Java CAPS 6/JBI and OpenESB
Using JBI, Note 5

HTTP BC to SMTP BC, No BPEL

Michael Czapski, July 2008

Table of Content

1 Introduction

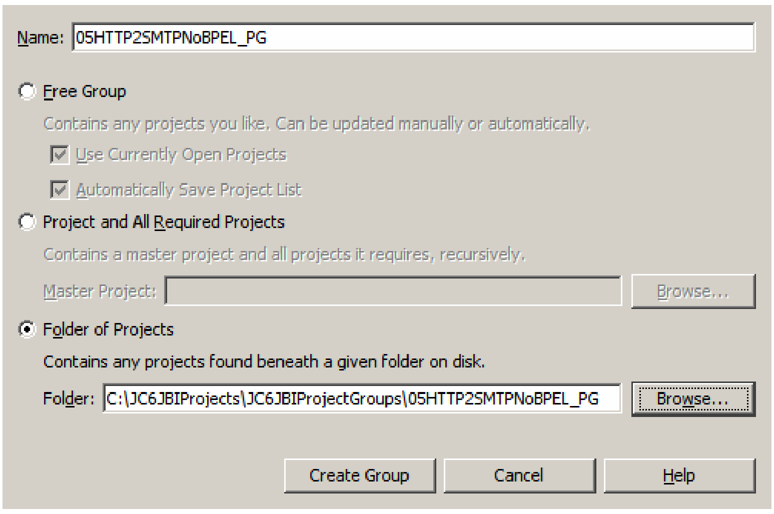
Someone asked a question along the lines of "Is it possible to develop a solution in OpenESB where the HTTP BC receives a request and the SMTP BC uses it to send electronic mail with no BPEL logic to tie the two together". I though that the answer was "Yes" but I felt I had to verify it. Vishnuvardhan Piskalaramesh from Sun, who is looking after the SMTP BC, and Sherry Weng from Sun, who is looking after the HTTP BC, helped along and here is the result.

This note describes, with illustrations, a mini integration solution wherein a appropriately formulated HTTP GET request is used to submit an electronic mail to a SMTP server, using the HTTP Binding Component and the SMTP Binding Component, without the need to provide any transformation logic. This is another example where a practical JBI-based integration solution can be constructed in minutes.

2 Create Project Group 05HTTP2SMTPNoBPEL_PG

As on previous occasions, let's create a new project group to contain projects we will be building in this Note. The group will be called "05HTTP2SMTPNoBPEL_PG".

Let's right-click anywhere in the Projects tab and choose Project Groups -> New Group … to start the wizard. Figure 2-1 shows the dialog box where project group name and file system directory path are specified.


Name: 05HTTP2SMTPNoBPEL_PG

( ) Free Group

Contains any projects you like. Can be updated manually or automatically.

☑ Use Currently Open Projects

☑ Automatically Save Project List

( ) Project and All Required Projects

Contains a master project and all projects it requires, recursively.

Master Project:

(●) Folder of Projects

Contains any projects found beneath a given folder on disk.

Folder: C:\JC6JBIProjects\JC6JBIProjectGroups\05HTTP2SMTPNoBPEL_PG

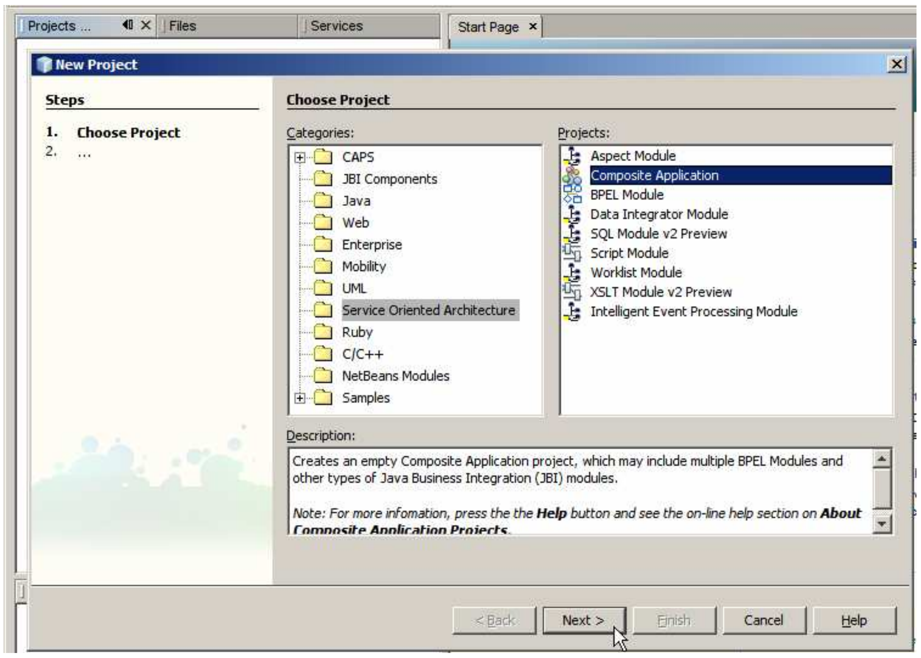
Figure 2-1 Creating a new Project Group

3 Create HTTP BC to SMTP BC Composite App

It is sometimes assumed that BPEL 2.0 process is necessary to route messages between an inbound Binding Component and an outbound Binding Component. As has been demonstrated in Java CAPS Field Technical Tips Notes JBI Note 3 and JBI Note 2, it is not necessary to have a BPEL 2.0 process do the routing. NMR will happily route between two Binding Components as long as no business logic is required to transform messages as they travel between the source and the destination.

The Notes mentioned above used the File BC as the illustrative vehicle. It should be easy enough to extrapolate to other BCs. To assist in the extrapolation this document discusses the OpenESB solution that uses the HTTP BC as the source of messages and the SMTP BC as the destination.

Let's create a Composite Application Module, HTTP2SMTP_CA. Figures 3-1 and 3-2 illustrate the major steps.


New Project

Steps
1. Choose Project
2. ...

