



OPEN ESB 2.3 TUTORIAL

HTTP BC to EMAIL BC, No BPEL

ABOUT LOGICOY

- LogiCoy incorporated in February 2009.
- Headquarters in Los Angeles, California with 24/7 offices in US, U.K., and India, is a Global Information Technology and Services company providing products and services in integration and middleware across various industries including financial, healthcare, telecommunications, manufacturing, and government.
- We are the original architects, developers and engineering managers and directors for Seebeyond eGate, ICAN, Sun Java CAPS, Mural MDM, Open/GlassFish ESB products from inception to end and know the code inside and out.
- We are the primary committers of the OpenESB codebase in the OpenESB Open source repository and actively working on the enhancements and new versions of OpenESB.
- We have augmented our product development with very seasoned SOA/EAI Technical Leads and Architects for integration solutions with field implementation experience using Sun Java CAPS, GlassFish ESB, eGate and other integration tools in healthcare integration projects.

Contact us at: info@logicoy.com



ABOUT THE AUTHOR

Michael Czapski was the solution architect in Sun Microsystems. He is currently working with Oracle. He has contributed enormous amount of time/resources for Open ESB in the form of architecting, designing, testing and documenting. He also has written a book on JavaCAPS, an older version of OpenESB. He is considered as a SOA guru.



TABLE OF CONTENTS

Introduction.....	3
Create Project Group 05HTTP2SMTPNoBPEL_PG.....	4
Create HTTP BC to EMAIL BC Composite App.....	4
Exercise HTTP BC to EMAIL BC Composite App.....	17
Summary	20



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 EMAIL BC uses it to send electronic mail with no BPEL logic to tie the

two together”. I thought that the answer was “Yes” but I felt I had to verify it. Vishnuvardhan Piskalaramesh from Sun, who is looking after the EMAIL 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 an 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.

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.

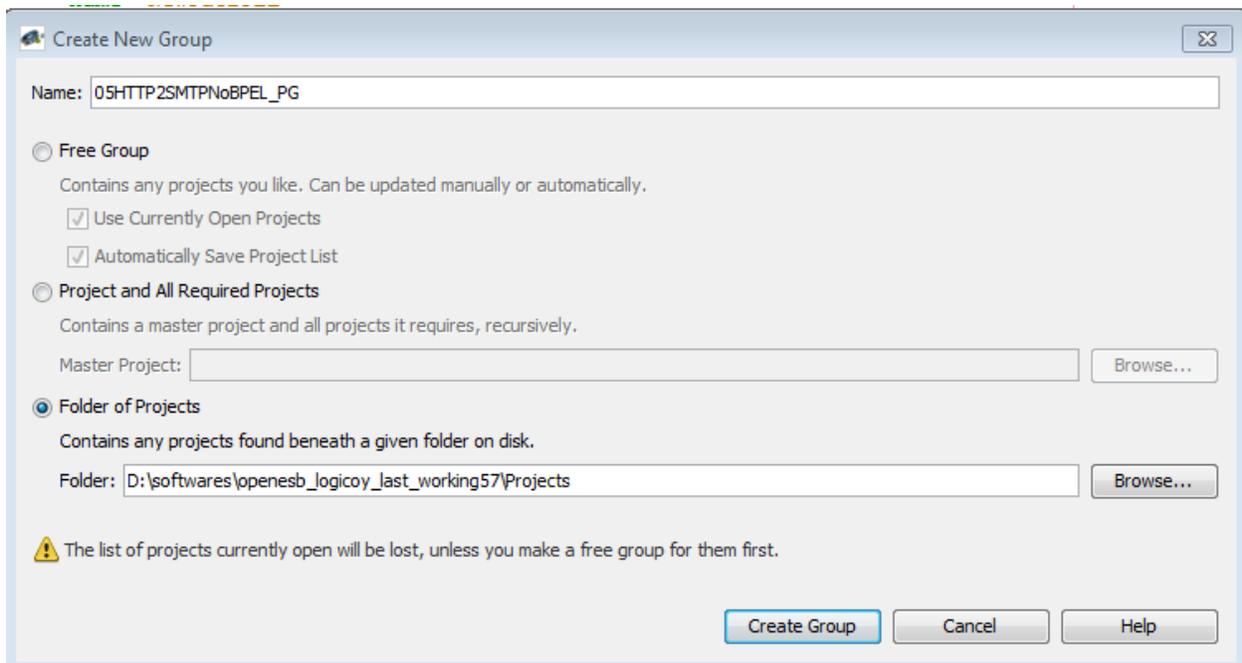


FIGURE 0-1 CREATING A NEW PROJECT GROUP

Create HTTP BC to EMAIL BC Composite App

It is sometimes assumed that BPEL 3.0 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 3.0 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 EMAIL BC as the destination.

