OpenESB, GlassFish ESB
Configuring SOAP/HTTP BC
For plain WS-Security 1.0 Username Token

Michael.Czapski@sun.com, August 2009

Table of Contents

1 Introduction

Every now and then there arises a need to provide authentication credentials for a web service invocation. The WS-Security standards prescribe how a SOAP message must be decorated to convey authentication information. The mater being non-trivial, vendors provide libraries and infrastructure solution that assist developers in declaring, developing and deploying solutions which use WS-Security.

This document discusses how the SOAP/HTTP Binding Component can be configured, in a service provider and in a service consumer, to use WS-Security 1.0 (2004) Username Token Profile support. WS-Security 1.0 (2004) provided support for the Username Token, which could be sent over the wire in the clear. This was insecure but Sun JAX-RPC libraries allowed this, since the standard allowed this. Through Project Metro release 1.4 it was impossibly to formulate a WS-Security policy that decorated a SOAP message with the Username Token headers, without requiring to also encrypt parts of the message. This prevented solutions built on top Metro 1.4, or earlier, from supporting cleartext Username Token. Metro 1.5 relaxed this requirement. The WS-Security policy configured using the GlassFish ESB NetBeans WS-Security wizard will be modified to require and provide a Plain text Username Token.

It is assumed that the reader is sufficiently familiar with the GlassFish ESB / OpenESB BPEL Service Engine and the SOAP/HTTP Binding Component to be able to build projects without a step-by-step pictorial document.

2 Preliminaries

This document assumes the use of the GlassFish ESB v2.1 as the base installation. See https://open-esb.dev.java.net/Downloads.html for download and installation instructions.

GlassFish ESB v2.1 is distributed, to my knowledge, with Metro 1.4 libraries. I don't know for sure but I do know that what needs to be accomplished in this walkthrough can not be accomplished unless Metro gets upgraded to 1.5.

Download Metro 1.5 from https://metro.dev.java.net/1.5/ and install it following the instructions provided, using the GlassFish installation directory of the GlassFish ESB as the target. This will modify your GlassFish ESB / OpenESB. I have not done

extensive testing on the GlassFish ESB environment after installation of Metro 1.5. Bear in mind that your GlassFish ESB support may be compromised, so if you have issues Sun support may be unhappy with you.

3 Build Provider Composite Application

Here is a simple XML Schema Document which will provide definitions for the input and output messages for the service we will be creating.

The XSD is shown in Listing 3-1.

