Oracle® Fusion Middleware

Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack

11*g* Release 1 (11.1.1.7)

E17366-07

February 2013

Describes how to use Composite Application Validation System initiators and simulators to test AIA service integrations. Test initiators simulate service invocations and simulators simulate service endpoints. Describes how to use error handling and logging components, including error notifications and trace and error logs, to support services operating in an AIA ecosystem.

ORACLE®

Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack, 11*g* Release 1 (11.1.1.7)

E17366-07

Primary Author: Oracle Corporation

Contents

17 Using the AIA Message Resubmission Utility

18 Using Trace and Error Logs

19 Accessing Oracle B2B Errors

20 Using the Code Compliance Inspector

List of Examples

List of Figures

List of Tables

Preface

Welcome to *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide*. This document describes how to use Composite Application Validation System initiators and simulators to test AIA service integrations. Test initiators simulate service invocations and simulators simulate service endpoints. It describes how to use error handling and logging components, including error notifications and trace and error logs, to support services operating in an AIA ecosystem.

Audience

This document is intended for users of the components and utilities delivered with Oracle Application Integration Architecture Foundation Pack.

Oracle AIA Guides

Oracle Application Integration Architecture (AIA) provides the following guides and resources for this 11.1.1.7 release:

- *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack*
- *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*
- *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*
- *Oracle Fusion Middleware Reference Process Models User's Guide for Oracle Application Integration Architecture Foundation Pack*
- *Oracle Fusion Middleware Migration Guide for Oracle Application Integration Architecture*
- *Oracle Fusion Middleware Product-to-Guide Index for Oracle Application Integration Architecture Foundation Pack*

Related Documents

The following guides are relevant to Oracle AIA development activities and are provided as a part of the overall Oracle Fusion Middleware 11.1.1.7 documentation library:

- *Oracle Fusion Middleware Installation Guide for Oracle SOA Suite and Oracle Business Process Management Suite*

- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
- *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*
- *Oracle Fusion Middleware User Guide for Oracle Enterprise Repository*

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Enabling Accessibility Features for AIA Home

The Application Integration Architecture (AIA) Home page appears after installing Oracle AIA Foundation Pack. For more information, see *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack 11g Release 1 (11.1.1.7)*.

Users can set options to enable screen readers, high contrast colors, and large fonts:

- Screen Reader: If you are using a screen reader, we recommend that you select the Screen Reader option to ensure that your screen reader can access and read all components of the application. When screen-reader mode is enabled, the application displays a screen-reader optimized view of components. Screen-reader mode may degrade, but not obscure, the display for sighted users. If this option is not enabled, your screen reader may not be able to access and read all components.
- High Contrast Colors: Select the High Contrast Colors option to display the application using high-contrast-friendly visual content. Enabling high-contrast mode makes the application compatible with operating systems or browsers that have high-contrast features enabled. For example, the application will change its use of background images and background colors to prevent the loss of visual information. Note that high-contrast mode is more beneficial if it is used in conjunction with your browser's or operating system's high-contrast mode. Also, you may find it beneficial to use the large-font mode along with the high-contrast mode.
- Large Fonts: Select the Large Font option to display the application using browser-zoom-friendly content. Enabling large-font mode displays the application using text and containers that are scalable in size. This makes the application compatible with browsers that are set to larger font sizes and to work with browser-zoom capabilities. If you are not using the browser-based large-font mode or zoom capabilities, you should disable this option. Also, you may find it beneficial to use the high-contrast mode along with the large-font mode. If this

option is not enabled, most text and many containers will use a fixed size to provide a consistent look.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in This Guide for Release 11.1.1.7

The following table lists the content that has been added or updated.

For a list of known issues (release notes), see the "Known Issues for Oracle SOA Products and Oracle AIA Foundation Pack" at http://www.oracle.com/technetwork/middleware/docs/soa-aiafp-knownissuesindex-364630.html.

Content	Changes Made
Chapter 2	Added information about CAVS WS-Addressing.
Chapter 17	Added information about how to use the message resubmission user interface.
Chapter 20	New chapter that describes how to use the Code Compliance Inspector (CCI). This feature used to be called the PIP Auditor.

1

Introduction to the Composite Application Validation System

This chapter describes the purpose and key components of the Composite Application Validation System (CAVS). It also describes design assumptions and knowledge prerequisites.

The Composite Application Validation System (CAVS) is a framework that provides a structured approach to test integration of Oracle Application Integration Architecture (AIA) services. The CAVS includes test initiators that simulate web service invocations and simulators that simulate service endpoints.

This chapter includes the following sections:

1.1 Describing the Purpose of the Composite Application Validation System

In the context of AIA, where there is a sequence of service invocations; spanning Application Business Connector Services (ABCSs), Enterprise Business Services (EBSs), Enterprise Business Flows (EBFs), and participating applications; the CAVS test initiators and simulators enable a layered testing approach. Each component in an integration can be thoroughly tested without having to account for dependencies by using test initiators and simulators on either end.

Consequently, when you build an integration, you have the ability to add new components to an already tested subset, allowing any errors to be constrained to the new component or to the interface between the new component and the existing component. This ability to isolate and test individual web services within an integration provides the benefit of narrowing the test scope, thereby distancing the service test from possible faults in other components.

Test initiators and simulators can be used independent of each other, thereby allowing users to effectively substitute them for non-available AIA services or participating applications.

The CAVS provides a repository that stores these test initiator and simulator definitions created by the CAVS user, as well as an interactive user interface to create

and manage the same. Tests can be configured to run individually or in a single-threaded batch.

The CAVS provides value as a testing tool throughout the integration development life cycle:

- Development

  Because integration developers working with AIA are dealing with integrating disparate systems, they typically belong to different teams. To this end, the CAVS provides an effective way to substitute dependencies, letting developers focus on the functionality of their own service rather than being preoccupied with integrations to other services.

- Quality assurance

  The CAVS allows quality assurance engineers to unit and flow test integrations, thereby providing a way to easily certify different pieces of an integration. The reusability of test definitions, simulators, and test groups helps in regression testing and provides a quick way to certify new versions of services.

1.2 Describing Key Components of the CAVS Framework

The CAVS framework operates using the following key components:

- Test definition
- Simulator definition

Test Definition

The CAVS test initiator reads test data and feeds it to the web service being tested. You create the test data as a part of a test definition. The test definition is a configuration of the test initiator and contains test execution instructions.

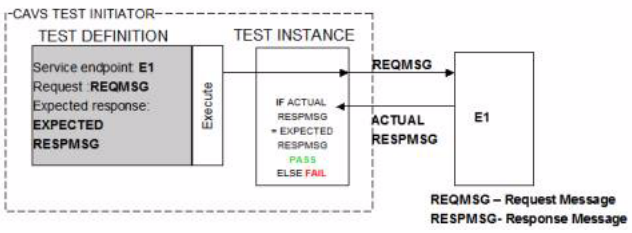
The CAVS user creates a definition using the CAVS user interface (UI) to define the service endpoint URL that needs to be invoked, as well as the request message that will be passed along with metadata about the test definition itself.

For more information about creating test definitions, see Chapter 4, "Creating and Modifying Test Definitions."

The test initiator is a logical unit that executes test definitions to call the endpoint URL defined and creates test instances. This call is no different from any other request initiated by other clients. If the test definition Service Type value is set to Synchronous or Asynchronous two-way, the actual response can be verified against predefined response data to validate the accuracy of the response.

Figure 1–1 illustrates the high-level concept of the test initiator.

Figure 1–1 CAVS Test Definition



Simulator Definition

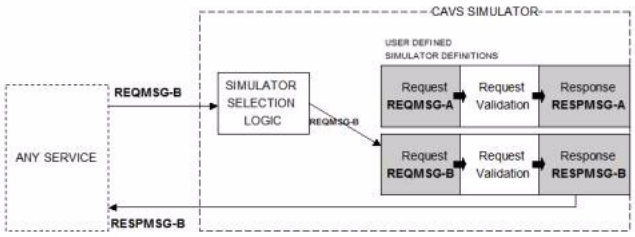
The CAVS simulator is used to simulate a web service. Simulators typically contain predefined responses for a specific request. CAVS users create several simulator definitions, each for a specific set of input.

At run time, the CAVS simulator framework receives data from the service being tested. Upon receiving the request, CAVS locates the appropriate simulator definition, validates the input against predefined request values, and then returns predefined response data so that the web service being tested can continue processing.

For more information about creating simulator definitions, see Chapter 5, "Creating and Modifying Simulator Definitions".

Figure 1–2 illustrates the high-level concept of the CAVS simulator:

Figure 1–2 CAVS Simulator



1.3 Describing the CAVS Design Assumptions and Knowledge Prerequisites

The CAVS operates with the following design assumptions:

- The CAVS assumes that the requester and provider ABCSs it is testing are implemented using BPEL.
- The CAVS is designed to initiate requests and simulate responses as SOAP messages using SOAP over HTTP. The request and response messages that you define in test and simulator definitions must contain the entire XML SOAP document, including the SOAP envelope, message header, and body (payload).
- The correlation logic between the test initiator and the response simulator is based on timestamps only. For this reason, test and simulator instances generated in the database schema will not always be reconcilable, especially when the same web service is invoked multiple times during a very short time period, as in during performance testing.
- The CAVS does not provide or authenticate security information for web services that are initiated by a test initiator or received by a response simulator. However, security information passed through the system by the web service can be used as a part of verification and validation logic.
- When a participating application is involved in a CAVS testing flow, execution of tests can potentially modify data in a participating application. Therefore, consecutive running of the same test may not generate the same results. The CAVS is not designed to prevent this kind of data tampering because it supports the user s intention to include a real participating application in the flow. The CAVS has no control over modifications that are performed in participating applications.

  This issue does not apply if your CAVS test scenario uses test definitions and simulator definitions to replace all participating applications and other

dependencies. In this case, all cross-reference data is purged after the test scenario has been executed. This enables rerunning of the test scenario.

Note: CAVS cross reference data is purged at the end of a test execution when executing a test definition and at the end of a test group execution when executing a test group definition. Therefore, if you want to execute test definitions that are dependent on cross referencing data created by earlier test executions, ensure that you include all dependent test definitions in a test group and execute the test group.

For more information about how to make test scenarios rerunnable, see Chapter 11, "Purging CAVS-Related Cross-Reference Entries to Enable Rerunning of Test Scenarios."

To work effectively with the CAVS, users must have working knowledge of the following concepts and technologies:

- AIA
- XML
- XPath
- SOAP

2

Preparing to Use the Composite Application Validation System

This chapter provides a high-level discussion of questions you should answer to help gather requirements for the tests you want to create and run in the Composite Application Validation System (CAVS).

Before you start creating and running tests in the CAVS, take the time to gather your test requirements and plan your approach to using CAVS.

This chapter includes the following sections:

2.1 What Can I Test Using CAVS?

The CAVS supports the following testing scenarios:

- Create and execute test definitions against actual services in participating applications.
- Create and execute test definitions that call services that call simulators, which simulate actual services in participating applications.
- Use actual services in participating applications in cooperation with simulators to simulate any unavailable services.

2.2 What Are the Oracle AIA Components That I Need to Test?

Examine the components involved in the scenario that you need to test. Which of the following components does the scenario include?

- Requester Application Business Connector Services (ABCSs)
- Provider ABCSs
- Enterprise Business Flows (EBFs)
- Exposed services in participating applications

2.3 Which Message Exchange Pattern Is Being Used by the Components Being Tested?

This section includes the following topics:

After you have assessed which components you need to test, identify the message exchange pattern (MEP) being used by the components to help you determine which types of CAVS test and simulator definitions you need to create. Based on the sequence of service calls and the MEPs employed, you can determine if you need to use synchronous, notify, or asynchronous two-way test definitions and/or simulator definitions.

This section discusses CAVS process flows for testing the following MEPs:

- Synchronous (request-and-response)
- Asynchronous (notify)
- Asynchronous (two-way)

> **Note:** The information in this chapter provides CAVS processing details that can inform your creation of test and simulator definitions in the CAVS user interface (UI). As you prepare to define and run tests for a particular web service, refer to the section in this chapter that corresponds to the message exchange pattern of the service you want to test.

2.3.1 Describing CAVS Process Flows for Testing the Synchronous Message Exchange Pattern

The following diagrams describe CAVS process flows for testing a provider ABCS using a synchronous MEP.

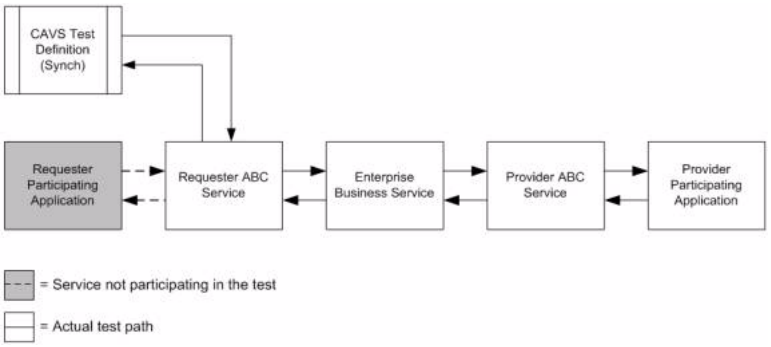
These sample flows can be used as the basis for testing other artifacts as well, such as requester ABCSs, EBFs, or provider services.

Synchronous MEP Testing Flow Using a Test Definition

The requester participating application is replaced by the CAVS test definition. The test definition points to the URL of the requester ABCS. It uses a composed request message to invoke the ABCS and expects a message in response.

Figure 2–1 illustrates testing a synchronous MEP using a CAVS test definition.

Figure 2–1 Testing a Synchronous MEP Using a CAVS Test Definition



Synchronous MEP Testing Flow Using a Test Definition and Simulator Definition

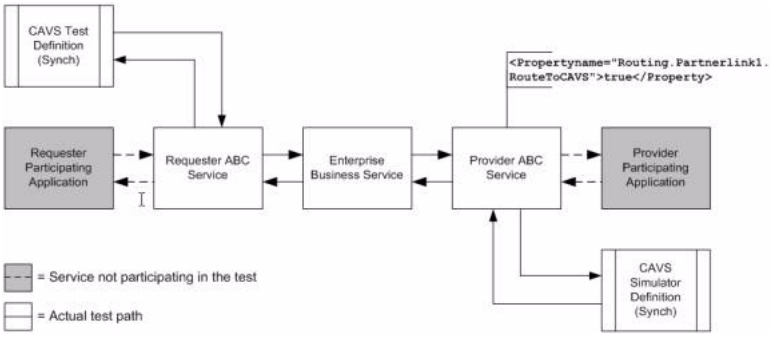
The requester participating application is replaced by the CAVS test definition. The test definition points to the URL of the requester ABCS. It uses a composed request message to invoke the ABCS and expects a message in response.

The provider participating application is replaced by the CAVS simulator definition. The provider ABCS is programmed to route to this simulator instead of the provider participating application. The simulator definition contains a predefined request and response message pair.

The simulator definition performs validations on message input from the provider ABCS and sends the message back to the provider ABCS. The provider ABCS sends the message back to the test definition, which validates this actual response against its predefined expected response.

Figure 2–2 illustrates testing a synchronous MEP using CAVS test and simulator definitions.

Figure 2–2 Testing a Synchronous MEP Using a CAVS Test Definition and Simulator Definition



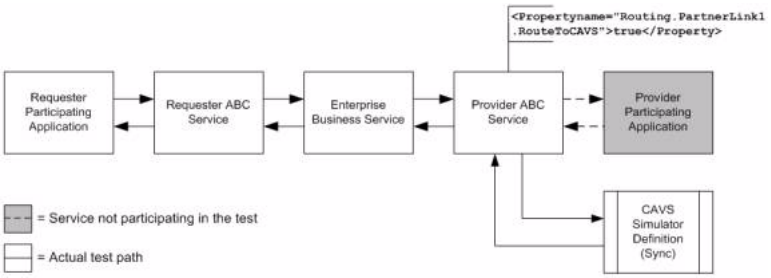
Synchronous MEP Testing Flow Using a Simulator Definition

The provider participating application is replaced by the CAVS simulator definition. The provider ABCS is programmed to route to this simulator instead of the provider participating application. The simulator definition contains a predefined request and response message pair.

The simulator definition performs validations on message input from the provider ABCS and sends the message back to the provider ABCS. The provider ABCS sends the message back to requester participating application.

Figure 2–3 illustrates testing a synchronous MEP using a CAVS simulator definition.

Figure 2–3 Testing a Synchronous MEP Using a CAVS Simulator Definition



2.3.2 Describing CAVS Process Flows for Testing the Asynchronous (Notify) Message Exchange Pattern

The following diagrams describe CAVS process flows for testing a provider ABCS using an asynchronous (notify) MEP.

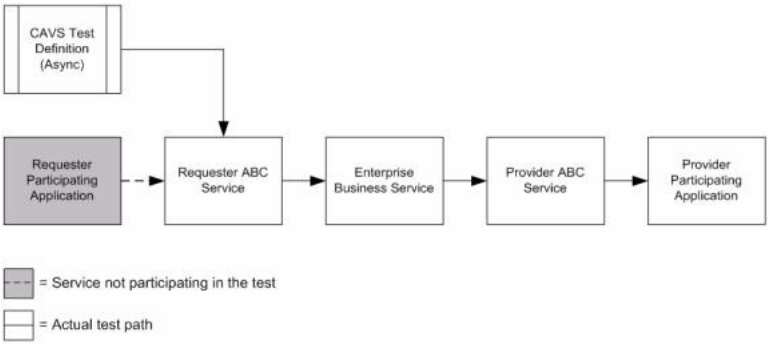
These sample flows can be used as the basis for testing other artifacts as well, such as the requester ABCS, EBF, or the provider service itself.

Asynchronous (Notify) MEP Testing Flow Using a Test Definition

The requester participating application is replaced by the CAVS test definition. The test definition points to the URL of the requester ABCS. It uses a composed request message to invoke the ABCS and does not expect a message in response.

Figure 2–4 illustrates testing an asynchronous (notify) MEP using a CAVS test definition.

Figure 2–4 Testing an Asynchronous (Notify) MEP Using a CAVS Test Definition



Asynchronous (Notify) MEP Testing Flow Using a Test Definition and Simulator Definition

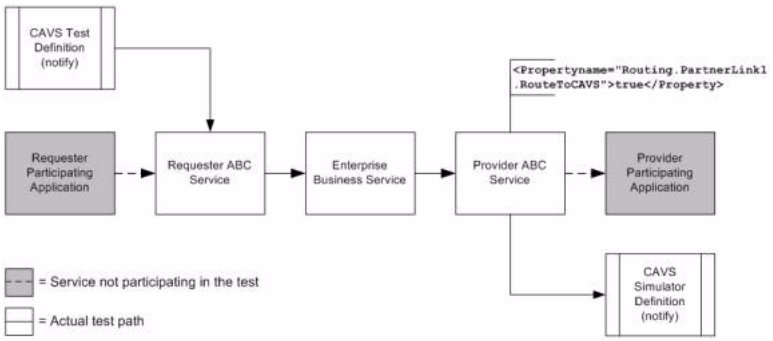
The requester participating application is replaced by the CAVS test definition. The test definition points to the URL of the requester ABCS. It uses a composed request message to invoke the ABCS and does not expect a message in response.

The provider participating application is replaced by the CAVS simulator definition. The provider ABCS is programmed to route to this simulator instead of the provider participating application. The simulator definition contains a predefined expected request message.

The simulator definition performs validations on message input from the provider ABCS.

Figure 2–5 illustrates testing an asynchronous (notify) MEP using CAVS test and simulator definitions.

Figure 2–5 Testing an Asynchronous (Notify) MEP Using a CAVS Test Definition and Simulator Definition



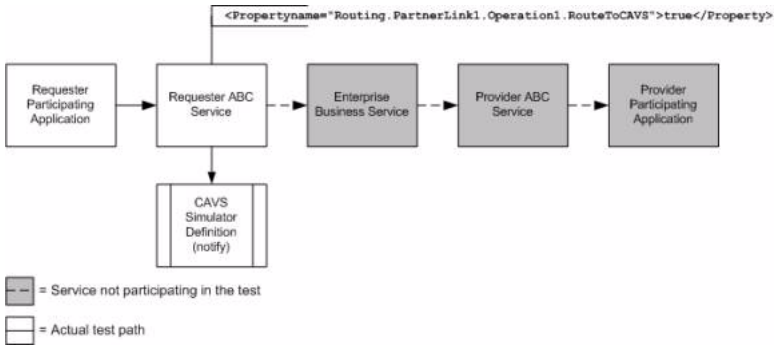
Asynchronous (Notify) MEP Testing Flow Using a Simulator Definition

The provider participating application is replaced by the CAVS simulator definition. The requester ABCS is programmed to route to this simulator instead of the provider participating application. The simulator definition contains a predefined expected request message.

The simulator definition performs validations on message input from the provider ABCS.

Figure 2–6 illustrates testing an asynchronous (notify) MEP using a CAVS simulator definition.

Figure 2–6 Testing an Asynchronous (Notify) MEP Using a CAVS Simulator Definition



2.3.3 Describing Flows for Testing the Asynchronous Two-Way Message Exchange Pattern

The following diagrams describe CAVS process flows for testing a provider ABCS using an asynchronous two-way MEP.

These sample flows can be used as the basis for testing other artifacts as well, such as the requester ABCS, EBF, or the provider service itself.

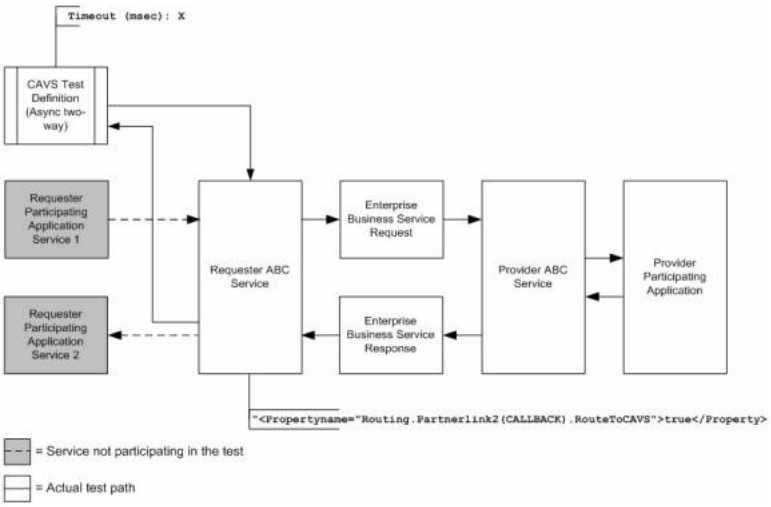
Asynchronous Two-Way MEP Testing Flow Using a Test Definition

The requester participating application is replaced by the CAVS test definition. The test definition points to the URL of the requester ABCS. It uses a composed request

message to invoke the ABCS and expects an eventual message in response. The test definition includes a timeout value. If no response message is received within this timeout value, the test definition will experience a timeout failure.

