Practices for Lesson 14: Testing and Debugging

Practices for Lesson 14: Overview

Practices Overview

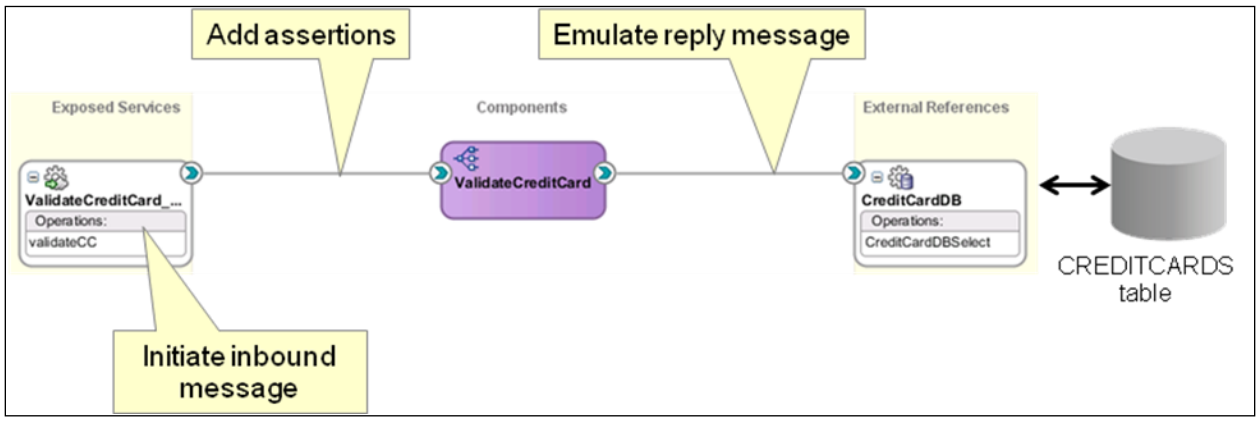
In this practice, you create and execute a test suite that contains two test cases for a composite application.

Recall that in an earlier practice, you built a composite application to validate a credit card purchase by checking the card number and the credit limit against the information in a database. Suppose that your company is preparing to accept private cards that begin with the four digits 9000. Since this modification, the routing rule has not been tested. The reason for the path not being tested is that the `BCA_CREDITCARDS` database table does not have credit cards that begin with the digits 9000. However, with the use of test cases, you have the ability to emulate the Database adapter response to return a valid or invalid status for credit cards that begin with 9000 as if the credit card existed in the database.

With the preceding scenario in mind, in this practice, you create a test suite for the CCValidate composite process and the following test cases:

- Test a credit card number starting with the digits 9000 and returning a `VALID` status.
- Test a credit card number starting with the digits 9000 and returning an `INVALID` status.

The following image illustrates these test case scenarios:



Practice 14-1: Creating a Test Suite for the CCValidate Composite

Overview

In this practice, your task is to create a test suite in the CCValidate composite application. When you create a test suite, you also create its first test case. The test case that you create:

- Emulates a request that contains a credit card number beginning with `'9000'` to test the message flow path through the Mediator component to the CreditCardDB service
- Emulates a `VALID` response being received as a reply from the CreditCardDB service. The response is emulated because the `BCA_CREDITCARDS` table in the database does not have any credit cards that begin with `'9000'`.

Assumptions

This practice assumes that you successfully created and deployed the CCValidate project in Practice 5 for the lesson titled “Using JMS and JDBC Adapters.”

Tasks

1. In the JDeveloper window, ensure that the Basics name is selected from the Application drop-down menu, and that the CCValidate project and its SOA Content folder are expanded.
2. Create the test suite with its first test case.
   a. In the Application Navigator, right-click CCValidate > SOA > testsuites and select Create Test Suite.
   b. Name the suite `testsuite_vcc`.
   c. Use the information in the following table to define the test:

Step	Pane	Action
1	Name	Name: `test_valid_9000` Click Next.
2	Service and Operation	Click Next.

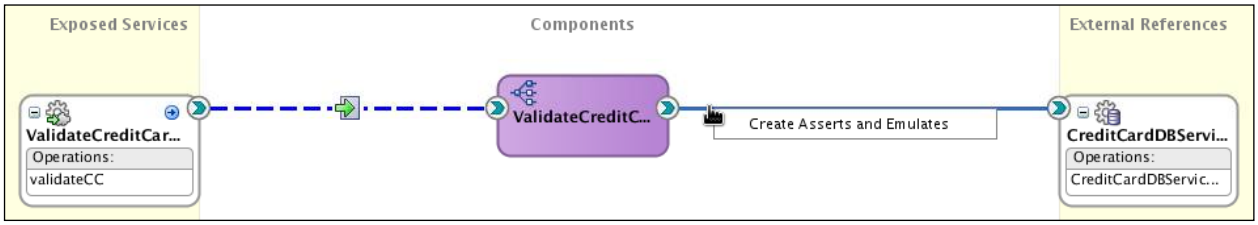
3	Input Message	Click Generate Sample.
		Replace the generated data:
		`<CCNumber>: 9000-1234-1234-1234`
		`<amount>: 200`
		Click Next.

4	Output Message	Click Generate Sample. Replace the generated data: <status>: VALID Click Finish.

The `test_valid_9000.xml` file is opened in the test case editor window, which resembles the composite assembly model, except that the Exposed Services and External References columns are in a different color.

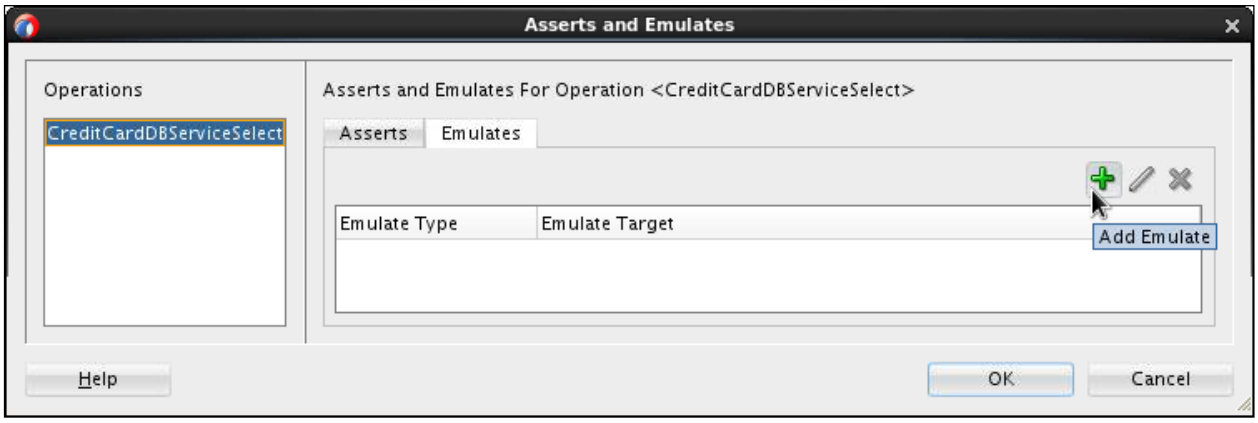
3. Create the emulated output of `VALID` from the CreditCardDB service for the `test_valid_9000` test case.

   a. Right-click the wire that connects the ValidateCreditCard Mediator component with the CreditCardDB service and select Create Asserts and Emulates.



   The Asserts and Emulates window opens.

   b. Click the Emulates tab and click the Add Emulate icon.



   The Create Emulate window opens. Observe that the Emulate Output option is selected.

   c. Generate and modify the message contents.

      1) Click Generate Sample.

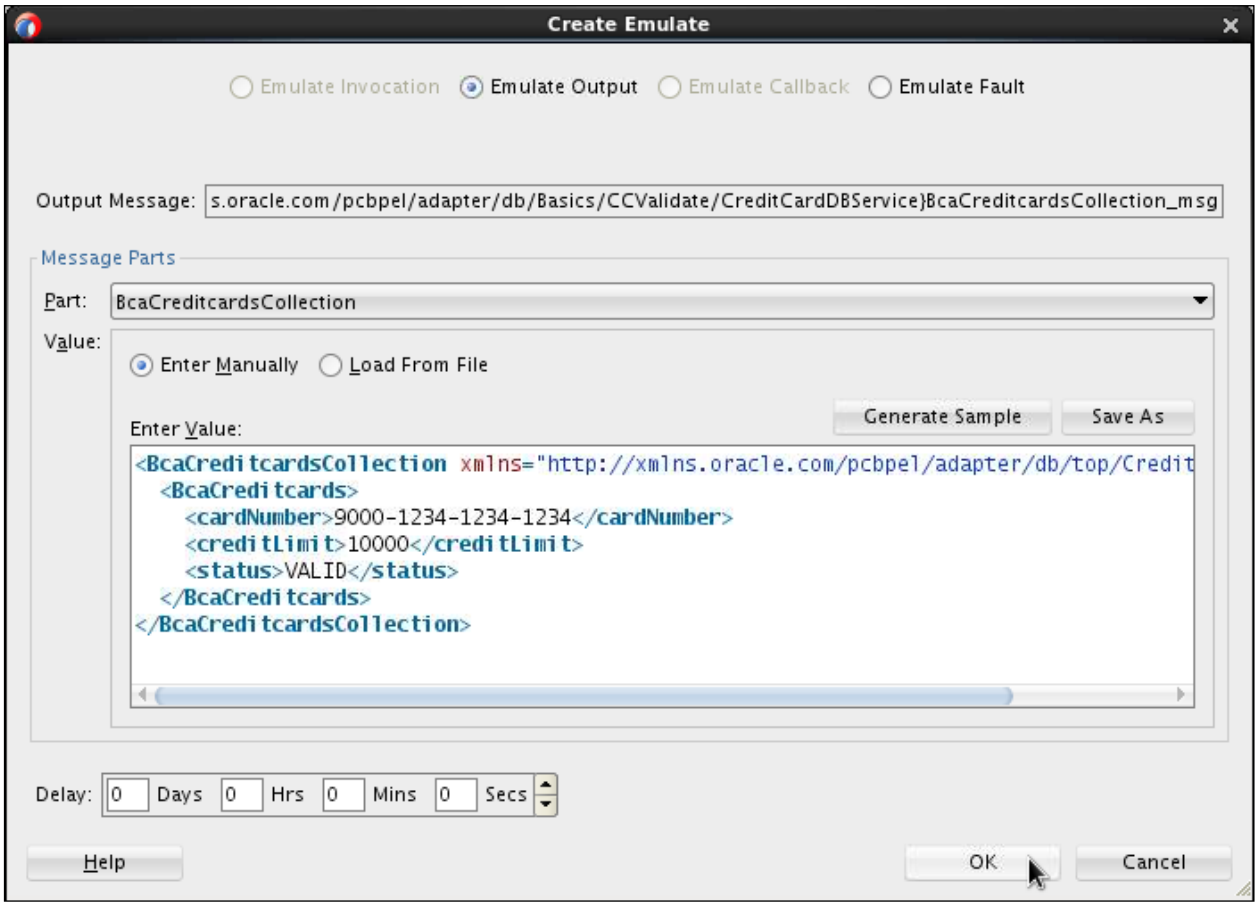
         The Enter Value section is populated with a generated message. The initial generated XML sample contains three <`BcaCreditcards`> elements with their associated child elements.