Listing 3-1 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="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:schema>
```

Create New Project -> SOA -> BPEL Module, named PersonSvc.

Create New -> XML Schema. Open the new schema in the editor, select all content, Figure 3-1, and paste the XML Schema shown in Listing 3-1 in its place. Save, Validate, Figure 3-2.

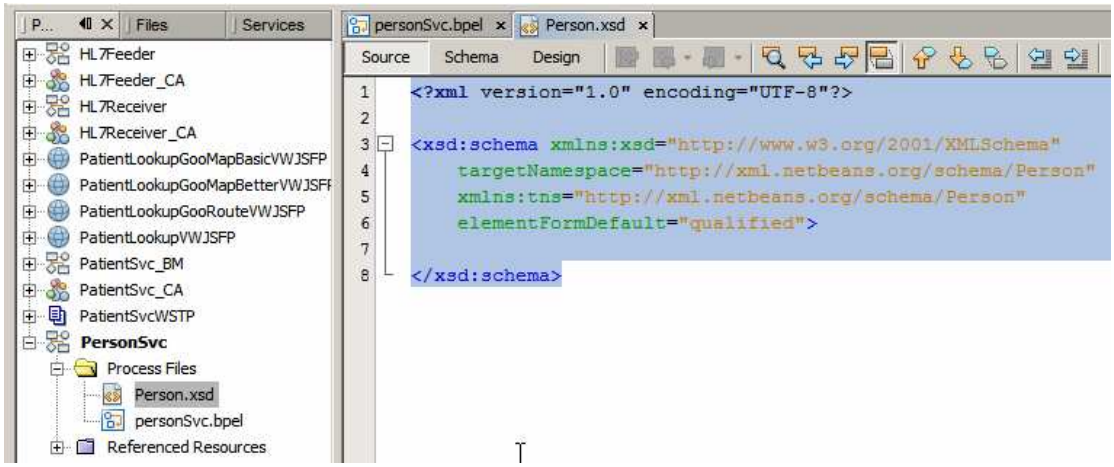

Figure 3-1 Select the whole content of the new schema

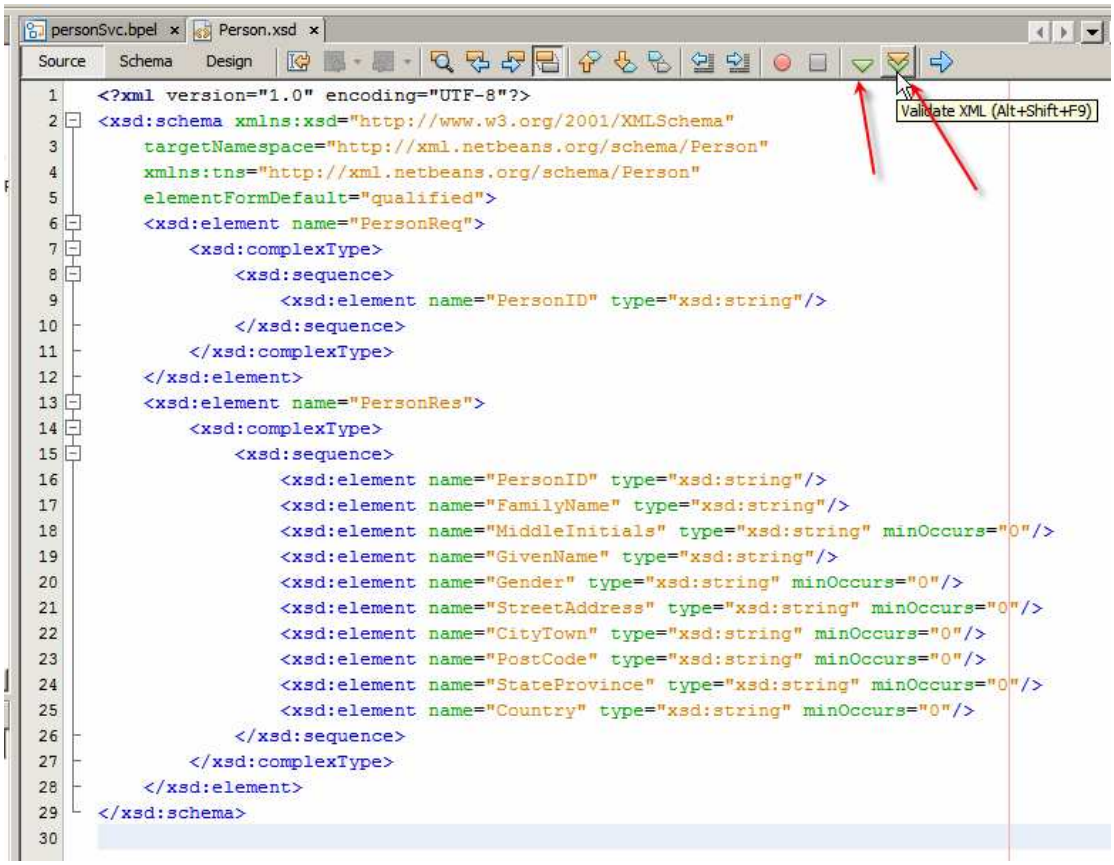

Figure 3-2 Paste, Check, Validate

Create new Concrete WSDL Document, using SOAP binding, of type Document Literal, named PersonSvc, Figure 3-3.

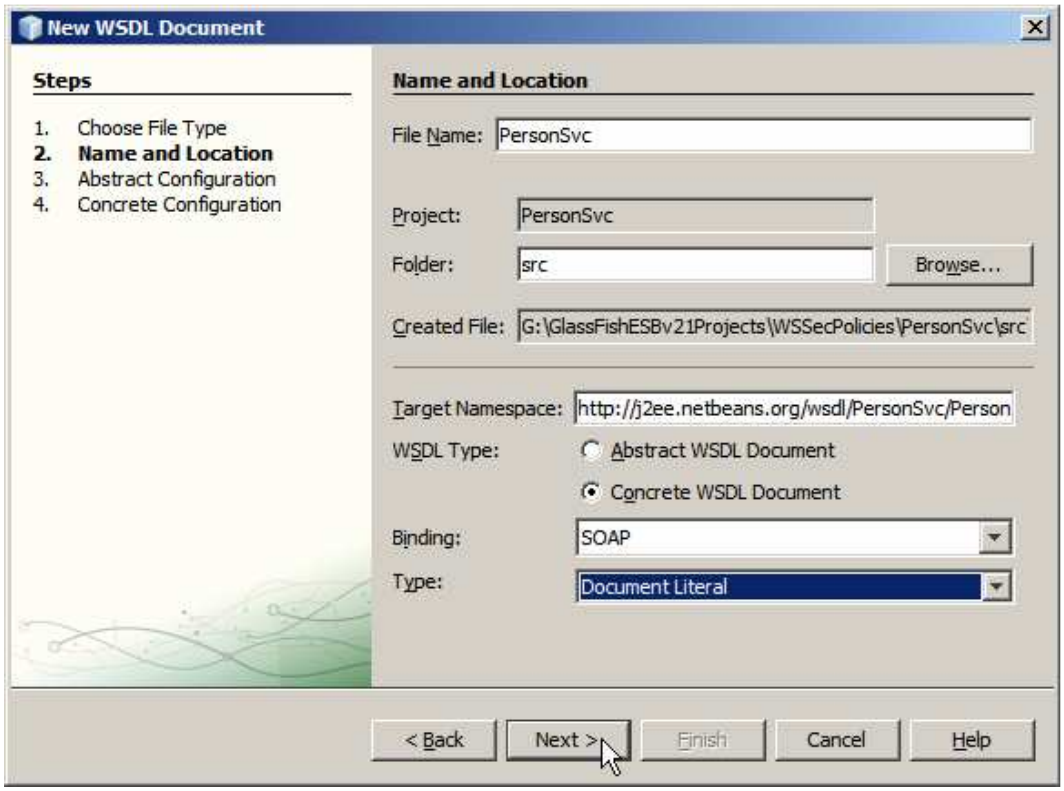

Figure 3-3 Concrete WSDL, SOAP binding, Type Document Literal

Use PersonReq as the request part type and PersonRes as the response part type, Figure 3-4.

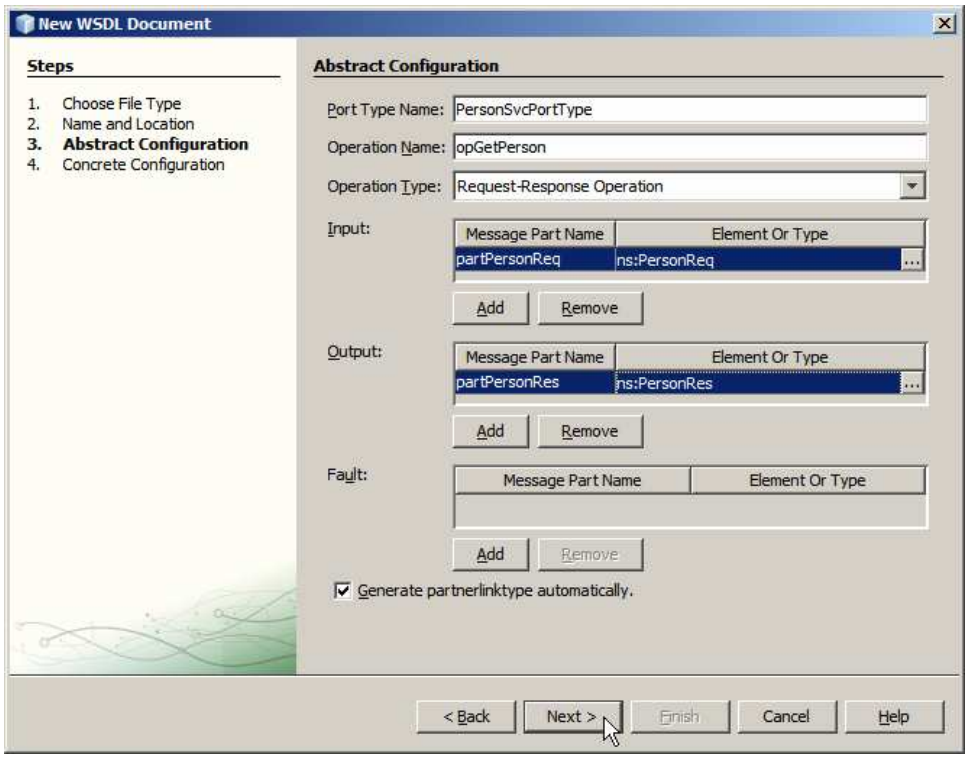

Figure 3-4 Use PersonReq and PersonRes for input and output

Drag the new WSDL onto the BPEL canvas at the providing side, add receive, assign and reply activities, create input and output variables for the receive and the reply activities. Figure 3-5 shows a step in this proves.

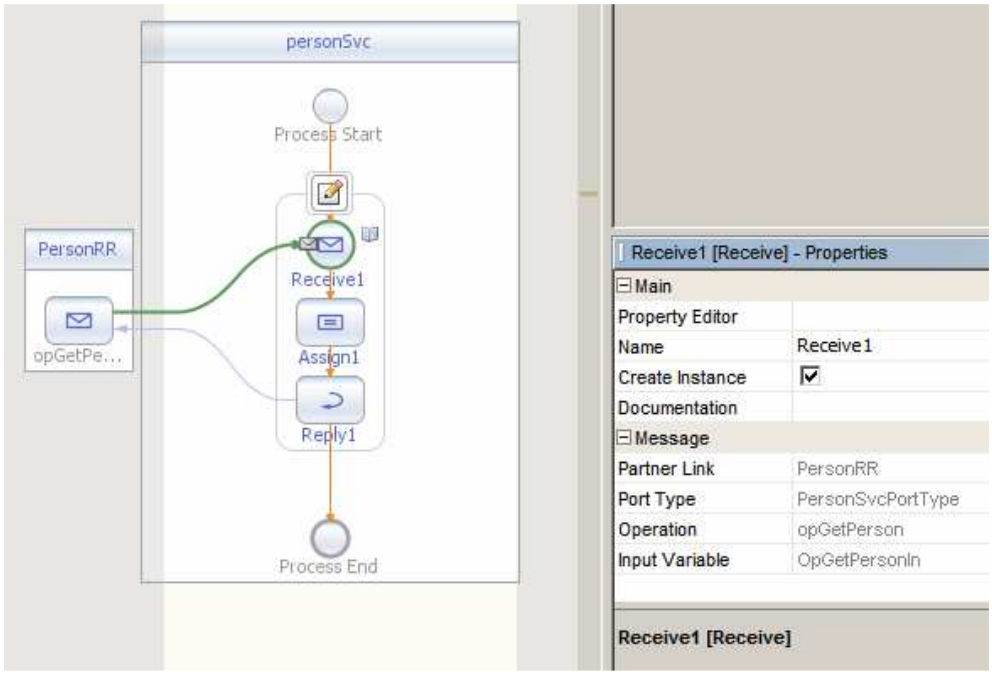

Figure 3-5 Process model and Receive activity properties

Develop a basic mapping which copies the PersonId from the request to the response and copies a bunch of literal strings to the other required nodes of the response. See Figure 3-6 for an example.

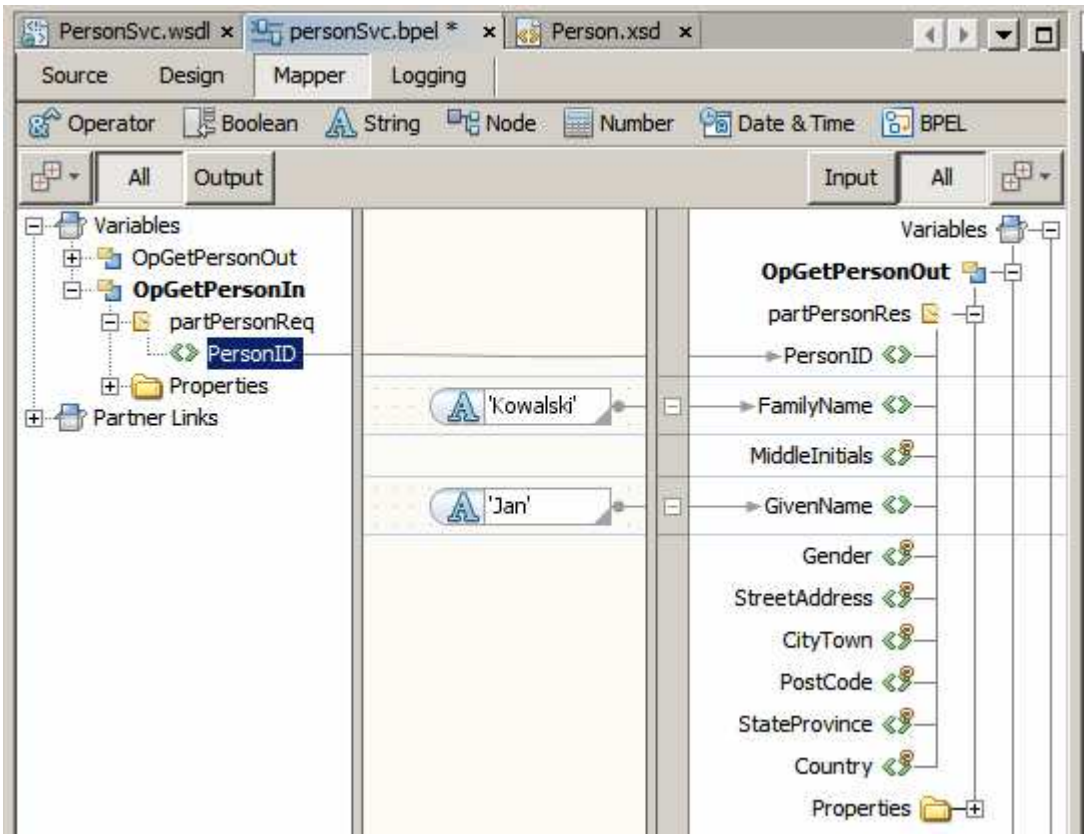

Figure 3-6 Example mapping

Build this project.

Create a New -> SOA -> Composite Application, named PersonSvc_CA. Drag the PersonSvc BPEL Module onto the CASA Editor canvas and build. Figure 3-7 illustrates a step in the process.

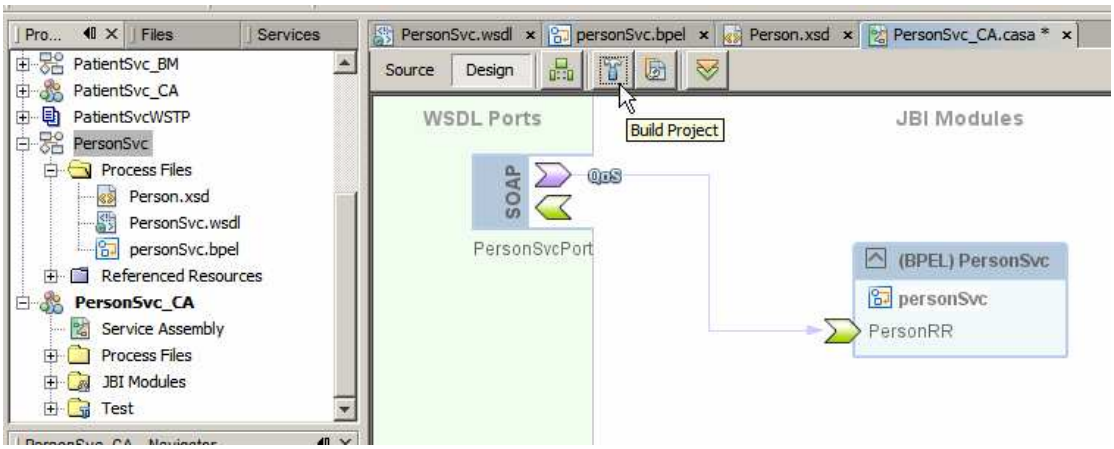

Figure 3-7 Composite Application built

Right-click the SOAP BC graphic and choose “Clone WSDL Port to Edit”, as shown in Figure 3-8.

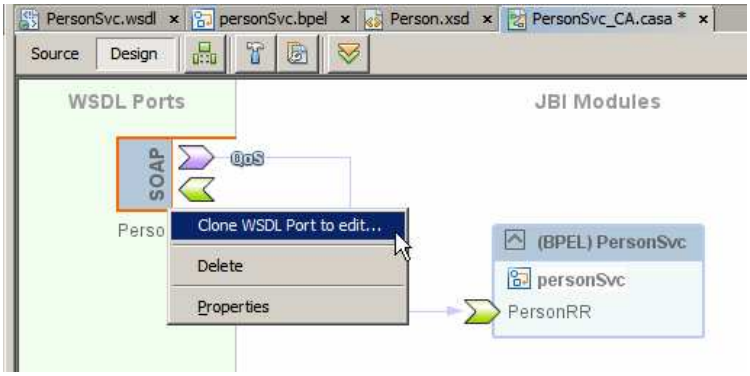

Figure 3-8 Clone WSDL Port

Click on the “Paper with Key” icon and choose “Server Configuration”, see Figure 3-9.

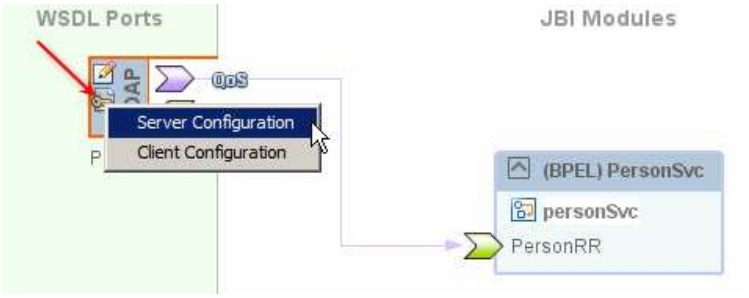