Figure 2–7 illustrates testing an asynchronous (two-way) MEP using a CAVS test definition.

Figure 2–7 Testing an Asynchronous Two-Way MEP Using a CAVS Test Definition



Asynchronous Two-Way MEP Testing Flow Using a Test Definition and Simulator Definition

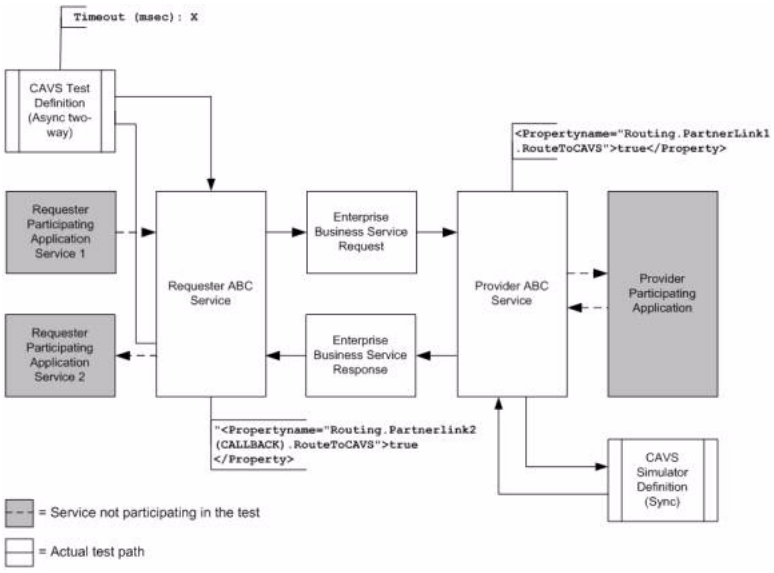
The requester participating application is replaced by the CAVS test definition. The test definition points to the URL of the requester ABCS. It uses a composed request message to invoke the ABCS and expects an eventual message in response. The test definition includes a timeout value. If no response message is received within this timeout value, the test definition will experience a timeout failure.

The provider participating application is replaced by the CAVS simulator definition. The provider ABCS is programmed to route to this simulator instead of the provider participating application. The simulator definition contains a predefined request and response message pair.

The simulator definition performs validations on message input from the provider ABCS and sends the message back to the provider ABCS. The provider ABCS sends the message back to the test definition, which validates this actual response against its predefined expected response.

Figure 2–8 illustrates testing an asynchronous (two-way) MEP using CAVS test and simulator definitions.

Figure 2–8 Testing an Asynchronous Two-Way MEP Using a CAVS Test Definition and Simulator Definition



Asynchronous Two-Way MEP Testing Flow Using a Simulator Definition

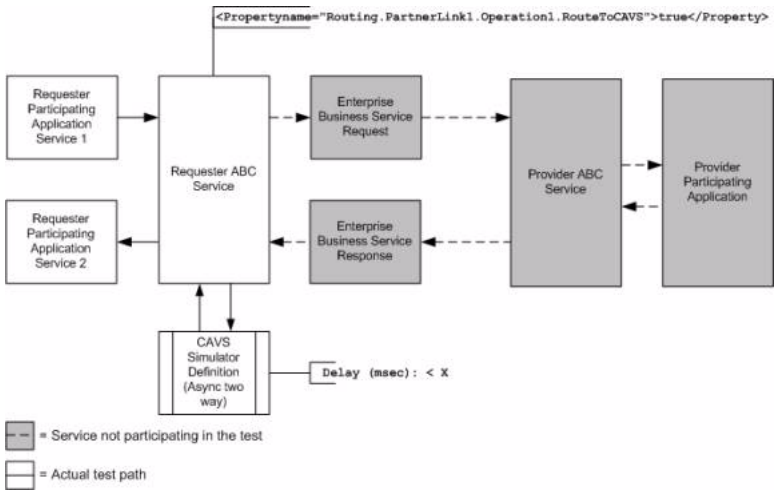
The provider ABCS is replaced by the CAVS simulator definition. The requester ABCS is programmed to route to this simulator instead of having the flow reach the provider ABCS.

The simulator definition contains a predefined request and response message pair, as well as a user-defined delay value. The simulator definition will delay its response by this amount of time to simulate the asynchronous two-way nature of the provider participating application.

The simulator definition performs validations on message input from the requester ABCS and sends the message back to the requester ABCS.

Figure 2–9 illustrates testing an asynchronous (two-way) MEP using a CAVS simulator definition.

Figure 2–9 Testing an Asynchronous Two-Way MEP Using a CAVS Simulator Definition



2.4 Does the Scenario Need to be Unit or Flow Tested?

This section discusses different configurations for test and simulator definitions to achieve unit and flow tests.

This section includes the following topics:

- Section 2.4.1, "Describing a Unit Test Configuration"
- Section 2.4.2, "Describing a Flow Test Configuration"
- Section 2.4.3, "Describing a Complex Flow Test Configuration"

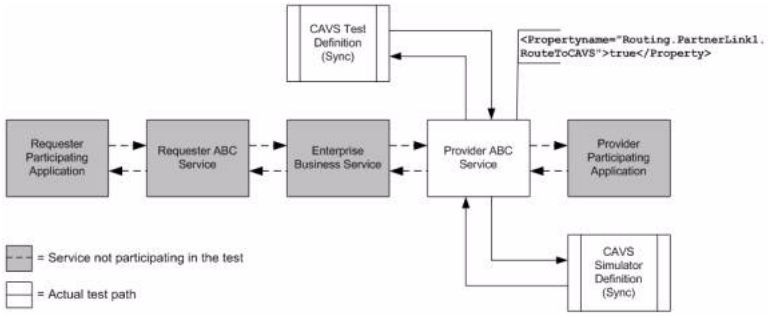
2.4.1 Describing a Unit Test Configuration

This section will use a synchronous provider ABCS as the focus of the test example. However, this test configuration is not specific to message exchange patterns, so it can be applied to asynchronous (notify) and asynchronous two-way components as well.

To unit test a component, place a test definition before the component and a simulator definition after it. This isolates the focus of the test to the single component.

Figure 2–10 illustrates how to unit test a provider ABCS.

Figure 2–10 Unit Testing a Provider ABCS

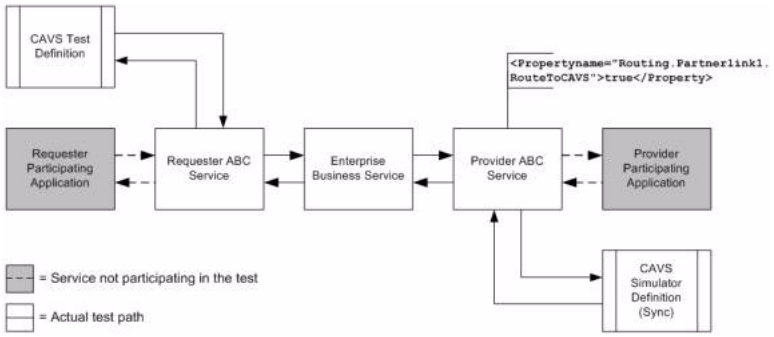


2.4.2 Describing a Flow Test Configuration

This section will use a synchronous provider ABCS as the focus of the test example. However, this test configuration is not specific to message exchange patterns, so it can be applied to asynchronous (notify) and asynchronous two-way components as well.

After you have unit tested the components in a scenario, you can flow test the scenario. To flow test a scenario, place a test definition before the requester ABCS at the front of the scenario and a simulator definition after the provider ABCS at the end of the scenario, as shown in Figure 2–11.

Figure 2–11 Flow Testing a Scenario

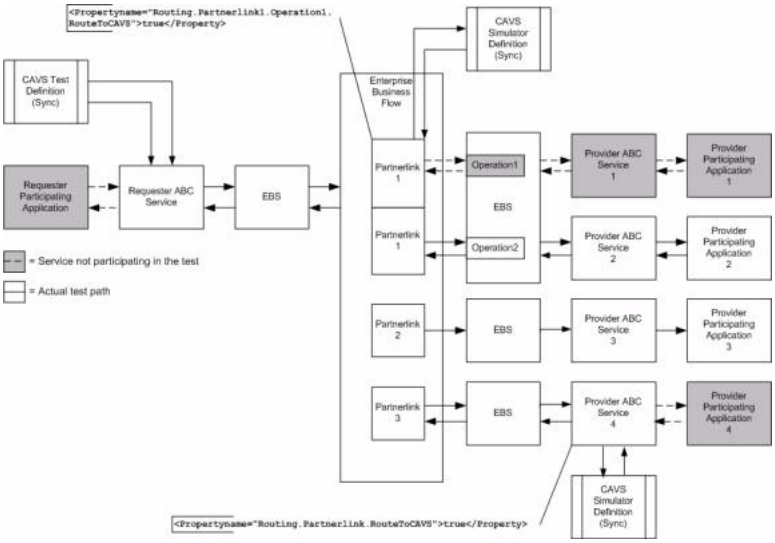


2.4.3 Describing a Complex Flow Test Configuration

This section uses an EBF as the focus of the test example. However, this test configuration is not specific to EBFs, so it can be applied to any service that conducts chatty conversations.

You can place a test definition before the requester ABCS at the front of a scenario and enable the EBF to make calls out to simulator definitions whenever required. You can then go on to replace some of the provider applications with simulators at the end of the scenario, as shown in Figure 2–12.

Figure 2–12 Complex Flow Testing an EBF



2.5 Do I Have the Content I Need to Create the Definitions?

This section includes the following topic: Section 2.5.1, "How to Obtain Message XML Text from a BPEL Process".

After you know what you need to test and which CAVS definitions you need to create, assess whether you have all of the content you need to create the definitions.

To create your test definitions and simulator definitions, you will primarily need request and response XML text.

If you are creating a Test Definition (or an asynchronous two-way simulator), you will need the endpoint URL of the web service you are testing.

The endpoint URL value can be found in the WSDL of the web service that you want to test.

When the endpoint URL is provided, CAVS will present you with available SOAP actions. After you select the required SOAP action, CAVS will automatically generate the message stub for the service being called. You can then include data within the XML tags generated.

In either case, you can use the BPEL Console to obtain request and response XML text. Run the processes you are testing at least once with all participating applications and services in place and with the desired results. The XML messages generated by this successful run of the processes will provide your request and response XML for test and simulator definitions. The following section describes how to use the BPEL console to obtain these XML messages.

Note: Obtaining response message XML text is only applicable when testing synchronous and asynchronous two-way processes.

2.5.1 How to Obtain Message XML Text from a BPEL Process

To obtain request and response message XML text:

1. Access the BPEL Console for your AIA implementation.
2. Click the **Instances** tab.
3. In the **BPEL Process** drop-down list box, select the BPEL process for which you are creating a test definition and click **Go**. BPEL process instances for the selected BPEL process display in the List of BPEL Process Instances frame. Sort by **Last Modified**, if you want to access the most recent instance.
4. Click the link for the instance you want to use for your request and response XML message text.
5. Click the **Flow** link.

Note: If the instance you selected contains any faults, you may want to consider selecting a different instance. However, if you are trying to test fault messaging in the BPEL, you must select a BPEL process instance that contains the fault.

---

6. To obtain the Request Message XML text:
   a. Click the **receiveInput** element to get your test definition request message XML text.
   b. Click the **Copy details to clipboard** link at the bottom of the pop-up box displaying input Variable data.
   c. Open an XML editor and paste the XML text into a blank document.
   d. Remove the opening and closing `inputVariable (XYZ_InputVariable`, in the case of a non-BPEL service, such as a participating application service) and `part` elements.
   e. Copy and paste the remaining XML text in the default SOAP envelope provided in the **Request Message** field on the Test Definition page or Simulator Definition page. Paste the XML text into the area indicated by the `Paste your SOAP Message Content here` placeholder text.

7. To obtain the Response Message XML text:

> **Note:** Obtaining response message XML text is only applicable when testing a synchronous process.

   a. Click the **reply Output** element to get your test definition response message XML text.

   b. Click the **Copy details to clipboard** link at the bottom of the pop-up box displaying output Variable data.

   c. Open an XML editor and paste the XML text into a blank document.

   d. Remove the opening and closing `inputVariable (XYZ_InputVariable`, in the case of a non-BPEL service, such as a participating application service) and `part` elements.

   e. Copy and paste the remaining XML text in the default SOAP envelope provided in the **Response Message** field on the Test Definition page or Simulator Definition page.

3

Introduction to Defining and Running CAVS Tests Using the CAVS UI

This chapter describes the Composite Application Validation System (CAVS) user interface and provides an overview of how to define and run tests. It also provides instructions on how to execute CAVS definitions as web services.

This chapter includes the following sections:

3.1 Describing the CAVS UI

The Composite Application Validation System (CAVS) enables you to configure test data, execute tests, review test results, and migrate tests using the following user interface (UI) components.

Test Definitions

A test definition is a configuration of a single execution of the test initiator service. The test definition stores test data and test execution instructions. A test definition can be executed alone, or in a single-threaded batch as a part of a group definition.

You will find that the values you set for a test definition and simulator definition are similar. The test definition differs from the simulator definition in that it is an active participant in the CAVS framework, initiating tests. The test definition carries the following values that are not a part of the simulator definition. These values inform the active state of the test definition:

- Endpoint URL
- SOAP Action

For more information about test definitions, see Chapter 4, "Creating and Modifying Test Definitions."

If required by the business service pattern of the web service you are testing, you can assign a simulator definition to the test definition.

Simulator Definitions

A simulator definition is a configuration of a single execution of response simulator service. The simulator definition simulates a web service and receives data from the

tested web service and returns previously defined data so that the tested web service can continue processing.

You will find that the values you set for a simulator definition and test definition are similar. The simulator definition differs from the test definition in that it is a passive participant in the CAVS framework, awaiting initiation.

The simulator definition carries the following additional XPath attributes that are not a part of the test definition. These values participate in simulator definition request matching:

- Is Node Key
- Key Node Value

For more information about simulator definitions, see Chapter 5, "Creating and Modifying Simulator Definitions."

The success of the test is verified based on the simulator definition's previously defined data being accurately returned and matched to the expected response results defined in the test definition.

Group Definitions

A group definition is a configuration of a single execution of one or more test definitions in a single-threaded batch.

Test Instances

A test instance captures the details of the execution of a test definition.

Simulator Instances

A simulator instance captures the details of a simulator definition's behavior during the execution of a test definition with which it is associated.

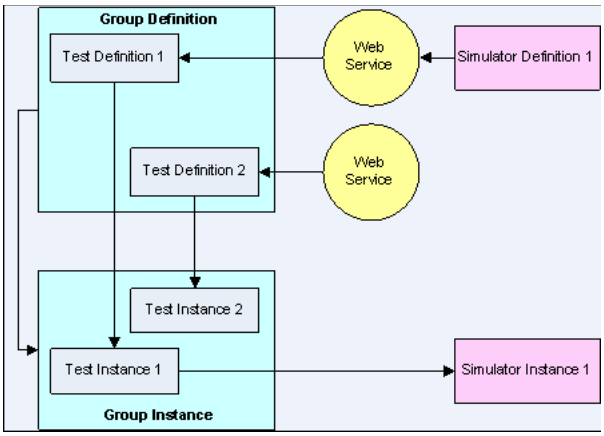
Group Instances

A group instance captures the details of the execution of a group definition.

Component Overview

Figure 3–1 provides a high-level overview of the relationships among the CAVS components discussed in this section.

Figure 3–1 Overview of CAVS Component Relationships



3.2 Overview of Defining and Running CAVS Tests

This high-level procedure provides the steps involved in defining and running tests using the CAVS UI.

To define and run tests using the CAVS UI:

1. Assess your test requirements and gather required content.

   For more information, see Chapter 2, "Preparing to Use the Composite Application Validation System."

2. If your test requires a test definition, access the Create Test page to create your test definition.

   For more information, see Chapter 4, "Creating and Modifying Test Definitions."

3. If your test requires a simulator definition, access the Create Simulator page to create your simulator definition.

   For more information, see Chapter 5, "Creating and Modifying Simulator Definitions."

4. If your test requires a test definition and simulator definition, you can link their definitions on either Modify Test Definition page or Modify Simulator Definition page.

5. If your test requires multiple (single-threaded) executions of the same or different test definitions, create a group definition on the Create Group Definition page.

   For more information, see Chapter 7, "Working with Group Definitions."

6. If your test or group definition utilizes a simulator definition, you must set the Application Business Connector Service (ABCS) being tested to route to the response simulator. To do this:

   a. Access the **Routing Setup** page.

   b. Create a routing setup ID for the invoking service being tested.

   c. Associate this routing setup ID with the test definition being used to test your scenario.

      For more information about routing setup IDs, see Chapter 8, "Defining CAVS Routing Setup IDs."

   d. Alternatively, you can quickly set up a routing configuration on the Configurations page.

      For more information about routing configurations, see Section 8.5, "How to Set Up CAVS Routing Configurations Without Creating Routing Setup IDs."

7. To run a single test, access the test definition on the Definitions page or Modify Test Definitions page to run the test.

   For more information, see Chapter 6, "Searching for Test and Simulator Definitions."

   To run a group test, access the Group Definition page or Group Definition Detail page to run the group test.

   For more information, see Chapter 7, "Working with Group Definitions."

8. After an individual test has been executed, view the test results generated for the test instance on the Test Instances Detail page. If the test definition includes an

associated simulator definition, view the simulator instance on the Simulator Instances Detail page.

For more information, see Chapter 9, "Working with Test and Simulator Instances."

After a group of tests has been executed, view the test results generated for the group instance on the Group Instance Detail page.

For more information, see Chapter 10, "Working with Group Instances."

9. After testing is complete for a test that involved a simulator definition for which you defined a routing configuration on the Configurations page, be sure to reset the ABCS tested to return to routing to its usual production destination and no longer route to the response simulator. To do this:

   a. Access the Configurations page.

   b. Clear the **Route To CAVS** option.

   c. Click **Reload**.

   For more information about routing configurations, see Section 8.5, "How to Set Up CAVS Routing Configurations Without Creating Routing Setup IDs."

3.3 How to Execute CAVS Definitions as Web Services

The CAVS provides a web service that enables you to execute test definitions and test group definitions without the use of the CAVS user interface.

You can call this CAVS web service from any system: http://<hostname>:<port>/AIAValidationSystemAPIService/AIAValidationSystemAPIServiceSoapHttpPort.

This web service provides two operations:

- **executeDefinition**

  Executes a given test definition ID.

- **executeGroupDefinition**

  Executes a given test group definition ID.

Typically, these operations can be consumed by third-party testing tools or other systems to execute test definitions and test group definitions whenever desired, without the use of the CAVS UI.

The WSDL that defines the service contract is: http://<hostname>:<port>/AIAValidationSystemAPIService/AIAValidationSystemAPIServiceSoapHttpPort?wsdl.

4

Creating and Modifying Test Definitions

This chapter describes how to create and modify test definitions and how to provide multiple request and response message sets in a single test definition.

A test definition is a configuration of a single execution of the test initiator service. The test definition stores test data and test execution instructions. A test definition can be executed alone, or in a single-threaded batch as a part of a group definition.

This chapter includes the following sections:

4.1 How to Create a Test Definition

To create a test definition:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. Click the **Create Test** button.

   The Create Test page displays, as shown in Figure 4–1 and Figure 4–2.

Figure 4–1 Create Test Page (1 of 2)

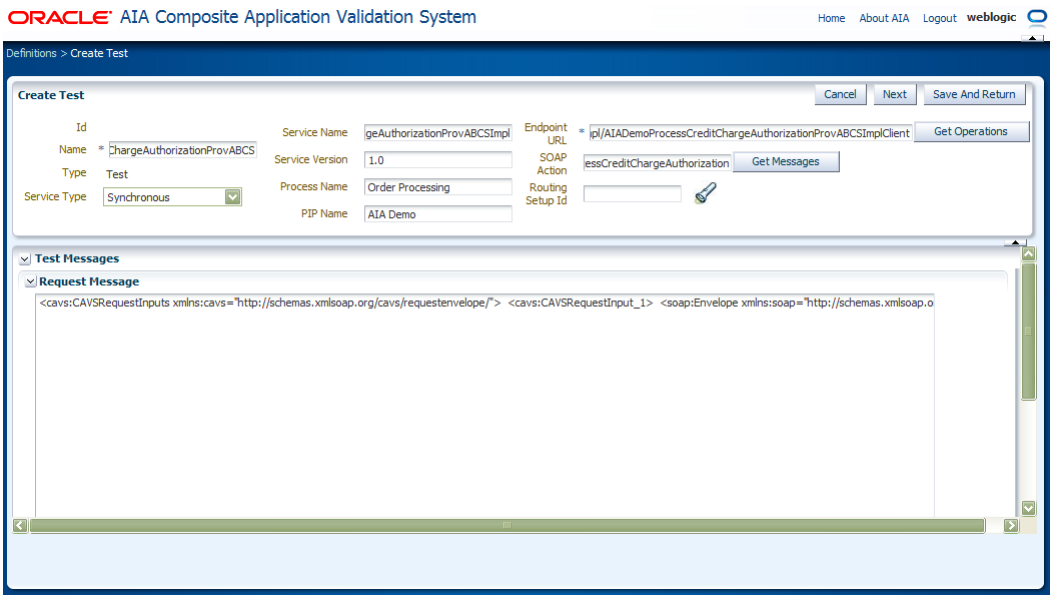
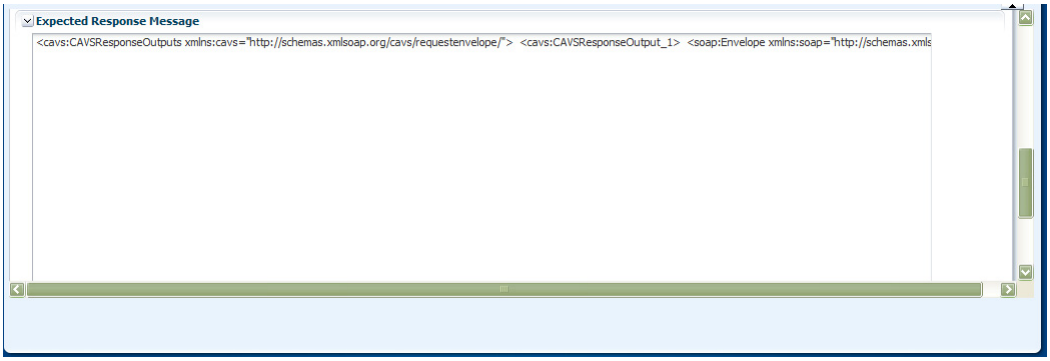


Figure 4–2 Create Test Page (2 of 2)



2. On the Create Test page, use the page elements discussed in Table 4–1 to create test definitions.

Table 4–1 Create Test Page Elements

Element	Description
Id	Upon saving the test definition, displays a unique key identifier that is assigned to the test definition.
Name	Enter a descriptive name that you want to use for the test definition.
Type	Displays the type of definition you chose to create. On the Create Test page, this value will always be set to Test.
Service Type	Select the business service pattern of the web service that you want to test using the test definition: **Synchronous (request-and-reply)**, **Notify (asynchronous request-only)**, or **Asynchronous two way**.
Service Name	Enter the name of the web service that you want to test using the test definition. This is the name of the web service being called by the URL provided in the **Endpoint URL** field.

