# Creating an OpenESB Application

## Summary

This document details how to create an OpenESB application that acceps SOAP messages, logs the message, and forwards it to its final destination. Two examples applications exist already; ExecuteDERGroupDispatchesApp, and GetDERGroupStatusesApp. This ‘how to’ document will extensively refer to GetDERGroupStatyusesApp for examples and instruction.

Due to the nature of the OpenESB architecture (the language of applications is an xml file, not a language at all, but rather declarations of methods to accomplish tasks) a first time coder of an application will experience many issues. This document specifically addresses its instruction to the cloning of the existing applications, which are identical (different wsdl’s, but the exact same process flow), and may be insufficient if a developer wishes to implement new features for the applications.

# The Environment

The document (stored in git) for installing the tools needed to create and run OpenESB is *NREL Mediator Installation Instructions.docx*. This document details how to install the OpenESB binaries on a server. Note, a version of netbeans is part of this release, and the example images in this how-to document are screen shots taken from that application.

## Netbeans Editor

* OpenEsb : /usr/local/OpenESB-SE-3.0.5
* Netbeans : /usr/local/OpenESB-SE-3.0.5/OE-Studio/netbeans/bin/openesb

## Startup Script

The following is a startup script that can be used to start netbeans and the OpenESB server. Note: the server can also be started from within netbeans from the server tab.

#!/bin/ksh

/usr/local/OpenESB-SE-3.0.5/OE-Studio/netbeans/bin/openesb &

/usr/local/OpenESB-SE-3.0.5/OE-Instance/bin/openesb.sh &

## Source Location

The source directory for the application must be set to /usr/local/git/nrel (atleast for the NREL application). This is due to the fact that OpenESB hardcodes the full directory paths of its WSDL’s when building an application, and reading the CIM WSDLs from a file path. If you load a new WSDL from a URL, then the hardcoded path is not necessary. However, be aware that the OpenESB server will fail to initialize the application if the WSDL referred to is not available for reading from its URL (http://…).

NREL Project Repository : <https://github.com/sixthcolumn/nrel>

# Tutorial

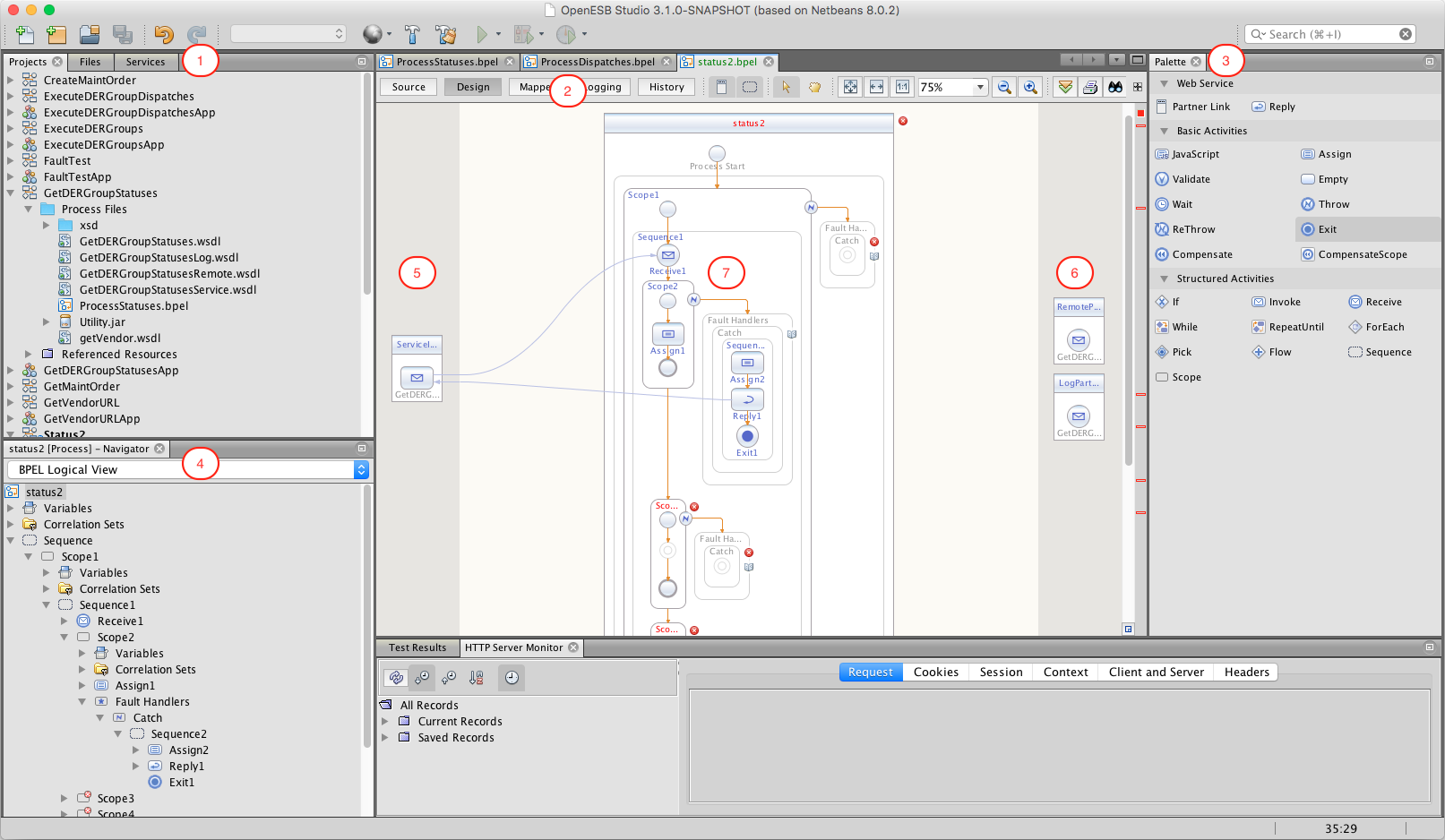
In this tutorial we will re-create the GetDERGroupStatuses application from scratch. Good luck!

## Start Netbeans

/usr/local/OpenESB-SE-3.0.5/OE-Studio/netbeans/bin/openesb &

## Netbeans Views

Following is an illustration of how I set up my Netbeans interface. If, after starting netbeans, any of the views I mention are missing, select Window menu, and click on any missing views to display them.



1. Project View – Displays the projects you have loaded into netbeans, along with project files. Note in this example the following files :
   1. Project wsdl files
   2. The bpel we are going to create – ProcessStatuses.bpel
   3. Xsd subdirectory – contains xsd files used by the wsdls
   4. Jar file – utility.jar contains java methods utilized by this bpel
   5. Referenced Resources – don’t worry about these
2. Design Views
   1. Source – shows the raw source code
   2. Design – The default view, shown in this illustration
   3. Mapper – Used for assigning attributes, more on this later
   4. Logging – A view for creating log entries for various steps
   5. History – I don’t use this
3. Palette – We drag activities from the palette into the ‘Design View’, adding the activities to our flow. Most used activities :

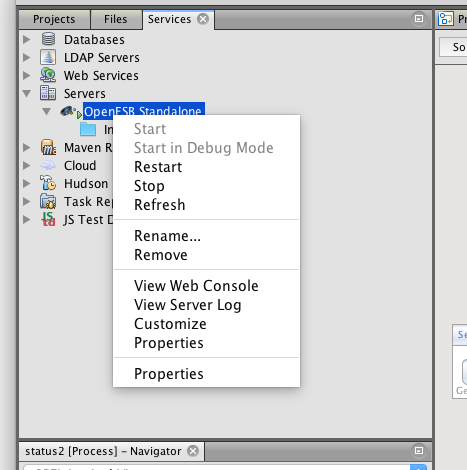
* Receive – get data from Service Input (number 5)
* Reply – reply back to Service Input (number 5)
* Invoke – Call an external service (number 6)
* Assign – Copy variables from one process to another
* Exit – End the process flow
* Scope – Process Scope, think of it as a java try/catch block

1. Logical View – We only use this once, to create the RemoteURL variable. It shows the view of the project parts, from initial Sequence, and including inner scopes, and all the process flow steps we write.
2. Service input – This is the Entry point for our BPEL process. In affect, this is what gets the SOAP message that we are going to process. More on this later
3. Service Output – External processes that we call. In our case we will call the final endpoint SOAP server, and the logging server.

Don’t worry if this all doesn’t make sense quite yet. We’ll go over each piece as necessary.

## Services View

You can contol the OpenESB server from within netbeans, or you can start it from the command line. I’m just taking a moment to show you how to do it from within netbeans here.



You can select ‘Services’ from the tabs above the project view (number 1) from the previous illustration. Once you do that you can do the following :

* Expand Servers – this will show you your OpenESB Standalone. If you click this and your server does not show, you can ‘add’ it as your server.
* Right click on ‘OpenESB Standalone’ to display contest menu
  + Start – the OpenESB server
  + View Web Console – Displays web browser to administer the server

A few notes / problems with this environment (for me on a mac system)

* The context menu items Restart and Stop do not work
* Occasionally the entire environment will lock up and I have to go to a shell and kill netbeans and OpenESB server
* The OpenESB server doesn’t always shut down when it should and when you try to restart you will get a message stating port in use. Use shell and kill OpenESB, etc… when this occurs
* I save my project often. YMMV.

## Create BPEL Module

* File / New Project
* SOA / BPEL Module
* Next
* Project Name : Status2
* Project Location : /usr/local/git/nre
* Project Folder : (do nothing, it will be set based on project name)
* Finish

Congratulations, you have created a new BPEL module.

