Code Bundle Notes

This document explains the contents of the code bundle associated with each chapter.

# Chapter 1, Building an SOA Suite Cluster

SOA-Cookbook-Cluster-Workbook.xlsx is an Excel format workbook to assist in setting up a SOA Suite cluster according to the Enterprise Deployment Guide. The following worksheets are in the workbook:

* WLS\_Machines to identify physical machines and their associated WebLogic servers.
* OHS\_Machines to identify the physical machines hosting OHS and the port numbers and protocols used by those OHS instances.
* WLS\_Ports to identify the port numbers used by different types of WebLogic servers.
* WLS\_VIPs to identify the floating IP addresses used by some WebLogic managed servers.
* Mount\_Points to identify the shared files systems required the cluster.
* Floating\_Hostnames to identify the hostnames used by WebLogic managed servers with floating IP addresses.
* LB\_Endpoints to identify the endpoints exposed through the load balancer.

# Chapter 3 Working with Transactions

This contains a JDeveloper application (SOACookbookChapter3Transactions.jws) that can be examined to understand different aspects of transactional behaviour in SOA Suite. Each aspect is examined in a different project.

Deploy the projects in the order given below, as some projects have dependencies on other projects.

## Configuration.jpr

This project contains files needed to set up the environment for the other projects.

As a database user run the CreateUser.sql script to create a user called “TEST” in a database. The default password for the user is “test”. The user needs to be created in the same database as the SOA schemas.

Having created the user then login to the database as user “TEST” and run the createTestScehma.sql script to create the database tables required by other projects.

After creating the schema then perform the following to configure the WebLogic environment from the WebLogic console:

* Create a new Data Source (Services->Data Sources)
  + Set the Name to “TestDS”
  + Set the JNDI Name to “jdbc/testDS”
  + Use an XA Database Driver for the target database such as “\*Oracle’s Driver (Thin XA) for Instance connections; Versions:9.0.1 and later”.
  + Provide connection details for your target database, database user should be “TEST” user created earlier.
  + Target the data source at the server or cluster running SOA Suite.
* Create a new Database Adapter Connection Factory (Deployments->DbAdapter->Configuration)
  + Set the JNDI Name “eis/DB/Test”
  + If necessary create a new Configuration Plan, a good location for the plan is <MW\_HOME>/user\_projects/plans/<DOMAIN\_NAME>/DB.
* Configure the Database Adapter Connection Factory
  + Set xaDataSourceName to “jdbc/testDS”

In JDeveloper modify the settings of the application specific Database Connection “SOAVBOX\_TEST” to point to the newly created Test schema.

## TransactionID.jpr

This project obtains the current transaction ID as reported by the database and the WebLogic server. The DB transaction ID is obtained by raw SQL execution through the DB Adapter.

Deploy the target TransactionID to a SOA server. If there are problems deploying then apply patch 13569360.

The composite takes two parameters

* JavaRequired – set to “true” to retrieve Java XA Transaction ID
* DBRequired – set to “true” to retrieve Database Local Transaction ID

Verify it works by using the Test function and setting both parameters to “true”.

## SingleTransactionTest.jpr

This project checks that a database insert occurs in the same transaction context as the SOA transaction.

Modify the SingleTransactionTest\_cfgplan.xml to point to the SOA server where the composite will execute, replacing SOA\_HOST\_NAME and SOA\_PORT\_NUMBER with appropriate values (this is used to invoke the TransactionID composite).

Deploy the composite to the SOA server. Test it by passing in the following parameters

* JavaRequired – set to “false” as we are not interested in Jave for this test.
* DBRequired – set to “true” to get the DB transaction ID.

The response has the following XPath values

* TransactionResponsePair/preCall/DBTXID show the DB transaction ID used by SOA Suite before the call, this is obtained by using the transactionID composite.
* TransactionResponsePair/inCall/DBTXID shows the DB transaction ID used by SOA Suite during the call, this is obtained by inserting data into the database and having a pre-insert trigger add the current database transaction ID to the table. A select statement then retrieves the transaction id from the table.
* TransactionResponsePair/postCall/DBTXID show the DB transaction ID used by SOA Suite after the call, this is obtained by using the transactionID composite.

All three DBTXID values should be the same showing that the DB transaction is part of a single global transaction.

## RollbackTest.jpr

This project consists of a controlling BPEL process (TestXABPELProcess) and two sub-processes (RollbackBPELProcess and XARollbackBPELProcess). The two sub-processes are actually the same BPEL code. The sub-process BPEL inserts a row into a database table and then checks if it should throw a RollBack fault. One sub-process is configured to start a new transaction when it is invoked (RollbackBPELProcess) and the other is configured to use the existing (callers) transaction (XARollbackBPELProcess, configured in composite.xml with property transaction=’required’).

