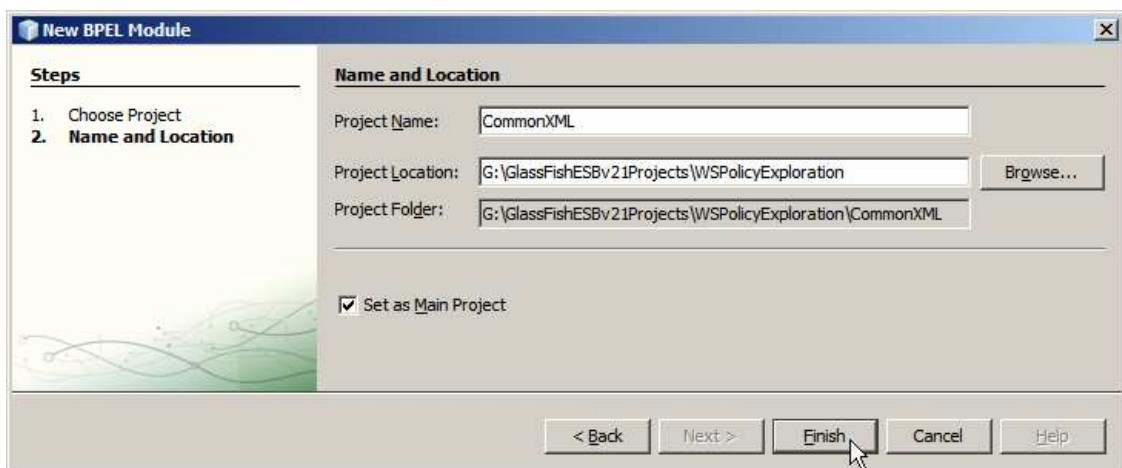


Figure 5.4.2 Naming the project and specifying location

The skeleton BPEL process model, commonXML.bpel, can be deleted since it will not be used.

Right-click the project name and choose New ... -> Other ... -> XML -> XML Schema, Figure 5.4.3.

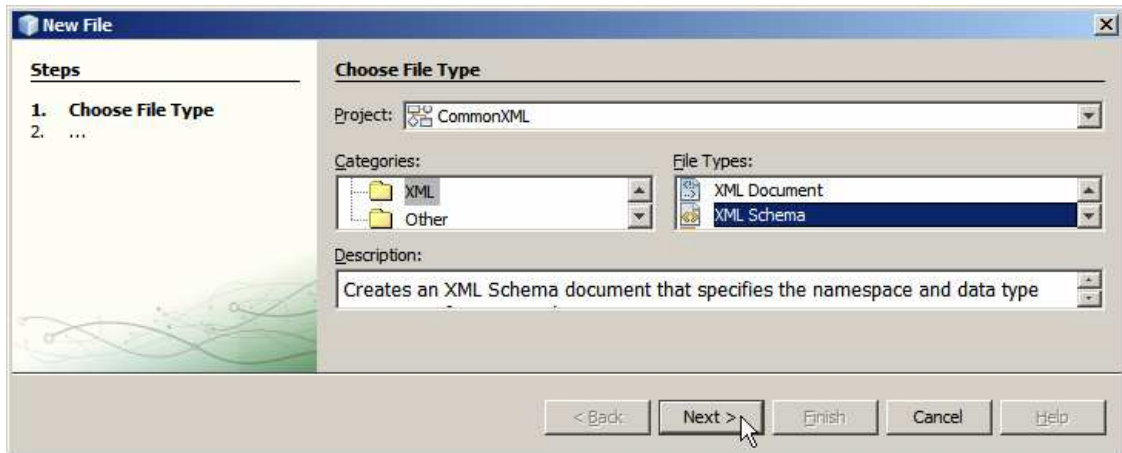


Figure 5.4.3 Create a new XML Schema

Name this schema Person and click Finish.

When the new XML Schema document opens in the editor window, switch to the Source view and select all the text. Figure 5.4.4 illustrates this.

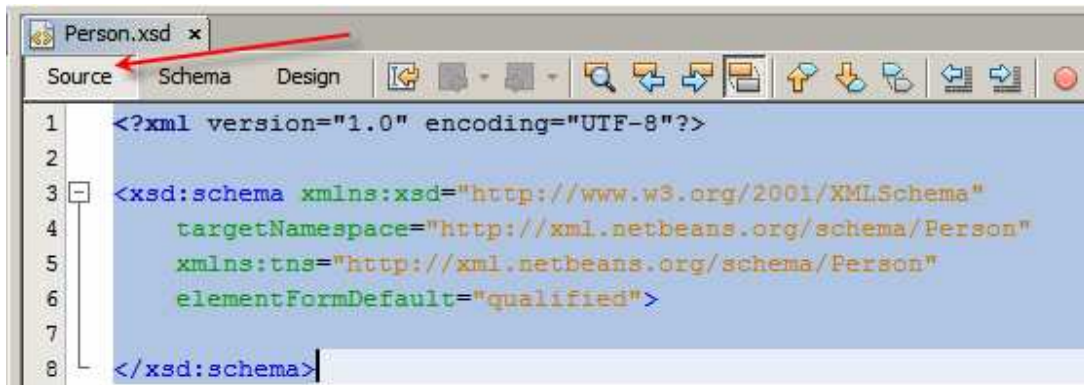


Figure 5.4.4 Select the content of the new XML Schema document

Paste XML Schema text from Listing 5.3.1 in place of the selected text. Check XML and Validate XML, illustrated in Figure 5.4.5, and resolve any issues that might have arisen.

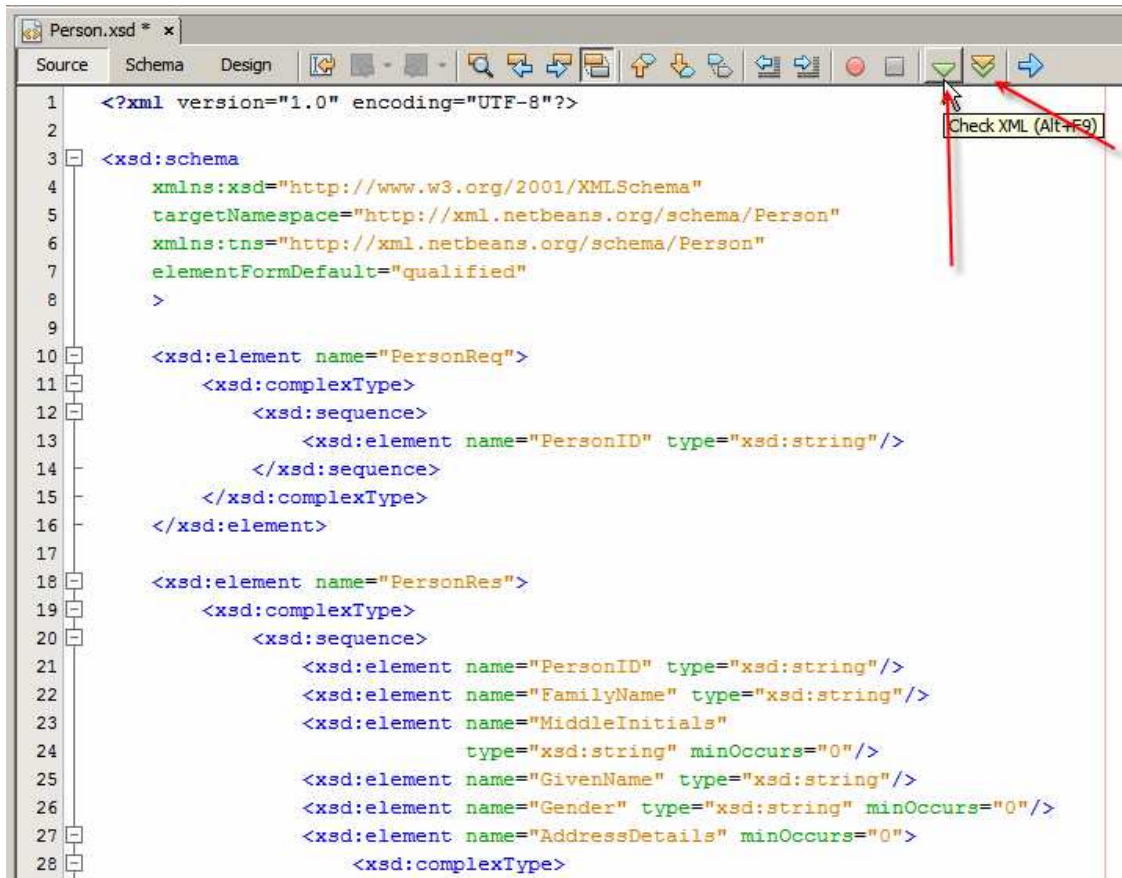


Figure 5.4.5 *Check and Validate*

Save the new schema.

Create a New ... -> WSDL Document ..., named PersonAbsSvc. This will be an Abstract WSDL Document. Figure 5.4.6 illustrates the first dialogue box involved in the process.

Click Next to advance to the next panel.

Change the name of the operation to `getPersonDetails`. Change names of Input and Output message parts to `msgPersonDetailsReq` and `msgPersonDetailsRes` respectively. Add a new Fault message part and name it `msgPersonDetailsFlt`. Figure 5.4.7 illustrated the dialogue box at this point in the process.

For each message part click the small ellipsis button and choose appropriate element from the Person XML Schema. For `msgPersonDetailsReq` it will be `PersonReq`, for `msgPersonDetailsRes` it will be `PersonRes` and for `msgPersonDetailsFlt` it will be `PersonFlt`. Figure 5.4.8 illustrates a step in this process. Figure 5.4.9 shows the dialogue box with all parts with correct elements.

Click Finish to complete the wizard. The resulting WSDL should look like that shown in Listing 5.3.2.

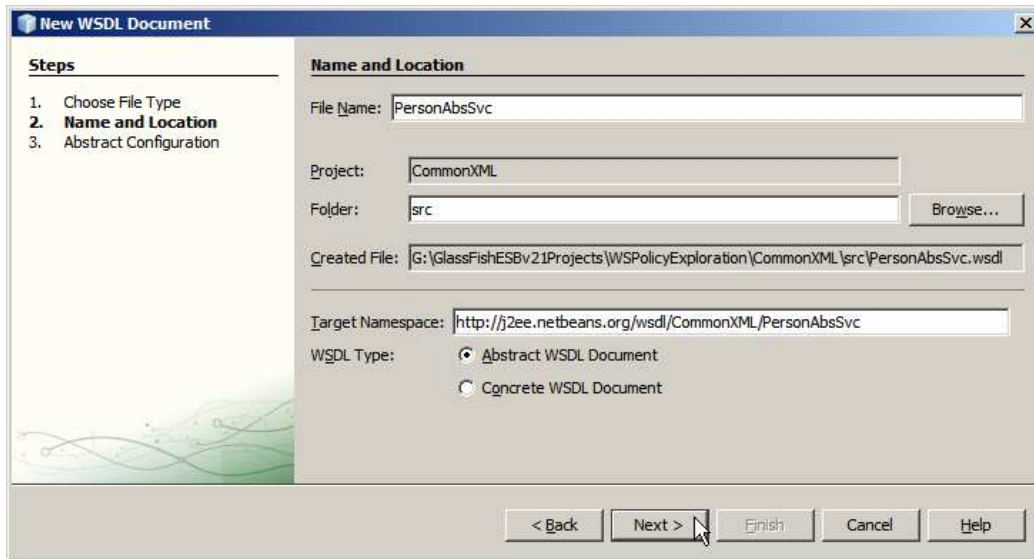


Figure 5.4.6 Create a new Abstract WSDL, step 1

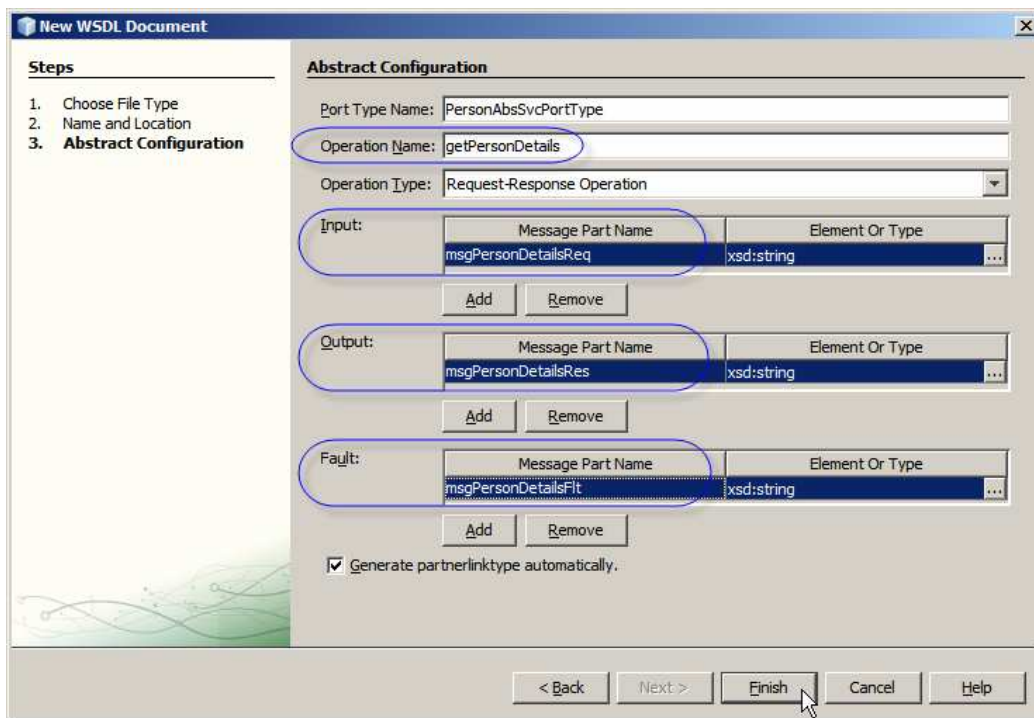


Figure 5.4.7 Name operation and message parts

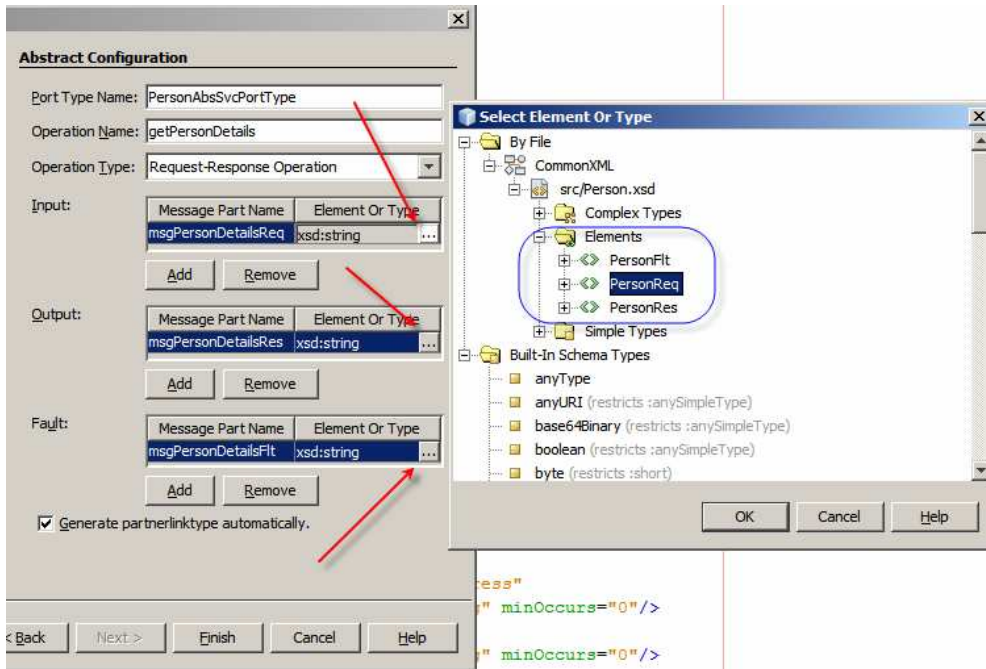


Figure 5.4.8 Change message part types

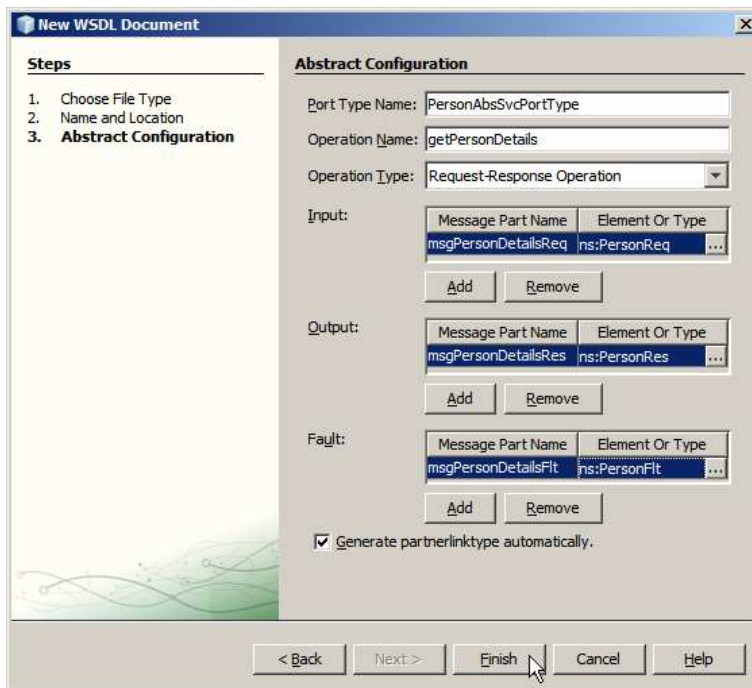


Figure 5.4.9 Completed Abstract Configuration

Finally, let's create the TriggerCon WSDL, which will be used to expose the client implementation as a web service so it can be triggered by a SopaUI web service testing project.

Create a New -> WSDL Document, names TriggerCon. It will be a Concrete WSDL, SOAP Binding, Document/Literal Type. Figure 5.4.10 shows the dialogue panel at this step in the process.

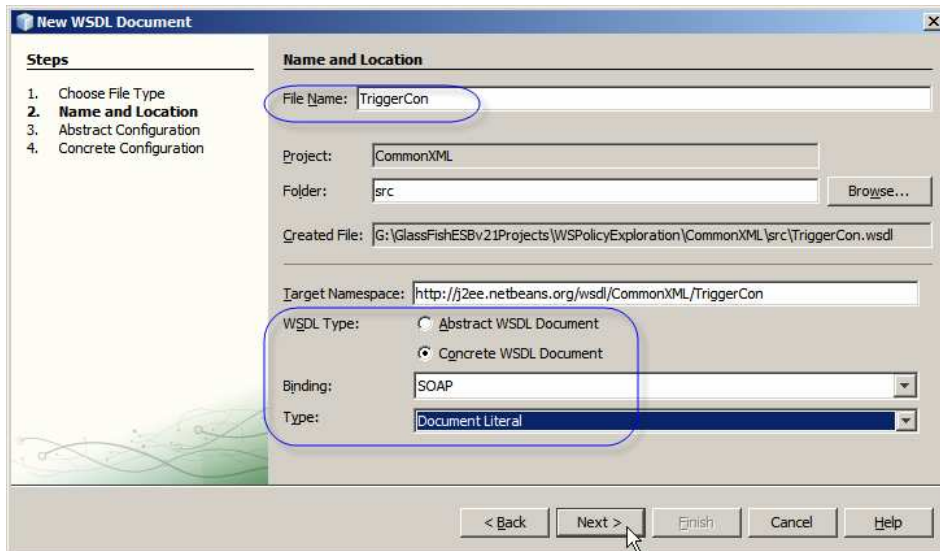


Figure 5.4.10 Concrete WSDL, SOAP, Document/Literal

Click Next to advance o the next panel. Change operation name to triggerPerson. Change message part names to msgPersonDetailsReq, msgPersonDetailsRes, add a Fault part and name it msgPersonDetailsFlt. Change “Element or Type” for the message parts to Personreq, PersonRes and PersonFlt, much the same way as was done for the PersonAbsSvc WSDL earlier. Figure 5.4.11 illustrates the final panel.

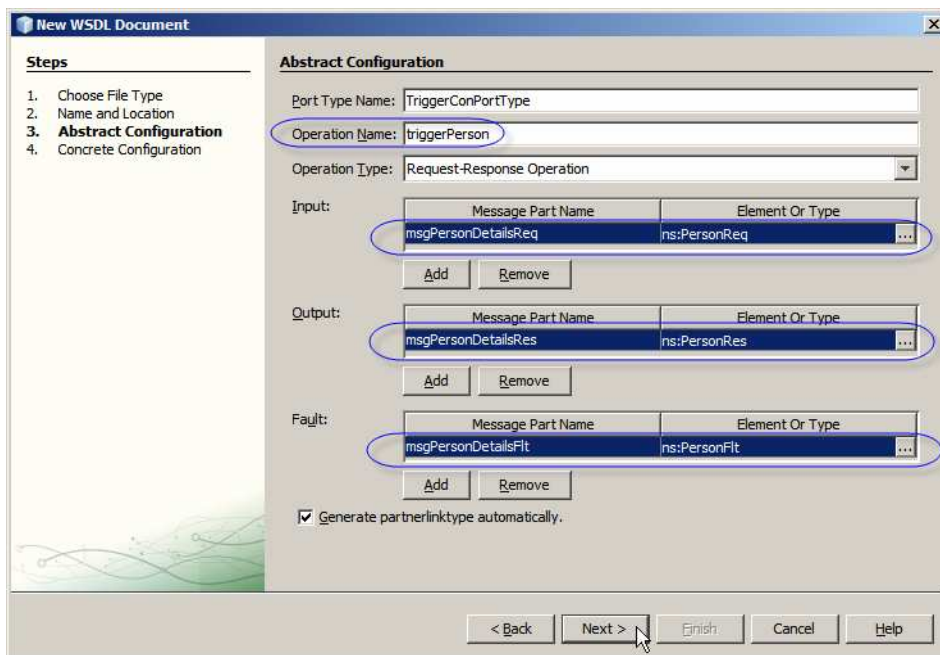


Figure 5.4.11 TriggerCon WSDL abstract configuration

Click Next, accept defaults and click Finish.

Our project, CommonXML, should look like that shown in Figure 5.4.12.

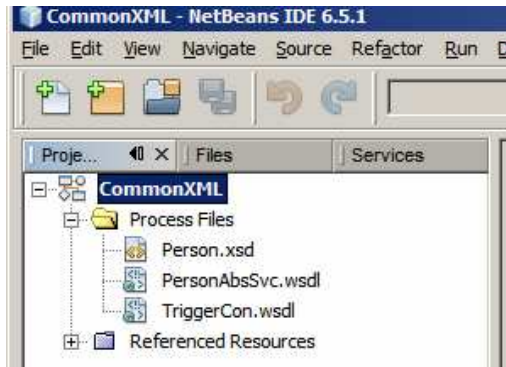


Figure 5.4.12 XML Schema and WSDL in CommonXML

5.5 PersonSvc BPEL Module

Let's create the BPEL Module project, PersonSvc, to implement, in BPEL 2.0, the service whose interface is defined by the PersonSvc WSDL, in CommonXML project.

Right-click in any blank area of the Project Explorer window and choose New Project ... -> SOA -> BPEL Module. Name the project PersonSvc.

Expand the node tree to "Referenced Resources". Right-click "Referenced Resources" and choose Add -> Project Resource ..., as illustrated in Figure 5.5.1.

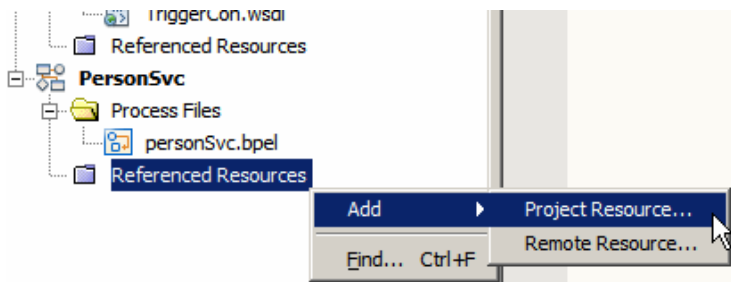


Figure 5.5.1 Add Project Resource

Locate the WSDL PersonAbsSvc and click Open, as illustrated in Figure 5.5.2.

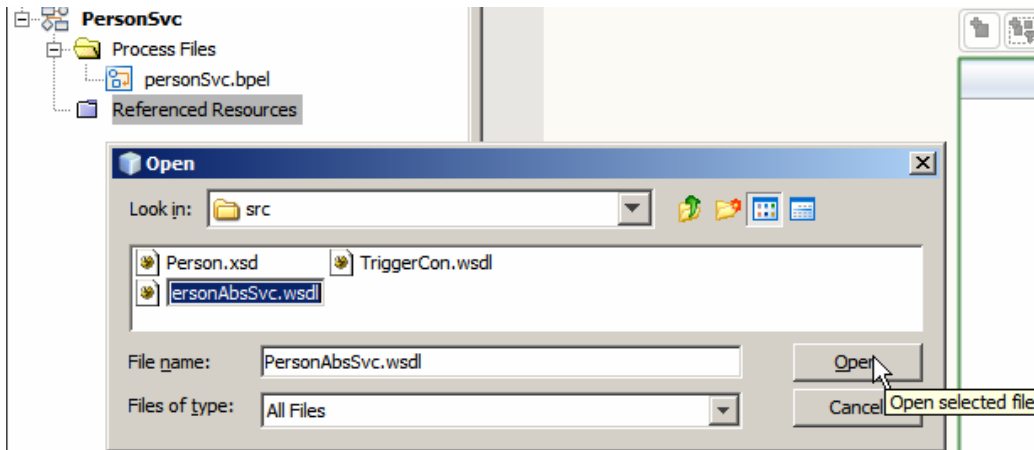


Figure 5.5.2 Locate *PersonAbsSvc* WSDL

Open the BPEL process, *personSvc.bpel*, if it is not already open, and drag the reference *CommonXML/PersonAbsSvc* onto the target marker in the left-hand swim line, as shown in Figure 5.5.3.

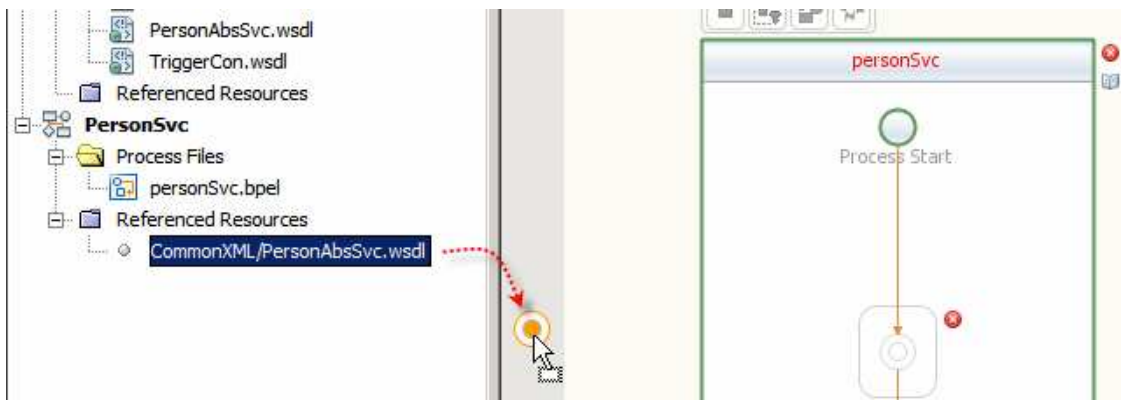


Figure 5.5.3 Drag *PersonAbsSvc* WSDL reference onto the process canvas

Name the partner link *PersonRR*.

From the Web Service Palette drag *Receive*, *Assign* and *Reply* activities onto the target markers inside the *personSvc* process scope, as shown in Figure 5.5.4.

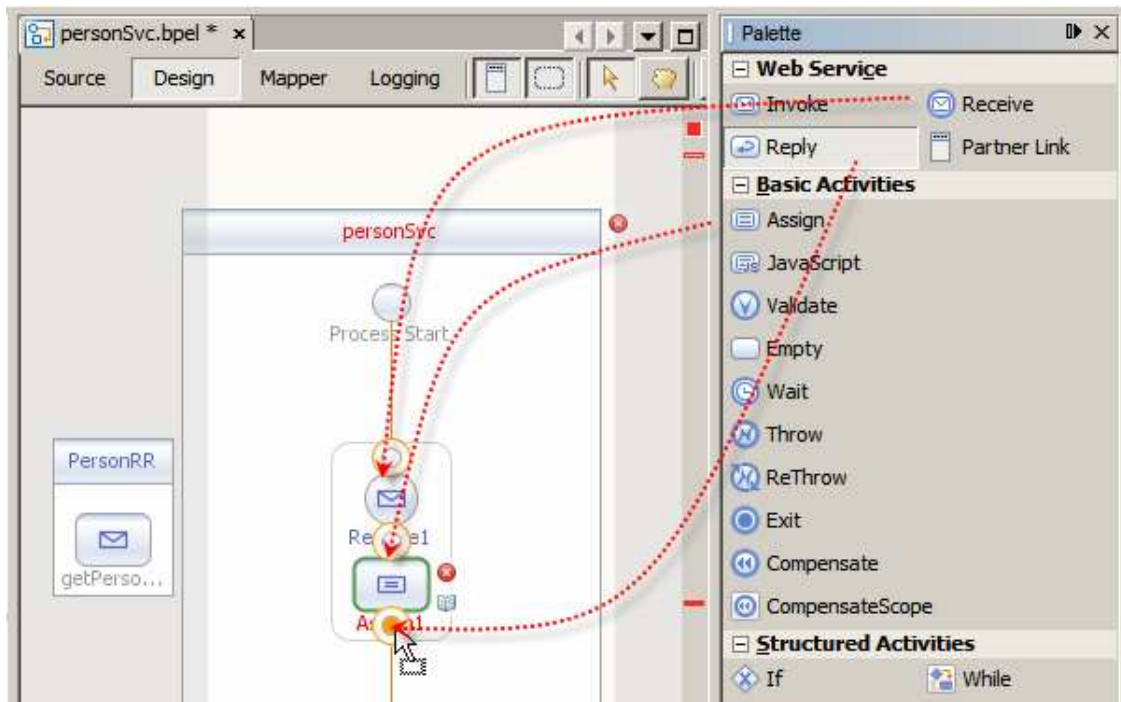


Figure 5.5.4 Add Receive, Assign and Invoke activities

Connect Receive and Reply activities to the PersonRR partner Link, as illustrated in Figure 5.5.5.

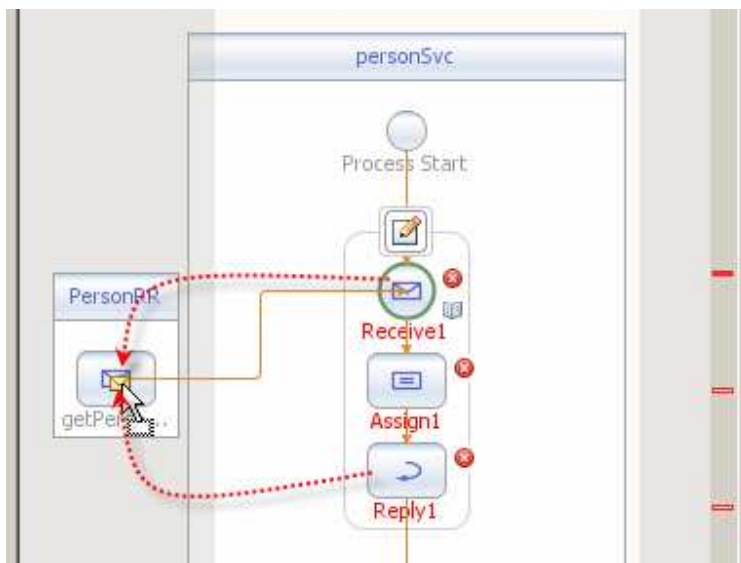


Figure 5.5.5 Connect Receive and Reply activities to the Partner Link

Select the Receive activity, click the Edit icon, click the Create button alongside the “Input variable”, change the name of the variable to GetPersonReq, click OK. This will add a variable, GetPersonreq, which will contain the request message. Figure 5.5.6 illustrates the interesting points.