Choose Project

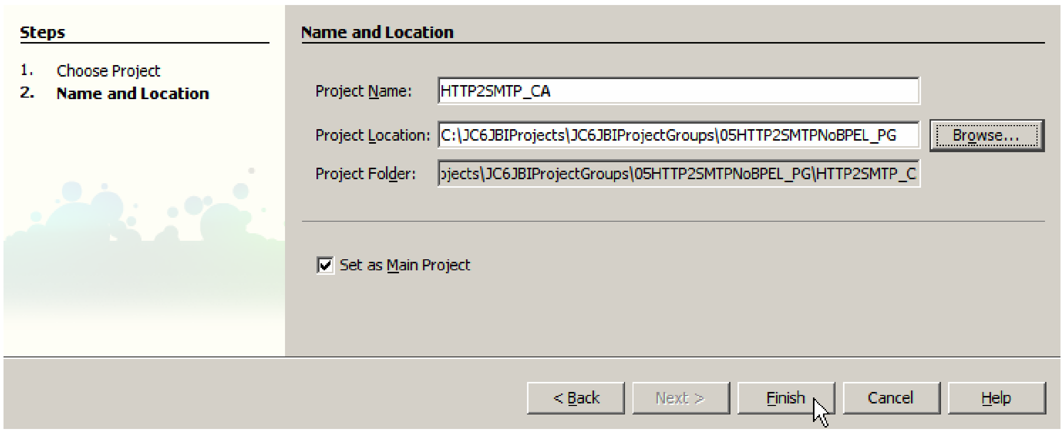
Categories: CAPS, JBI Components, Java, Web, Enterprise, Mobility, UML, Service Oriented Architecture, Ruby, C/C++, NetBeans Modules, Samples

Projects: Aspect Module, Composite Application, BPEL Module, Data Integrator Module, SQL Module v2 Preview, Script Module, Worklist Module, XSLT Module v2 Preview, Intelligent Event Processing Module

Description:
Creates an empty Composite Application project, which may include multiple BPEL Modules and other types of Java Business Integration (JBI) modules.

Note: For more infomation, press the the **Help** button and see the on-line help section on **About Composite Application Projects**.

Figure 3-1 Choose Composite Application project



Project Name: HTTP2SMTP_CA
Project Location: C:\JC6JBIProjects\JC6JBIProjectGroups\05HTTP2SMTPNoBPEL_PG
Project Folder: ›jects\JC6JBIProjectGroups\05HTTP2SMTPNoBPEL_PG\HTTP2SMTP_C
☑ Set as Main Project

Figure 3-2 Name the project and nominate project folder

Let’s create a new WSDL, wsdlSMTPOut, for the SMTP BC and configure it with the correct physical environment property values, including sender, receiver, etc.. Figures 3-3 through 3-6 illustrate the major steps.

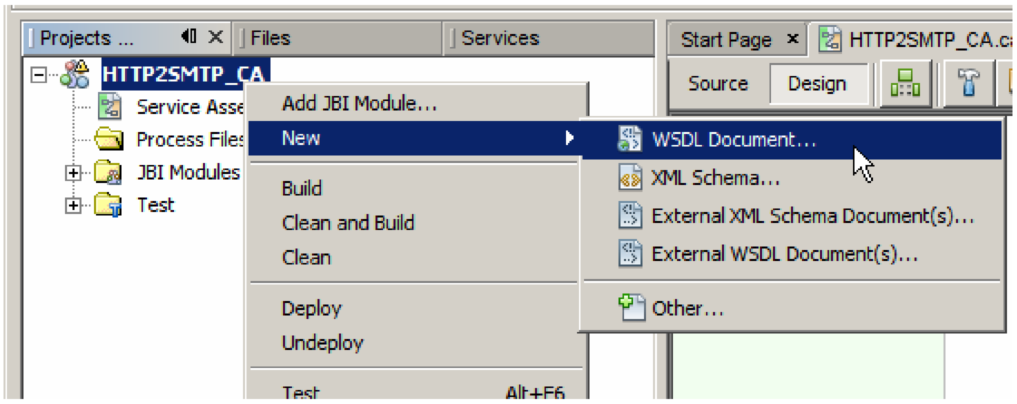
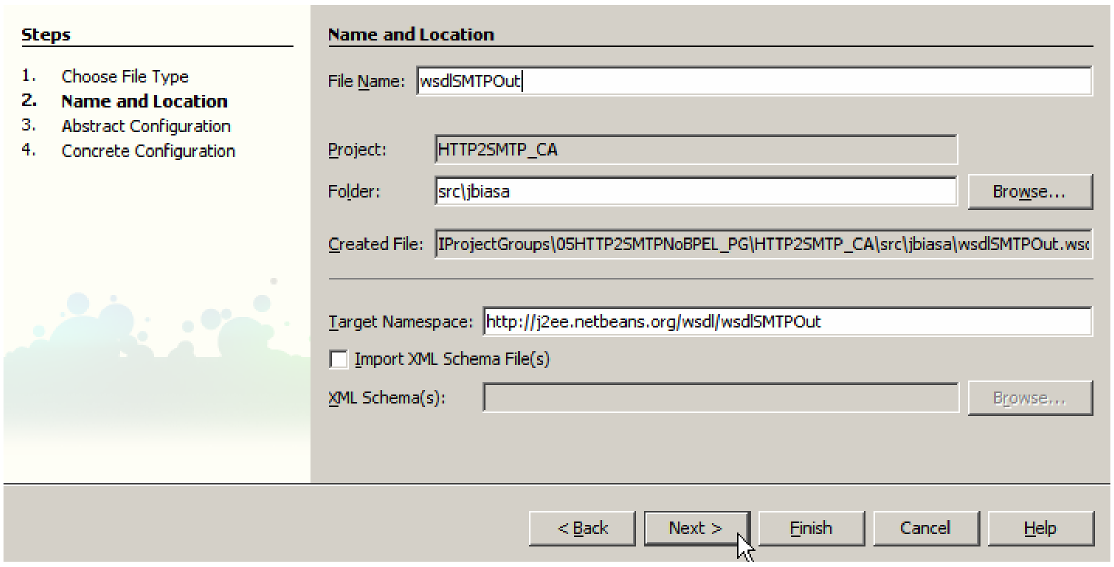
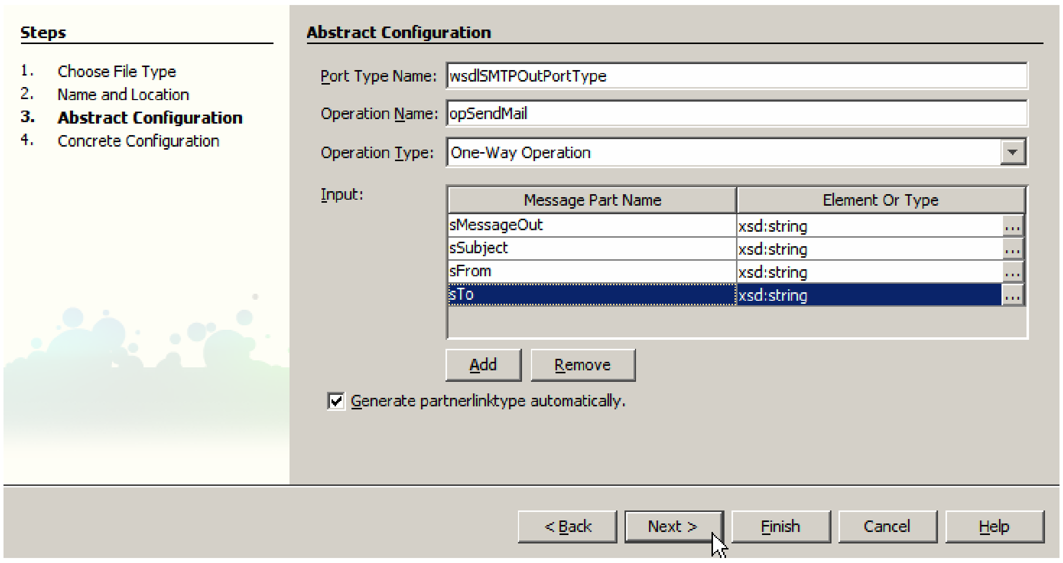