The controlling process (TestXABPELProcess.bpel) calls one of the two sub-processes twice, the first time with Rollback=false and the second time with Rollback set to the input Rollback parameter.

The composite takes 3 parameters:

* remark – A comment to describe the test.
* rollback – true if the second sub-process call should throw a Rollback fault.
* xa – true if the sub-process should be part of the same XA transaction as the caller.

Deploy the composite to the SOA server and then run four tests as outlined in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Remark | Rollback | XA | Notes |
| RB=F,XA=T | false | true | This will cause both inserts to complete and both rows can be seen in the database table, with the same remarks and same ACTUALTXID value. This shows that both sub-process invocations were part of the same XA transaction. BPEL sub-process has transaction=’required’ in composite.xml to cause this. |
| RB=F,XA=F | false | false | This will cause both inserts to complete and both rows can be seen in the database table, with the same remarks but different ACTUALTXID values. This shows that the sub-process invocations were part of different transactions. BPEL sub-process has transaction=’requiresNew’ or blank (default value) in composite.xml to cause this. |
| RB=T,XA=T | true | true | This will cause neither insert to complete and no rows can be seen in the database table for this case. This shows that both sub-process invocations were part of the same XA transaction and the whole XA transaction was rolled back, including the calling composiste. Because the caller was part of the transaction when the Rollback fault was thrown it was unable to catch it. BPEL sub-process has transaction=’required’ in composite.xml to cause this. |
| RB=T,XA=F | true | false | This will cause the first insert to complete and the second to fail. One row can be seen in the database table because the sub-process executed in its own transaction. This shows that the sub-process invocations were part of different transactions and the transactions committed or rolled back independently. It also shows that the caller was part of a separate transaction and so was able to catch the Rollback fault thrown by the sub-process. BPEL sub-process has transaction=’requiresNew’ or blank (default value) in composite.xml to cause this. |

## CompensationTest.jpr

This project demonstrates compensation handling by writing transactions and reversing transactions to a file. Each file entry has a date, the input parameter and a comment about the action being performed within the process.

The file location should be configured in the CompensationTest\_cfgplan.xml configuration plan by replacing TARGET\_FILE\_DIRECTORY with the actual target directory where the file should be written. After setting up the configuration plan then the composite may be deployed to the SOA server.

A single BPEL process (CompensationBPELProcess) performs the following within the scopes indicated:

* Scope 1
* Write to File
* If input is 1a then throw a fault
* If input is 1 then throw a fault
* Scope 2
* Write to File
* If input is 2a then throw a fault
* If input is 2 then throw a fault

Each scope has a compensation handler that writes a reversing transaction to the file. A process wide fault handler is used to invoke compensation when a fault is thrown.

The composite takes a single parameter ‘failStep’ that indicates where in the process the fault should be thrown. Test the composite with the parameter set according to the following table:

|  |  |
| --- | --- |
| failStep | Notes |
| 0 | This causes no faults to be thrown and two entries will be written to the file. |
| 1a | This causes a fault to be thrown within the first scope and caught by the process wide fault handler. As the first scope did not complete then no compensation is performed and only the first entry will be written to the file with no reversing entry. This shows the importance of avoiding faults in a scope after completing an action in the scope that may need reversing. |
| 1 | This causes a fault to be thrown after completion of the first scope and caught by the process wide fault handler. As the first scope completed then the compensation handler for that scope is executed to provide a reversing transaction. This reversing transaction can be seen in the output file. The file will have the first entry and reversing entry. This shows how a reversing transaction can be applied by putting a compensation handler on a scope. |
| 2a | This causes a fault to be thrown within the second scope and caught by the process wide fault handler. As the second scope did not complete then no compensation is performed for the second scope and only the first scope compensation handler will be invoked. The file will show the first and second entries with a reversing entry for the first entry. Again this shows the importance of avoiding faults in a scope after completing an action in the scope that may need reversing. It also shows that completed scopes have their compensation handlers invoked. |
| 2 | This causes a fault to be thrown after completion of the second scope and caught by the process wide fault handler. As the second scope completed then the compensation handler for both that scope and the first scope is executed to provide reversing transactions. The reversing transactions can be seen in the output file. The file will have the first and second entries followed by reversing entries for the second and first entries. This shows that compensation handlers are invoked in reverse order to the order in which the scopes were executed. |

# Chapter 4 Mapping Data

This contains a JDeveloper application (SOACookbookChapter4Mapping.jws) that can be examined to understand different aspects of data mapping. Each aspect is examined in a different project.