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

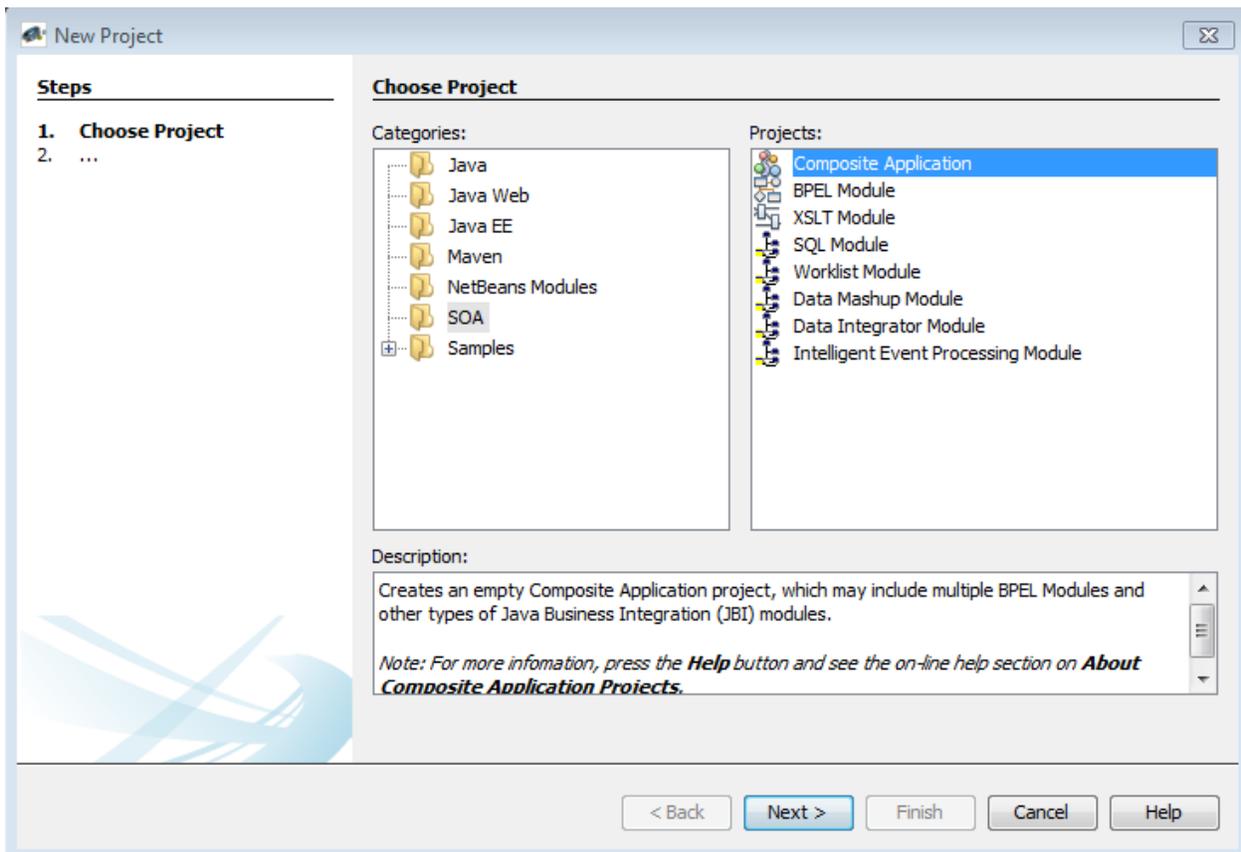


FIGURE 0-1 CHOOSE TO CREATE A WSDL DOCUMENT

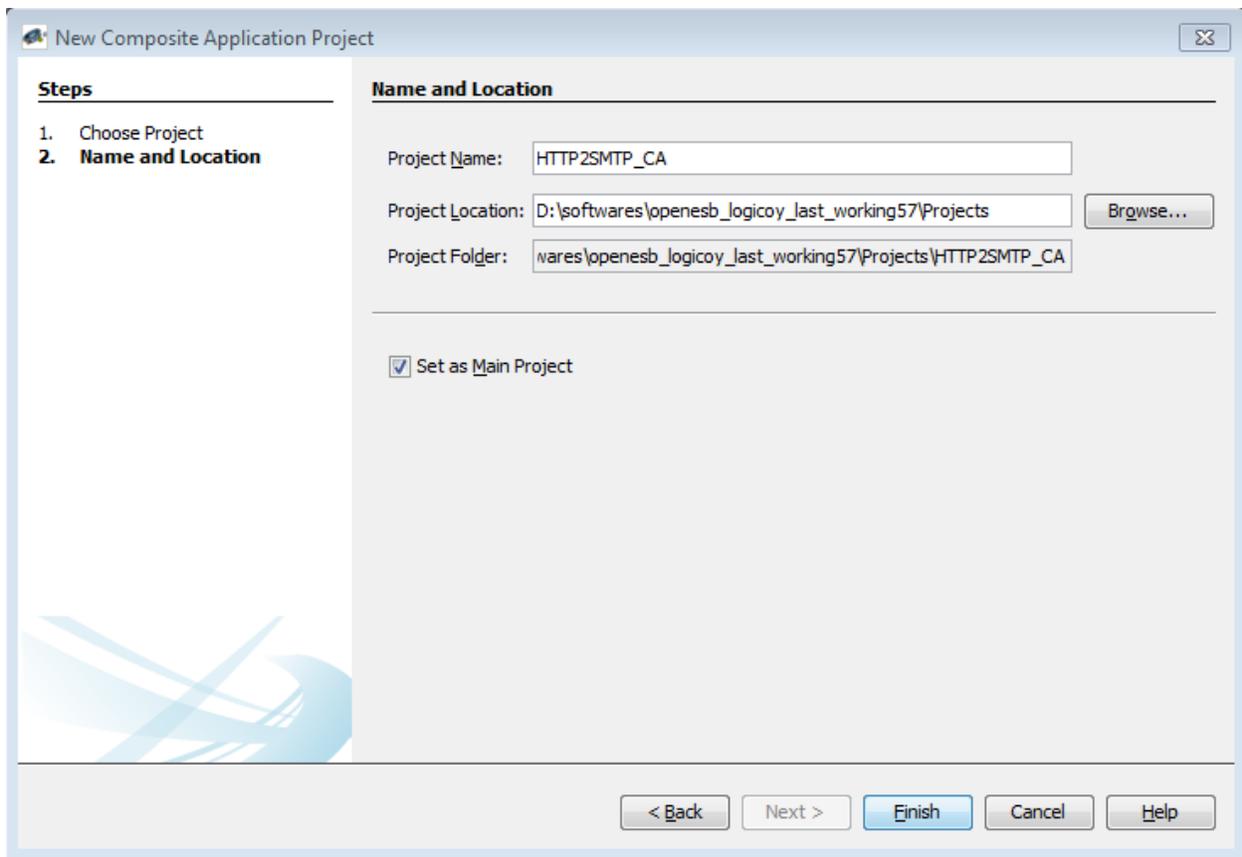


FIGURE 0-2 CHOOSE TO CREATE A WSDL DOCUMENT

Let's create a new WSDL, `wsdISMTPSendEmail`, for the EMAIL 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.