Table 4–1 (Cont.) Create Test Page Elements

Element	Description
Service Version	Enter the version of the web service that you want to test using the test definition. This is the version of the web service being called by the URL provided in the **Endpoint URL** field.
Process Name	Enter the name of the process that includes the web service that you want to test using the test definition.
PIP Name (Process Integration Pack name)	Enter the name of the PIP that includes the web service that you want to test using the test definition.
Endpoint URL	Enter the URL of the web service that you want to test using the test definition. The endpoint URL value can be found in the WSDL of the web service that you want to test.
Get Operations	Click to display the list of operations supported by the WSDL associated with the Endpoint URL value you provided. Supported operations display in the Select WSDL Operations window. Select the operation that you want to test using the test definition. The selected operation displays in the SOAP Action field.
SOAP Action	If you clicked **Get Operations** to select an operation in the Select WSDL Operations window, the selected operation displays here. Alternatively, you can manually enter the operation called by the web service that you want to test using the test definition. The value you enter must match an action provided in the WSDL of the web service that you want to test.
Get Messages	Click to generate a request stub message for the operation specified in the SOAP Action field. For test definitions with the Service Type field set to Synchronous, the response stub message will also be generated.
Routing Setup Id	Select a routing configuration that you want to use for the test. For more information about routing configurations, see Chapter 8, "Defining CAVS Routing Setup IDs."

Test Messages

Use the Test Messages group box to enter request and response XML message text. By default, SOAP envelope XML text is provided in these fields. You can use the **Get Messages** button to generate request and response stub messages based on selected endpoint URL and operation values. Alternatively, you can paste XML text within this default SOAP envelope or paste your own XML text already enclosed in an envelope into these fields.

Elements available in the Test Messages group box are discussed in Table 4–2.

For more information about obtaining request and response XML message text, see Section 2.5.1, "How to Obtain Message XML Text from a BPEL Process."

For more information about how to create test request and response messages that hold multiple sets of test data in a single definition, see Section 4.3, "How to Provide Multiple Request and Response Message Sets in a Single Test Definition."

Table 4–2 Create Test Page - Test Messages Group Box Elements

Element	Description
Request Message	Entering request message XML text for a test definition is required, whether the Service Type field value is set to **Synchronous**, **Notify**, or **Asynchronous two way**. When you first access the Create Test page, the Request Message text box is populated with a SOAP stub message. You can use the **Get Messages** button to generate a request stub message based on selected endpoint URL and operation values. If you are manually entering your request message, the *Paste your SOAP Message Content here* text in the stub message indicates where you should paste your actual request message text. This request message should mimic the XML message text sent by the service that normally initiates the service. If the Service Type field value is set to **Synchronous** or **Asynchronous two way**, you may choose to not enter response message XML text in this field. You do not need to enter response message XML if you are manually entering XPath values directly on the Modify Test Definition page or if the test you are running does not require validation of the response message. For example, your test may be focused on just populating data.
Expected Response Message	The ability to enter response message XML text is available when the Service Type field value is set to **Synchronous** or **Asynchronous two way**. When you first access the Create Test page, the Expected Response Message text box is populated with a SOAP stub message. If you are manually entering your request message, the *Paste your SOAP Message Content here* text in the stub message indicates where you should paste your actual response message text. Enter a response message that is the expected response message XML. This facilitates the generation of XPath values, which are used to validate the actual response message returned in the test. You may also choose to manually enter or modify the XPath values directly on the Modify Test Definition page. If you are manually entering XPath values, you do not need to enter response message XML text. For test definitions with the Service Type field set to **Synchronous**, the response message stub will have been generated when you clicked the **Get Messages** button during request message generation. When you enter response message XML text on this page, you can click the **Generate Xpath** button on the Modify Test Definition page to generate the XPath values that will be used to validate the expected response message you entered on this page against the actual response returned by the test. The Expected Response Message text box is unavailable when the Service Type field value is set to **Notify**.
Cancel	Click to exit the page and return to the Definitions page.
Next	Click to save entries on the Create Test page and go to the Modify Test Definition page, where you can further edit your test definition, generate XPaths, and execute the test.
Save and Return	Click to save entries on the Create Test page and return to the Definitions page.

4.2 How to Modify a Test Definition

To modify a test definition:

1. Access the Modify Test Definition page, as shown in Figure 4–3, Figure 4–4, Figure 4–5, Figure 4–6, and Figure 4–7.

   To access the Modify Test Definition page, use one of the following navigation paths:

   - Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. Click the **Create Test** button. Enter required values on the Create Test page and click **Next**.
   - Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. Click an **Id** link for an unlocked test definition in the Search Result Selection grid on the Definitions page.
   - Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Instances** tab. Click a **Definition Id** link for an unlocked test definition on the Instances page.
   - Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Instances** tab. Click an **instance ID** link. Click a **Definition Id** link for an unlocked test definition on the Test Instances Detail page.

Figure 4–3 Modify Test Definition Page (1 of 5)

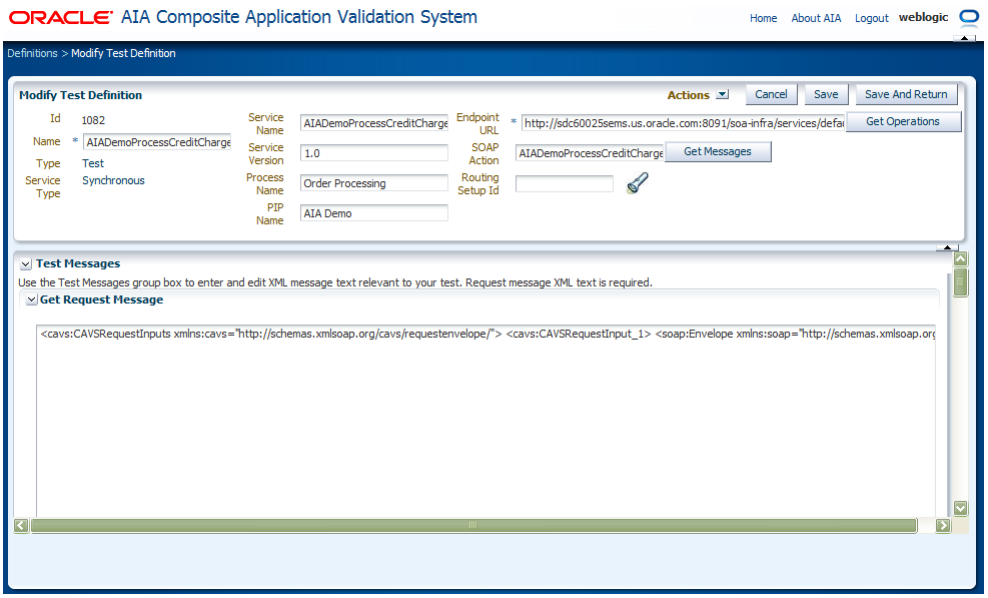


Figure 4–4 Modify Test Definition Page (2 of 5)

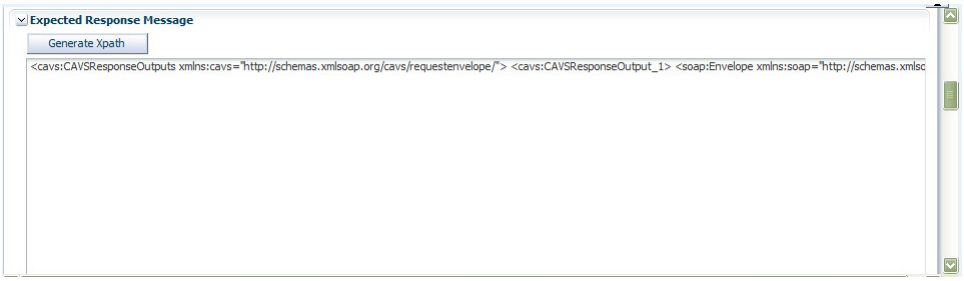


Figure 4–5 Modify Test Definition Page (3 of 5)

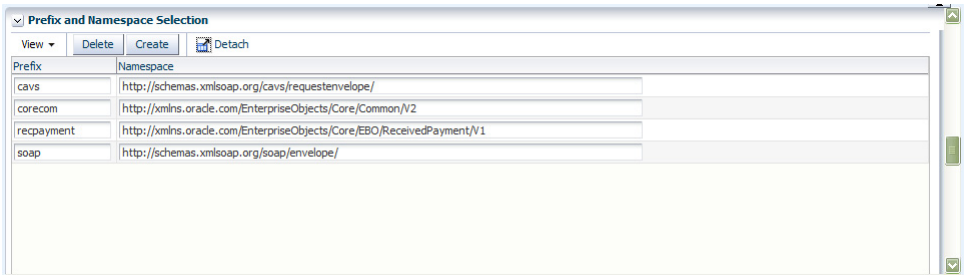


Figure 4–6 Modify Test Definition Page (4 of 5)

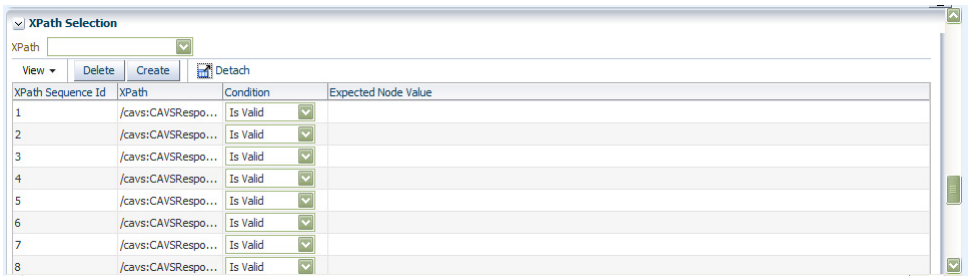
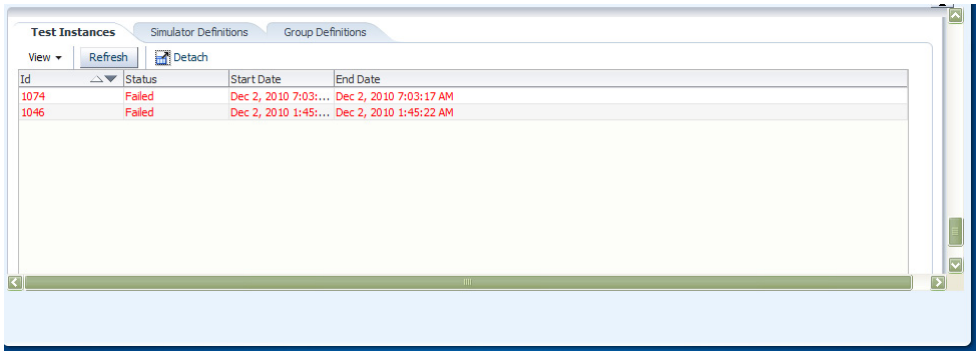


Figure 4–7 Modify Test Definition Page (5 of 5)



2. Use the page elements on the Modify Test Definition page to modify an existing test definition and execute and manage existing test definitions.

   The page displays values that have already defined for the test definition. You can modify the values in editable fields.

Most of the elements on this page also appear on the Create Test Definition page and are documented in Section 4.1, "How to Create a Test Definition." Any additional elements are discussed here.

The Time-out (msec) (in milliseconds) field displays only for a test definition with a Service Type value of **Asynchronous two way**.

Enter the number of milliseconds that you want the test definition to remain available for the asynchronous reply before timing out. If this length of time passes before the asynchronous response is returned, a failure will be issued.

If your test includes a simulator definition, the Time-out (msec) value you provide here must be greater than the Delay (msec) value defined on the simulator definition.

For more information about the Delay (msec) field, see Chapter 5, "Creating and Modifying Simulator Definitions"

Test Messages

Use the Test Messages group box to generate XPath values based on provided response XML message text. By default, SOAP envelope XML text is provided in these fields. You can use the **Get Messages** button to generate request and response stub messages based on selected endpoint URL and operation values. Alternatively, you can paste XML text within this default SOAP envelope, or paste your own XML text already enclosed in an envelope into these fields.

Available Test Message group box elements are discussed in Table 4–3.

For more information about obtaining request and response XML message text, see Section 2.5.1, "How to Obtain Message XML Text from a BPEL Process."

For more information about how to create test request and response messages that hold multiple sets of test data in a single definition, see Section 4.3, "How to Provide Multiple Request and Response Message Sets in a Single Test Definition."

Table 4–3 Modify Definitions Page - Test Messages Group Box Elements

Element	Description
Request Message	If request message XML text was entered on the Create Test page, it is accessible and editable on this page. Entering request message XML text for a test definition is required, whether the Service Type field value is set to **Synchronous**, **Notify**, or **Asynchronous two way**. You can use the **Get Messages** button to generate a request stub message based on selected endpoint URL and operation values. If you are manually entering your request message, the *Paste your SOAP Message Content here* text in the stub message indicates where you should paste your actual request message text. This request message should mimic the XML message text sent by the service that normally initiates the service.

Table 4–3 (Cont.) Modify Definitions Page - Test Messages Group Box Elements

Element	Description
Request CorrelationId Message	This field only displays for a test definition with the Service Type field value set to **Asynchronous two way**. For this service type, entering a correlation ID value ensures that when the asynchronous response is actually received, the Composite Application Validation System (CAVS) is able to correlate it to the correct request. If your request message is an Enterprise Business Message (EBM), leave this field blank, as the EBM header ID is automatically used as the correlation ID. In this case, because the EBM header ID is used as the correlation ID, do not use it as a key column in the simulator definition, if applicable. If your request message is not an EBM, you must enter a correlation ID value. This correlation must be based on a unique key of the message. For example, CreateOrder can use Order ID as the correlation ID. Click **Lookup** to access the Choose Request Correlation Id page, where you can select a correlation ID from XPath variables available in the message.
Expected Response Message	The ability to enter response message XML text is available when the Service Type field value is set to **Synchronous** or **Asynchronous two way**. If expected response message XML test was entered on the Create Test page, it is accessible and editable on this page. You can manually enter the response message text on this page, or for test definitions with the Service Type field set to **Synchronous**, you can use the **Get Messages** button to generate a response stub message based on selected endpoint URL and operation values. Entering the expected response message XML facilitates the generation of XPath values, which are used to validate the actual response message returned in the test. You may also choose to manually enter or modify the XPath values directly on the Modify Test Definition page. If you are manually entering XPath values, you do not need to enter response message XML text. When you enter response message XML text on this page, you can click **Generate Xpath** button on the Modify Test Definition page to generate the XPath values that will be used to validate the expected response message you entered on this page against the actual response returned by the test. If the Service Type field value is set to **Synchronous** or **Asynchronous two way**, you may choose to not enter response message XML text in this field. You do not need to enter response message XML if you are manually entering XPath values directly on the Modify Test Definition or if the test you are running does not require validation of the response message. For example, your test may be focused on just populating data. The Expected Response Message text box is unavailable when the Service Type field value is set to **Notify**. In this case, a response message is not a test requirement.
Generate Xpath	Click to generate namespace and XPath values based on available Endpoint URL and Response Message values. After you have generated XPath values, consider deleting any rows that will not be used in the testing effort. The Generate Xpath button is unavailable when the Service Type field value is set to **Notify**. In this case, a response message is not a test requirement.

Table 4–3 (Cont.) Modify Definitions Page - Test Messages Group Box Elements

Element	Description
Response Message Correlation ID	This field only displays for a test definition with the Service Type field value set to **Asynchronous two way**. For this service type, entering a correlation ID value ensures that when the asynchronous response is actually received, the CAVS is able to correlate it to the correct request. If your response message is an EBM, leave this field blank, as the EBM header ID is automatically used as the correlation ID. In this case, because the EBM header ID is used as the correlation ID, do not use it as a key column in the simulator definition, if applicable. If your response message is not an EBM, you must enter a correlation ID value. This correlation must be based on a unique key of the message. For example, CreateOrder can use Order ID as the correlation ID. Click **Lookup** to access the Choose Response Correlation Id page, where you can select a correlation ID from XPath variables available in the message.

Prefix and Namespace Selection

Use the Prefix and Namespace Selection grid to define namespace data that will be used in the XPath values defined in the XPath Selection grid. Elements available in the Prefix and Namespace Selection grid are discussed in Table 4–4.

Table 4–4 Prefix and Namespace Selection Grid Elements

Element	Description
Delete	Select one or more namespace rows and click **Delete** to execute the deletion. This button only appears when namespace rows are present.
Create	Click to manually add and populate a namespace row.
Prefix	Prefix that should be used for the namespace.
Namespace	Namespace to be used in the XPath data for the test definition.

XPath Selection

Use the XPath Selection grid to work with XPath values that are used to compare the actual response message returned in the test to the expected response message defined in the Response Message text box on this page. The values in this grid use the namespace values set in the Prefix and Namespace Selection grid.

A common adjustment you will likely need to make to XPath conditions and expected node values in this grid is to generalize certain specific values, such as EBM IDs. For example, an EBM ID is unique for each transaction, so your test definition will likely not want to specify a particular EBM ID as response criteria. Instead, you may want to generalize the criteria to just verify that the EBM ID is a number greater than zero or use the `Is Valid` condition value.

Note: If you are entering XPath values manually, it is important to maintain correlations with the values entered in the Prefix and Namespace Selection grid. Each XPath node must have a prefix (namespace alias) that has been defined in the Prefix and Namespace Selection grid, unless it is an XPath expression.

The XPath Selection grid is unavailable when the Service Type field value is set to **Notify**. In this case, a response message is not a test requirement.

Elements available in the XPath Selection grid are discussed in Table 4–5.

Table 4–5 XPath Selection Grid Elements

Element	Description
Xpath	When working with a test definition that contains multiple request and response data sets, use the **Xpath** drop-down list box to select the data set you want to use to run the test. For more information about providing multiple data sets in a test definition, see Section 4.3, "How to Provide Multiple Request and Response Message Sets in a Single Test Definition."
Delete	Select one or more XPath rows and click **Delete** to execute the deletion. This button only appears when XPath rows are present.
Create	Click to manually add and populate an XPath row.
XPath Sequence Id	Indicates the sequence of the XPath expressions. This value is required. This value is read-only when it has been generated using the **Generate Xpath** button.
XPath	XPath data to be used in the test definition. These values can include XPath nodes and expressions. This value is read-only when it has been generated using the **Generate Xpath** button.
Condition	Select an available value: ■ **Is Valid:** The value provided in the XPath field is valid and no Expected Node Value is supplied. ■ **Equals To:** The value provided in the XPath field is valid and an Expected Node Value is supplied. ■ **Not Equal To** ■ **Less Than** ■ **Greater Than** ■ **Less Than Equal** ■ **Greater Than Equal** ■ **Not Null** ■ **Expected Node Value** The value expected in the response XML message. When you use the **Generate Xpath** button to generate XPath data, this value may be populated, but can be modified as necessary. The **Condition** field value is used to qualify this value.

Test Instance Selection

Select the **Test Instances** tab to display the Test Instance Selection grid, which displays information about test instances generated using the test definition.

Click **Id** to access the test instance on the Test Instance Detail page.

For more information about the Test Instance Detail page, see Section 9.2, "How to View Test Instance Details."

Linked Simulator Definition Selection

Select the **Simulator Definitions** tab to display the Linked Simulator Definition Selection grid, which displays information about simulator definitions that are linked to the selected test definition.

Available elements on the Simulator Definition tab are discussed in Table 4–6.

Table 4–6 Linked Simulator Definition Selection Grid Elements

Element	Description
Unassign	Select one or more simulator definition rows that you want to disassociate with the test definition. Click **Unassign** to execute the disassociation.
Assign	Click to access the Search Definitions - Simulator page, where you can search for a simulator definition that you want to assign to the test definition. Making this assignment facilitates reporting. After the test definition runs and generates a test instance, all simulator instances generated by the simulator definition associated with the test definition will automatically be linked to the test instance. After you have assigned a simulator definition using the Search Definitions - Simulator page, the Modify Test Definition page appears, and displays the selected simulator definition.
Refresh	Click to refresh the Modify Test Definition page.
Simulator Definition Id	Click for an unlocked simulator definition to access the Modify Simulator Definition page. Click for a locked simulator definition to access the View Simulator Definition page, where you can access a read-only view of the simulator definition.

Group Definition Selection

Select the **Group Definitions** tab to display the Group Definition Selection grid, which displays information about group definitions that include the test definition.

Elements available in the Group Definition Selection grid are discussed in Table 4–7.

Table 4–7 Group Definition Selection Grid Elements

Element	Description
Group Definition Id	Click to access the group definition on the Group Definition Detail page. For more information about the Group Definition Detail page, see Chapter 7, "Working with Group Definitions."
Group Name	Displays the descriptive name assigned to the group definition.
Sequence Id within Group	Displays the sequence in which the test definition is initiated by the group definition.
Cancel	Click to discard any updates you have made and return to the Definitions page.
Actions	Select the action you want to take with the test definition. ▪ **Execute**: Select to execute the test definition. The status of the test execution appears at the top of the page. When a test definition has successfully executed, you can view details of the test instance on the Test Instance Details page. For more information about the Test Instance Details page, see Chapter 9, "Working with Test and Simulator Instances." ▪ **Lock**: Select to lock the test definition and view the test definition on the View Test Definition page. A locked definition cannot be edited. ▪ **Duplicate**: Select to duplicate the test definition. The duplicate definition is created using the exact values of the original, with the exception of being given a unique Id value.

Table 4–7 (Cont.) Group Definition Selection Grid Elements

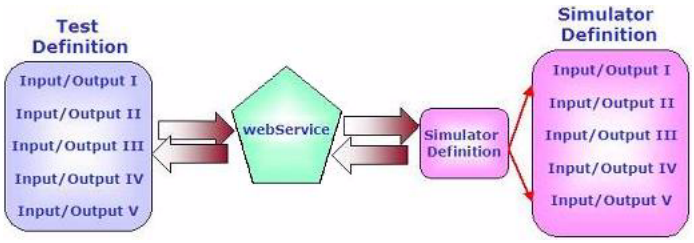
Element	Description
Apply	Click to apply and save any changes you have made to values on the page.
Save	Click to save entries on the page and go to the Definitions page. For more information about the Definitions page, see Chapter 6, "Searching for Test and Simulator Definitions."