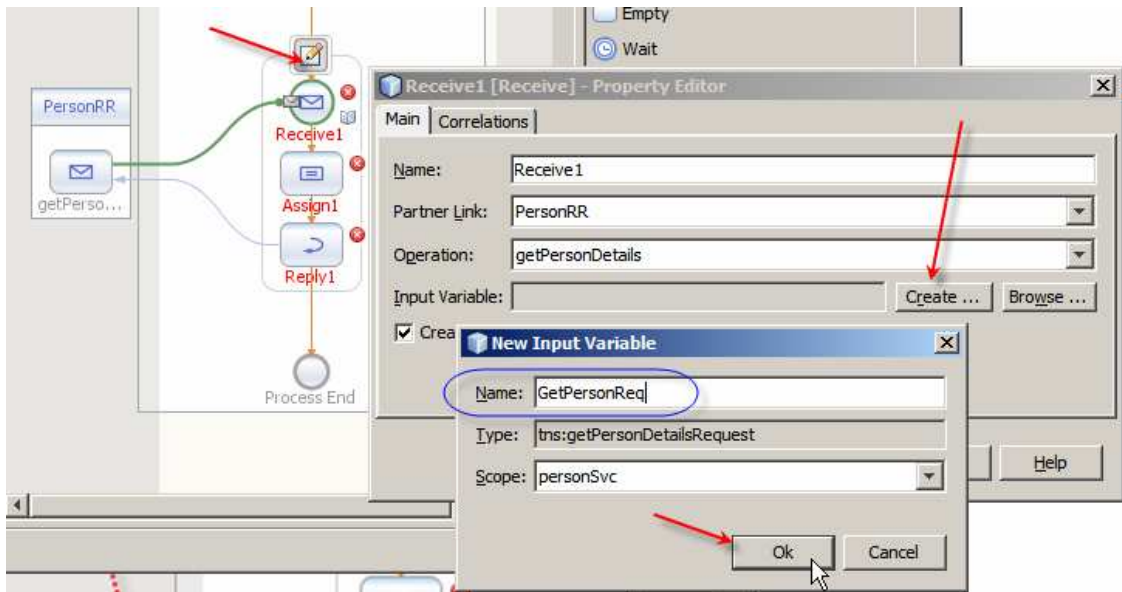


Figure 5.5.6 Add variable to contain request message

Repeat the process for the Reply activity, naming the variable GetPersonRes, as shown in Figure 5.5.7.

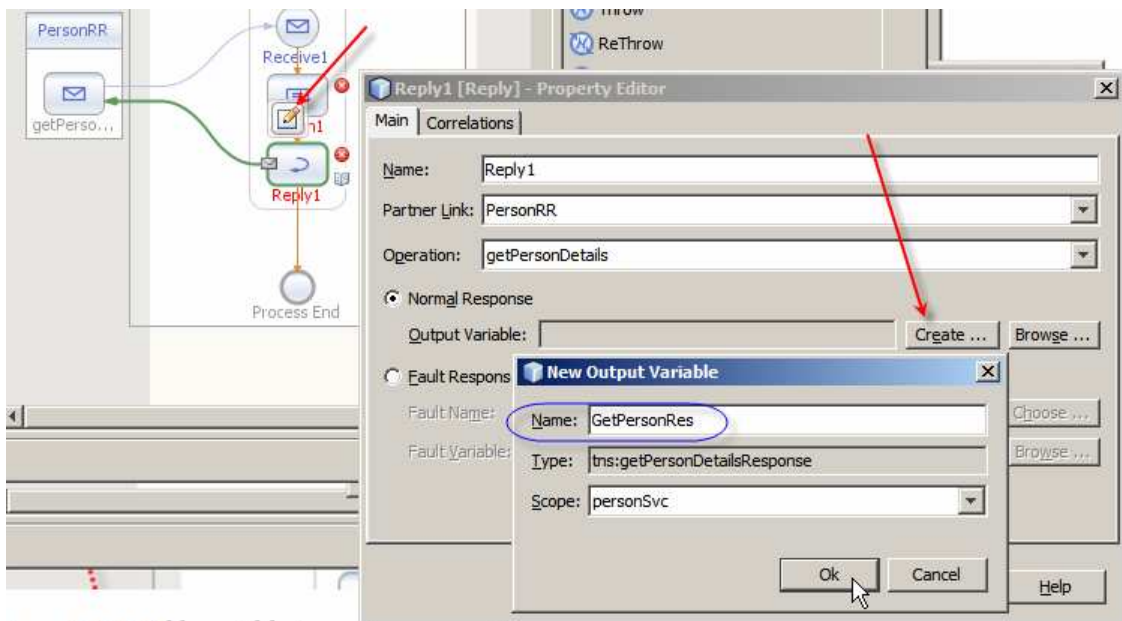


Figure 5.5.7 Add variable GetPersonRes to the reply activity

Double-click the Assign activity, or select the Assign activity and switch to Mapper mode. When in Mapper, map the request values and literal to the appropriate nodes of the response message. Figure 5.5.8 illustrates the mapping. Feel free to provide your own values for the literals.

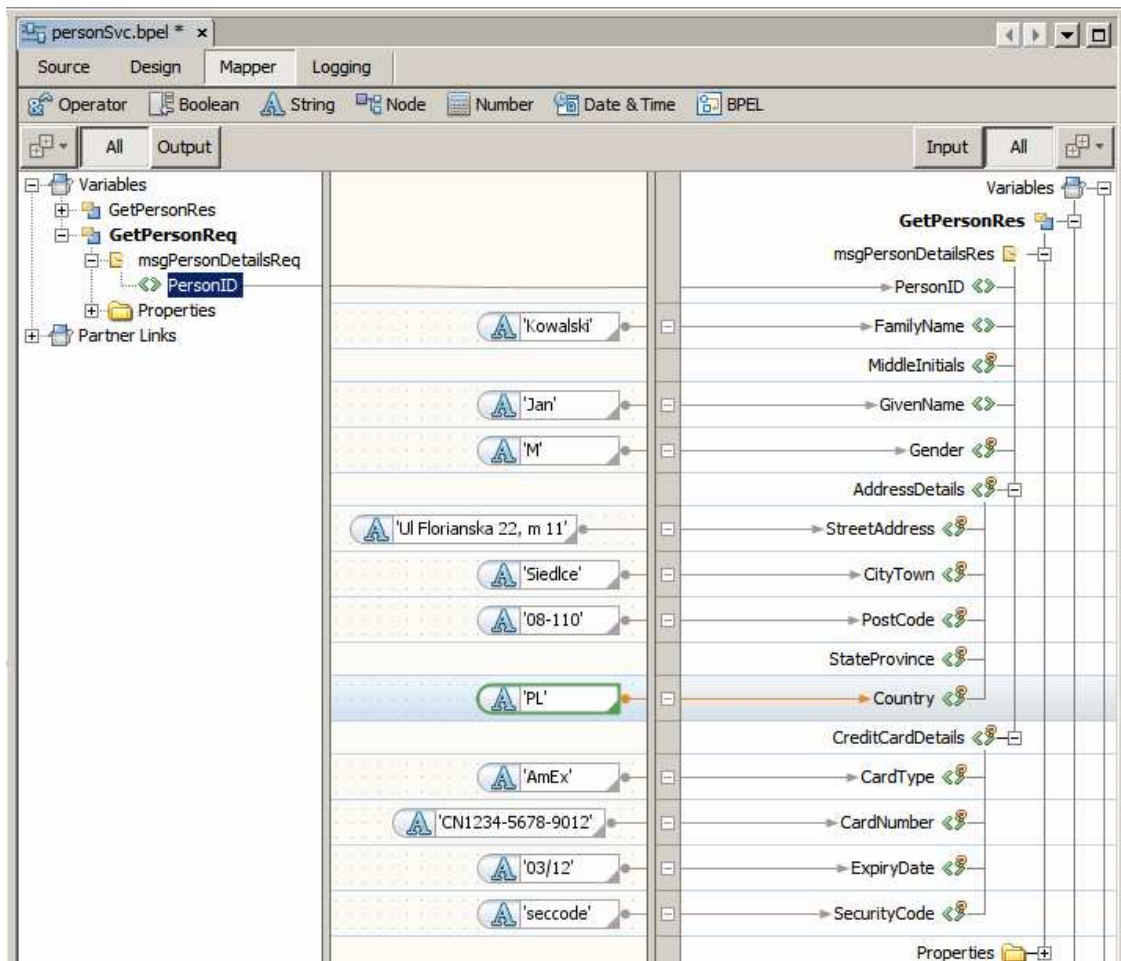


Figure 5.5.8 Mapping response values

Right-click the name of the project and choose Build. Figure 5.5.9 illustrates this.

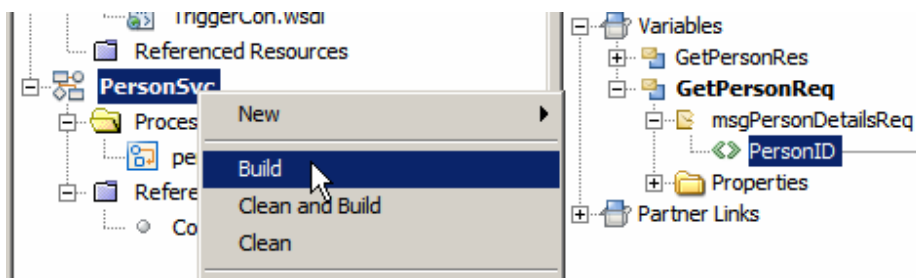


Figure 5.5.9 Build the project

The PersonSvc project, which implements service logic, is ready. We will develop the composite application that will encapsulate this logic and deploy it to runtime in subsequent sections.

5.6 PersonCli BPEL Module

Let's create the BPEL Module project, PersonCli, to implement, in BPEL 2.0, the invoker of the PersonSvc service. The BPEL process, implemented in this project, will be exposed as a web service using the TriggerCon WSDL, developed earlier. This process will, in turn, invoke the PersonSvc service using the Abstract WSDL interface defined in PersonAbsSvc.

In the new BPEL Module project add two project resource references, the PersonAbsSvc WSDL and the TriggerCon WSDL. Figure 5.6.1 illustrates project hierarchy after these resources have been added.

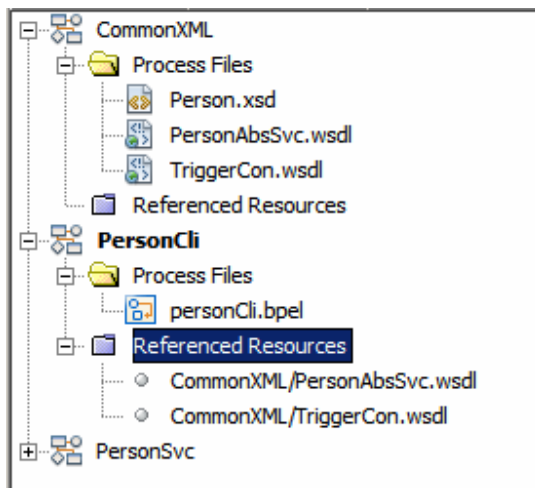


Figure 5.6.1 PersonAbsSvc and TriggerCon WSDL References Resources

Open the personCli business process, if it is not already open. Drag the TriggerCon WSDL reference to the left-hand (provide) swim line and the PersonAbsSvc WSDL reference to the right-hand (invoke) swim line. Name the partner links TriggerRR and PersonWS respectively, as shown in Figure 5.6.2.

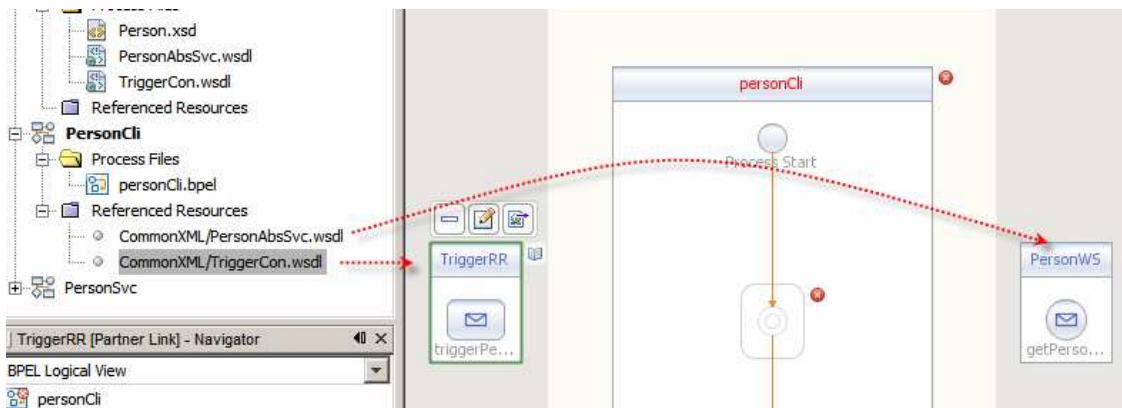


Figure 5.6.2 Provide and Invoke Partner Links

Recall that both TriggerCon and PersonAbsSvc use the same request and response structures. BPEL logic we are developing will consist of copying the request message from TriggerCon to PersonAbsSvc and the response message from PersonAbsSvc to TriggerCon. The TriggerCon

interface will not be secured in any way so we can conveniently invoke the client service using SoapUI. Security policies, if any, will be applied to the PersonAbsSvc interaction.

Let's add Receive, Assign, Invoke, Assign and Reply activities to the process canvas, connect Request and Reply to the TriggerRR partner Link and Invoke activity to the PersonWS Partner Link. Figure 5.6.3 illustrates the process at this point in development.

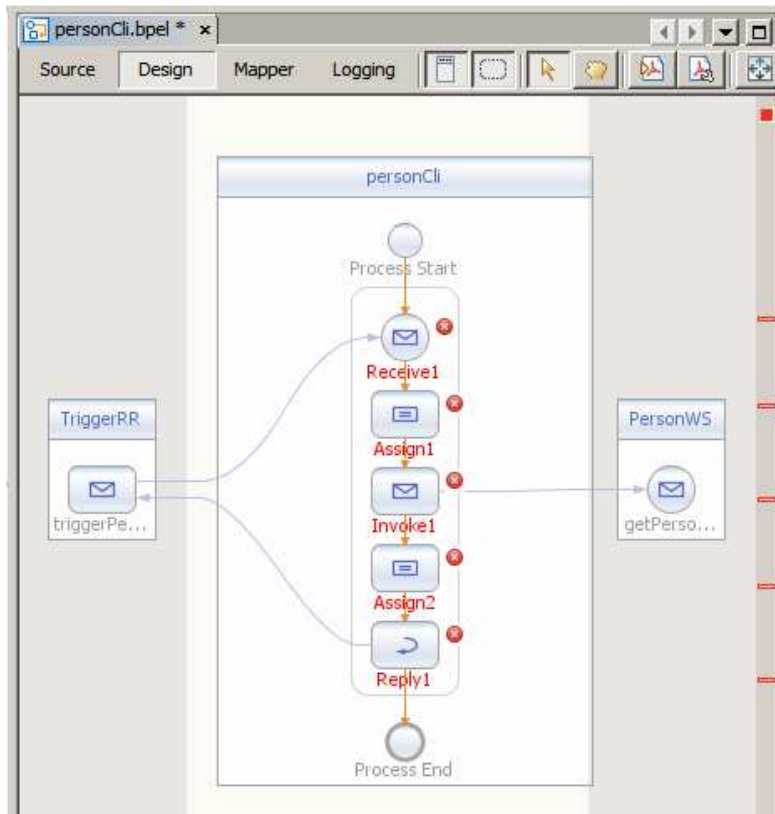


Figure 5.6.3 Activities added and connected

Note the “error indicators”. These tell us that activities are not configured. Figures 5.6.4 and 5.6.5 show error messages for the Assign and the Reply activities.

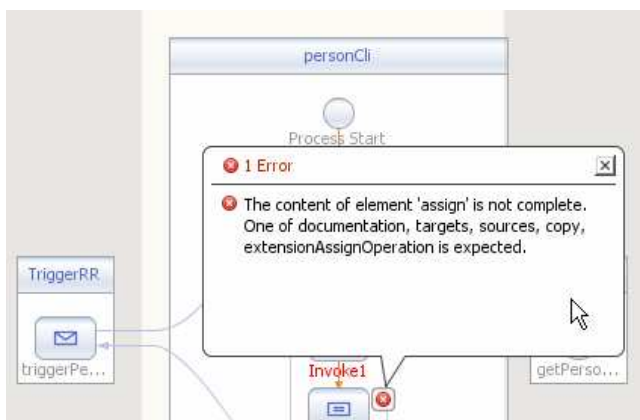


Figure 5.6.4 Assign error – no mapping

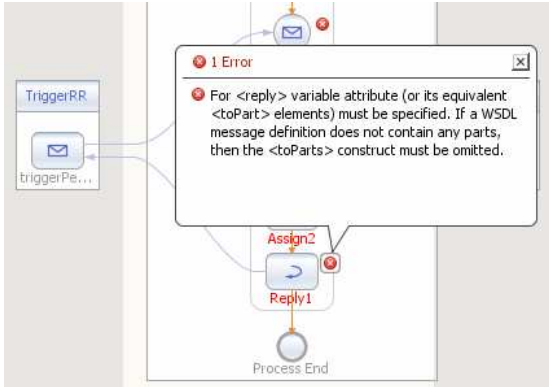


Figure 5.6.5 Error on Reply activity

BPEL Editor performs continuous background validation so it picks up the fact that we did not finish configuring activities. All these error will be resolved as we continue to work in the process.

Edit Receive, Reply and Invoke activities and add variables that will contain messages – vTriggerReq for Receive, vTriggerRes for the Reply and vPersonReq and vPersonRes for the Invoke. This is done the same way as has been done in the ProcessSvc so no pictures should be necessary. Figure 5.6.6 calls out variable names configured for the Invoke activity.

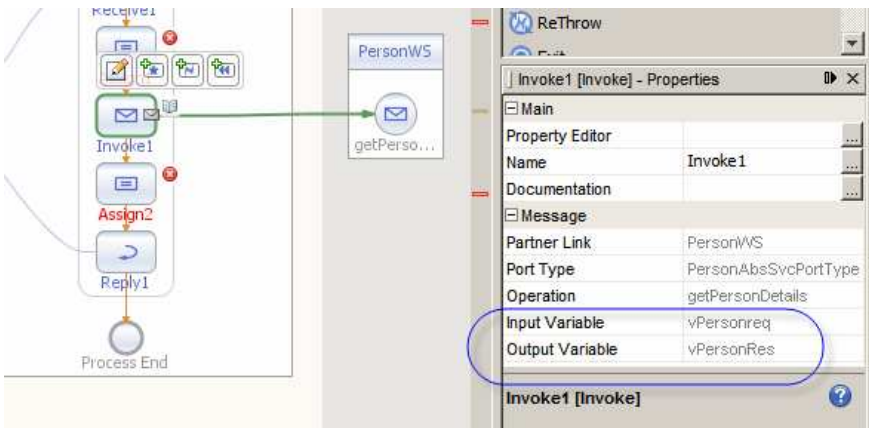


Figure 5.6.6 Variable names for the Invoke activity

We can now complete the Assign activities. Mapping in Assign1 are shown in Figure 5.6.7.



Figure 5.6.7 Mapping in Assign1

Mapping in Assign2 are shown in Figure 5.6.8.

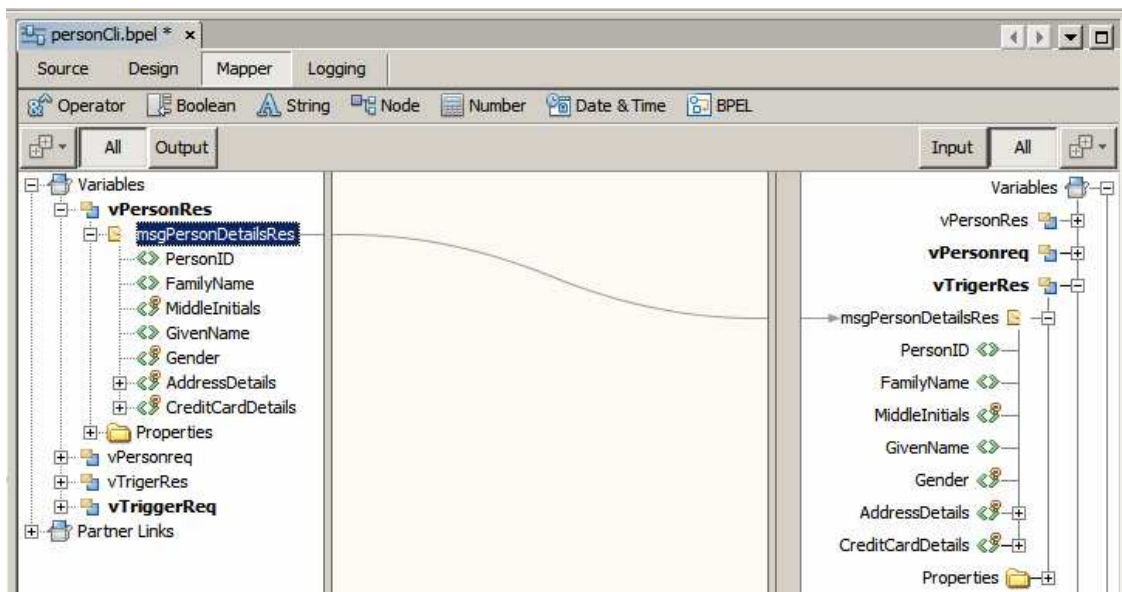


Figure 5.6.8 Mapping in Assign2

In Assign2 we map the root nodes, instead of mapping each individual field. We can do this because both the source and the destination messages are of the same structure.

Finally, let's configure the process so that it is lenient with respect to missing data. Switch to Design view, click the personCli process scope and choose "Yes" for the value of process property "Ignore Missing From Data". Figure 5.6.9 illustrates this.

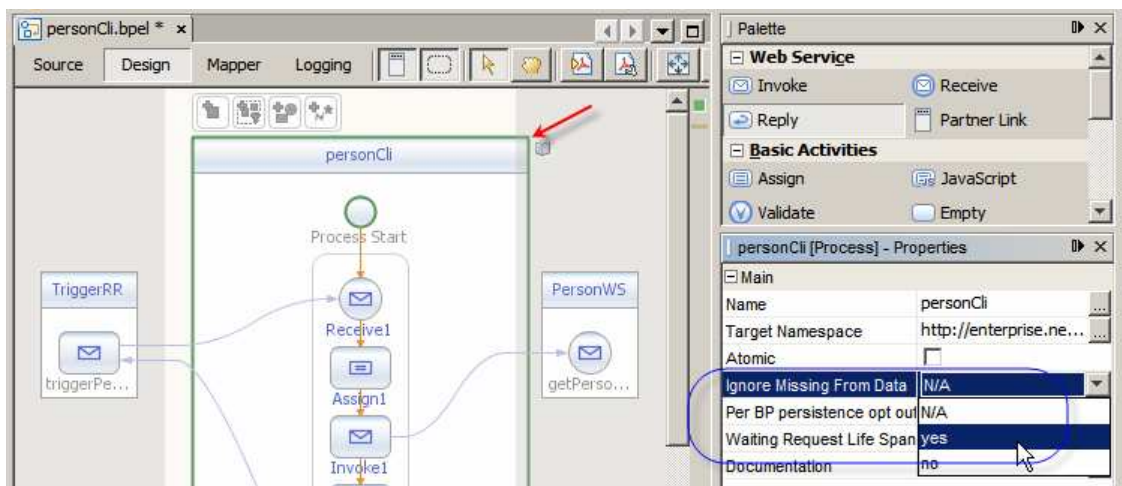


Figure 5.6.9 Set "Ignore Missing From Data" to "Yes"

The PersonCli project, which implements client-side logic, is ready. Build the project.

We will develop the composite application that will encapsulate this logic module and deploy it to runtime in the subsequent sections.

5.7 Person Service – Plain End-to-End

The service provider and service invoker BPEL Module are ready. We are now in a position to create Composite Applications for each and to exercise the solution end-to-end.

Let's start by creating the composite application, PersonSvc_CA_Plain, for the PersonSvc BPEL module, a web service testing project, PersonSvc_WSTP, to exercise this application, then perform the service implementation test.

Create a New Project -> SOA -> Composite Application, named PersonSvc_CA_Plain. Once created, drag the BPEL Module PersonSvc onto the Composite Application Service Assembly canvas and click Build. Figure 5.7.1 illustrates this.

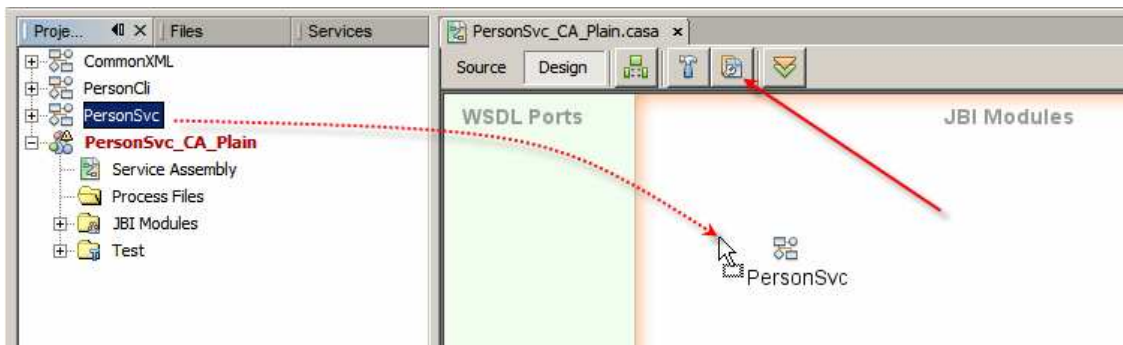


Figure 5.7.1 Add BPLE Module to the CASA canvas and Build

Because the service interface WSDL is an Abstract WSDL we don't see a Binding Component on the CASA canvas once the build is finished. We need to provide a concrete binding. At this point we could use any available binding. Since we are building a web service implementation we will drag the soap binding to the canvas. Figure 5.7.2 illustrates this.

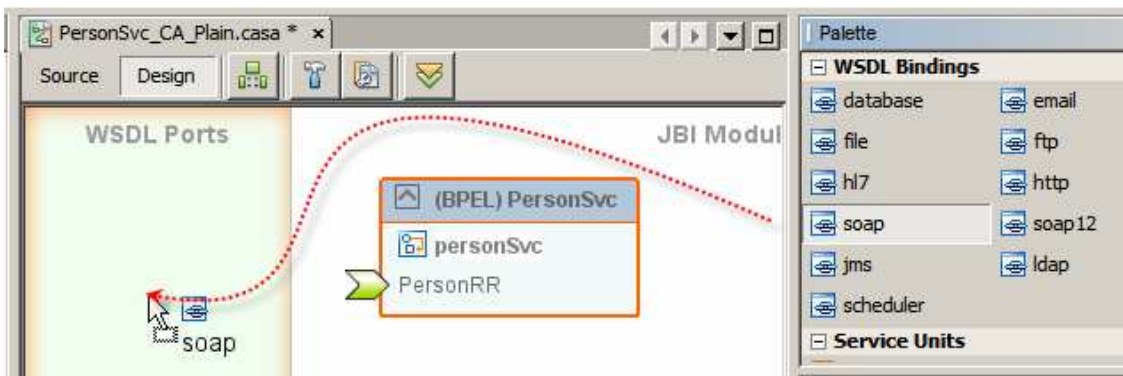


Figure 5.7.2 Add soap binding to the CASA canvas

Connect Consume connector of the SOAP BC to the Provide connector of the BPEL Module then build the process again. Figure 5.7.3 illustrates the final CASA map.

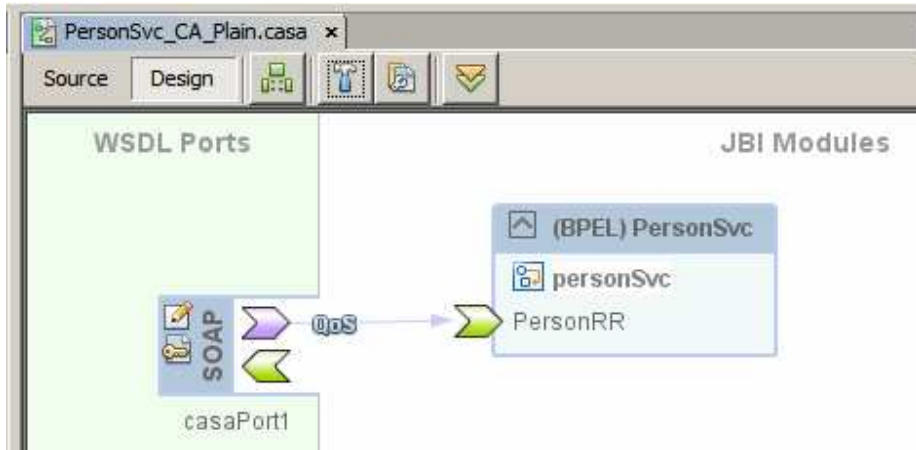


Figure 5.7.3 Completed CASA map

Click the “pencil and paper” icon to open SOAP BC properties and note the endpoint address in the Location property, shown in Figure 5.7.4.

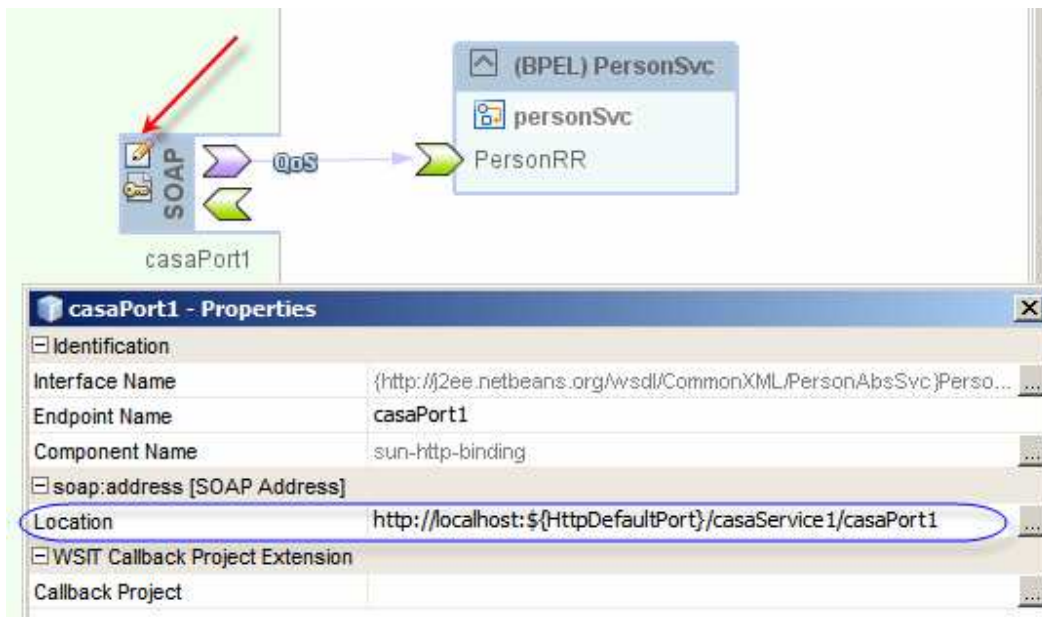


Figure 5.7.4 Location property

Note the construction “\${HttpDefaultPort}”. The HttpDefaultPort is the name of the environment variable that gets replaced, at build time, with the value configured for the default HTTP port used by the JBI container. By default this will be 9080. For me this will be 29080. You can find out what it is by looking at properties of Services -> Servers -> GlassFish v2 -> JBI -> Binding Components -> sun-http-binding, specifically property named “Default HTTP Port Number”. While at it, also note the value of the “Default HTTPS Port Number”. This is the port for the SSL/TLS protocol. More on that later.