## Load SOAP WSDL

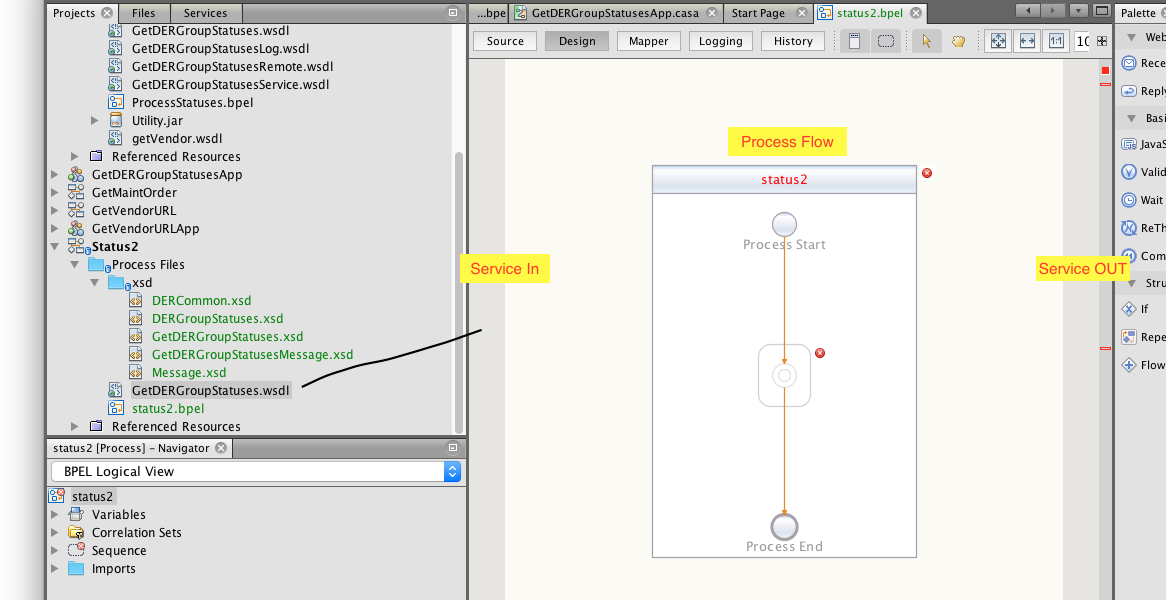
In order to receive, process, and re-route SOAP messages, the WSDL for GetDERGroupStatuses must be loaded into the project as a process file.

* Right click on ‘Status2’ project in left side tab
* Click on ‘New external Wsdl’
* From local file system
* Browse to location of your GetDERGroupStatuses.wsdl
* The file used by this application is currently under git.  
  /usr/local/git/nrel/61968/wsdl/GetDERGroupStatuses.wsdl
* Select file and click finish.

The SOAP file that will allow you to process DERGroupStatuses is now located in the project : Status2/Process files/GetDERGroupStatuses.wsdl, along with other xsd files utilized by the WSDL, under xsd.

## A Short Diversion on Drag and Drop

Netbeans for BPEL module utilizes drag and drop for creating flows. A basic explanation of this functionality is given after the image which follows.



### Editor Layout

* Left side : project browser
* Center : project flow
  + Left side (Dark gray area) drag services into this section (ie: GetDERGroupStatuses.wsdl). This section represents service input. When you drag into this section, a circle will appear into which you can drop your wsdl.
  + Center : Process flow. The majority of the work is done in this section, creating flows.
  + Right side (Dark gray area) : Service outputs are dropped here. For this project we will drop a second GetDERGroupStatuses.wsdl in here, which will represent the service that the app will pass the SOAP message to after logging it.
  + Top section (Source, Design, Mapper, Logging, History). These are the various design views. Their use will be explained as needed throughout.
* Right side (cut off) : processes, such as receive, reply, assign, invoke…

## Service Input

* Grab the GetDERGroupStatuses.wsdl on the left side (the one you just imported), and drag it to the left side of the main design window.
* When you do, a small circle will appear. Drop it in there.
* When you drop it a ‘Service Partner’ dialog will appear. You can just choose the defaults, but I like to modify my links thusly :
  + Name : ServiceInPartnerLink
  + Create in File : GetDERGroupStatusesService
* Click ‘OK’
* Note that a new wsdl has been created in the project view : GetDERGroupStatusesService.wsdl. Don’t worry. Each partner link gets its own wsdl.
* Save your project… often.

Congratulations. You have just set up your BPEL module service input point. In essence, you have told BPEL that you will process SOAP messages that conform to GetDERGroupStatuses.wsdl.

## Service Output

This is an identical operation to the service input you just completed.

* Grab the GetDERGroupStatuses.wsdl
* Drag it to the right side of the design window (the dark gray thin vertical bar)
* A circle will appear
* Drop it in the circle.
* The Partner Link dialog appears. Make the following changes :
  + Name : RemotePartnerLink
  + Create File In : GetDERGroupStatusesRemote

Congratulations. You have just defined the ‘out’ service. Once a message has been processed, it will be forwarded to this service.

## Log Output

We log messages to a SOAP server, which stores information about the request in a database. This is a one-way transaction, in that we forward the SOAP messages to this log server, but ignore the messages it returns.

This is identical to the service output you just created. The only difference is naming the partner linkes.

* Grab GetDERGrtoupStatuses.wsdl
* Drag it to right dark gray bar, a new circle will appear
* Drop it into the circle
* Modify the Partner link attributes :
  + Name : LogPartnerLink
  + Create File in : GetDERGroupStatusesLog
  + Press ‘OK’
* Save your project

## Add Utuility jar file

BPEL supports calling of static java methods from a jar file. There are two methods this project uses for convenience :

* Com.sixthc.bpel.spring.getURLByVendor - Returns a URL (string) containing a URL retrieved from the database for the final destination of the SOAP message
* Com.sixthc.bpel.spring.log – logs a debug message to the database using the unique identifier associated with each SOAP message
* More on the inner workings of these later

### Build Jar File

* Csh> Cd /usr/local/git/nrel/BpelJava
* Csh> mvn package

### Install Jar File

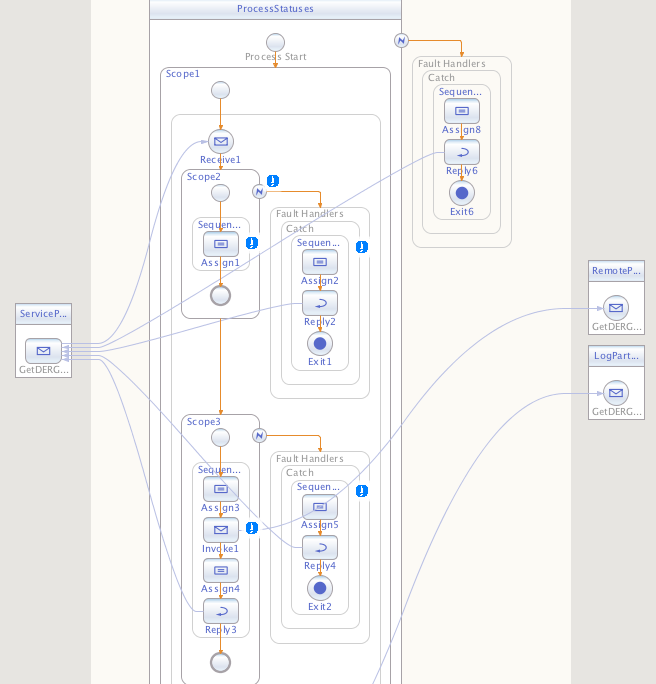
* By copying the jar file to Status2’s src directory, it becomes accessible to the BPEL module
* Csh> cp /usr/local/git/nrel/BpelJava/target/Utility.jar /usr/local/git/nrel/Status2/src
* When this is done, the Utility.jar will immediately show up under the Status2 explorer window (left side of netbeans) under Process Files

## Summary So Far

At this point you have set up the BPEL module to accept a SOAP message (left side), and (once you create the flow) send the messages to the remote server, and the log server. This was the easy part.

## Process Flow Scary Image

The Image below (the bottom third is cut off) is a design view of the process flow you are going to create.



# Scope1

The outter scope catches any exceptions on the Service call itself. This is our first step in creating the app. We’re going to create the scope and its exception handler.

A scope is a unit of execution, I guess. Each scope can have it’s own exception handler. As each block we execute can fail, we provide each with its own scope in order to catch the exception and reply to the caller with a usefull error reply.

* Drag Scope from palette to center dot in design view

## Create Fault Handler for Scope 1

* Right click within the Scope1 border on the design screen, to the right of the word Scope1 should bring up a context menu
* Click Add
* Click Fault Handlers
* A box to the right of Scope1 with the text ‘Fault Handler’ is created

### Create Catch Block within Fault Handler

* Within the box titled ‘Fault Handler’ right click
* Click Add
* Click Catch (not Catch all)

### Set Properties for Catch

* Right click in ‘Catch’
* Click Properties
* Set the following properties for this Catch block
  + Fault Name : Standard Faults / invalidVariables
  + Fault Variable Name : ESBFault
  + Fault Variable Type : Fault message / faultMessage
  + Documentation : Thrown when the esb variables are not provided

This handler fires at the outer most level if the OpenESB http variables (mentioned later), that determine the location of the log SOAP server are not set

For now we’re going to stub it out. Do the following :

* Drag and drop Exit from palette into the drop dot of the Catch block

### Compile Application

There are a number of ways to compile the BPEL application.

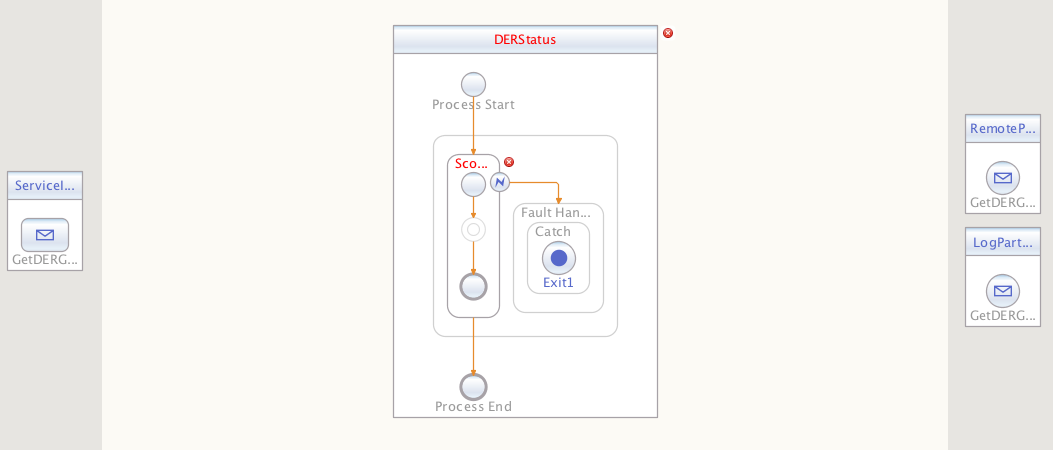
* While in design view, click the hammer in the top level icon menu
* Or… click the hammer/brush (does a clean and compile)
* Right click in the project view (on left), and click compile

Select the ‘Output’ view. This is above the design view we are currently in. It will show the results of the compile. Let’s fix one thing and ignore the other errors.