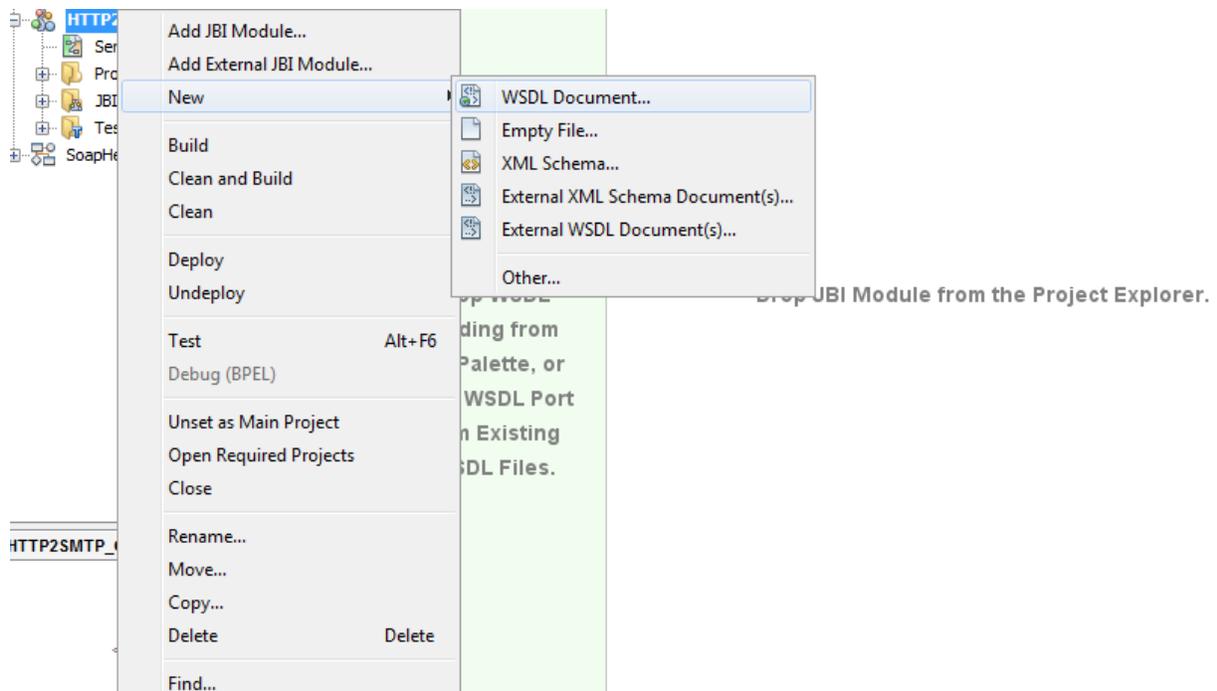


FIGURE 0-1 CHOOSE TO CREATE A WSDL DOCUMENT

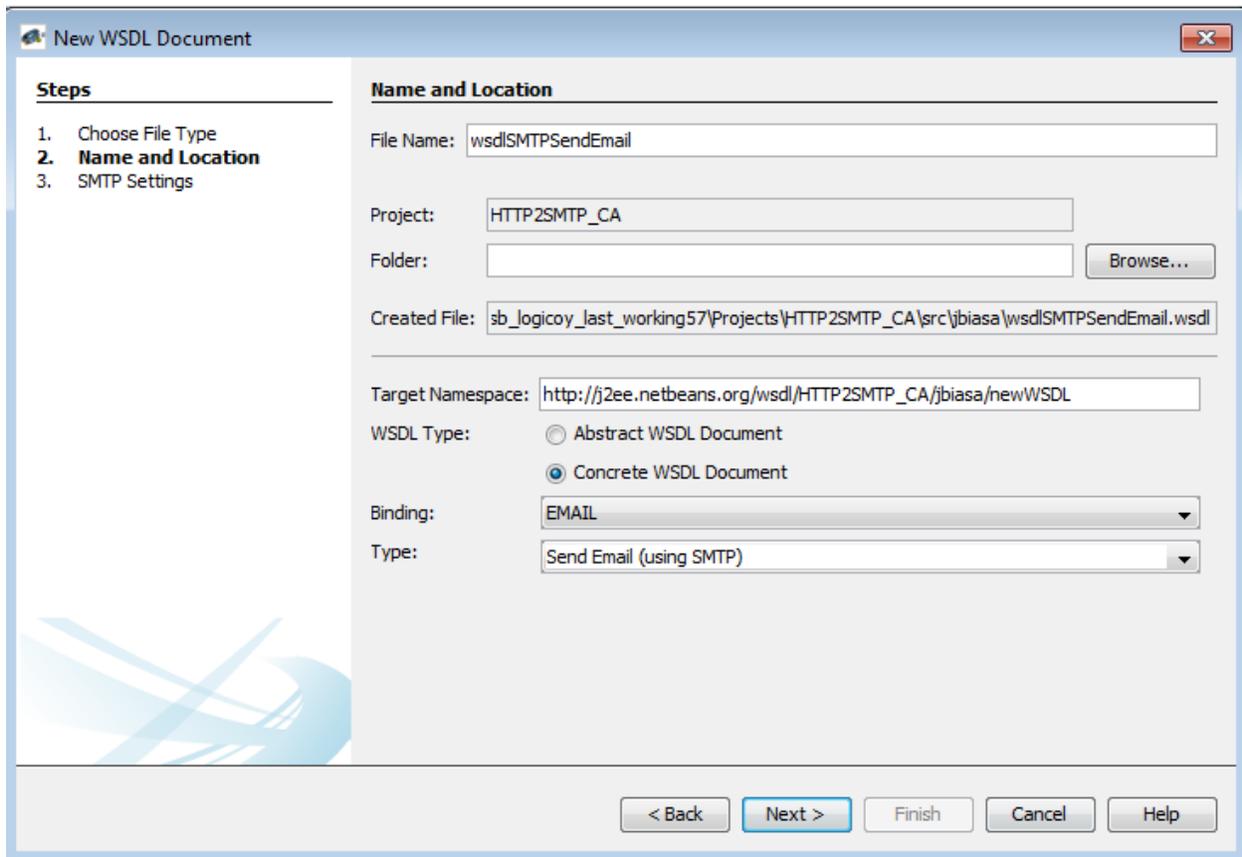