4.3 How to Provide Multiple Request and Response Message Sets in a Single Test Definition

You can create a test definition that contains multiple pairs of request and response message data, as shown in Figure 4–8. This means that test definitions only need to be created per usage requirements, not per test data requirements.

For example, if you want to test a process against five sets of test data, you can create a single test definition to test the process and include in it all five sets of test data against which you want the process to operate. This is as opposed to creating five separate test definitions, one per combination of process and set of test data.

Figure 4–8 Providing Multiple Request and Response Message Sets in a Single Test Definition



When multiple sets of test data are included in a test definition, each set will be executed in sequence. Separate test instances will be generated for each set of data. Test instances will reflect the success or failure of each segment of the test run using each set of test data.

Request Message Format

Use the format shown in Example 4–1 to include multiple sets of request data in the test definition.

The `CAVSRequestInputs` and `CAVSRequestInput_1` envelope are autogenerated. Use copy and paste commands to create more sets; `CAVSRequestInput_2` and `CAVSRequestInput_3`, for example.

Example 4–1 Request Message Format

```
<cavs:CAVSRequestInputs
xmlns:cavs="http://schemas.xmlsoap.org/cavs/requestenvelope/">
<cavs:CAVSRequestInput_1>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns1="http://xmlns.oracle.com/SimpleProcess">
    <ns1:SimpleProcessProcessRequest>
   . . .
    </ns1:SimpleProcessProcessRequest>
    </soap:Body>
```

```
</soap:Envelope>
</cavs:CAVSRequestInput_1>

<cavs:CAVSRequestInput_2>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns1="http://xmlns.oracle.com/SimpleProcess">
    <ns1:SimpleProcessProcessRequest>
    . . .
    </ns1:SimpleProcessProcessRequest>
    </soap:Body>
</soap:Envelope>
</cavs:CAVSRequestInput_2>
</cavs:CAVSRequestInputs>
```

Response Message Format

Use the format shown in Example 4–2 to include multiple sets of response data in the test definition.

Example 4–2 Response Message Format

```
<cavs:CAVSResponseOutput_1>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns1="http://xmlns.oracle.com/SimpleProcess">
    <ns1:SimpleProcessProcessResponse>
    . . .
    </ns1:SimpleProcessProcessResponse>
    </soap:Body>
</soap:Envelope>
</cavs:CAVSResponseOutput_1>

<cavs:CAVSResponseOutput_2>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns1="http://xmlns.oracle.com/SimpleProcess">
    <ns1:SimpleProcessProcessResponse>
    . . .
    </ns1:SimpleProcessProcessResponse>
    </soap:Body>
</soap:Envelope>
</cavs:CAVSResponseOutput_2>
</cavs:CAVSResponseOutputs>
```

After entering request and response data sets and clicking the **Generate Xpath** button on the Modify Test Definition page, the XPath Selection grid provides access to the **Please select an Xpath** drop-down list box, where you can select the set of test data you want to use to run the test.

For more information about the Modify Test Definition page, see Section 4.2, "How to Modify a Test Definition."

If your testing scenario includes simulator definitions, you can likewise create simulator definitions that contain multiple request and response message sets that work with the sets defined in your test definition.

For more information, see Section 5.3, "How to Provide Multiple Request and Response Message Sets in a Single Simulator Definition."

5

Creating and Modifying Simulator Definitions

This chapter describes how to create and modify simulator definitions. It also provides instructions for how to provide multiple request and response message sets in a single simulator definition, how to create a simulator definition that supports chatty services, and how to send dynamic responses in a simulator response.

A simulator definition is created by the Composite Application Validation System (CAVS) user to simulate a particular service in an Oracle Application Integration Architecture (AIA) integration or a participating application. A simulator receives data from the tested web service and returns predefined data so that the tested web service can continue processing.

This chapter includes the following sections:

5.1 How to Create a Simulator Definition

To create a simulator definition:

1. Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. Click the **Create Simulator** button. The Create Simulator page displays, as shown in Figure 5–1 and Figure 5–2.

Figure 5–1 Create Simulator Page (1 of 2)

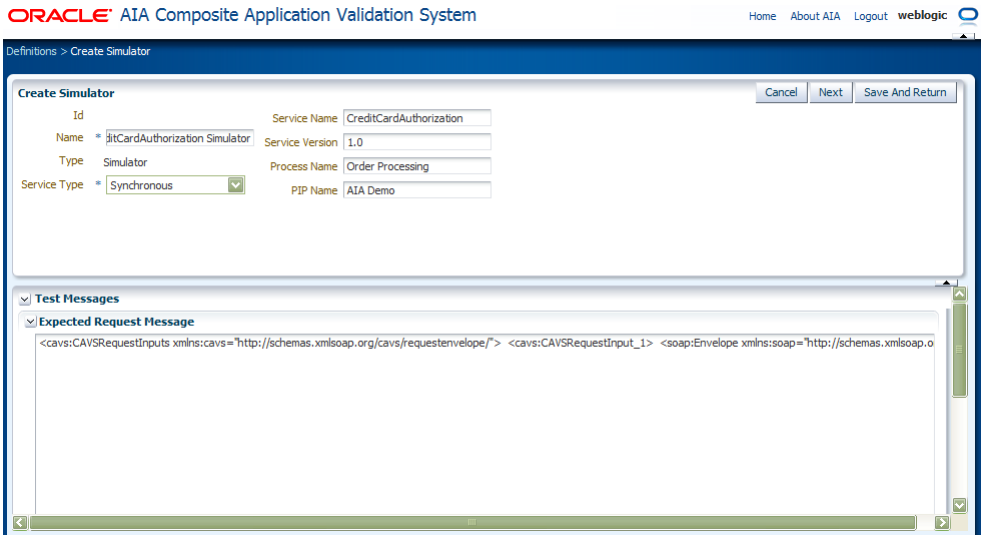
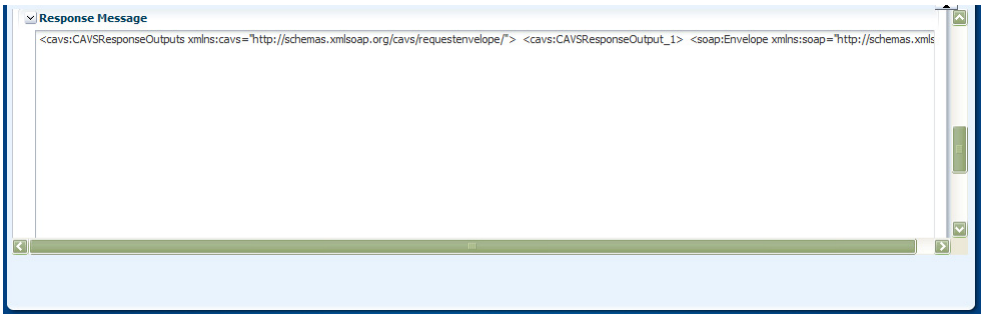


Figure 5–2 Create Simulator Page (2 of 2)



2. Use the page elements on the Create Simulator page to create simulator definitions. Available elements are discussed in Table 5–1.

Table 5–1 Create Simulator Page Elements

Element	Description
Id	Upon saving the simulator definition, a unique key identifier is assigned to the simulator definition.
Name	Enter the descriptive name you want to use for the simulator definition.
Type	Displays the type of definition you have chosen to create. On the Create Simulator page, this value will always be set to **Simulator**.
Service Type	Select the business service pattern of the web service the simulator definition is simulating. ▪ **Synchronous** (request-and-reply) ▪ **Notify** (asynchronous request-only) ▪ **Asynchronous two way**
Service Name	Enter the name of the web service that you want to simulate using the simulator definition.
Service Version	Enter the version of the web service you want simulate using the simulator definition.

Table 5–1 (Cont.) Create Simulator Page Elements

Element	Description
Process Name	Enter the name of the process that includes the web service that you want to simulate using the simulator definition.
PIP Name (Process Integration Pack name)	Enter the name of the PIP that includes the web service that you want to simulate using the simulator definition.
Use WS-Addressing	For more information, see Section 5.2.1, "Using WS-Addressing in Asynchronous Two-Way Simulator Definitions"

Test Messages

Use the Test Messages group box to generate XPath values based on provided request XML message text. By default, SOAP envelope XML text is provided in these fields. You can paste XML text within this default SOAP envelope, or paste your own XML text already enclosed in an envelope into these fields.

For more information about how to create simulator request and response messages that hold multiple sets of test data in a single definition, see Section 5.3, "How to Provide Multiple Request and Response Message Sets in a Single Simulator Definition."

For more information about how to create simulator request and response messages that support chatty service conversations, see Section 5.4, "How to Create a Simulator Definition that Supports Chatty Services."

Available elements in the Test Messages group box are discussed in Table 5–2.

Table 5–2 Test Messages Group Box Elements

Element	Description
Expected Request Message	Entering request message XML text facilitates the generation of XPath values that are used to match a received request with this simulator's expected request, as well as to validate values in this received request message. That is, the XPath values you supply provide a signature for the simulator definition that the simulator service attempts to match with arriving request actions. In addition to enabling the simulator service to match a test request with a simulator definition, the XPath criteria you provide can also serve to validate data sent in the test request. If a simulator has been directly designated for use in the AIAConfigurationProperties.xml using the Routing Configurations page, the simulator definition will be identified directly. However, after the simulator has been identified, there may be multiple requests within it. If so, the XPath key field values provide an efficient search method for request matching. For more information about the Routing Configurations page, see Chapter 8, "Defining CAVS Routing Setup IDs." You can enter expected request message XML text on this page and click the **Generate Xpath** button on the Modify Simulator Definition page to generate XPath values used to validate the actual request sent by the test definition. You may also choose to manually enter or modify the XPath values directly on the Modify Simulator Definition page. You do not need to enter request message XML if you are manually entering XPath values directly on the Modify Simulator Definition page. You may choose to copy and paste messages from the BPEL Console, instead of manually entering them. For more information, see Section 2.5.1, "How to Obtain Message XML Text from a BPEL Process."
Response Message	Entering response message XML text for a simulator definition is required when the Service Type field value is set to **Synchronous** or **Asynchronous two way**. Enter the XML text of the response message that you want to use for the simulator definition. This response message should mimic the actual response message that would be sent by the service that the simulator definition is simulating. This text box is hidden when the Service Type field value is set to **Notify**. In this case, a response message is not a simulator requirement. You may choose to copy and paste messages from the BPEL Console, instead of manually entering them. For more information, see Section 2.5.1, "How to Obtain Message XML Text from a BPEL Process."
Cancel	Click to discard any updates you have made and return to the Definitions page.
Next	Click to save entries on the Create Simulator page and go to the Modify Simulator Definition page, where you can generate XPaths and further edit and manage the simulator definition.
Save	Click to save entries on the Create Simulator page and return to the Definitions page.

5.2 How to Modify a Simulator Definition

To modify a simulator definition:

1. Access the Modify Simulator Definition page, as shown in Figure 5–3, Figure 5–4, Figure 5–5, Figure 5–6, and Figure 5–7.

Access the page using one of the following navigation paths:

- Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. Click the **Create Simulator** button. Enter required values on the Create Simulator page and click **Next**.
- Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. Click an **Id** link for an unlocked simulator definition in the Search Result Selection grid on the Definitions page.
- Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Instances** tab. Click a **Definition Id** link for an unlocked simulator definition on the Instances page.

Figure 5–3 Modify Simulator Definition Page (1 of 5)

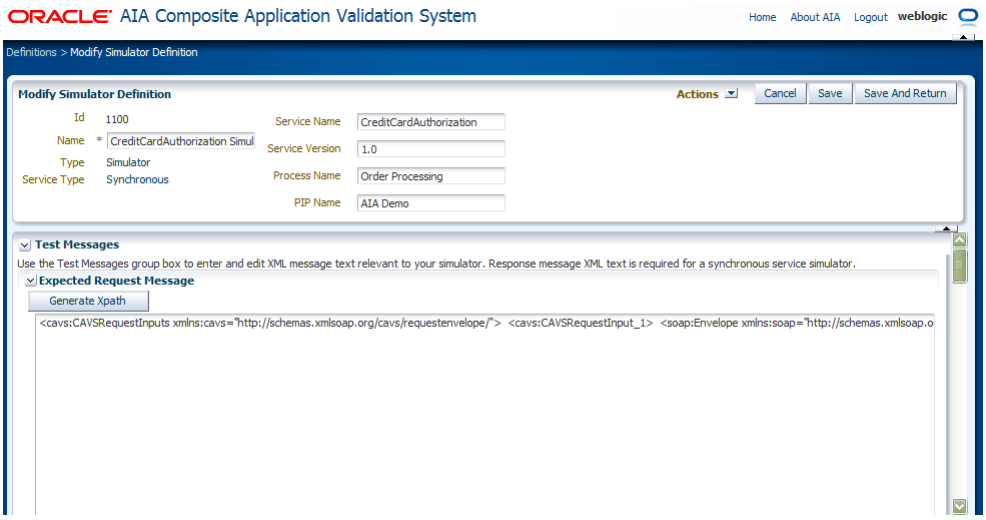


Figure 5–4 Modify Simulator Definition Page (2 of 5)

Figure 5–5 Modify Simulator Definition Page (3 of 5)

Figure 5–6 Modify Simulator Definition Page (4 of 5)

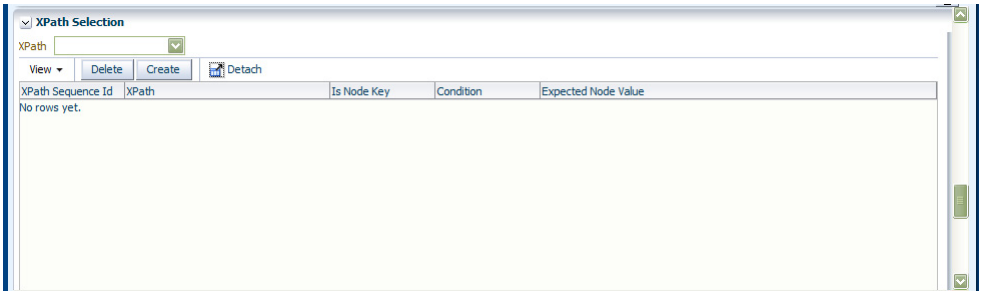
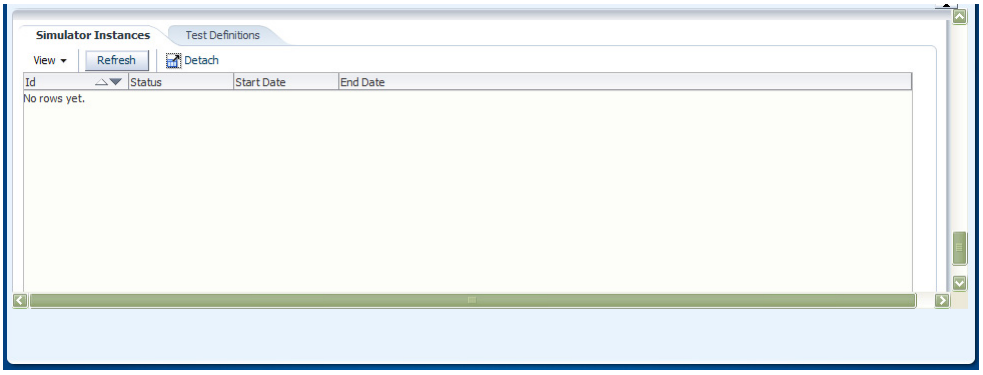


Figure 5–7 Modify Simulator Definition Page (5 of 5)



2. Use the page elements on the Modify Simulator Definition page to modify a simulator definition. The page displays values you defined for the simulator definition. You can modify the values in editable fields.

   Most of the elements on this page also appear on the Create Simulator Definition page and are documented in Section 5.1, "How to Create a Simulator Definition." Any additional elements are discussed here in Table 5–3.

Table 5–3 Modify Simulator Definition Page Elements

Element	Description
Actions	Select the action you want to take with the simulator definition. ■ **Lock**: Select to lock the simulator definition and view the simulator definition on the View Simulator Definition page. A locked definition cannot be edited. ■ **Duplicate**: Select to duplicate the simulator definition. The duplicate definition is created using the exact values of the original, with the exception of being given a unique Id value.
Cancel	Click to discard any updates you have made and return to the Definitions page.
Apply	Click to apply and save any changes you have made to values on the page.
Save	Click to save entries on the page and go to the Definitions page. For more information, see Chapter 6, "Searching for Test and Simulator Definitions."
Callback URL	If you are creating a simulator with a Service Type of **Asynchronous two way**, enter the URL of the web service that should be called back by the simulator.
SOAP Action	If you are creating a simulator with a Service Type of **Asynchronous two way**, enter the operation of the callback URL.
Delay (msec)	If you are creating a simulator with a Service Type of **Asynchronous two way**, enter the number of milliseconds that you want the simulator definition to wait before issuing the call back service invocation. If you are using this simulator along with an asynchronous two-way test definition, ensure that the Delay (msec) value you provide is less than the Time-out (msec) value defined for any test definition. For more information about the Time-out (msec) field, see Section 4.2, "How to Modify a Test Definition."
Use WS-Addressing	Selecting the WS-Addressing check box eliminates the need for you to reconfigure the Callback URL on asynchronous simulator definitions each time you want to call a different endpoint. For more information, see Section 5.2.1, "Using WS-Addressing in Asynchronous Two-Way Simulator Definitions"

Test Messages

For more information about the elements in the Test Messages group box, see Section 5.1, "How to Create a Simulator Definition."

Prefix and Namespace Selection

Use the Prefix and Namespace Selection grid to define namespace data that will be used in the XPath values defined in the XPath Selection grid.

Available elements in the Prefix and Namespace Selection grid are discussed in Table 5–4.

Table 5–4 Prefix and Namespace Selection Grid Elements

Element	Description
Delete	Select one or more namespace rows and click **Delete** to execute the deletion. This button only appears when namespace rows are present.

Table 5–4 (Cont.) Prefix and Namespace Selection Grid Elements

Element	Description
Create	Click to manually add and populate a namespace row.
Prefix	Prefix that should be used for the namespace.
Namespace	Namespace to be used in the XPath data for the simulator definition.

XPath Selection

Use the XPath Selection grid to work with XPath values that are used to match the simulator definition with arriving requests. XPath values can also be used to validate data send in the test request. The values in this grid use the namespace values set in the Prefix and Namespace Selection grid.

> **Note:** If you are entering XPath values manually, it is important to maintain correlations with the values entered in the Prefix and Namespace Selection grid. Each XPath node must have a prefix that has been defined in the Prefix and Namespace Selection grid, unless it is an XPath expression.

Available elements in the XPath Selection grid are discussed in Table 5–5.

Table 5–5 XPath Selection Grid Elements

Element	Description
Delete	Select one or more XPath rows and click **Delete** to execute the deletion. This button only appears when XPath rows are present.
Create	Click to add and manually populate an XPath row.
XPath Sequence Id	Indicates the sequence of the XPath expressions. This value is required. This value is read-only when it has been generated using the Generate Xpath button.
Xpath	XPath value used to help match the simulator definition with arriving requests. These values can include XPath nodes and expressions. This value is read-only when it has been generated using the Generate Xpath button.
Is Node Key	Select if the XPath node is a key value to be used in matching arriving test requests with the simulator.

Table 5–5 (Cont.) XPath Selection Grid Elements

Element	Description
Condition	Select the condition you want to use: ■ **Is Valid**: The value provided in the XPath field is valid and no Expected Node Value is supplied. If you are using WS-Addressing, ensure that all of the WS-Addressing parameters pass the XPath validations. Change the Condition for those expressions to Is Valid. For more information, see Section 5.2.1, "Using WS-Addressing in Asynchronous Two-Way Simulator Definitions". ■ **Equals To**: The value provided in the XPath field is valid and an Expected Node Value is supplied. ■ **Not Equal To** ■ **Less Than** ■ **Greater Than** ■ **Less Than Equal** ■ **Greater Than Equal** ■ **Not Null**
Expected Node Value	The value that the simulator expects to receive from the service that invokes it. When the simulator is actually executed, this value is compared with the actual value based on the validation condition selected in the Condition field When you use the **Generate Xpath** button to generate XPath data, this value may be populated, but can be modified as necessary. The Condition field value is used to qualify this value.

Simulator Instance Selection

Select the **Simulator Instances** tab to display the Simulator Instance Selection grid, which displays information about simulator instances generated using the simulator definition.

Available elements in the Simulator Instance Selection grid are discussed in Table 5–6.

Table 5–6 Simulator Instance Selection Grid

Element	Description
Refresh	Click to refresh the Modify Simulator Definition page.
Id	Click to display the selected instance ID on the Simulator Instances Detail page. For more information about the Simulator Instances Detail page, see Section 9.3, "How to View Simulator Instance Details."
Status	Displays the status of the simulator instance generated by the simulator definition. ■ Initiated: The simulator instance has been initiated. ■ Ended: This status is only applicable to simulator instances that do not involve validations. Indicates that the instance has ended. ■ Faulted: The simulator instance could not execute properly due to exceptions or faults. ■ Failed: The simulator instance did not pass validation. ■ Passed: The simulator instance passed validation.
Start Date	Displays the date and time at which the simulator instance was initiated.

Table 5–6 (Cont.) Simulator Instance Selection Grid

Element	Description
End Date	Displays the date and time at which the simulator instance ended.

Test Definition Selection

Select the **Test Definitions** tab to display the Linked Test Definition Selection grid, which displays information about test definitions associated with the simulator definition.

Available elements in the Linked Test Definition Selection grid are discussed in Table 5–7.

Table 5–7 Linked Test Definition Selection Grid

Element	Description
Delete	Select one or more test definition rows that you want to delete and click **Delete** to execute the deletion.
Assign	Click to access the Search Definitions - Test page, where you can search for a test definition to which you want to assign the simulator definition.
Refresh	Click to refresh the Modify Simulator Definition page.

5.2.1 Using WS-Addressing in Asynchronous Two-Way Simulator Definitions

This section describes how to configure a WS-Addressing enabled simulator definition for the Asynchronous Two-Way Message Exchange Pattern.