Figure 3-3 Choose to create a WSDL document



File Name: wsdlSMTPOut
Project: HTTP2SMTP_CA
Folder: src\jbiasa
Created File: lProjectGroups\05HTTP2SMTPNoBPEL_PG\HTTP2SMTP_CA\src\jbiasa\wsdlSMTPOut.wsc
Target Namespace: http://j2ee.netbeans.org/wsdl/wsdlSMTPOut
☐ Import XML Schema File(s)
XML Schema(s):

Figure 3-4 Name the WSDL document



Steps

1. Choose File Type
2. Name and Location
3. **Abstract Configuration**
4. Concrete Configuration

Abstract Configuration

Port Type Name: wsdlSMTPOutPortType

Operation Name: opSendMail

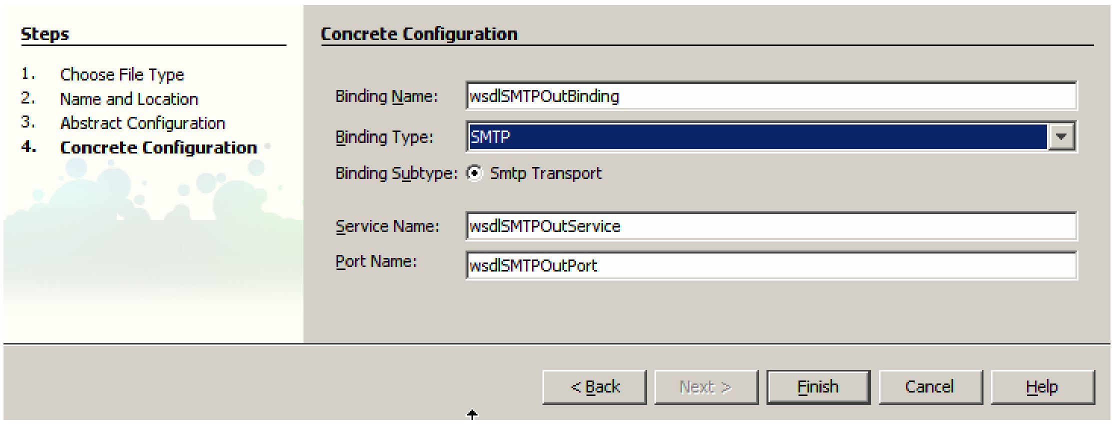
Operation Type: One-Way Operation

Input:

Message Part Name	Element Or Type
sMessageOut	xsd:string
sSubject	xsd:string
sFrom	xsd:string
sTo	xsd:string

☑ Generate partnerlinktype automatically.

Figure 3-5 Add message parts



Steps

1. Choose File Type
2. Name and Location
3. Abstract Configuration
4. **Concrete Configuration**

Concrete Configuration

Binding Name: wsdlSMTPOutBinding

Binding Type: SMTP

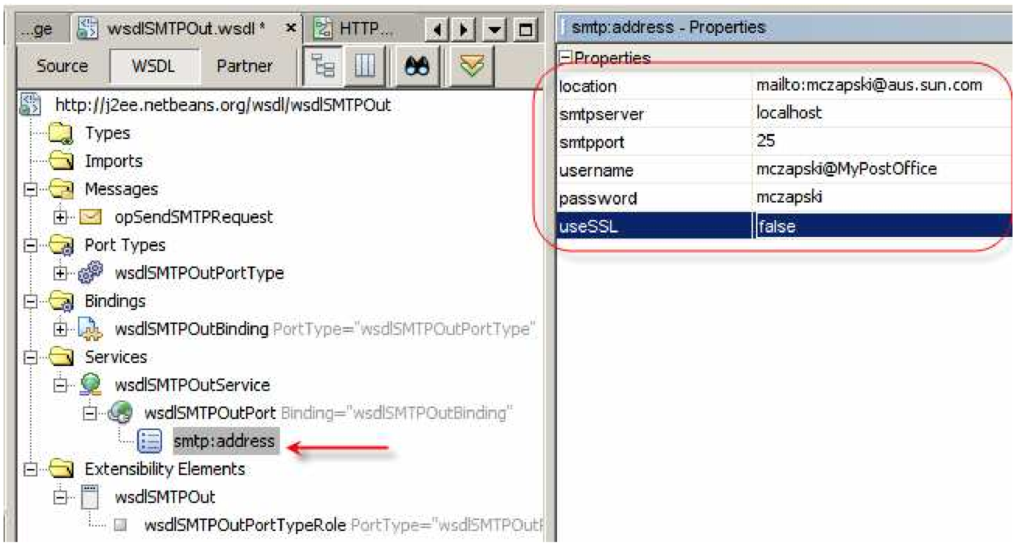
Binding Subtype: ● Smtp Transport

Service Name: wsdlSMTPOutService

Port Name: wsdlSMTPOutPort

Figure 3-6 Nominate the Binding Component to use

Let’s configure the smtp:address node under the Services tree to point at the appropriate SMTP Server and provide correct credentials. Figure 3- illustrates this. The From and To addresses will come from the message so the Location property seems to be superfluous. I configured it anyway.



```
http://j2ee.netbeans.org/wsdl/wsdlSMTPOut
  Types
  Imports
  Messages
    opSendSMTPRequest
  Port Types
    wsdlSMTPOutPortType
  Bindings
    wsdlSMTPOutBinding PortType="wsdlSMTPOutPortType"
  Services
    wsdlSMTPOutService
      wsdlSMTPOutPort Binding="wsdlSMTPOutBinding"
        smtp:address
  Extensibility Elements
    wsdlSMTPOut
      wsdlSMTPOutPortTypeRole PortType="wsdlSMTPOut
```