      2) Delete the last two <`BcaCreditcards`> elements and the child elements they contain.

      3) In the remaining <`BcaCreditcards`> element and its child elements, replace the data as follows:

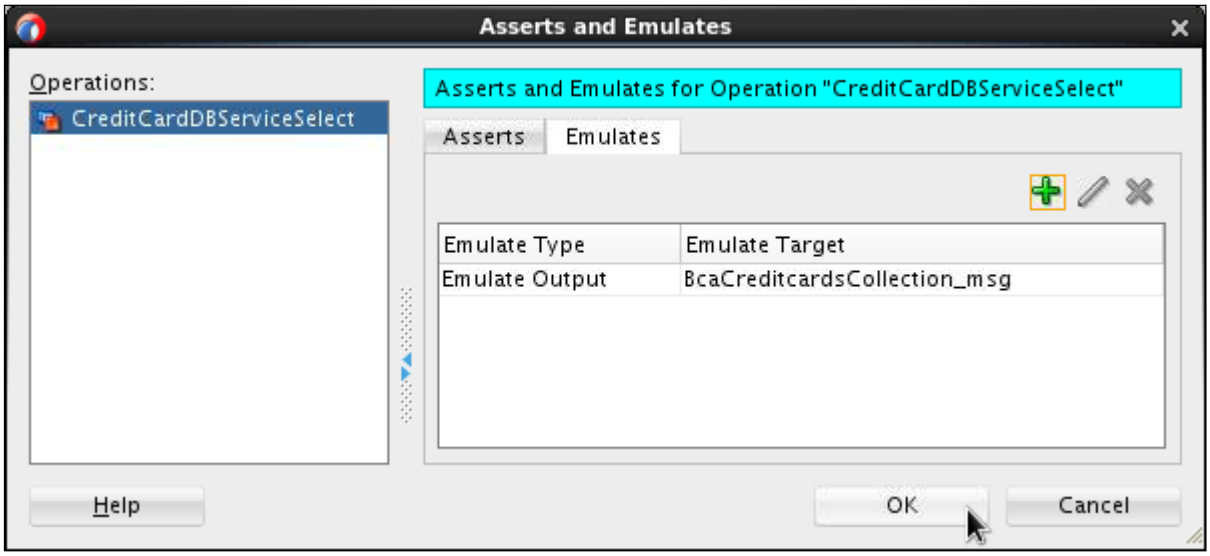
         - `<cardNumber>: 9000-1234-1234-1234`
         - `<creditLimit>: 10000`
         - `<status>: VALID`

4) Verify your work and click OK to create the emulated message.



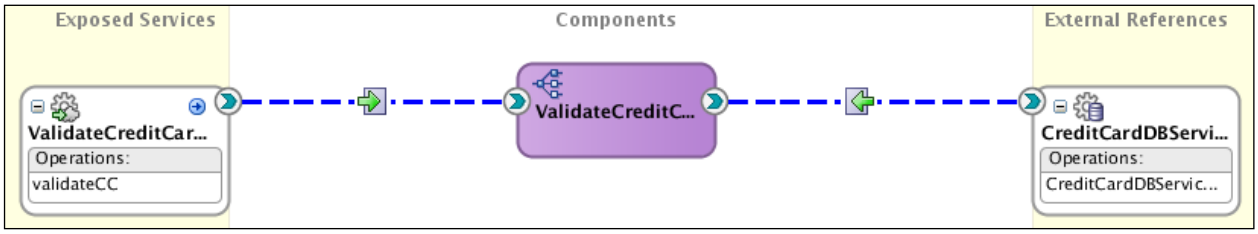
You are returned to the Asserts and Emulates window.

d. Confirm the addition of the emulated output (response) message and click OK.



You are returned to the `test_valid_9000.xml` window.

e. Verify that the emulated output message is added with the presence of a grey box icon containing a green arrow that points to the left. In addition, the wire between the Mediator component and its target service is displayed as a dotted line.



4. Save your work and close the `test_valid_9000.xml` window.

Practice 14-2: Creating Test Cases in the CCValidate Test Suite

Overview

In this practice, you create two more test cases for the `testsuite_vcc` test suite that was created in Practice 14-1. In this test suite, the initiating message data is slightly different and the emulated output (response) from the CreditCardDB service is `INVALID`. The final test case uses an initiating message to test the message path flow through the application, and uses assertions to check whether the actual response that is returned to the initiator is an expected value of `VALID` or `INVALID`.

Assumptions

This practice assumes that you have completed Practice 14-1.

Tasks

1. Create the new test.
   a. In the JDeveloper Application Navigator, right-click the testsuites > testsuit_vcc > tests folder and select Create Test.
   b. Use the information in the following table to define the test:

Step	Pane	Action
1	Name	Name: `test_invalid_9000` Click Next.
2	Service and Operation	Click Next.

3	Input Message	Click Generate Sample.
		Replace the generated data:
		`<CCNumber>: 9000-1111-2222-3333`
		`<amount>: 2500`
		Click Next.

4	Output Message	Click Generate Sample. Replace the generated data: <status>: `INVALID` Click Finish.

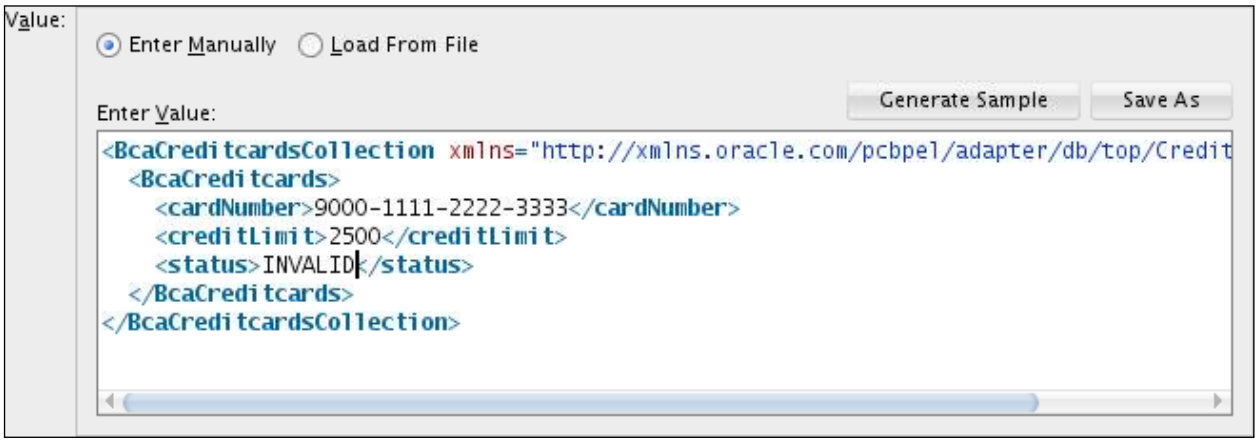
2. Create the emulated output wire action for the `INVALID` response.
   a. Right-click the wire connecting the ValidateCreditCard Mediator component with the CreditCardDB service and select Create Asserts and Emulates.
   b. Click the Emulates tab and click the Add Emulate icon.
   c. Click Generate Sample.

d. Delete the last two `<BcaCreditcards>` elements and the child elements they contain.

e. In the remaining `<BcaCreditcards>` element and its child elements, replace the data as follows:

- `<cardNumber>: 9000-1111-2222-3333`
- `<creditLimit>: 2500`
- `<status>: INVALID`

f. Verify your work and click OK.



You are returned to the Asserts and Emulates window.

g. Click OK.

h. Save your work and close the `test_invalid_9000.xml` window.

3. Create a third test case with assertions in the `testsuite_vcc` test suite.

a. Right-click the `tests` folder and select Create Test.

b. Use the information in the following table to define the test:

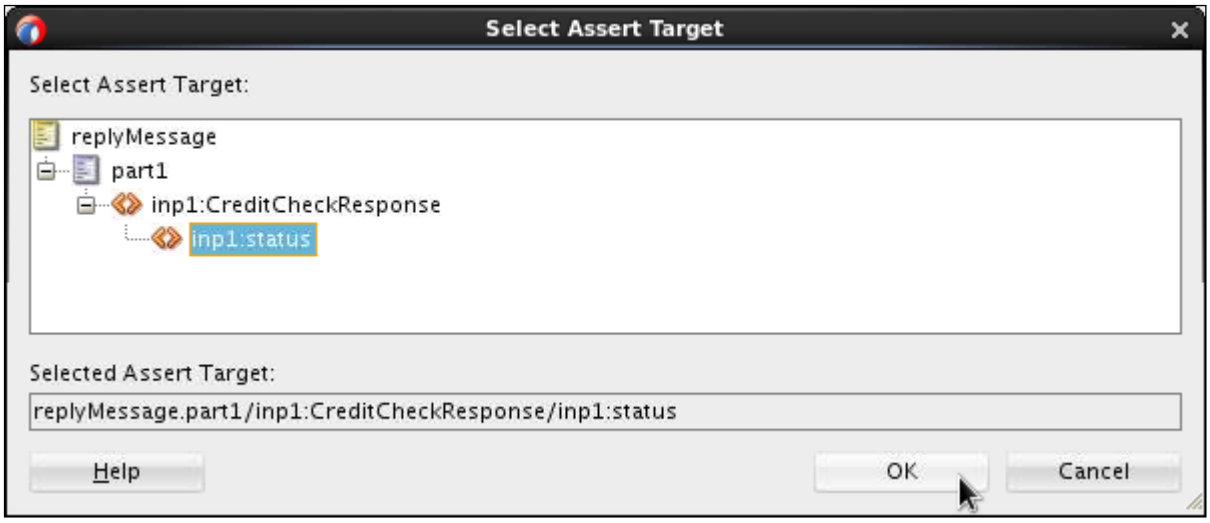
Step	Pane	Action
1	Name	Name: `test_response_9999` Click Next.
2	Service and Operation	Click Next.

3	Input Message	Click Generate Sample.
		Replace the generated data:
		`<CCNumber>: 9999-9999-9999-9999`
		`<amount>: 3500`
		Click Next.