* One of the errors is concerned with obj2:GetDERGroupStatuses – click that error. Remove the offending line from the wsdl file and save.

The other two errors can be ignored for now. They are just stating that we don’t have a valid BPEL flow yet, which we don’t.

### Your Current Project View

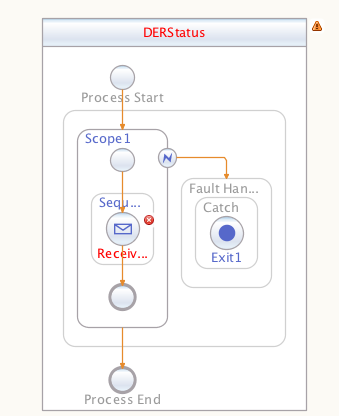


Warning : When building the project, there are problems with the UI. I find it a good idea to back the current state of the project up so that I can back up and start over at a good save point if things go bad.

Now is a good point top tar-zip your Status2 directory and set it off to the side, just in case.

# Receive SOAP Message

The first step in processing a SOAP message is passing it to the flow. This is done thusly :

* In the palette on the right, grab ‘Receive’ and drag it to the design area. This is a bit tricky the first time (image follows), you need to drag it to the ‘landing dot’ within Scope1.
* See below, ‘Receive 1’  
    
  
* Double click on the ‘envelope’, Receive 1
* This brings up a property editor
* Do the following :
  + Click PartnerLink and select ‘ServicePartnerLink’
  + Click the ‘Create’ button next to Input Variable
  + Name it ServiceIn
  + Click OK
* If you did this correclty, there is now an arrow line running from ServiceInPartnerLink to ‘Receive1’.

You have just ‘told’ the BPEL process to pass incoming SOAP messages to the flow. Congratulations.

If you compile now, it will give you an error stating that there is no reply activity associate with the receive. This is correct.

Save Project

# Scope2 Get Vendor URL

The first thing the process needs to do is retrieve the final destination URL for our SOAP message. The client sends the message to our BPEL process mediator, and the mediator looks up the final destination from our database for that particular message, and passes it along. Hence this first process.

This part of the tutorial is not fun. The interface doesn’t seem to correctly handle graphically calling java methods so we will have to drop into the ‘source code’ view, and get tricky.

## Adding Util Name Space

In the design window (center screen), above, are four buttons, Source, Design, Mapper, Logging, History.

* Click ‘Source’ to get to source view of the BPEL
* Near the top of the source code are name space definitions (xmlns:…
* Add the following line within the ‘xmlns…’ lines

xmlns:util="java://com.sixthc.bpel.spring.Utility"

* This is required in order to access the utility methods from BpelJava (log, GetURLByVendor)
* Save project

## Create Scope2

The get vendor process gets its own scope. Let’s create that now.

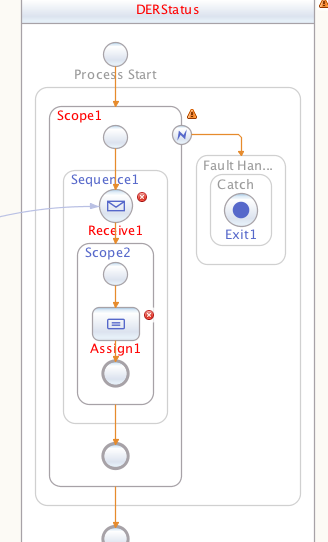
* Drag scope from the palette, and drop it below the receive activity in Scope1

You should now have Scope2 within Scope1

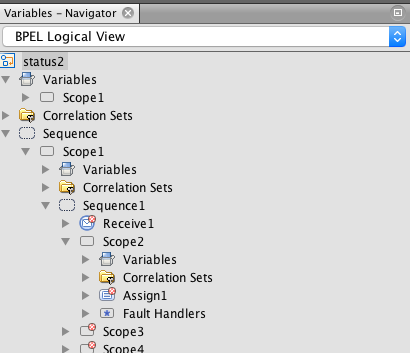
## Assigning Variables to Get Vendor URL

This is the first time you are seeing this, so I’ll be careful to be thourough in explaining it. We need to assign parts of the incoming SOAP message to our GetURLByVendor java call. This is how to do it.

Note : Assign is a big part of BPEL, and I will further address intricacies of it later within the document.

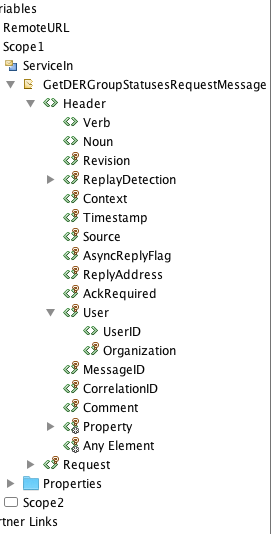
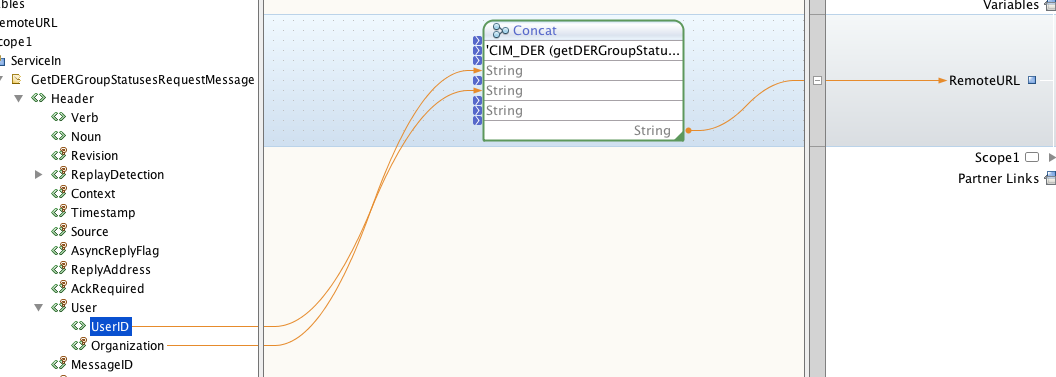
* Drag ‘assign’ from the palette (on the right) into Scope2 center stage
* The flow should like like the following when you’re done  
    
  
* Now the trickey part, new stuff

We’re going to assign attributes from the incoming SOAP message to a ‘concatenate’ (bear with me), then go into source view and replace the contatenate with our getVendorURL

* On the left side of the UI, below the navigator window there should be a Navigator window. If not, click menu / Window / Navigator
* The view should be in ‘BPEL Logical View’ mode and look like this :  
    
  
* We need to create a variable to hold the URL we’re going to fetch
* Right click on Variables
* Click ‘Add Variable’
  + Name : RemoteURL
  + Built-in Types / string
  + Click ok
* A new variable should appear in Navigator under variables

Save your project!

Now for the hard part.

* Double click on Assign1
* This brings up the Mapper view, variables down the left and right side of the mapping view
* On left side, expand the following : Scope 1 / ServiceIn / GetDERGroupStatusesrequest / Header / User
* This should show the attributes that are part of the Multispeak Header,and looks like this :  
    
  
* Now, on the ride side, click on ‘RemoteURL’ under variables
* A blue shaded line is drawn across the screen
* Above the mapper view, click on ‘String’ / ‘Concatenate’
* There should now be a ‘Concat’ box within the blue shaded line for RemoteURL
* Now type CIM\_DER (getDERGroupStatuses) within the first ‘String’ line in the Concat
* Click on Organization in left variable view, and drag it to the blue/white arrow on the left side of Concat box, drag it to the blue arrow below and closest to the string you entered.
* Click on UserID, and drag that to the blue arrow below the one you dragged Organization to
* Finally, drag the green arrow (right side bottom) of the Concat across the ‘RemoteURL’. When you are done the map will look like :  
    
  

Save your project!

What you have just done is concatenates three strings to the RemotURL variable. Congratulations, but this is not what we want to do.

What we want to do is pass the three variables, Operation name, userID, organization to the util.getURLByVendor java function, which will do a database lookup for us and return the correct final destination for the soap message. And this is how we do it.