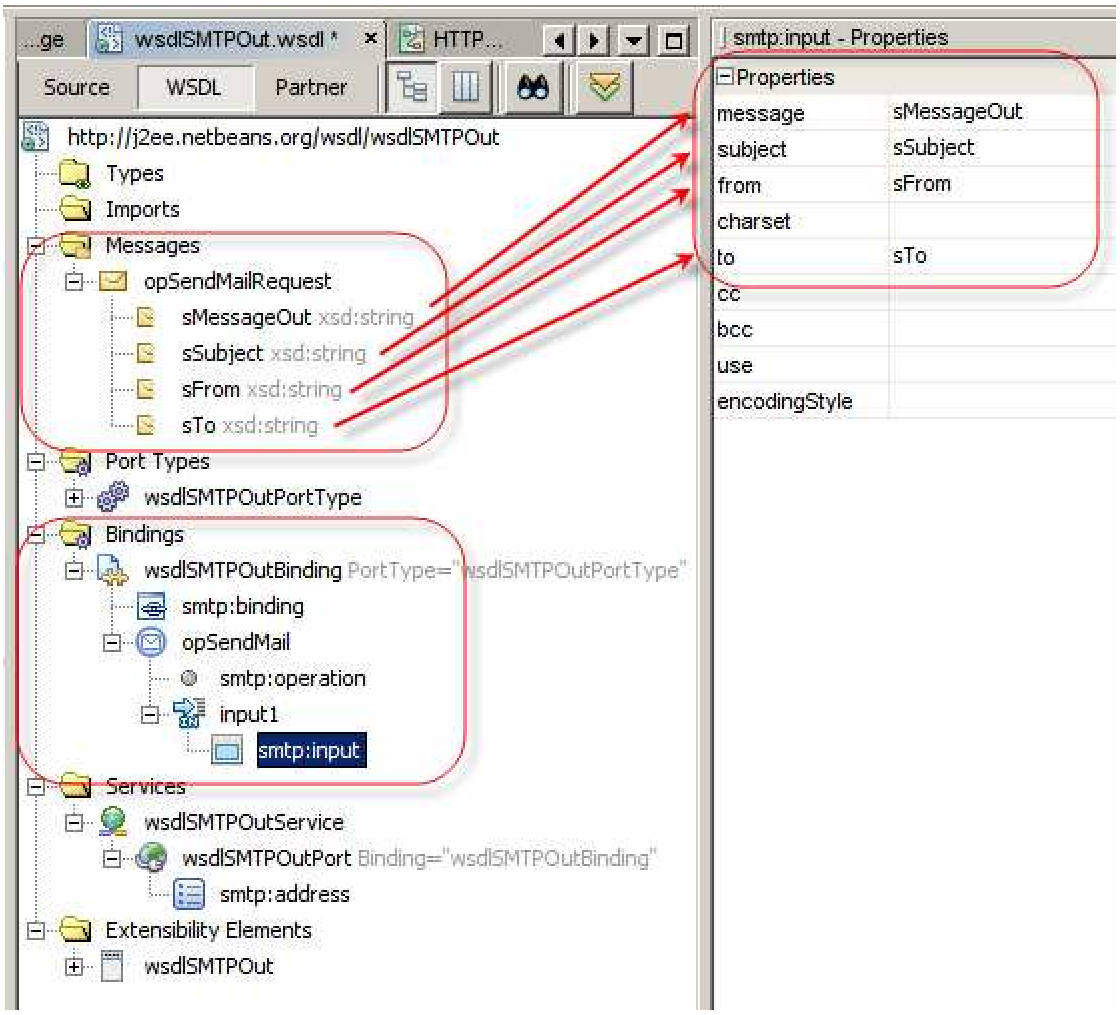
smtp:address - Properties

Properties

location	mailto:mczapski@aus.sun.com
smtpserver	localhost
smtpport	25
username	mczapski@MyPostOffice
password	mczapski
useSSL	false

Figure 3-7 Configuring address properties for the SMTP Server

The smtp:input property under the Bindings tree, is used to associate various properties relating to message sending with the parts of the message used to provide their values. Recall that the message in the WSDL we constructed for the SMTP BC has four parts. sMessageOut will contain message body, sSubject will contain mail subject text, sFrom will contain sender's address and sTo will contain recipient's address. We configure message, subject, from and to properties by selecting the corresponding message parts from the drop down. Figure 3-8 illustrates associatin of message parts and properties. Should we wished to use CC and BCC addresses we would have to provide appropriate message parts through which to supply values.



```
http://j2ee.netbeans.org/wsdl/wsdlSMTPOut
  Types
  Imports
  Messages
    opSendMailRequest
      sMessageOut xsd:string
      sSubject xsd:string
      sFrom xsd:string
      sTo xsd:string
  Port Types
    wsdlSMTPOutPortType
  Bindings
    wsdlSMTPOutBinding PortType="wsdlSMTPOutPortType"
      smtp:binding
      opSendMail
        smtp:operation
        input1
          smtp:input
  Services
    wsdlSMTPOutService
      wsdlSMTPOutPort Binding="wsdlSMTPOutBinding"
        smtp:address
  Extensibility Elements
    wsdlSMTPOut
```

Properties	
message	sMessageOut
subject	sSubject
from	sFrom
charset	
to	sTo
cc	
bcc	
use	
encodingStyle	

Figure 3-8 Associating message parts with configuration properties

To use the WSDL in the Service Assembly we load it.

Let's load the SMTP WSDL Port into the Service Assembly Editor canvas by right-clicking anywhere in the empty WSDL Ports swim line, selecting Load WSDL Port … option, Figure 3-9, and selecting the only WSDL port available – that of the SMTP BC WSDL, Figure 3-10.

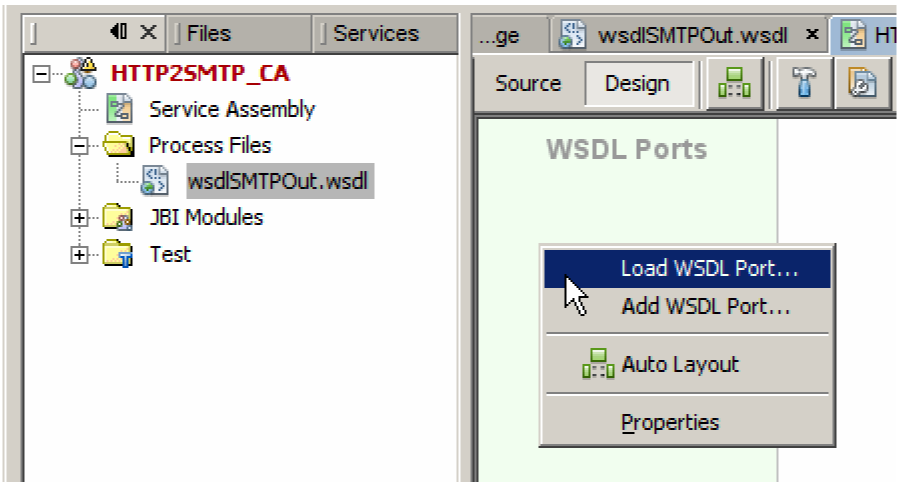


Figure 3-9 Loading SMTP WSDL into the Service Assembly

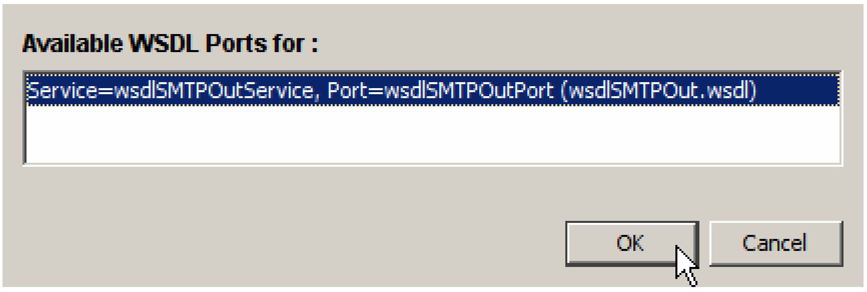

Available WSDL Ports for :
Service=wsdlSMTPOutService, Port=wsdlSMTPOutPort (wsdlSMTPOut.wsdl)
OK
Cancel

Figure 3-10 Selecting WSDL Port

This places the configured SMTP BC on the Service Assembly Editor canvas, Figure 3-11.