4	Output Message	Click Generate Sample. Replace the generated data: `<status>: VALID` Click Finish.

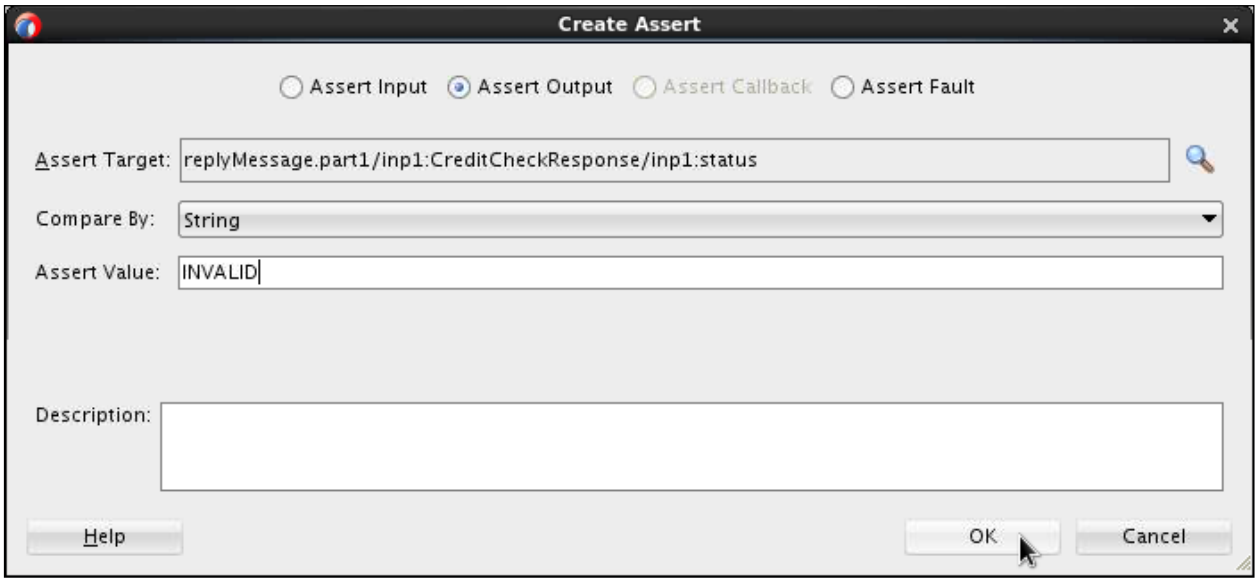
Creating a Value-Based Assertion

4. Create a value-based assertion for a possible `INVALID` response from the ValidateCreditCard Mediator component.
   a. Right-click the wire connecting the ValidateCreditCard_ep Exposed Services icon and the ValidateCreditCard Mediator and select Edit Asserts and Emulates.
   b. Click the Asserts tab.
   c. Click the Add Assert icon.
   d. Select the Assert Output option.
   e. Assert Target: Click Browse.
   f. Expand the message hierarchy, select `status`, and click OK.



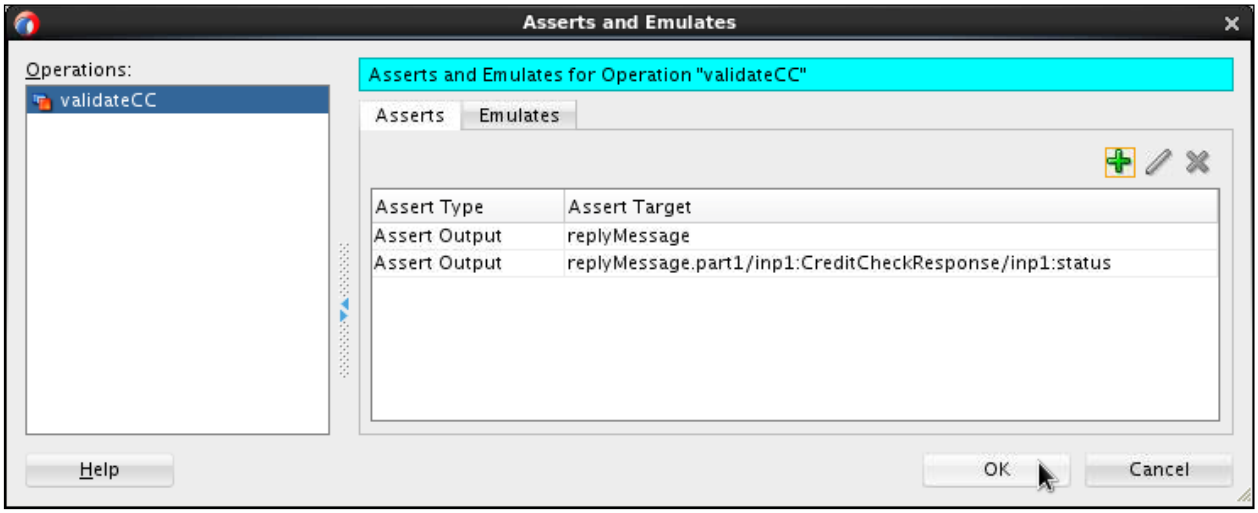
   g. With the Assert Output option and Assert Target selected, set the following values:
      1) Compare By: String
      2) Assert Value: `INVALID`

h. Verify your work and click OK.

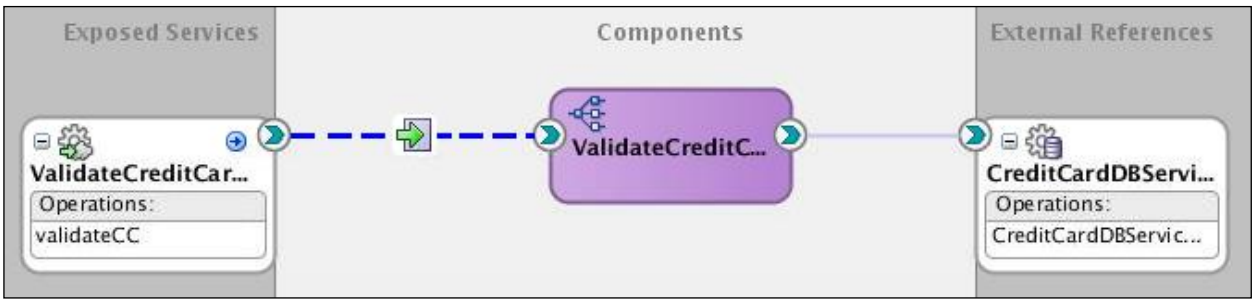


You are returned to the Asserts and Emulates window.

i. On the Asserts tabbed page, confirm that you have two assertions of the Assert Output type, and click OK.



5. On the `test_response_9999.xml` tabbed page, verify the creation of the wire action.



6. Save your work and close the `test_response_9999.xml` window.

Practice 14-3: Deploying and Executing the CCValidate Test Suite

Overview

In this practice, you deploy the CCValidate composite application along with its test suite and test cases. When it is deployed, by using Oracle Enterprise Manager Fusion Middleware Control, you can initiate and examine the results of the test suite and test cases.

Assumptions

This practice assumes that you have successfully completed all work up to this point.

Tasks

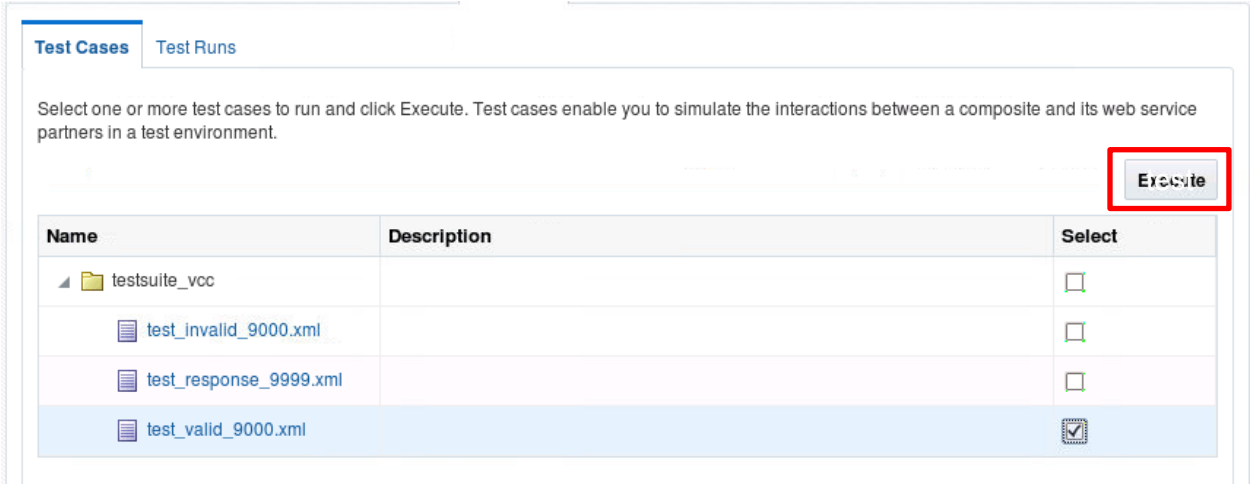
Redeploying the Composite Application

1. In JDeveloper, redeploy the CCValidate composite application to IntegratedWebLogicServer.

Accessing the Unit Test Window in Enterprise Manager

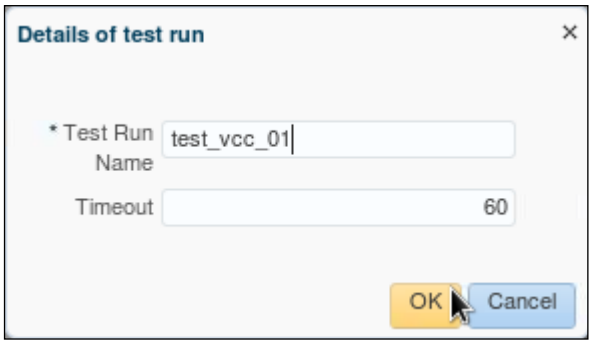
2. Open the Oracle Enterprise Manager page (http://localhost:7101/em).
   a. In the Target Navigation pane, expand the SOA node and right-click soa-infra and select Home > Deployed Composites. Click the “CCValidate [1.0]” link.
   b. On the “CCValidate [1.0]” page, click the Unit Tests tab.

3. Run the `test_valid_9000` test case.
   a. Expand the `testsuite_vcc` entry.
   b. Select the check box next to the row with the `test_valid_9000.xml` test case.
   c. Click Execute.



d. In the “Details of test run” window, enter the Test Run Name value: `test_vcc_01`.

Note: It is a good idea to choose unique names for your test runs. The test run name that is selected can be used in the test runs search criteria to locate the results for that test run instance.



e. Click OK.

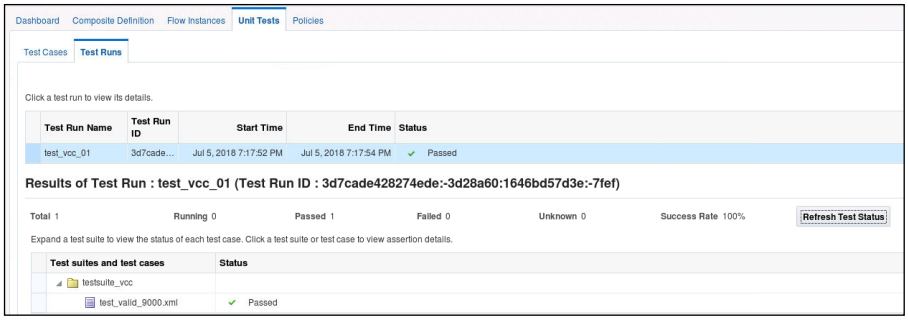
The web browser is refreshed and the Test Runs tabbed page is displayed.

f. Collapse the Search section to hide the search criteria entry fields.

g. In the “Results of Test Run: test_ccv_01” section, select the test_vcc_01 run and click Refresh Test Status.

Note: You may need to click Refresh Test Status several times before the test run results begin to appear on the page.

After a short wait and some clicks of the Refresh Test Status button, the test_vcc_01 test row appears in the Test Run Name column below the collapsed Search section. The `test_valid_9000.xml` entry is displayed. The Status column contains a green check mark with a `Passed` result.



Note: The `Passed` status indicates that the test was successful for the conditions that were created. In other words, the routing rule in the Mediator component functions as it was designed to work.