Deploy the project to the local GlassFish instance. For me this will be the “GlassFish v2” running on host mcz02.aus.sun.com.

Let's now create a New Project ... -> Java EE -> Web Service Testing Project and name it PersonSvc_WSTP. We will use this project to submit a SOAP request to the PersonSvc_CA_Plain service, which we just built and deployed, to verify that it works. Enter, or paste, the endpoint URL from the Location property, discussed above, with host and port configured as required, into the property "Initial WSDL (URL)". For me this will be:

`http://mcz02.aus.sun.com:29080/casaService1/casaPort1?WSDL`

For you the FQDN of the host will be different and the port number will be 9080 if you have a default GlassFish installation.

Once the project is created, expand the nodes all the way to getPersonDetails, right-click and choose New Request. Figure 5.7.5 illustrates this.

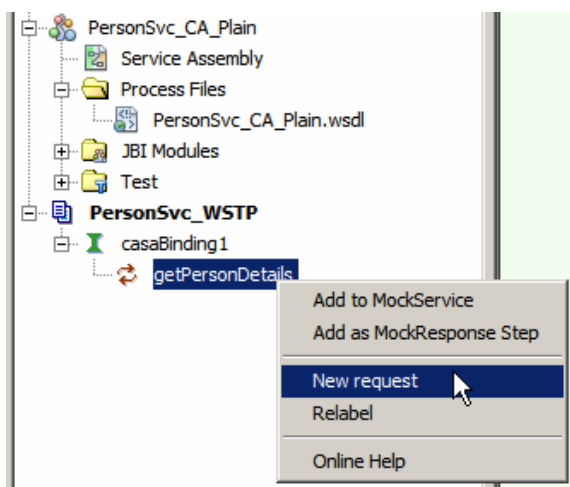


Figure 5.7.5 Create new Soap Request

Modify the request by replacing "gero et" with "q2345", or whatever value you find attractive, and submit the request as shown in Figure 5.7.6.



Figure 5.7.6 Submit SOAP Request with the PersonID of 12345

With mapping in the PersonSvc BPEL Module as shown in Figure 5.5.8 the SOAP Response will look like that in Figure 5.7.7.

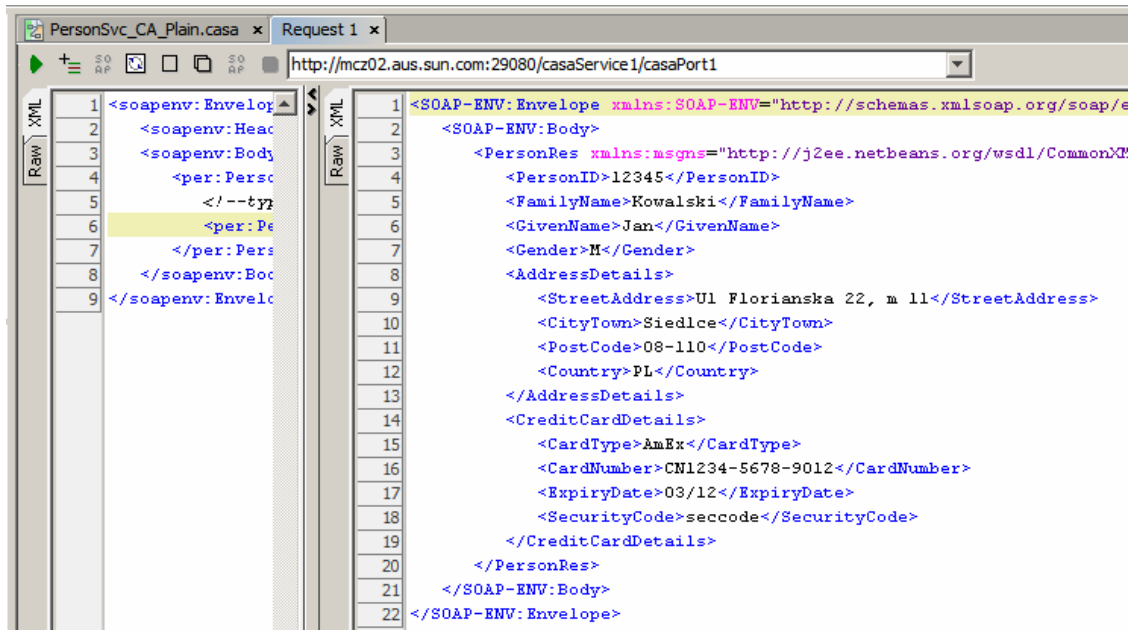


Figure 5.7.7 SOAP Response

The service PersonSvc works as expected. We will not use the testing project PersonSvc_WSTP again. We will create the composite application for the PersonCli BPEL Module and will use it to exercise the end-to-end solution.

Create a New Project -> SOA -> Composite Application, named PersonCli_CA_Plain. Drag the PersonCli BPEL Module onto the CASA canvas and Build. Figure 5.7.8 illustrates the key points.

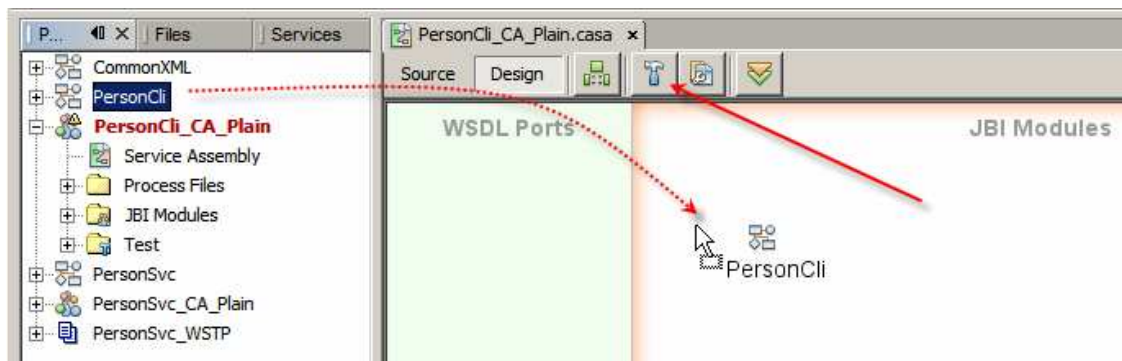


Figure 5.7.8 Create PersonCli_CA_Plain Composite Application

Note that the Consume connector of the PersonCli BPEL Module is not connected to a binding component. This is because the PersonAbsSvc WSDL, which we used in the BPEL process, is an Abstract WSDL. Figure 5.7.9 illustrates the CASA canvas at this point.

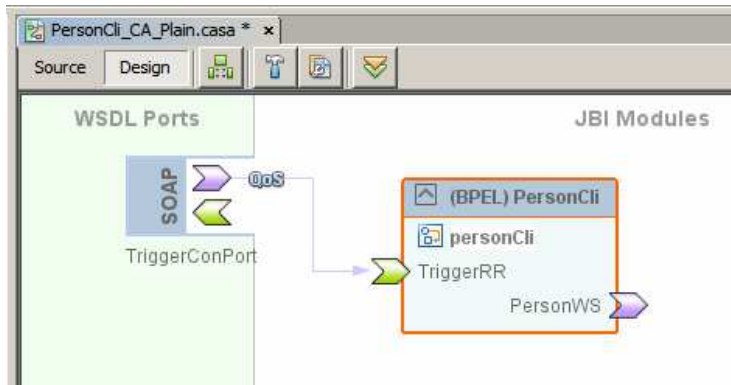


Figure 5.7.9 CASA canvas with unconnected PersonWS Partner Link

We will add a concrete WSDL to this composite application project to provide concrete binding for the PersonWS partner link.

Copy the WSDL URL of the PersonSvc_CA_Plain service to the clipboard. For me this will be:

`http://mcz02.aus.sun.com:29080/casaService1/casaPort1?WSDL`

For you the FQDN of the host will be different and the port number will be 9080 if you have a default GlassFish installation.

Right-click on the name of the project, PersonCli_CA_Pain, choose New -> Other -> XML -> “External WSDL Document(s)” and paste the WSDL URL into the “From URL” text box. Figure 5.7.10 illustrates the dialog box.

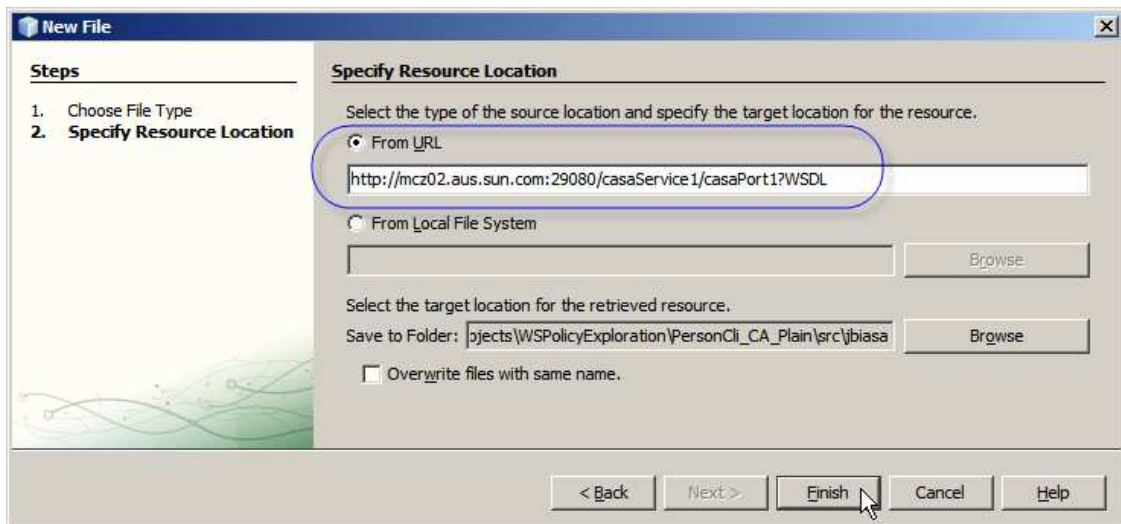


Figure 5.7.10 New External WSDL document being created

New WSDL and XSD objects will be added under the Process File node in the project hierarchy. Figure 5.7.11 illustrates this.

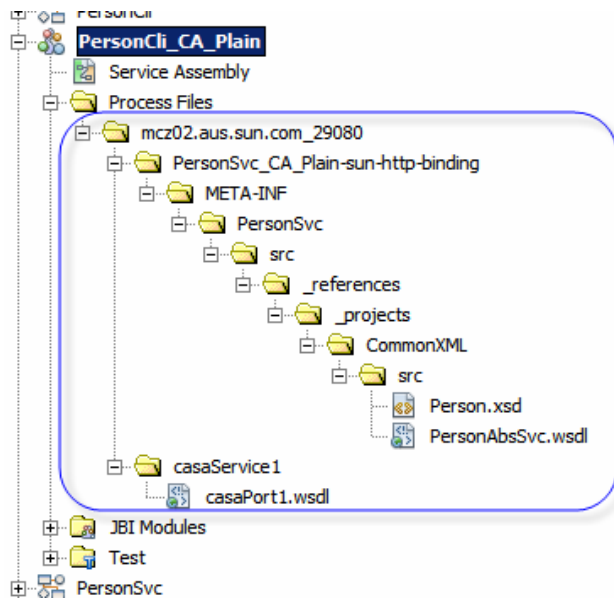


Figure 5.7.11 WSDLs and XSD added to the Composite Application project

Right-click on the CASA canvas inside the “WSDL Ports” swim line and choose “Load WSDL Port...”, as shown in Figure 5.7.12, to add the SOAP BC, configured to communicate with the PersonSvc service, to the CASA canvas.

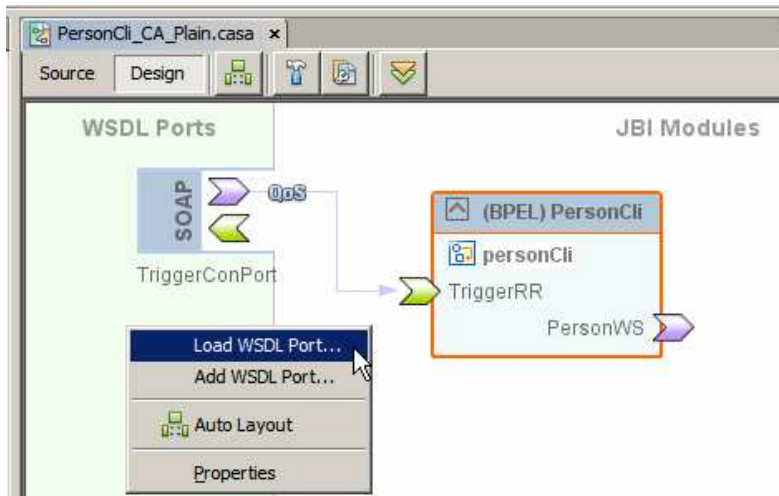


Figure 5.7.12 Load WSDL Port, part 1

In the dialogue box that appears select the one and only WSDL Port, as shown in Figure 5.7.13, and click OK. Build the CA project.

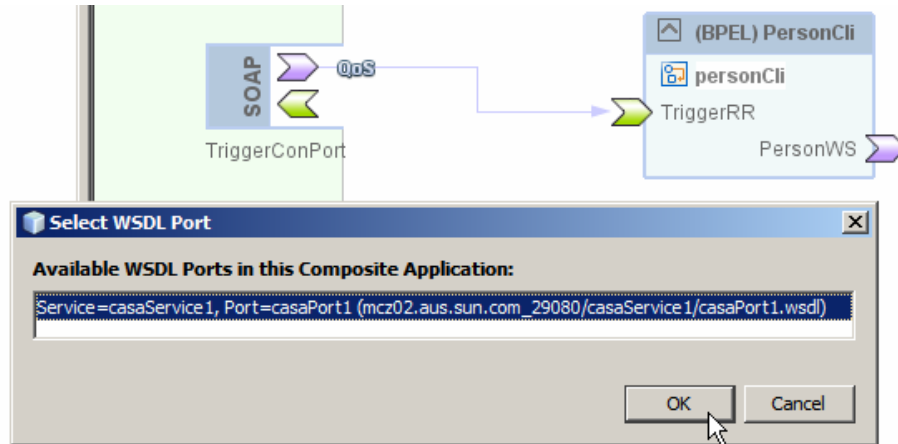


Figure 5.7.13 Select WSDL Port to add

The CASA canvas should now look like that shown in Figure 5.7.14.

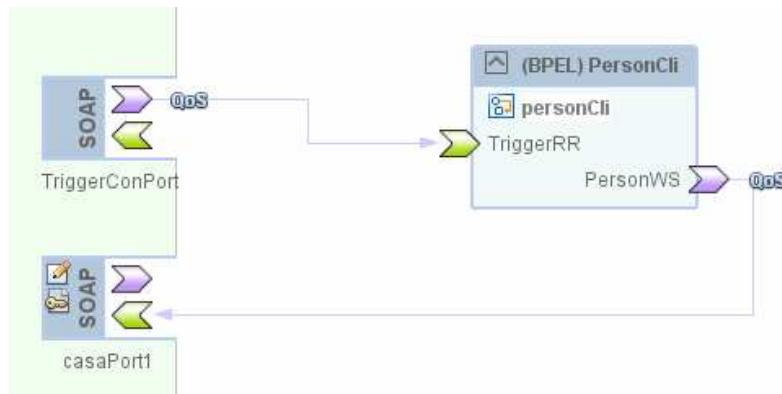


Figure 5.7.14 CASA canvas with both binding components added and connected

Deploy the project to the local instance of the GlassFish Applciatin Server. For me this will be “GlassFish v2” running on mcz02.aus.sun.com.

Locate and copy to the clipboard the endpoint URL for the TriggerCon connector. The WSDL associated with that endpoint, TriggerCon.wsdl in project CommonXML, will have that value. For me this is:

```
http://localhost:${HttpDefaultPort}/TriggerConService/TriggerConPort
```

Replace `${HttpDefaultPort}` with the correct value. For me this will be 29080, and append “?WSDL?” to the end of the URL. For me the final value will be:

```
http://localhost:29080/TriggerConService/TriggerConPort?WSDL
```

This is the URL to which the web service testing project, created next, will submit SOAP requests. Create a New -> Java EE -> Web Service testing Project, names PersonCli_WSTP. Use the URL shown above as the “Initial WSDL (URL/file)”.

Expand the nodes, right-click on triggerPerson binding, choose “New Request” and create a request. Modify PersonID value to 54321, or whatever value you find attractive, and submit the request. Figure 5.7.15 illustrates the request.

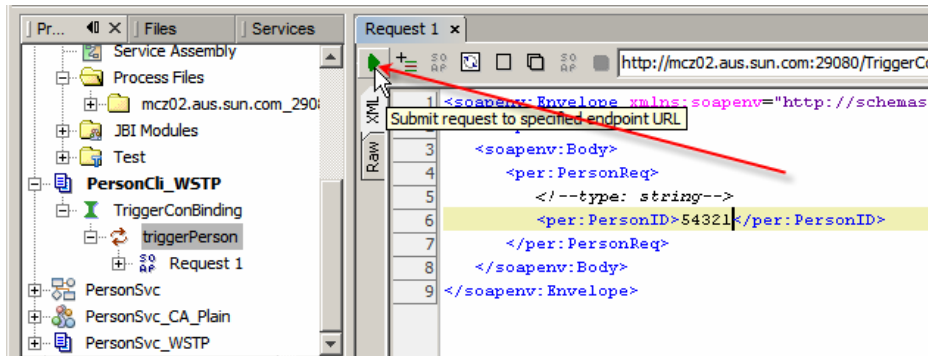


Figure 5.7.15 SOAP Request to be submitted to the PersonCli service

Observe the response – it should be like that shown in Figure 5.7.16.

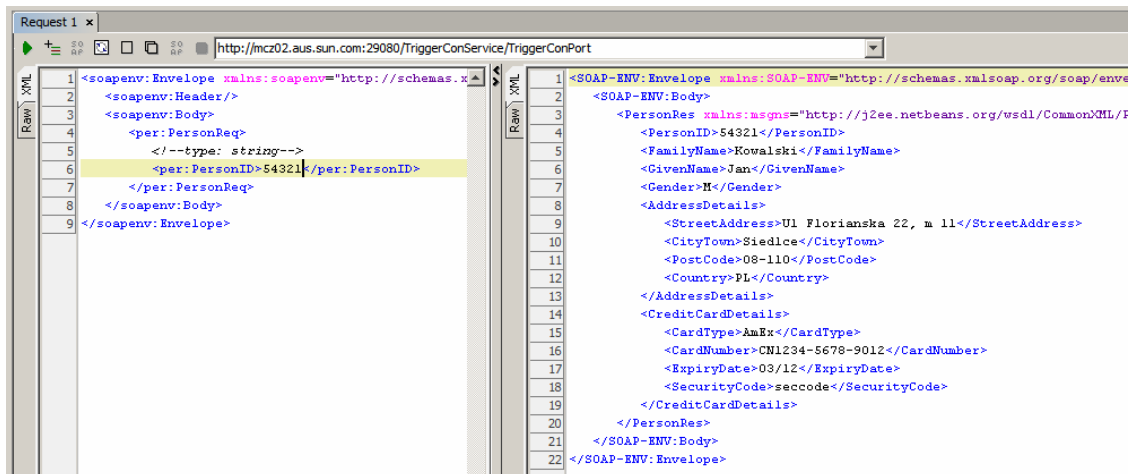


Figure 5.7.16 SOAP Response

We have the client / invoker (PersonCli) invoking the service / provider, PersonSvc. The end-to-end project works, as does the BPEL logic in the client and the service implementations.

Note that we did not do anything about security policies, nor did we even mention them until now. SOAP requests and SOAP responses are exchanged “in the clear”, in plain text. Anybody eavesdropping on the wire can see the content of the messages.

Before proceeding to the next section undeploy both the invoker and the provider projects.

5.8 Person Service - SSL Server-side Authentication

One way to prevent eavesdropping on messages being exchange between the invoker and the provider is to encrypt the channel between the two. Since web services use the HTTP protocol one can use the Secure Sockets Layer (SSL) / Transport Layer Security (TLS) to encrypt the

channel. This is a common mechanism used for securing message exchange with electronic commerce sites in order to prevent intercept of credit card details and other sensitive commercial information submitted by purchasers.

There is a great deal to SSL /TLS. More than I am prepared to discuss in this section. I assume that the reader is either sufficiently familiar with the protocol's operation to not require elaboration, or that the reader does not care for the theory and will be satisfied with the practice as discussed here. All others are referred to the excellent book by Eric Rescorla, "SSL and TLS: Designing and Building Secure Systems", ISBN-10: 0201615983, for elaboration.

SSL with Server-side Authentication adds security to the message exchange in two ways.

By requiring the server to provide the X.509 Certificate, expected to be issued by a trusted Certification Authority (CA) for a specific Host, the client is able to assure itself that the FQDN of the server is the same as the FQDN of the host in the certificate, therefore no substitution of hosts took place. The client is also able to validate the server certificate by verifying the digital signature of the CA, if the certificate was issued by a well know CA. Signature verification ensures that the certificate was not tampered with and the FQDN of the host was not altered. If the certificate signature is not valid, the certificate FQDN host name is not the same as the server host name, the certificate is a self-signed certificate or the CA is not a well know CA, then client would typically reject the certificate and abort the SSL Handshake. It is possible that the FQDN of the server will not be the same as the FQDN in the certificate. This may be legitimate inside an enterprise. To prevent rejection of the certificate a custom Hostname Verifier class can be provided that resolves this discrepancy. This is beyond the scope of this text. To prevent rejection of a certificate issued by a non-well know CA one can add the CA's certificate to the Client's truststore and mark it as trusted. This will make all certificates issued by the CA trusted by extension. This can also be legitimate inside an enterprise or between enterprises that explicitly trust each other. A self-signed certificate can be made trusted, therefore acceptable, the same way as an unknown CA can be made trusted – by being added to the Client's truststore and marked as trusted.

By encrypting the channel over which messages travel both the Client and the Server ensure that message exchange can not be profitably eavesdropped upon.

These two ways are typically used together. It is possible, though not common, to use channel encryption without certificate exchange. An internal enterprise application concerned with channel security, but not with endpoint authentication, might do that. Endpoint authentication in such an application might be provided in some other way, for example by embedding credentials in messages themselves. Channel encryption will protect these credentials.

Before proceeding with development let's do some groundwork.

SSL Handshake can be logged to the server.log by adding "-Djavax.net.debug=ssl:handshake" to the GlassFish Application Server's JVM Options. Figure 5.8.1 illustrates this in the GlassFish Application Server Admin Console.



Figure 5.8.1 JVM Options

If you don't have this incantation in the JVM Options at both end of the SSL Handshake you will not be able to see the log of the handshake. Please add the JVM Option to both instances of the GlassFish Application Server, if you are using two as I am doing. My local instance runs on mcz02.aus.sun.com and my remote instance runs on orad1.ssc.

When looking at the JVM Options also note the names and locations of the keystore.jks and cacerts.jks, see Figure 5.8.2. These are the cryptographic stores GlassFish uses at runtime. We will work with both in the not too distant future.

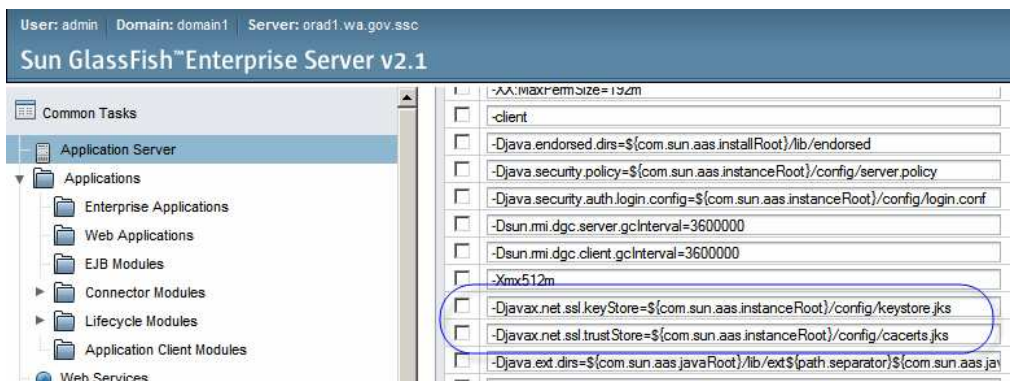


Figure 5.8.2 Cryptographic objects stores

Recall, or note, that at installation time the GlassFish installer generates the cryptographic key pair and the server certificate, which embeds the FQDN of the host on which it is being installed. This private key gets added to the keystore.jks, which by default resides in `<glassfishinstallroot>/domains/domain1/config`, under the alias `alias`. When requested to provide a certificate, as is the case when SSL with Server-side Authentication is configured, the server will return its X.509 certificate as part of the SSL Handshake. The client/invoke is expected to use that certificate to verify whether it “trusts” the server enough to allow the SSL Handshake to succeed. The client verifies that the certificate is “trusted”, that is it is either signed by a trusted certification authority (CA) or it is explicitly trusted, if it is a self-signed certificate, by there being a copy of it in the client’s trust store, typically `<glassfishinstallroot>/domains/domain1/config/cacerts.jks`.

Using a tool like “Portecle Key Manager”, <http://linux.softpedia.com/progDownload/Portecle-Download-3110.html>, inspect the keystore.jks. Note the presence of the one and only private key,

with the alias of `s1as`. Note, too, that its corresponding certificate is associated with the host `mcz02.aus.sun.com` - for you it will be the fully qualified name of the host on which you installed the GlassFish Application Server whose `keystore.jks` you are inspecting. Figure 5.8.3 illustrates this.

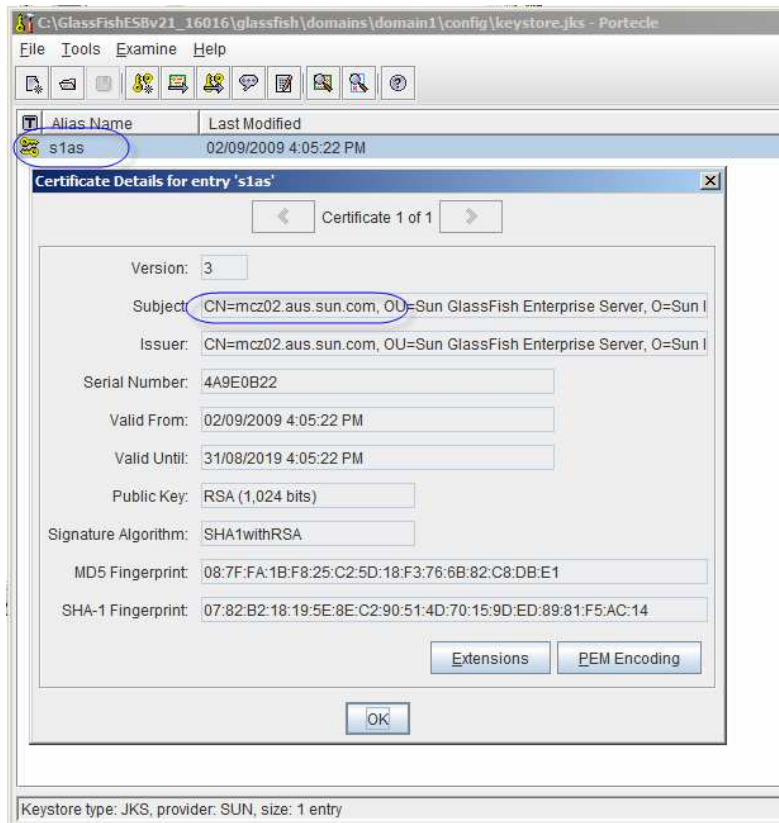


Figure 5.8.3 Private key, `s1as`, and its certificate

You can surmise from this that if `mcz02.aus.sun.com` is the server hosting the `PersonSvc` web service, and it is required to provide the client with a certificate, the certificate associated with the alias `a1as` will be provided to the client.

Note, also, that the Issuer of the certificate is the same as the Subject of the certificate in Figure 5.8.3. That makes this certificate a self-signed certificate. There is no separate Certification Authority which issued the certificate so it is unlikely that this certificate will be trusted by any other host unless explicitly told to do so.

Let's interrupt the certificate discussion at this point. We will resume it later.

Let's create the `PersonSvc_CA_SSLServerAuth` Composite Application, drag the `PersonSvc BPEL Module` onto the `CASA` canvas, add and connect a `SOAP binding` and `Build`, much as was illustrated in Figures 5.7.1, 5.7.2 and 5.7.3.

Click on the "pencil and paper" icon to open properties of the `SOAP BC` and modify `Location URL` to a) use the `https` scheme instead of the `http` scheme, b) use the `FQDN` of the remote host (for me this will be `orad1.ssc`) and c) modify port number variable name from `HttpDefaultPort` to `HttpsDefaultPort`. For me, the modified URL will be:

https://orad1.ssc:\${HttpsDefaultPort}/casaService1/casaPort1

Figure 5.8.4 illustrates the key points.

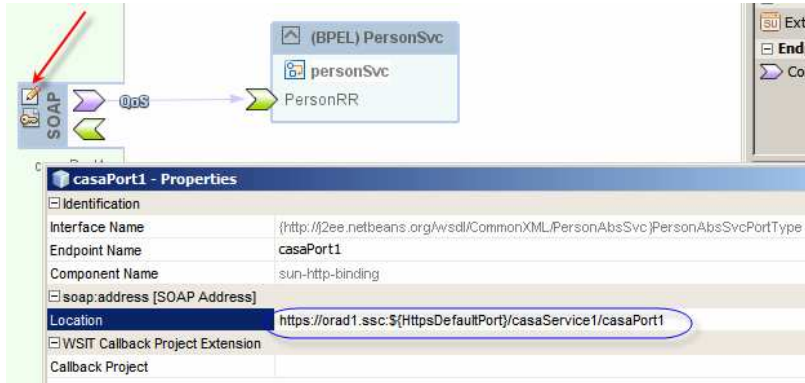


Figure 5.8.4 Modified Location property

Here comes a twist. The original WSDL does not use any security policies at all. In fact it can not because most security policies are applied to the concrete part of the WSDL and our original WSDL does not have a concrete part. By dragging the SOAP BC onto the CASAS canvas and connecting it to the BPEL Module we created a WSDL which imports our original WSDL and adds the concrete part. Explore the PersonSvc_CA_SSLServerAuth -> Process Files, Figure 5.8.5, and note the WSDL PersonSvc_CA_SSLServerAuth. Open this WSDL and look at the concrete part, Figure 5.8.5.