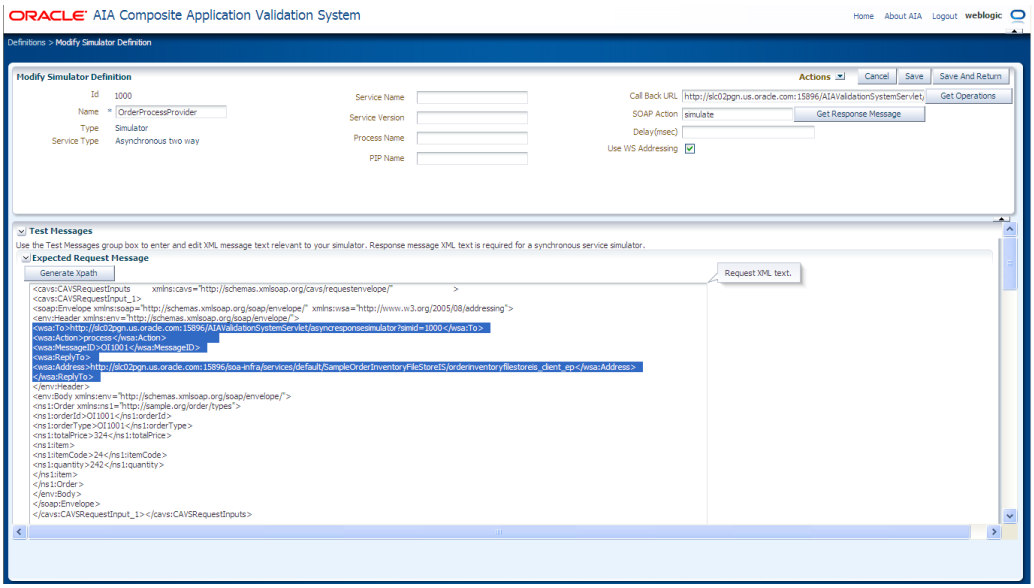
Adding support for Web Services Addressing (WS-Addressing) eliminates the need for you to reconfigure the Callback URL on asynchronous simulator definitions each time you want to call a different endpoint.

When you create or modify a simulator definition, you can use the **Use WS-Addressing** check box to pull the WS-Address from the request and post the delayed response to the specified address in the Callback URL. This enables you to avoid the manual task of modifying the Callback URL with context information every time there is a change or redeployment.

In an asynchronous request/reply pattern, a consumer that sends a request to the provider as part of a request SOAP header can populate the replyToAddress to whom the provider has to send the response, shown in Figure 5–8, Figure 5–9, Figure 5–10, Figure 5–11, and Figure 5–12.

The fields for these pages are defined in Section 5.2, "How to Modify a Simulator Definition".

Figure 5–8 Modify Simulator Definition Page for WS-Addressing (1 of 5)



The highlighted text in Figure 5–8 and Figure 5–9 shows the replyToAddress and SOAP header when using WS-Addressing.

Figure 5–9 Modify Simulator Definition Page for WS-Addressing (2 of 5)

Figure 5–10 Modify Simulator Definition Page for WS-Addressing (3 of 5)

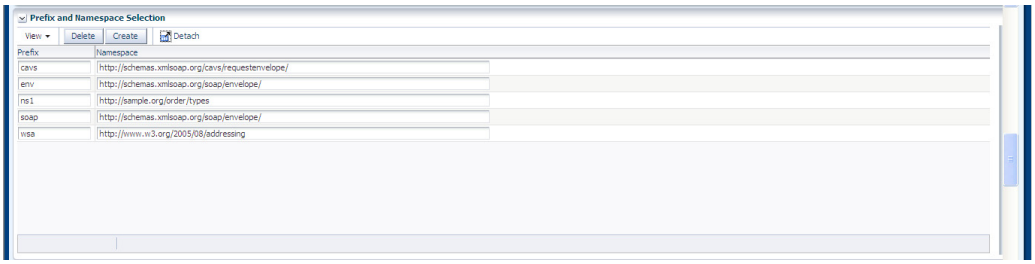
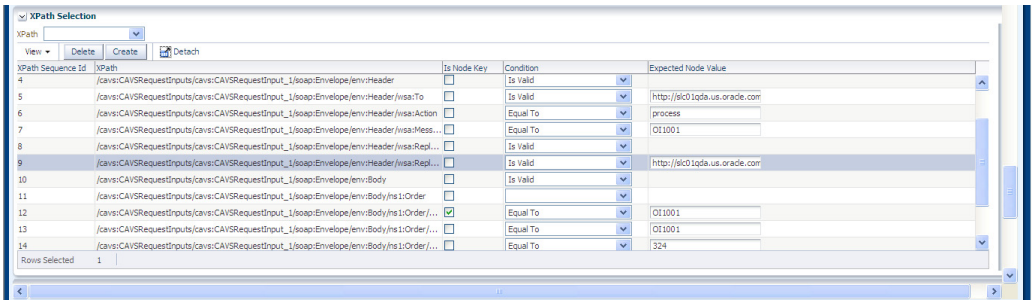
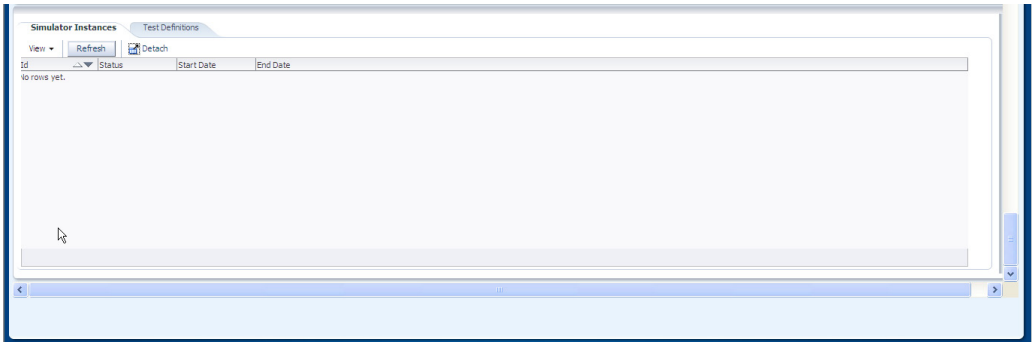


Figure 5–11 Modify Simulator Definition Page for WS-Addressing (4 of 5)



If you are using WS-Addressing, ensure that all of the WS-Addressing parameters pass the XPath validations. Change the Condition for those expressions to Is Valid. See Figure 5–11.

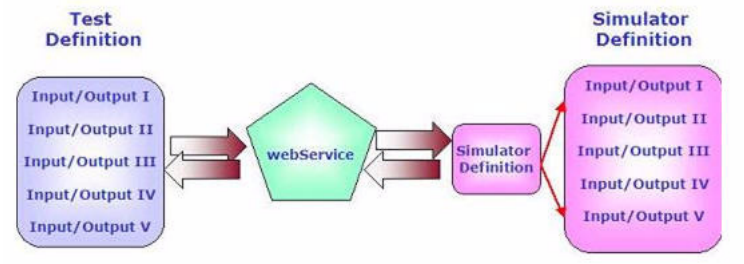
Figure 5–12 Modify Simulator Definition Page for WS-Addressing (5 of 5)



5.3 How to Provide Multiple Request and Response Message Sets in a Single Simulator Definition

You can create a simulator definition that contains multiple pairs of request and response message data, as shown in Figure 5–13. This means that simulator definitions only need to be created per usage requirements, not per test data requirements.

Figure 5–13 Providing Multiple Request and Response Message Sets in a Single Simulator Definition



For example, if you want to simulate a service against five sets of test data, you can create a single simulator definition to simulate the service and include in it all five sets

of test data with which you can the service to operate. This is as opposed to creating five separate simulator definitions, one per combination of service and set of test data.

When a simulator definition that includes multiple test data sets is invoked, the appropriate data set is matched for use based on key attributes identified in the request. At this point, the request validation and response provision can occur. Since we would typically use such definitions to handle several sets of data, it is recommended that you choose the same key values for every set of data.

Request Message Format

Use the format provided in Example 5–1 to include multiple sets of request data in the simulator definition.

The `CAVSRequestInputs` and `CAVSRequestInput_1` envelopes are autogenerated upon the input of the endpoint URL value on the test definition. Use copy and paste commands to create more sets, such as `CAVSRequestInput_2` and `CAVSRequestInput_3`.

Example 5–1 Request Message Format

```
<cavs:CAVSRequestInputs
xmlns:cavs="http://schemas.xmlsoap.org/cavs/requestenvelope/">
<cavs:CAVSRequestInput_1>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns1="http://xmlns.oracle.com/SimpleProcess">
    <ns1:SimpleProcessProcessRequest>
. . .
    </ns1:SimpleProcessProcessRequest>
    </soap:Body>
</soap:Envelope>
</cavs:CAVSRequestInput_1>

<cavs:CAVSRequestInput_2>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns1="http://xmlns.oracle.com/SimpleProcess">
    <ns1:SimpleProcessProcessRequest>
. . .
    </ns1:SimpleProcessProcessRequest>
    </soap:Body>
</soap:Envelope>
</cavs:CAVSRequestInput_2>
</cavs:CAVSRequestInputs>
```

Response Message Format

Use the format shown in Example 5–2 to include multiple sets of response data in the simulator definition.

Example 5–2 Response Message Format

```
<cavs:CAVSResponseOutput_1>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns1="http://xmlns.oracle.com/SimpleProcess">
    <ns1:SimpleProcessProcessResponse>
. . .
    </ns1:SimpleProcessProcessResponse>
    </soap:Body>
</soap:Envelope>
</cavs:CAVSResponseOutput_1>
```

```
<cavs:CAVSResponseOutput_2>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns1="http://xmlns.oracle.com/SimpleProcess">
    <ns1:SimpleProcessProcessResponse>
. . .
    </ns1:SimpleProcessProcessResponse>
    </soap:Body>
</soap:Envelope>
</cavs:CAVSResponseOutput_2>
</cavs:CAVSResponseOutputs>
```

Envelope text is prepopulated. Enter actual message content within appropriate tags provided within the envelopes.

After entering request and response data sets and clicking the **Generate Xpath** button on the Modify Simulator Definition page, the XPath Selection grid provides access to available XPath values and enables you to select the XPaths that must be treated as key nodes.

For more information about the Modify Simulator Definition page, see Section 5.2, "How to Modify a Simulator Definition."

If your testing scenario includes test definitions, you can likewise create test definitions that contain multiple request and response message sets that work with the sets defined in your simulator definition.

For more information, see Section 4.3, "How to Provide Multiple Request and Response Message Sets in a Single Test Definition."

5.4 How to Create a Simulator Definition that Supports Chatty Services

You can create a simulator definition that can simulate multiple services, each with a different schema.

In general, we recommend that you create simulators that simulate a single specific service. However, in the case of chatty conversations, for the ease of maintenance, you may choose to simulate all callouts of an Application Business Connector Service (ABCS) using a single simulator definition.

Using this method, you have the advantage of using one simulator for a particular ABCS, regardless of the number of callouts that need to be made. This method also provides ease of maintenance because linked callouts can all be viewed and modified in one place.

For example, in some integration scenarios, participating applications do not provide services at the same level of granularity as operations in Enterprise Business Services (EBSs). In these scenarios, a requester ABCS may need to adopt patterns such as message enrichment, splitting, and aggregation and disaggregation as required by an EBS. Likewise, a provider ABCS may need to adopt patterns as required by participating application services.

These ABCSs, which are typically implemented using BPEL process, call out to several services. To test this chatty ABCS using CAVS, there will likely be a need to replace the services that the ABCS calls out to with several simulators. It will also be required that these multiple request/response simulators be correlated, so that they accurately emulate the transaction of the same entity.

When this type of simulator is called, CAVS initiates the following general flow:

1. Selects simulator definition.

2. Validates the first request message based on the selected simulator.
3. Returns the appropriate response message, if the selected simulator is a two-way simulator.
4. Repeats steps 2 and 3 until the chatty service conversation is complete.

Request Message Format

Use the format shown in Example 5–3 to create a simulator definition that supports chatty service conversations. This format provides the ability to specify a set of request and response messages, along with success criteria for each of them. This format is the same as that used for multiple requests and responses in a simulator definition. However, in this case, the schemas for each set will be different.

Example 5–3 Request Message Format

```
<cavs:CAVSRequestInputs
xmlns:cavs="http://schemas.xmlsoap.org/cavs/requestenvelope/">
<cavs:CAVSRequestInput_1>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns1="http://xmlns.oracle.com/Service1">
    <ns1:Service1Request>
. . .
    </ns1: Service1Request>
    </soap:Body>
</soap:Envelope>
</cavs:CAVSRequestInput_1>

<cavs:CAVSRequestInput_2>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body xmlns:ns2="http://xmlns.oracle.com/Service2">
    <ns2: Service2Request>
. . .
    </ns2: Service2Request>
    </soap:Body>
</soap:Envelope>
</cavs:CAVSRequestInput_2>
```

After you have provided request and response messages, click the **Generate Xpath** button on the Modify Simulator Definition page to generate XPath values. Modify the generated XPath values, if necessary.

For more information about the Modify Simulator Definition page, see Section 5.2, "How to Modify a Simulator Definition."

When this type of simulator is called, separate simulator instances are created for each request and response pair. The evaluation of actual response versus expected response is handled per instance created for the same simulator definition.

5.5 How to Send Dynamic Responses in a Simulator Response

CAVS simulator definitions are actually predefined request and response message sets. In some cases, you may not know the values for all the fields in the request message. Additionally, you may want to send these unknown dynamic values in a response to the service that called the simulator.

For example, consider the Enterprise Business Message (EBM) ID. This value is normally generated on the fly by AIA services. If you create a simulator that talks to

this AIA service, you do not have a way to validate the value in the EBM ID field of the request message because the value is dynamically generated.

You may to choose to avoid validations of this value by setting the CAVS XPath validation for the EBM ID field to **isValid**. However, you may have a requirement in which you need to send this dynamic value back in a particular field of the simulator response. To meet this requirement, you can let the simulator pick the particular field (such as EBM ID) in the request and send it back as a field in the response.

To send a dynamic response in a simulator response:

1. Map a field from the request message and add it to the response message. These are two valid formats you can use:
   - `#@#XPATH.{copy the XPath from request msg Ex./soap:Envelop/soap:Body..}#@#`
   - `#@#SYSTEM.{SYSDATE}#@#`
2. Before sending the response, the simulator will pick up this ID from the generated XPath, substitute the actual value, and send it in the response.

   The strings referenced above will form a part of the response message. To know what the request message XPath values are, use the output that was generated by clicking the **Generate Xpath** button.

   For example, let's say that the request SOAP message has the nodes shown in Example 5–4:

Example 5–4 Request SOAP Message Nodes

```
<corecom:PersonName>
    <corecom:FirstName>CAVS</corecom:FirstName>
    <corecom:MiddleName>FP</corecom:MiddleName>
    <corecom:FamilyName>AIA</corecom:FamilyName>
    <corecom:CreationDateTime></corecom:CreationDateTime>
</corecom:PersonName>
```

You would define your response SOAP message as shown in Example 5–5:

Example 5–5 Response SOAP Message

```
<corecom:PersonName>

<corecom:FirstName>#@#XPATH.{/soap:Envelope/soap:Body/corecom:CreateCustomerParty
    ListEBM/ebo:DataArea/ebo:CreateCustomerPartyList/

corecomx:Contact/corecomx:PersonName/corecomx:FamilyName}#@#2dot1</corecom:FirstNa
me>

<corecom:MiddleName>#@#XPATH.{/soap:Envelope/soap:Body/corecom:CreateCustomerParty
ListEBM/ebo:DataArea/ebo:CreateCustomerPartyList/corecomx:Contact/corecomx:Person
Name/corecomx:MiddleName}#@#</corecom:MiddleName>

<corecom:FamilyName>#@#XPATH.{/soap:Envelope/soap:Body/corecom:CreateCustomerParty
ListEBM/ebo:DataArea/ebo:CreateCustomerPartyList/corecomx:Contact/corecomx:Person
Name/corecomx:FirstName}#@#</corecom:FamilyName>
    <corecom:CreationDateTime>#@#SYSTEM.{SYSDATE}#@#</corecom:CreationDateTime>
</corecom:PersonName>
```

In this case, the response would be modified by the CAVS engine by copying values from the request as shown in Example 5–6.

Example 5–6 Response Message Modified by CAVS

```
<corecom:PersonName>
    <corecom:FirstName>AIA2dot1</corecom:FirstName>
    <corecom:MiddleName>FP</corecom:MiddleName>
    <corecom:FamilyName>CAVS</corecom:FamilyName>
    <corecom:CreationDateTime>2008-05-12T15:26:43+05:30</corecom:CreationDateTime>
</corecom:PersonName>
```

Note: `2dot1` is a static string that is always appended to the FamilyName value.

6

Searching for Test and Simulator Definitions

This chapter describes how to search for and work with test and simulator definitions.

This chapter includes the following section Section 6.1, "How to Search for and Work with Test and Simulator Definitions."

6.1 How to Search for and Work with Test and Simulator Definitions

To search for and work with test and simulator definitions:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. The **Definitions** page displays, as shown in Figure 6–1 and Figure 6–2.

Figure 6–1 Definitions Page (1 of 2)

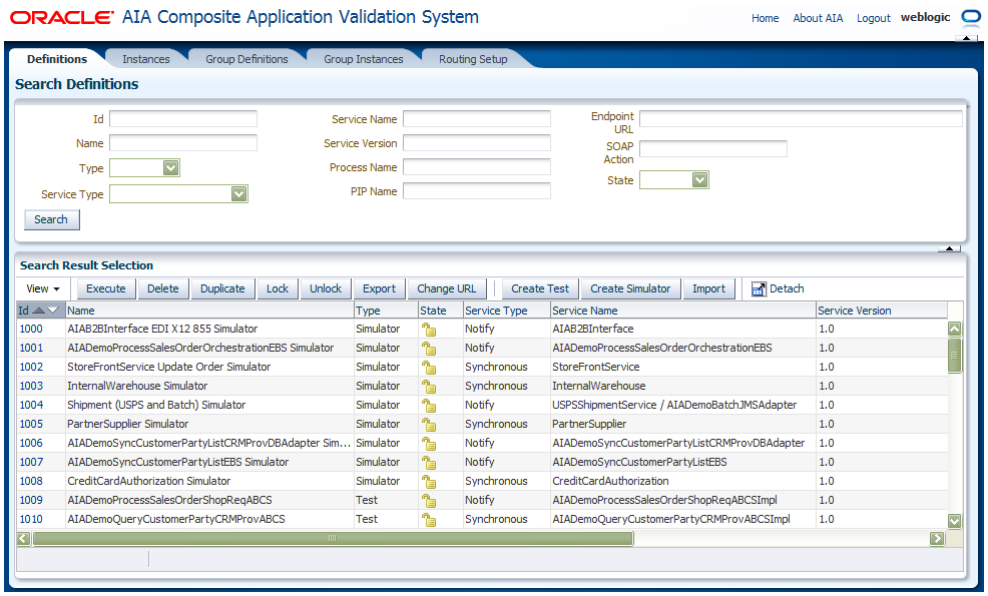
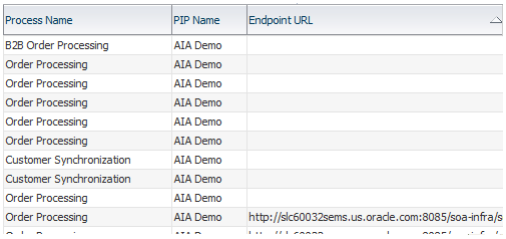


Figure 6–2 Definitions Page (2 of 2)



2. Use the page elements on the Definitions page to search for, execute, migrate, and manage existing test and simulator definitions. You can also access pages you can use to create and modify test and simulator definitions.

Search Definitions

Use the Search Definitions group box to enter search criteria to find the test or simulator definition you are searching for.

Available elements in the Search Definitions group box are discussed in Table 6–1.

Table 6–1 Search Definitions Group Box Elements

Element	Description
Id	Enter the unique key identifier assigned to the test or simulator definition.
Name	Enter the descriptive name assigned to the test or simulator definition.
Type	Select the type of definition for which you are searching: ■ **<Value Not Selected>:** Select to display all definition types. ■ **Test** ■ **Simulator**
Service Type	Select the business service pattern of the web service for which the definition was created: ■ **<Value Not Selected>:** Select to display definitions for all service types. ■ **Synchronous** ■ **Notify** ■ **Asynchronous two way**
Service Name	Enter the name of the web service for which the definition was created.
Service Version	Enter the version of the service for which the definition was created. This is the web service whose URL is provided in the Endpoint URL field.
Process Name	Enter the name of the process that includes the web service for which the definition was created.
PIP Name (Process Integration Pack name)	Enter the name of the Process Integration Pack that includes the web service for which the definition was created.
Endpoint URL	Enter the URL of the web service for which the definition was created.
SOAP Action	Enter the operation called by the web service for which the definition was created.

Table 6–1 (Cont.) Search Definitions Group Box Elements

Element	Description
State	Select the state of the definition: ■ **<Value Not Selected>:** Select to display definitions in all states. ■ **Locked** ■ **Unlocked**
Search	Click to execute a search for definitions using the search criteria entered in the Search Definitions group box.

Search Result Selection

Use the Search Result Selection grid to work with definitions returned in your search results. Upon accessing this page, the grid displays all definitions.

Available elements in the Search Result Selection grid are discussed in Table 6–2.

Table 6–2 Search Result Selection Grid Elements

Element	Description
Execute	Select one or more test definitions that you want to run and click **Execute** to execute the test definition. When a test definition has successfully executed, you can view details of the test instance generated by the test execution on the Test Instance Details page. For more information about the Test Instance Details page, see Section 9.2, "How to View Test Instance Details." Simulator definitions cannot be executed.
Delete	Select one or more definitions that you want to delete and click **Delete** to execute the deletion.
Duplicate	Select one or more definitions that you want to duplicate and click **Duplicate** to execute the duplication. The duplicate definition is created using the exact values of the original, with the exception of being assigned a unique ID value.
Lock	Select one or more definitions that you want to lock and click **Lock** to lock the definitions. A definition with its State value set to Locked cannot be edited.
Unlock	Select one or more definitions that you want to unlock and click **Unlock** to unlock the definitions. An unlocked definition can be edited. A definition with its State value set to Unlocked is editable.
Export	For more information about exporting definitions and instances, see Chapter 12, "Exporting and Importing CAVS Definitions and Instances."
Change URL	Select one or more test definitions for which you want to change the endpoint URL value. Click **Change URL** to launch a pop-up window in which you can enter the new endpoint URL value that you want to use for the selected test definitions.
Create Test	Click to access the Create Test page, where you can create a test definition. For more information about the Create Test page, see Section 4.1, "How to Create a Test Definition."
Create Simulator	Click to access the Create Simulator page, where you can create a simulator definition. For more information about the Create Simulator page, see Section 5.1, "How to Create a Simulator Definition."