FIGURE 0-2 NAME THE WSDL DOCUMENT

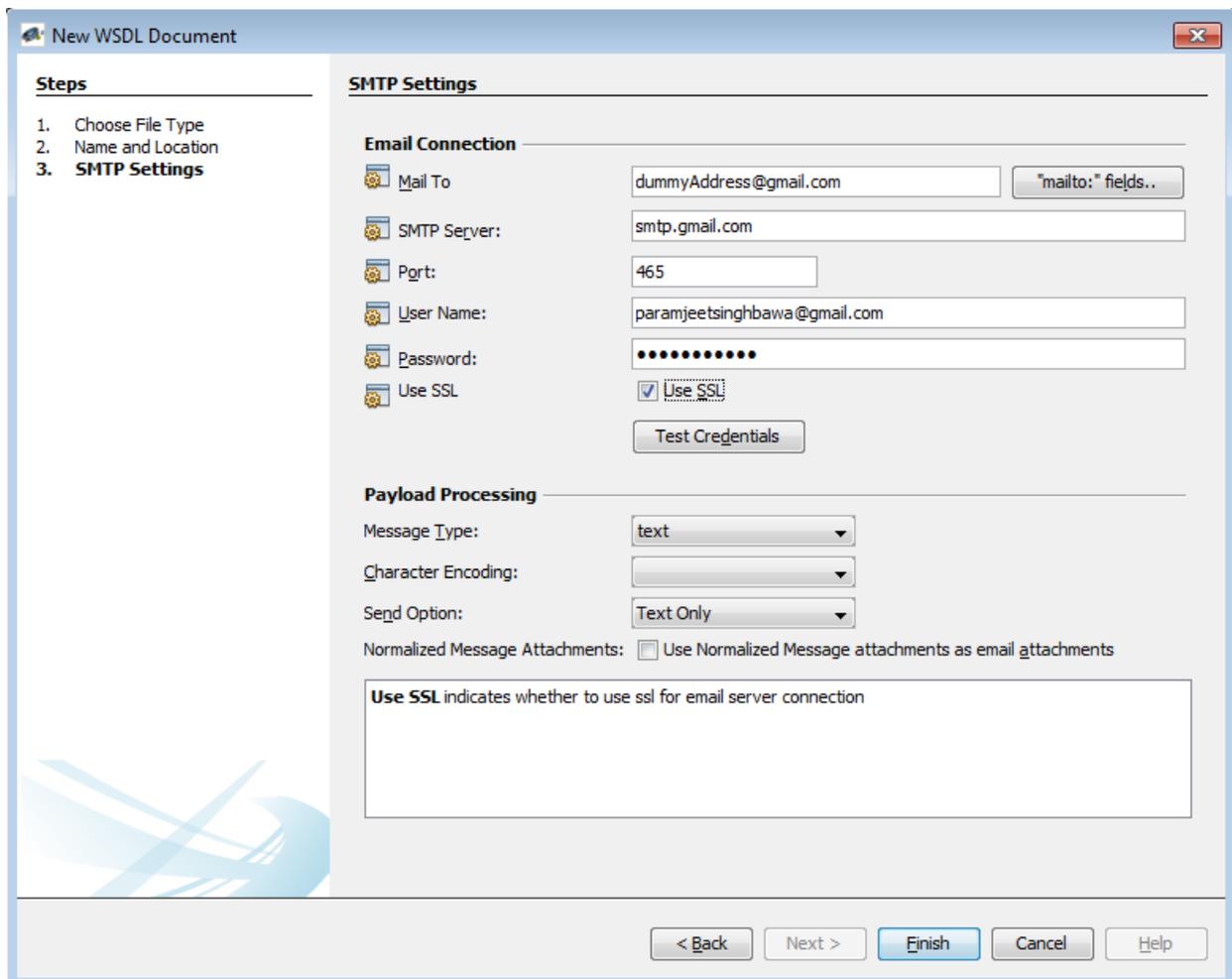
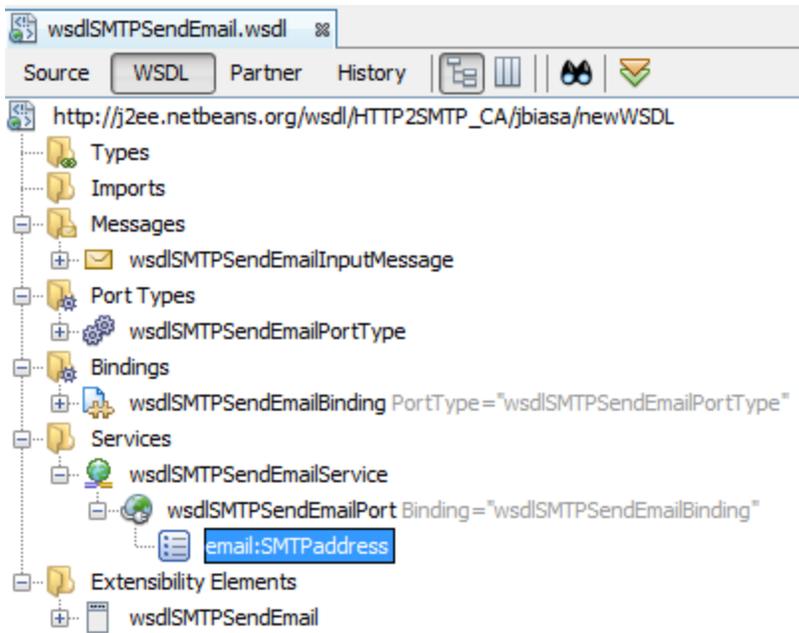