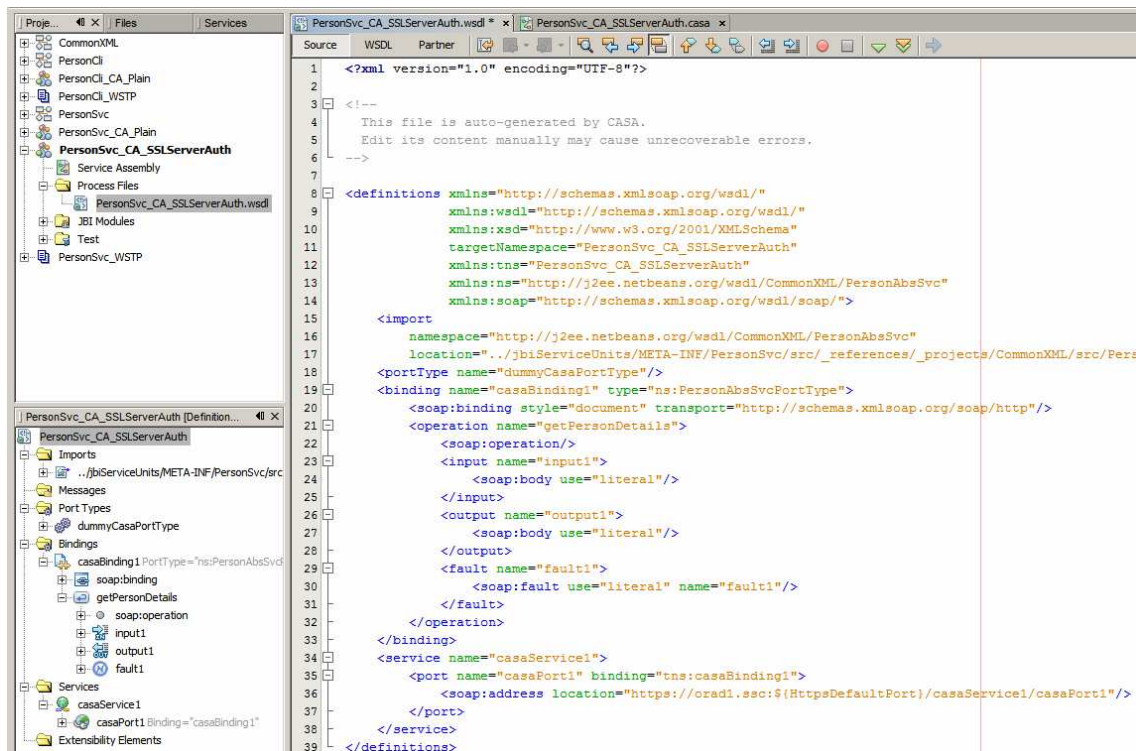


Figure 5.8.5 Concrete WSDL

Note that the imported WSDL location is relative to the location of this WSDL. At build time NetBeans will be able to resolve this but at runtime it will not. To make sure the project can deploy successfully we need to “import” the abstract WSDL from the CommonXML Project to this project’s “Process Files” folder. Right-click on the “process File” folder, choose “New” -> “External WSDL Document(s)”, locate the WSDL in the CommoXML/src folder, select it and import it into the CASA project. This should not be required but ... Figure 5.8.9 shows the project structure after this activity.

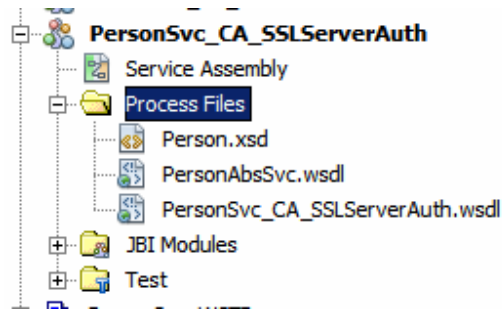


Figure 5.8.9 Project structure with abstract WSDL and XSD “imported”

Switch back to the CASA canvas, click the “paper with a key” icon and choose “Server Configuration”. Figure 5.8.10 illustrates this step.

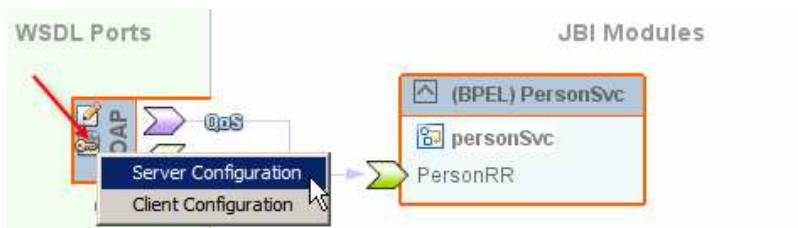


Figure 5.8.10 Edit Server Configuration

Check the “Secure Service” checkbox, choose “Transport Security (SSL)” from the Security Mechanism drop-down and click “Configure”, as shown in Figure 5.8.11.

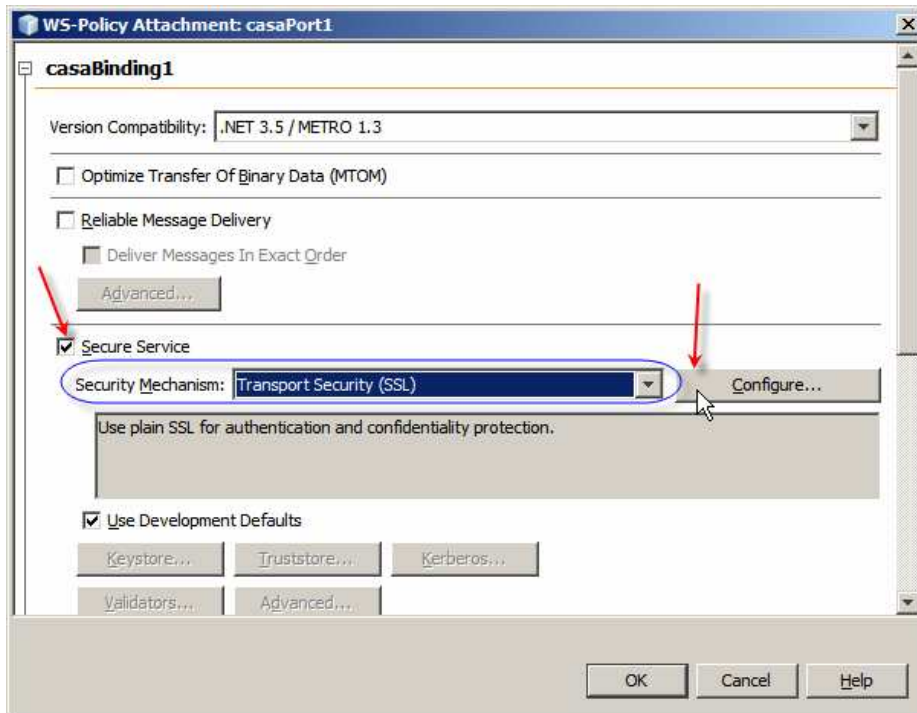


Figure 5.8.11 Enable and configure SSL / TLS channel security

Choose a suitable algorithm suite from the dropdown of supported algorithm suites. This selection does not guarantee that the specified algorithm suite will be used. Final algorithm suite is subject to negotiation between the client and the server and is settled during the SSL Handshake. Leave “Require Client Certificate” checkbox unchecked. We are configuring Server-side Authentication here so we don’t need client’s certificate. Figure 5.8.12 illustrates the dialogue box.

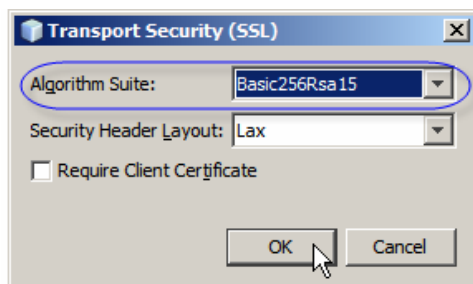


Figure 5.8.12 Choose algorithm suite

Dismiss the dialog and look again at the PersonSvc_CA_SSLServerAuth WSDL. Switch to the Source view and inspect the policy formulation.

Note, on line 20, that a policy reference attribute, shown in Figure 5.8.13, was added to the Biding.

```

18 <portType name="dummyCasaPortType"/>
19 <binding name="casaBinding1" type="ns:PersonAbsSvcPortType">
20 <wsp:PolicyReference URI="#casaBinding1Policy"/>
21 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
22 <operation name="getPersonDetails">

```

Figure 5.8.13 Policy Reference in the Binding

Were we to remove this attribute and its value no policy would be applied to the binding, even if one was there in the WSDL.

Scroll down and note the following, called out in Figure 5.8.14:

```

40 <wsp:Policy wsu:Id="casaBinding1Policy">
41 <wsp:ExactlyOne>
42 <wsp:All>
43 <wsam:Addressing wsp:Optional="false"/>
44 <sp:TransportBinding>
45 <wsp:Policy>
46 <sp:TransportToken>
47 <wsp:Policy>
48 <sp:HttpsToken RequireClientCertificate="false"/>
49 </wsp:Policy>
50 </sp:TransportToken>
51 <sp:Layout>
52 <wsp:Policy>
53 <sp:Lax/>
54 </wsp:Policy>
55 </sp:Layout>
56 <sp:IncludeTimestamp/>
57 <sp:AlgorithmSuite>
58 <wsp:Policy>
59 <sp:Basic256Rsa15/>
60 </wsp:Policy>
61 </sp:AlgorithmSuite>
62 </wsp:Policy>
63 </sp:TransportBinding>
64 <sp:Wss10/>
65 </wsp:All>
66 </wsp:ExactlyOne>
67 </wsp:Policy>

```

Figure 5.8.14 Policy elements of special interest

- Line 43: `<wsam:Addressing wsp:Optional="false"/>`
One would expect this to mean “WS-Addressing is mandatory”. The presence of the `wsp:Optional="false"` attribute make the implementation ignore WS-Addressing altogether. WS-Addressing is not required for SSL and is unrelated to SSL so let’s get rid of this element.
- Line 48: `<sp:HttpsToken RequireClientCertificate="false"/>`
This attribute indicates that Server-side authentication is used – no client certificate is required.
- Line 56: `<sp:IncludeTimestamp/>`
This element requires addition of a timestamp token. Timestamp token is not related to or required by SSL so delete this element.
- Line 59: `<sp:Basic256Rsa15/>`
This specified the preferred algorithm suite

The final policy, after removal of addressing and timestamp elements, is shown in Figure 5.8.15.

```
40 <wsp:Policy wsu:Id="casaBinding1Policy">
41   <wsp:ExactlyOne>
42     <wsp>All>
43       <sp:TransportBinding>
44         <wsp:Policy>
45           <sp:TransportToken>
46             <wsp:Policy>
47               <sp:HttpsToken RequireClientCertificate="false"/>
48             </wsp:Policy>
49           </sp:TransportToken>
50         <sp:Layout>
51           <wsp:Policy>
52             <sp:Lax/>
53           </wsp:Policy>
54         </sp:Layout>
55       <sp:AlgorithmSuite>
56         <wsp:Policy>
57           <sp:Basic256Rsa15/>
58         </wsp:Policy>
59       </sp:AlgorithmSuite>
60     </wsp:Policy>
61   </sp:TransportBinding>
62   <sp:Wss10/>
63 </wsp>All>
64 </wsp:ExactlyOne>
65 </wsp:Policy>
```

Figure 5.8.15 Final policy

Save and close the modified WSDL and Build, but do not Deploy the project.

This project will be deployed to the remote instance of the GlassFish Application Server – for me orad1.ssc. Before we can deploy the project we need to add the GlassFish instance to the NetBeans IDE so it can address it at deployment time. Switch to the Services Tab in the project explorer, right-click on the Servers node and choose Add Server. Figure 5.8.16 illustrates this. If you already have the remote server in the list, skip this.

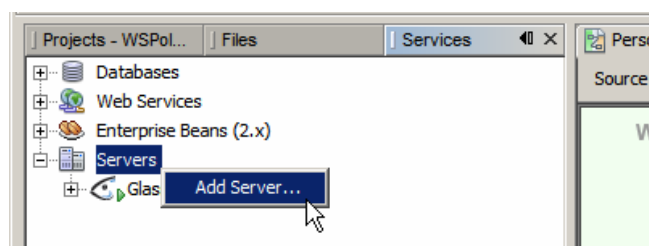


Figure 5.8.16 Add Server ...

Choose GlassFish v2, modify the name to reflect host name and click Next, as shown in Figure 5.8.17.

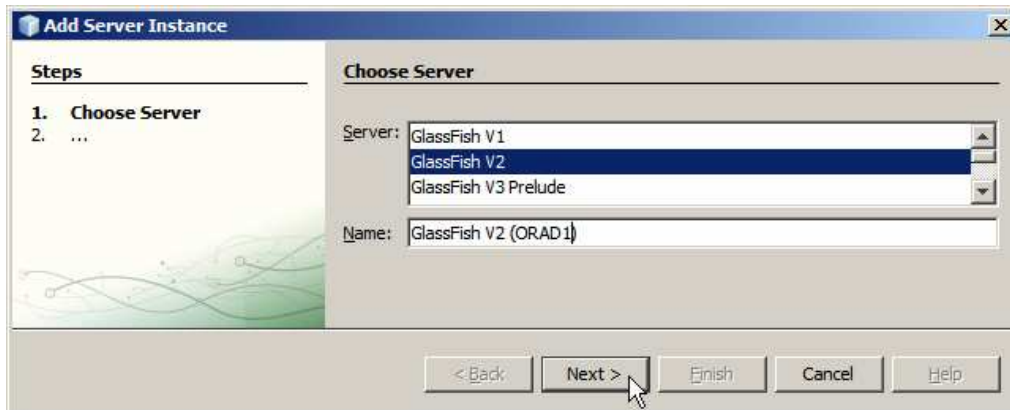


Figure 5.8.17 Choose server type and name it

Choose “Register Remote Domain” and click Next. Figure 5.8.18 illustrates key points.

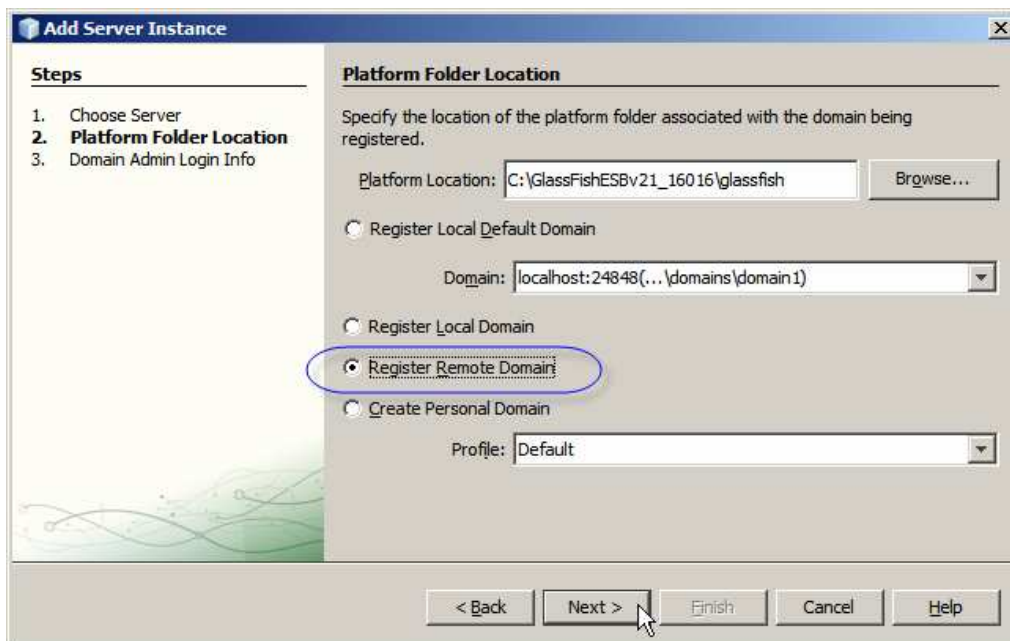


Figure 5.8.18 Choose to Register Remote Domain

Enter FQDN of the remote host, as specified for the server certificate when the remote GlassFish instance was installed, specify the appropriate administrative port number, if different from default, and click Next. Figure 5.8.19 illustrates this.



Figure 5.8.19 Specify host and port

Provide credentials and Finish.

If the remote GlassFish instance is running and correct configuration information was provided to NetBeans, the GlassFish instance will appear in the list of servers, as shown in Figure 5.8.20.

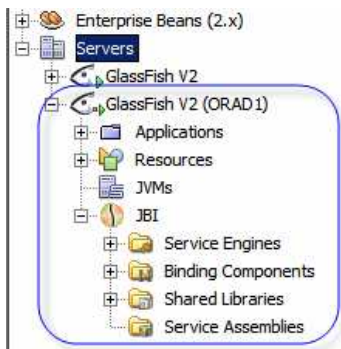


Figure 5.8.20 Remote GlassFish instance in NetBeans

Switch back to the Project Explorer's Project tab, right-click on the name PersonSvc_CA_SSLServerAuth, choose Properties, click the "Running Project" property and select the remote GlassFish instance as the deployment target. Figure 5.8.21 illustrates key points.

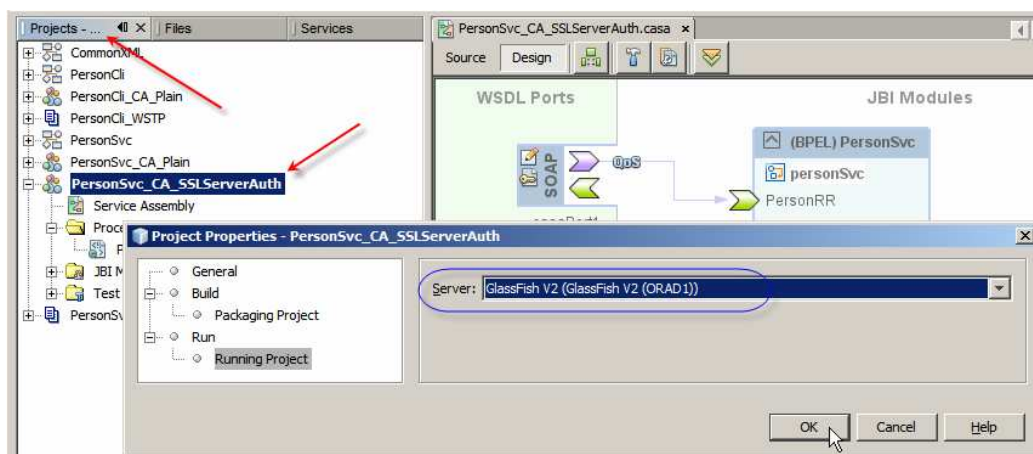


Figure 5.8.21 Choose the remote GlassFish instance as the deployment target

Build the project and Deploy it.

Because this is a project that requires SSL with Server-side Authentication we can use the SoapUI plugin to test the service and observe SSL Handshake at the server side. We will implement and exercise the PersonCli SSL with Server-side Authentication project a little later.

Let's create a "New Project" -> "Java EE" -> "Web Service Testing Project", named PersonSvc_SSLServerAuth_WSDP, using the WSDL location from the CASA SOAP BC's Location Property, replacing the `${HttpsDefaultPort}` with the appropriate port number, For me this will be:

```
https://orad1.ssc:29181/casaService1/casaPort1?WSDL
```

As the project is created, a dialogue box my pop up asking you to accept remote GlassFish instance's certificate, similar to what I saw for orad1, Figure 5.8.22.



Figure 5.8.22 Accept remote GlassFish instance's certificate

This will happen once, the first time a reference is made to the remote host. Thereafter NetBeans will trust the certificate and will not ask for confirmation.

Once the project is created add a New Request to the `getPersonDetails` interface, modify PersonID to 342312 and submit the request. Figure 5.8.23 illustrates the request.

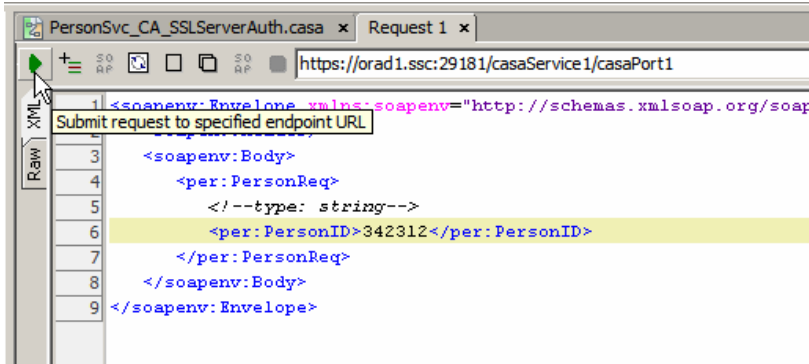


Figure 5.8.23 SOAP Request

Observe a SOAP Response response. Click on the “SSL Info” tab and observe the orad1.ssc’s certificate, figure 5.8.24.

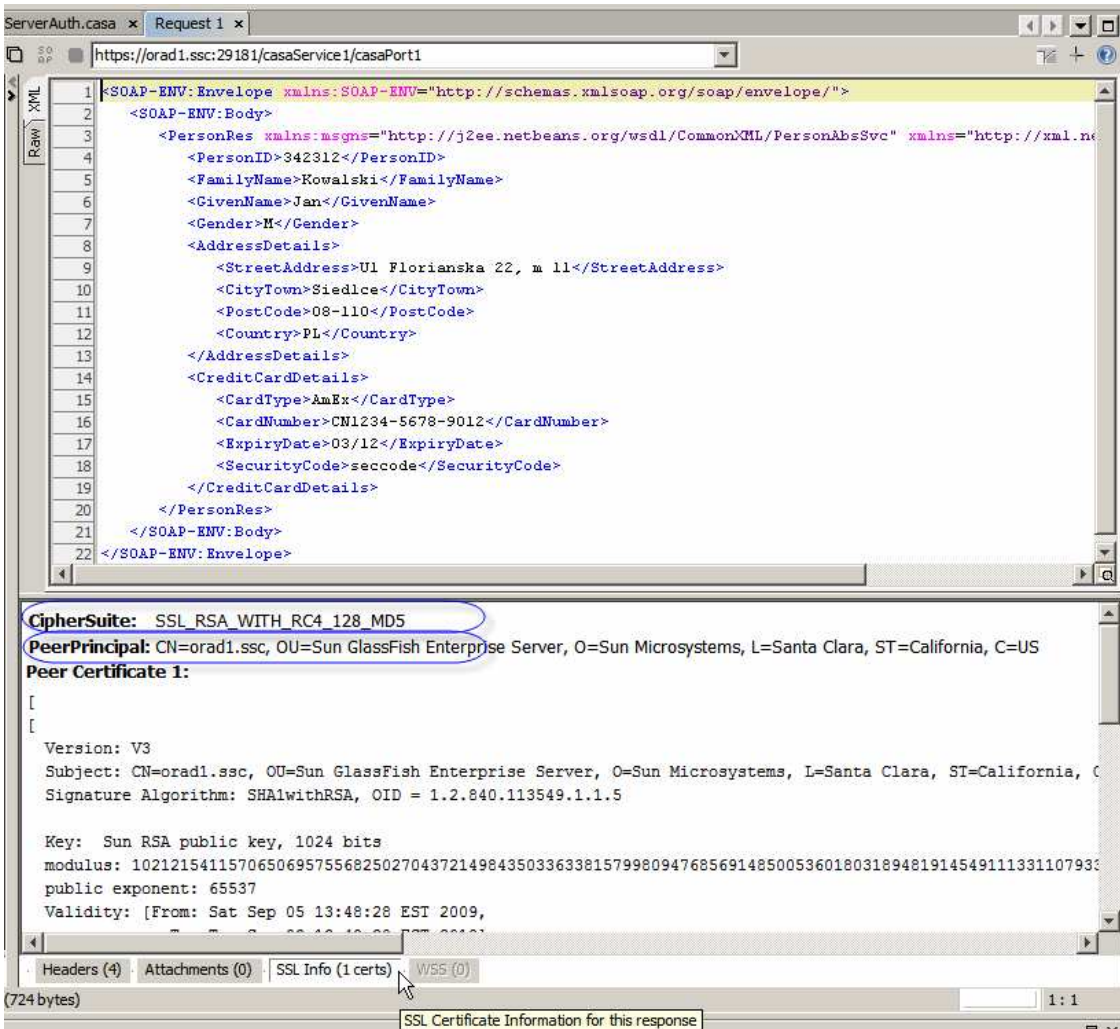


Figure 5.8.24 SOAP Response and orad1.ssc’s certificate

The SSL Handshake was successful. Let's look at selected lines from the server.log of the remote GlassFish instance to see how the SSL Handshake looked like there. Listing 5.8.1 shows just key lines.

Listing 5.8.1 Key lines from the SSL Handshake log

```
[#|2009-09-07T13:51:55.324+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=25;_ThreadName=SelectorThread
29181-1;|
Using SSLEngineImpl. [#]
[#|2009-09-07T13:51:55.326+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
httpSSLWorkerThread-29181-1, READ: SSL v2, contentType = Har
149|#]
[#|2009-09-07T13:51:55.327+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
*** ClientHello, TLSv1 [#]
...
[#|2009-09-07T13:51:55.337+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=
ad-29181-1;|
Session ID: [#]
[#|2009-09-07T13:51:55.337+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|{ }|#]
[#|2009-09-07T13:51:55.337+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Cipher Suites: [SSL_RSA_WITH_RC4_128_MD5, SSL_RSA_WITH_RC4_128_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA, TLS_ECDH_ECDSA_WITH_RC4_128_SHA,
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA, TLS_ECDH_RSA_WITH_RC4_128_SHA,
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA, TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, TLS_ECDHE_RSA_WITH_RC4_128_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA, SSL_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA, TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA, TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA, SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,
SSL_RSA_WITH_DES_CBC_SHA, SSL_DHE_RSA_WITH_DES_CBC_SHA, SSL_DHE_DSS_WITH_DES_CBC_SHA,
SSL_RSA_EXPORT_WITH_RC4_40_MD5, SSL_RSA_EXPORT_WITH_DES40_CBC_SHA,
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA, SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA,
SSL_RSA_WITH_NULL_MD5, SSL_RSA_WITH_NULL_SHA, TLS_ECDH_ECDSA_WITH_NULL_SHA,
TLS_ECDH_RSA_WITH_NULL_SHA, TLS_ECDHE_ECDSA_WITH_NULL_SHA, TLS_ECDHE_RSA_WITH_NULL_SHA,
SSL_DH_anon_WITH_RC4_128_MD5, TLS_DH_anon_WITH_AES_128_CBC_SHA,
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA, SSL_DH_anon_WITH_DES_CBC_SHA,
TLS_ECDH_anon_WITH_RC4_128_SHA, TLS_ECDH_anon_WITH_AES_128_CBC_SHA,
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA, SSL_DH_anon_EXPORT_WITH_RC4_40_MD5,
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA, TLS_ECDH_anon_WITH_NULL_SHA,
TLS_KRB5_WITH_RC4_128_SHA, TLS_KRB5_WITH_RC4_128_MD5, TLS_KRB5_WITH_3DES_EDE_CBC_SHA,
TLS_KRB5_WITH_3DES_EDE_CBC_MD5, TLS_KRB5_WITH_DES_CBC_SHA, TLS_KRB5_WITH_DES_CBC_MD5,
TLS_KRB5_EXPORT_WITH_RC4_40_SHA, TLS_KRB5_EXPORT_WITH_RC4_40_MD5,
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA, TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5] [#]
[#|2009-09-07T13:51:55.338+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_
ad-29181-1;|
Compression Methods: { } [#]
[#|2009-09-07T13:51:55.338+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|0|#]
...
```

SSL Engine starts processing

Start SSL Handshake

Client Hello message received

No Session ID – need new session

Client is willing to accept any of the cipher suites listed

Client will not use compression

```

[#|2009-09-07T13:51:55.338+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Crteate new session
%% Created: [Session-19, SSL_RSA_WITH_RC4_128_MD5|#]
[#|2009-09-07T13:51:55.339+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Server sends Hello message
*** ServerHello, TLSv1|#]
Session ID generated
...
[#|2009-09-07T13:51:55.349+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Session ID: [#]
Server chose the cipher suite
[#|2009-09-07T13:51:55.349+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|{74, 164, 131, 91, 63, 222, 251, 80, 243, 87, 244, 51, 122, 138, 49, 114, 24,
244, 67, 8, 250, 124, 74, 146, 191, 69, 3, 249, 26, 3, 159, 81}|#]
[#|2009-09-07T13:51:55.350+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Cipher Suite: SSL_RSA_WITH_RC4_128_MD5|#]
[#|2009-09-07T13:51:55.350+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Compression Method: 0|#]
[#|2009-09-07T13:51:55.350+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
***|#]
Server sends its certificate
[#|2009-09-07T13:51:55.350+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Cipher suite: SSL_RSA_WITH_RC4_128_MD5|#]
[#|2009-09-07T13:51:55.350+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
*** Certificate chain|#]
[#|2009-09-07T13:51:55.351+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
chain [0] = [
[
Version: V3
Subject: CN=orad1.ssc, OU=Sun GlassFish Enterprise Server, O=Sun Microsystems,
L=Santa Clara, ST=California, C=US
Signature Algorithm: SHA1withRSA, OID = 1.2.840.113549.1.1.5

Key: Sun RSA public key, 1024 bits
modulus:
102121541157065069575568250270437214984350336338157998094768569148500536018031894819145491
113311079336949399472032140208618356816163637202618727298710465300230334324506184949208474
446933178623253726110968175800084405517043740120715529443624373833850530287673442674333181
408072531272948503339758724948550982873
public exponent: 65537
Validity: [From: Sat Sep 05 13:48:28 EST 2009,
To: Tue Sep 03 13:48:28 EST 2019]
Issuer: CN=orad1.ssc, OU=Sun GlassFish Enterprise Server, O=Sun Microsystems, L=Santa
Clara, ST=California, C=US
SerialNumber: [ 4aa1df8c]

Certificate Extensions: 1
[1]: ObjectID: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [

```

```

0000: 9E 79 9C E9 59 86 34 8F   FD 75 09 F7 82 D0 82 CE   .y..Y.4..u.....
0010: BE 9A 44 EE                               ..D.
]
]

```

```

]
  Algorithm: [SHA1withRSA]
  Signature:
0000: 23 A7 FD 51 1F 81 9E 8C   34 3A 58 01 EF 5A 04 CD   #..Q....4:X..Z..
0010: AD 35 2C 67 17 40 3A B8   EA 19 37 DB B2 B3 C8 EA   .5,g.@:...7.....
0020: 5B 4F 0E 30 4E 9D 42 23   52 FE E8 53 44 8B 64 21   [O.ON.B#R..SD.d!
0030: CF 5F EE 07 D5 60 1E F2   1B EA 68 99 E4 BB 6C 89   ._...`....h...l.
0040: 02 21 1D A5 AE 6C 26 14   8C 92 02 92 E3 C1 74 56   .!...l&.....tV
0050: 6A 69 96 8E 2D 1E 7D 6C   52 5E 99 38 20 8B 19 C4   ji...-..lR^.8 ...
0060: 52 11 89 B3 73 D0 6C 61   B2 DB BF CA 58 0A 3A 5D   R...s.la....X.:]
0070: 40 81 97 CC 3F 60 A6 1E   B5 D6 60 8A C6 6B B6 F6   @...?`.....`..k..

```

```

]|#]
  [#|2009-09-07T13:51:55.352+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
*** ServerHelloDone|#]
  [#|2009-09-07T13:51:55.352+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
httpSSLWorkerThread-29181-1, WRITE: TLSv1 Handshake, length = 794|#]
  [#|2009-09-07T13:51:55.524+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
httpSSLWorkerThread-29181-1, READ: TLSv1 Handshake, length = 134|#]
  [#|2009-09-07T13:51:55.527+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
*** ClientKeyExchange, RSA PreMasterSecret, TLSv1|#]
  [#|2009-09-07T13:51:55.528+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
SESSION KEYGEN:|#]
  [#|2009-09-07T13:51:55.528+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
PreMaster Secret:|#]
  [#|2009-09-07T13:51:55.528+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
0000: |#]
  [#|2009-09-07T13:51:55.528+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|03 |#]
...
  [#|2009-09-07T13:51:55.543+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
CONNECTION KEYGEN:|#]
  [#|2009-09-07T13:51:55.543+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Client Nonce:|#]
  [#|2009-09-07T13:51:55.544+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
0000: |#]
...