* Click ‘Source’ button at top of design view
* Ctrl-f (search) for concat
* The line you find will look like this :

<assign name="Assign1">

<copy>

<from>concat('CIM\_DER (getDERGroupStatuses)', $ServiceIn.GetDERGroupStatusesRequestMessage/ns1:Header/ns2:User/ns2:Organization, $ServiceIn.GetDERGroupStatusesRequestMessage/ns1:Header/ns2:User/ns2:UserID)</from>

<to variable="RemoteURL"/>

</copy>

</assign>

* The above is the ‘language’ of OpenESB, xml declarative statements. The above basically says concat the text fields into one, and copy to the variable RemoteURL.
* And now for the trick. Replace the keyword ‘concat’ with ‘util:getURLByVendor’
* Insure there are no null vars within the call, sometimes Concat puts place holders in there, and verify the order of the arguments : ‘Cim\_DER…’, Organization, UserID
* The final code looks like :

<assign name="Assign1">

<copy>

<from>util:getURLByVendor('CIM\_DER (getDERGroupStatuses)', $ServiceIn.GetDERGroupStatusesRequestMessage/ns1:Header/ns2:User/ns2:Organization, $ServiceIn.GetDERGroupStatusesRequestMessage/ns1:Header/ns2:User/ns2:UserID)</from>

<to variable="RemoteURL"/>

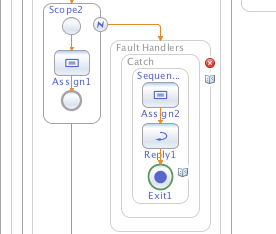
</copy>

</assign>

* If you jump back to design view and double click on assign1, you’ll see that your code has been transformed into a java call that takes 3 variables and assigns its result to RemoteURL.
* And because of our catch block, if the GetURLByVendor throws its user define exception, we’ll catch it… and handle it in the next section

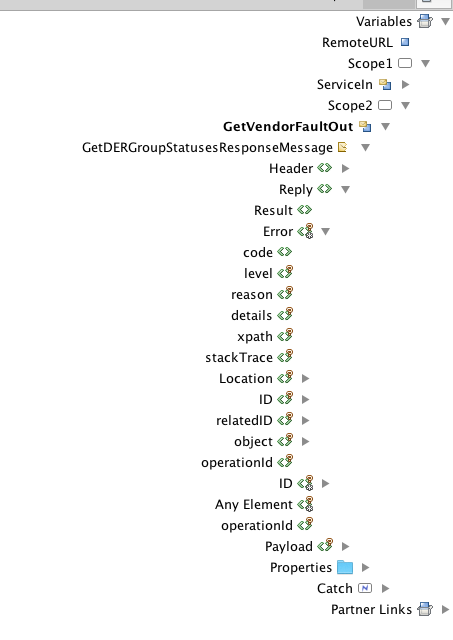
### Catching the VendorURL Exception

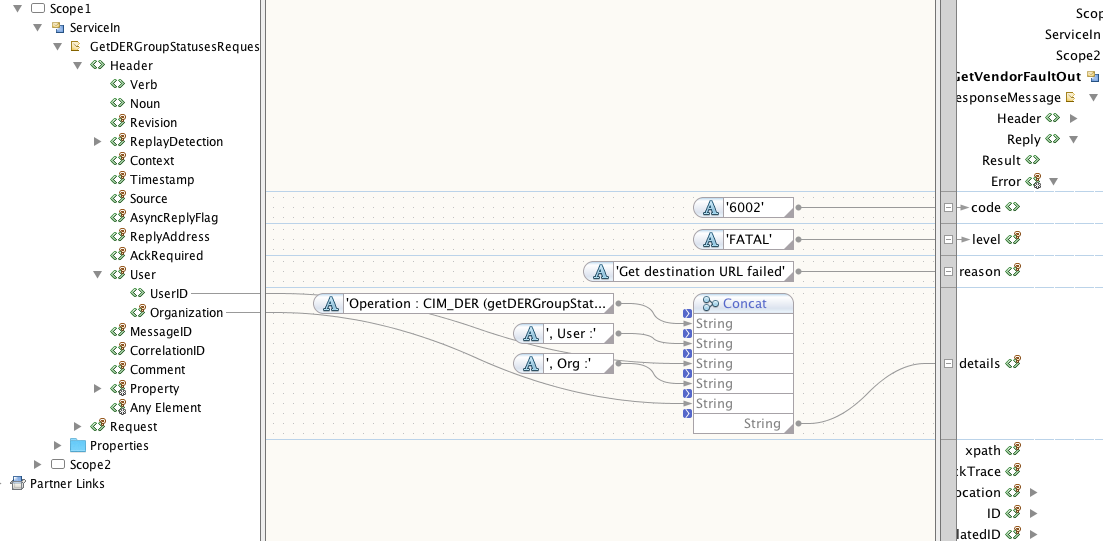
If all goes well, the remote URL has been set at this point and the flow moves on to Scope3, which will send the message to its final destination. Here we handle what to do if the database lookup fails for some reason.

* Right click within the Scope2 box for context menu
* Click add / Fault Handlers
* Right click within new Fault Handler box
* Click add / catch
* Drag assign from the palette into the catch block of scope2
* Drag reply from the palette into catch block of scope2, below the assign
* Drag exit form the palette into catch block of scope2, below reply
* The design view will look like this when you’re done :  
    
  
* Double click on Reply1 Within Sequence (you just dropped the reply into it)
* The reply property editor is display. Do the following :
  + Partner Link : ServiceInPartnerLink
  + Click ‘Create’ new to Output Variable :
    - Name : GetVendorFaultOut
  + Click OK
* A line should be drawn from your reply back to the ServiceIn Partner link. The BPEL will reply to the SOAP caller, passing back information we are about to impart via the Assign box above our reply.

A note on process flow : The purpose of the ‘Exit’ method below our reply is due to the workings of a flow chart. If we don’t specifically exit after sending back our message, Scope3 will be executed, and since we have a fatal error, we want to return our error message and end the process.

Now we’re going to create an error message to return to the calling process.

* Double click on the assign within the scope 2 fault handler, bringing up the mapper window.
* In the mapper, expand on the right side the following : Scope1 / Scope2 / GetVendorFaultOut / GetDERGropuStatusesResponseMessage / Reply / Error  
    
  
* I’m going to explain how to set the first variable. For the rest of them, refer to the image below and make your mapping view look like mine
  + Click the output var ‘code’ line within the reply error variable. This gives you a blue line across the mapper
  + Above mapper click String / String Literal
  + Type 6002 within the literal
  + Drag the blue arrow on right side of literal to the code variable
  + At this point you’ve asked the BPEL engine to copy ‘6002’, our error number, to the response it’s going to make to the caller
* Refer to the image below for the rest of the mapping for this fault handler



If you refer back to the flow, I’ll explain what you’ve just done.

1. If the vendor lookup throws an exception, the flow drops into the catch block
2. The assign statement copies variables and string literals to the GetVendorFaultOut reply you dragged into the sequence
3. The GetVendorFaultOut will send a message back to the caller containing the variables you assigned to the Error reply
4. The exit statement ends the process flow

Compile the application. There should be no errors now, only three warnings. For the pedantically inclined, below I’ve explained how to get rid of the rpc-literal warnings. The RemoteURL warning is fine for now.

Do the following to remove the rpc-literal warning :

* Click on the error
* Replace the line producing the warning with the following. The namespace prevents the warning from displaying. In Netbeans 8.2 this issue should not be evidenced

<soap:body use="literal" namespace="<http://j2ee.netbeans.org/wsdl/WS-I_R2717/src/WS-I_R2717violation>"/>

Now is a good save point. Save the project, and back it up to a tar-zip.

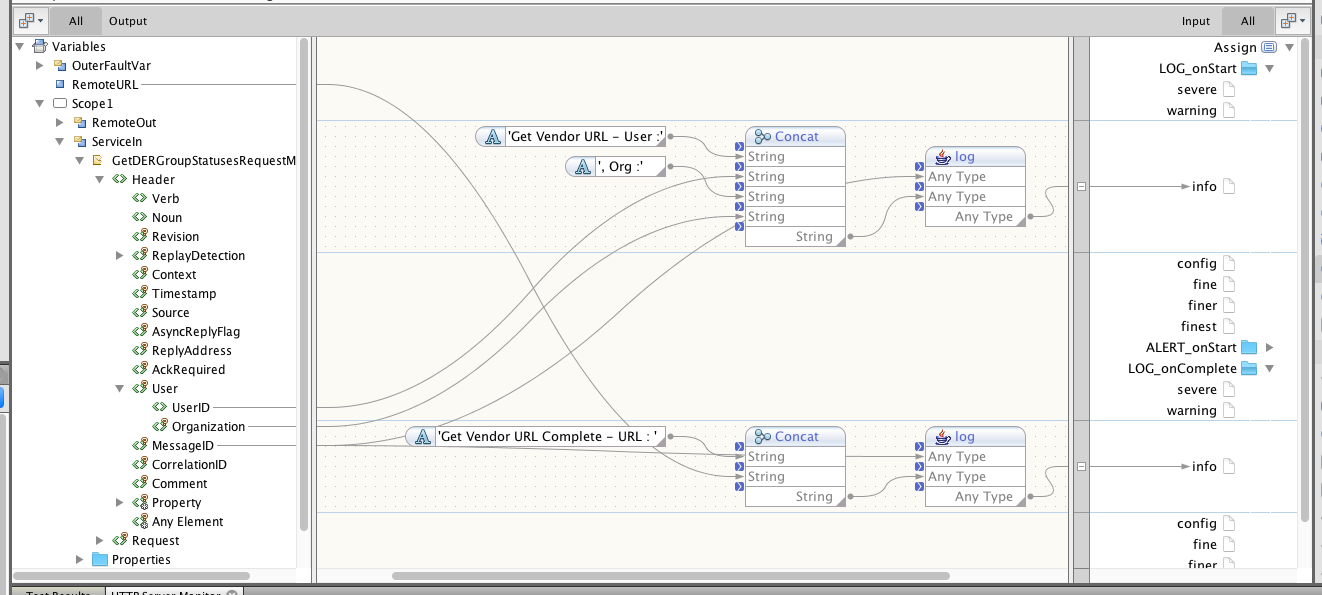
### Logging

It was requested that the various steps log certain variables during process flow, saving them to the database, to allow for better debugging by developpers. OpenESB supports logging to the console, but not to an external database, so I hacked their log method to not only send ‘debug’ messages to the OpenESB console (which you can see from within Netbeans or on the web console), but to send the log messages, keyed by the Unique ID each SOAP message ‘must’ have to our logging database.

Here I show you the ‘graphic’ way to do logging, which requires us to drop into source view due to the fact that our log utilizes a java method to save to the database.

* Click once on ‘assign1’ in Scope2, selecting it
* Click on ‘Logging’ button at top of design view
* Once in logging view, you may need to expand ‘Assign’ on the right side of the view, or it may already be expanded, showing you Log\_onStart, and other options
* Expand Log\_onStart, display sub items : severe, warning, info…

When we’re done the Logging view will look like the following illustration.