Table 6–2 (Cont.) Search Result Selection Grid Elements

Element	Description
Import	For more information about importing test definitions, see Chapter 12, "Exporting and Importing CAVS Definitions and Instances."
Id	Click for an unlocked test definition to access the Modify Test Definition page. Click for a locked test definition to access the View Test Definition page, where you can access a read-only view of the test definition. For more information, see Section 4.2, "How to Modify a Test Definition." Click for an unlocked simulator definition to access the Modify Simulator Definition page. Click for a locked simulator definition to access the View Simulator Definition page, where you can access a read-only view of the simulator definition. For more information, see Section 5.2, "How to Modify a Simulator Definition."

7

Working with Group Definitions

This chapter describes how to create, modify, and work with group definitions.

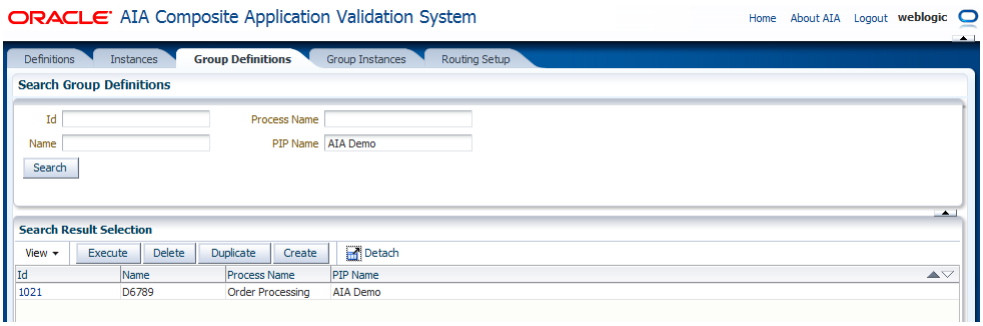
This chapter includes the following sections:

7.1 How to Work with Group Definitions

To work with group definitions:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Group Definitions** tab. The Group Definitions page displays as shown in Figure 7–1.

Figure 7–1 Group Definitions Page



2. Use the page elements on the Group Definitions page to search for, execute, and manage existing group definitions. You can also access a page you can use to create and modify group definitions.

Search Group Definitions

Use the Search Group Definitions group box to enter search criteria to find the group definition you are searching for.

Available elements in the Search Group Definitions group box are discussed in Table 7–1.

Table 7–1 Search Group Definitions Group Box Elements

Element	Description
Id	Enter the unique key identifier assigned to the group definition.
Name	Enter the descriptive name assigned to the group definition.
Process Name	Enter the name of the process associated with the group definition.
PIP (Process Integration Pack) Name	Enter the name of the Process Integration Pack (PIP) associated with the group definition.
Search	Click to execute a search for group definitions using the search criteria entered in the Search Group Definitions group box.

Search Result Selection

Use the Search Result Selection grid to work with group definitions returned in your search results. Upon accessing this page, the grid is populated by all group definitions.

Available elements in the Search Result Selection grid are discussed in Table 7–2.

Table 7–2 Search Result Selection Grid Elements

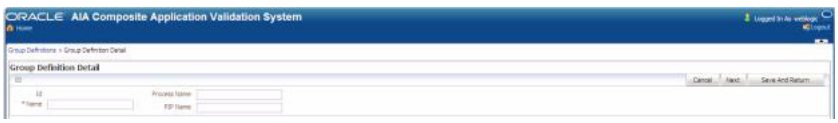
Element	Description
Execute	Select one or more group definitions that you want to run and click **Execute** to execute the group definition. When a group definition has successfully executed, you can view details of the group instance on the Group Instances Detail page. For more information about the Group Instances Detail page, see Section 10.2, "How to View Group Instance Details."
Delete	Select one or more group definitions that you want to delete and click **Delete** to execute the deletion.
Duplicate	Select one or more group definitions that you want to duplicate and click **Duplicate** to execute the duplication. The duplicate group definition is created using the exact values of the original, with the exception of being given a unique Id value.
Create	Click to access the Group Definition Detail page, where you can create a group definition
Id	Click to access the Group Definition Detail page.

7.2 How to Create and Modify a Group Definition

To create a group definition:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Group Definitions** tab. Click the **Create** button. The Group Definition Detail page for a new group definition displays as shown in Figure 7–2.

Figure 7–2 Group Definition Detail Page (New Group Definition)

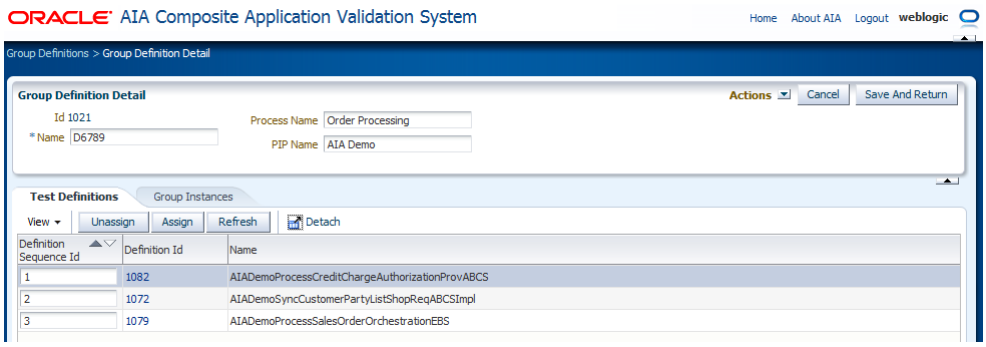


2. Use the page elements on the Group Definition Detail page, as discussed in Table 7–3, to create a group definition that combines one or more tests and executes them in a single-threaded batch sequence.
3. Click **Next** to save entries and display further group definition details elements, discussed in Table 7–3, on the Group Definition Detail page for the newly created and existing group definitions.

To modify a group definition:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Group Definitions** tab. Select a group definition **Id** link. The Group Definition Detail page for existing group definitions displays as shown in Figure 7–3.

Figure 7–3 Group Definition Detail Page (Existing Group Definition)



2. Use the page elements on the Group Definition Detail page, as discussed in Table 7–3, to modify an existing group definition.

Table 7–3 Group Definition Detail Page Elements for New and Existing Definitions

Element	Description
Actions	Select the action you want to take with the group definition. ▪ **Execute**: Select to execute the group definition. When a group definition has successfully executed, you can view details of the test instance on the Group Instances page. ▪ **Duplicate**: Select to duplicate the group definition. The duplicate definition is created using the exact values of the original, with the exception of being given a unique Group Definition Id value.
Cancel	Click to discard updates to the page and return to the Group Definitions page.
Next	For a new group definition, click to save entries and display further group definition details on the Group Definition Detail page. This button does not appear for existing group definitions.
Apply	Click to apply and save any changes you have made to values on the page.
Save	Click to save entries on the Group Definition Detail page and return to the Group Definitions page.

Table 7–3 (Cont.) Group Definition Detail Page Elements for New and Existing

Element	Description
Id	Upon saving a new group definition, a unique key identifier is assigned to the group definition. For an existing group definition, displays the unique key identifier assigned to the group definition.
Name	For a new group definition, enter a descriptive name for the group definition. For an existing group definition, displays the descriptive name assigned to the group definition.
Process Name	For a new group definition, enter the name of the process you want to associate with the group definition. For an existing group definition, displays the process name associated with the group definition. This value is editable.
PIP (Process Integration Pack) Name	For a new group definition, enter the name of the PIP you want to associate with the group definition. For an existing group definition, displays the PIP associated with the group definition. This value is editable.

Test Definition Selection

Select the **Test Definitions** tab to access the Test Definition Selection grid, where you can associate test definitions with the group definition.

Available elements in the Test Definition Selection grid are discussed in Table 7–4.

Table 7–4 Test Definition Selection Grid Elements

Element	Description
Unassign	Select one or more test definition rows that you want to disassociate from the group definition. Click **Unassign** to execute the disassociation.
Assign	Click to access the Search Definitions - Test page, where you can search for a test definition that you want to assign to the simulator definition.
Refresh	Click to refresh the Group Definition Detail page.
Definition Sequence Id	Displays the sequence in which the test definition is initiated by the group definition.
Definition Id	Click for an unlocked test definition to access the Modify Test Definition page. Click for a locked test definition to access the View Test Definition page, where you can access a read-only view of the test definition. For more information, see Section 4.2, "How to Modify a Test Definition."

Group Instance Selection

Select the **Group Instances** tab to display the Group Instance Selection grid, which displays information about group instances generated by the group definition.

Available elements in the Group Instances Selection grid are discussed in Table 7–5.

Table 7–5 Group Instance Selection Grid Elements

Element	Description
Refresh	Click to refresh the Group Definition Detail page.

Table 7–5 (Cont.) Group Instance Selection Grid Elements

Element	Description
Id	Click to access the Group Instances Detail page.
Start Date	Displays the date and time at which the group instance was initiated.

8

Defining CAVS Routing Setup IDs

This chapter provides an introduction to CAVS routing setup IDs, and how to create, search, and modify them. It also provides information about setting up routing configurations without creating routing setup IDs.

Composite Application Validation System (CAVS) routing setups are used when CAVS test definitions call services that in turn, call CAVS simulators and when actual applications and services call CAVS simulators instead of calling subsequent actual services.

This chapter includes the following sections:

8.1 Introduction to CAVS Routing Setup IDs

CAVS routing setup IDs are used to route the service calls to the CAVS simulators. Use the pages covered in this chapter to set up CAVS routing setup IDs before executing tests. These CAVS routing setup IDs are stored as RouteToCAVS properties in the AIAConfigurationProperties.xml file in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config. This file is read during run time to determine whether routing needs to be made to a CAVS simulator or to an actual system.

For example, you could create three routing setup IDs for the scenarios illustrated below.

- To test the requester Application Business Connector Service (ABCS) or when the provider ABCS is not available, you would want the requester ABCS to call a simulator instead of actual Oracle AIA services. For this scenario, create a routing setup ID to set the RouteToCAVS property to *TRUE* on the requester ABCS. This will ensure that the message is routed to the CAVS simulator, as indicated in red.

> **Note:** An actual participating application or test definition can be used to invoke the requester ABCS.

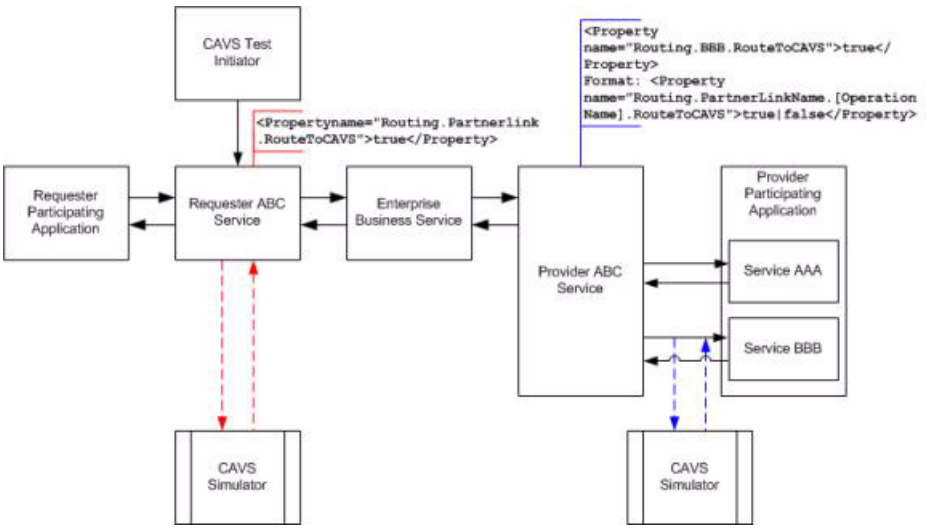
- To test the provider ABCS or when the provider application is not available, you would want the provider ABCS to call a simulator instead of an actual provider application service. For this scenario, create a routing setup ID to set the RouteToCAVS property to *TRUE* on the provider ABCS. This will ensure that the message is routed to the CAVS simulator, as indicated in blue.

  Note: If there is more than one callout from the provider ABCS, the CAVS user can have fine-grained control over the routing by setting the routing at the PartnerLink level (and optionally at the operation level). This is indicated in the figure.

- To test the requester ABCS and the provider ABCS together, you would create a routing setup ID to set the RouteToCAVS property to *FALSE* on the requester ABCS so that it can go on to call the provider ABCS and *TRUE* on the provider ABCS.

Figure 8–1 helps to illustrate the need for different routing setup IDs to test each of these three scenarios. When creating test definitions that will be used to initiate these test scenarios, CAVS enables you to associate the test definition with a specific routing setup ID. This routing setup ID determines the configuration that is required and automatically applies it before executing the test.

Figure 8–1 Sample Scenarios for Using CAVS Routing Setup IDs



For example, if these three test scenarios are grouped into a single test group for execution, each test requires a different routing setup. In this case, you would create three routing setup IDs, 1001, 1002, and 1003, for example.

Each routing setup ID is required by one of the scenarios. You assign routing setup ID 1001 to the test definition for scenario 1, 1002 to the test definition for scenario 2, and so forth. When these three test definitions are executed as a part of the test group, the CAVS system automatically applies routing setup IDs 1001, 1002, and 1003 when executing the appropriate test definition. This eliminates the need to manually modify routing configurations between test scenario executions.

If, for example, you did not associate routing setup ID 1002 with the test definition for scenario 2, the test definition for scenario 2 would use routing setup ID 1001, because it was the last applied routing setup ID.

For more information about assigning a routing setup ID to a test definition, see Section 4.1, "How to Create a Test Definition."

Another option for applying routings is to directly modify them on the Configuration page.

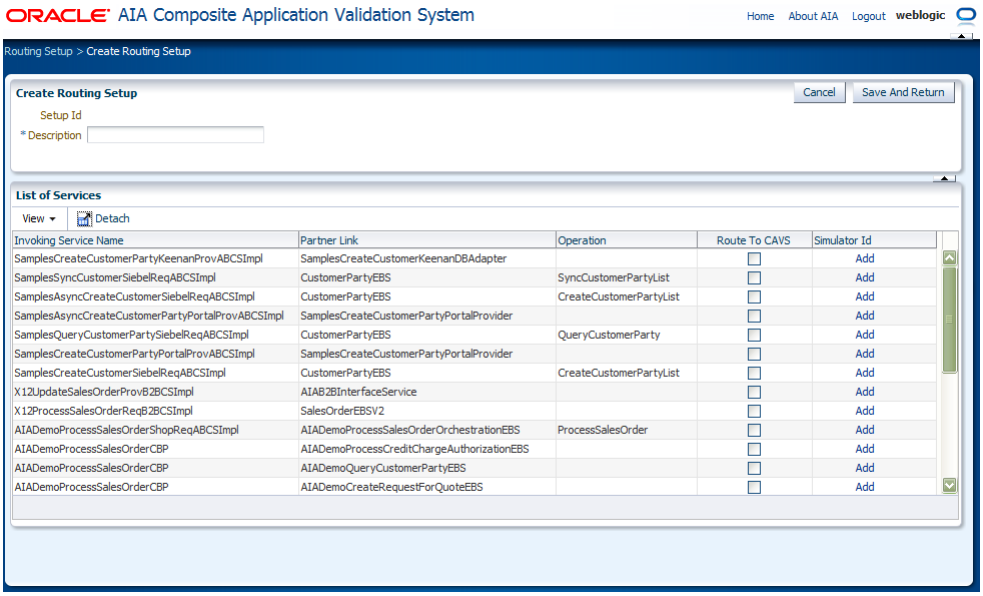
For more information about the Configuration page, see Section 8.5, "How to Set Up CAVS Routing Configurations Without Creating Routing Setup IDs."

8.2 How to Create CAVS Routing Setup IDs

To create CAVS routing setup IDs:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Routing Setup** tab. Click the **Create** button. The Create Routing Setup page displays as shown in Figure 8–2.

Figure 8–2 Create Routing Setup Page



2. Upon access, the Create Routing Setup page displays routing information for all services with a RoutetoCAVS property defined in the AIAConfigurationProperties.xml file in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config.

   Use this page to perform a one-time setup of routing setup IDs that you can later associate with test definitions using the SetupId field on the Create Test page. By making this association, the required routing setup will be automatically applied during the execution of the test definition.

   For more information about the SetupId field, see Section 4.1, "How to Create a Test Definition."

   Data saved on this page is stored in a CAVS table, rather than in the AIAConfigurationProperties.xml file.

For more information about how to quickly define a routing configuration that is stored in AIAConfigurationProperties.xml, see Section 8.5, "How to Set Up CAVS Routing Configurations Without Creating Routing Setup IDs."

Use the page elements on the Create Routing Setup page to create a new CAVS routing.

Available elements on the Create Routing Setup page are discussed in Table 8–1.

Table 8–1 Create Routing Setup Page Elements

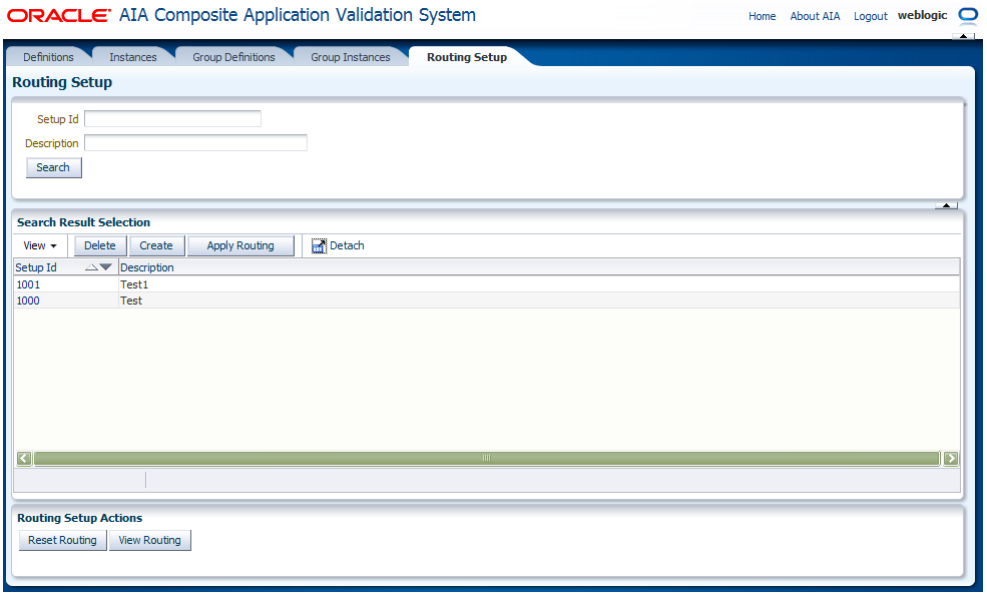
Element	Description
SetupId	Upon saving, a sequentially generated ID is assigned to the routing setup ID.
Description	Enter a description of the routing setup ID you are creating.
InvokingServiceName	Lists all services defined in the AIAConfigurationProperties.xml file in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
PartnerLink	The PartnerLink that is invoked by the service that you want to route to the CAVS simulator.
Operation	The operation of the PartnerLink that you want to route to the CAVS simulator. Displays a value only when multiple operations on the service are invoked using the same PartnerLink, typically when calling an Enterprise Business Service.
RouteToCavs	Select to indicate that the invoking service should route to the selected CAVS simulator.
SimulatorId	Click **Add** to access the Search Definitions page, where you can select the simulator definition that you want an invoking service to route to. Upon access, the page displays all available CAVS simulator definition IDs. Select the simulator definition to which you want to route an invoking service and click the **Select** button. If a simulator definition has already been selected, the simulator ID displays. Click **Modify** to select a different simulator ID. Click **Clear** to clear the selection.

8.3 How to Search for CAVS Routing Setup IDs

To search for CAVS routing setup IDs:

1. Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Routing Setup** tab. The Routing Setup page displays, as shown in Figure 8–3.

Figure 8–3 Routing Setup Page



2. Use the page elements on the Routing Setup page to search for an existing CAVS routing setup ID, or access functionality to create and delete routings.

   Available elements on the Routing Setup page are discussed in Table 8–2.

Table 8–2 Routing Setup Page Elements

Element	Description
SetupId	Enter the ID assigned to the routing setup ID you are searching for.
Description	Enter description text used for the routing setup ID you are searching for.
Search	Click to execute a search for routing setup IDs using the search criteria entered in the Search Routing Setups group box.
Delete	Select one or more routing setup IDs that you want to delete and click **Delete** to execute the deletion.
Create	Click to access the Create Routing Setup page, where you can create a routing setup ID. For more information, see Section 8.2, "How to Create CAVS Routing Setup IDs."
Apply Routing	After you have created a new routing setup ID, you may apply it to populate the AIAConfigurationProperties.xml file. To do this, select a single routing setup ID and click **Apply Routing**. If you apply the routing setup ID to the AIAConfigurationProperties.xml file, it becomes a routing configuration that is applied in all executions of the associated invoking service, not just when the routing setup ID is referenced on a test definition.
SetupId	Click to access the Routing Setup page, where you can modify an existing routing setup ID. For more information the Routing Setup page, see Section 8.4, "How to Modify Routing Setup IDs."

Routing Setup Actions

Available elements in the Routing Setup Actions area are discussed in Table 8–3.

Table 8–3 Routing Setup Actions Area Elements

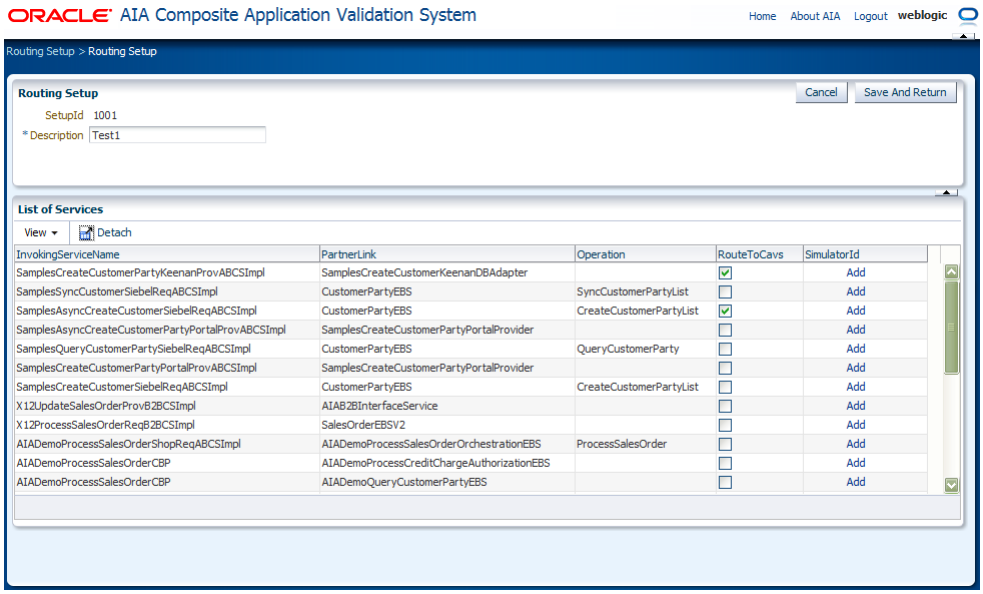
Element	Description
Reset Routing	Click to set all routing configurations to *FALSE*. This means that all routings to simulators (RoutetoCAVS property settings) in the AIAConfigurationProperties.xml file will be set to *FALSE*, whether you have defined them through the Routing Setup pages or directly in the file.
View Routing	Click to access the Configuration page, where you can access a read-only view of the last applied, or active, routing setup ID.