## ArrayProcessing.jpr

A sample transformation (ArrayTransformation.xsl) is used to convert the data into the required format by using an XSLT for loop. Quotes are sorted by using the XSLT sort directive. The min-value-among-nodeset function is also used to calculate the lowest price offered. There is a sample input document (ArrayTransformation-Source.xml) for testing the XSLT transform.

This project has a sample BPEL process (ArrayProcess.bpel) illustrating how to iterate over an array in BPEL. Deploy the composite to a SOA instance. The composite can be testing by pasting the root element from the sample input for the XSL transformation (ArrayTransformation-Source.xml) into the XML view of the test window in EM.

## EXMEJB.jpr

A simple EJB project that needs to be deployed to the SOA server and is used by the EXMMapping project. JDeveloper may ask for an MDS repository, any MDS destination can be specified. The default application name is SOACookbookMapping\_EXMEJB\_ejb1.

## EXMMapping.jpr

A project that demonstrates the use of EXM mapping file (Spring QuoteInterface.exm, EJB EXM\_Mapping\_EJB.exm). The mediator maps to a Spring component or an EJB reference depending on the input. Deploy the composite to a SOA server and sample input files are provided (SampleInputEJB.xml and SampleInputSpring.xml)

## MiscMappings.jpr

A project that demonstrates a number of different mappings inside a BPEL process.

InitialTransform and CorrectedInitialTransform demonstrate how to avoid unwanted mappings in XSL.

InitialAssign, CorrectedInitialAssign, AssignMissingElement and Corrected AssignMissingElement demonstrate how to avoid errors with non-existent nodes in the source or target of an assign.

To test provide a “val1” string but leave the “val2” string blank.

Test the composite in Development mode to be able to see the effect of the different mappings.

# Chapter 9 Integrating Java with SOA Suite

This contains a JDeveloper application (SOACookbookChapter9Java.jws) that can be examined to understand different aspects of integrating Java. Each aspect is examined in a different project.

## CustomXPath.jpr

This is a Java project that builds a custom XPath library. This project includes the Apache Commons Math library which is available here <http://commons.apache.org/math/download_math.cgi>. Deploy the project to create a jar file.

The jar file can be registered with JDeveloper by adding the deploy/XPathStats.jar file in Tools->Preferences->SOA.

The jar file must be copied to the SOA Suite <ORACLE\_HOME>/soa/modules/oracle.soa.ext\_11.1.1 directory. Register the jar file by running the command “<MW\_HOME>/ modules/org.apache.ant\_1.7.1/bin/ant” in the <ORACLE\_HOME>/soa/modules/oracle.soa.ext\_11.1.1 directory.

Before being used both JDeveloper and the SOA server must be restarted.

## XPathTest.jpr

This SOA project tests the custom XPath function built and deployed in the previous project.

To test the XPath function in JDeveloper test the XSLT transform Request\_to\_Response.xsl using the input file Request\_to\_Response-Source.xml.

Deploy the project to the SOA server and then test it by providing an array of numbers.

The composite consists of three components:

* XsltTestMediator – a mediator that takes the original input and uses an XSLT transform to convert the data to result format and also stores the standard deviation of the input in the XsltStdDev element of the result by using the custom XPath function in an XSLT transform.
* AssignTestMediator – a mediator that takes the output from the previous mediator and adds to it the standard deviation of the input in the MediatorStdDev element of the result by using the custom XPath function in a mediator assign. This mediator also converts the output of the BPEL process into the correct format, again using an Assign.
* AssignTestBPEL – a BPEL process that takes the output from the previous mediator and adds to it the standard deviation of the input in the BpelStdDev element of the result by using the custom XPath function in a BPEL assign.

## EJBSample.jpr

This EJB project provides the EJB used by other projects.

Deploy the EmployeeEJB deployment to the SOA server. Deploy the EmployeeClient deployment to a jar file.

## EJBClient.jpr

This Java project contains a Java Bean that calls the previously deployed EJB.

You can test the bean and the previous EJB by modifying the PROVIDER\_URL in EmployeeEJBClient.java to point to your SOA server and then running the EmployeeEJBClient from within JDeveloper. You should see a list of addresses.

Deploy the EJBClientBean to a jar file.

## SOAejb.jpr

This SOA project tests calling the EJB in the previous project.