4. Run all the tests in the test suite.

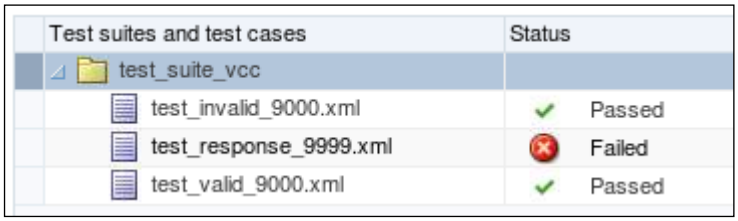
a. Click the Test Cases tab to return to the Test Cases tabbed page.

b. Select the entry in testsuite_vcc row, and click Execute.

Note: Selecting the test suite entry causes all the test cases to be executed.

c. In the “Details of test run” window, enter the Test Run Name value: `test_vcc_all`.

d. Wait for a short time (a couple of minutes) and click Refresh Test Status until the `test_vcc_all` test run entry appears.



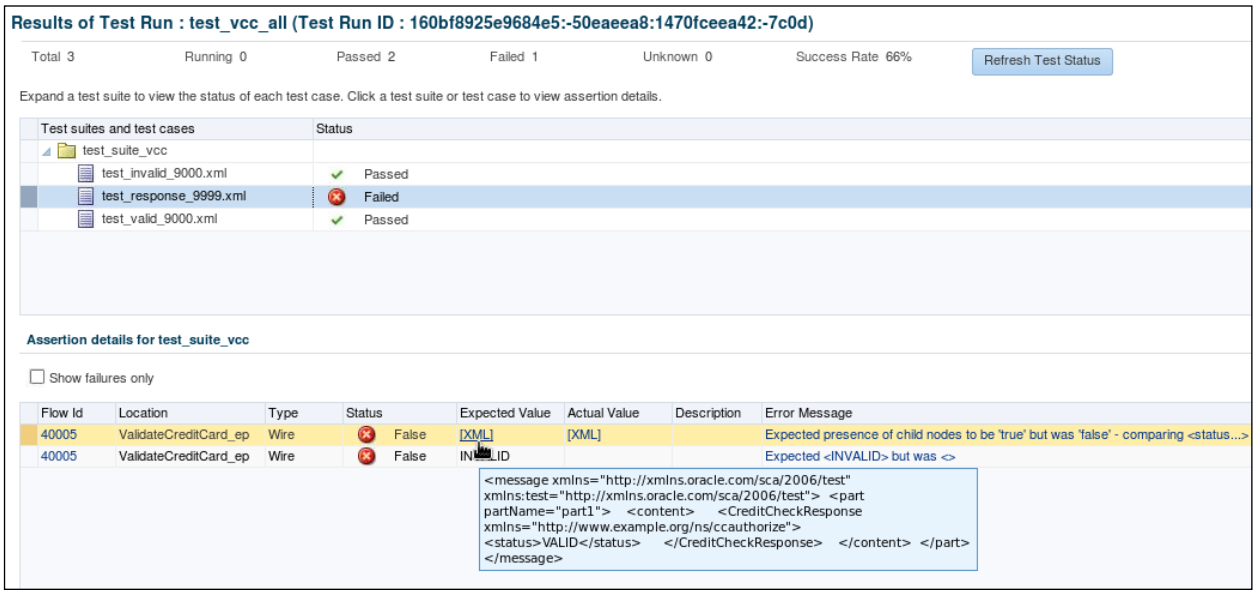
In this test run, all tests have a `Passed` status except the `test_response_9999.xml` test case.

e. Click the `test_response_9999.xml` row to display more information about the failure in the "Assertion details for testsuite_vcc" section.

Note: Why are the failures assertion failures? Recall that the `test-response_9999.xml` test case is the only test case that contains assertions. It appears that an assertion failed. This could indicate a process execution problem or data problem.

f. In the "Assertion details for testsuite_vcc" section, examine the cause for the XML-based assertion failure by clicking either of the [XML] links in the row entry that appears. Click the [XML] link in the Expected Value column.

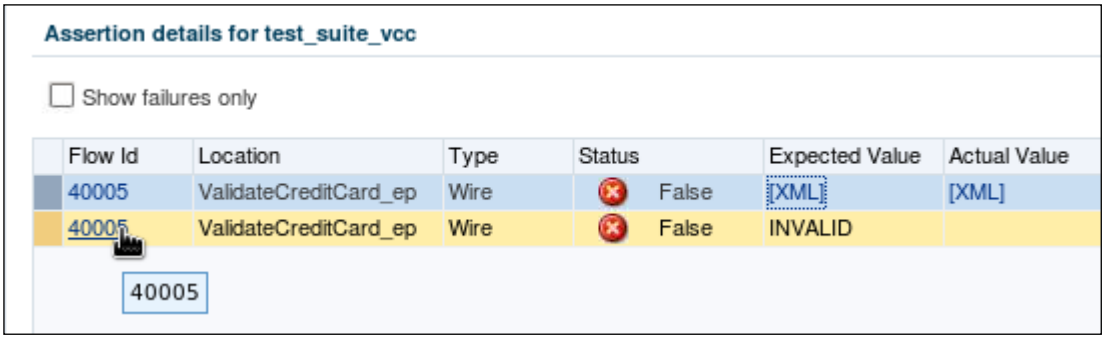
Note: The value-based assertion also failed because it expected the value `VALID` when the actual value return is an empty string. This is visible in the table row.



g. On the Compare XML Values page, for the XML-based assertion details, examine the actual XML value returned and compare it to the expected value. This enables you to identify the reason for the test failure and determine the next action to take, such as examining the process flow and logic to locate the cause of the problem. In this case, the assertion failed because the status element is an empty element when the value `VALID` was expected inside the status element.

Note: The actual cause of this problem is built in to the course application scenario. In this case, the cause is that the CreditCardDB service returns an empty status value if the credit card value does not exist in the database table. The choices of solution are varied and depend on the business requirements. You do not take action to repair this problem in the course practices.

5. In addition, for each test case, a composite instance is created and run as would be the case for an instance that is created from receiving an actual message (rather than an emulated test) on initiation. In the “Assertion details” table, you can click the instance ID link in the Flow Id column to view the Flow Trace for that composite application instance to examine the process audit trail and flow pages.



6. (Optional) Click the Composite Instance link to view the Flow Trace details.

Practice 14-4: Debugging a Composite Application in JDeveloper

Overview

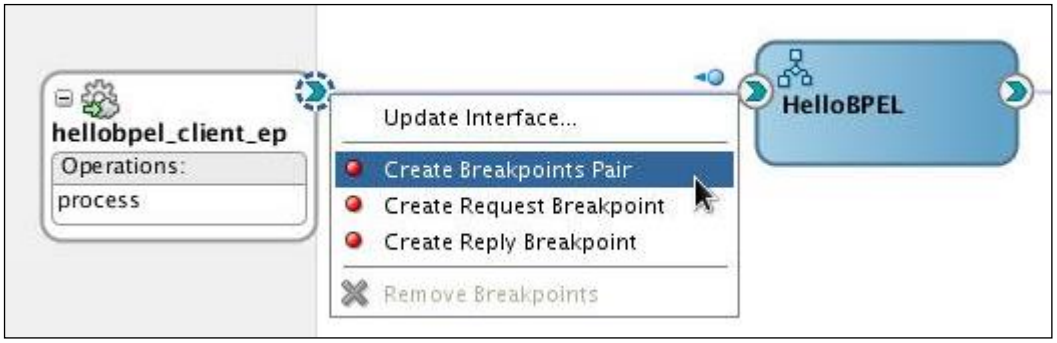
In this practice, you use the SOA debugger to step through a running application.

Assumptions

This practice assumes that you successfully created and deployed the project HelloBPEL in Practice 6 for the lesson titled “Introduction to BPEL.”

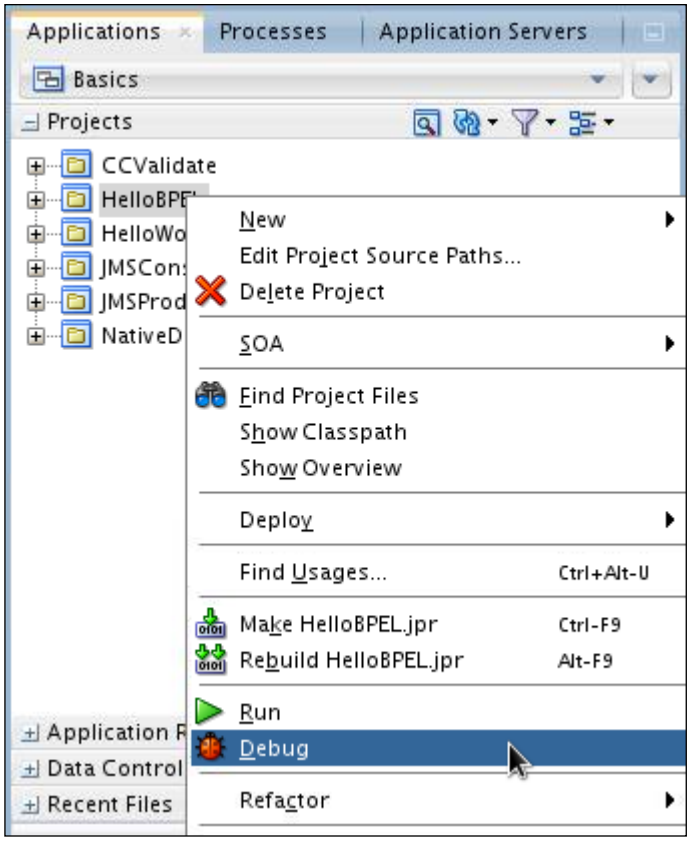
Tasks

1. In the JDeveloper window, ensure that the BPELProjects name is selected from the Application drop-down menu, and that the HelloBPEL project is expanded.
2. Open the HelloBPEL `composite.xml` file for editing.
3. Add breakpoint pairs to the interface and the BPEL process.
   a. Right-click the interface (blue arrow) icon and select Create Breakpoints Pair.

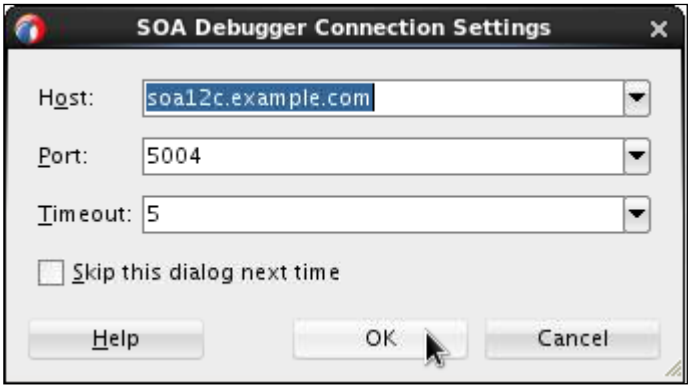


   b. Create a second breakpoints pair on the BPEL process.