FIGURE 0-3 CONFIGURING ADDRESS PROPERTIES FOR THE SMTP SERVER

configure the smtp address node under the Services tree to point at the appropriate SMTP Server and provide correct credentials. Figure 3-7 illustrates this. The user will be from and To addresses will come from the message so the Location property seems to be superfluous. I configured it anyway.



email:IMAPAddress - Properties

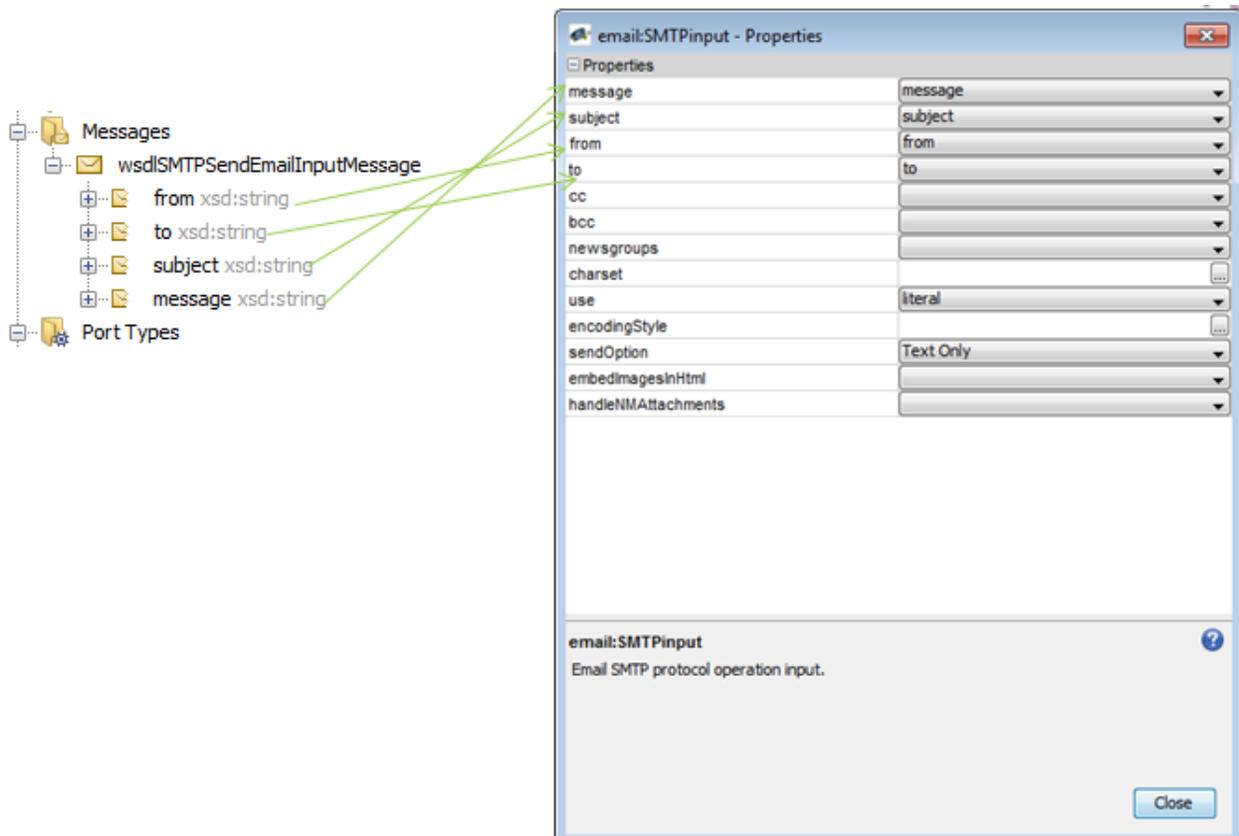
Properties

emailServer	imap.gmail.com	...
port	993	...
userName	uremailAddress@gmail.com	...
password	mfkjdfkjf	...
useSSL	false	▼
mailFolder		...
maxMessageCount		...
messageAckMode	automatic	▼
messageAckOperation	markAsRead	▼
pollingInterval		...

email:IMAPAddress ?

Email IMAP protocol service address.

Close



The email:SMTPBinding 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 EMAIL BC has four parts. Message will contain message body, subject will contain mail subject text, from will contain sender's address and to 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 associating 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.

FIGURE 0-4 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 EMAIL BC WSDL, Figure 3-10.