Figure 3-9 Edit Server Configuration

Check the “Secure Service” checkbox, leave “Username Authentication with Symmetric Key” and click OK. Figure 3-10 highlights the high spots.

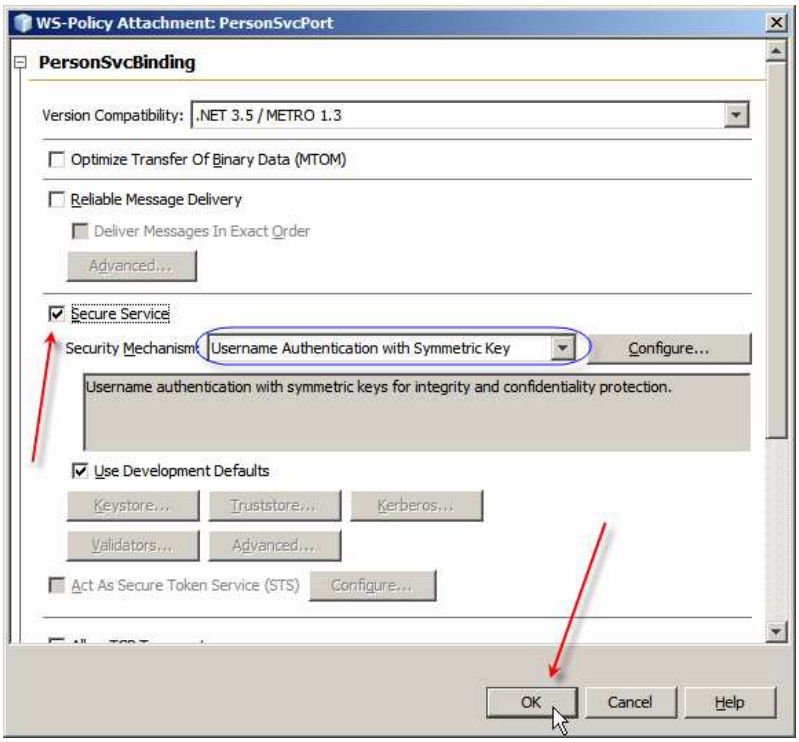

Figure 3-10 Apply security

Save the Composite Application.

When the Clone WSDL Port to Edit was selected the WSDL was cloned and a copy of the WSDL pas placed in the PersonSvc_CA -> Process Files -> PersonSvc folder, see Figure 3-11.

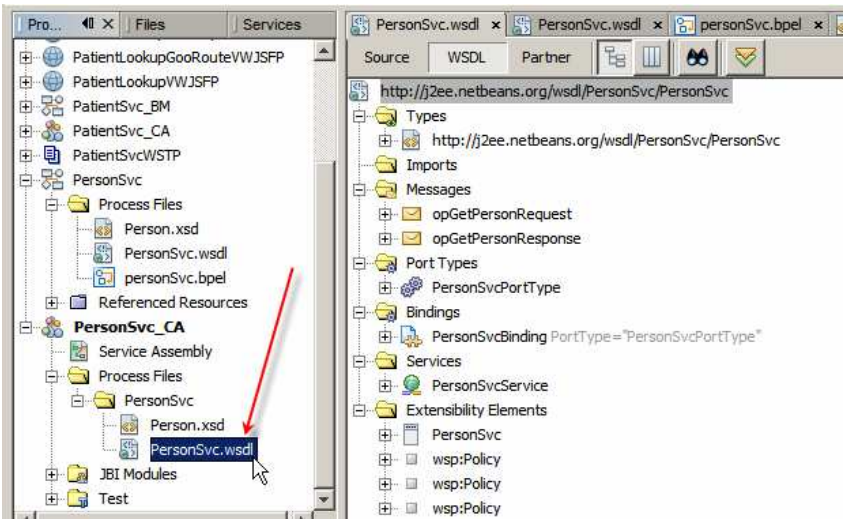

Figure 3-11 Clone of the original WSDL

This WSDL will be used, instead of the original one, when the service is built and deployed. This gives us an opportunity to add security policies without disturbing the original process. Open the WSDL in the editor, switch to Source mode and scroll all the way down to the bottom of the WSDL.

Select and delete the XML text highlighted in Figure 3-12.

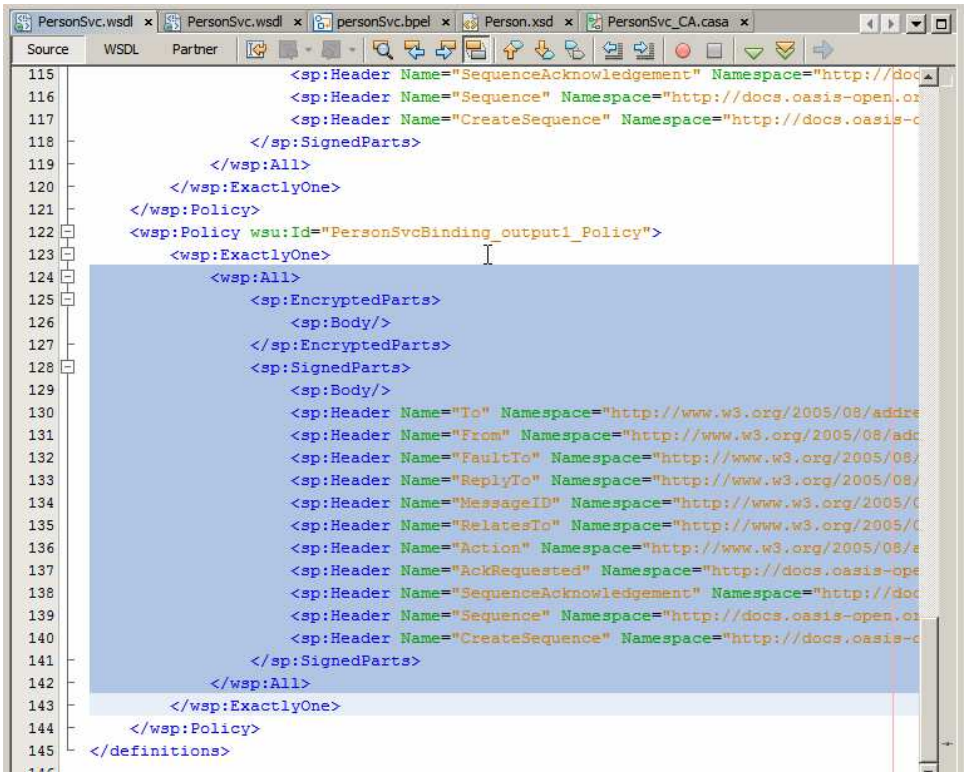

Figure 3-12 Delete Encryption and Signing content of the response message

Delete the XML text highlighted in Figure 3-13.

Figure 3-13 Delete Encryption and Signing content of the request message

The empty policies for the request and response message will remove the requirement to sign and encrypt parts of the message, which was produced by the wizard generating the policy. Insert "<wsp:All/>" in place of the removed fragments. Figure 3-14 illustrates the policy fragment after this change.

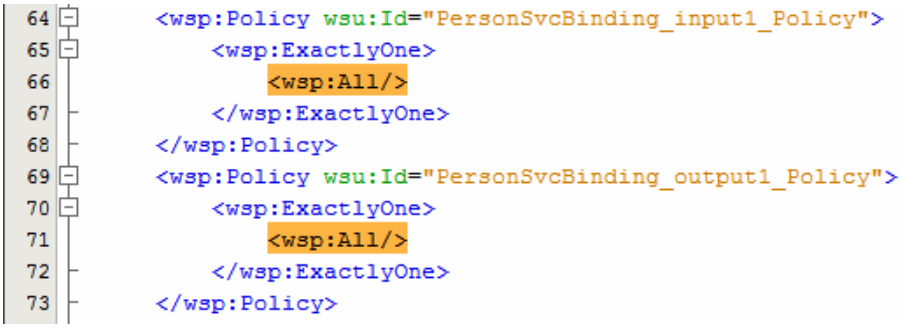

Figure 3-14 Policy with encryption and signing of request and response removed

One can completely dispense with the PersonSvcBinding_input1_Policy and the PersonSvcBinding_output1_Policy, which also requires removal of all references to these policies from elsewhere in the WSDL.

Remove XML text, highlighted in Figure 3-15, starting with the line that reads "`<wsam:Addressing wsp:Optional="false"/>`", thorough and including the line that reads "`</sp:Wss11>`".

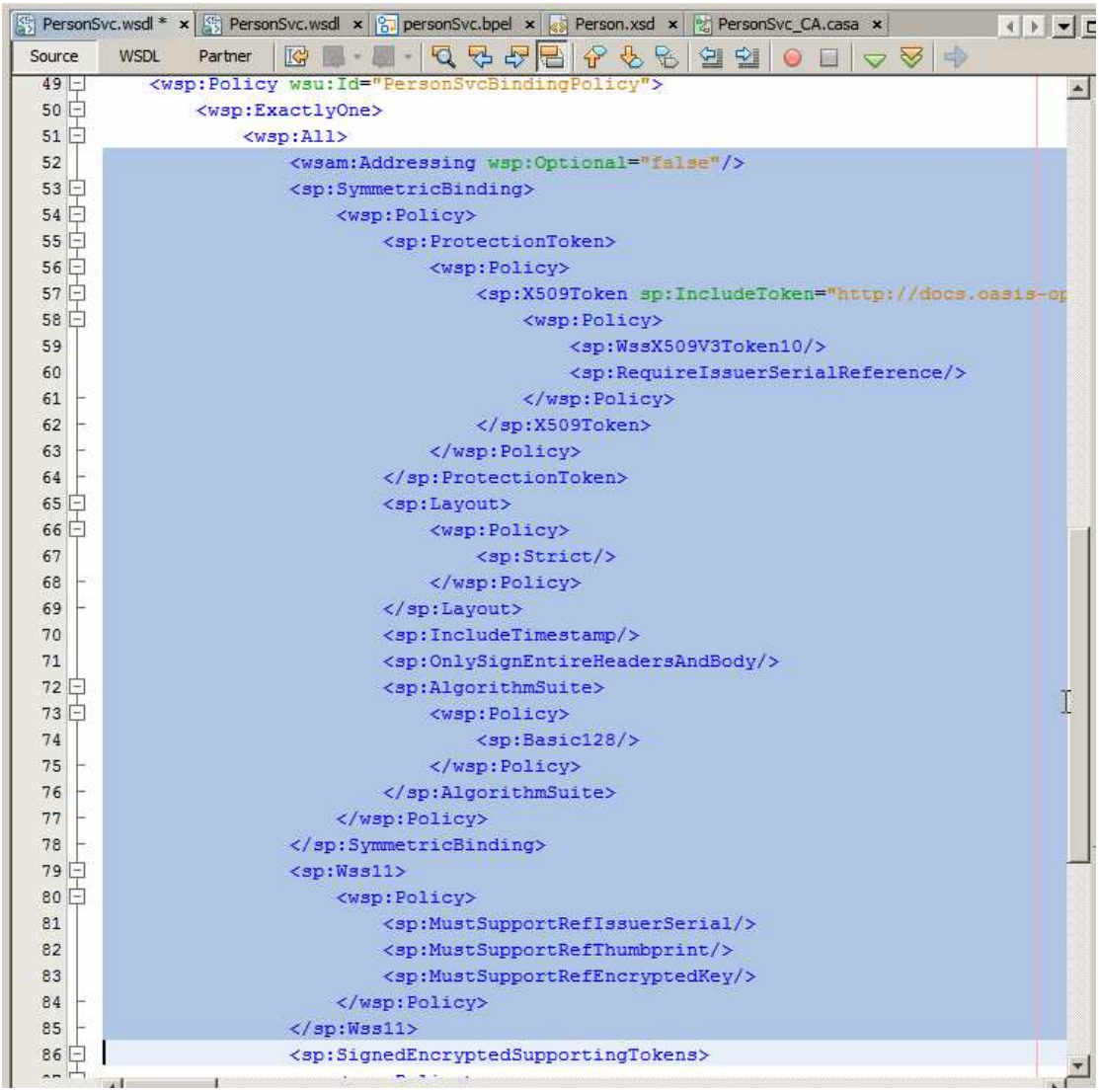

Figure 3-15 Remove remaining encryption and signing policy stanzas

In its place paste the following text:

```
<sp:SymmetricBinding>
      <wsp:Policy/>
</sp:SymmetricBinding>
```

Figure 3-16 show the resulting fragment.

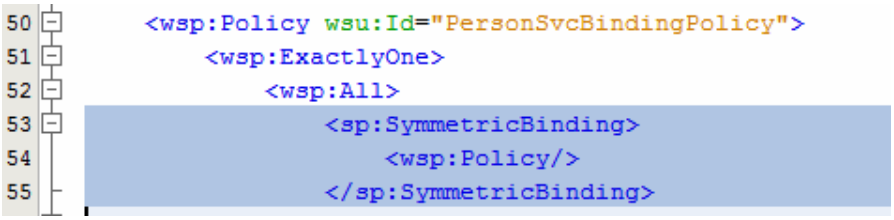

Figure 3-16 Replacement SymmetricBinding policy

One could remove the SymmetricBinding stanza completely, without affecting the server side, however doing this will prevent the client-side wizard from recognising

the Username token stanzas and will make it impossible to use the Metro wizard to configure client-side authentication. That section will be missing in the wizard.

Remove the line starting with "<sc:KeyStore", see Figure 3-17. Since we are no longer using cryptography there is no need for a KeyStore.

```
50      <wsp:Policy wsu:Id="PersonSvcBindingPolicy">
51          <wsp:ExactlyOne>
52              <wsp:All>
53                  <sp:SymmetricBinding>
54                      <wsp:Policy/>
55                  </sp:SymmetricBinding>
56                  <sp:SignedEncryptedSupportingTokens>
57                      <wsp:Policy>
58                          <sp:UsernameToken sp:IncludeToken="http://docs.oasis-op
59                              <wsp:Policy>
60                                  <sp:WssUsernameToken10/>
61                              </wsp:Policy>
62                          </sp:UsernameToken>
63                      </wsp:Policy>
64                  </sp:SignedEncryptedSupportingTokens>
65                  <sc:KeysStore wspp:visibility="private" type="JKS" storepass="c
66              </wsp:All>
67          </wsp:ExactlyOne>
```

Figure 3-17 Remove reference to the keystore

Change XML text "`sp:SignedEncryptedSupportingTokens`" to "`sp:EncryptedSupportingTokens`", in the two places where it occurs, see Figure 3-18.