4. In the Application Navigator pane, right-click the HelloBPEL project and select Debug.

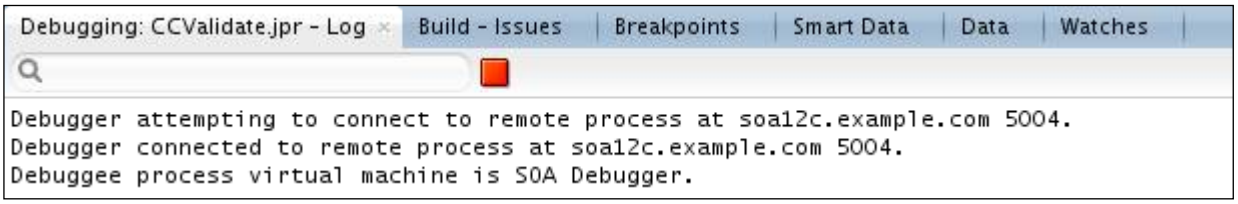


5. Click OK to accept the default connection settings.

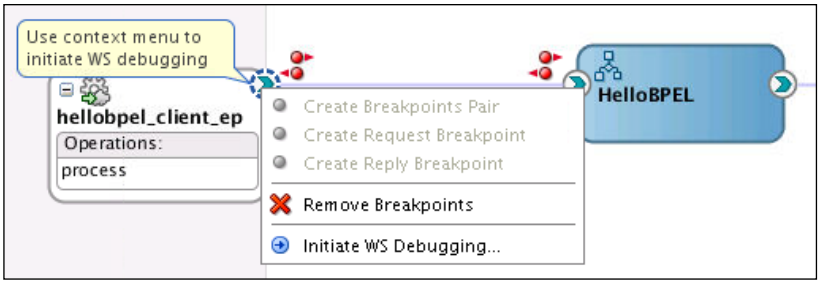


The Deploy HelloBPEL dialog box is displayed.

6. Deploy the application per usual. When the Information dialg appears, eread the contents and check the “Skip This Message Next Time” check box and click OK.
7. After deployment finishes, verify that the debugger is successfully connected.



8. Right-click the interface icon and select Initiate WS Debugging.

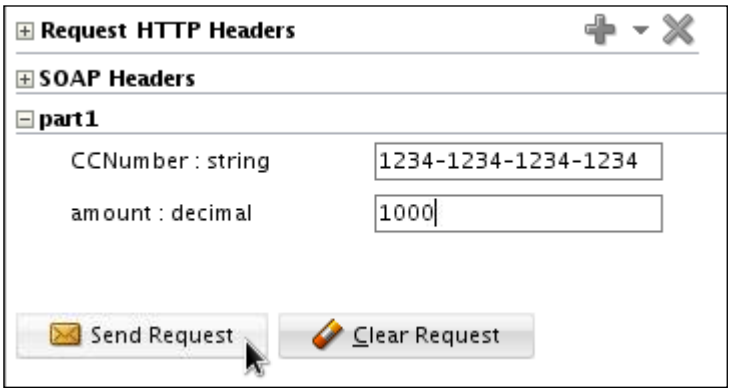


The HTTP Analyzer is opened.

9. Enter the following values:

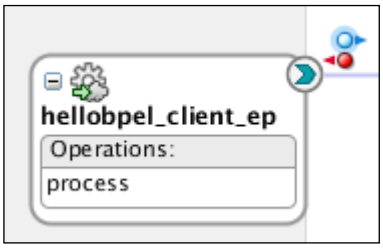
CCNumber: `1234-1234-1234-1234`

amount: `1000`

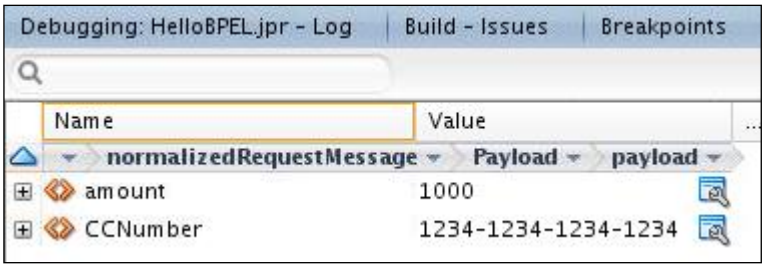


10. Click Send Request.

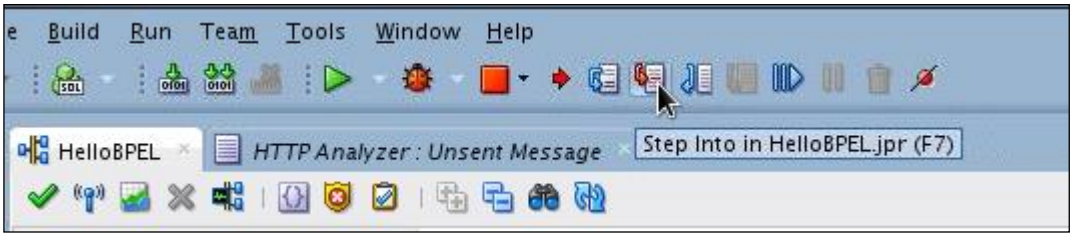
The debugger stops at the first breakpoint (which turns blue and starts pulsing).



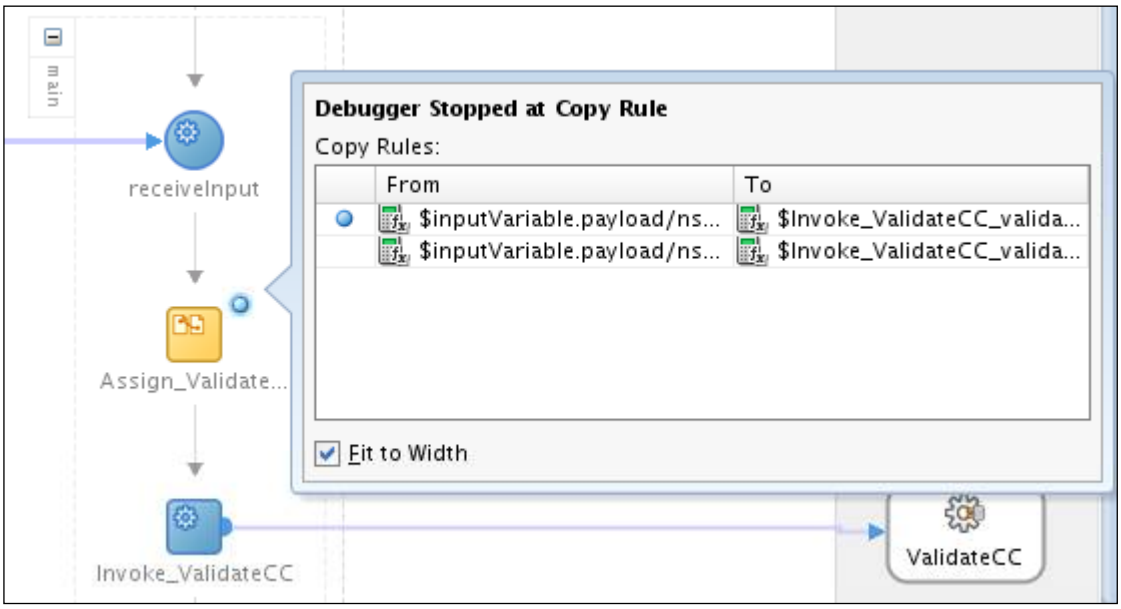
11. Click the Data tab and examine the incoming (normalizedRequestMessage payload) values.



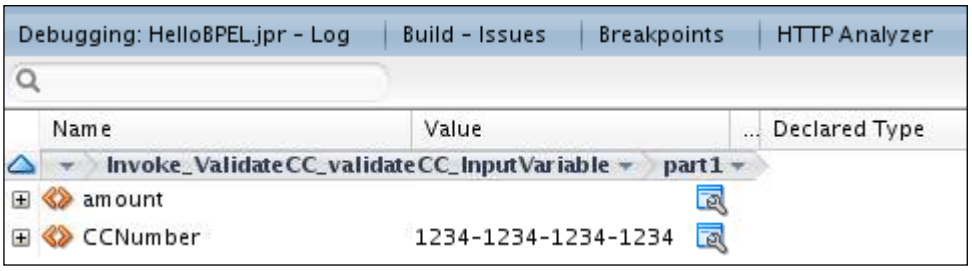
12. On the JDeveloper main menu, click Step Into (F7).



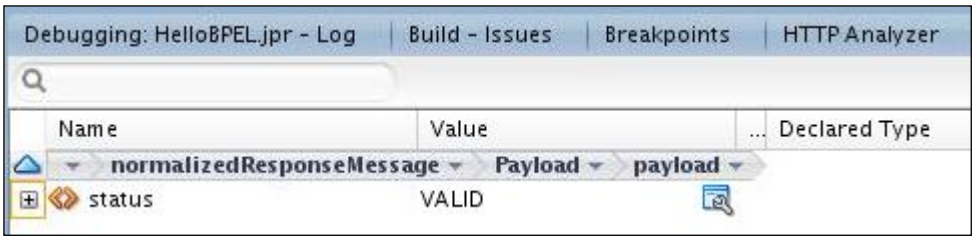
13. Repeat clicking Step Into until the debugger arrives at the first Assign activity and the Debugger Stopped dialog appears.



14. Press F7 again to observe the progress of the individual copy statements in the Assign activity. Expand the variables in the data window as needed.



15. Continue to use the F7 key to move forward through the BPEL process. After exiting the BPEL process and returning to the hellobpel_client_ep interface, pause to observe the value of the response message.



16. On completion of the composite instance, on the main JDeveloper menu, click the Terminate icon, and select to terminate the debugger session.

Practices for Lesson 15: Securing Composite Applications and Invoking Secured Services

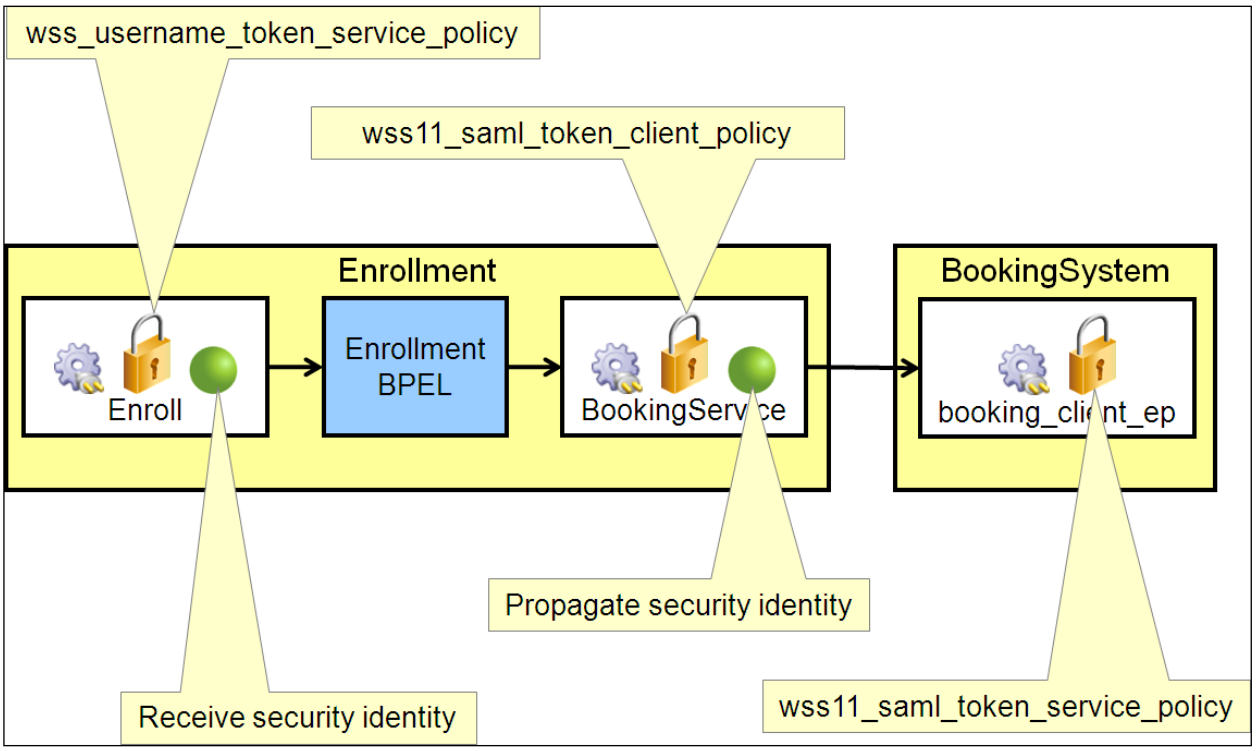
Practices for Lesson 15: Overview

Practices Overview

The goal of this practice is to apply security policies to web services endpoints. Applying security policies is done in two ways:

- Post-deployment, by using Oracle Enterprise Manager Fusion Middleware Control to attach a security policy to selected service endpoints
- Before deployment (at design time), by attaching policies to service endpoints in JDeveloper

You apply policies both at design time and post-deployment to the `Enroll` and `BookingSystem` applications that you created earlier. The following image illustrates the security policy attachments that are applied in this practice:



Your tasks in the practice are to:

- Secure the invocation of the `Enroll` composite application entry point (`Enroll`) by using the UsernameToken policy. This requires that you enter a valid set of credentials (username and password) to initiate the service.
- Secure the `BookingSystem` composite application entry point (`Booking_client_ep`) by using SAML service tokens, which requires the security identity to be propagated
- Propagate identity by using SAML client tokens on the ValidCCService, which is the external reference for the `Booking_client_ep` entry point

In the practice, you apply the policies to the deployed applications by using Oracle Enterprise Manager. After removing the policies in Oracle Enterprise Manager, you use JDeveloper to secure the `BookingSystem` service and ensure that it is invoked securely by using design-time policy attachments. You then remove the design-time policies so that the remaining practices in the course do not require the additional step of providing WS-Security credentials for each test.

Practice 15-1: Applying Security Policies Post Deployment

Overview

In this practice, you attach the WS-Security UsernameToken policy to the `Enroll` service entry point of the `Enroll` composite application. You test the service with and without supplying security credentials. You then attach SAML policies to secure the `BookingSystem` composite application entry point and its client to propagate the security identity that is acquired through the use of the UsernameToken policy. You test to see if execution is successful. Finally, you remove the UsernameToken policy and test to observe what happens if you try to propagate security credentials that have not been supplied.

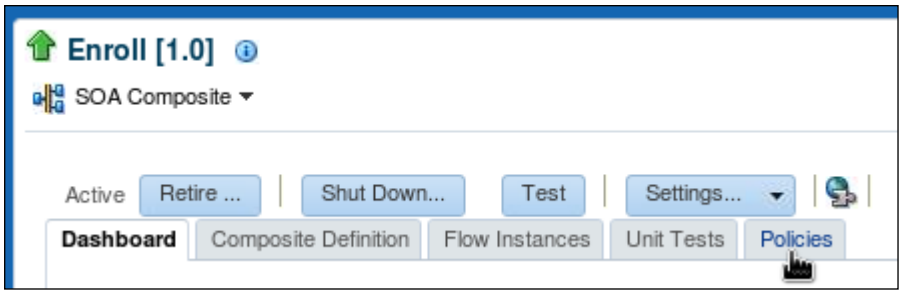
Assumptions

This practice assumes that you have completed the practice for lesson 8 titled “Handling Faults in Composite Applications” successfully and that the `Enroll` and `BookingSystem` applications are still deployed.

Tasks

Attaching the UsernameToken Policy to `Enroll`

1. Access the policy page for the `Enroll` composite application entry point called `Enroll`.
   a. Open the Oracle Enterprise Manager page (http://localhost:7101/em).
   b. In the Target Navigation pane, expand the SOA > soa-infra > node in the tree and select Home > Deployed Composites.
   c. Click the “Enroll [1.0]” link.
   d. On the “Enroll [1.0]” page, click the Policies tab.



2. Attach `oracle/wss_username_token_service_policy` to the `Enroll` component:
   a. Select `Enroll` from the “Attach To/Detach From” drop-down menu.
   b. In the Search field above the Name column, enter `wss_username` and press Enter.

      Note: Using the search feature makes it easier to locate the desired policy.
   c. In the Available Policies section, select the row containing the Name `oracle/wss_username_token_service_policy`.
   d. Click Attach.

e. Verify your work and click OK.

The `Enroll` application now requires a username and password.

Testing the WSS UsernameToken Security Policy

3. Perform a test of the attached security policy.
   a. On the “Enroll [1.0]” page, click Test.
   b. On the Test Web Service page, click the Request tab. In the Input Arguments section, use the Browse button to replace the supplied XML text with the contents of the file `/home/oracle/labs/files/xml_in/enrollment_input.xml`.

   Note: If the Test page displays an error when attempting to open the enrollment_input.xml file, then open the file using an editor like gedit and copy and paste the contents into the Request XML View window and proceed with testing.

   c. Click Test Web Service.
   d. What is the result of this test?

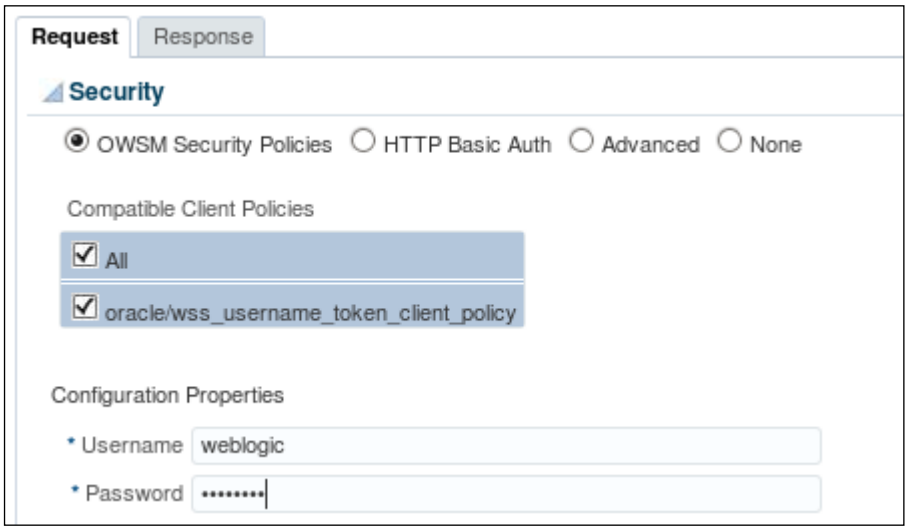
      Answer: The “Webservice invocation failed” window is displayed.

   e. In the “Webservice invocation failed” window, expand Show Additional Trace Information. What does it tell you?

The text stating “`error in processing the WS-Security security header`” hints at the answer. You did not provide the WS-Security information (a valid username and password), which is sent in the request header.

   f. Close the “Webservice invocation failed” window.

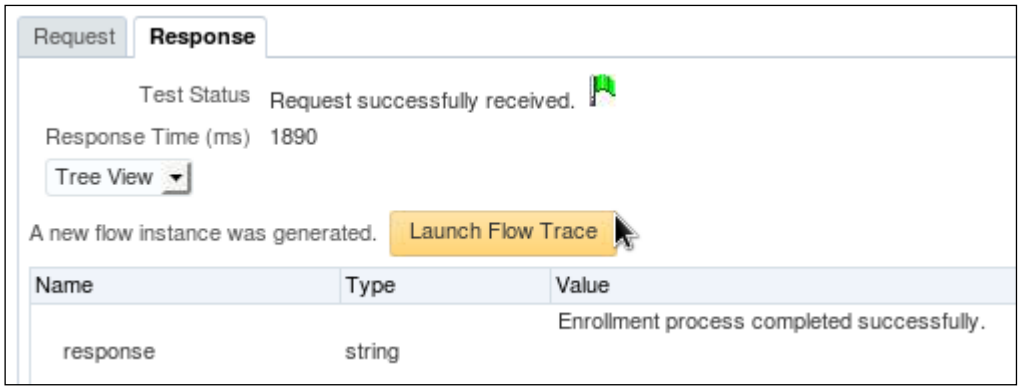
4. Execute another test. This time, provide the WS-Security credentials for the WSS UsernameToken style of security.
   a. On the Test Web Service page, expand the Security section at the top of the Request tab area.
   b. Select the OWSM Security Policies option.
   c. In the Compatible Client Policies section, select the oracle/wss_username_token_client_policy check box.
   d. Enter `weblogic` in the Username field and `welcome1` in the Password field.



   e. Click Test Web Service.

      Note: The XML View text should have retained the data copied from the `/home/oracle/labs/files/xml_in/enrollment_input.xml` file for this test.
   f. What happened this time?
      1) Did you get an error message?

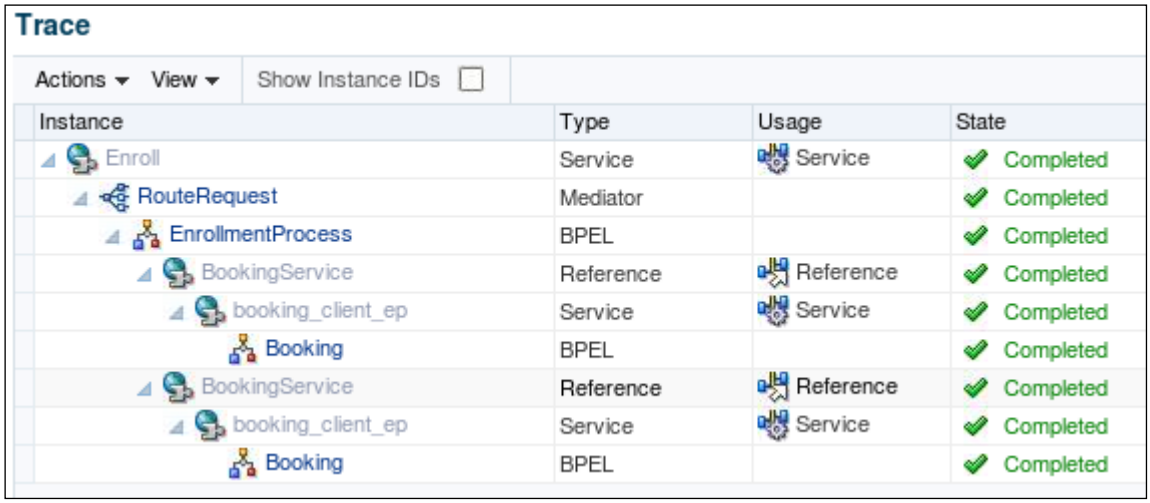
         Answer: No error message is displayed. The `Enroll` composite instance is initiated successfully, as indicated by the appearance of the Launch Flow Trace button on the Response tabbed page.