#### Log On Start

I’m going to show you how to do the first logging message. The second is left to you. You will be graded.

Note above in the illustration, the java call ‘log’ is the right-most operation, and is done twice, once before the assign, and once after. This next part is concerned with the first operation (Log\_onStart).

* Click on the ‘info’ item under LOG\_onStart, highlighting its line
* Drop a Concat down from the ‘String’ menu
* Type ‘Get Vendor URL – User :’ into the concat’s first line
* Type ‘, Org : ‘ into the concat’s second line
* Now it’s time to drag some attributes
* On the left side, expand Variables / Scope1 / ServiceIn / GetDERGrtoupStatusesRequestMessage / Header (Look at the left side of the above image)
* Expand the ‘User’ attribute on the left side under Header
* Drag UserID to the blue arrow beneath ‘Get Vendor URL…’
* Drag Organization to blue arrow beneath ‘, Org…’
* Please note : at this point your ‘concat’ looks different than the one pictured above, in that yours does not show string literals going into the concat. This is okay.

At this point you have created the message to be logged before Assign1 is executed. Now we are going to modify this to call our db logging method. Warning, source view upcoming.

* Drag another String Concat down, placing it to the right of the one you just created
* Drag the MessageID (from header on left side of screen) to the first blue arrow on this new concat.
* Drag the green output arrow of the first concat you created (the message concat you just created) to the second blue arrow of the new concat.
* Drag the green output arrow of the new concat to the ‘info’ item on the right side of the screen

Time to switch to Source View to finish up this trace.

* Select Source tab at top of design view
* Search source view for the string ‘get vendor url’
* You will see a <from> line containing an outter and inner concat. We are going to modify the first concat, the outer.
* Change ‘concat’ to ‘util:log’ The code will now look like (approximately)

<sxt:trace>

<sxt:log level="info" location="onStart">

<ns0:from>util:log($ServiceIn.GetDERGroupStatusesRequestMessage/ns1:

Header/ns2:MessageID, concat('Get Vendor URL - User :', $ServiceIn.GetDERGroupStatusesRequestMessage/ns1:Header/ns2:User/ns2:UserID, ', Org :', $ServiceIn.GetDERGroupStatusesRequestMessage/ns1:Header/ns2:User/ns2:Organization))</ns0:from>

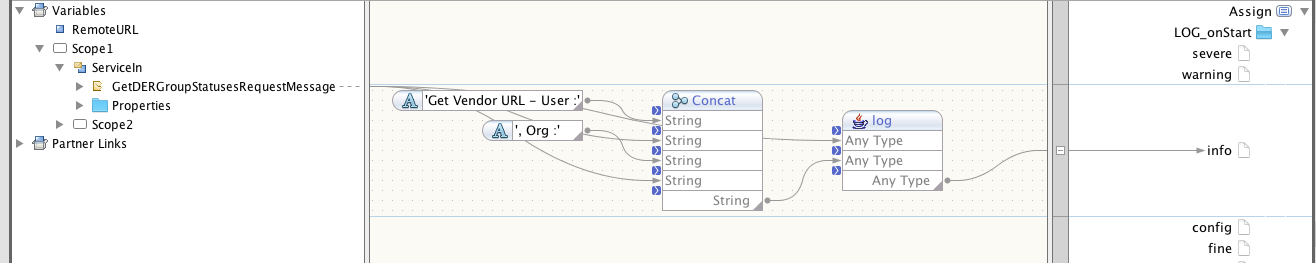
</sxt:log>

</sxt:trace>

An explanation of the ‘code’ above :

* The <trace> within the <assign name=”Assign1”> outer block will cause OpenESB to write to the console
* The <log> tag states level=info, log on start before the assign is done
* Tracing only has a <from> tag, ie; get data from somewhere and log it to our console
* If you look at the BpelJava, you will see that it :
  + takes two strings, the first is a unique id used to store the log entry within the database (note, this ID is unique to the SOAP message, not on a per log item basis), and a message to be logged.
  + Returns the message string (the second argument it is given)
  + This is a hack/trick basically, in that we log the message to our database, and then return it to the <from>, which logs it to the console.

To check your progress do the following :

* Click back to design view
* Click the light blue symbol to the right of scope2 that was just added to represent your trace
* The editor should switch to ‘Logging’ view and hopefully you see the following  
    
  

#### Log On Complete

When Assign1 completes the RemoteURL variable will be set. This is the final destination for the SOAP message our mediator BPEL received. This variable is derived based on a table in the database that returns a value based on the operation name, the User.UserID, and the User.Organization.

You get to do this one on your own. I will give the following hints though:

* On the left side of the Mapper, at top is the variable RemoteURL. That is the value you will be logging.
* Refer to the first illustration in this section for the final ‘look’ of the two log entries created for Assign1
* The OnComplete logging message will be : ‘Get Vendor URL Complete – URL : http://…

At this point Scope2 is complete. From this point on, I am going to be providing high level directions, rather than step by step. Be sure to save and possibly back up your versions of the code, as you might otherwise end up making a mistake and having to start all over.

## Summary of Scope2

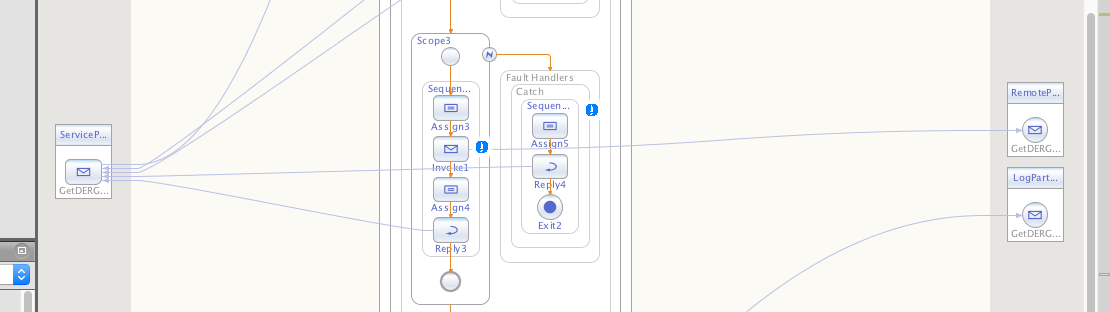
If you already know OpenESB, this section explains what we are trying to accomplish in Scope2.

1. The first step is to receive the SOAP DERStatus SOAP message. We dropped a Receive item from the palette as the first item of the sequence in scope1 to accomplish this.
2. We created a scope (Scope2) for the purpose of retrieving the URL of the SOAP server the mediator will pass the message to. The purpose of the scope is to allow us to catch the fault that our java method util:getVendorURL
3. We wrote a catch block that catches the getVendorURL’s exception, returns an error message to the calling client, and exits the flow
4. We logged the process on start and complete, sending message to the OpenESB log, and our database with information on the call
5. At this point, if there was no exception, process drops into Scope3

## Send SOAP message to Final Destination (Scope 3)

This part of the flow forwards the SOAP message onto the final destination. We are going to use the ‘Invoke’ action from the palette to do this. The hand holding is over at this point. Good luck.

Here is what the final product of Scope3 looks like.



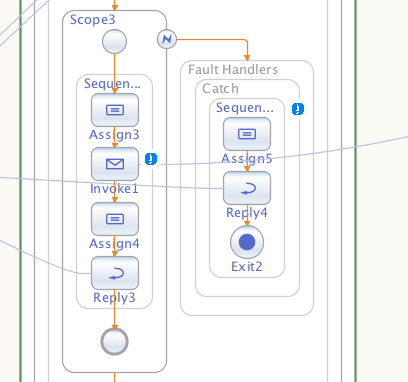
### Overview of the Flow

* Scope2 completes and flow drops to Scope3
* Scope3 Assigns the incoming SOAP message data to an ‘Invoke’
* The Invoke calls the remote server, the final destination
* Assign4 takes the results of the call to the Remote destination and assigns them to Reply3
* Reply3 sends the response from the remote server back to the originator of the message

### Building the Flow

* Drag and Drop A Scope below Scope2 (along the main flow)
* Drop Assign from Palette into Scope3
* Drop Invoke below Assign
* Drop Assign below Invoke
* Drop Reply below Assign
* Click Scope 2, right click, add / fault handler
  + Right Click in Fault Handler
  + Add / Catch
* Drop Assign into Catch Block
* Drop Reply below Assign in Catch block
* Drop Exit into Catch block

Below is illustration of Scope3 and its fault handler



#### Link the Invoke to the RemotePartner Link

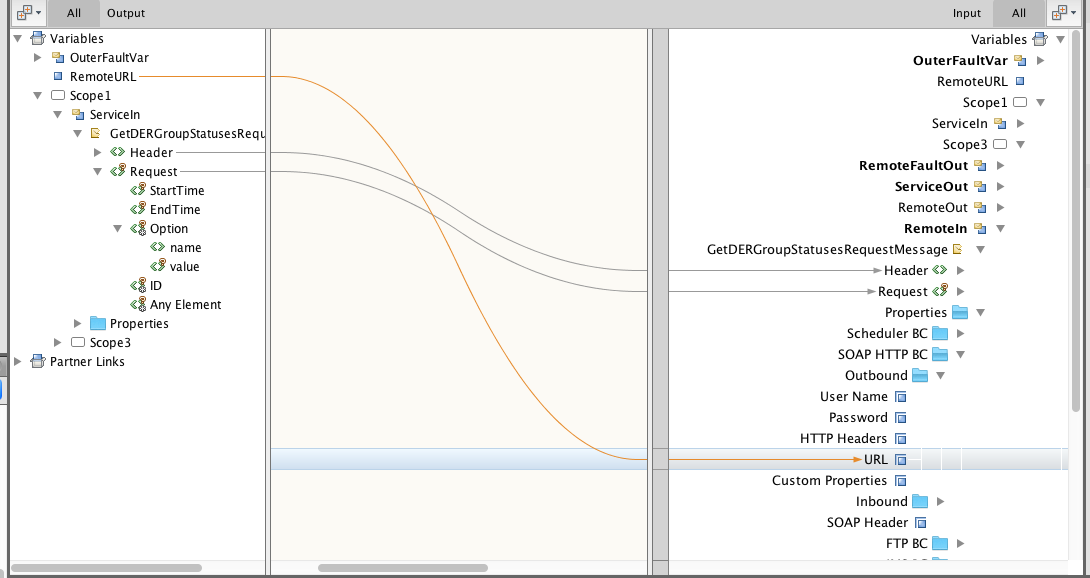
* Double click on the Invoke you just dropped
* Leave the Name as is
* Set Partner Link : RemotePartnerLink (this is the external link on right)
* Input Variable :
  + Click Create
  + Name RemoteIn
* Output Variable :
  + Click Create
  + Name ReouteOut
* At this point, the invoke should have drawn a line to the RemotePartnerLink, indicating it will invoke that lank (the remote SOAP server)