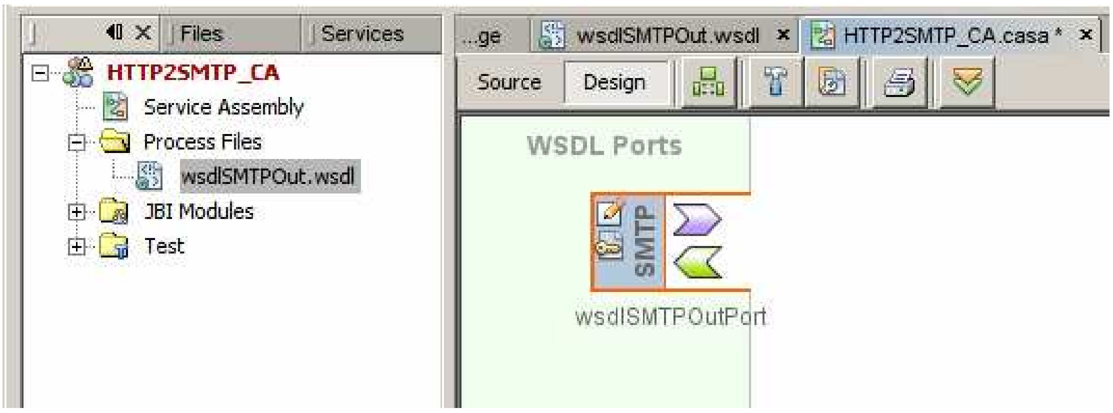


Figure 3-11 SMTP BC on the Service Assembly canvas

Let’s now drag the HTTP BC onto the Service Assembly Editor canvas as shjown in Figure 3-12.

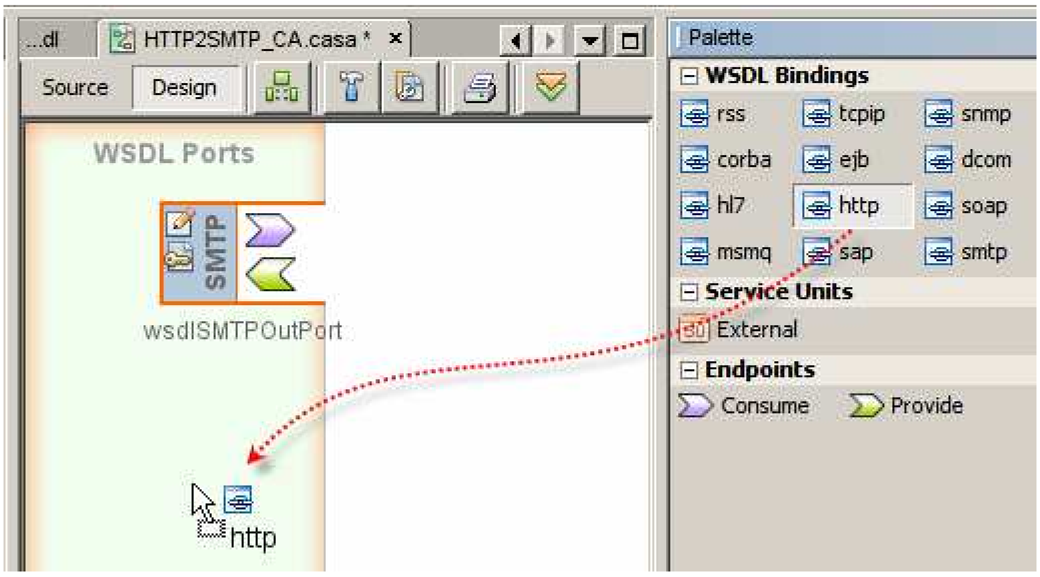


Figure 3-12 Adding HTTP BC to the canvas

This creates a new WSDL in the Project Files folder as shown in Figure 3-13.

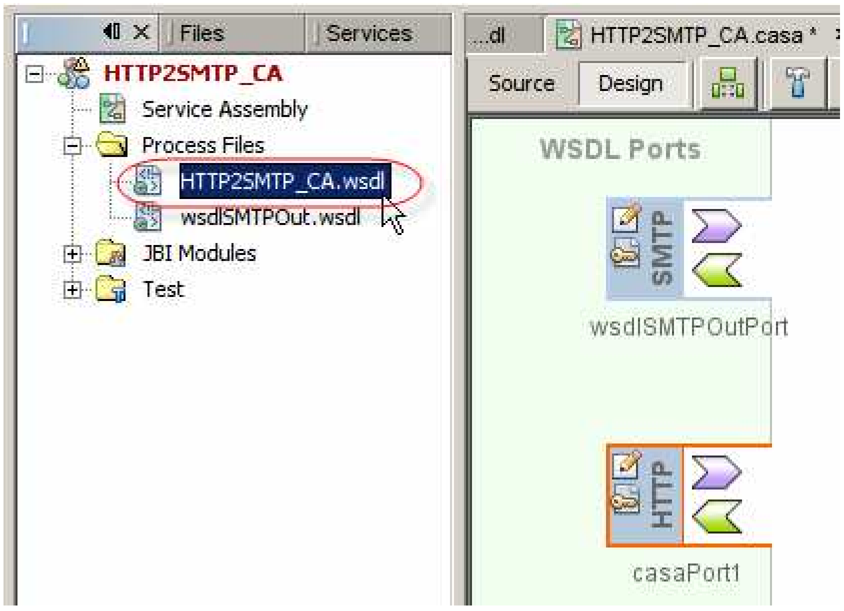


Figure 3-13 New WSDL in the process Files folder

Let’s connect the HTTP BC and the SMTP BC on the Service Assembly Editor canvas by dragging from the HTTP BC’s Consume icon to the SMTP BC’s Provide icon. Figure 3-14 illustrates this.