8.4 How to Modify Routing Setup IDs

To modify routing setup IDs:

1. Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Routing Setup** tab. Click a **SetupId** link. The Routing Setup page displays as shown in Figure 8–4.

Figure 8–4 Routing Setup Page



2. Use the page elements on the Routing Setup page to modify existing routing setups. Available elements are discussed in Table 8–4.

   Data saved on this page is stored in a CAVS table, rather than in the AIAConfigurationProperties.xml file.

   If you want to apply the data to the AIAConfigurationProperties.xml file, you must click **Apply Routing** for the routing setup ID on the Search Routing Setups page.

   For more information about the Apply Routings button, see Section 8.3, "How to Search for CAVS Routing Setup IDs."

Table 8–4 Routing Setup Page Elements

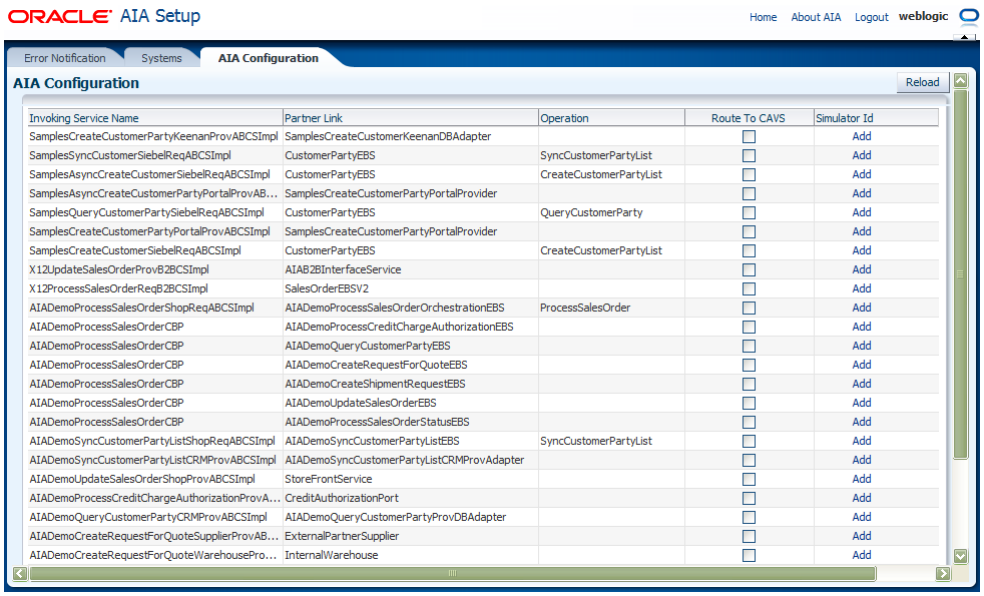
Element	Description
SetupId	Displays the ID you assigned to routing setup ID on the Create Routing Setup page.
Description	If applicable, edit the routing setup ID description.
Invoking Service Name	This is the service after which the service routing to CAVS should happen.
PartnerLink	The PartnerLink that is invoked by the service that you want to route to the CAVS simulator.
Operation	The operation of the PartnerLink that you want to route to the CAVS simulator. Displays a value only when multiple operations on the service are invoked using the same PartnerLink, typically when calling an enterprise business service.
RouteToCavs	Select to indicate that the invoking service should route to the selected CAVS simulator.
SimulatorId	Click the icon to access the Search Definitions page, where you can select the simulator definition that you want an invoking service to route to. If a simulator definition has already been selected, the simulator ID displays. Click **Modify** to select a different simulator ID. Click **Clear** to clear the selection.

8.5 How to Set Up CAVS Routing Configurations Without Creating Routing Setup IDs

To set up CAVS routing configurations without creating routing setup IDs:

1. Access the AIA Home Page. In the Setup area, click the **Go** button. Select the **Configuration** tab. The AIA Configuration page displays as shown in Figure 8–5.

Figure 8–5 AIA Configuration Page



2. Use this page to quickly set up a CAVS routing configuration without having to create routing setup IDs. This is particularly useful when you are only interested in using CAVS simulators without CAVS test definitions.

   For example, you may only need to use the CAVS simulator feature for your development purposes and you may not need to uptake the complexity involved in setting up routing setup IDs. In this case, you can use this page to directly modify service routing configurations in the AIAConfigurationProperties.xml file.

   Note: If you use this page to modify these service routing configurations, there is no need to manually reload the configurations.

   However, if you are using CAVS for extensive testing purposes, we recommend that you use the Routing Setup pages to create your routing setup.

   For more information about the Routing Setup page, see Section 8.2, "How to Create CAVS Routing Setup IDs."

9

Working with Test and Simulator Instances

This chapter describes how to work with test and simulator instances, how to view test instance details, and how to view simulator instance details.

A test instance captures the details of the execution of a test definition. A simulator instance captures the details of a simulator definition's behavior during the execution of a test definition with which it is associated.

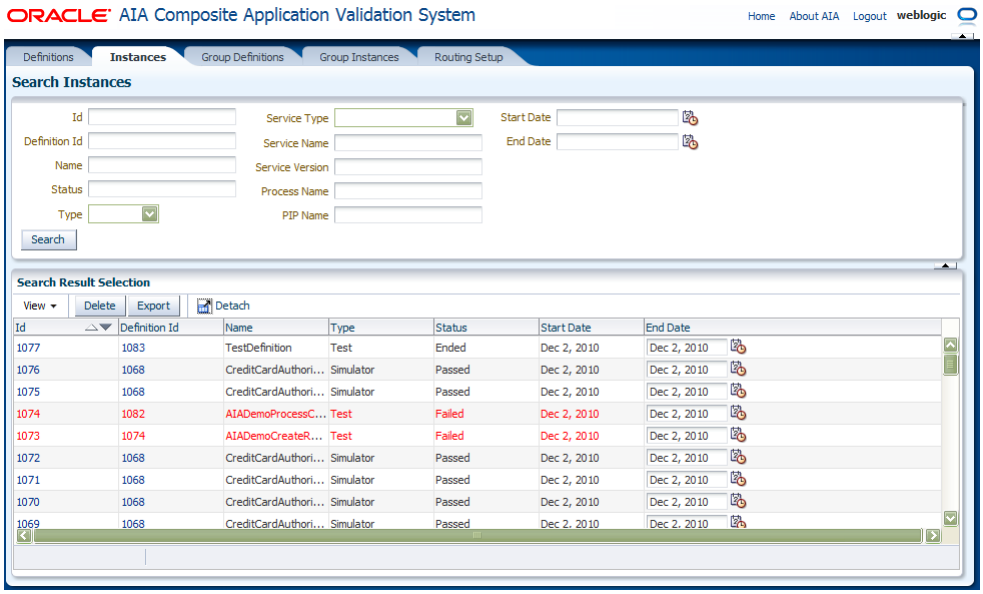
This chapter includes the following sections:

9.1 How to Work with Test and Simulator Instances

To work with test and simulator instances:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Instances** tab. The Instances page displays, as shown in Figure 9–1.

Figure 9–1 Instances Page



2. Use the page elements on the Instance page to search for test and simulator instances. You can also access pages you can use to view test and simulator instance details.

Search Instances

Use the Search Instances group box to enter search criteria to locate the instance you are searching for.

Available elements in the Search Instances group box are discussed in Table 9–1.

Table 9–1 Search Instances Group Box Elements

Element	Description
Id	Enter the unique key assigned to the instance.
Definition Id	Enter a definition ID associated with the definition that generated the instance.
Name	Enter the descriptive name given to the definition that generated the instance.
Status	Enter the status of the instance. ■ *Ended*: This status is only applicable to instances that do not involve validations. Indicates that the instance has ended. ■ *Faulted*: The instance could not execute properly due to exceptions or faults. ■ *Failed*: The instance did not pass validation. ■ *Passed*: The instance passed validation. ■ *Delayed*: For an asynchronous two-way test instance, indicates that the test instance is still active and waiting for an asynchronous reply.

Table 9–1 (Cont.) Search Instances Group Box Elements

Element	Description
Type	Select the type of instance for which you are searching. ■ **<Value Not Selected>** ■ **Test** ■ **Simulator**
Service Type	Select the business service pattern of the web service associated with the instance. For example, if you are searching for a test instance, this is the business service pattern of the web service tested by the test definition that generated the test instance. If you are searching for a simulator instance, this is the business service pattern of the web service simulated by the simulator definition that generated the simulator instance. ■ **<Value Not Selected>** ■ **Synchronous** ■ **Notify** ■ **Asynchronous two way**
Service Name	Enter the name of the web service associated with the definition that created the instance.
Service Version	Enter the version of the web service associated with the definition that created the instance.
Process Name	Enter the name of the process associated with the definition that created the instance.
PIP Name (Process Integration Pack name)	Enter the name of the Process Integration Pack (PIP) associated with the definition that created the instance.
Start Date	Enter a start date and time that you want to use as search criteria. The search will look for all instances that were created on and after the given date and time.
End Date	Enter an end date and time that you want to use as search criteria. The search will look for all instances that were created before and on the given date and time.
Search	Click to execute a search for instances using the search criteria entered in the Search Instances group box.

Search Result Selection

Use the Search Result Selection grid to work with instances returned in your search results. Upon accessing this page, the grid is populated by all instances.

Available elements in the Search Result Selection grid are discussed in Table 9–2.

Table 9–2 Search Result Selection Grid Elements

Element	Description
Delete	Select one or more instances that you want to delete and click the **Delete** button to execute the deletion.
Export	For more information about exporting instances, see Chapter 12, "Exporting and Importing CAVS Definitions and Instances."
Id	Click for a test instance to access the Test Instance Detail page.

Table 9–2 (Cont.) Search Result Selection Grid Elements

Element	Description
Definition Id	Click for a simulator instance to access the Simulator Instance Detail page. For a test instance, click to access details about the test definition that generated the test instance. An unlocked test definition displays on the Modify Test Definition page. A locked test definition displays on the View Test Definition page. For more information, see Section 4.2, "How to Modify a Test Definition." For a simulator instance, click to access details about the simulator definition that generated the simulator instance. An unlocked simulator definition displays on the Modify Simulator Definition page. A locked test definition displays on the View Simulator Definition page. For more information, see Section 5.2, "How to Modify a Simulator Definition."

9.2 How to View Test Instance Details

To view test instance details:

1. Access the Test Instances Detail page, as shown in Figure 9–2 and Figure 9–3.

   To access the page, use one of the following navigation paths:

   - Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. Click a test definition **Id** link. The Modify Test Definition page displays. Select **Execute** in the **Actions** drop-down list box.
   - Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. Click a test definition **Id** link. The Modify Test Definition page displays. Click an instance **Id** link in the **Test Instances** group box.
   - Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Instances** tab. Click an instance **Id** link.

Figure 9–2 Test Instances Detail Page (1 of 4)

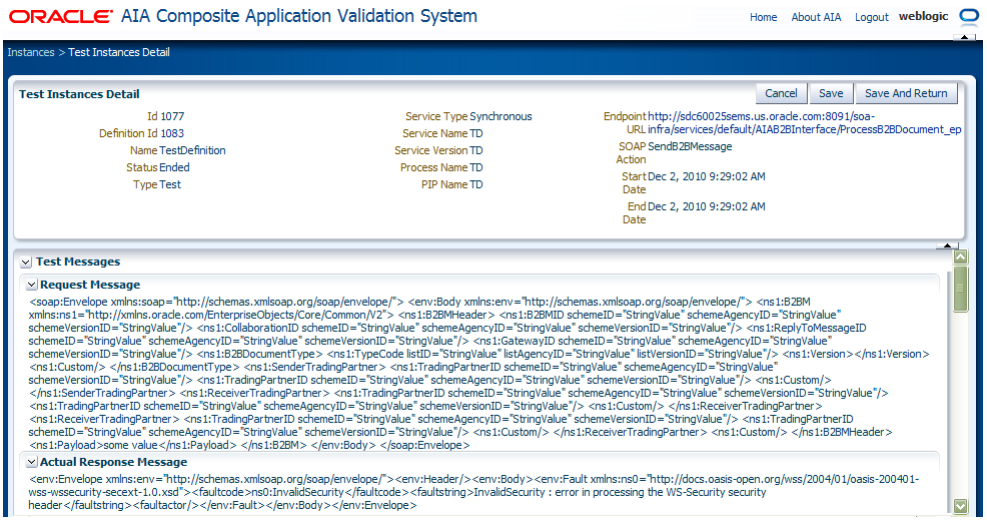


Figure 9–3 Test Instances Detail Page (2 of 4)

Figure 9–4 Test Instances Detail Page (3 of 4)

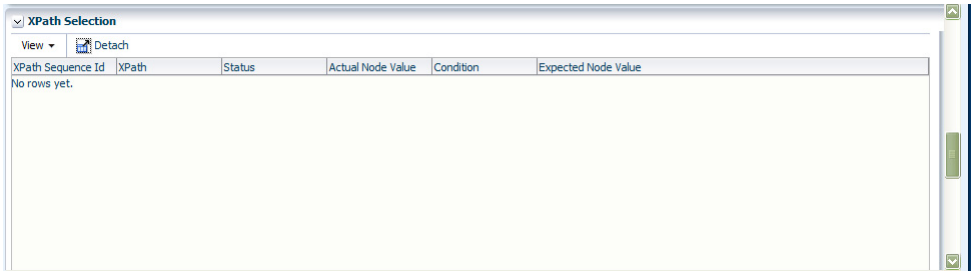
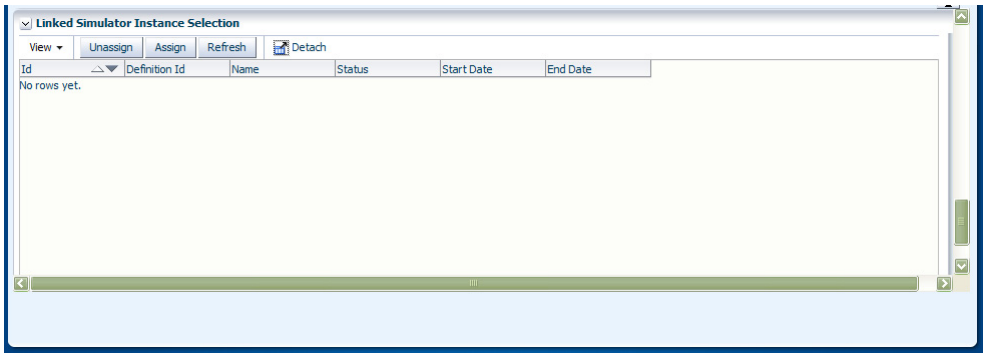


Figure 9–5 Test Instances Detail Page (4 of 4)



2. Use the page elements on the Test Instances Detail page, as discussed in Table 9–3, to view the details of a test instance.

Table 9–3 Test Instances Detail Page Elements

Element	Description
Cancel	Click to discard any updates to the page and return to the Instances page.
Apply	Click to apply and save any updates you have made to the page.
Save	Click to save any updates you have made to the page and go to the Instances page.
Id	Displays the unique ID assigned to the instance.
Definition Id	Displays the ID of the test definition that generated the test instance. Click for an unlocked test definition to access the Modify Test Definition page. Click for a locked test definition to access the View Test Definition page, where you can access a read-only view of the test definition. For more information, see Section 4.2, "How to Modify a Test Definition."
Name	Displays the descriptive name associated with the test definition that generated the instance.
Status	Displays the status of the test instance. ▪ **Ended**: This status is only applicable to test instances that do not involve validations. Indicates that the instance has ended. ▪ **Faulted**: The test instance could not execute properly due to exceptions or faults. ▪ **Failed**: The test instance did not pass validation. ▪ **Passed**: The instance passed validation. ▪ **Delayed**: For an asynchronous two-way test instance, indicates that the test instance is still active and waiting for an asynchronous reply.
Type	Displays the type of definition that generated the test instance. On the Test Instances Detail page, this value will always be **Test**.

Table 9–3 (Cont.) Test Instances Detail Page Elements

Element	Description
Service Type	Displays the business service pattern of the web service tested by the test definition that generated the test instance. ■ **Synchronous** ■ **Notify** ■ **Asynchronous two way**
Service Name	Displays the name of the web service tested by the test definition that created the instance.
Service Version	Displays the version of the web service tested by the test definition that created the instance.
Process Name	Displays the name of the process associated with the test definition that created the instance.
PIP Name (Process Integration Pack name)	Displays the name of the PIP associated with the test definition that created the instance.
Endpoint URL	Displays the URL of the web service tested by the test definition that created the instance.
SOAP Action	Displays the operation called by the web service tested by the test definition that created the instance.
Start Date	Displays the date and time at which the test instance was initiated.
End Date	Displays the date and time at which the test instance ended.

Test Messages

Use the elements in the Test Messages group box, as discussed in Table 9–4, to view the request and response XML messages associated with the test definition that generated the instance.

Table 9–4 Test Messages Group Box Elements

Element	Description
Request Message	Displays request message XML defined for the test definition that generated the test instance. For more information about the Request Message field, see Section 4.1, "How to Create a Test Definition."
Actual Response Message	Displays response message XML defined for the test definition that generated the test instance. For more information about the Response Message field, see Section 4.1, "How to Create a Test Definition."

Prefix and Namespace Selection

Displays namespace data created for the test definition that generated the test instance. This namespace data is used in the XPath values defined in the XPath Selection grid.

For more information about the Prefix and Namespace Selection grid, see Section 4.2, "How to Modify a Test Definition."

XPath Selection

Displays XPath data created for the test definition that generated the test instance. The values in this grid use the namespace values set in the Prefix and Namespace Selection grid.

For more information about the XPath Selection grid, see Section 4.2, "How to Modify a Test Definition."

Linked Simulator Instance Selection

Use the elements in Linked Simulator Instance Selection grid, as discussed in Table 9–5, to work with associations between test instances and simulator instances.

If no correlation logic has been defined between the test definition and the simulator definition, the test and simulator instances will not always be reconcilable, especially when the same web service is invoked multiple times during a very short time period, as in during performance testing.

However, if a simulator definition is associated with a test definition, any test instances generated by the test definition will automatically reflect associations to simulator instances generated by associated simulator definitions.

You can manually adjust these associations in this grid area.

Table 9–5 Linked Simulator Instance Selection Grid Elements

Element	Description
Unassign	Select one or more simulator instance rows that you want to disassociate with the test instance. Click the **Unassign** button to execute the disassociation.
Assign	Click to access the Search Instances - Simulator page, where you can search for a simulator instance that you want to manually associate with the test instance. After you have associated a simulator instance with the test instance using the Search Instances - Simulator page, the Test Instances Detail page displays the selected simulator instance.
Refresh	Click to refresh the Test Instances Detail page.
Id	Click to access the Simulator Instances Detail page.
Definition Id	Click to view details about the test definition that generated the test instance. An unlocked test definitions display on the Modify Test Definition page. A locked test definition displays on the View Test Definition page. For more information, see Section 4.2, "How to Modify a Test Definition."

9.3 How to View Simulator Instance Details

To view simulator instance details:

1. Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Instances** tab. Click the **Instance Id** link for a simulator instance. The Simulator Instances Detail page displays, as shown in Figure 9–6, Figure 9–7, Figure 9–8, and Figure 9–9.

Figure 9–6 Simulator Instances Detail Page (1 of 4)

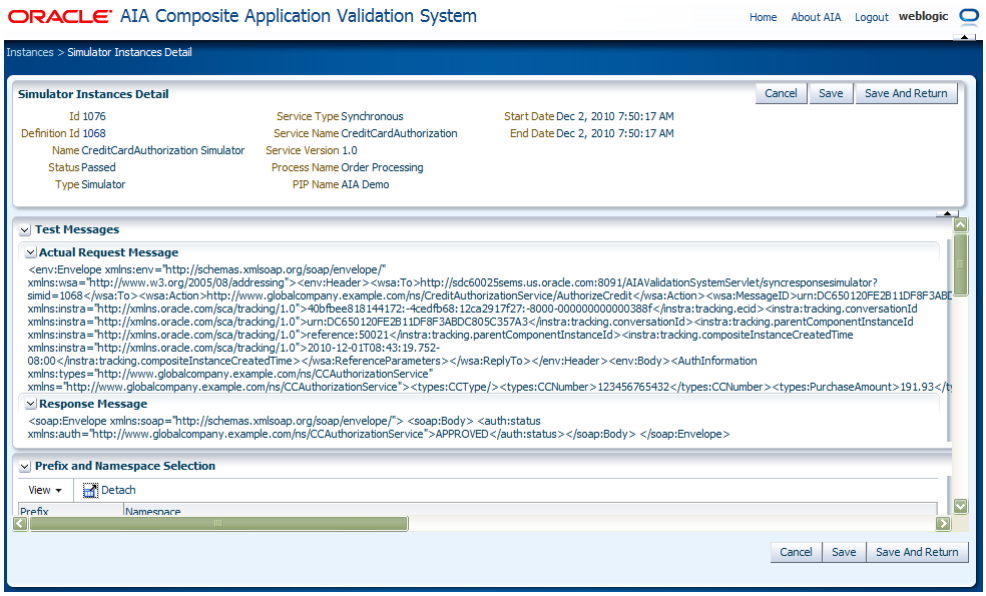


Figure 9–7 Simulator Instances Detail Page (2 of 4)

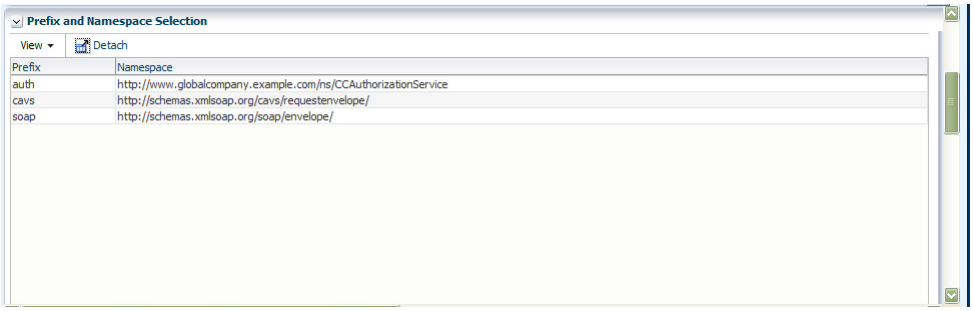


Figure 9–8 Simulator Instances Detail Page (3 of 4)

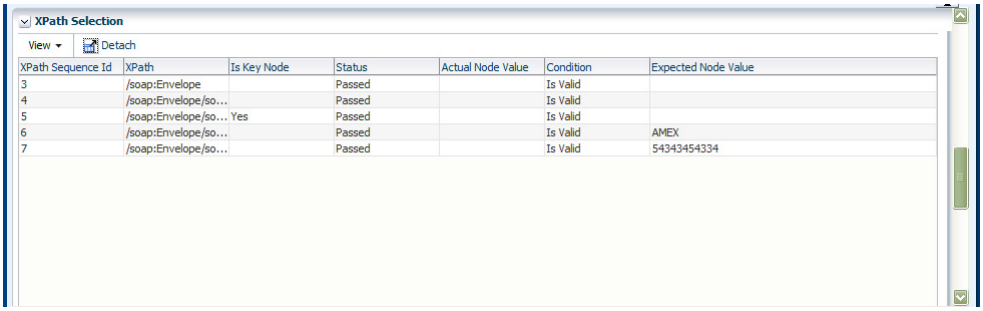
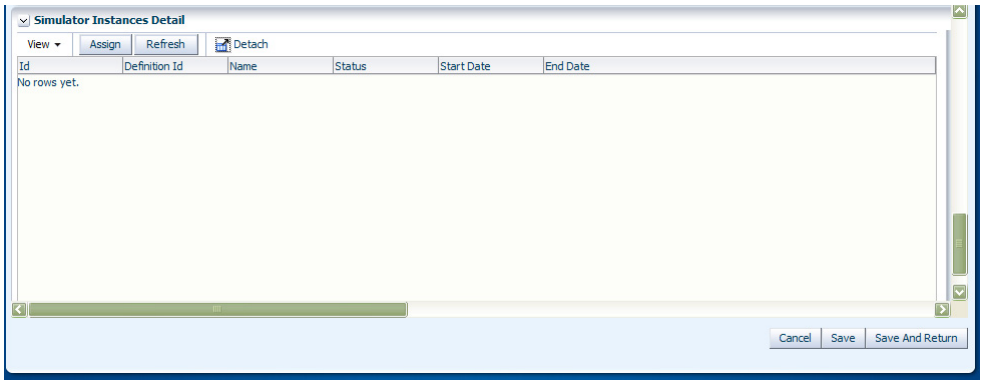


Figure 9–9 Simulator Instances Detail Page (4 of 4)



2. Use the page elements on the Simulator Instances Detail page, as discussed in Table 9–6, to view the details of a simulator instance.