#### Assign the SOAP input data to the invoke statement

* Double-click on the assign above your invoke, bringing up mapper view
* Expand all the entries on left side under Scope1 / ServiceIn / GetDERGroupStatusesRequest (note : ServiceIn represents the data from the Service In Partner link on the left)
* We are trying to assign data from the service in to the invoke we’re about to do. The variables we just created above when we linked the Invoke to RemotePartnerLink were named RemoteIn, and RemoteOut.
  + RemoteIn represents data we want to send to the remote server
  + RemouteOut represents the data it will return to us.
* With that in mind, on the right side expand Scope1 / Scope3 / RemoteIn / and its children, exposing the Header and the Request attributes
* Now just drag ‘Request’ from ServiceIn / GetDERGroupStatusesRequest to the ‘Request’ under RemoteIn on the right side
* Drag the Header from the left to the Header on the right

#### Assign (part 2) assigning the RemoteURL

* Examine the illustration below. On the right side, under Request is the ‘Properties’ item. Expand it to show SOAP HTTP BC / Outbound
* Drag the RemoteURL variable from the left side to the URL in the illustration. This is going to override the URL given by the caller, and send the message to the URL we stored in RemoteVariable in Scope2.
* Also, this illustration shows what your first assign within Scope3 should look like.



#### Ignore Missing Data (Request)

The Request portion of the SOAP message is not required. It’s optional. However, since our assign statement assigns it to the output, OpenESB expects it to be there and will throw an exception if it’s not.

The solution is shown below. In the ‘BPEL Logical View’, left side bottom (?), do the following :

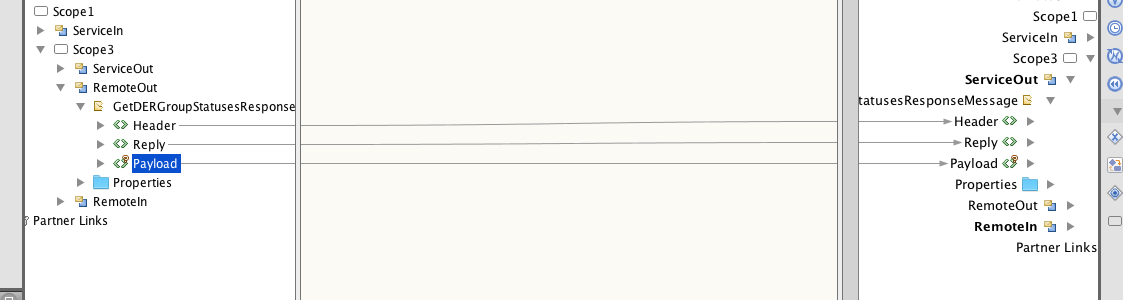
* Expand items until you uncover Scope3 / Sequence? / Assign? (the assign above)
* If you expand Assign3 (your number may differ), you will see three expressions beneath it : query = exp, query = exp, remoteIn=RemoteURL
* We’re looking for one of the query = exp. Right Click one of them, if the dialog that displays has a Copy From of ‘…Request’, that is the one we’re looking for. If not, right click the other
* The second config from the bottom is ‘Ignore Missing From Data’. Click the double arrow in the right side, select ‘yes’ and close.
* At this point, if a caller does not send a ‘request’ within their SOAP, OpenESB will not throw a fault, but will just pass on the header

#### Scope3 Bottom-most Reply

* Double-click the Reply2 at the bottom of Scope3
  + PartnerLink : ServiceInPartnerLink
  + Normal Response / Output Variable:
    - Click Create
    - Variable Name : ServiceOut
  + Click Ok
* The Reply should have drawn an arrow back to the service in partner link

#### Scope3 Bottom Most Assign (above bottom-most reply)

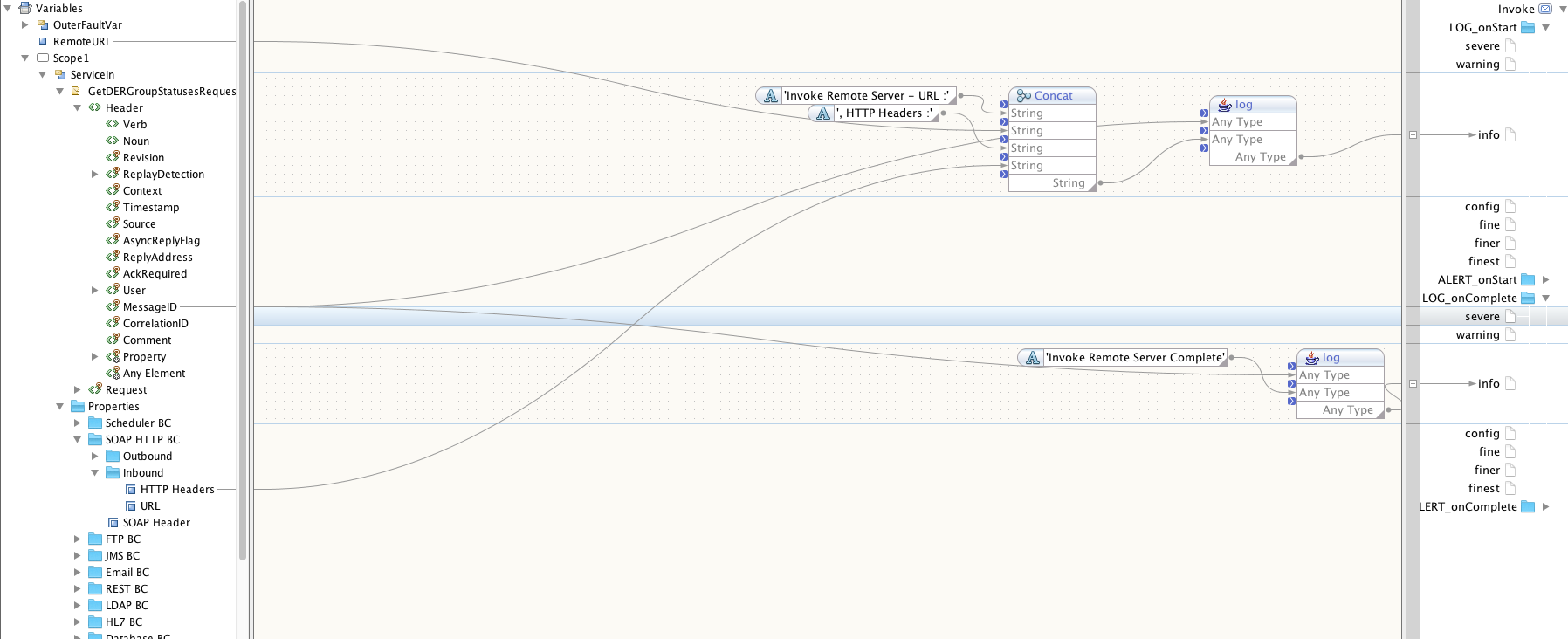
* Double click on the second Assign within Scope3
* On the left, find Scope3, RemoteOut, and expand the GetDerGroupStatusesResponseMessage to show Header, Reply, Payload
* On the right side, find Scope3, ServiceOut, GetDerGroupStatusesResponseMessage, and expand to show Header, Reply, Payload
* Drag left side Header to right side Header, left side Reply to right side Reply, and left side Payload to right side Payload



The above assigns the response data from the remote server to the reply, which will send the data back to the originator of the call.

#### Tracking Scope3’s Invoke

Now we’re going to log an ON\_start and ON\_complete trace for the invoke in Scope3. The illustration below should be enough for you to create the log request, but I provide a few notes after the illustration to help.



#### Log\_onStart

* Click once on the invoke in Scope3
* Click Logging at top to bring up logging view
* String 1 : Invoke Remote Server – URL :
* Drag RemoteURL as next string
* String 2 : , HTTP headers :
* Drag ServiceIn / Properties/ SOAP HTTP BC / Inbound / HTTP Headers
* Do a second concat like above
  + Remember first arg to this concat is MessageID
  + Second string is from the first concat
* Go to source view and replace the outer concat with Util:log

#### Log\_onComplete

* Same basic steps as Log\_onStart
* Expand Log\_onComplete
* Create a Concat
  + First String : MessageID (variable)
  + Second String :
* Go to source view and change concat to Util:log

#### Catch Handler for Scope 3

#### First we set the fault type

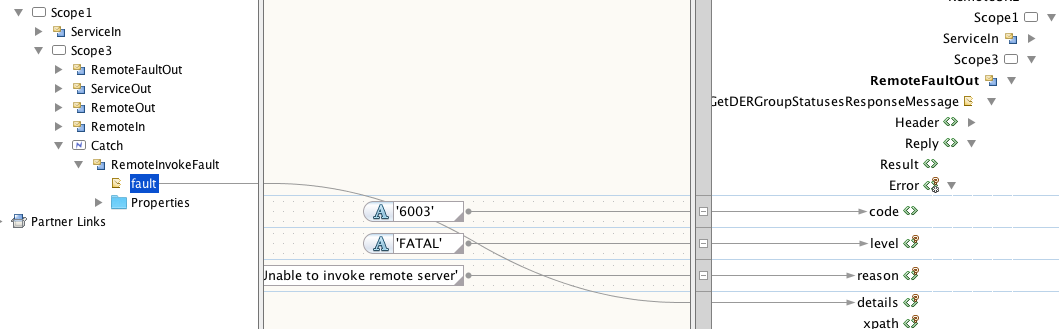
* Click the Catch block, then right click, properties
  + Fault Name : systemFault
  + Fault Variable Name : RemoteInvokeFault
  + Fault Variable Type : Fault Message / faultMessage