```
51          <wsp:ExactlyOne>
52              <wsp:All>
53                  <sp:SymmetricBinding>
54                      <wsp:Policy/>
55                  </sp:SymmetricBinding>
56                  <sp:EncryptedSupportingTokens>
57                      <wsp:Policy>
58                          <sp:UsernameToken sp:IncludeToken="htt
59                              <wsp:Policy>
60                                  <sp:WssUsernameToken10/>
61                              </wsp:Policy>
62                          </sp:UsernameToken>
63                      </wsp:Policy>
64                  </sp:EncryptedSupportingTokens>
65              </wsp:All>
66          </wsp:ExactlyOne>
```

Figure 3-18 Change SignedEncryptedSupportingTokens to SupportingTokens

Save and close the modified WSDL. Build and Deploy the Composite Application.

Create a New Project -> Java EE -> Web Services Testing Project (assuming you have the SoapUI Plugin installed). Name this project PersonSvc_WSTP. Use the "soap:address location" property value, with the literal "?WSDL" appended and "${HttpDefaultPort}" variable replaced with the correct port (9080 for a default installation). Figure 3-19 illustrates this.

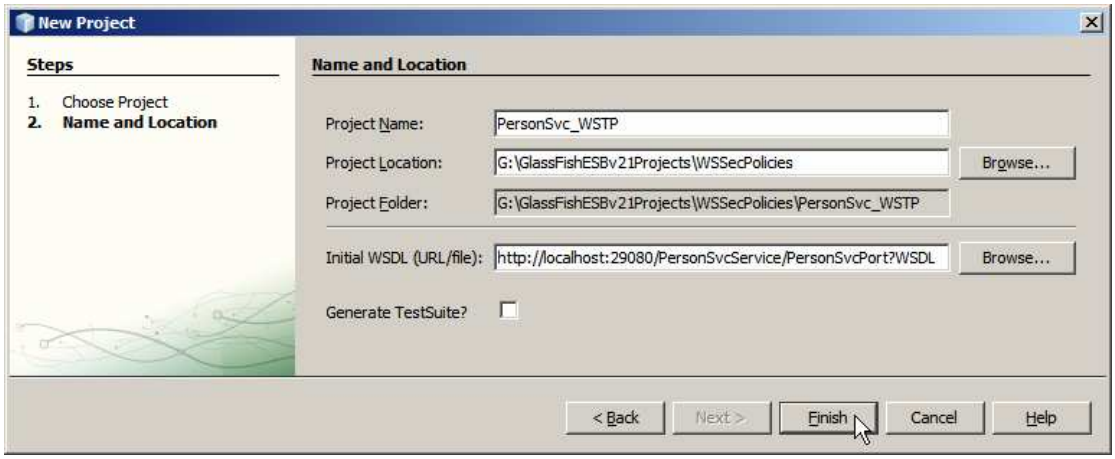

Figure 3-19 Create a Web Service Testing Project

Create a request and fill in PersonID with a number of you choice, see Figure 3-20.

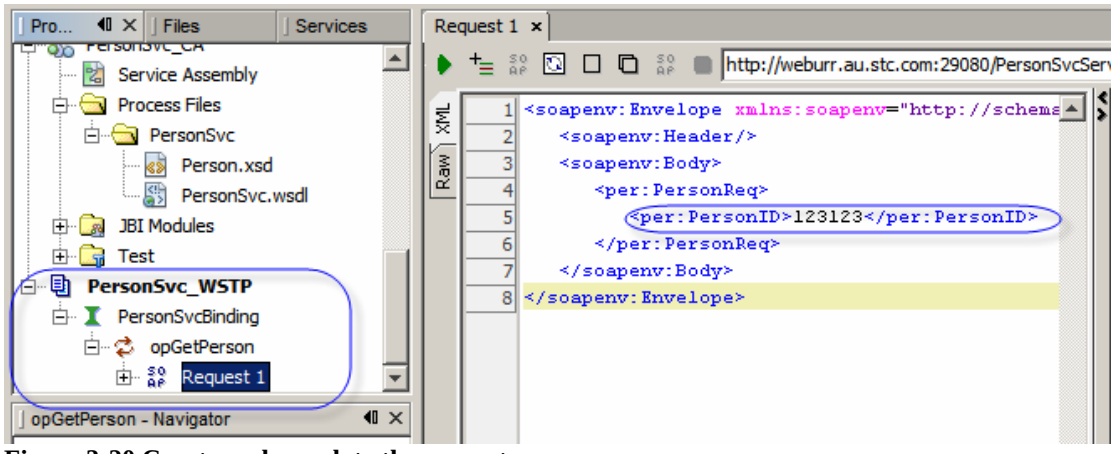

Figure 3-20 Create and populate the request message

Make sure Properties pane is visible. Enter aaa and aaa as Username and Password property values. Choose PlaintextPassword from the WSS-PassowrdType drop-down. See Figure 3-21.

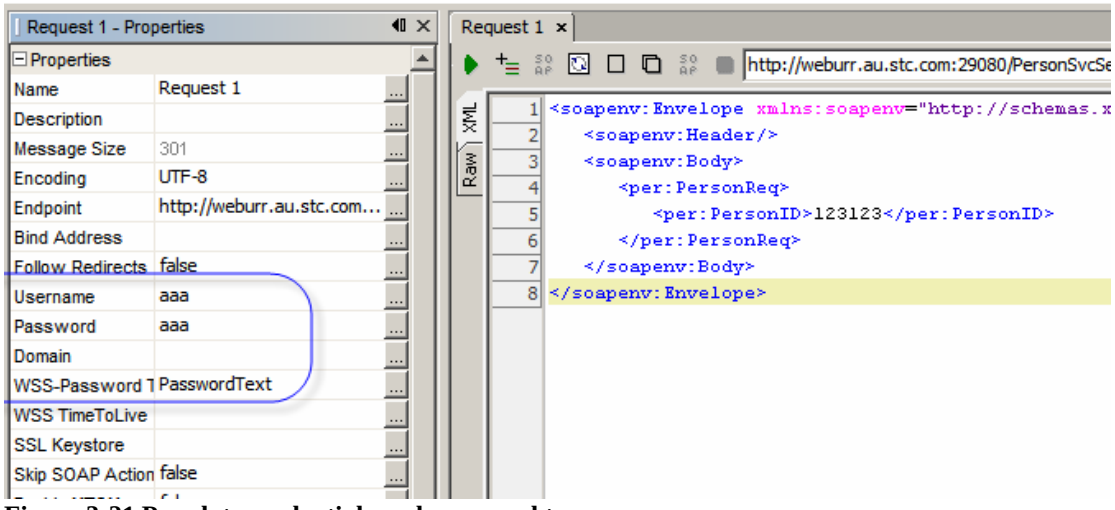

Figure 3-21 Populate credentials and password type

Submit the request and observe a Fault being returned.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd">wsse:FailedAuthentication</faultcode>
         <faultstring>Authentication of Username Password Token Failed</faultstring>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

We have not introduced the user to the GlassFish Application Server. Let's do this now. Switch to the Services Tab/Explorer, expand Servers node, right click on the GlassFisg=h v2 node and choose View Admin Console. Log in, expand Configuration -> Security -> Realms. Click File and click Manage users. Figure 3-22 illustrates this.

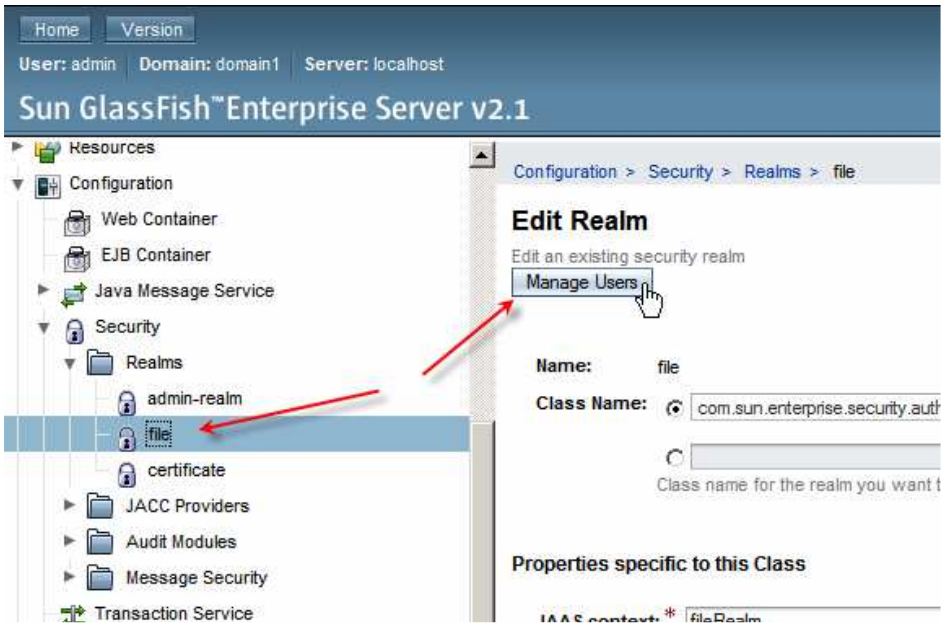

Figure 3-22 Add user process

Click New, enter aaa for username and aaa for password. Click OK, as shown in Figure 3-23.

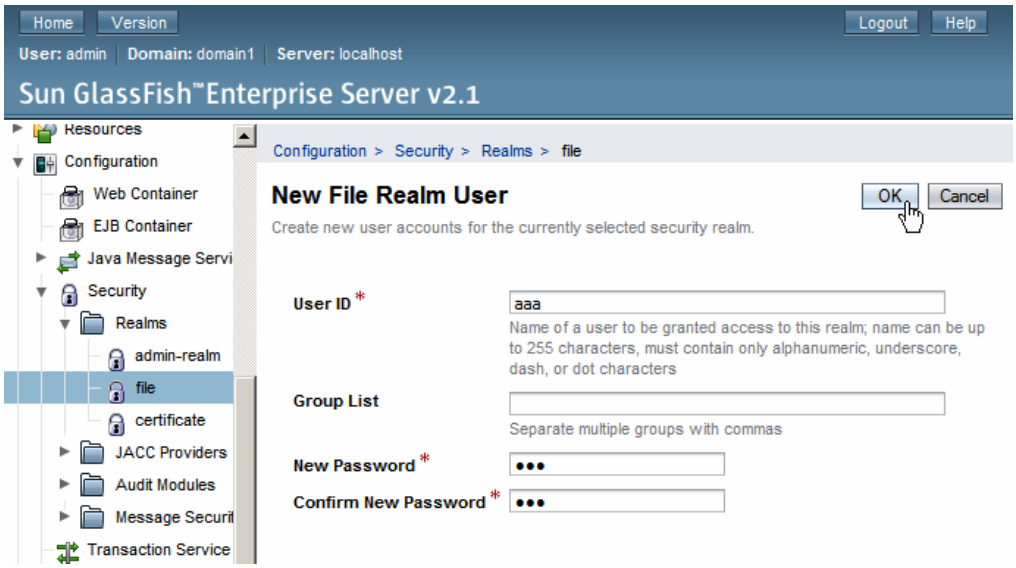

Figure 3-23 Add user aaa with password aaa

Close the console and submit the request again. The request results in an expected response see Figure 3-24.

Figure 3-24 Service response

To view, in greater detail, what is happening during the process, add the following to the GlassFish Applicatin Server's JVM Options:

```
-Dcom.sun.xml.ws.transport.local.LocalTransportPipe.dump=true
-Dcom.sun.xml.ws.assembler.server.transport=true
-Dcom.sun.xml.ws.transport.http.HttpAdapter.dump=true
-Dcom.sun.xml.ws.transport.http.client.HttpTransportPipe.dump=true
```

This will require application server restart.

Also add the following logging properties to the GlassFish Applicatin Server Logging categories:

```
com.sun.xml.wss.logging.impl.opt.level -> FINEST
com.sun.xml.wss -> FINEST
com.sun.xml.wss.logging.impl.opt.crypto.level -> FINEST
com.sun.xml.wss.logging.impl.opt.signature.level -> FINEST
com.sun.xml.ws -> FINEST
com.sun.xml.wss.logging.impl.opt.token.level -> FINEST
com.sun.xml.ws.api.pipe.Fiber -> INFO
com.sun.xml.ws.assembler -> FINEST
```

Submit the request again and see requests, responses, and other stuff logged in the server.log.

4 Build Consumer Composite Application

The consumer will be built using the WSDL exposed by the provider.

Create a New Project -> SOA -> BPEL Module, named PersonCli.

Create New -> External WSDL Document(s), using the service-exposed URL for the WSDL – for me this will be:

```
http://localhost:29080/PersonSvcService/PersonSvcPort?WSDL
```

For the default installation this will be:

```
http://localhost: 9080/PersonSvcService/PersonSvcPort?WSDL
```

Create a new WSDL Document, TriggerSvc, which will be used to trigger the client. Make this WSDL a concrete WSDL, SOAP Binding, Document Literal type, using PersonReq and PersonRes elements, from the Person schema, as input and output messages. Figures 4-1 and 4-3 illustrate two of the steps.

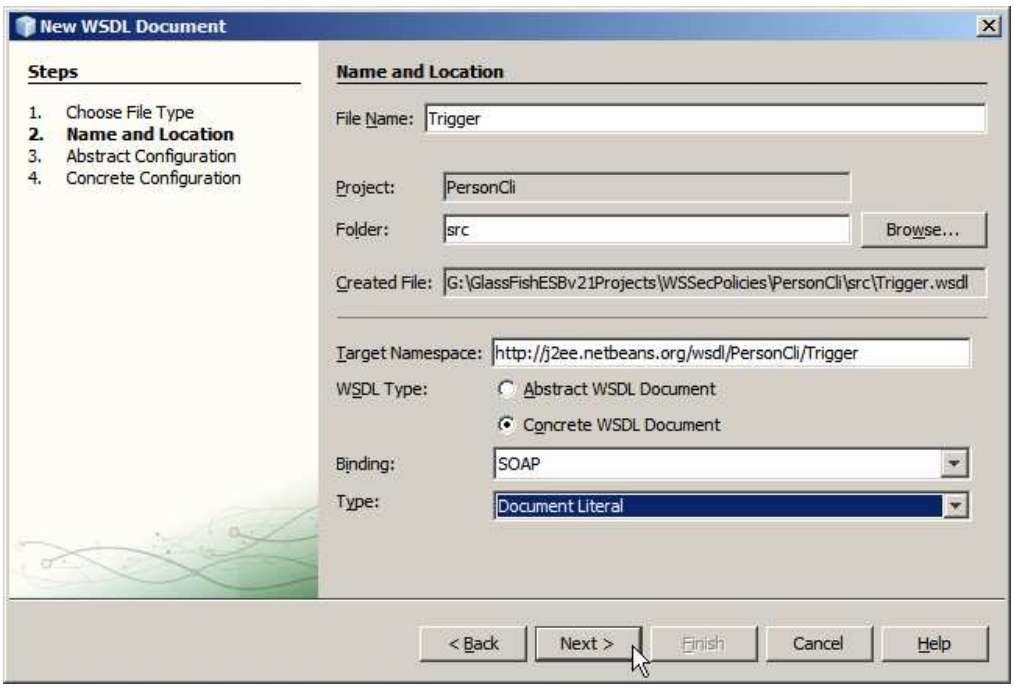

Figure 4-1 Name and configure the WSDL

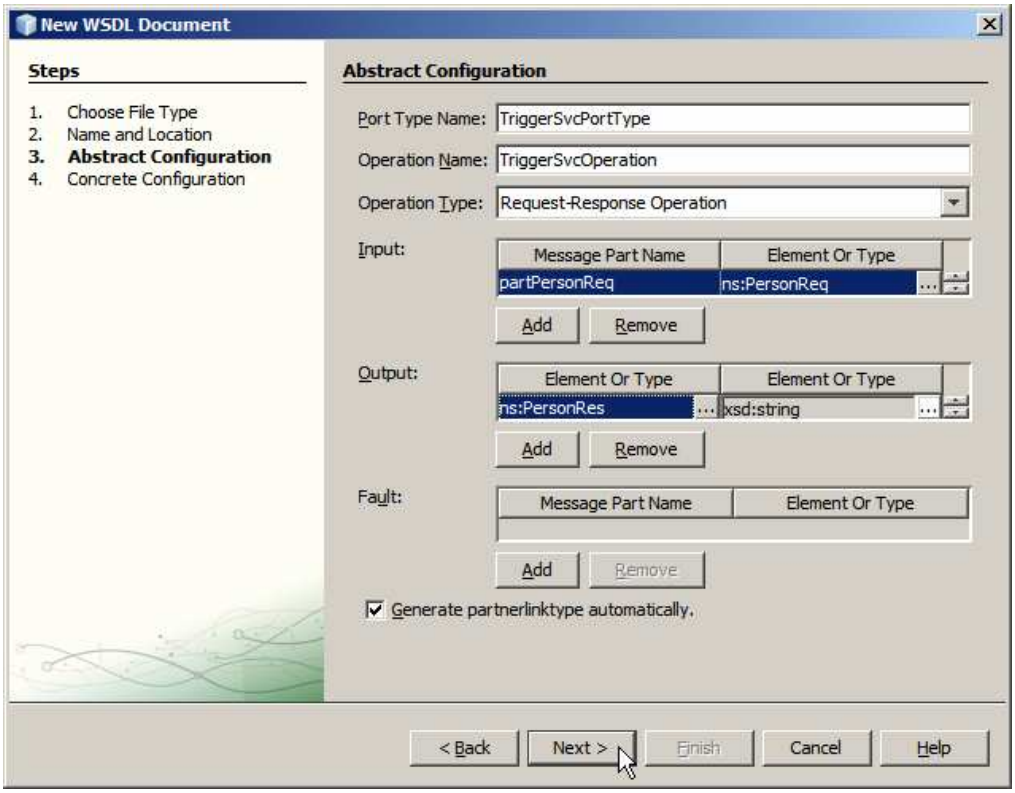

Figure 4-2 Choose Elements from the schema imported from the service

Drag the Trigger WSDL to the left swim line and the PersonSvc WSDL to the right swim line. Configure the process model as shown in Figure 4-3, making sure to add input and output variables for the Receive, Invoke and Reply activities.

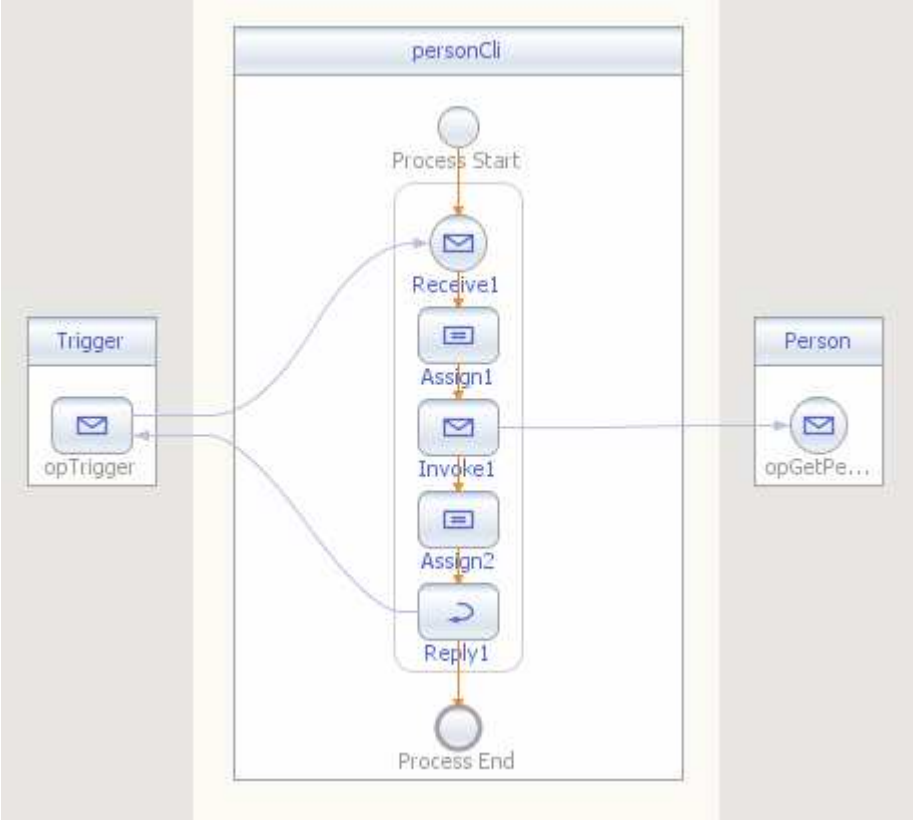

Figure 4-3 Process model

Configure the Assign1 activity as shown in Figure 4-4.

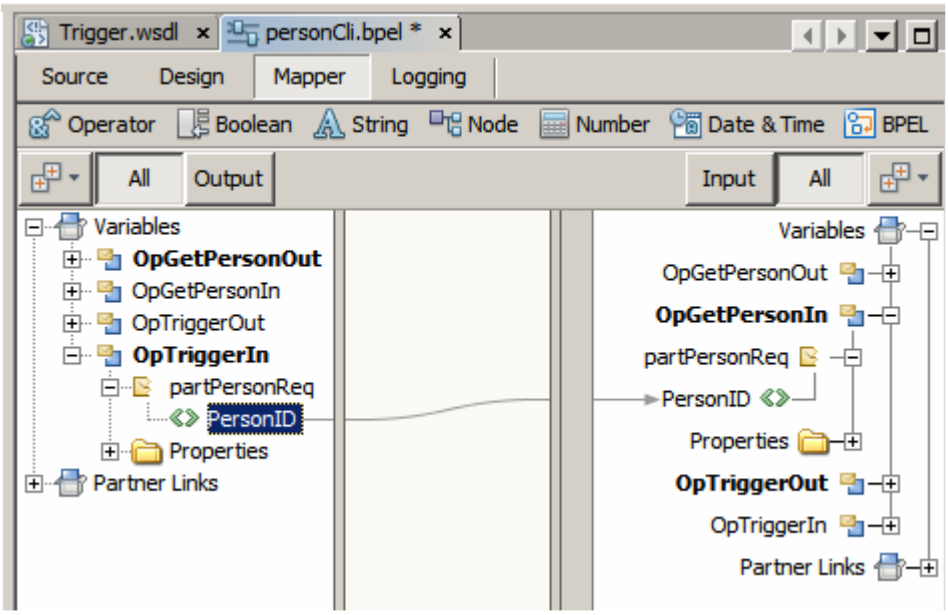

Figure 4-4 Mapping in Assign1 activity

Configure Assign2 activity as shown in Figure 4-5.

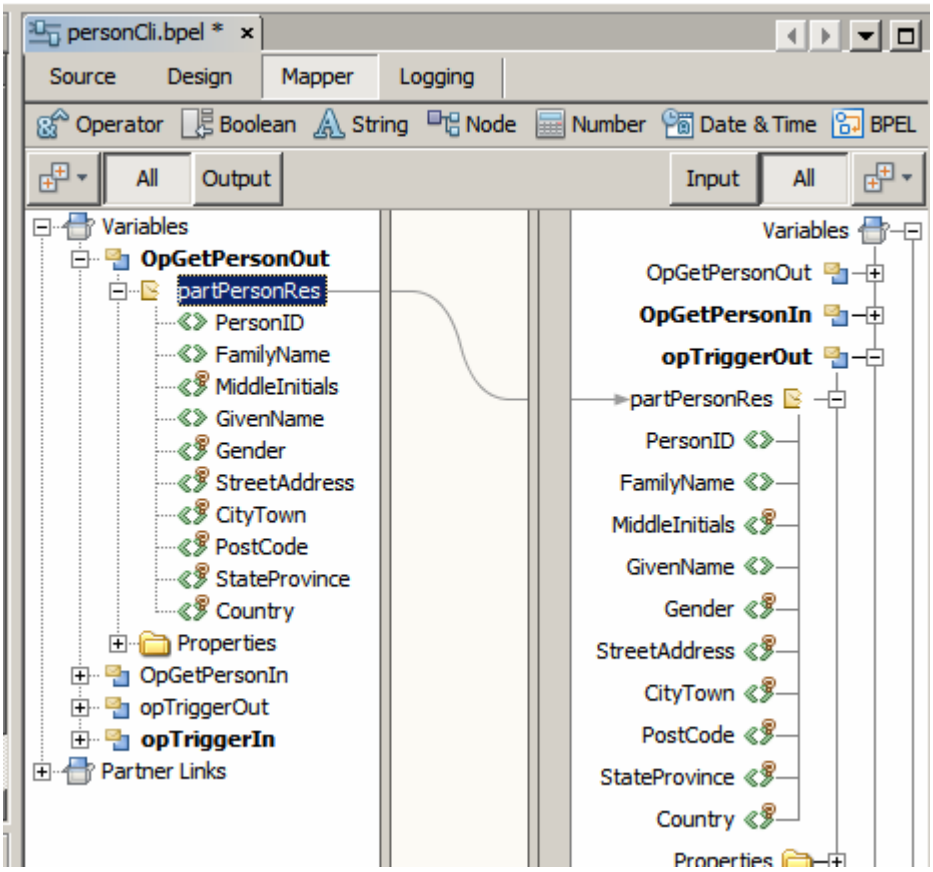

Figure 4-5 Mapping in Assign2 activity

Build the process.

Create a New Project -> SOA -> Composite Application, named PersonCli_CA. Drag the PersonCli BPLE Module onto the CASA Editor canvas and Build. Figure 4-6 illustrates this.

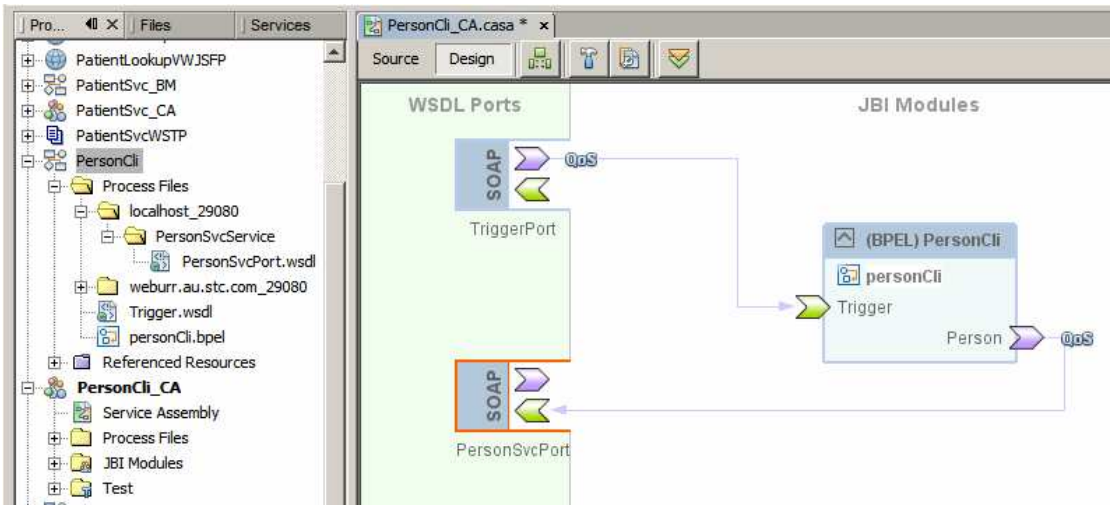

Figure 4-6 CASA Map with PersonCli BPEL Module

Right-click the PersonSvcPort SOAP BC icon and choose “Clone WSDL Port to edit …”.

Click the Paper and Key icon and choose Client Configuration, as shown in Figure 4-7.

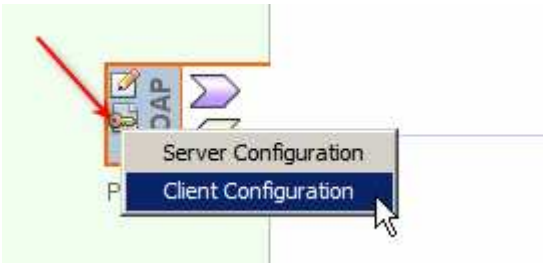

Figure 4-7 Edit Client Configuration

Enter aaa and aaa for the default username and password, as shown in Figure 4-8. Note that the Security part of this wizard would have been missing if we removed the empty SymmetricBinding policy in the server-side WSDL.

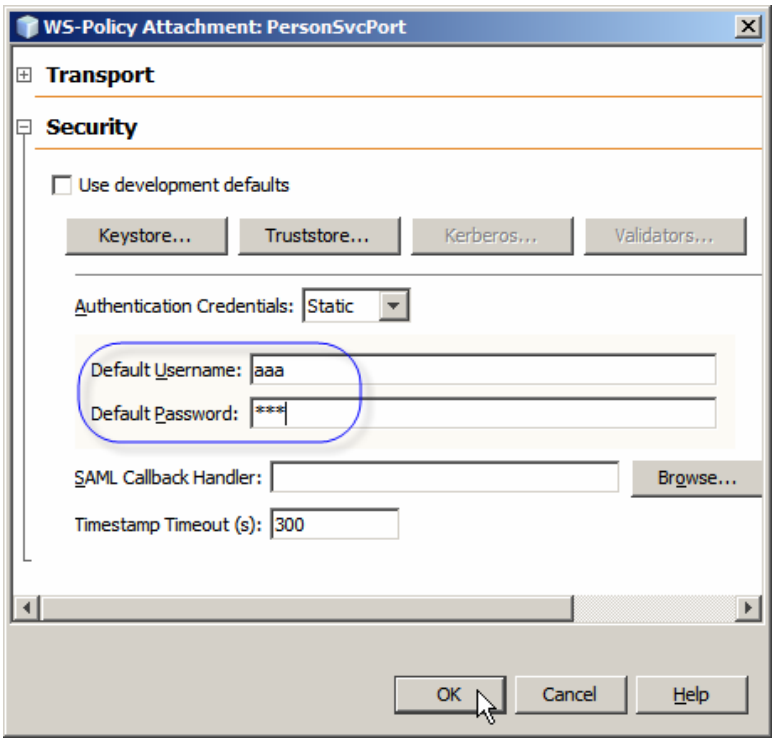

Figure 4-8 Provide static credentials

Build and Deploy the project.

Create a New Project -> Java EE -> Web Service Testing Project, named PersonCli_WSTP. Use the WSDL URL corresponding to the Trigger endpoint. For me this will be:

```
http://localhost:29080/TriggerSvcService/TriggerSvcPort?WSDL
```

For the default installation this will be:

```
http://localhost:9080/TriggerSvcService/TriggerSvcPort?WSDL
```

Create a New Request, populate it with a value and submit it. Figure 4-9 illustrates this.

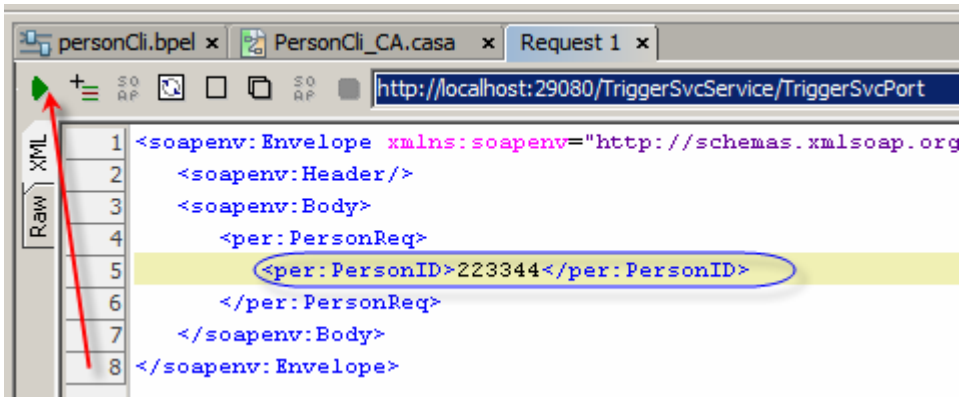

Figure 4-9 Create, populate and submit the request

The response comes through, as expected. See Figure 4-10.

Figure 4-10 Service response

Assuming you added the JVM Options and Logging Categories as suggested, a look at the server.log reveals the following course of events:

Plain SOAP request from SoapUI to PersonCli on the Trigger interface:

```
[#|2009-08-06T09:03:50.531+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=75;_ThreadN
ame=httpWorkerThread-29080-3;|
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:per="http://xml.netbeans.org/schema/Person">
   <soapenv:Header/>
   <soapenv:Body>
      <per:PersonReq>
         <per:PersonID>223344</per:PersonID>
      </per:PersonReq>
   </soapenv:Body>
</soapenv:Envelope>|#]
```

Security-decorated SOAP request from PersonCli to PersonSvc on the PersonSvc interface (note the Username token-related security headers added automaically):

```
[#|2009-08-06T09:03:51.062+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=79;_ThreadN
ame=HTTPBC-JAXWS-Engine-5;|
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <S:Header>
    <wsse:Security
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
S:mustUnderstand="1">
      <wsse:UsernameToken xmlns:ns14="http://docs.oasis-open.org/ws-
sx/ws-secureconversation/200512"
xmlns:ns13="http://www.w3.org/2003/05/soap-envelope"
wsu:Id="uuid_6b6bb0d2-a43d-4cb1-ab05-545c5f726cb5">
        <wsse:Username>aaa</wsse:Username>
        <wsse:Password Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordText">aaa</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </S:Header>
  <S:Body>
    <PersonReq
xmlns:msgns="http://j2ee.netbeans.org/wsdl/PersonSvc/PersonSvc"
xmlns="http://xml.netbeans.org/schema/Person">
      <PersonID>223344</PersonID>
    </PersonReq>
  </S:Body>
</S:Envelope>|#]
```

SOAP response from PersonSvc to PersonCli:

```
[#|2009-08-06T09:03:51.109+1000|INFO|sun-
appserver2.1|javax.enterprise.system.stream.out|_ThreadID=60;_ThreadN
ame=HTTPBC-JAXWS-Engine-1;|
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header />
  <SOAP-ENV:Body>
    <PersonRes
xmlns:msgns="http://j2ee.netbeans.org/wsdl/PersonSvc/PersonSvc"
xmlns="http://xml.netbeans.org/schema/Person">
      <PersonID>223344</PersonID>
      <FamilyName>Kowalski</FamilyName>
      <GivenName>Jan</GivenName>
    </PersonRes>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>|#]
```

Figure 4-10 shows that response in NetBeans.

Done.

5 Summary

This document discussed the configuration of the SOAP/HTTP Binding Component, in a service provider and in a service consumer, to implement plain text (unencrypted) WS-Security 1.0 (2004) Username Token Profile support.