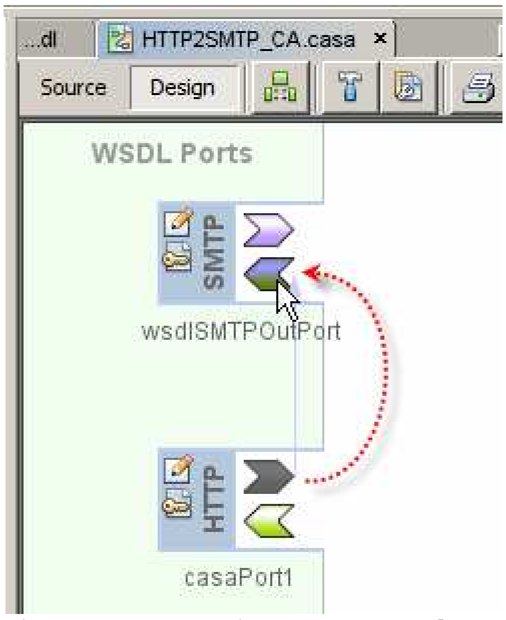
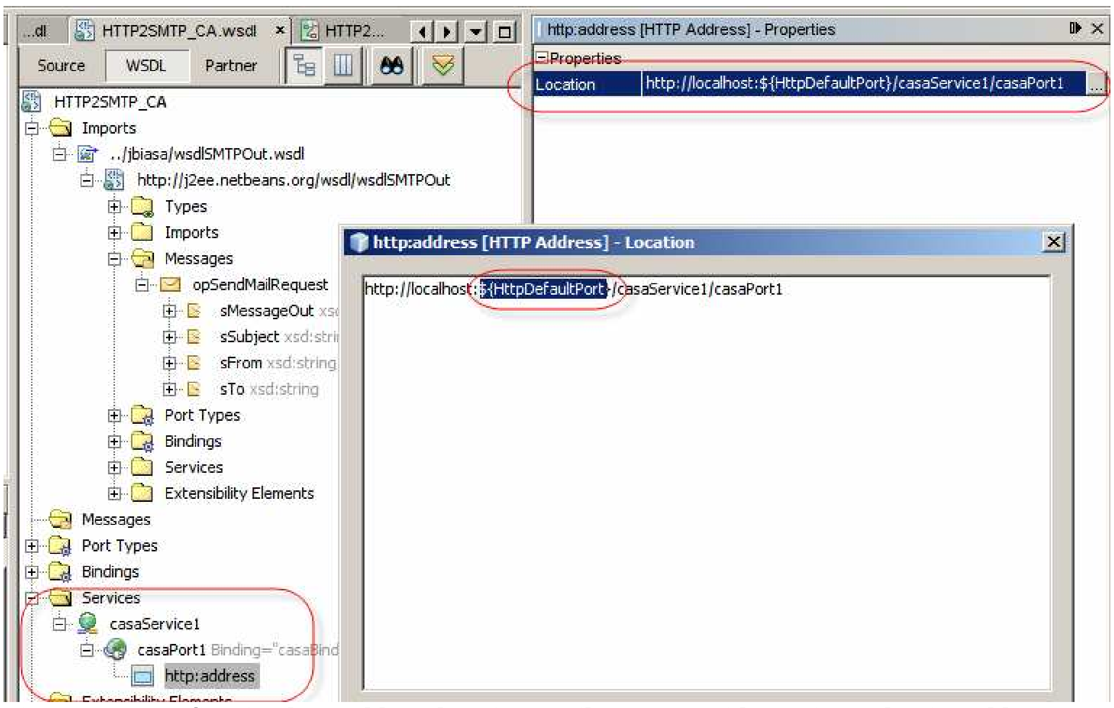


Figure 3-14 Connecting HTTP BC and SMTP BC

Let's now configure the HTTP BC's WSDL to provide the end point address information as shown in Figure 3-15.



Location http://localhost:${HttpDefaultPort}/casaService1/casaPort1

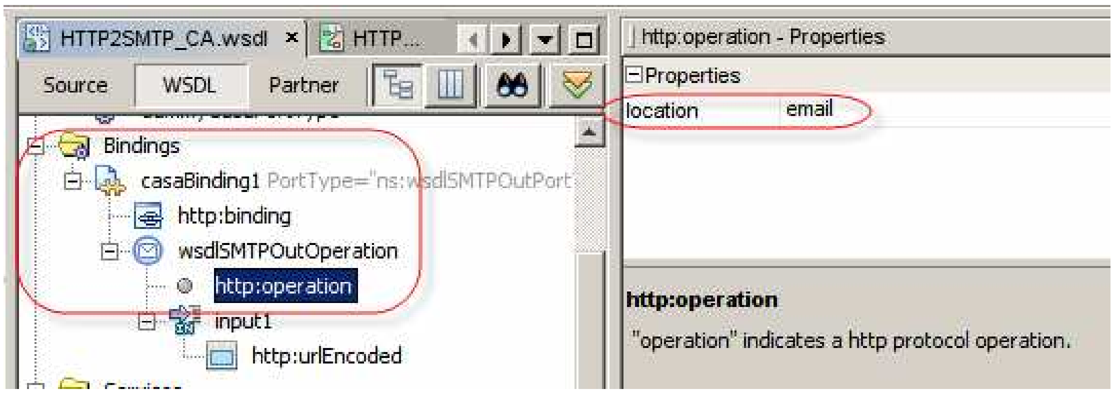
http://localhost:${HttpDefaultPort}/casaService1/casaPort1

Figure 3-15 Configuring http:address location attribute to provide service end-point address

Let's replace ${HttpDefaultPort} with an unused port number, say 58080.

Note that by default the HTTP BC will be configured for the HTTP GET request.

To enable the HTTP BC to work out which operation to invoke when a request arrives we must also provide a value to the location attribute of the http:operation node of the Bindings tree. Figure 3-16 calls out the relevant parts of the WSDL.



location email

http:operation

"operation" indicates a http protocol operation.

Figure 3-16 Adding http:operation location attribute value to the WSDL

Notice that the message structure, featuring the four message parts specified when configuring the SMTP WSDL are ‘inherited’ in this WSDL by virtue of us connecting the two BCs together. Take note of the names of these message parts – they will be required to properly format the HTTP GET request that triggers processing.