#### Next we set the reply

* Double click on Reply within Scope 3 Fault Handler
  + Partner Link : ServiceInPartnerLink
  + Normal Response / Output Variable : click ‘Create’
    - Name : RemoteFaultOut

#### Finally we assign return data to the reply

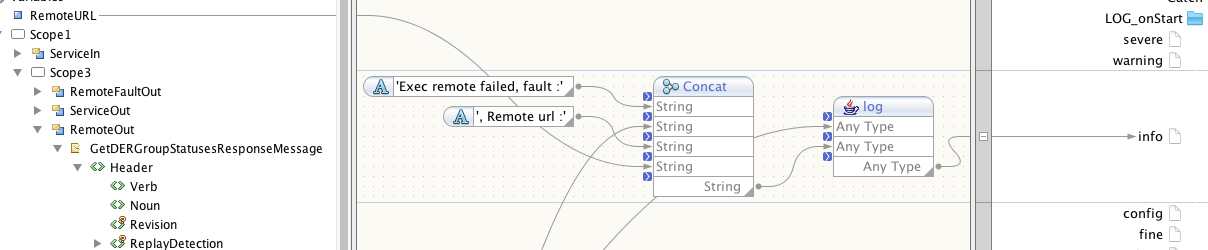
* Double click on Assign within Scope 3 Fault Handler
* Expand Scope1 / Scope3 / RemoteFaultOut / DerGroupStatusesResponseMessage / Reply / Error
* String Literal ‘6003’ to ‘code’
* String Literal ‘FATAL’ to ‘level’
* String Literal ‘Unable to invoke remote server’ to ‘reason
* Expand (left side) Scope1 / Scope3 / Catch / RemoteInvokeFault
* Drag ‘fault’ to details on right side



### Trace For Scope 3 Catch Handler

We’re going to log whenever the Scope3 fault handler is called

* Click on Catch of Scope3
* Click Logging view
* Log the failure
  + Concat (point strings to it)
  + String 1: Exec remote failed, fault :
  + Scope3 / Catch / RemoteInvokeFault / fault
  + String 2: , Remote URL :
  + RemoteUrl
  + Concat 2
    - Service In / MessageID
    - Concat1 (above)
    - Point concat2 at LOG\_onStart
  + Edit source, change concat2 to util:log



## Summary of Scope3

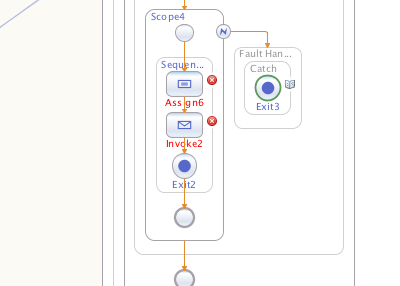
Scope2 set the url for the final destination of the SOAP message received by the mediator. Scop3 attempts to send the SOAP message to the final destination, receive the reply, and pass it back to the caller.

## Scope 4 (Logging Server)

The Scope4 process takes the incoming message and sends it to a logging server that stores the SOAP data in a database. This allows for the original message to be viewed at a later date. This is basically the same as Scope3, only we will not send a reply back to the originator, as two SOAP replies to the same message would be confusing/ bad.

Let’s get to it.

* Drag Scope below Scope3
* Drag Assign from Palette to Scope4
* Drag Invoke to Scope4
* Drag Exit to Scope4
* Drag Exit to the Scope4 fault handler



#### Invoke

* Double click the Invoke within Scope4
  + Partner Link : LogPartnerLink (this will cause the SOAP message to be sent to our log server)
  + Input Variable : Create ‘LogIn’
  + Output Variable : Create ‘LogOut’
  + Close Dialog

#### Assign

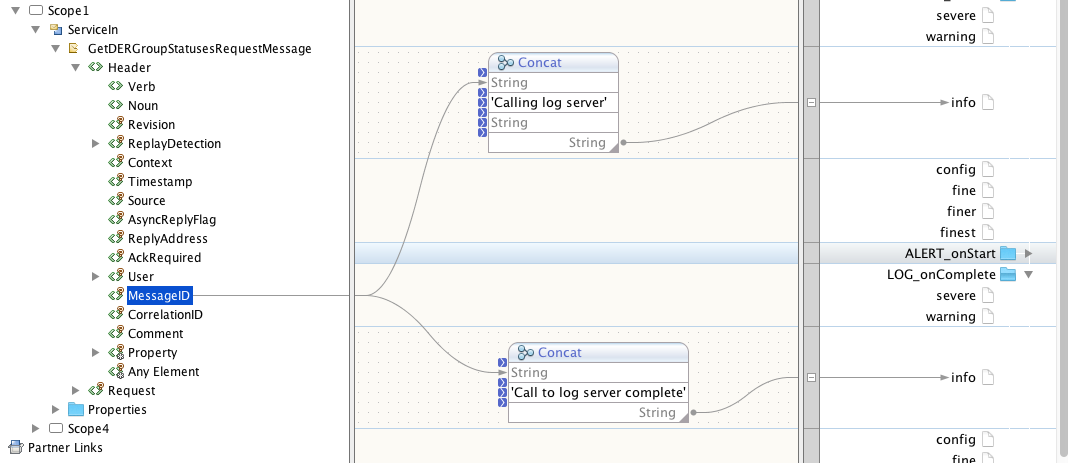
* Double Click Assign
* Expand left side to show ServiceIn / GetDERGroupStatusesRequestMessage, Header, Request
* Expand right side to show Scope1 / Scope4 / LogIn / … / Header, Request
* Drag header from left to right
* Drag request from left to right
* Edit properties of Assign to ‘ignore missing data’

#### Catch

* Right Click the ‘catch’
* Click properties
  + Fault Name : systemFault
  + Fault Variable Name : LogFault
  + Fault Variable Type : sxeh:faultMessage

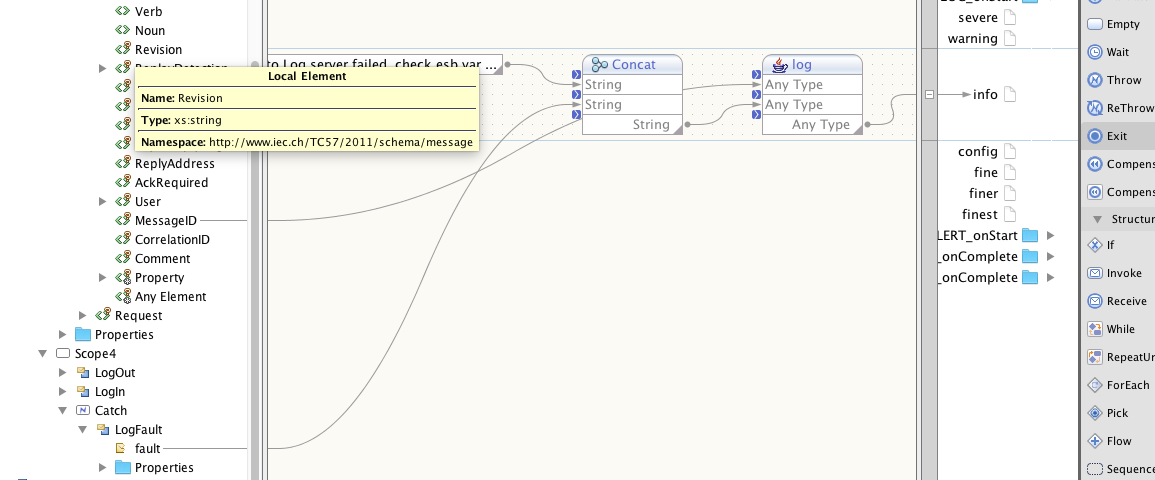
#### Trace Invoke

* LOG\_onStart : Calling log server
* LOG\_onComplete : Call to Log server complete



#### Trace Fault

* Log\_onStart
* Concat
  + Call to Log server failed. Check esb var…
  + Scope4 / Catch / LogFault / fault

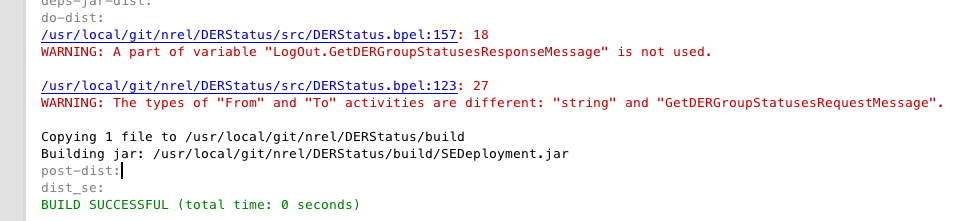


### Summary of Logging

That is it for logging. You will note that we don’t have a ‘Reply’ action because we don’t want to replies to be sent to the originator. The mediator throws away the reply from our logging server for this reason.

## Final Compile

Compile the app. There might be a few warnings. These are mine. I ignore them.