```

Hello exchange done, perform secrets exchange

```

[#|2009-09-07T13:51:55.554+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Server Nonce:|#]
[#|2009-09-07T13:51:55.554+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
0000:|#]
[#|2009-09-07T13:51:55.554+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|4A|#]
...
[#|2009-09-07T13:51:55.564+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Master Secret:|#]
[#|2009-09-07T13:51:55.564+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
0000:|#]
[#|2009-09-07T13:51:55.564+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|6F|#]
...
[#|2009-09-07T13:51:55.577+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|Client MAC write Secret:|#]
[#|2009-09-07T13:51:55.577+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
0000:|#]
[#|2009-09-07T13:51:55.577+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|C3|#]
...
[#|2009-09-07T13:51:55.582+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
Server MAC write Secret:|#]
[#|2009-09-07T13:51:55.582+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
0000:|#]
[#|2009-09-07T13:51:55.583+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|6F|#]
...
[#|2009-09-07T13:51:55.587+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|Client write key:|#]
[#|2009-09-07T13:51:55.587+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
0000:|#]
[#|2009-09-07T13:51:55.587+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|C2|#]
...
[#|2009-09-07T13:51:55.592+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|Server write key:|#]
[#|2009-09-07T13:51:55.592+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|

```

```

0000: [#]
[#|2009-09-07T13:51:55.593+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|BF|#]
...
[#|2009-09-07T13:51:55.598+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|... no IV used for this cipher|#]
[#|2009-09-07T13:51:55.598+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
httpSSLWorkerThread-29181-1, READ: TLSv1 Change Cipher Spec, length = 1|#]
[#|2009-09-07T13:51:55.599+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
httpSSLWorkerThread-29181-1, READ: TLSv1 Handshake, length = 32|#]
[#|2009-09-07T13:51:55.599+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
*** Finished|#]
[#|2009-09-07T13:51:55.599+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
verify_data: {|#]
[#|2009-09-07T13:51:55.599+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|58|#]
...
[#|2009-09-07T13:51:55.603+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
httpSSLWorkerThread-29181-1, WRITE: TLSv1 Change Cipher Spec, length = 1|#]
[#|2009-09-07T13:51:55.604+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
*** Finished|#]
[#|2009-09-07T13:51:55.604+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
verify_data: {|#]
[#|2009-09-07T13:51:55.604+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|220|#]
...
[#|2009-09-07T13:51:55.609+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
httpSSLWorkerThread-29181-1, WRITE: TLSv1 Handshake, length = 32|#]
[#|2009-09-07T13:51:55.609+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=35;_ThreadName=httpSSLWorkerThre
ad-29181-1;|
% Cached server session: [Session-19, SSL_RSA_WITH_RC4_128_MD5]|#]
[#|2009-09-07T13:51:55.662+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=36;_ThreadName=httpSSLWorkerThre
ad-29181-0;|
---[HTTP request]---|#]
...

```

Successful SSL Handshake begin data transfer

The listing shows key points in the SSL Handshake, as seen at the server. The client view will be looked at shortly. The SoapUI plugin does not produce a log I can see, however it received the server certificate as can be seen in Figure 5.8. SoapUI plugin did not object to the certificate as

not being trusted. It seems that SoapUI will accept any certificate. It is a testing tool, after all, not an application used to exchange real data.

Let's now create a composite application, PersonCli_CA_SSLServerAuth, for the client side. Drag the PersonCli BPEL module onto the CASA canvas and click Build. Right-click on the name of the project and create a "New" -> "External WSDL Document(s)", providing the endpoint URL exposed by the PersonSvc_CA_SSLServerAuth composite application, with the suffix "?WSDL". Accept the certificate if the dialog box, like that shown in Figure 5.8.22 pops up.

Right-click on the WSDL Ports swim line and choose "Load WSDL Port". Accept the one and only port and click Build to build the project. These steps are illustrated in Figures 5.7.8 through 5.7.14.

Click on the "paper and key" icon on the SOAP BC and choose Client Configuration, as shown in Figure 5.8.25.

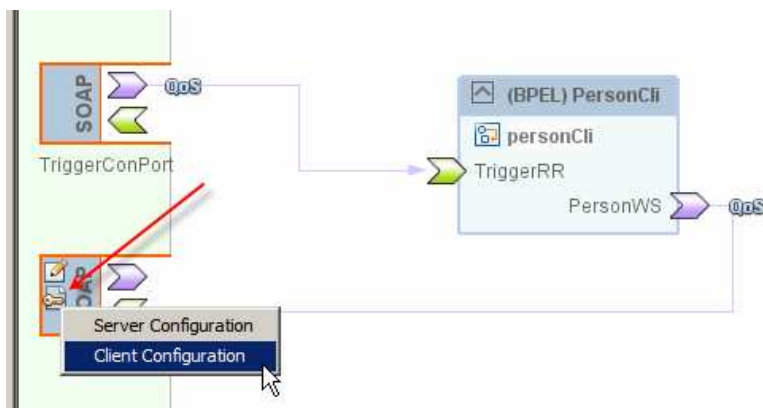


Figure 5.8.25 Client policy configuration

Note, as shown in Figure 5.8.26, that there are no specific configuration options for SSL with Server-side Authentication.

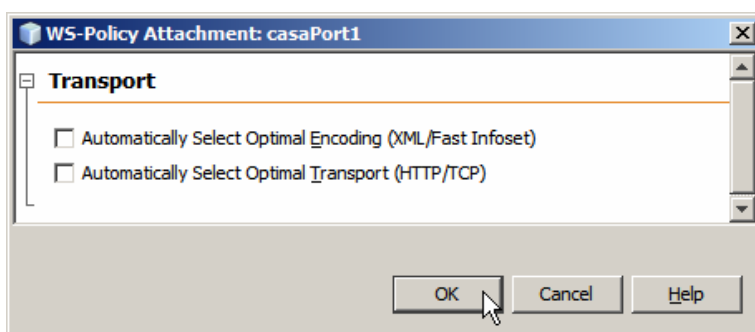


Figure 5.8.26 No SSL configuration options at the client side

Build the project.

Attempt to deploy the project to the local GlassFish instance. The expectation is that deployment will fail with an unenlightening message like that shown in Listing 5.8.2.

Listing 5.8.2 Deployment error messages

```
ERROR: Successful execution of Deploy:
G:\GlassFishESBv21Projects\WSPolicyExploration\PersonCli_CA_SSLServerAuth/dist/PersonCli_CA_SSLServerAuth.zip
WARNING: (JBIMA0404) Deployment of service assembly PersonCli_CA_SSLServerAuth succeeded partially; some service units failed to deploy.
  * Component: sun-http-binding
    ERROR: (SOAPBC_DEPLOY_2) HTTPBC-E00201: Deployment failed.
javax.wsdl.WSDLException: WSDLException (at /definitions/import):
faultCode=OTHER_ERROR: Unable to resolve imported document at
'https://orad1.ssc:29181/PersonSvc_CA_SSLServerAuth-sun-http-binding/PersonAbsSvc.wsdl', relative to
'file:/C:/GlassFishESBv21_16016/glassfish/domains/domain1/jbi/service-assemblies/PersonCli_CA_SSLServerAuth.1/PersonCli_CA_SSLServerAuth-sun-http-binding/sun-http-binding/orad1.ssc_29181/casaService1/casaPort1.wsdl':
javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to
find valid certification path to requested target
G:\GlassFishESBv21Projects\WSPolicyExploration\PersonCli_CA_SSLServerAuth\nbproject\build-impl.xml:201: Deployment failure.
BUILD FAILED (total time: 2 seconds)
```

What happened? The deployer attempted to start the composite application and the SOAP Binding Component in the PersonCli_CA_SSLServerAuth on mcz02.aus.sun.com attempted to connect to the SOAP BC in the PersonSvc_CA_SSLServerAuth on orad1.ssc. Since the GlassFish instance on mcz02.aus.sun.com does not know about the certificate returned by the GlassFish instance on orad1.ssc it rejected it with a rude message.

Listing 5.8.3 shows selected messages from the mcz02.aus.sun.com's server.log, relating to the SSL Handshake failure.

Listing 5.8.3 Client-side SSL Handshake

```
[#|2009-09-07T18:18:18.343+1000|INFO|sun-appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLServerAuth-sun-http-binding;|
%% No cached client session|#]
[#|2009-09-07T18:18:18.343+1000|INFO|sun-appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLServerAuth-sun-http-binding;|
*** ClientHello, TLSv1|#]
...
[#|2009-09-07T18:18:18.343+1000|INFO|sun-appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLServerAuth-sun-http-binding;|
Session ID:  |#]
[#|2009-09-07T18:18:18.343+1000|INFO|sun-appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLServerAuth-sun-http-binding;|{|}|#]
[#|2009-09-07T18:18:18.343+1000|INFO|sun-appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLServerAuth-sun-http-binding;|
Cipher Suites: [SSL_RSA_WITH_RC4_128_MD5, SSL_RSA_WITH_RC4_128_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA, SSL_RSA_WITH_3DES_EDE_CBC_SHA,
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA, SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,
SSL_RSA_WITH_DES_CBC_SHA, SSL_DHE_RSA_WITH_DES_CBC_SHA, SSL_DHE_DSS_WITH_DES_CBC_SHA,
```

```

SSL_RSA_EXPORT_WITH_RC4_40_MD5, SSL_RSA_EXPORT_WITH_DES40_CBC_SHA,
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA, SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA|#]
[#|2009-09-07T18:18:18.343+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
Compression Methods: { |#]
...
[#|2009-09-07T18:18:18.343+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
PersonCli_CA_SSLServerAuth-sun-http-binding, WRITE: TLSv1 Handshake, length = 73|#]
[#|2009-09-07T18:18:18.343+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
PersonCli_CA_SSLServerAuth-sun-http-binding, WRITE: SSLv2 client hello message, length =
98|#]
[#|2009-09-07T18:18:18.640+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
PersonCli_CA_SSLServerAuth-sun-http-binding, READ: TLSv1 Handshake, length = 794|#]
[#|2009-09-07T18:18:18.640+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
*** ServerHello, TLSv1|#]
...
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
Session ID: |#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|{74, 164, 193, 203, 245, 191, 123, 38, 91, 15, 20, 255, 117,
203, 207, 130, 17, 102, 76, 59, 54, 207, 0, 9, 12, 125, 143, 33, 189, 59, 111, 26}|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
Cipher Suite: SSL_RSA_WITH_RC4_128_MD5|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
Compression Method: 0|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
***|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
%% Created: [Session-3, SSL_RSA_WITH_RC4_128_MD5]|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
** SSL_RSA_WITH_RC4_128_MD5|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
*** Certificate chain|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
chain [0] = [
[
Version: V3

```

Subject: CN=**oradl.ssc**, OU=Sun GlassFish Enterprise Server, O=Sun Microsystems,
L=Santa Clara, ST=California, C=US
Signature Algorithm: SHA1withRSA, OID = 1.2.840.113549.1.1.5

Key: Sun RSA public key, 1024 bits
modulus:
102121541157065069575568250270437214984350336338157998094768569148500536018031894819145491
113311079336949399472032140208618356816163637202618727298710465300230334324506184949208474
446933178623253726110968175800084405517043740120715529443624373833850530287673442674333181
408072531272948503339758724948550982873
public exponent: 65537
Validity: [From: Sat Sep 05 13:48:28 EST 2009,
To: Tue Sep 03 13:48:28 EST 2019]
Issuer: CN=oradl.ssc, OU=Sun GlassFish Enterprise Server, O=Sun Microsystems, L=Santa
Clara, ST=California, C=US
SerialNumber: [4aaldf8c]

Certificate Extensions: 1
[1]: ObjectID: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 9E 79 9C E9 59 86 34 8F FD 75 09 F7 82 D0 82 CE .y..Y.4..u.....
0010: BE 9A 44 EE ..D.
]
]

Algorithm: [SHA1withRSA]
Signature:
0000: 23 A7 FD 51 1F 81 9E 8C 34 3A 58 01 EF 5A 04 CD #..Q....4:X..Z..
0010: AD 35 2C 67 17 40 3A B8 EA 19 37 DB B2 B3 C8 EA .5,g.@:...7.....
0020: 5B 4F 0E 30 4E 9D 42 23 52 FE E8 53 44 8B 64 21 [O.ON.B#R..SD.d!
0030: CF 5F EE 07 D5 60 1E F2 1B EA 68 99 E4 BB 6C 89 _...`....h...l.
0040: 02 21 1D A5 AE 6C 26 14 8C 92 02 92 E3 C1 74 56 .!...l&.....tV
0050: 6A 69 96 8E 2D 1E 7D 6C 52 5E 99 38 20 8B 19 C4 ji...-..lR^8 ...
0060: 52 11 89 B3 73 D0 6C 61 B2 DB BF CA 58 0A 3A 5D R...s.la....X.:]
0070: 40 81 97 CC 3F 60 A6 1E B5 D6 60 8A C6 6B B6 F6 @...?`....`..k..

]|#]
...
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|, SEND TLSv1 ALERT:|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|fatal,|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|description = certificate_unknown|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
PersonCli_CA_SSLServerAuth-sun-http-binding, WRITE: TLSv1 Alert, length = 2|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
PersonCli_CA_SSLServerAuth-sun-http-binding, called closeSocket()|#]
[#|2009-09-07T18:18:18.656+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=81;_ThreadName=PersonCli_CA_SSLS
erverAuth-sun-http-binding;|
PersonCli_CA_SSLServerAuth-sun-http-binding, handling exception:
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path
building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to
find valid certification path to requested target|#]

```

[#|2009-09-07T18:18:18.671+1000|SEVERE|sun-
appserver2.1|com.sun.jbi.httpsoapbc.HttpSoapBindingDeployer|_ThreadID=81;_ThreadName=Perso
nCli_CA_SSLServerAuth-sun-http-binding;_RequestID=c5817253-ec00-4d70-9d29-
218bfc560ba3;|HTTPBC-E00201: Deployment failed. javax.wsdl.WSDLException: WSDLException
(at /definitions/import): faultCode=OTHER_ERROR: Unable to resolve imported document at
'https://orad1.ssc:29181/PersonSvc_CA_SSLServerAuth-sun-http-binding/PersonAbsSvc.wsdl',
relative to 'file:/C:/GlassFishESBv21_16016/glassfish/domains/domain1/jbi/service-
assemblies/PersonCli_CA_SSLServerAuth.1/PersonCli_CA_SSLServerAuth-sun-http-binding/sun-
http-binding/orad1.ssc_29181/casaService1/casaPort1.wsdl':
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path
building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to
find valid certification path to requested target
javax.jbi.JBIException: javax.wsdl.WSDLException: WSDLException (at /definitions/import):
faultCode=OTHER_ERROR: Unable to resolve imported document at
'https://orad1.ssc:29181/PersonSvc_CA_SSLServerAuth-sun-http-binding/PersonAbsSvc.wsdl',
relative to 'file:/C:/GlassFishESBv21_16016/glassfish/domains/domain1/jbi/service-
assemblies/PersonCli_CA_SSLServerAuth.1/PersonCli_CA_SSLServerAuth-sun-http-binding/sun-
http-binding/orad1.ssc_29181/casaService1/casaPort1.wsdl':
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path
building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to
find valid certification path to requested target
    at
com.sun.jbi.httpsoapbc.ServiceUnitImpl.createEndpoints(ServiceUnitImpl.java:601)
    at com.sun.jbi.httpsoapbc.ServiceUnitImpl.deploy(ServiceUnitImpl.java:201)
    at
com.sun.jbi.httpsoapbc.HttpSoapBindingDeployer.deploy(HttpSoapBindingDeployer.java:146)
    at
com.sun.jbi.framework.ServiceUnitOperation.process(ServiceUnitOperation.java:177)
    at com.sun.jbi.framework.Operation.run(Operation.java:104)
    at java.lang.Thread.run(Thread.java:619)
Caused by: javax.wsdl.WSDLException: WSDLException (at /definitions/import):
faultCode=OTHER_ERROR: Unable to resolve imported document at
'https://orad1.ssc:29181/PersonSvc_CA_SSLServerAuth-sun-http-binding/PersonAbsSvc.wsdl',
relative to 'file:/C:/GlassFishESBv21_16016/glassfish/domains/domain1/jbi/service-
assemblies/PersonCli_CA_SSLServerAuth.1/PersonCli_CA_SSLServerAuth-sun-http-binding/sun-
http-binding/orad1.ssc_29181/casaService1/casaPort1.wsdl':
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path
building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to
find valid certification path to requested target
    at com.ibm.wsdl.xml.WSDLReaderImpl.parseImport(WSDLReaderImpl.java:561)
    at com.ibm.wsdl.xml.WSDLReaderImpl.parseDefinitions(WSDLReaderImpl.java:331)
    at com.ibm.wsdl.xml.WSDLReaderImpl.readWSDL(WSDLReaderImpl.java:2324)
    at com.ibm.wsdl.xml.WSDLReaderImpl.readWSDL(WSDLReaderImpl.java:2288)
    at com.ibm.wsdl.xml.WSDLReaderImpl.readWSDL(WSDLReaderImpl.java:2341)
    at com.ibm.wsdl.xml.WSDLReaderImpl.readWSDL(WSDLReaderImpl.java:2249)
    at com.ibm.wsdl.xml.WSDLReaderImpl.readWSDL(WSDLReaderImpl.java:2211)
    at
com.sun.jbi.wsdlvalidator.impl.ValidatingWSDLReaderImpl.readWSDL(ValidatingWSDLReaderImpl.
java:88)
    at
com.sun.jbi.wsdlvalidator.impl.ValidatingWSDLReaderImpl.readWSDL(ValidatingWSDLReaderImpl.
java:95)
    at
com.sun.jbi.wsdlvalidator.impl.ValidatingWSDLReaderImpl.readWSDL(ValidatingWSDLReaderImpl.
java:95)
    at
com.sun.jbi.wsdlvalidator.impl.ValidatingWSDLReaderImpl.readWSDL(ValidatingWSDLReaderImpl.
java:95)
    at
com.sun.jbi.httpsoapbc.ServiceUnitImpl.createEndpoints(ServiceUnitImpl.java:486)
    ... 5 more
Caused by: javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException:
PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
    at com.sun.net.ssl.internal.ssl.Alerts.getSSLException(Alerts.java:174)

```

```

    at com.sun.net.ssl.internal.ssl.SSLSocketImpl.fatal(SSLSocketImpl.java:1611)
    at com.sun.net.ssl.internal.ssl.Handshaker.fatalSE(Handshaker.java:187)
    at com.sun.net.ssl.internal.ssl.Handshaker.fatalSE(Handshaker.java:181)
    at
com.sun.net.ssl.internal.ssl.ClientHandshaker.serverCertificate(ClientHandshaker.java:1035
)
    at
com.sun.net.ssl.internal.ssl.ClientHandshaker.processMessage(ClientHandshaker.java:124)
    at com.sun.net.ssl.internal.ssl.Handshaker.processLoop(Handshaker.java:516)
    at com.sun.net.ssl.internal.ssl.Handshaker.process_record(Handshaker.java:454)
    at com.sun.net.ssl.internal.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:884)
    at
com.sun.net.ssl.internal.ssl.SSLSocketImpl.performInitialHandshake(SSLSocketImpl.java:1112
)
    at
com.sun.net.ssl.internal.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1139)
    at
com.sun.net.ssl.internal.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1123)
    at sun.net.www.protocol.https.HttpsClient.afterConnect(HttpsClient.java:434)
    at
sun.net.www.protocol.https.AbstractDelegateHttpsURLConnection.connect(AbstractDelegateHttp
sURLConnection.java:166)
    at
sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java:1049)
    at
sun.net.www.protocol.https.HttpsURLConnectionImpl.getInputStream(HttpsURLConnectionImpl.ja
va:234)
    at java.net.URL.openStream(URL.java:1010)
    at com.ibm.wsdl.util.StringUtils.getContentAsStream(StringUtils.java:184)
    at com.ibm.wsdl.xml.WSDLReaderImpl.parseImport(WSDLReaderImpl.java:442)
    ... 16 more
Caused by: sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid
certification path to requested target
    at sun.security.validator.PKIXValidator.doBuild(PKIXValidator.java:285)
    at sun.security.validator.PKIXValidator.engineValidate(PKIXValidator.java:191)
    at sun.security.validator.Validator.validate(Validator.java:218)
    at
com.sun.net.ssl.internal.ssl.X509TrustManagerImpl.validate(X509TrustManagerImpl.java:126)
    at
com.sun.net.ssl.internal.ssl.X509TrustManagerImpl.checkServerTrusted(X509TrustManagerImpl.
java:209)
    at
com.sun.net.ssl.internal.ssl.X509TrustManagerImpl.checkServerTrusted(X509TrustManagerImpl.
java:249)
    at
com.sun.net.ssl.internal.ssl.ClientHandshaker.serverCertificate(ClientHandshaker.java:1014
)
    ... 30 more
Caused by: sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
    at
sun.security.provider.certpath.SunCertPathBuilder.engineBuild(SunCertPathBuilder.java:174)
    at java.security.cert.CertPathBuilder.build(CertPathBuilder.java:238)
    at sun.security.validator.PKIXValidator.doBuild(PKIXValidator.java:280)
    ... 36 more
|#]

```

The client commenced SSL Handshake by sending the Client Hello message nominating cryptographic algorithms and compression method it is willing to use, the server responded with the Server Hello message, nominating the selected cryptographic algorithm, session id and its

server certificate. The client looked at the certificate, did not find the CA that issued it in its truststore and aborted the SSL Handshake.

Recall from earlier discussion that for a self-signed certificate, which the one returned by orad1.ssc is, it must be explicitly imported into the client's GlassFish instance's cacerts.jks truststore. When I look at the client's (mcz02) GlassFish instance's cacert.jks I don't see the certificate that corresponds to orad1.ssc. Figure 5.8.27.

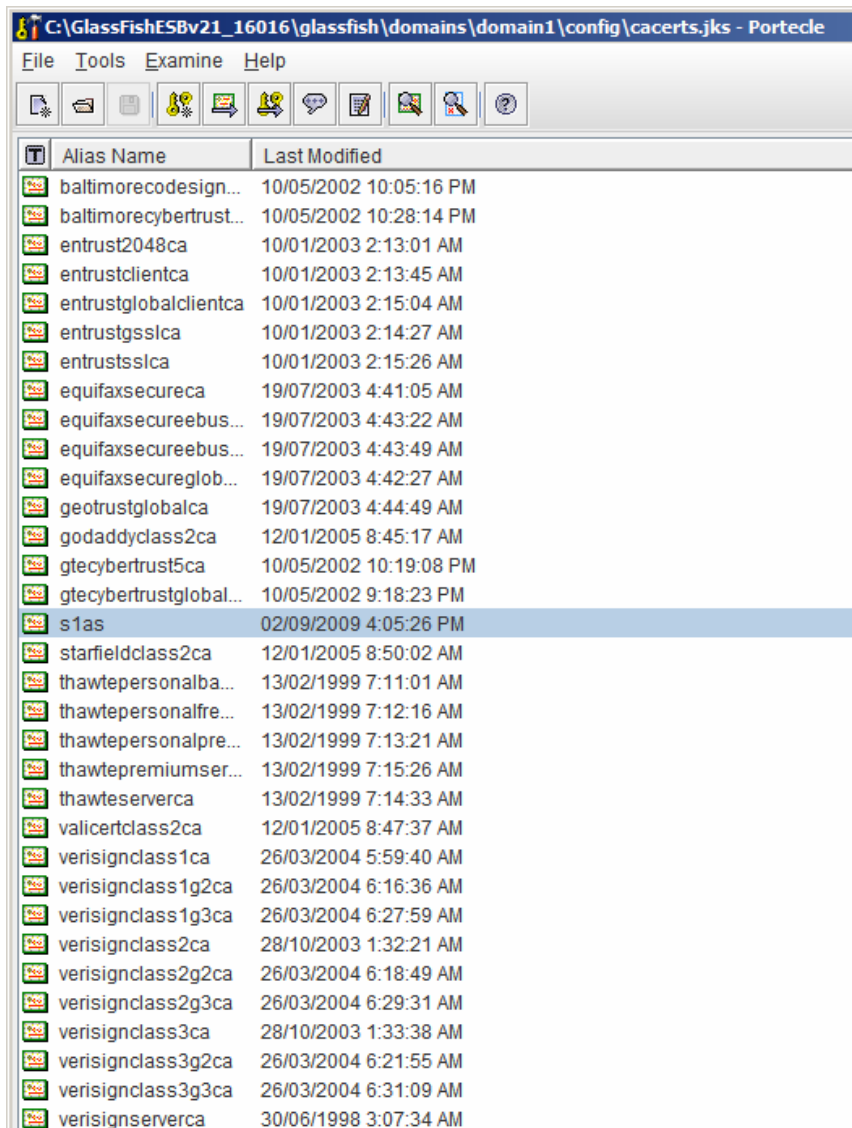


Figure 5.8.27 Default GlassFish list of trusted certificates

We need to extract the orad1.ssc's certificate from its keystore.jks and import it into the mcz02's cacerts.jks.

The steps to extract a certificate from a keystore using the Portecle tool are discussed in the next few paragraphs. If you have the certificate of the remote host, as you might, or use another tool to work with keystores, skip past this section.

I transferred the truststore.jks and the cacerts.jks from orad1.ssc to the machine on which I have Portacle installed so I can easily manipulate them.

Open the remote host's (orad1 for me) truststore.jks with Portacle. Password, by default, is changeit. Figure 5.8.28 shows the keystore content and the content of the s1as certificate.

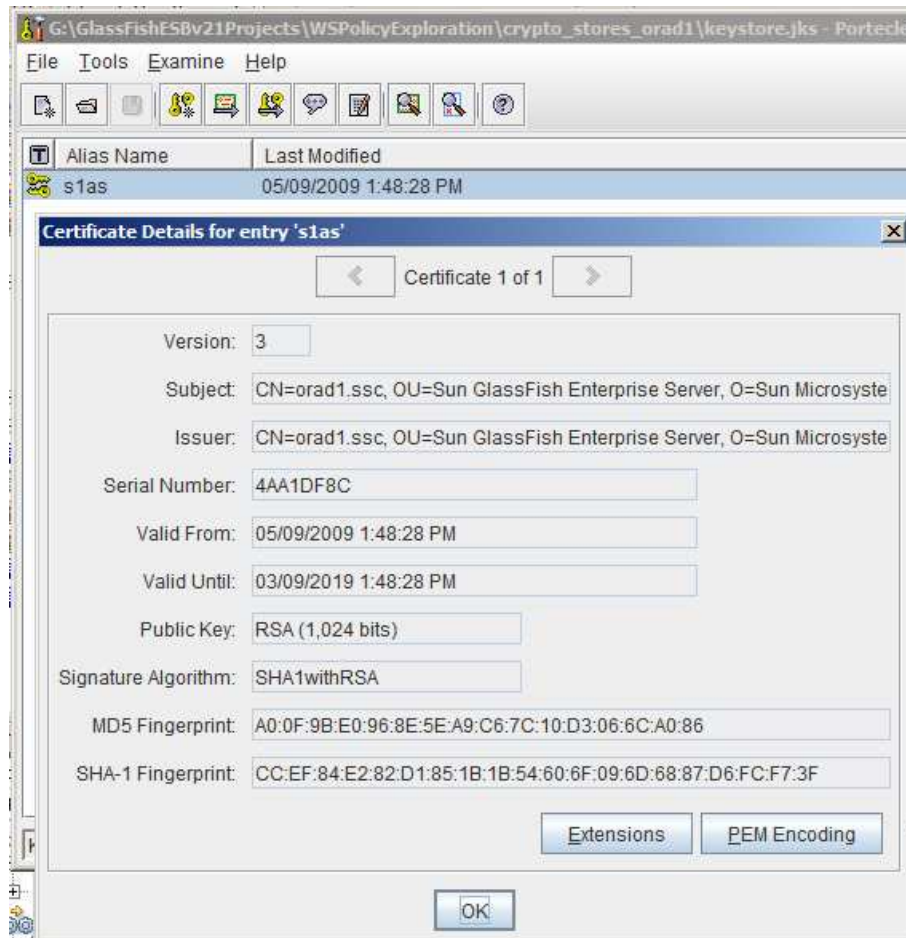


Figure 5.8.28 orad1's keystore with s1as private key and certificate

Right-click s1as key and choose "Export", as shown in Figure 5.8.29.

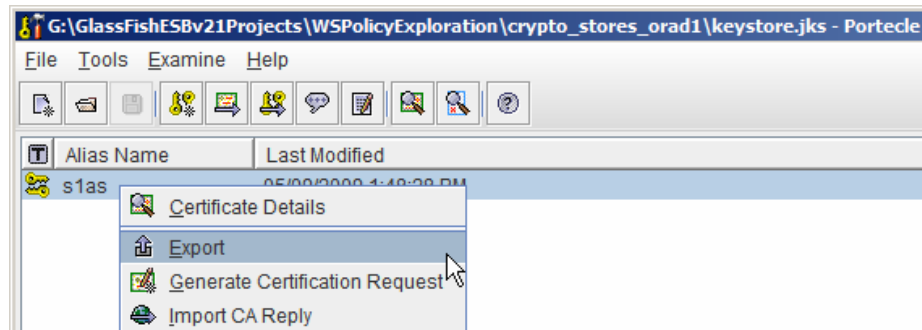


Figure 5.8.29 Trigger export of s1as certificate

Choose to export just the “head certificate” and store it in PEM Encode form – Figure 5.8.30. Since the certificate is a self-signed certificate it does not matter whether we export the head certificate (just the certificate itself) or the Certificate Chain (including all related CA certificates). The PEM Encoded, for Privacy Enhanced Mail (PEM) is basically a Base64 encoded binary certificate.

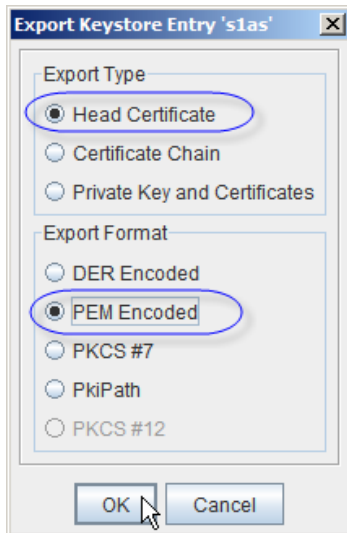


Figure 5.8.30 Choose export options

Complete the wizard by nominating the folder to which to save the certificate. By default the name of the file will be derived from the CN (Common Name) value in the certificate. Figure 5.8.31 illustrates this for my environment.

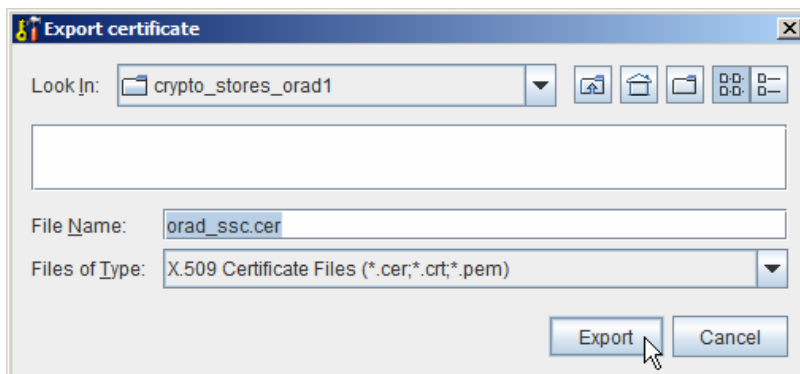


Figure 5.8.31 Save certificate to a file

If you happen to be on a Windows machine, as I am for the client-side development, you can inspect the certificate with windows tools. Merely double-click the certificate file and see what you see. What I see is shown in Figure 5.8.32.