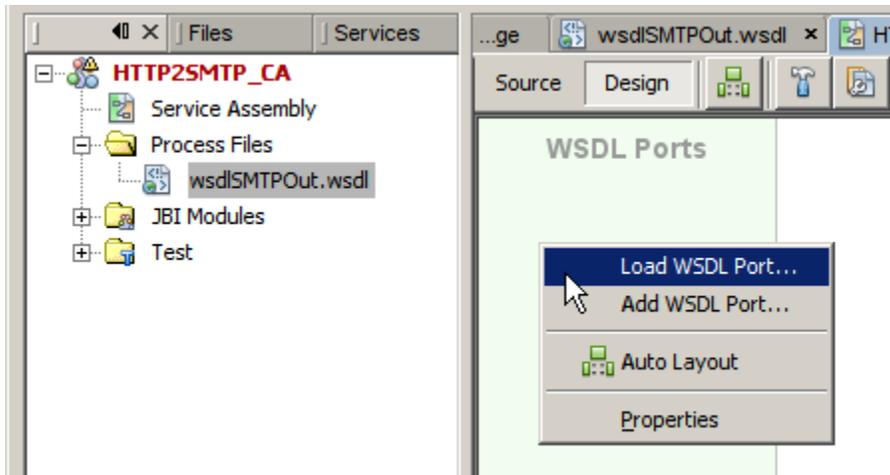


FIGURE 0-5 LOADING SMTP WSDL INTO THE SERVICE ASSEMBLY

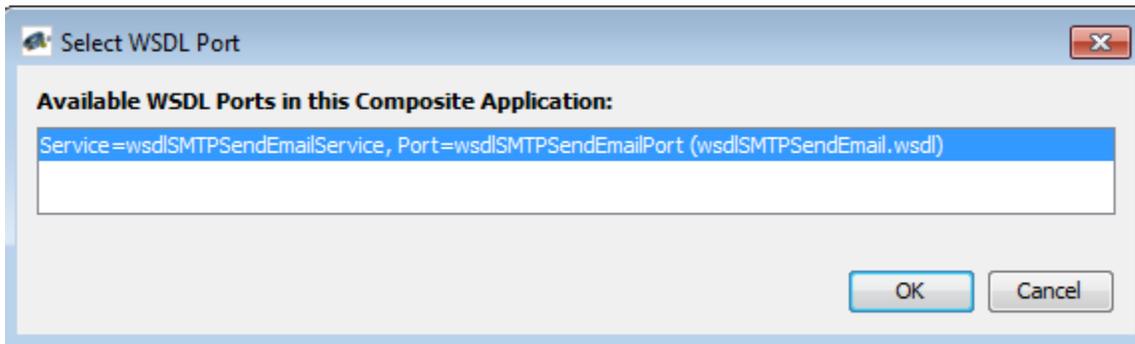


FIGURE 0-6 SELECTING WSDL PORT

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



FIGURE 0-7 EMAIL BC ON THE SERVICE ASSEMBLY CANVAS

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

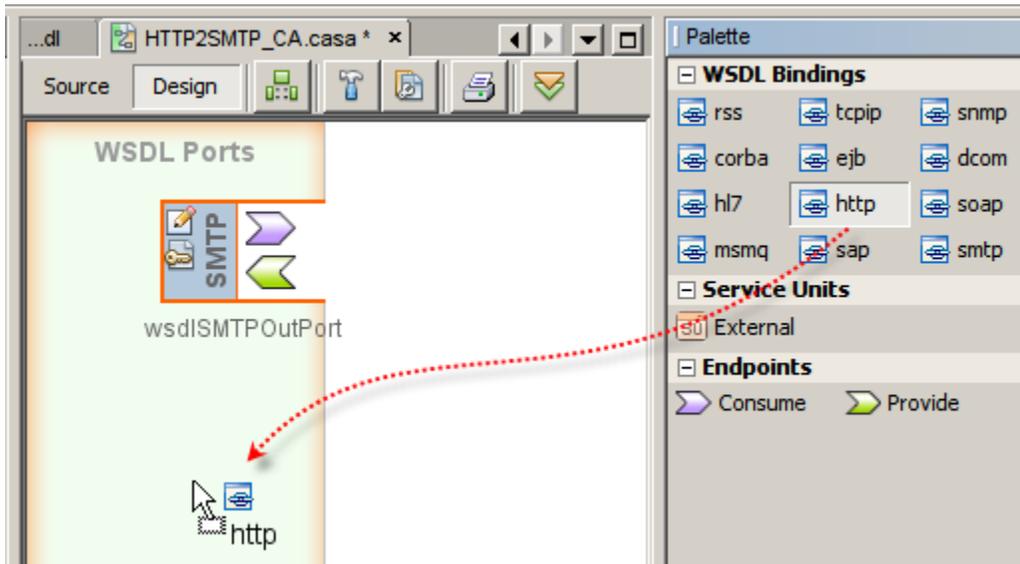


FIGURE 0-8 ADDING HTTP BC TO THE CANVAS

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

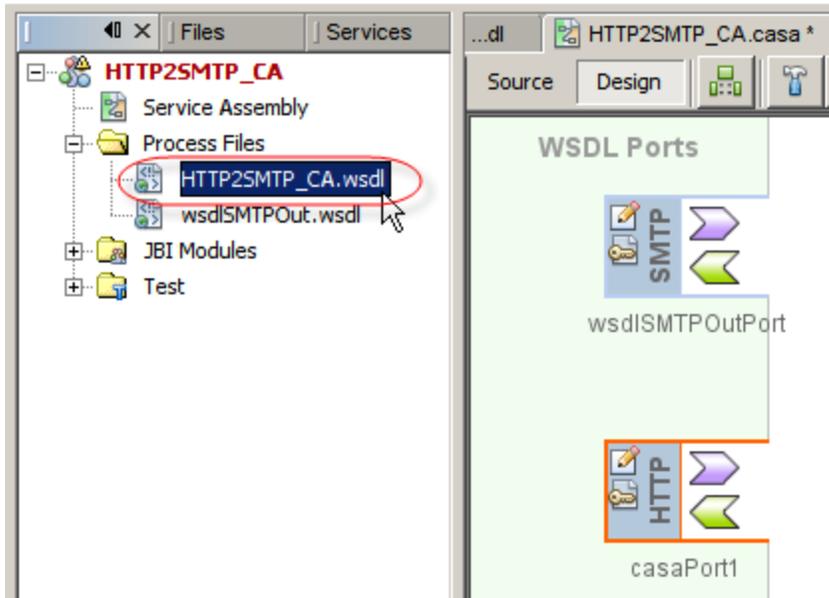


FIGURE 0-9 NEW WSDL IN THE PROCESS FILES FOLDER

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