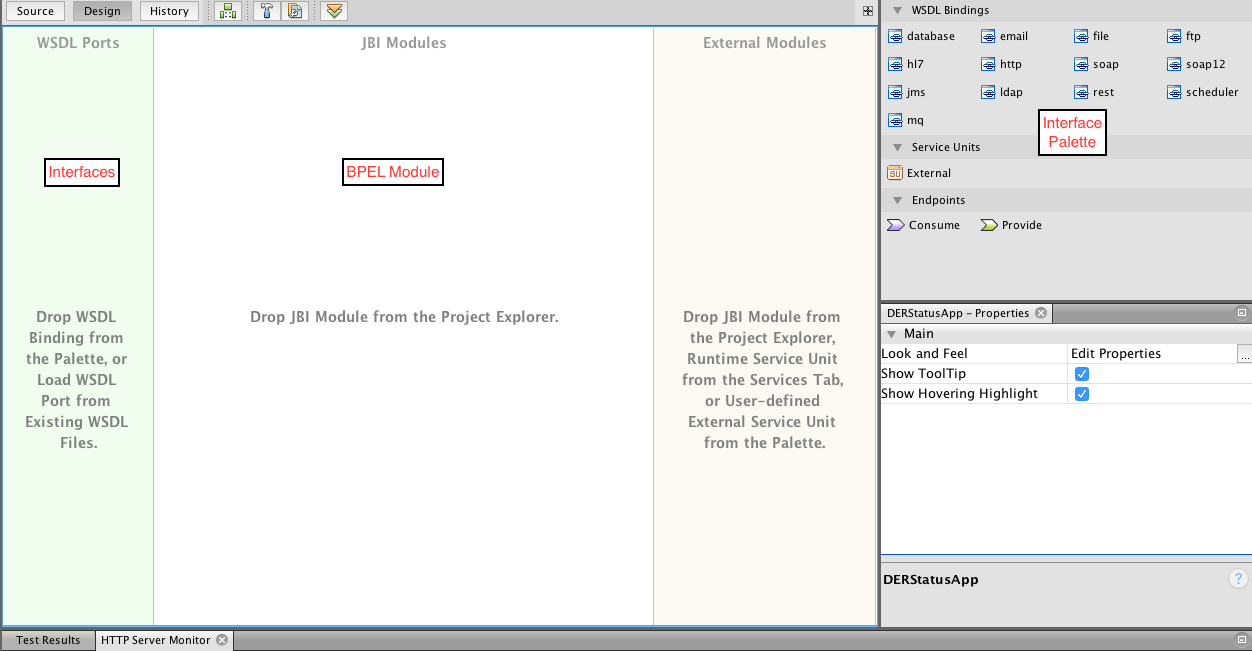


At this point it’s time to build the composite application in which the BEP process sits.

# Tutorial 2 – Composite Application

## Create Project

* New Project / SOA / Composite Application
* Next >
* Project Name : DERStatusApp (I generally use bpel project name plus ‘App’)
* In project window on left of screen, double click on ‘Service Assembly’
* You are presented with a few that looks like this :

  
Design View

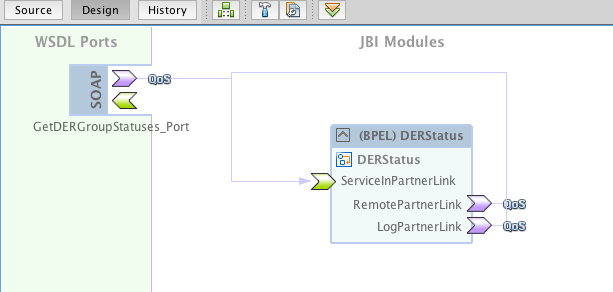
* Interfaces : Inputs and outputs to from the bpel module you creatd in tutorial part 1
* BPEL Module : Where you drag/drop bpel module created in tutorial part 1
* Palette : of interfaces

Set an important project property

* Right click on the project DERStatusApp
* Select properties
* Encoding : UTF-8
* Check : Skip (Composite Application Project) validation During Build

Let’s get to it. Listen carefully…

* On the left side of your UI is the project called DERStatus. Grab the project and drop it into the BPEL module area of the app view
* This is dicey (buggy), but at this point, save the app, hit build/clean, build, jiggle the item that you dropped into the BPEL module view until it transforms into this (it may take a few minutes, etc… before netbeans updates the look)



What this illustration represents :

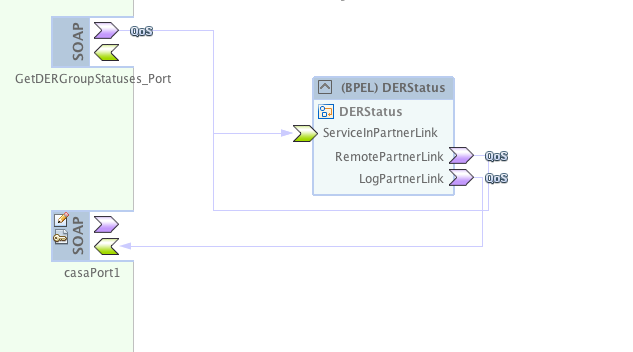
The SOAP box on the left side is the input. If you click on the line coming from it, it will be hi-lited and you will see that it feeds into the ServiceInPartnerLink. Get it? You did this when you built the BPEL module. This view is showing you’re your partner linkes.

There is another interface line. Click on the interface line that is coming out of LogPartnerLink. It seems to indicate this this is recursive, and as it is now, it is. But we’re going to change things around.

### Re-route LogPartnerLink

We want the LogParternLink to send the SOAP data to our logging service. This is how we go about doing it.

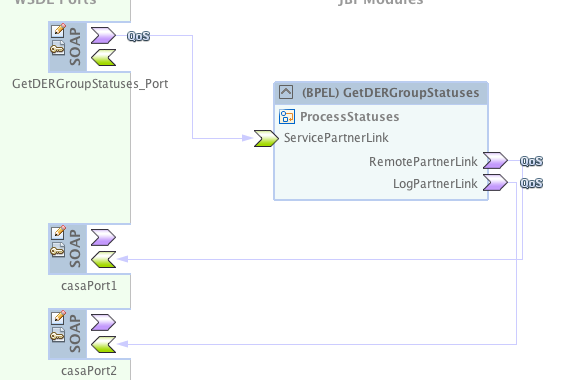
* Click on the link extending from LogPartnerLink back to ServiceInPartnerLink. It should hi-lite (glow)
* Click delete – The link disappears
* From the WSDL Bindings (palette on the right), drag ‘soap’, not ‘soap12’ to the left side (WSDL Ports) and drop it
* Click the purple (?) arrow coming out of LogPartnerLink, and drag to the green input arrow of the SOAP binding you just dropped. Your screen should look something like…



### Re-route RemotePartnerLink

We are going to repeat what we just did but with RemotePartnerLink. This is not really necessary, as we override its’ final destination, but we do it anyway.

* Click interface link coming out of RemotePartnerLink
* Delete it
* Drag SOAP from palette to left side
* Drag RemotePartnerLink (purple arrow) to input arrow of new SOAP binding
* We end up with this



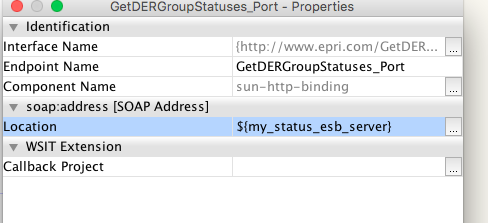
### Set HTTP Variables

Once we’re doing compiling this project we are going to set up some variables in the OpenESB web console. These variables, when set, override the default URL’s for the mediator web service, this is necessary because most WSDLs have a nonsense default-url that points to a server that doesn’t exist.

#### ServicePartnerLink SOAP IN

The first variable we are setting up is for the ServiceInPartnerLink.

* So click on what is probably the top-most SOAP in the WSDL Ports window, the one that connects to ServiceInPartnerLink
* Right click on it : Clone wsdl port to edit (it will warn you, ignore warning)
* Now there is a pen/paper icon in the SOAP
* Click it
* Make up a variable (Location) name : the one I did for this SOAP port was ${status\_esb\_server}.
* Set Location : ${my\_status\_esb\_server}



RemotePartnerInLink SOAP IN

* Set Location for the SOAP that links to RemotePartnerInLink
* Location : ${my\_status\_remote\_server}

LogPartnerLinkIn SOAP IN

* Set Location for the SOAP that links to LogPartnerInLink
* Location : ${my\_status\_log\_server}

### Compile the App

Compile the application and check the output. A few warnings that we ignore. We are almost ready to deploy the application to the OpenESB server.

# Tutorial Part 3 – OpenESB Server Setup

We created three variables within our DERStatusApp;

* ${status\_esb\_server} – the URL of the mediator service
* ${my\_status\_remote\_server} – URL of end point mediator forward soap message to, this is actually not used, but we set it anyway
* ${my\_status\_log\_server} – the destination of your logging server (if you have one)

Now we’re going to set those variables within OpenESB.

## HTTP Variables

Let’s go.

* In netbeans, on left side (project window area) click ‘Services’ at the top to see services view. If you haven’t already pointed netbeans to your OpenESB instance, you’ll have to do that (not in scope of this document)
* Expand Servers
* Right click on OpenESB Standalone
* Start (if not already running)
* Right click on OpenESB Standalone
* View Web Console
* Log in : admin / not sure what password is
* Click Components on left side
* Click sun-http-binding
* Click Application Variables
* Set your variables :
  + +Add
    - Name : status\_esb\_server
    - Type : string
    - Value : <http://localhost:8081/epriConnect/getDERGroupStatuses>
  + +Add
    - Name : my\_status\_remote\_server
    - Type : string
    - Value : <http://localhost:8080/epriConnect/executeDERGroupDispatches>
  + +Add
    - Name : my\_status\_log\_server
    - Type : string
    - Value : <http://localhost:8080/epriConnect/getDERGroupStatuses>

What you have just done is created three variables that will override the default URLs for the Service Port (in), remote Port (not used), and logging port for your application.

## Promote Application to Server

Back to netbeans. Let’s copy the new application to the esb server and start it up.

* Click on the project (go back to project view from services) DERStatusApp
* Right click it
* Click Deploy

If all goes well, the output window will end with ‘Build Successful’

Go to the web console

* Click on left side Service Assemblies
* Your DERStatusApp should be there an running.

### End Tutorial.

As for testing, that is not the purpose of this document. However, verify that you have servers running at the URLs you gave my\_status\_remote\_server and my\_status\_log\_server, and send your SOAP message to the Url defined by status\_esb\_server. Good luck.