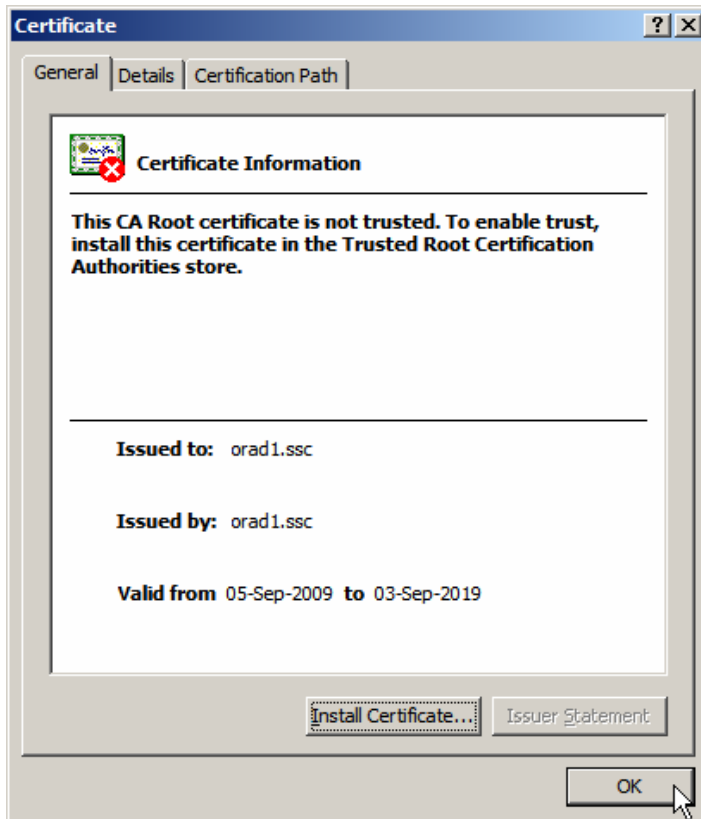


Figure 5.8.32 *orad1_ssc.cer* shown in Windows

Now let's import the certificate to the local GlassFish instance's, mcz02 for me, cacerts.jsk truststore. Figure 5.8.33 illustrates the first step in this process.

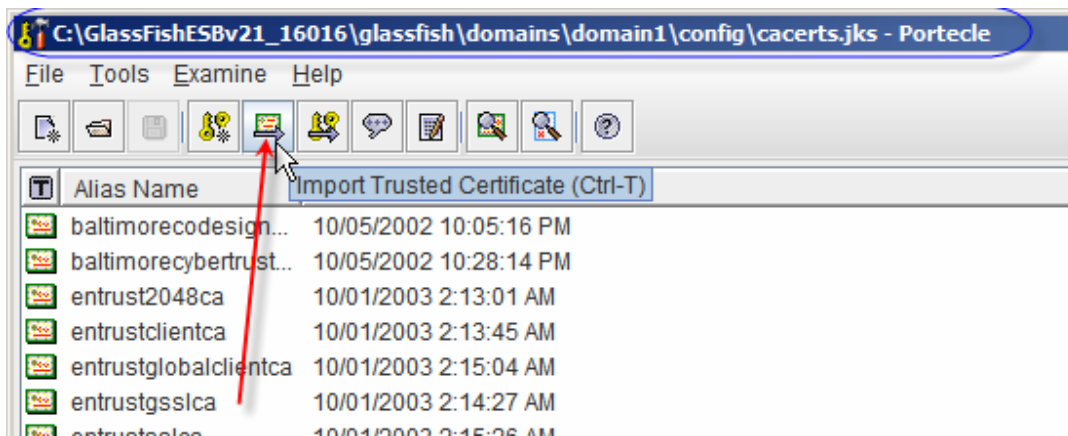


Figure 5.8.33 Start the certificate import process

Locate the certificate and select it for import, as is shown in Figure 5.8.34.

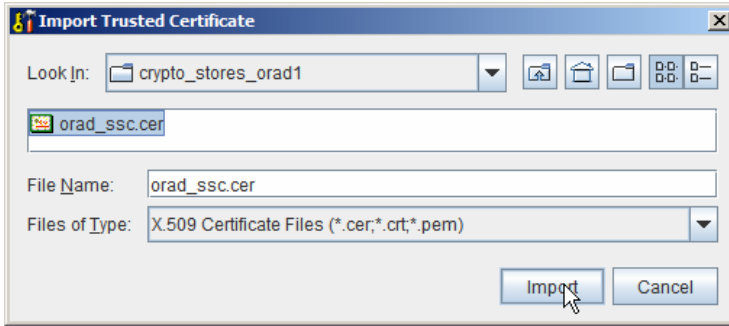


Figure 5.8.34 Select and import the certificate

This is a self-signed certificate so the tool will advise that it can not establish trust for the certificate, as shown in Figure 5.8.35. Acknowledge the message.

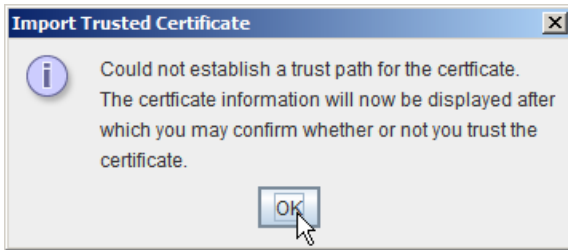


Figure 5.8.45 Trust path can not be established message

Then the certificate details will be shown, as can be seen in Figure 5.8.36. Acknowledge this by clicking OK.

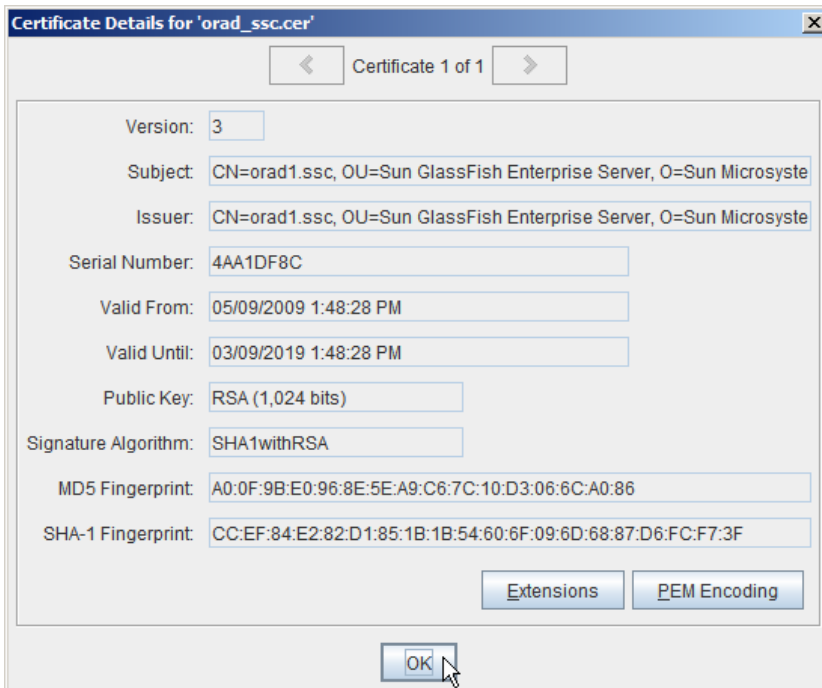


Figure 5.8.36 Certificate details

Finally, click the Yes button to accept the certificate as trusted, Figure 5.8.37, and accept the provided or modified certificate alias, Figure 5.8.38.

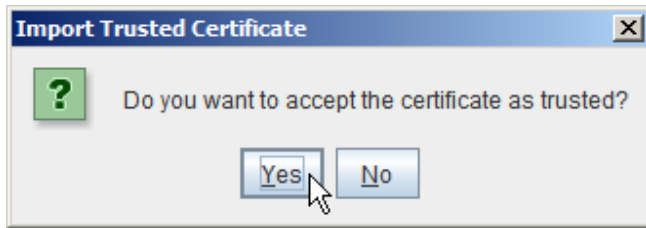


Figure 5.8.37 *Accept certificate as trusted*

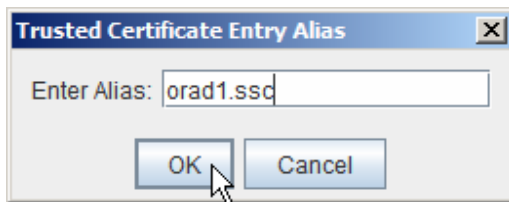


Figure 5.8.38 *Accept certificate alias*

Once this is done the certificate will be imported into the cacerts.jks truststore, as trusted certificate with alias of orad1.ssc. Figure 5.8.39 shows the final feedback.

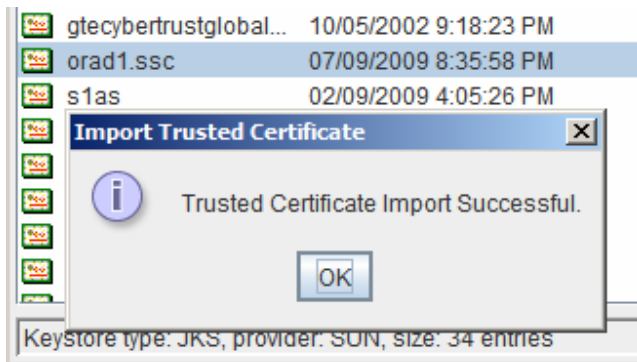


Figure 5.8.39 *Certificate was imported*

Exit from Portacle Key Manager, saving the modified cacerts.jks keystore on the way.

The GlassFish Application Server appear to be caching the truststore content. It is necessary to re-start GlassFish after corticated is imported.

We are ready to attempt to deploy the client application again. This time, since the remote GlassFish instance's certificate is in the cacerts.jks truststore, and is trusted, we should succeed. Listing 5.8.4 shows the feedback from the NetBeans IDE.

Listing 5.8.4 *Deployment successful*

```
run-jbi-deploy :
[undeploy-service-assembly]
  Undeploying a service assembly...
    host=localhost
    port=24848
```

```

        name=PersonCli_CA_SSLServerAuth
[deploy-service-assembly]
    Deploying a service assembly...
        host=localhost
        port=24848

file=G:\GlassFishESBv21Projects\WSPolicyExploration\PersonCli_CA_SSLServerAuth/dist/Person
Cli_CA_SSLServerAuth.zip
[start-service-assembly]
    Starting a service assembly...
        host=localhost
        port=24848
        name=PersonCli_CA_SSLServerAuth
run:
BUILD SUCCESSFUL (total time: 8 seconds)

```

The service implementation composite application, `PersonSvc_CA_SSLServerAuth`, is deployed to host `orad1.wa.gov.ssc`. The client implementation composite application, `PersonCli_CA_SSLServerAuth`, is deployed to `mcz02.aus.sun.com`.

Let's exercise the solution using the `PersonCli_WSTP` web service testing project by submitting the SoapUI request, as we did before. Listing 5.8.5 shows abbreviated trace of the SSL Handshake as seen on the client side.

Listing 5.8.5 Client-side SSL Handshake trace

```

[#|2009-09-07T20:52:16.500+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
%% Client cached [Session-1, SSL_RSA_WITH_RC4_128_MD5]|#]
[#|2009-09-07T20:52:16.500+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
%% Try resuming [Session-1, SSL_RSA_WITH_RC4_128_MD5] from port 4707|#]
[#|2009-09-07T20:52:16.500+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
*** ClientHello, TLSv1|#]
...
[#|2009-09-07T20:52:16.515+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Session ID:  |#]
[#|2009-09-07T20:52:16.515+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|{74, 164, 228,
147, 242, 12, 228, 182, 189, 239, 197, 106, 83, 181, 198, 176, 62, 55, 7, 142, 242, 27,
58, 223, 237, 12, 12, 62, 224, 73, 109, 208}|#]
[#|2009-09-07T20:52:16.515+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Cipher Suites: [SSL_RSA_WITH_RC4_128_MD5, SSL_RSA_WITH_RC4_128_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA, SSL_RSA_WITH_3DES_EDE_CBC_SHA,
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA, SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,
SSL_RSA_WITH_DES_CBC_SHA, SSL_DHE_RSA_WITH_DES_CBC_SHA, SSL_DHE_DSS_WITH_DES_CBC_SHA,

```



```

SSL_RSA_EXPORT_WITH_RC4_40_MD5, SSL_RSA_EXPORT_WITH_DES40_CBC_SHA,
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA, SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA|#]
[#|2009-09-07T20:52:16.515+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Compression Methods: {|#]
...
[#|2009-09-07T20:52:16.515+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
HTTPBPC-OutboundReceiver-2, WRITE: TLSv1 Handshake, length = 105|#]
[#|2009-09-07T20:52:16.843+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
HTTPBPC-OutboundReceiver-2, READ: TLSv1 Handshake, length = 74|#]
[#|2009-09-07T20:52:16.843+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
*** ServerHello, TLSv1|#]
...
[#|2009-09-07T20:52:16.890+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Session ID:|#]
[#|2009-09-07T20:52:16.890+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|{74, 164, 228,
147, 242, 12, 228, 182, 189, 239, 197, 106, 83, 181, 198, 176, 62, 55, 7, 142, 242, 27,
58, 223, 237, 12, 12, 62, 224, 73, 109, 208}|#]
[#|2009-09-07T20:52:16.890+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Cipher Suite: SSL_RSA_WITH_RC4_128_MD5|#]
[#|2009-09-07T20:52:16.890+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Compression Method: 0|#]
[#|2009-09-07T20:52:16.890+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
***|#]
[#|2009-09-07T20:52:16.890+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
CONNECTION KEYGEN:|#]
[#|2009-09-07T20:52:16.890+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Client Nonce:|#]
...
Server Nonce:|#]
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-

```

```

OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
0000:|#]
...
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Master Secret:|#]
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
0000:|#]
...
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Client MAC write Secret:|#]
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
0000:|#]
...
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|Server MAC write
Secret:|#]
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
0000:|#]
...
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
Client write key:|#]
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
0000:|#]
...
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|Server write
key:|#]
[#|2009-09-07T20:52:16.906+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
0000:|#]
...
...
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
... no IV used for this cipher|#]

```

```

[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
%% Server resumed [Session-1, SSL_RSA_WITH_RC4_128_MD5]|#]
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
HTTPBPC-OutboundReceiver-2, READ: TLSv1 Change Cipher Spec, length = 1|#]
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
HTTPBPC-OutboundReceiver-2, READ: TLSv1 Handshake, length = 32|#]
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
*** Finished|#]
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
verify_data: {|#]
...
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
HTTPBPC-OutboundReceiver-2, WRITE: TLSv1 Change Cipher Spec, length = 1|#]
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
*** Finished|#]
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
verify_data: {|#]
...
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
HTTPBPC-OutboundReceiver-2, WRITE: TLSv1 Handshake, length = 32|#]
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
HTTPBPC-OutboundReceiver-2, WRITE: TLSv1 Application Data, length = 323|#]
[#|2009-09-07T20:52:16.921+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
HTTPBPC-OutboundReceiver-2, WRITE: TLSv1 Application Data, length = 549|#]
[#|2009-09-07T20:52:17.171+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-
OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
HTTPBPC-OutboundReceiver-2, READ: TLSv1 Application Data, length = 856|#]
[#|2009-09-07T20:52:17.187+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=58;_ThreadName=HTTPBPC-

```

```

OutboundReceiver-2;Context=PersonCli_CA_SSLServerAuth-sun-http-binding-
{http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc}getPersonDetails;|
---[HTTP response 200]---|#]
...
*** a lot of stuff here - messages exchanged and so on ****
...
[#|2009-09-07T20:52:27.171+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=61;_ThreadName=Keep-Alive-
Timer;|
Keep-Alive-Timer, called close()|#]
[#|2009-09-07T20:52:27.171+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=61;_ThreadName=Keep-Alive-
Timer;|
Keep-Alive-Timer, called closeInternal(true)|#]
[#|2009-09-07T20:52:27.171+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=61;_ThreadName=Keep-Alive-
Timer;|
Keep-Alive-Timer|#]
[#|2009-09-07T20:52:27.171+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=61;_ThreadName=Keep-Alive-
Timer;|, SEND TLSv1 ALERT:  |#]
[#|2009-09-07T20:52:27.171+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=61;_ThreadName=Keep-Alive-
Timer;|warning,  |#]
[#|2009-09-07T20:52:27.171+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=61;_ThreadName=Keep-Alive-
Timer;|description = close_notify|#]
[#|2009-09-07T20:52:27.171+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=61;_ThreadName=Keep-Alive-
Timer;|
Keep-Alive-Timer, WRITE: TLSv1 Alert, length = 18|#]

```

Note, in Listing 5.8.5, that rather than establishing a new session, the two parties resumed an earlier SSL session. The client requested session resumption and the server agreed to resume the session. No certificate was sent from the server to the client.

When the session expires a complete SSL Handshake will be performed again, including supply of the server certificate to the client. Host validation and certificate trust path validation.

The end-to-end solution using SSL with Server-side Authentication works. Let's undeploy both composite applications in preparation for the next section – channel security using SSL with Mutual Authentication.

5.9 Person Service - SSL Mutual Authentication

***What is described in this section ought to work.
Note that at this point in time this configuration does not work so this
discussion is theoretical.***

SSL with Server-side Authentication is a good choice for enforcing message privacy as it travels between two end points. It is also a good choice if the client cares about authenticity of the server but the server does not use SSL to establish authenticity of the client. This is common in electronic commerce application where the client needs to make sure it is communicating with the expected server before providing credit card and similar information to it. Channel security

takes care of privacy for the credit card details and server certificate allows the client to validate the server. The commerce site uses credit card information to obtain the payment, activity which is completely unrelated to the message exchange between two endpoints. All that the commerce site cares about is that credit card information is valid and the payment can be exacted. It does not need to authenticate the client machine because it is not relevant to the transaction.

Still, there are situations where both the server and the client, in addition to maintaining message privacy through channel encryption, need to authenticate one another. SSL with Mutual Authentication can be used for this purpose. The SSL Handshake is modified in such a way that as well as server sending its certificate to the client for verification the client sends its certificate to the server for validation. Either side can abort the handshake if it is not happy with the other's certificate. Once the handshake completes successfully the channel is encrypted as normal and the same protocol operating applies (periodic change of keys, etc.).

In this section the end-to-end solution which uses secures the channel using SSL with Mutual Authentication will be developed and exercised.

As, hopefully, is clear by now, security policies, if any, can be applied to a service endpoint in a composite application without having to disturb or redevelop application logic. PersonSvc BPEL Module and PersonCli BPEL Module implement business logic. PersonSvc_CA_xxx and PersonCli_CA_xxx are used to configure varying policies.

To save ourselves the trouble of creating a composite application from scratch, not that is is a big deal, let's copy the PersonSvc_CA_SSLServerAuth composite application as PersonSvc_CA_SSLSMutualAuth. Figure 5.9.1 provides an illustration of this.

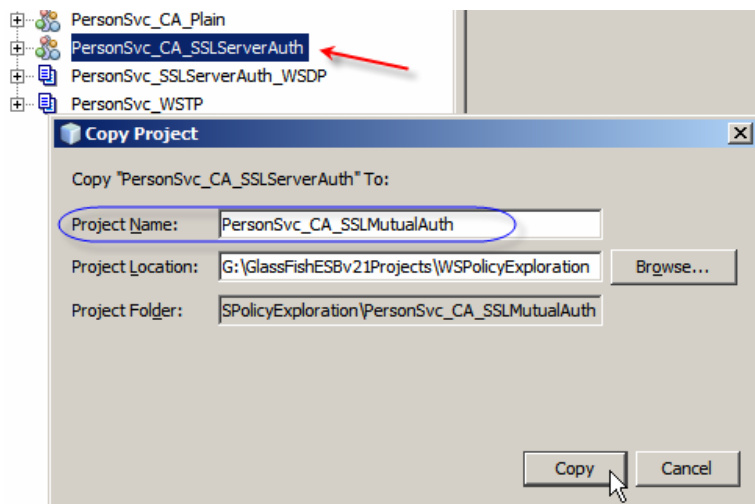


Figure 5.9.1 Clone PersonSvc_CA_SSLServerAuth as PersonSvc_CA_SSLSMutualAuth

Open “Server Configuration” properties pane as illustrated in Figure 5.9 2.

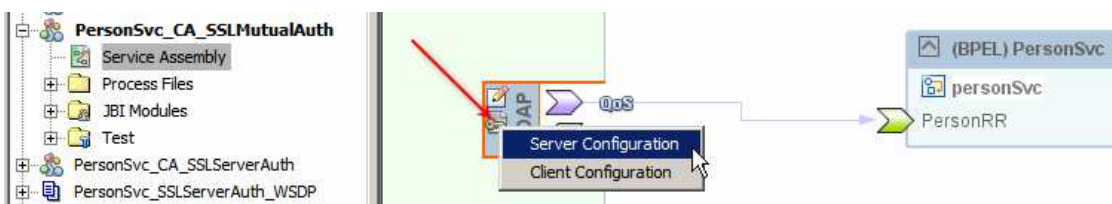


Figure 5.9.2 *Open Server Configuration Properties Pane*

Click the Configure button, check the “Require Client Certificate” checkbox and dismiss both dialogue boxes by clicking OK. Figure 5.9.3 illustrates key points.

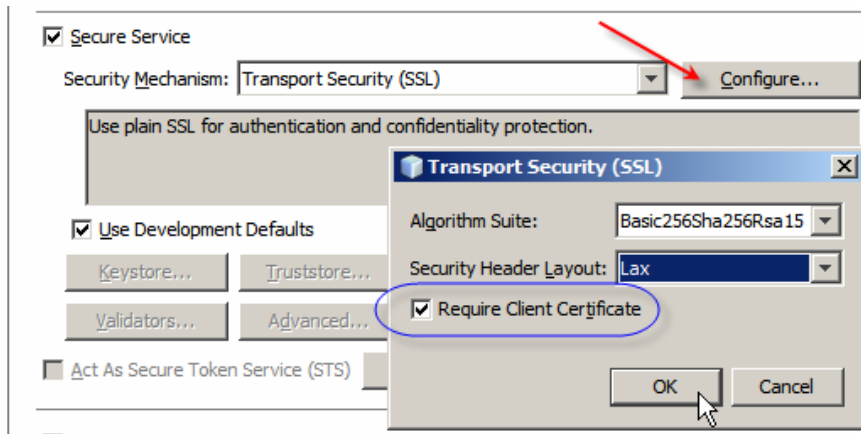


Figure 5.9.3 *Require Client Certificate*

Build and Deploy the application to the remote GlassFish instance.

Since the service WSDL has changed we can not re-use the PersonCli_CA_SSLServerAuth and trivially modify configuraitn before re-deploying, as we have done with the PersonSvc_CA_SSLServerAuth. This is because the composite application has a copy of the server WSDL, provided when we create a “New -> External WSDL Document(s)”. This copy of the WSDL is a copy of the old WSDL and there is no way to refresh it short of rdeleting and re-creating it from scratch. By the time we do that we might as well create a new PersonCli_CA_SSLSMutualAuth project form scratch. This is what we will do.

Create a “New Project” -> SOA -> “Composite Application”, named PersonCli_CA_SSLSMutualAuth.

Determine the WSDL URL for the PersonSvc_CA_SSLSMutualAuth service. For me this will be:

`https://orad1.ssc:29181/casaService1/casaPort1?WSDL`

Using this WSDL URL create a “New” -> “External WSDL Document(s)” in the new composite application project. Figure 5.9.4 illustrates the outcome of this process.

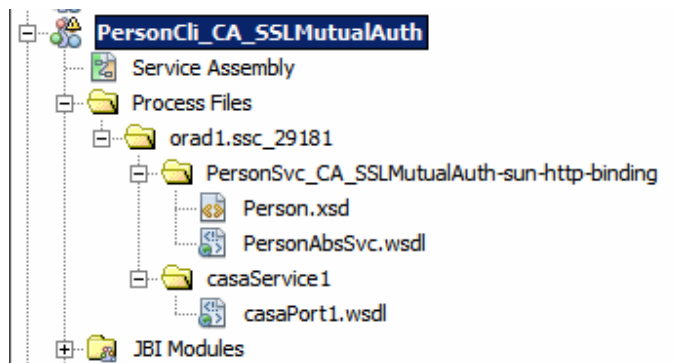


Figure 5.9.3 External WSDL in the composite application project

As we have done twice before, drag the PersonCli BPEL module onto the CASA and build the project. Notice two SOAP BCs on the canvas, connected to the BPEL Module. The Provider (TriggerCon) and the consumer (PersonSvc).

Open and inspect “Client Configuration” properties of the Consume SOAP BC, illustrated in figure 5.9.4. Notice that, as before, there are no SSL-related properties to configure.

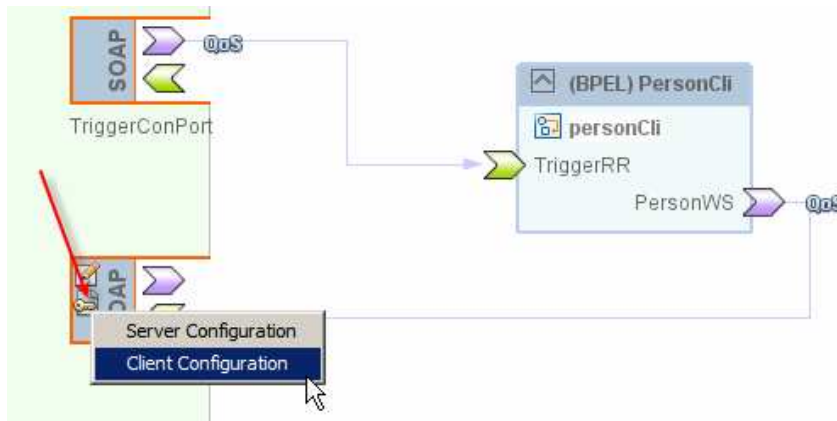


Figure 5.9.4 Open Client Configuration of the service consumer endpoint

We are expecting the SSL Handshake to call for client’s certificate. The only certificate the client’s (local) GlassFish instance know about is the one with the alias of s1as in the GlassFish instance’s keystore.jsk (<glassfish_root>/domains//domain1/config/keystore.jks).

Build and deploy the project to the local GlassFish instance. Inspecting the local server.log will show that the deployment process results in a SSL Handshake and that the SSL session established when we created a new external WSDL is resumed. SSL session gets resumed if it is between the same two hosts and it is still “fresh” enough.

Use the SOAP request in PersonCli_WSTP web service testing project to trigger this solution.

5.10 EJB-based Person Svc with No Channel Security

In this section the EJB-based web service provider and consumer will be built and exercised, in preparation for configuration of security policies in subsequent sections.

Listing 5.3.2 shows the Abstract WSDL that describes the interface to our service. In this section we will implement this service in Java as an EJB-based web service.

Create “New Project” -> “Java EE” -> “EJB Module”, choosing the remote GlassFish instance as the deployment target. Name this module EJBPersNoSecSvc.

In the new project create “New” -> “Web Service From WSDL”, naming the service EJBPersNoSecSvc, placing it in the package pkg.EJBPersNoSecSvc and using the CommonXML/PersonAbsSvc “Local WSDL File”.

Figure 5.10.1 shows the source of the service implementation, at this point in time, reformatted for better readability.

```

1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  package pkg.EJBPersonNoSecSvc;
7
8  import javax.ejb.Stateless;
9  import javax.jws.WebService;
10 import org.netbeans.j2ee.wsdl.commonxml.personabssvc.GetPersonDetailsFault;
11 import org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType;
12
13 /**
14  *
15  * @author mczapski
16  */
17 @WebService
18     (serviceName = "PersonAbsSvcService",
19     portName = "PersonAbsSvcPort",
20     endpointInterface = "org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType",
21     targetNamespace = "http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc",
22     wsdlLocation = "META-INF/wsdl/EJBPersonNoSecSvc/PersonAbsSvcWrapper.wsdl")
23 @Stateless
24 public class EJBPersonNoSecSvc implements PersonAbsSvcPortType {
25
26     public org.netbeans.xml.schema.person.PersonRes getPersonDetails
27         (org.netbeans.xml.schema.person.PersonReq msgPersonDetailsReq)
28         throws GetPersonDetailsFault {
29         //TODO implement this method
30         throw new UnsupportedOperationException("Not implemented yet.");
31     }
32
33 }
34

```

Figure 5.10.1 Skeleton service implementation

Replace lines 29-30 (in my source) with the java code shown in Listing 5.10.1.

Listing 5.10.1 Service implementation

```

System.out.println("\n%%%%%%%% PersonID: " + msgPersonDetailsReq.getPersonID());

org.netbeans.xml.schema.person.PersonRes sRes
    = new org.netbeans.xml.schema.person.PersonRes();
sRes.setPersonID(msgPersonDetailsReq.getPersonID());
sRes.setFamilyName("Doe");
sRes.setGivenName("John");
sRes.setGender("M");

org.netbeans.xml.schema.person.PersonRes.AddressDetails addr
    = new org.netbeans.xml.schema.person.PersonRes.AddressDetails();

addr.setStreetAddress("33 Berry Street");
addr.setCityTown("North Sydney");
addr.setStateProvince("NSW");
addr.setPostCode("2160");
addr.setCountry("Australia");
sRes.setAddressDetails(addr);

org.netbeans.xml.schema.person.PersonRes.CreditCardDetails card

```

```

        = new org.netbeans.xml.schema.person.PersonRes.CreditCardDetails();
card.setCardNumber("123-456-7689-0123");
card.setCardType("Passport");
card.setExpiryDate("01/21");
card.setSecurityCode((new java.util.Date()).toString());
sRes.setCreditCardDetails(card);

if (msgPersonDetailsReq.getPersonID().equalsIgnoreCase("FAULT")) {
    PersonFlt pFlt = new PersonFlt();
    pFlt.setPersonID(msgPersonDetailsReq.getPersonID());
    pFlt.setFaultDetail("Induced GetPersonDetailsFault");
    GetPersonDetailsFault sFlt
        = new GetPersonDetailsFault
            ("Induced GetpersonDetailsFault"
            ,pFlt);
    throw sFlt;
}
return sRes;

```

Right-click inside the source window and choose “Fix Imports” to resolve import-related issues.

Build and Deploy the project.

Create “New Project” -> “Java EE” -> “EJB Module”, named EJBPersonNoSecCli, to be deployed to the local GlassFish instance.

Create “New” -> “Web Service From WSDL”, named EJBPersonNoSecCli, in package pkg.EJBPersonNoSecCli, using WSDL from the CommonXML/TriggerCon.wsdl.

Create “New” -> “Web Service Client”. Click “Project”, browse to the EJBPersonNoSecSvc project and choose the web service implementation. Figure 5.10.2 illustrates this step.

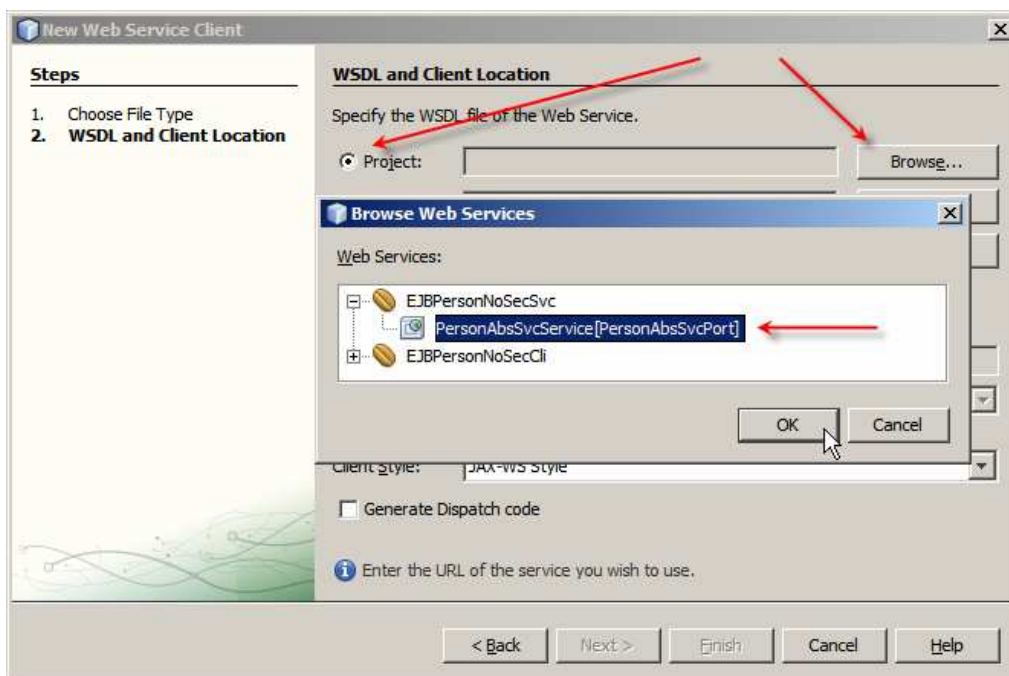


Figure 5.10.2 Choose web service implementation for the client to invoke

Replace the two lines of code, “//TODO ...” and “throw new Unsupported...” with the code shown in Listing 5.10.2.