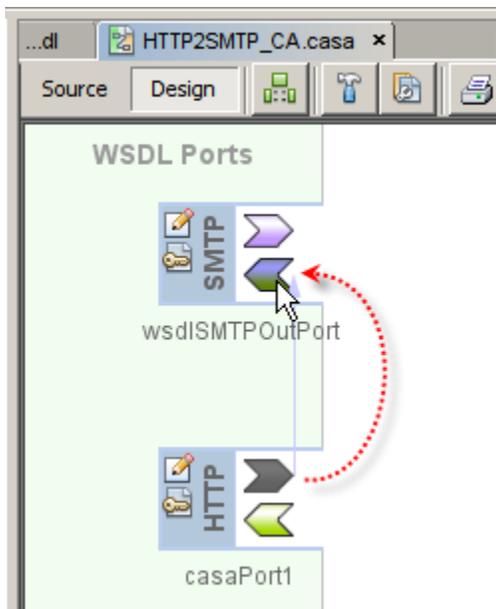


FIGURE 0-10 CONNECTING HTTP BC AND EMAIL BC

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

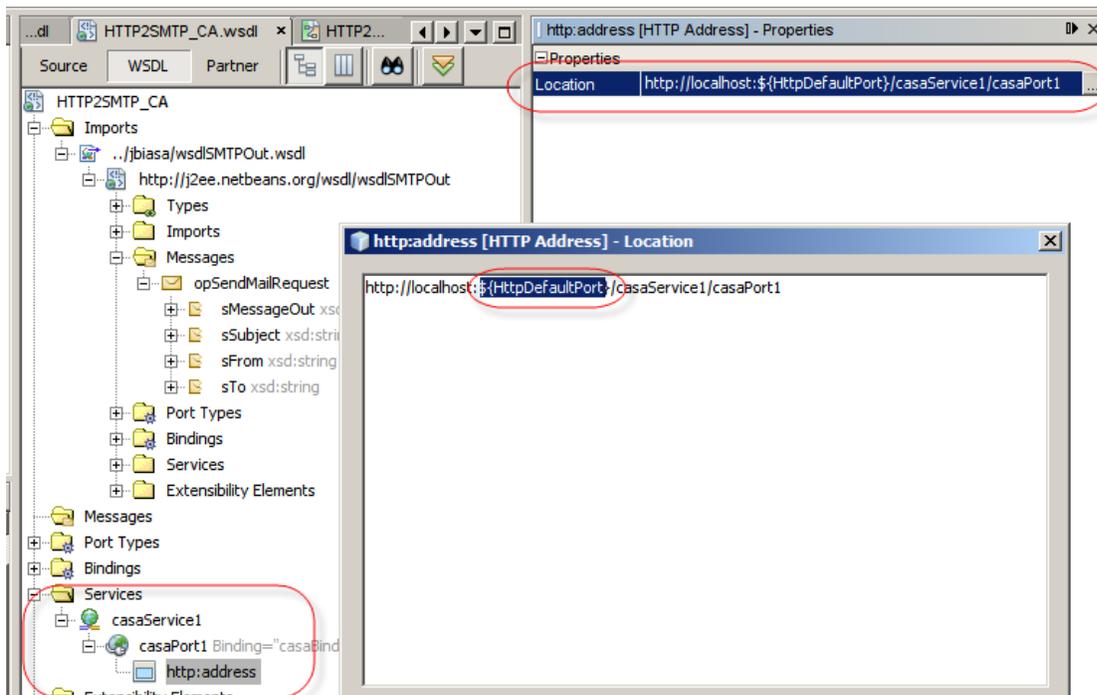


FIGURE 0-11 CONFIGURING HTTP:ADDRESS LOCATION ATTRIBUTE TO PROVIDE SERVICE END-POINT ADDRESS

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.

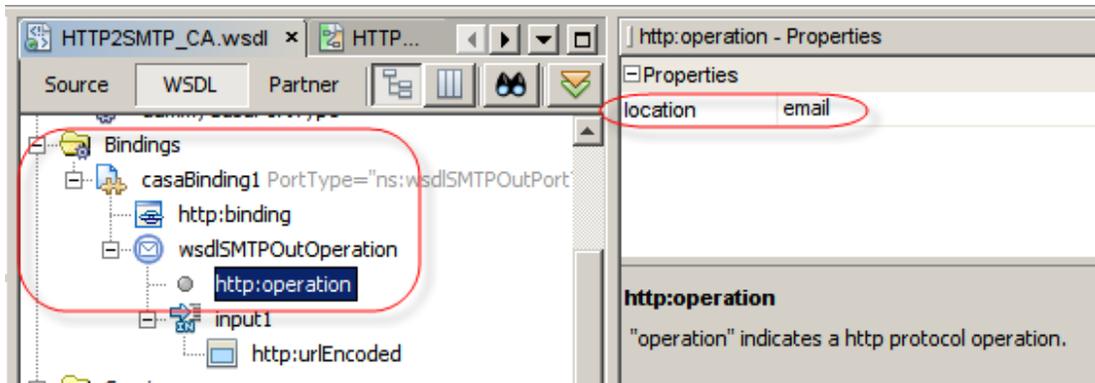


FIGURE 0-12 ADDING HTTP:OPERATION LOCATION ATTRIBUTE VALUE TO THE WSDL

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.

Exercise HTTP BC to EMAIL 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 than 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.