      2) Did the process complete successfully based on the supplied data?

         Answer: Yes

3) Click the Launch Flow Trace button. Confirm that the `BookingService` application, which has not been secured yet, was invoked successfully as well.

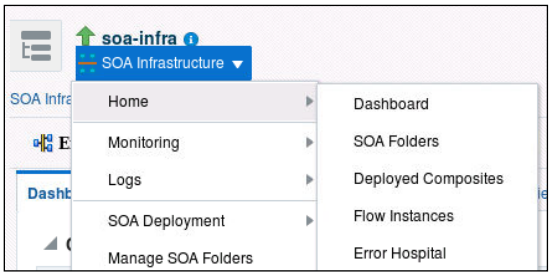


4) Close the Flow Trace window.

Attaching the SAML Service Policy to `Booking_client_ep`

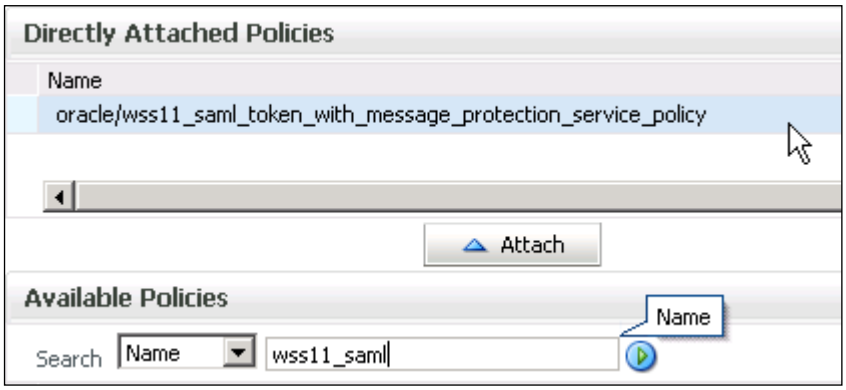
In this section, you attach the SAML service security policy to the `Booking_client_ep` entry point of the `BookingSystem` composite application.

5. Access the policy page for the `BookingSystem` composite application entry point called `Booking_client_ep`.
   a. Select SOA Infrastructure > Home > Deployed Composites.



   b. Click the “BookingSystem [1.0]” link.
   c. Click the Policies tab.
6. Attach `oracle/wss11_saml_token_with_message_protection_service_policy` to `Booking_client_ep`.
   a. Click “Attach To/Detach From” and click booking_client_ep.
   b. Enter `wss11_saml` in the Search field above the Name column and press Enter.
   c. In the Available Policies section, select the row containing the name `oracle/wss11_saml20_token_with_message_protection_service_policy`
   d. Click Attach.

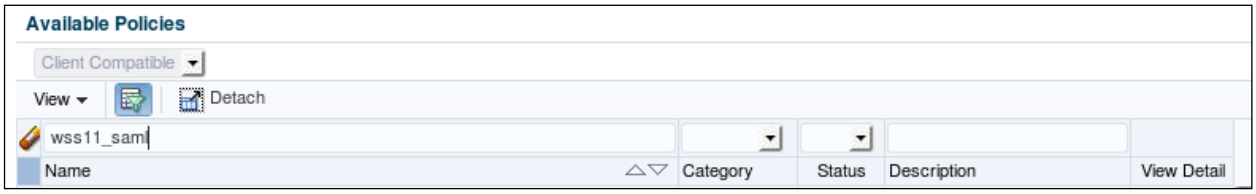
e. Verify your work and click OK.



Attaching the SAML Client Policy to `BookingService`

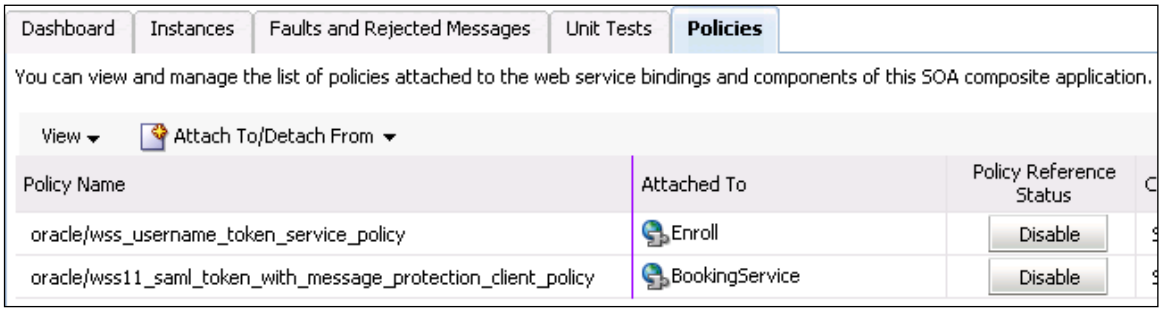
In this section, you configure the SAML client security policy on the `BookingService` external reference of the `Enroll` composite application. The SAML client security policy ensures that the security credentials, which are received through the WS-Security heading (from the attached UsernameToken security policy requirement), are propagated to the `booking_client_ep` entry point. Remember that the `booking_client_ep` entry point is accessed through the `BookingService` external reference, which is invoked by the `Enrollment` BPEL Process component.

7. Access the `Enroll` composite application policies page.
   a. Select SOA Infrastructure > Home > Deployed Composites.
   b. Click the “Enroll [1.0]” link.
   c. On the “Enroll [1.0]”page, click the Policies tab.
8. Attach `oracle/ wss11_saml_token_with_message_protection_client_policy` (the SAML client policy) to the `BookingService` external reference in the `Enroll` composite application.
   a. Click “Attach To/Detach From,” and then click `BookingService`.
   b. In the Available Policies section, enter `wss11_saml` in the Name search field and press Enter.



   c. In the Available Policies section, select the row containing the name `oracle/wss11_saml20_token_with_message_protection_client_policy`.
   d. Click Attach.
   e. The policy is listed in the Directly Attached Policies section.
   f. Click OK.

g. Confirm that the two policies, `oracle/wss_username_token_service_policy` and the newly attached `oracle/wss11_saml20_token_with_message_protection_client_policy`, are listed in the policy table.



Testing Security Propagation with the UsernameToken and SAML Policies

In this section, you initiate a test of the `Enroll` composite application and provide security credentials to test if the process works, indicating that the security credentials are successfully propagated. To verify this, you perform another test after disabling the UsernameToken security policy to observe what happens when the credentials are not enforced and therefore not propagated.

9. Initiate a security propagation test of the `Enroll` composite application through to the `BookingSystem` composite application.
   a. On the “Enroll [1.0]” home page, click Test.
   b. On the Test Web Service page, expand the Security section at the top of the Request tab area.
   c. Select the OWSM Security Policies option.
   d. Select the oracle/wss_username_token_client_policy check box.
   e. Enter `weblogic` in the Username field and `welcome1` in the Password field.
   f. On the “Enroll [1.0]” Web Service Test page, scroll down to the Input Arguments section on the Request tab. Select the XML View mode and replace the supplied XML text with the contents of the `/home/oracle/labs/files/xml_in/enrollment_input.xml` file.
   g. Click Test Web Service.

      The Response tabbed page is displayed.

h. Click the Launch Message Flow Trace button. Confirm that all appropriate process components executed with a Completed status.

Trace

Actions ▾ View ▾ Show Instance IDs ☐

Instance	Type	Usage	State
Enroll	Service	Service	Completed
RouteRequest	Mediator		Completed
EnrollmentProcess	BPEL		Completed
BookingService	Reference	Reference	Completed
booking_client_ep	Service	Service	Completed
Booking	BPEL		Completed
BookingService	Reference	Reference	Completed
booking_client_ep	Service	Service	Completed
Booking	BPEL		Completed

i. Close the Flow Trace window.

10. Disable the WSS UsernameToken policy in the `Enroll` composite application.

Note: This eliminates the need to provide security credentials when initiating the `Enroll` composite application. No security identity information can be propagated to the `booking_client_ep` entry point of the `BookingSystem` composite application.

a. Select SOA Infrastructure > Home > Deployed Composites.

b. Click the “Enroll [1.0]” link.

c. On the “Enroll [1.0]” home page, click the Policies tab.

d. On the “Enroll [1.0]” Policies tabbed page, from the list of attached policies, select the `oracle/wss_username_token_service_policy` row and click Disable.

e. In the Confirmation dialog box, click Yes.

f. On the “Enroll [1.0]” page, click Test.

g. In the Request tab section, scroll down to the Input Arguments section, select the XML View mode, and replace the supplied XML text with the contents of the `/home/oracle/labs/files/xml_in/enrollment_input.xml` file.

Note: Because you disabled the policy, there is no need to provide the WS UsernameToken credentials.

h. Click Test Web Service.

The web service invocation fails. An error message indicating that the endpoint cannot be accessed is displayed. This result is expected because no credentials were provided to the `Enroll` composite application. Therefore, there is no authenticated security identity that can be propagated to the `BookingSystem` service entry point, which still enforces the requirement to provide security information because of the attached SAML security policies.

Webservice invocation failed

The selected operation execute could not be invoked.

An exception occurred while invoking the webservice operation. Please see logs for more details.

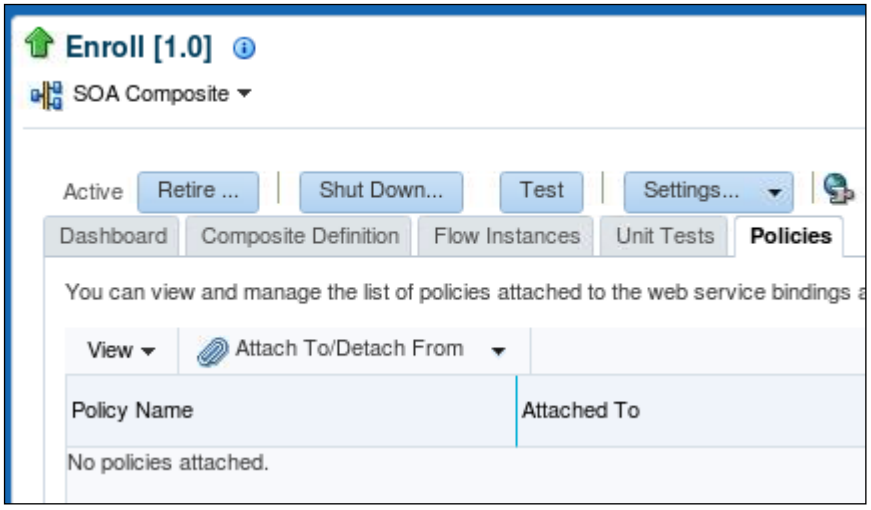
oracle.sysman.emInternalSDK.webservices.util.SoapTestException: Client received SOAP Fault from server : oracle.fabric.common.FabricInvocationException: Unable to invoke endpoint URI "http://soa12c.example.com:7101/soa-infra/services/default/BookingSystem!1.0*soa_97545006-24ad-4be4-ba3f-4afeab6062b7/booking_client_ep" successfully due to: oracle.fabric.common.PolicyEnforcementException: WSM-00237 : KSS keystore with URI kss://owsm/keystore can not be loaded due to java.security.PrivilegedActionException.