Listing 5.10.2 Replacement code

```
org.netbeans.xml.schema.person.PersonReq cReq
    = new org.netbeans.xml.schema.person.PersonReq();
cReq.setPersonID(msgPersonDetailsReq.getPersonID());

org.netbeans.xml.schema.person.PersonRes cRes
    = new org.netbeans.xml.schema.person.PersonRes();
cRes.setPersonID(msgPersonDetailsReq.getPersonID());

// invoke service here

return cRes;
```

The source, reformatted for better readability, is shown in Figure 5.10.3.



```
17 @WebService
18     (serviceName = "TriggerConService",
19     portName = "TriggerConPort",
20     endpointInterface = "org.netbeans.j2ee.wsdl.commonxml.triggercon.TriggerConPortType",
21     targetNamespace = "http://j2ee.netbeans.org/wsdl/CommonXML/TriggerCon",
22     wsdlLocation = "META-INF/wsdl/EJBPersonNoSecCli/TriggerCon.wsdl")
23 @Stateless
24 public class EJBPersonNoSecCli implements TriggerConPortType {
25
26     public org.netbeans.xml.schema.person.PersonRes triggerPerson
27         (org.netbeans.xml.schema.person.PersonReq msgPersonDetailsReq)
28         throws TriggerPersonFault {
29         org.netbeans.xml.schema.person.PersonReq cReq
30             = new org.netbeans.xml.schema.person.PersonReq();
31         cReq.setPersonID(msgPersonDetailsReq.getPersonID());
32
33         org.netbeans.xml.schema.person.PersonRes cRes
34             = new org.netbeans.xml.schema.person.PersonRes();
35         cRes.setPersonID(msgPersonDetailsReq.getPersonID());
36
37         // invoke service here
38
39         return cRes;
40     }
```

Figure 5.10.3 Client implementation code before service invocation is added

Variables cReq and cRes will contain request to the service and response from the service respectively.

Expand Web Service References node all the way to the service operation and drag the service operation onto the source window following the comment “// invoke service here”, as shown in Figure 5.10.4.

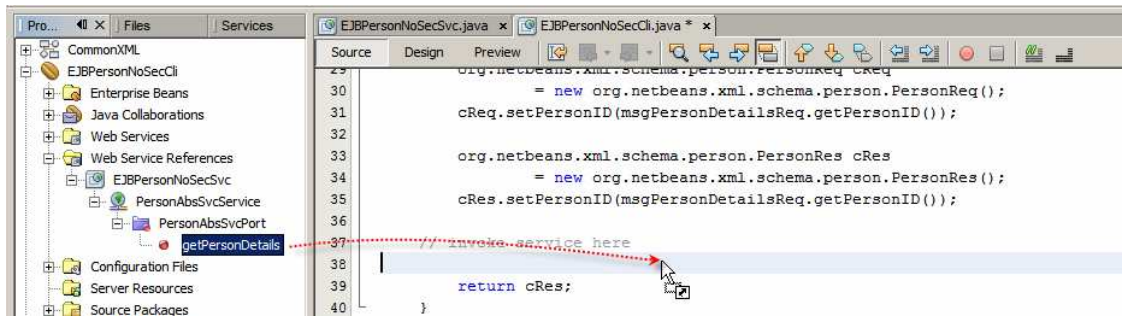


Figure 5.10.4 Drag the service operation onto the source window

The code fragment, which was generated by the NetBeans tooling, reformatted for better readability, is shown in Figure 5.10.5.

```

36
37     org.netbeans.xml.schema.person.PersonRes cRes
38         = new org.netbeans.xml.schema.person.PersonRes ();
39     cRes.setPersonID(msgPersonDetailsReq.getPersonID());
40
41     // invoke service here
42
43     try { // Call Web Service Operation
44         org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType port
45             = service.getPersonAbsSvcPort();
46         // TODO initialize WS operation arguments here
47         org.netbeans.xml.schema.person.PersonReq msgPersonDetailsReq
48             = new org.netbeans.xml.schema.person.PersonReq ();
49         // TODO process result here
50         org.netbeans.xml.schema.person.PersonRes result
51             = port.getPersonDetails(msgPersonDetailsReq);
52         System.out.println("Result = "+result);
53     } catch (Exception ex) {
54         // TODO handle custom exceptions here
55     }
56
57     return cRes;
58 }

```

Figure 5.10.5 Service invocation skeleton code

Replace the lines inside the try { ... } catch block with the code in Listing 5.10.3.

Listing 5.10.3 Code to invoke the service and process the reply

```

org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType port
    = service.getPersonAbsSvcPort();
org.netbeans.xml.schema.person.PersonRes result = port.getPersonDetails(cReq);

System.out.println("\n===>>> " + result.getFamilyName());

cRes.setFamilyName(result.getFamilyName());
cRes.setMiddleInitials(result.getMiddleInitials());
cRes.setGivenName(result.getGivenName());
cRes.setGender(result.getGender());

org.netbeans.xml.schema.person.PersonRes.AddressDetails cAddr
    = new org.netbeans.xml.schema.person.PersonRes.AddressDetails();
cAddr.setStreetAddress(result.getAddressDetails().getStreetAddress());

```



```

cAddr.setCityTown(result.getAddressDetails().getCityTown());
cAddr.setStateProvince(result.getAddressDetails().getStateProvince());
cAddr.setPostCode(result.getAddressDetails().getPostCode());
cAddr.setCountry(result.getAddressDetails().getCountry());
cRes.setAddressDetails(cAddr);

org.netbeans.xml.schema.person.PersonRes.CreditCardDetails cCard =
    new org.netbeans.xml.schema.person.PersonRes.CreditCardDetails();
cCard.setCardNumber(result.getCreditCardDetails().getCardNumber());
cCard.setCardType(result.getCreditCardDetails().getCardType());
cCard.setExpiryDate(result.getCreditCardDetails().getExpiryDate());
cCard.setSecurityCode(result.getCreditCardDetails().getSecurityCode());
cRes.setCreditCardDetails(cCard);

```

I will leave analysis of what this slab of code does to the reader. Suffices it to say that the new request, `cReq`, populated with the `PersonID` from the client, is submitted to the service and the response, `cRes`, is used to populate the response of the client.

The resulting source is illustrated in Figure 5.10.6.

```

41 // invoke service here
42
43 try { // Call Web Service Operation
44     org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType port
45         = service.getPersonAbsSvcPort();
46     org.netbeans.xml.schema.person.PersonRes result = port.getPersonDetails(cReq);
47
48     System.out.println("\n====>>> " + result.getFamilyName());
49
50     cRes.setFamilyName(result.getFamilyName());
51     cRes.setMiddleInitials(result.getMiddleInitials());
52     cRes.setGivenName(result.getGivenName());
53     cRes.setGender(result.getGender());
54
55     org.netbeans.xml.schema.person.PersonRes.AddressDetails cAddr
56         = new org.netbeans.xml.schema.person.PersonRes.AddressDetails();
57     cAddr.setStreetAddress(result.getAddressDetails().getStreetAddress());
58     cAddr.setCityTown(result.getAddressDetails().getCityTown());
59     cAddr.setStateProvince(result.getAddressDetails().getStateProvince());
60     cAddr.setPostCode(result.getAddressDetails().getPostCode());
61     cAddr.setCountry(result.getAddressDetails().getCountry());
62     cRes.setAddressDetails(cAddr);
63
64     org.netbeans.xml.schema.person.PersonRes.CreditCardDetails cCard =
65         new org.netbeans.xml.schema.person.PersonRes.CreditCardDetails();
66     cCard.setCardNumber(result.getCreditCardDetails().getCardNumber());
67     cCard.setCardType(result.getCreditCardDetails().getCardType());
68     cCard.setExpiryDate(result.getCreditCardDetails().getExpiryDate());
69     cCard.setSecurityCode(result.getCreditCardDetails().getSecurityCode());
70     cRes.setCreditCardDetails(cCard);
71 } catch (Exception ex) {
72     // TODO handle custom exceptions here
73 }

```

Figure 5.10.6 *Submit request and process response*

To complete client implementation let's replace the one line comment "`// TODO handle ...`" with the slab of code shown in Listing 5.10.4, then right-click inside the source window and choose "Fix Imports".

Listing 5.10.4 *Convert exception into a Fault*

```

ex.printStackTrace();
String sFltMsg = ex.getMessage();
PersonFlt pFlt = new PersonFlt();
pFlt.setPersonID(msgPersonDetailsReq.getPersonID());
pFlt.setFaultDetail(sFltMsg);
org.netbeans.j2ee.wsdl.commonxml.triggercon.TriggerPersonFault sFlt
    = new org.netbeans.j2ee.wsdl.commonxml.triggercon.TriggerPersonFault
        (sFltMsg, pFlt);
throw sFlt;

```

Figure 5.10.7 shows the completed implementation code.

```

42 // invoke service here
43
44 try { // Call Web Service Operation
45     org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType port
46         = service.getPersonAbsSvcPort();
47     org.netbeans.xml.schema.person.PersonRes result = port.getPersonDetails(cReq);
48
49     System.out.println("\n====>>> " + result.getFamilyName());
50
51     cRes.setFamilyName(result.getFamilyName());
52     cRes.setMiddleInitials(result.getMiddleInitials());
53     cRes.setGivenName(result.getGivenName());
54     cRes.setGender(result.getGender());
55
56     org.netbeans.xml.schema.person.PersonRes.AddressDetails cAddr
57         = new org.netbeans.xml.schema.person.PersonRes.AddressDetails();
58     cAddr.setStreetAddress(result.getAddressDetails().getStreetAddress());
59     cAddr.setCityTown(result.getAddressDetails().getCityTown());
60     cAddr.setStateProvince(result.getAddressDetails().getStateProvince());
61     cAddr.setPostCode(result.getAddressDetails().getPostCode());
62     cAddr.setCountry(result.getAddressDetails().getCountry());
63     cRes.setAddressDetails(cAddr);
64
65     org.netbeans.xml.schema.person.PersonRes.CreditCardDetails cCard =
66         new org.netbeans.xml.schema.person.PersonRes.CreditCardDetails();
67     cCard.setCardNumber(result.getCreditCardDetails().getCardNumber());
68     cCard.setCardType(result.getCreditCardDetails().getCardType());
69     cCard.setExpiryDate(result.getCreditCardDetails().getExpiryDate());
70     cCard.setSecurityCode(result.getCreditCardDetails().getSecurityCode());
71     cRes.setCreditCardDetails(cCard);
72 } catch (Exception ex) {
73     ex.printStackTrace();
74     String sFltMsg = ex.getMessage();
75     PersonFlt pFlt = new PersonFlt();
76     pFlt.setPersonID(msgPersonDetailsReq.getPersonID());
77     pFlt.setFaultDetail(sFltMsg);
78     org.netbeans.j2ee.wsdl.commonxml.triggercon.TriggerPersonFault sFlt
79         = new org.netbeans.j2ee.wsdl.commonxml.triggercon.TriggerPersonFault
80             (sFltMsg, pFlt);
81     throw sFlt;
82 }
83
84 return cRes;

```

Figure 5.10.7 Client implementation

Build and Deploy the project.

The web service client we just built is itself a web service. We designed this client in this way so that we can use the SoapUI plugin to invoke the client, rather than go to the trouble of working out how else to invoke it and to build the appropriate implementation.

To create a SopUI project we need a WSDL URL. Expand the Web Service node in the client project, right-click the web service name and choose Test Web Service, as illustrated in Figure 5.10.8.

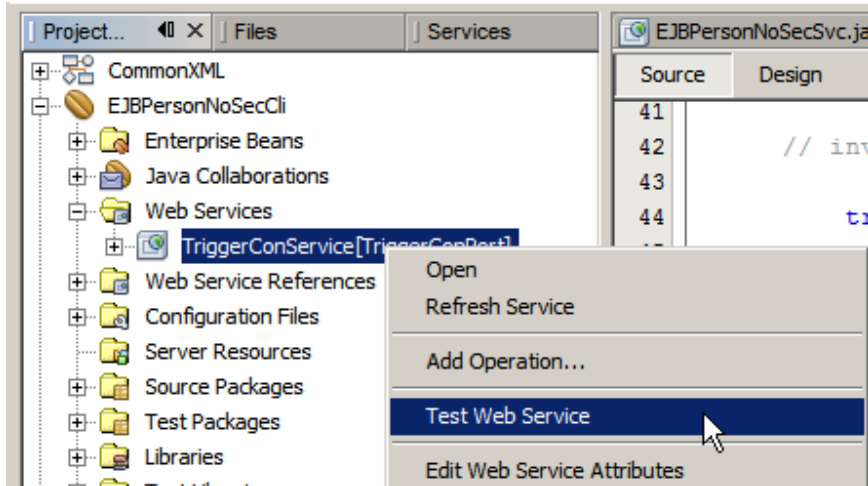


Figure 5.10.8 Choose to test Web Service

Once the web browser window opens with the test page, right-click the WSDL link and choose Copy Link Location, or equivalent. Figure 5.10.9 shows this in the Firefox web browser.

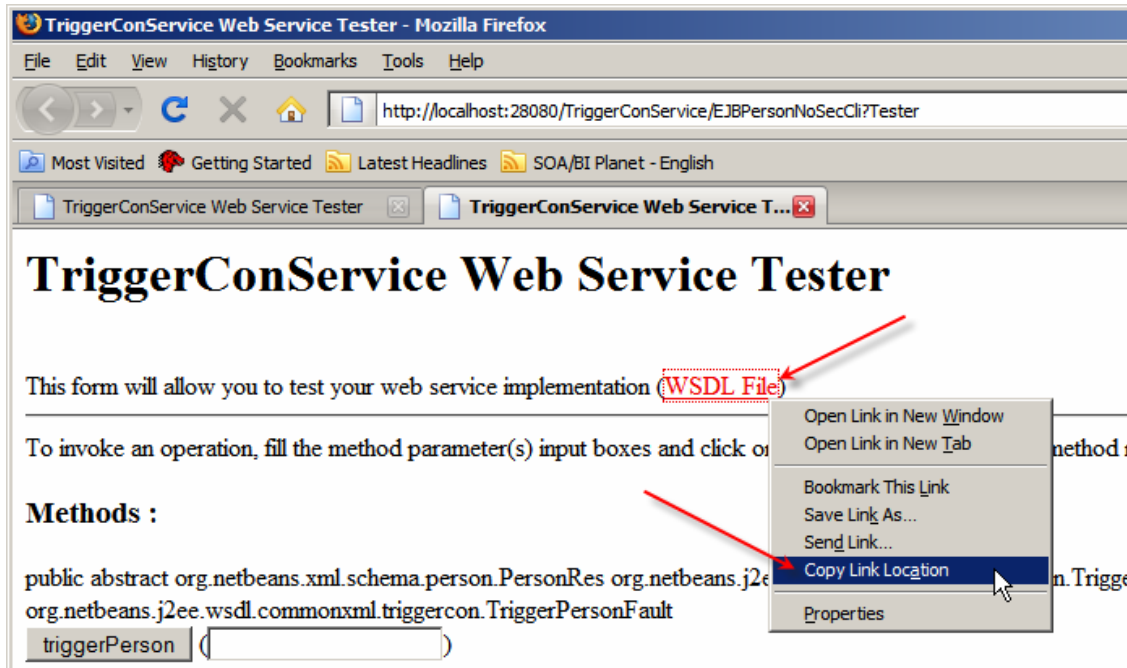


Figure 5.10.9 Copy WSDL link location

For me the URL will be:

http://localhost:28080/TriggerConService/EJBPersonNoSecCli?WSDL

Default port number is 8080. I changed mine to 28080.

We can not actually test this service through the web browser. The service expects a structured message, conforming to Person XML Schema. We would have to construct a XML instance document and paste it into the text box. Even if we did go to that trouble the XML text would have gotten “escaped” by the form processor and the resulting message would have been garbled anyway.

As mentioned, we will use the SoapUI to trigger the client.

Create “New Project” -> “Java EE” -> “Web Service Testing Project”, named EJBPersonNoSecCli_WSTP (this assumes that the SoapUI plugin has been installed – if it has not then now is the time to obtain and install it). Use the WSDL URL just copied to the clipboard as the service URL. Figure 5.10.10 illustrates this step.

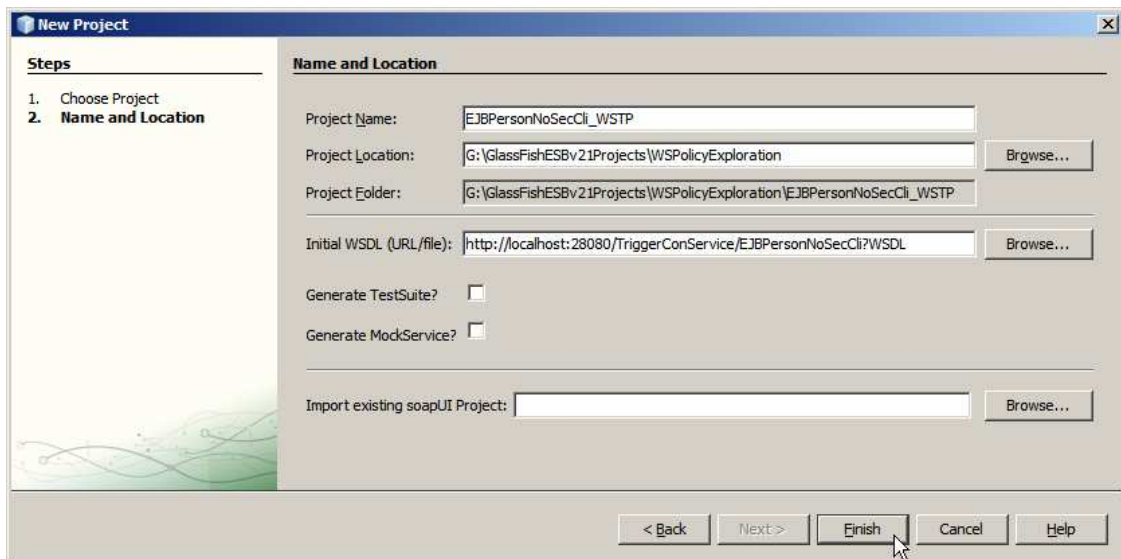


Figure 5.10.10 Create new web service testing project

Once the project is created expand the nodes, right-click the service operation and choose “New request”, as illustrated in Figure 5.10.11.

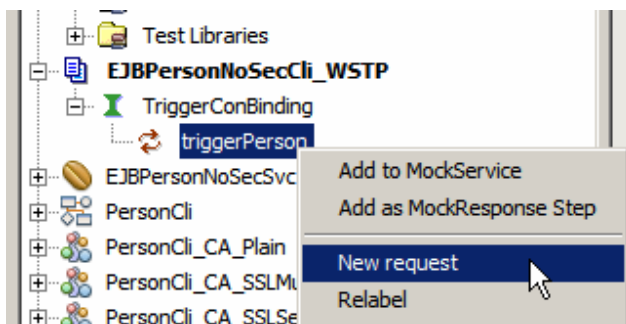


Figure 5.10.11 Add new request

Set PersonID to some favourite value and click the Submit request “button”. This is illustrated in Figure 5.10.12.

```

EJBPersonNoSecSvc.java x EJBPersonNoSecCli.java x Request 1 x
http://localhost:28080/TriggerConService/EJBPersonNoSecCli
Submit request to specified endpoint URL : soapenv="http://schemas.xmlsoap.org/s
2 <soapenv:Header/>
3 <soapenv:Body>
4 <per:PersonReq>
5 <!--type: string-->
6 <per:PersonID>123123</per:PersonID>
7 </per:PersonReq>
8 </soapenv:Body>
9 </soapenv:Envelope>

```

Figure 5.10.12 Submit SOAP Request

If all goes well, the service response will look similar to that shown in Figure 5.10.13.

```

EJBPersonNoSecSvc.java x EJBPersonNoSecCli.java x Request 1 x
http://localhost:28080/TriggerConService/EJBPersonNoSecCli
1 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
2 <S:Body>
3 <PersonRes xmlns="http://xml.netbeans.org/schema/Person">
4 <PersonID>123123</PersonID>
5 <FamilyName>Doe</FamilyName>
6 <GivenName>John</GivenName>
7 <Gender>M</Gender>
8 <AddressDetails>
9 <StreetAddress>33 Berry Street</StreetAddress>
10 <CityTown>North Sydney</CityTown>
11 <PostCode>2160</PostCode>
12 <StateProvince>NSW</StateProvince>
13 <Country>Australia</Country>
14 </AddressDetails>
15 <CreditCardDetails>
16 <CardType>Passport</CardType>
17 <CardNumber>123-456-7689-0123</CardNumber>
18 <ExpiryDate>01/21</ExpiryDate>
19 <SecurityCode>Sat Sep 19 11:16:55 EST 2009</SecurityCode>
20 </CreditCardDetails>
21 </PersonRes>
22 </S:Body>
23 </S:Envelope>

```

Figure 5.10.13 Service Response

To test exception processing in the client, undeploy the service and submit the request again. Figure 5.10.14 shows the Fault response.

```

Raw XML
1 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
2 <S:Body>
3 <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
4 <faultcode>S:Server</faultcode>
5 <faultstring>HTTP Status-Code 404: Not Found - Not Found</faultstring>
6 <detail>
7 <PersonFlt:PersonFlt xmlns:PersonFlt="http://xml.netbeans.org/schema/">
8 <PersonID>123123</PersonID>
9 <FaultDetail>HTTP Status-Code 404: Not Found - Not Found</FaultDet:
10 </PersonFlt:PersonFlt>
11 </detail>
12 </S:Fault>
13 </S:Body>
14 </S:Envelope>

```

Figure 5.10.14 Fault response

The service provider and the service consumer were implemented and exercised. This will help with the next section. Before going on let's undeploy the client as well so that neither the client nor the service are deployed.

5.11 EJB-based Person Svc with Server-side Authentication

In this section I will discuss how to configure SSL with Server-side Authentication for an EJB-based Web Service, using the “Interface First” programming model, though this equally applies to “Implementation First” programming model for EJB Web Services.

WS-Security Policy and Metro / WSIT are not used for this. As far as I can tell the EJB-based web service implementation, and the infrastructure that supports it, completely ignores the WS-Security Policy and Metro / WSIT policies relating to channel security – SSL / TLS, whether with Server-side or Mutual Authentication. This section is not, then, an exploration of security policy but an exploration of a practical implementation of SSL- / TLS-based channel security.

The security parameters, port, certificate alias, cryptographic suites, are configured in the GlassFish Application Server. Figures 5.11.1 and 5.11.2 show the areas of particular interest.

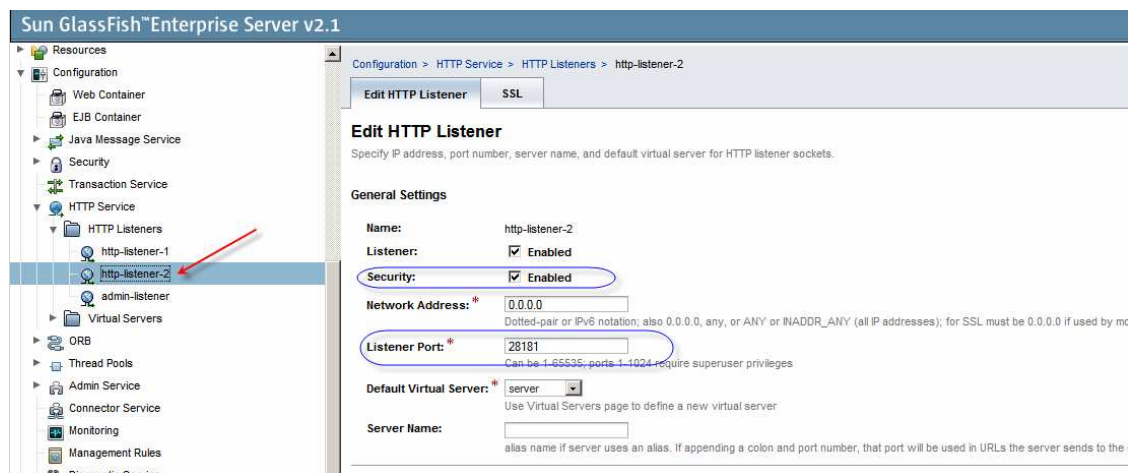


Figure 5.11.1 *http-listener-2* port configuration

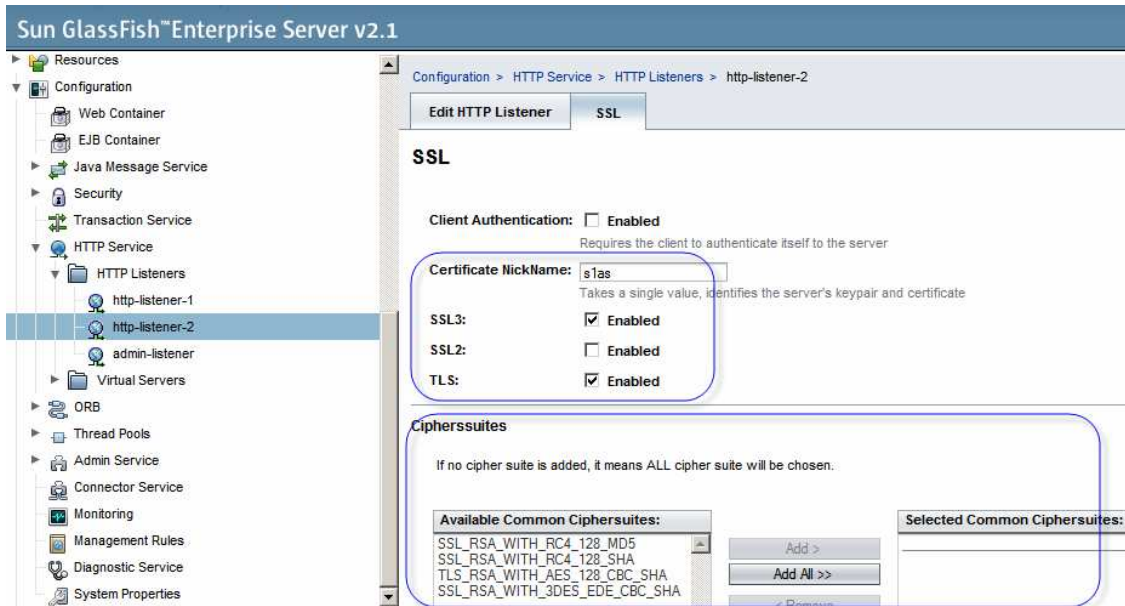


Figure 5.11.2 SSL / TLS Configuration

Unlike in the JBI world, where logic and policy can be separated into a BPLE Module and a Composite Application projects, with EJB-based Web Services a single project will implement both.

Click anywhere in the empty area of the Project Explorer and create “New Project” -> “Java EE” -> “EJB Module”. Name this module EJBPersonSSLServerAuthSvc and have it deploy to the remote GlassFish instance.

Right-click on the name of the new project. Choose “New” -> “Other” -> “Web Services” -> “Web Service From WSDL”. Name the service EJBPersonSSLServerAuthSvc, name the package pkg.EJBPersonSSLServerAuthSvc, browse to the CommonXML/PersonAbsSvc WSDL, select it and click Finish.

As the wizard completes a skeleton Java Source of the implementation class will appear in a window.

Replace the line containing the comment “// TO DO ...” and the following line with the java statements shown in Listing 5.11.1.

Listing 5.10.1 Method body

```

PersonRes res = new PersonRes();
res.setPersonID(msgPersonDetailsReq.getPersonID());
res.setFamilyName("Doe");
res.setGivenName("John");
res.setGender("M");

PersonRes.AddressDetails addr = new PersonRes.AddressDetails();

addr.setStreetAddress("33 Berry Street");
addr.setCityTown("North Sydney");
addr.setStateProvince("NSW");
addr.setPostCode("2160");
addr.setCountry("Australia");

```

```

res.setAddressDetails(addr);

PersonRes.CreditCardDetails card = new PersonRes.CreditCardDetails();
card.setCardNumber("123-456-7689-0123");
card.setCardType("Passport");
card.setExpiryDate("01/21");
card.setSecurityCode("SecurityCode");
res.setCreditCardDetails(card);

return res;

```

The class implementation should look like that shown in Figure 5.11.3.

```

18  @WebService
19      (serviceName = "PersonAbsSvcService",
20      portName = "PersonAbsSvcPort",
21      endpointInterface = "org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType",
22      targetNamespace = "http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc",
23      wsdlLocation = "META-INF/wsdl/EJBPersonSSLServerAuthSvc/PersonAbsSvcWrapper.wsdl")
24  @Stateless
25  public class EJBPersonSSLServerAuthSvc implements PersonAbsSvcPortType {
26
27      public org.netbeans.xml.schema.person.PersonRes getPersonDetails
28          (org.netbeans.xml.schema.person.PersonReq msgPersonDetailsReq)
29          throws GetPersonDetailsFault {
30          PersonRes res = new PersonRes();
31          res.setPersonID(msgPersonDetailsReq.getPersonID());
32          res.setFamilyName("Doe");
33          res.setGivenName("John");
34          res.setGender("M");
35
36          PersonRes.AddressDetails addr = new PersonRes.AddressDetails();
37
38          addr.setStreetAddress("33 Berry Street");
39          addr.setCityTown("North Sydney");
40          addr.setStateProvince("NSW");
41          addr.setPostCode("2160");
42          addr.setCountry("Australia");
43          res.setAddressDetails(addr);
44
45          PersonRes.CreditCardDetails card = new PersonRes.CreditCardDetails();
46          card.setCardNumber("123-456-7689-0123");
47          card.setCardType("Passport");
48          card.setExpiryDate("01/21");
49          card.setSecurityCode("SecurityCode");
50          res.setCreditCardDetails(card);
51
52          return res;
53      }

```

Figure 5.11.3 Implementation code

Move the “@Stateless” annotation from its location on line 23 to just before the “@WebService annotation”. Figures 5.11.4 and 5.11.5 illustrate the “before” and “after” state.


```

17 L
18 @WebService
19     (serviceName = "PersonAbsSvcService",
20      portName = "PersonAbsSvcPort",
21      endpointInterface = "org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType",
22      targetNamespace = "http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc",
23      wsdlLocation = "META-INF/wsdl/EJBPersonSSLServerAuthSvc/PersonAbsSvcWrapper.wsdl")
24 @Stateless
25 public class EJBPersonSSLServerAuthSvc implements PersonAbsSvcPortType {

```

Figure 5.11.4 “Before” state

```

17 L */
18 @Stateless
19 @WebService
20     (serviceName = "PersonAbsSvcService",
21      portName = "PersonAbsSvcPort",
22      endpointInterface = "org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType",
23      targetNamespace = "http://j2ee.netbeans.org/wsdl/CommonXML/PersonAbsSvc",
24      wsdlLocation = "META-INF/wsdl/EJBPersonSSLServerAuthSvc/PersonAbsSvcWrapper.wsdl")
25 public class EJBPersonSSLServerAuthSvc implements PersonAbsSvcPortType {
26

```

Figure 5.11.5 “After” state

The empirical reason for this is that processing of the `@WebService` annotation before the `@Stateless` annotation generates code that ignores channel security.

Build the project.

Click on the “Configuration Files” folder to select it. Click on the drop down “File” NetBeans menu. Choose “New File” -> “Other” -> “Empty File”. Figure 5.11.6 illustrates a step in this process.