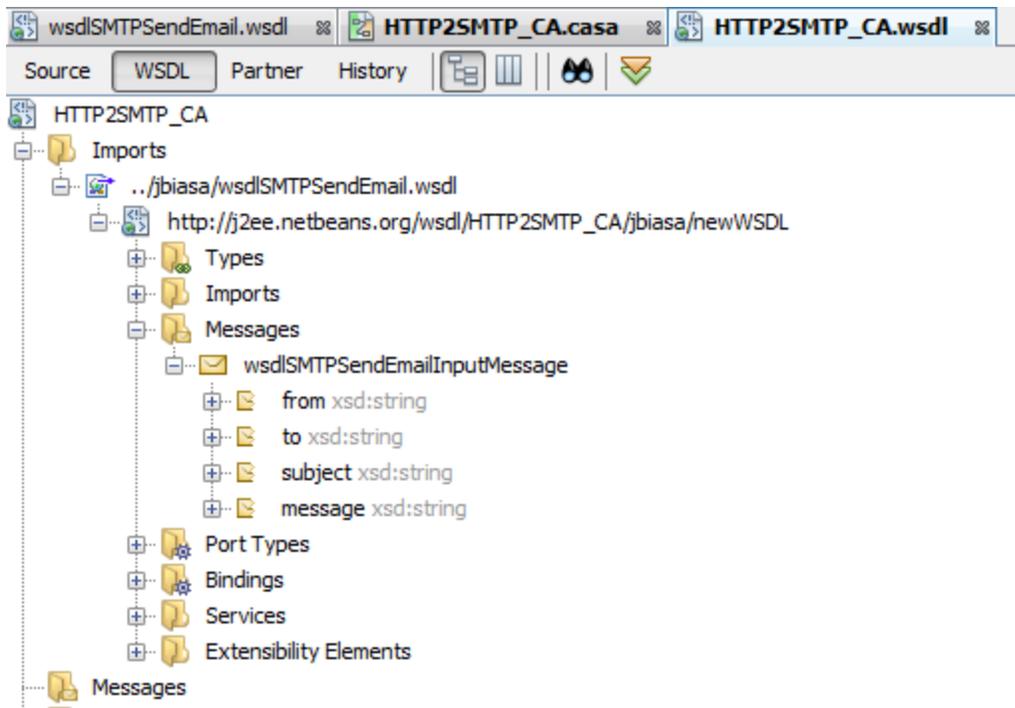


FIGURE 0-1 SOURCES OF DATA FOR THE HTTP URL

Given the http:address location of “http://localhost: 58080/HTTP2SMTP_CAService1/casaPort1” and http:operation location of “email”, and message parts names “message”, “subject”, “from” and “to”, the URL will be:

```
http://localhost:
58080/HTTP2SMTP_CAService1/casaPort1/email?message=mymessage&subject=mysubject&from=user
@email.server.com&to= user@email.server.com
```

Given the 1 to 1 correspondence between message parts of the HTTP BC and the EMAIL BC the value of the message element will be used as the SMTP message body, the value of the subject element will be used as the mail subject, the value of the from element will be used as the sender’s address and the value of the to 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 subject elements.

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

http://localhost:58080/casaService1/casaPort1/email?message=
This%20text%20is%20supplied%20as%20email%20message%20body%20by%20providing%20it,%20
encoded,%20as%20the%20value%20of%20the%20message%20element.%0D%0DYours%20truly%0D
%0DMichael&subject=Test%20Message&from=KComilla@aus.sun.com&to=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.

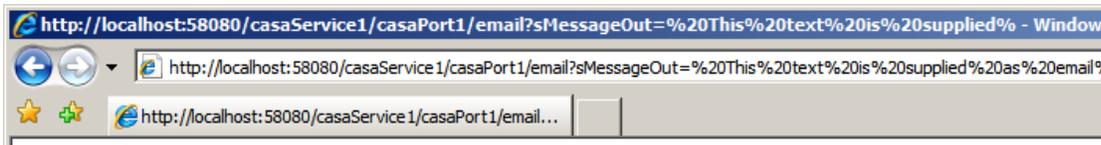


FIGURE 0-2 URL IN INTERNET EXPLORER

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 EMAIL 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.EmailConfiguration|_ThreadID=34;_ThreadName=Thread-38;ClassName=com.sun.jbi.smtpbc.extservice.EmailConfiguration;MethodName=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.EmailConfiguration|_ThreadID=34;_ThreadName=Thread-38;ClassName=com.sun.jbi.smtpbc.extservice.EmailConfiguration;MethodName=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.EmailClientAgent|_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.EmailClientAgent|_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;_ThreadName=Thread-38;|DEBUG: setDebug: JavaMail version 1.4.1|#]
```

FIGURE 0-3 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.

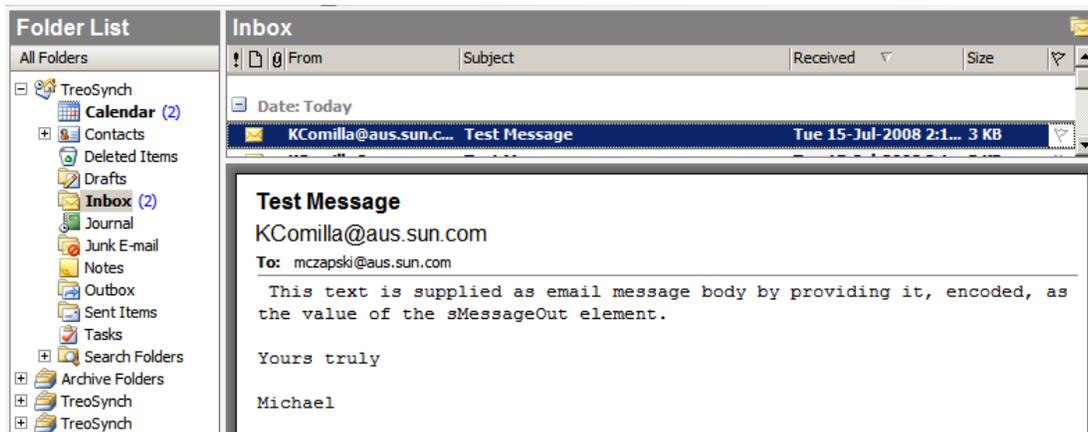


FIGURE 0-4 MESSAGE IN MICROSOFT OUTLOOK

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 EMAIL 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.