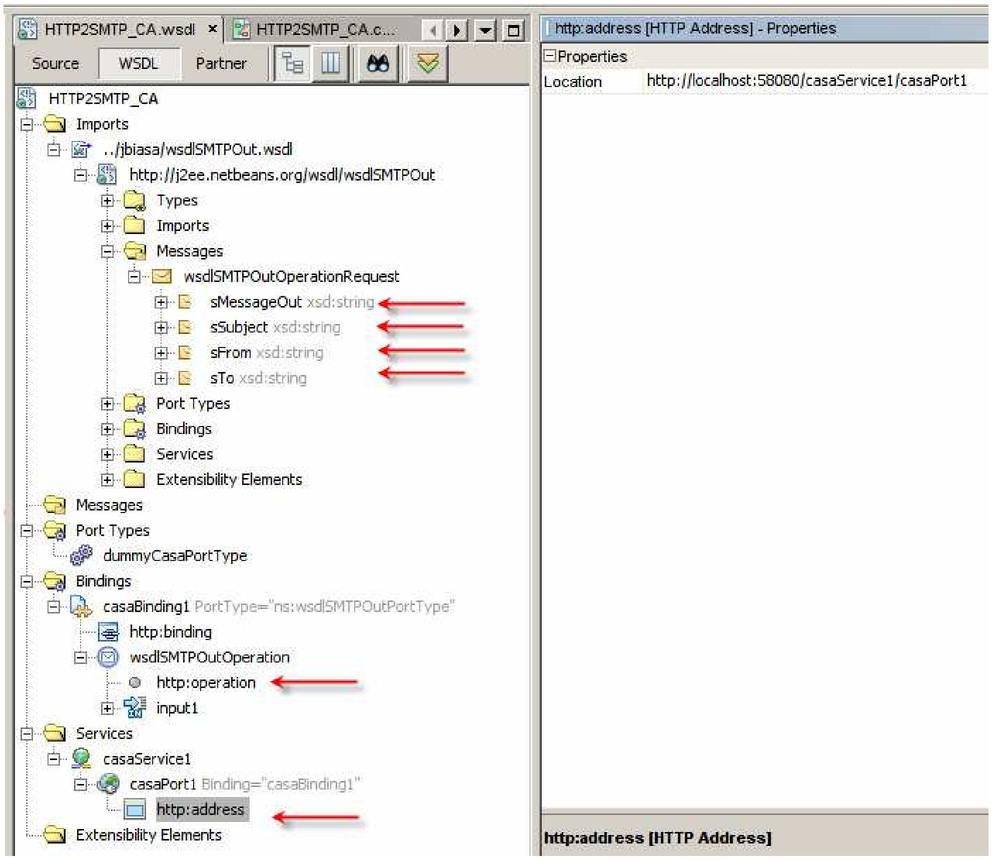
Build and deploy the project.

4 Exercise HTTP BC to SMTP BC Composite App

To exercise the solution we built we can use a Web Browser to submit a properly formatted HTTP GET request. The HTTP GET request is none other then the Universal Resource Locator (URL) we provide as the address of the resource.

For the HTTP BC, in which the input message is a multi-part message, the URL must be constructed using values provided in the HTTP BC WSDL.

The request URL consists of the content of the http:address node’s location attribute value, followed by a forward slash, http:operation node’s location attribute value, followed by a question mark, followed by a series of &-separated name-value pairs wherein the name is the name of the message part and the value is the value to be provided for that message part. Figure 4-1 highlights relevant parts of the WSDL.



Location http://localhost:58080/casaService1/casaPort1

```
HTTP2SMTP_CA
  Imports
    ../jbiasa/wsdlSMTPOut.wsdl
      http://j2ee.netbeans.org/wsdl/wsdlSMTPOut
        Types
        Imports
        Messages
          wsdlSMTPOutOperationRequest
            sMessageOut xsd:string
            sSubject xsd:string
            sFrom xsd:string
            sTo xsd:string
        Port Types
        Bindings
        Services
        Extensibility Elements
  Messages
  Port Types
    dummyCasaPortType
  Bindings
    casaBinding1 PortType="ns:wsdlSMTPOutPortType"
      http:binding
      wsdlSMTPOutOperation
        http:operation
        input1
  Services
    casaService1
      casaPort1 Binding="casaBinding1"
        http:address
  Extensibility Elements
```

http:address [HTTP Address]

Figure 4-1 Sources of data for the HTTP URL

Given the http:address location of "http://localhost:58080/casaService1/casaPort1" and http:operation location of "email", and message parts names "sMessageOut", "sSubject", "sFrom" and "sTo", the URL will be:

http://localhost:58080/casaService1/casaPort1/email?sMessageOut=mymessage&sSubject=mysubject&sFrom=user@email.server.com&sTo= user@email.server.com

Given the 1 to 1 correspondence between message parts of the HTTP BC and the SMTP BC the value of the sMessageOut element will be used as the SMTP message body, the value of the sSubject element will be used as the mail subject, the value of the sFrom element will be used as the sender's address and the value of the sTo element will be used as the recipient's address.

Certain special characters, to be included in the URL, must be encoded. RFC 3986, "Uniform Resource Identifier (URI): Generic Syntax", spells this out. See Percent-encoding Wikipedia page, http://en.wikipedia.org/wiki/Percent-encoding, for more accessible definition and a table of encoding sequences. This is important if you intend to include spaces and special characters as values of sMessage and sSubject elements.

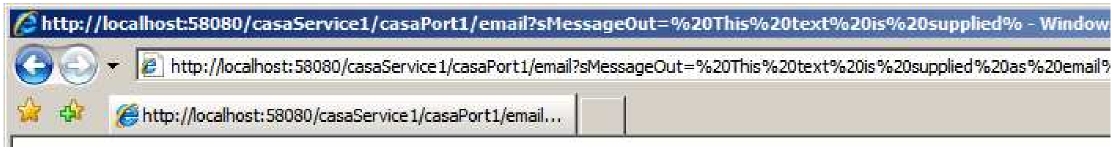
Let's provide the following URL to the web browser:

http://localhost:58080/casaService1/casaPort1/email?sMessageOut=
This%20text%20is%20supplied%20as%20email%20message%20body%20by%20providing%20it,%20encoded,%20as%20the%20value%20of%20the%20sMessageOut%2

0element.%0D%0DYours%20truly%0D%0DMichael&sSubject=Test%20Message&sFrom=KComilla@aus.sun.com&sTo=mczapski@aus.sun.com

Here %20 encodes spaces and %0D encodes carriage returns.

Figure 4-2 shows the Internet Explorer with the URL in the Address field.



http://localhost:58080/casaService1/casaPort1/email?sMessageOut=%20This%20text%20is%20supplied% - Window