Detaching All Security Policies

In this section, you detach the security policies so that you can perform the same security attachment tasks at design time in the JDeveloper Composite Editor in the next practice.

11. Access the Policies tab of the `Enroll` composite application.
    a. Select SOA Infrastructure > Home > Deployed Composites.
    b. Click the “Enroll [1.0]” link.
    c. On the “Enroll [1.0]” page, click the Policies tab.
12. Detach the security policies from the `Enroll` composite application.
    a. Click “Attach To/Detach From” > `BookingService`.
    b. In Attached Policies, select the `oracle/wss11_saml20_token_with_message_protection_client_policy` entry.
    c. Click Detach.
    d. Verify that no policies are attached and click OK.
    e. Select “Attach To/Detach From” > Enroll.
    f. In Attached Policies, select the `oracle/wss_username_token_service_policy` entry.
    g. Click Detach.
    h. Verify that there are no policies attached and click OK.

i. On the “Enroll [1.0]” Policies page, verify that all security policies have been detached from the composite application.



13. Repeat the previous steps to access the Policies tab of the `BookingSystem` composite application and detach the policy `oracle/wss11_saml_token_with_message_protection_service_policy` from `Booking_client_ep`.
14. Verify your work.

Practice 15-2: Applying Security Policies at Design Time

Overview

In this practice, you effectively repeat the attachment of the policies used in the previous practice. However, this time you attach the policies to component endpoints at design time by using the Composite Editor. After deploying the secured composite applications, you initiate a test, and then disable the security and redeploy the applications so that the remaining course practices do not require security credentials to be supplied for every test that is subsequently performed.

Note: In JDeveloper, to attach policies to endpoints, right-click an exposed service, external references, or components in the Composite Editor and select Configure SOA WS Policies.

Assumptions

This practice assumes that you have completed Practice 12-1 titled “Creating and Configuring a Composite Application” successfully.

Tasks

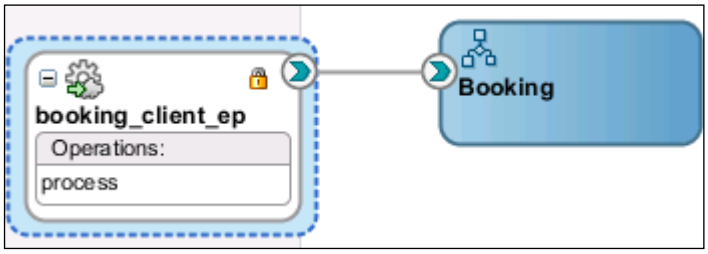
1. In JDeveloper, open the `BookingSystem composite.xml` application file.

Attaching the SAML Service Policy to `Booking_client_ep`

2. Attach `oracle/wss11_saml_token_with_message_protection_service_policy` to the `Booking_client_ep` exposed service.
   a. Right-click the `booking_client_ep` exposed service icon and select Configure SOA WS Policies.
   b. For the Security section, click the Add Security policies icon.
   c. Scroll down and select the `oracle/wss11_saml20_token_with_message_protection_service_policy` entry.
   d. Click OK.
   e. Confirm the addition of the `oracle/wss11_saml20_token_with_message_protection_service_policy` entry to the Security section and click OK.

Note: The green check mark next to the policy indicates that the policy is enabled.

In the Exposed Services column, the `Booking_client_ep` icon contains a lock icon to indicate that it has been secured.



3. Save your work and close the `BookingSystem composite.xml` window.

Attaching the SAML Client Policy to `BookingService`

In this section, you modify the `Enroll` composite application. You attach the appropriate security policies to propagate the SAML security to the `BookingSystem` composite from the `BookingService` external reference.

4. In JDeveloper, open the `Enroll` project composite overview window.
5. Attach `oracle/wss11_saml20_token_with_message_protection_client_policy` to the `BookingService` external reference.
   a. Right-click the `BookingService` external reference icon and select Configure SOA WS Policies.
   b. In the Security section, click the Add Security Policy icon.
   c. Scroll down and select the `oracle/wss11_saml20_token_with_message_protection_client_policy` entry.
   d. Click OK.
   e. Confirm the addition of the `oracle/wss11_saml20_token_with_message_protection_client_policy` entry to the Security section and click OK.

      The `BookingService` icon displays a lock icon to indicate that it has been secured.

Attaching the UsernameToken Policy to `Enroll`

In this section, you enforce the requirements for obtaining credentials for the `Enroll` composite by using the UsernameToken policy on its `Enroll` entry point. Credentials are required so that they can be propagated to the `BookingService`.

6. Attach `oracle/wss_username_token_service_policy` to the `Enroll` exposed service.
   a. Right-click the `Enroll` exposed service icon and select Configure SOA WS Policies.
   b. For the Security section, click the Add Security Policy icon.

c. Scroll down and select the `oracle/wss_username_token_service_policy` entry.

d. Click OK.

e. Confirm the addition of the `oracle/wss_username_token_service_policy` entry to the Security section and click OK.

   In the Exposed Services column, the `Enroll` icon contains a lock icon to indicate that it has been secured.

7. Save your work and close the `Enroll composite.xml` window.

Deploying `BookingSystem` and Deploying and Testing `Enroll`

8. Deploy the `BookingSystem` project with the security attachments to IntegratedWebLogicServer.
9. Deploy the `Enroll` project with the security attachments to IntegratedWebLogicServer.

Testing the Design-Time Security Settings in Oracle Enterprise Manager

10. Test if the UsernameToken security is enforced.

    a. On the Oracle Enterprise Manager home page, in the Target Navigation pane, expand the SOA > soa-infra > node and select Home > Deployed Composites node in the tree.

    b. Click the “Enroll [1.0]” link.

    c. On the “Enroll [1.0]” page, click Test.

       Do not specify the WS UsernameToken credentials.

    d. Scroll down to the Input Arguments section on the Request tab and select the XML View mode.

    e. Replace the supplied XML text with the contents of the `/home/oracle/labs/files/xml_in/enrollment_input.xml` file.

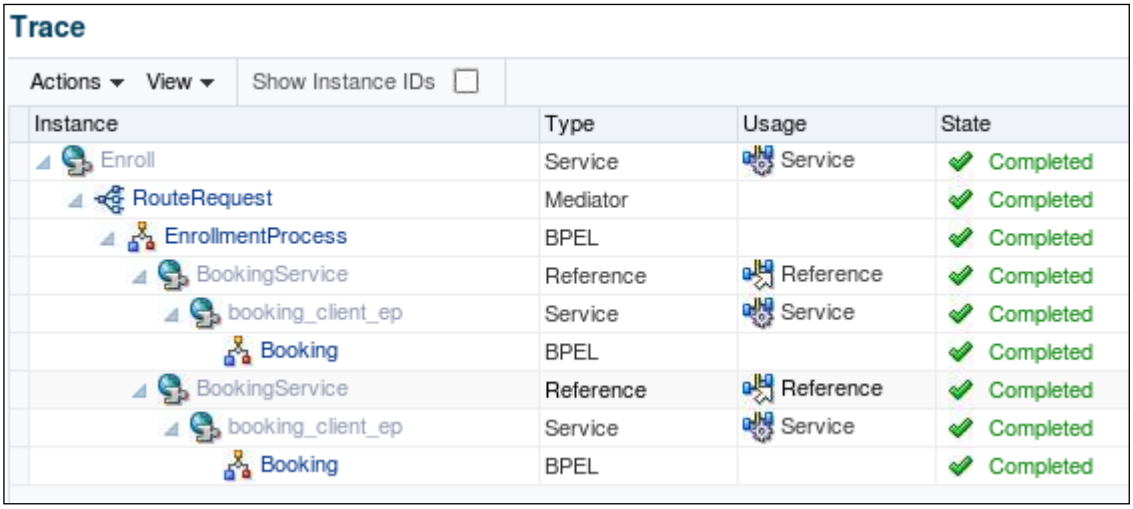
    f. Click Test Web Service.

       After a short delay, the “Webservice invocation failed” window is displayed. This indicates that the invocation of the `Enroll` application failed because security credentials were not provided when they were required.

11. Repeat the test. This time, supply a valid username and password in the WSS UsernameToken settings.

    This time, the “Enroll [1.0]” Test Web Service > Response tabbed page is displayed with the Launch Message Flow Trace link, indicating that the service invocation is successful.

12. On the “Enroll [1.0]” Test Web Service > Response tab, click the Launch Message Flow Trace link to verify that the process completed as expected, indicating that `BookingService` propagated the credentials to `Booking_client_ep` in the `BookingSystem` composite application.

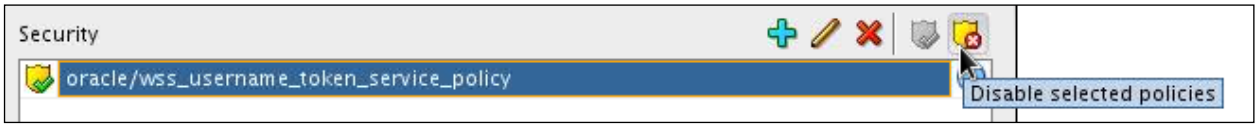


13. Close the web browser window that contains the Flow Trace page.

Disabling Security Policies and Redeploying Applications

14. Disable the UsernameToken security policies so that they are not enforced by the `Enroll` composite application.
    a. In JDeveloper, open the Enroll composite overview window.
    b. Right-click the Enroll icon and select Configure SOA WS Policies.

       The Configure SOA WS Policies window opens.
    c. Seelct the oracle/wss_username_token_service_Policy. Click the Disable icon for the `oracle/wss_username_token_service_policy` entry.
    d. Verify your work and click OK.



       This disables the security policy so that it is not enforced.
    e. In the `composite.xml` window, verify that the lock icon on the `Enroll` icon is displayed in an unlocked state.

15. Disable the SAML client security policy on the `BookingService` external reference of the Enroll composite application.
    a. Right-click the `BookingService` icon and select Configure SOA WS Policies.

b. Select `oracle/wss11_saml20_token_with_message_protection_client_policy` entry.
c. Click the Disable icon.
d. Click OK.
e. In the `composite.xml` window, verify that the lock icon on the `BookingService` icon is displayed in an unlocked state.

16. Save your work and close the `composite.xml` window.
17. Disable the SAML service security policy on the `BookingSystem` composite application exposed service (entry point).
    a. Open the `BookingSystem` composite overview window.
    b. Right-click the `booking_client_ep` icon and select Configure SOA WS Policies.
    c. Select `oracle/wss11_saml20_token_with_message_protection_service_policy` entry.
    d. Click the Disable icon.
    e. Click OK.
    f. In the `composite.xml` window, verify that the lock icon on the `booking_client_ep` icon is displayed in an unlocked state.
18. Save your work and close the `composite.xml` window.

Redeploying the Composite Applications with Security Policies Disabled

19. Deploy `BookingSystem` to the IntegratedWebLogicServer.
20. Deploy `Enroll` to the IntegratedWebLogicServer.
21. Return to Enterprise Manager. For each service, view its Policies tab. Notice that the policies are attached, but disabled. Hint: Use SOA Infrastructure > Home >Deployed Composites to access the composites.