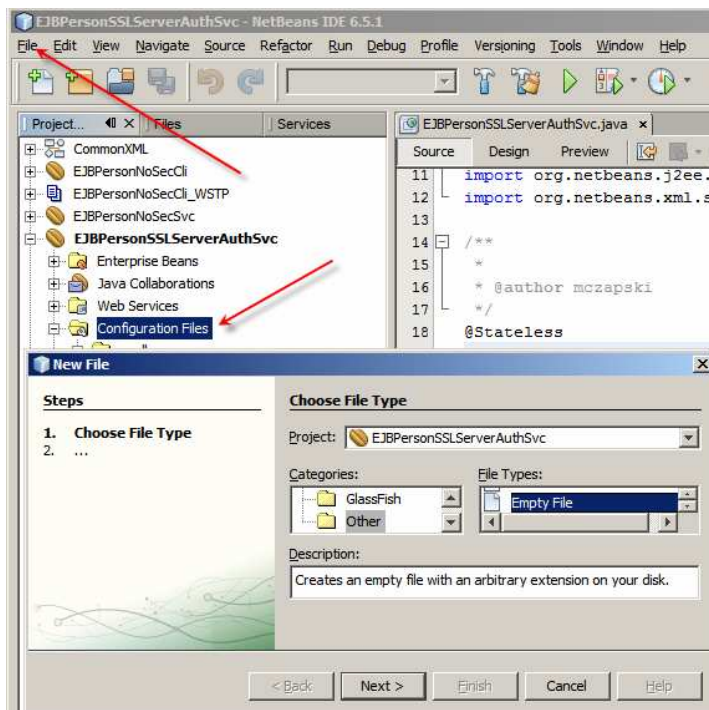


Figure 5.11.6 Create Empty File in the Configuration Files folder

Name the new file “sun-ejb-jar.xml”.

Paste the XML content shown in Listing 5.10.2 into the sun-ejb-jar.xml.

Listing 5.11.2 *Content of the sun-ejb-sar.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD
Application Server 9.0 EJB 3.0//EN"
'http://www.sun.com/software/appserver/dtds/sun-ejb-jar_3_0-0.dtd'>
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>EJBPersonSSLServerAuthSvc</ejb-name>
      <webservice-endpoint>
        <port-component-name>EJBPersonSSLServerAuthSvc</port-component-name>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </webservice-endpoint>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

This seems to be the standard way of requiring SSL-based channel security for EJB-based web endpoints deployed to the GlassFish Application Server. See <http://java.sun.com/developer/EJTEchTips/2006/tt0527.html> for discussion. The ejb-name and port-component-name values are the name of the implementation class, EJBPersonSSLServerAuthSvc.

Save the project, Build and Deploy.

Expand the “Web Service” node, right click on the port name and choose “Test Web Service”.

When the Web Browser opens, with the web service tester URL, there will be an error message in the content windows. Figure 5.17.7 shows the error message in the Firefox browser.



Figure 5.11.7 *Error invoking web service test functionality*

Clearly, there is no way to access the service over non-secure channel.

For me, because I don’t use default port numbers, the secure channel URL will be:

`https://orad1.ssc:28181/PersonAbsSvcService/EJBPersonSSLServerAuthSvc`

The default port number of the SSL-enabled listener will be 8181.

Let’s attempt to access the WSDL for this service using the following URL:

`https://orad1.ssc:28181/PersonAbsSvcService/EJBPersonSSLServerAuthSvc?WSDL`

Access succeeds. Figure 5.11.8 shows the WSDL in Firefox.

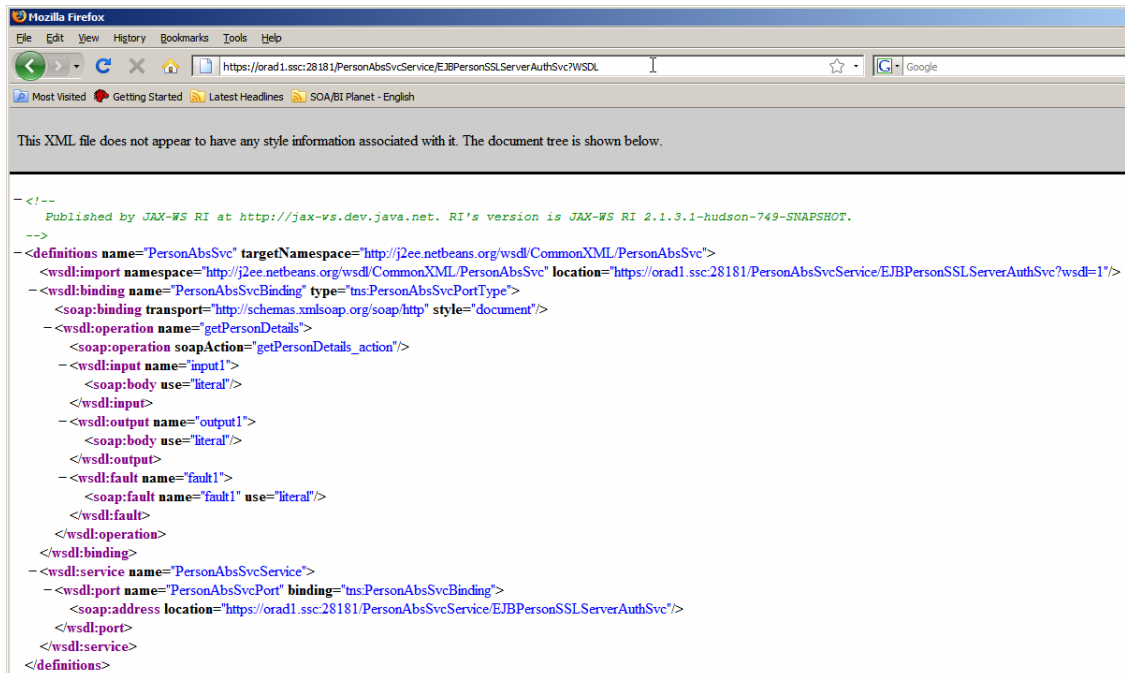


Figure 5.11.8 WSDL obtained over the secure channel

Let's now try to use the non-secure channel to get the WSDL. For me, the URL will be:

`http://orad1.ssc:28080/PersonAbsSvcService/EJBPersonSSLServerAuthSvc?WSDL`

Access will fail. The browser will show a blank page and the server.log will have a WARNING-level message:

```
[# | 2009-09-19T11:46:16.556+1000 | WARNING | sun-appserver2.1 | javax.enterprise.system.container.ejb | _ThreadID=30;_ThreadName=httpSSLWorkerThread-28080-2;_RequestID=b4cbae17-fdd9-4051-991b-71482e166246; | Invalid request scheme for Endpoint EJBPersonSSLServerAuthSvc. Expected https . Received http|#]
```

Let's now create a "New Project" -> "Java EE" -> "Web Service Testing Project", named EJBPersonSSLServerAuthSvc_WSTP. Use the secure WSDL URL, which for me will be:

`https://orad1.ssc:28181/PersonAbsSvcService/EJBPersonSSLServerAuthSvc?WSDL`

Create, populate and submit a SOAP request. Figure 5.11.9 illustrates the project and the request.

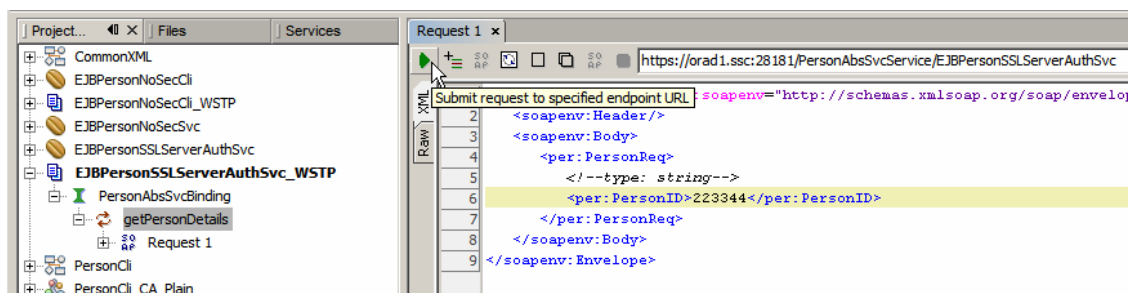


Figure 5.11.9 Submit SOAP request

Figure 5.11.10 shows the SOAP Response, the X.509 Certificate and the Crypto Suite, which was provided and negotiated by the GlassFish Application Server to the SoapUI client during the SSL Handshake.

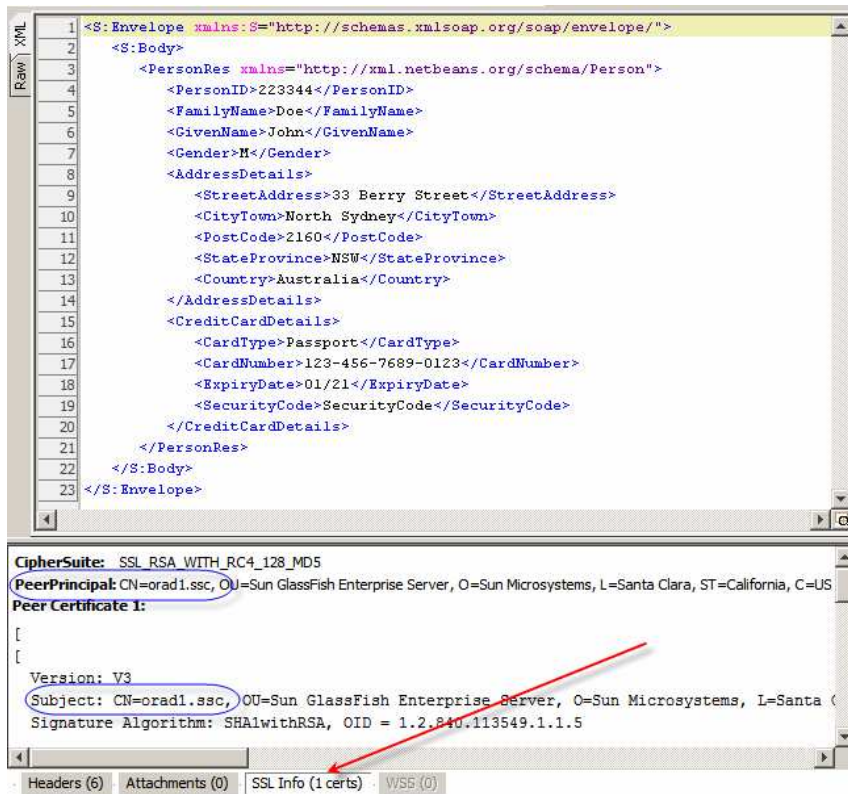


Figure 5.11.10 SOAP Response and SSL information

SSL Handshake can be logged to the server.log by adding “-Djavax.net.debug=ssl:handshake” to the GlassFish Application Server’s JVM Options. Figure 5.8.1 illustrates this in the GlassFish Application Server Admin Console.

The web service, secured using SSL with Server-side Authentication, is operational. Let’s see if we can access it using a non-secure URL, which for me would be:

<http://orad1.ssc:28080/PersonAbsSvcService/EJBPersonSSLServerAuthSvc>

Create a new SOAP Request in the EJBPersonSSLServerAuthSvc_WSTP project and change the endpoint URL to the non-secure one, as shown in Figure 5.11.11.



Figure 5.11.11 Create a new request and “[edit current ...]” endpoint URL

Submit the request.

The SoapUI will give a blank response, which is perhaps misleading, and the server.log will show the, familiar by now, WARNING-level message:

```
[# | 2009-09-19T11:55:18.389+1000 | WARNING | sun-  
appserver2.1 | javax.enterprise.system.container.ejb | _ThreadID=31;_ThreadName=http  
SSLWorkerThread-28080-3;_RequestID=92f0cc01-3b4e-4e89-8776-1ee02f2b5fac; | Invalid  
request scheme for Endpoint EJBPersonSSLServerAuthSvc. Expected https . Received  
http|#]
```

The web service endpoint is secured, using SSL with Server-side Authentication.

Let's create an EJB-based web service client to exercise the service we just implemented.

Create a "New Project" -> "Java EE" -> "EJB Module", named EJBPersonSSLServerAuthCli, with the local GlassFish instance as the deployment target.

Create "New" -> "Web Service From WSDL", named EJBPersonSSLServerAuthCli, in package pkg.EJBPersonSSLServerAuthCli, using WSDL from the CommonXML/TriggerCon.wsdl.

Create "New" -> "External WSDL Document(s)", using the service WSDL URL. For me this will be:

```
https://orad1.ssc:28181/PersonAbsSvcService/EJBPersonSSLServerAuthSvc?WSDL
```

Figure 5.11.12 shows the new WSDL in the project structure.

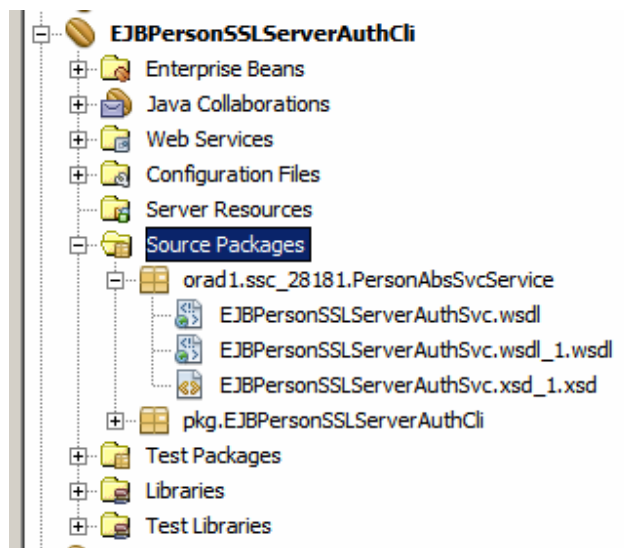


Figure 5.11.12 WSDL in the project structure

Create "New" -> "Web Service Client", click "Local File", Browse to the location of the WSDL EJBPersonSSLServerAuthSvc.wsdl and select it. Figure 5.11.13 illustrates this step.

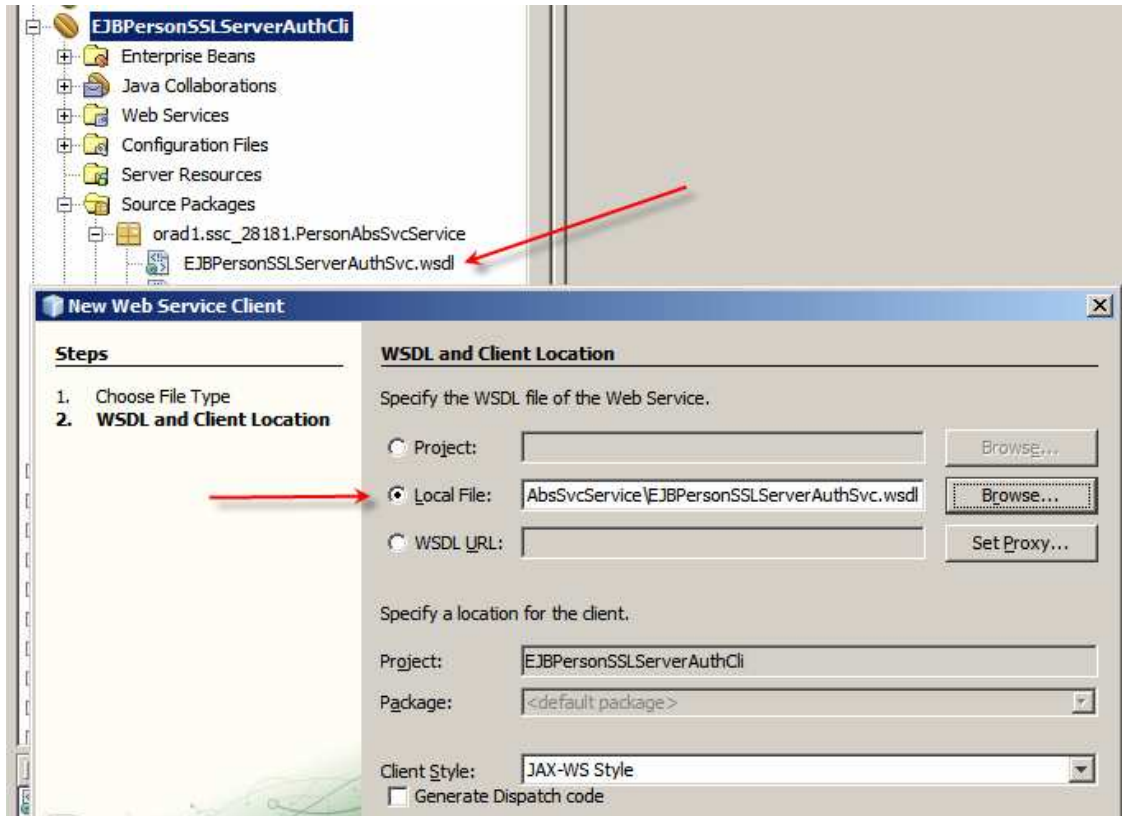


Figure 5.11.13 Use WSDL in the local file

Open the Java source of the client implementation, if not already open.

Replace the two lines of code, “//TODO ...” and “throw new Unsupported...” with the code shown in Listing 5.11.3.

Listing 5.11.3 Replacement code

```

org.netbeans.xml.schema.person.PersonReq cReq
    = new org.netbeans.xml.schema.person.PersonReq();
cReq.setPersonID(msgPersonDetailsReq.getPersonID());

org.netbeans.xml.schema.person.PersonRes cRes
    = new org.netbeans.xml.schema.person.PersonRes();
cRes.setPersonID(msgPersonDetailsReq.getPersonID());

// invoke service here

return cRes;

```

The source, reformatted for better readability, is shown in Figure 5.11.14.

```

17  @WebService
18  (serviceName = "TriggerConService",
19  portName = "TriggerConPort",
20  endpointInterface = "org.netbeans.j2ee.wsdl.commonxml.triggercon.TriggerConPortType",
21  targetNamespace = "http://j2ee.netbeans.org/wsdl/CommonXML/TriggerCon",
22  wsdlLocation = "META-INF/wsdl/EJBPersonSSLServerAuthCli/TriggerCon.wsdl")
23  @Stateless
24  public class EJBPersonSSLServerAuthCli implements TriggerConPortType {
25
26      public org.netbeans.xml.schema.person.PersonRes triggerPerson
27          (org.netbeans.xml.schema.person.PersonReq msgPersonDetailsReq)
28          throws TriggerPersonFault {
29      |
30      |   org.netbeans.xml.schema.person.PersonReq cReq
31      |       = new org.netbeans.xml.schema.person.PersonReq ();
32      |       cReq.setPersonID(msgPersonDetailsReq.getPersonID());
33
34      |   org.netbeans.xml.schema.person.PersonRes cRes
35      |       = new org.netbeans.xml.schema.person.PersonRes ();
36      |       cRes.setPersonID(msgPersonDetailsReq.getPersonID());
37
38      |       // invoke service here
39      |
40      |       return cRes;
41      |   }

```

Figure 5.11.14 Client implementation code before service invocation is added

Variables cReq and cRes will contain request to the service and response from the service respectively.

Expand Web Service References node all the way to the service operation and drag the service operation onto the source window following the comment “// invoke service here”, as shown in Figure 5.11.15.

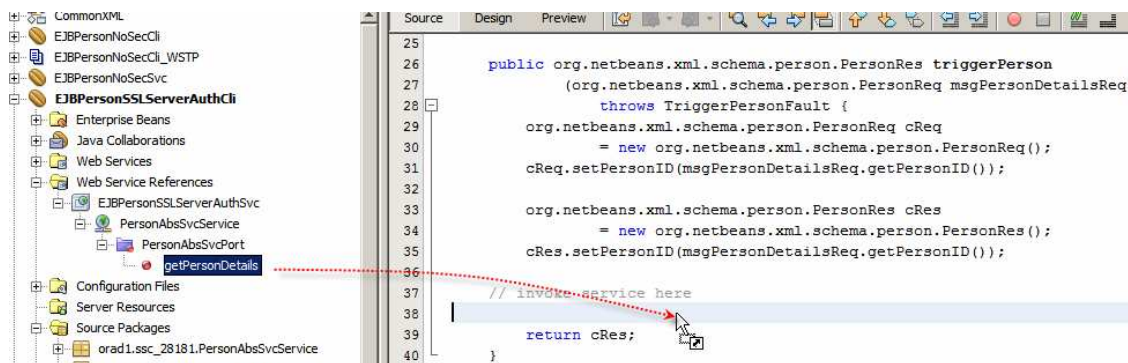


Figure 5.11.15 Drag the service operation onto the source window

The code fragment, which was generated by the NetBeans tooling, reformatted for better readability, is shown in Figure 5.11.16.


```

40     cRes.setPersonID(msgPersonDetailsReq.getPersonID());
41
42     // invoke service here
43
44     try { // Call Web Service Operation
45         org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType port
46             = service.getPersonAbsSvcPort();
47         // TODO initialize WS operation arguments here
48         org.netbeans.xml.schema.person.PersonReq msgPersonDetailsReq
49             = new org.netbeans.xml.schema.person.PersonReq();
50         // TODO process result here
51         org.netbeans.xml.schema.person.PersonRes result
52             = port.getPersonDetails(msgPersonDetailsReq);
53         System.out.println("Result = "+result);
54     } catch (Exception ex) {
55         // TODO handle custom exceptions here
56     }
57
58     return cRes;

```

Figure 5.11.16 Service invocation skeleton code

Replace the lines inside the try { ... } catch block with the code in Listing 5.11.4.

Listing 5.11.4 Code to invoke the service and process the reply

```

org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType port
    = service.getPersonAbsSvcPort();
org.netbeans.xml.schema.person.PersonRes result = port.getPersonDetails(cReq);

System.out.println("\n===>>> " + result.getFamilyName());

cRes.setFamilyName(result.getFamilyName());
cRes.setMiddleInitials(result.getMiddleInitials());
cRes.setGivenName(result.getGivenName());
cRes.setGender(result.getGender());

org.netbeans.xml.schema.person.PersonRes.AddressDetails cAddr
    = new org.netbeans.xml.schema.person.PersonRes.AddressDetails();
cAddr.setStreetAddress(result.getAddressDetails().getStreetAddress());
cAddr.setCityTown(result.getAddressDetails().getCityTown());
cAddr.setStateProvince(result.getAddressDetails().getStateProvince());
cAddr.setPostCode(result.getAddressDetails().getPostCode());
cAddr.setCountry(result.getAddressDetails().getCountry());
cRes.setAddressDetails(cAddr);

org.netbeans.xml.schema.person.PersonRes.CreditCardDetails cCard =
    new org.netbeans.xml.schema.person.PersonRes.CreditCardDetails();
cCard.setCardNumber(result.getCreditCardDetails().getCardNumber());
cCard.setCardType(result.getCreditCardDetails().getCardType());
cCard.setExpiryDate(result.getCreditCardDetails().getExpiryDate());
cCard.setSecurityCode(result.getCreditCardDetails().getSecurityCode());
cRes.setCreditCardDetails(cCard);

```

The new request, cReq, populated with the PersonID from the client, is submitted to the service and the response, cRes, is used to populate the response of the client.

To complete client implementation let's replace the one line comment “// TODO handle ...” with the slab of code shown in Listing 5.11.5, then right-click inside the source window and choose “Fix Imports”.

Listing 5.11.5 Convert exception into a Fault

```
ex.printStackTrace();
String sFltMsg = ex.getMessage();
PersonFlt pFlt = new PersonFlt();
pFlt.setPersonID(msgPersonDetailsReq.getPersonID());
pFlt.setFaultDetail(sFltMsg);
org.netbeans.j2ee.wsdl.commonxml.triggercon.TriggerPersonFault sFlt =
null;
sFlt = new TriggerPersonFault(sFltMsg, pFlt);
throw sFlt;
```

Figure 5.11.17 shows the completed implementation code.

```
43 // invoke service here
44
45 try { // Call Web Service Operation
46     org.netbeans.j2ee.wsdl.commonxml.personabssvc.PersonAbsSvcPortType port
47         = service.getPersonAbsSvcPort();
48     org.netbeans.xml.schema.person.PersonRes result = port.getPersonDetails(cReq);
49
50     System.out.println("\n====>>> " + result.getFamilyName());
51
52     cRes.setFamilyName(result.getFamilyName());
53     cRes.setMiddleInitials(result.getMiddleInitials());
54     cRes.setGivenName(result.getGivenName());
55     cRes.setGender(result.getGender());
56
57     org.netbeans.xml.schema.person.PersonRes.AddressDetails cAddr
58         = new org.netbeans.xml.schema.person.PersonRes.AddressDetails();
59     cAddr.setStreetAddress(result.getAddressDetails().getStreetAddress());
60     cAddr.setCityTown(result.getAddressDetails().getCityTown());
61     cAddr.setStateProvince(result.getAddressDetails().getStateProvince());
62     cAddr.setPostCode(result.getAddressDetails().getPostCode());
63     cAddr.setCountry(result.getAddressDetails().getCountry());
64     cRes.setAddressDetails(cAddr);
65
66     org.netbeans.xml.schema.person.PersonRes.CreditCardDetails cCard =
67         new org.netbeans.xml.schema.person.PersonRes.CreditCardDetails();
68     cCard.setCardNumber(result.getCreditCardDetails().getCardNumber());
69     cCard.setCardType(result.getCreditCardDetails().getCardType());
70     cCard.setExpiryDate(result.getCreditCardDetails().getExpiryDate());
71     cCard.setSecurityCode(result.getCreditCardDetails().getSecurityCode());
72     cRes.setCreditCardDetails(cCard);
73 } catch (Exception ex) {
74     ex.printStackTrace();
75     String sFltMsg = ex.getMessage();
76     PersonFlt pFlt = new PersonFlt();
77     pFlt.setPersonID(msgPersonDetailsReq.getPersonID());
78     pFlt.setFaultDetail(sFltMsg);
79     org.netbeans.j2ee.wsdl.commonxml.triggercon.TriggerPersonFault sFlt = null;
80     sFlt = new TriggerPersonFault(sFltMsg, pFlt);
81     throw sFlt;
82 }
```

Figure 5.11.17 Client implementation

Build and Deploy the project.

The web service client we just built is itself a web service. We designed this client in this way so that we can use the SoapUI plugin to invoke the client, rather than go to the trouble of working out how else to invoke it and to build the appropriate implementation.

To create a SoapUI project we need a WSDL URL. Expand the Web Service node in the client project, right-click the web service name and choose Test Web Service.

Once the web browser window opens with the test page, right-click the WSDL link and choose Copy Link Location, or equivalent, to copy WSDL URL to the clipboard.

For me the URL will be:

`http://localhost:28080/TriggerConService/EJBPersonSSLServerAuthCli?WSDL`

Default port number is 8080. I changed mine to 28080.

We can not actually test this service through the web browser. The service expects a structured message, conforming to Person XML Schema. We would have to construct a XML instance document and paste it into the text box. Even if we did go to that trouble the XML text would have gotten “escaped” by the form processor and the resulting message would have been garbled anyway.

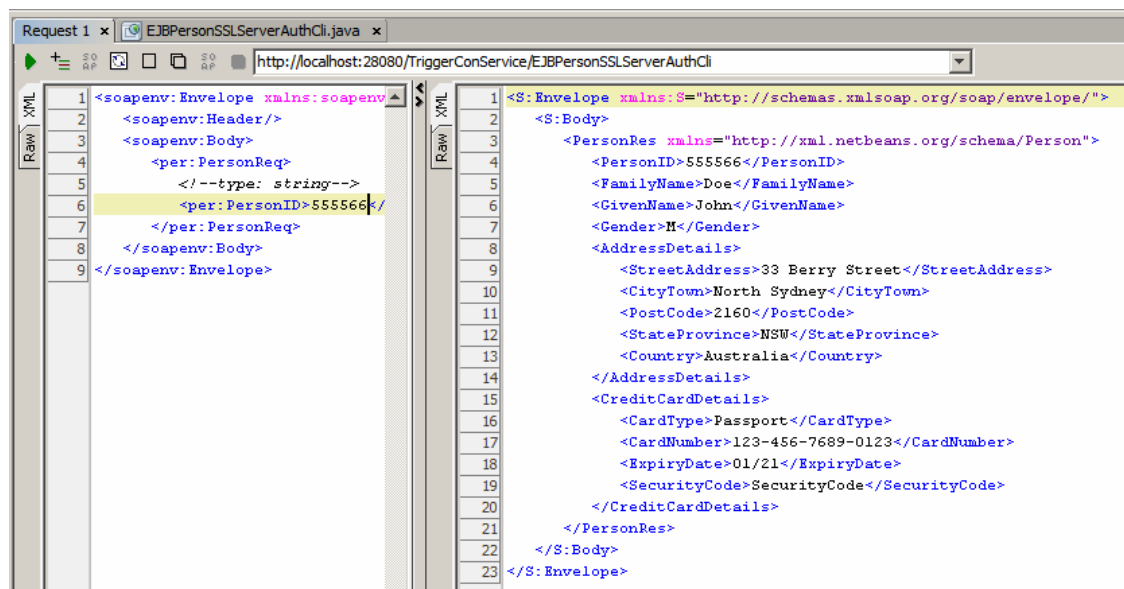
As mentioned, we will use the SoapUI to trigger the client.

Create “New Project” -> “Java EE” -> “Web Service Testing Project”, named EJBPersonSSLServerAuthCli_WSTP (this assumes that the SoapUI plugin has been installed – if it has not then now is the time to obtain and install it). Use the WSDL URL just copied to the clipboard as the service URL.

Once the project is created expand the nodes, right-click the service operation and choose “New request”.

Set PersonID to some favorite value and click the Submit request “button”.

If all goes well, the service response will look similar to that shown in Figure 5.11.18.



The screenshot shows the SoapUI interface with two panels displaying XML. The left panel shows the request XML, and the right panel shows the response XML.

```
Request 1 x EJBPersonSSLServerAuthCli.java x
http://localhost:28080/TriggerConService/EJBPersonSSLServerAuthCli

Raw XML
1 <soapenv:Envelope xmlns:soapenv=
2 <soapenv:Header/>
3 <soapenv:Body>
4 <per:PersonReq>
5 <!--type: string-->
6 <per:PersonID>555566</
7 </per:PersonReq>
8 </soapenv:Body>
9 </soapenv:Envelope>

Raw XML
1 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
2 <S:Body>
3 <PersonRes xmlns="http://xml.netbeans.org/schema/Person">
4 <PersonID>555566</PersonID>
5 <FamilyName>Doe</FamilyName>
6 <GivenName>John</GivenName>
7 <Gender>M</Gender>
8 <AddressDetails>
9 <StreetAddress>33 Berry Street</StreetAddress>
10 <CityTown>North Sydney</CityTown>
11 <PostCode>2160</PostCode>
12 <StateProvince>NSW</StateProvince>
13 <Country>Australia</Country>
14 </AddressDetails>
15 <CreditCardDetails>
16 <CardType>Passport</CardType>
17 <CardNumber>123-456-7689-0123</CardNumber>
18 <ExpiryDate>01/21</ExpiryDate>
19 <SecurityCode>SecurityCode</SecurityCode>
20 </CreditCardDetails>
21 </PersonRes>
22 </S:Body>
23 </S:Envelope>
```

Figure 5.11.18 Service Response

The service provider and the service consumer, communicating over secure channel using SSL with Server-side Authentication, were implemented and exercised. We can no undeploy both the client and the server.

Inspection of server.log of the local GlassFish instance will show the client-side of the SSL Handshake. Inspection of the server.log of the remote GlassFish instance will show the server-side of the SSL Handshake. The traces are worth studying at least once to see what a normal handshake looks like. Whenever there are issues with handshake, as might be the case when server certificate is not trusted or the partner can not agree on the appropriate cipher suite to use, the SSL handshake trace will be the first place to look to figure out who is objecting and to what they are objecting.

5.12 EJB-based Person Svc with Mutual Authentication

5.x Chapter Summary

This chapter explored selected methods of applying security to the channel over which SOAP messages are exchanged and the SOAP messages themselves, using a basic BPEL 2.0-based invoker and provider set.

A pair of projects, an invoker and a provider, were used to provide application logic.

Composite Applications were used to apply different variants of security policies.

The following security policies were explored:

- None
- HTTP BC Channel Security - SSL / TLS with Server-side Authentication
- HTTP BC Channel Security - SSL / TLS with Mutual Authentication
- EJB Channel Security - SSL / TLS with Server-side Authentication
-
- Message Encryption
-

For each variant an end-to-end solution was built and exercised. Server.log traces from both sides were inspected and discussed as necessary to clarify what was happening during the process.