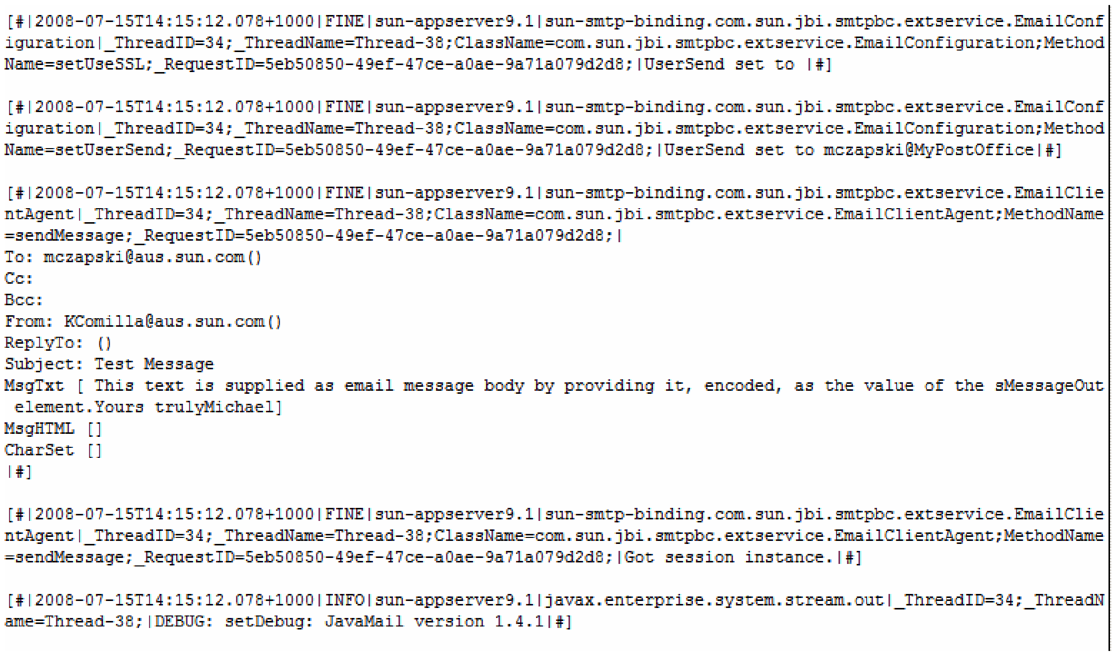
http://localhost:58080/casaService1/casaPort1/email?sMessageOut=%20This%20text%20is%20supplied%20as%20email%

http://localhost:58080/casaService1/casaPort1/email...

Figure 4-2 URL in Internet Explorer

The web browser will complete the request with a blank page, since the service implementation uses a One-way WSDL and does not return anything back to the browser.

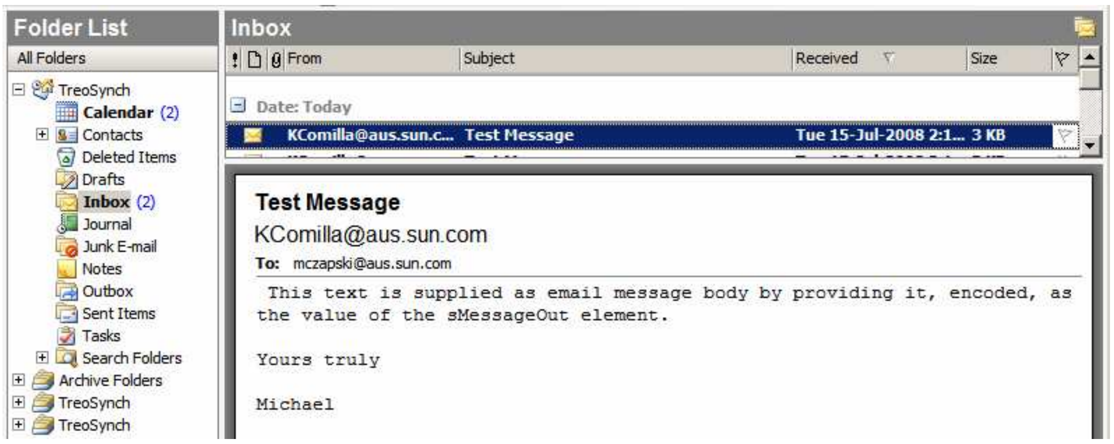
The server.log, when logging at the appropriate level is enabled for the HTTP BC and the SMTP BC, may show messages similar to these shown in Figure 4-3.



```
[#|2008-07-15T14:15:12.078+1000|FINE|sun-appserver9.1|sun-smtp-binding.com.sun.jbi.smtpbc.extservice.EmailConf
iguration|_ThreadID=34;_ThreadName=Thread-38;ClassName=com.sun.jbi.smtpbc.extservice.EmailConfiguration;Method
Name=setUseSSL;_RequestID=5eb50850-49ef-47ce-a0ae-9a71a079d2d8;|UserSend set to |#]

[#|2008-07-15T14:15:12.078+1000|FINE|sun-appserver9.1|sun-smtp-binding.com.sun.jbi.smtpbc.extservice.EmailConf
iguration|_ThreadID=34;_ThreadName=Thread-38;ClassName=com.sun.jbi.smtpbc.extservice.EmailConfiguration;Method
Name=setUserSend;_RequestID=5eb50850-49ef-47ce-a0ae-9a71a079d2d8;|UserSend set to mczapski@MyPostOffice|#]

[#|2008-07-15T14:15:12.078+1000|FINE|sun-appserver9.1|sun-smtp-binding.com.sun.jbi.smtpbc.extservice.EmailClie
ntAgent|_ThreadID=34;_ThreadName=Thread-38;ClassName=com.sun.jbi.smtpbc.extservice.EmailClientAgent;MethodName
=sendMessage;_RequestID=5eb50850-49ef-47ce-a0ae-9a71a079d2d8;|
To: mczapski@aus.sun.com()
Cc:
Bcc:
From: KComilla@aus.sun.com()
ReplyTo: ()
Subject: Test Message
MsgTxt [ This text is supplied as email message body by providing it, encoded, as the value of the sMessageOut
 element.Yours trulyMichael]
MsgHTML []
CharSet []
|#]

[#|2008-07-15T14:15:12.078+1000|FINE|sun-appserver9.1|sun-smtp-binding.com.sun.jbi.smtpbc.extservice.EmailClie
ntAgent|_ThreadID=34;_ThreadName=Thread-38;ClassName=com.sun.jbi.smtpbc.extservice.EmailClientAgent;MethodName
=sendMessage;_RequestID=5eb50850-49ef-47ce-a0ae-9a71a079d2d8;|Got session instance.|#]

[#|2008-07-15T14:15:12.078+1000|INFO|sun-appserver9.1|javax.enterprise.system.stream.out|_ThreadID=34;_ThreadN
ame=Thread-38;|DEBUG: setDebug: JavaMail version 1.4.1|#]
```

Figure 4-3 Fragment of verbose log of request processing showing SMTP interaction

In an email client the message will look similar to what is shown in Figure 4-4.



KComilla@aus.sun.c... Test Message Tue 15-Jul-2008 2:1... 3 KB

Test Message
KComilla@aus.sun.com
To: mczapski@aus.sun.com

```
 This text is supplied as email message body by providing it, encoded, as
the value of the sMessageOut element.

Yours truly

Michael
```

Figure 4-4 Message in Microsoft Outlook

5 Summary

This document walked the reader, step-by-step, through the process of creating and exercising a Java CAPS 6/JBI (or OpenESB) HTTP BC to SMTP BC solution that did not use any logic component for transformation. Routing of messages was accomplished using the Normalized Message Router. The solution used only JBI components.