Copy the EmployeeClient.jar file from the previous project to the SCA-INF/lib directory of this project.

Deploy the composite to the SOA server.

The composite consists of a Mediator that routes the request to the EJB deployed in the previous project, using XSLT transforms to map the data to/from EJB format.

To test it use the test function in EM.

## SpringSample.jpr

This SOA project test the Spring bean from the EJBClient project. It also demonstrates using an EJB as a reference to a Spring component.

Copy the EJBClient.jar from the EJBClient project to SOA-INF/lib.

Deploy the project to the SOA server.

The composite consists of a Mediator that routes the request to a Spring component that exposes the EJBClient bean. This bean in turn calls the EJB deployed in a previous project. The Mediator uses XSLT transforms to map the data to/from EJB format.

To test it use the test function in EM.

## EmbeddedJava.jpr

This SOA project demonstrates accessing and setting BPEL variables from Java within a Java Exec function.

Deploy the composite to the SOA server.

The Java Embedding in the BPEL process retrieves the input text and then sets the output variable to be a combination of the input text and the parent class of the executing code.

To test it use the test function in EM.

# Chapter 10 Securing Composites and Calling Secure Web Services

This contains a JDeveloper application (SOACookbookChapter10Security.jws) that demonstrates securing and calling secured web services through Web Service Manager policies.

Before deploying the composites in this application it is necessary to do the following:

* Within EM console use the WebLogic Domain-><Domain Name>->Security->Credentials to create a new key called “Role1” in the map “oracle.wsm.security”.
* Set the username and password for the weblogic user.
* Follow the steps in recipe “Creating a new, group-based authorization policy”.
* Name the policy “cookbook/binding\_authorization\_Role1s\_policy”
* Choose “Administrators” as the role.

## EchoApplication.jpr

This SOA project provides a simple Echo service that has three interfaces, an unprotected interface, an interface protected by HTTP Basic security and one protected by WSS Security.

Deploy the project and then test it from EM as follows:

* From the EchoApplication screen test the EchoService, notice that no security must be provided.
* From the EchoApplication screen test the BasicSecurityEchoService, notice that the invocation fails with a “403 Forbidden” error.
  + Add Security->HTTP Basic Auth credentials to the request for the weblogic user and notice that the test now succeeds.
* From the EchoApplication screen test the WSSecurityEchoService, notice that the invocation fails with a “WS-Security” error.
* Add Security->oracle/wss\_username\_token\_client\_policy credentials to the request for the weblogic user and notice that the test now succeeds.

## EchoClientApplication.jpr

This SOA project uses a BPEL process to invoke the three interfaces to the previous project, providing appropriate authentication as required.

Before deploying the project edit the EchoClientApplication\_cfgplan.xml and replace the “SOA\_HOSTNAME:SOA\_PORT” with the correct hostname and port of your SOA server.

Deploy the application and test it using the EM test screen.

Examine the trace of the BPEL process to see the call to the 3 different interfaces.

# Chapter 13 Monitoring and Management

This contains a JDeveloper project (SOACookbookChapter13Management.jws) that contains two projects that are instrumented for BAM.

Before deploying the projects it is necessary to do the following:

* Configure the BAM adapter as explained in recipe “Configuring BAM Adapter”.
* Deploy the Monitor Express dashboard to the BAM server as explained in recipe “Deploy Monitor Express to BAM”.

## EchoComposite.jpr

This composite is instrumented for BAM and can be deployed to the SOA server.

Use the EM test console to execute the composite a few times then open the BAM console and navigate to the Shared reports->Samples->Monitor Express Dashboard and verify that the dashboard is being populated.

## EchoClientComposite.jpr

This composite is instrumented for BAM and calls the previous composite through direct and web service bindings.

Edit the EchoCLientComposite\_cfgplan.xml and set the SOA\_HOSTNAME:SOA\_PORT to the correct hostname and port for your environment.

Also go into the source view of the BPEL process and replace SOA\_HOSTNAME:SOA\_PORT with the correct hostname and port for your environment.

Deploy the composite and test it using EM. The input parameter is the number of iterations of the BPEL process to perform.

The BPEL process invokes the previous composite in one of three ways:

* Via a Web Service call using the hostname which should be optimized into a local call by the SOA infrastructure.
* Via a Web Service call using the name localhost to use the loopback adapter.
* Via a SOA direct call which uses the EJB soa-infra infrastructure.

The time for each of these calls is totalled up and reported at the end of the run.

This results in the EchoComposite composite being called 3 x the number of iterations which provides a way to view continuous updates of the monitor express dashboard.