Table 9–6 Simulator Instances Detail Page Elements

Element	Description
Cancel	Click to discard any updates to the page and return to the Instances page.
Apply	Click to apply and save any updates you have made to the page.
Save	Click to save any updates you have made to the page and go to the Instances page.
Id	Displays the unique ID assigned to the instance.
Definition Id	Click for an unlocked simulator definition to access the Modify Simulator Definition page. Click for a locked simulator definition to access the View Simulator Definition page, where you can access a read-only view of the simulator definition.
Name	Displays the descriptive name associated with the simulator definition that generated the instance.
Status	Displays the status of the simulator instance. ▪ **Initiated**: The simulator instance has been initiated. ▪ **Ended**: This status is only applicable to simulator instances that do not involve validations. Indicates that the instance has ended. ▪ **Faulted**: The simulator instance could not execute properly due to exceptions or faults. ▪ **Failed**: The simulator instance did not pass validation. ▪ **Passed**: The simulator instance passed validation.
Type	Displays the type of definition that generated the simulator instance. On the Simulator Instances Detail page, this value will always be **Simulator**.
Service Type	Displays the business service pattern of the web service simulated by the simulator definition that generated the instance. ▪ **Synchronous** ▪ **Notify** ▪ **Asynchronous two way**

Table 9–6 (Cont.) Simulator Instances Detail Page Elements

Element	Description
Service Name	Displays the name of the web service simulated by the simulator definition that created the instance.
Service Version	Displays the version of the web service simulated by the simulator definition that created the instance.
Process Name	Displays the name of the process associated with the simulator definition that created the instance.
PIP Name (Process Integration Pack name)	Displays the name of the PIP associated with the simulator definition that created the instance.
Start Date	Displays the date and time at which the simulator instance was initiated.
End Date	Displays the date and time at which the simulator instance ended.

Test Messages

Use the elements in the Test Messages group box, as discussed in Table 9–7, to view the request and response XML messages associated with the simulator definition that generated the instance.

Table 9–7 Test Messages Group Box Elements

Element	Description
Actual Request Message	Displays request message XML defined for the simulator definition that generated the instance. For more information about the Request Message field, see Section 5.1, "How to Create a Simulator Definition."
Response Message	Displays response message XML defined for the simulator definition that generated the instance. For more information about the Response Message field, see Section 5.1, "How to Create a Simulator Definition."

Prefix and Namespace Selection

Displays namespace data created for the simulator definition that generated the simulator instance. This namespace data is used in the XPath values defined in the XPath Selection grid.

For more information about the Prefix and Namespace Selection grid, see Section 5.2, "How to Modify a Simulator Definition."

XPath Selection

Displays XPath data created for the simulator definition that generated the instance. The values in this grid use the namespace values set in the Prefix and Namespace Selection grid.

For more information about the XPath Selection grid, see Section 5.2, "How to Modify a Simulator Definition."

Linked Test Instance Selection

Displays the test instance with which the simulator instance is associated. This is a one-to-one association.

If no correlation logic has been defined between the test definition and the simulator definition, the test and simulator instances will not always be reconcilable, especially when the same web service is invoked multiple times during a very short time period, as in during performance testing.

However, if a simulator definition is associated with a test definition, any test instances generated by the test definition will automatically reflect associations to simulator instances generated by associated simulator definitions.

You can adjust the association between the simulator instance and a test instance using the elements on the page, which are discussed in Table 9–8.

Table 9–8 Linked Test Instance Selection Page Elements

Element	Description
Unassign	Select the test instance ID that you want to disassociate from the simulator instance and click the **Unassign** button to execute the disassociation.
Assign	Click to access the Search Instances - Test page, where you can search for a test instance that you want to manually associate with the simulator instance. After you have associated a test instance with the simulator instance using the Search Instances - Test page, the Simulator Instances Detail page displays the selected test instance.
Refresh	Click to refresh the Simulator Instances Detail page.
Id	Click to display the selected test instance on the Test Instances Detail page
Definition Id	Displays the ID of the test definition that generated the test instance. Click for an unlocked test definition to access the Modify Test Definition page. Click for a locked test definition to access the View Test Definition page, where you can access a read-only view of the test definition. For more information, see Section 4.2, "How to Modify a Test Definition."

10

Working with Group Instances

This chapter describes how to view group instances and how to view group instance details.

A group instance captures the details of the execution of a group definition.

This chapter includes the following sections:

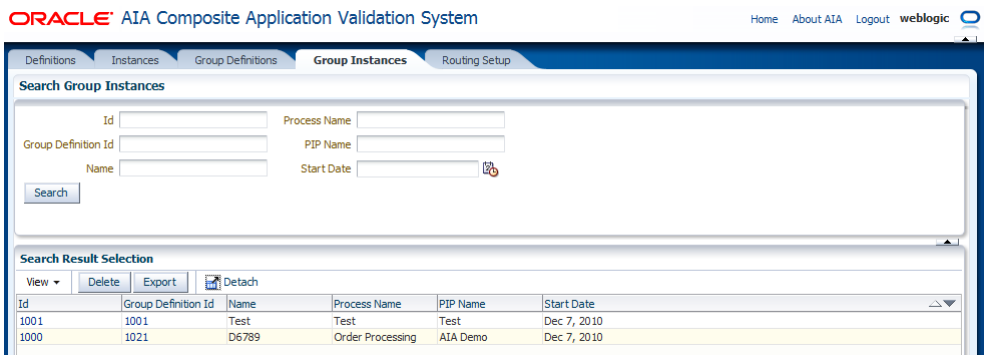
- Section 10.1, "How to View Group Instances"
- Section 10.2, "How to View Group Instance Details"

10.1 How to View Group Instances

To view group instances:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Group Instances** tab. The Group Instances page displays, as shown in Figure 10–1.

Figure 10–1 Group Instances Page



2. Use the page elements on the Group Instances page to search for group instances and access a page you can use to view group instance details.

Search Group Instances

Use the elements in the Search Group Instances group box, as discussed in Table 10–1, to enter search criteria to find the group instance you are searching for.

Table 10–1 Search Group Instances Group Box Elements

Element	Description
Id	Enter the unique key identifier assigned to the group instance.
Group Definition Id	Enter the unique key ID assigned to the group definition that generated the instance.
Name	Enter a descriptive name assigned to the group definition.
Process Name	Enter the name of the process associated with the group definition that generated the instance.
PIP Name (process integration pack)	Enter the name of the Process Integration Pack (PIP) associated with the group definition that generated the instance.
Start Date	Enter a start date and time that you want to use as search criteria. The search will look for all group instances that were created on and after the given date and time.
Search	Click to execute a search for group instances using the search criteria entered in the Search Group Instances group box.

Search Result Selection

Use the elements in the Search Result Selection grid, as discussed in Table 10–2, to work with group instances returned in your search results. Upon accessing this page, the grid is populated by all group instances.

Table 10–2 Search Result Selection Grid Elements

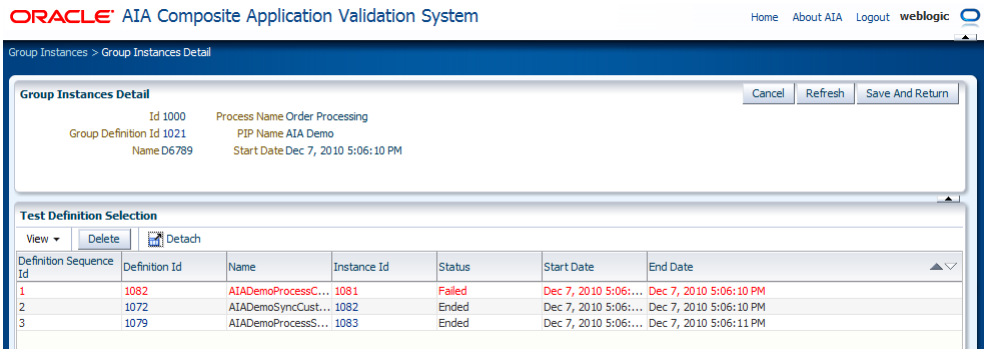
Element	Description
Delete	Select one or more group instances that you want to delete and click the **Delete** button to execute the deletion.
Export	For more information exporting group instances, see Chapter 12, "Exporting and Importing CAVS Definitions and Instances."
Id	Click to access the Group Instances Detail page.
Group Definition Id	Click to access the Group Definition Detail page. For more information about the Group Definition Detail page, see Chapter 7, "Working with Group Definitions."

10.2 How to View Group Instance Details

To view group instance details:

1. Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Group Instances** tab. Click a group instance **Id** link on the Group Instances page. The Group Instances Detail page displays as shown in Figure 10–2.

Figure 10–2 Group Instances Detail Page



2. Use the page elements on the Group Instances Detail page, as discussed in Table 10–3, to view the details of a group instance.

Table 10–3 Group Instances Detail Page Elements

Element	Description
Id	Displays the unique key identifier assigned to the group instance.
Group Definition Id	Click to access the Group Definition Detail page.
Name	Displays the descriptive name assigned to the group definition.
Process Name	Displays the name of the process associated with the group definition that generated the instance.
PIP Name	Enter the name of the PIP associated with the group definition that generated the instance.
Start Date	Displays the date and time at which the group instance was initiated.
Delete	Select one or more test instance rows that you want to delete and click the **Delete** button to execute the deletion.
Definition Sequence Id	Indicates the sequence in which the test definitions were initiated by the group definition that generated the group instance.
Definition Id	Click to access the Modify Test Definition page. For more information about the Modify Test Definition page, see Section 4.2, "How to Modify a Test Definition."
Instance Id	Click to access the Test Instances Detail page. For more information about the Test Instances Detail page, see Section 9.2, "How to View Test Instance Details."
Status	Displays the status of the test instance in the group instance. ■ **Initiated:** The test instance has been initiated. ■ **Ended**: This status is only applicable to test instances that do not involve validations. Indicates that the instance has ended. ■ **Faulted**: The test instance could not execute properly due to exceptions or faults. ■ **Failed**: The test instance did not pass validation. ■ **Passed**: The instance passed validation.
Start Date	Displays the date and time at which the test instance was initiated.
End Time	Displays the date and time at which the test instance ended.

11

Purging CAVS-Related Cross-Reference Entries to Enable Rerunning of Test Scenarios

This chapter describes how to purge Composite Application Validation System (CAVS)-related cross-reference entries to allow test scenarios to be rerun.

This chapter includes the following sections:

11.1 Introduction to Purging CAVS-Related Cross-Reference Entries

When a participating application is involved in a CAVS testing flow, execution of tests can potentially modify data in a participating application. Therefore, consecutive running of the same test may not generate the same results. The CAVS is not designed to prevent this kind of data tampering because it supports the user s intention to include a real participating application in the flow. The CAVS has no control over modifications that are performed in participating applications.

However, this issue does not apply if your CAVS test scenario uses test definitions and simulator definitions to replace all participating applications and other dependencies. In this case, all cross-reference data is purged after the test scenario has been executed. This enables rerunning of the test scenario.

11.2 How to Purge CAVS-Related Cross-Reference Entries to Enable Rerunning of Test Scenarios

To purge CAVS-related cross-reference entries to enable rerunning of test scenarios:

1. Process integration packs (PIPs) that are delivered to work with Oracle Application Integration Architecture (AIA) Foundation Packs are delivered with cross-reference systems in place. They are named CAVS_<XYZ>, where <XYZ> is the participating application system.

   For example, for systems EBIZ and SEBL, the PIP is delivered with cross-reference systems CAVS_EBIZ and CAVS_SEBL.

2. For every system type defined on the Systems page for which you want to make test scenarios rerunnable (<XYZ>), create a related CAVS system (CAVS_<XYZ>).

The System Type field value for the CAVS-related entry should match the name of the system for which it is created.

For more information about the Systems page, see "Building AIA Integration Flows" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

3. When testing a provider Application Business Connector Service (ABCS) in isolation, the Enterprise Business Message (EBM) will be passed from the CAVS to the provider ABCS with the NamespacePrefixedEBMName/EBMHeader/Target/ID element set as CAVS_<XYZ>.
4. When testing a requester ABCS in isolation, the element in the Application Business Message (ABM) that normally contains the Internal ID value will now contain the CAVS-specific Internal ID value set for the system on the Systems page.
5. When testing an entire flow (requester ABCS-to-Enterprise Business Service [EBS] -to-provider ABCS), you must set the Default.SystemID property of the provider ABCS to *CAVS*_<*XYZ*>, where <*XYZ*> is the system.
   a. To do this, edit the `Default.SystemID` property value in the AIAConfigurationProperties.xml file in the <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config directory.
   b. Reload updates to the AIAConfigurationProperties.xml file.

      For more information about reloading updates to AIAConfigurationProperties.xml, see "Building AIA Integration Flows" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*
   c. You can now commence testing the entire flow.

Note: If the test scenario is an entire flow that includes multiple instances of the same system, this approach will not work. In this case, data created in the cross-reference will remain making the same test case non-rerunnable.

12

Exporting and Importing CAVS Definitions and Instances

This chapter describes how to export and import Composite Application Validation System (CAVS) definitions and instances.

This chapter includes the following sections:

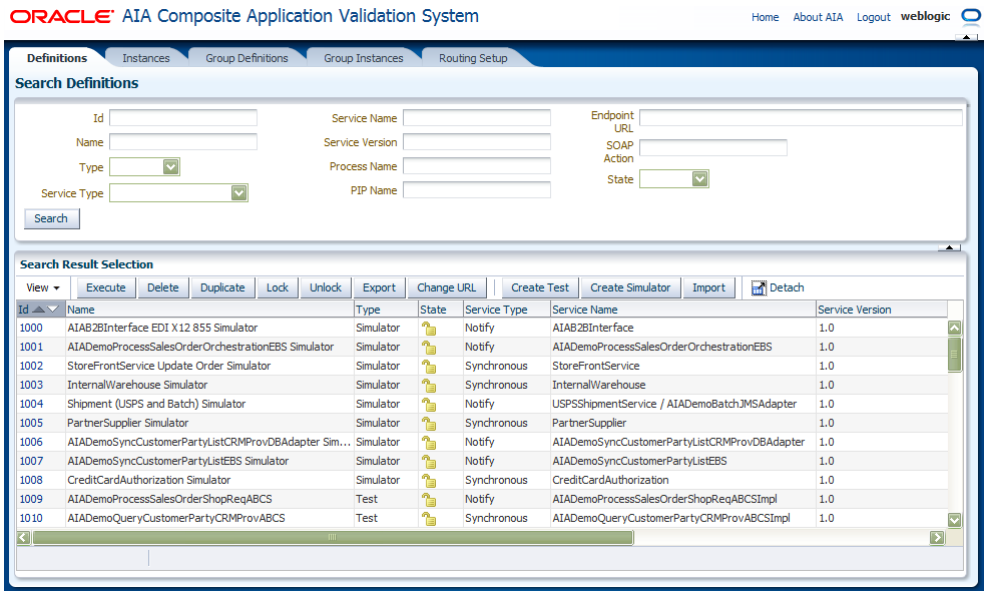
- Section 12.1, "How to Export and Import Definitions"
- Section 12.2, "How to Export Test and Simulator Instances"
- Section 12.3, "How to Export Group Instances"

12.1 How to Export and Import Definitions

To export and import CAVS definitions:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Definitions** tab. The Definitions page displays, as shown in Figure 12–1.

Figure 12–1 Definitions Page



Use the **Export** and **Import** buttons on this page to migrate test definitions, simulator definitions, and any associated group definitions in XML flat-file format between instances running on the same version of Foundation Pack.

Examples of uses for this export and import functionality include:

- QA may want to certify a set of definitions that have been run in one build in other builds.
- Support analysts and customers may want to exchange definition files.
- You may want to verify validity of new environments.

2. Select one or more definitions and click the **Export** button to initiate the export. The following options display:
   - Export selected Definition(s) only
   - Export selected Definition(s) and associated Group Definition(s)
   - Export selected Definition(s), associated GroupDefinition(s) and Test Definition(s) that belong to the associated GroupDefinition(s) but are not selected

   Select an option and click the **Proceed** button to create and save the definitions to a location on your local system. The default file name for the exported definition(s) is Definitions.xml.

   If a test definition that you are exporting is associated with a routing setup ID, the routing setup information will also be exported.

   If that routing setup is associated with one or more simulator definitions, which were provided when the Route To CAVS option was set to TRUE, then these simulator definitions will also be exported.

   For more information about the structure of the Definitions.xml file created by the CAVS export definition feature, see Appendix A, "XML Structures of Exportable CAVS Definitions and Instances."

3. Use the **Import** button to upload a test, simulator, or group definition in the XML flat-file format generated by CAVS export functionality.

   You can generate these files by clicking the **Export** button on this page. The definition file to be uploaded must be accessible by the local system being used to perform the upload.

   Click the **Import** button and browse for the file you want to upload. The CAVS validates the structure of the file being uploaded. If the structure is invalid, an error will be raised.

   If a test definition that you are importing is associated with a routing setup ID, the routing setup information will also be imported.

   If that routing setup is associated with one or more simulator definitions, which were provided when the Route To CAVS option was set to TRUE, then these simulator definitions will also be imported.

   For more information about the valid structure of the Definitions.xml file created by the CAVS export definition feature, see Appendix A, "XML Structures of Exportable CAVS Definitions and Instances."

4. Imported definitions will still reference endpoint URLs pointing to tested web services in the source system. You must update imported definition endpoint URL values to point to tested web services in the target system. The CAVS enables you to update these URLs directly on the following pages:

Click the **Change URL** button on this page to access the Modify Test Definitions page, where you can update the Endpoint URL field value.

For more information about the Endpoint URL field, see Section 4.1, "How to Create a Test Definition."

Because the sequential definition IDs assigned in the source system may not be valid in the target system, new sequential definition IDs will be assigned by the target system. As a result, associations between definitions will be severed in the target system and will need to be reestablished.

Because test, simulator, and group instance details that may be associated with definitions in the source system are not valid in the target system, they will not be imported.

If the same definition is uploaded multiple times, multiple duplicate definitions will be created in the target system.

For more information about the Definitions page, see Chapter 6, "Searching for Test and Simulator Definitions."

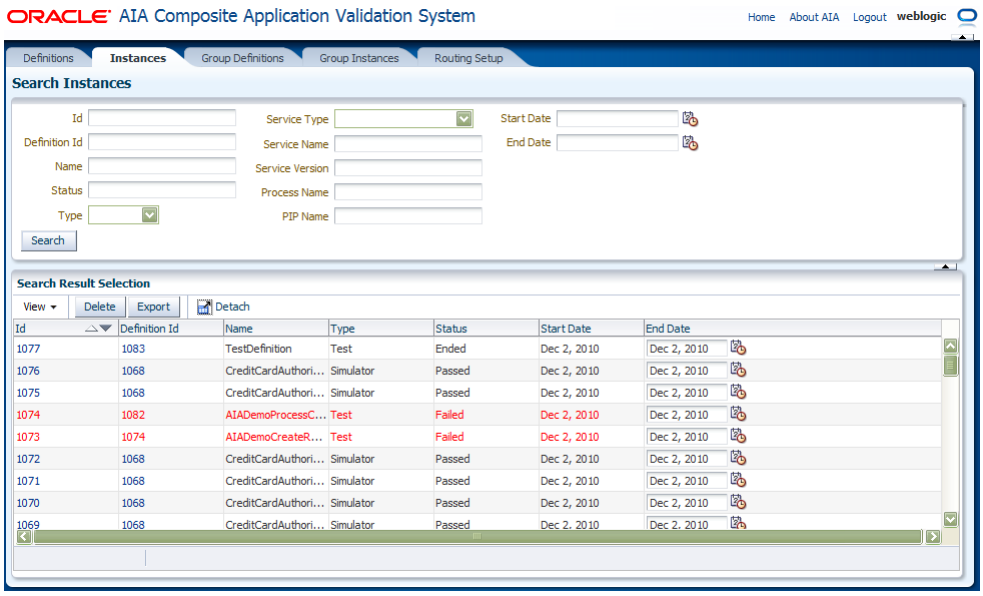
12.2 How to Export Test and Simulator Instances

To export test and simulator instances:

1. Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Instances** tab. The Instances page displays, as shown in Figure 12–2.

   For more information about the Instances page, see Chapter 9, "Working with Test and Simulator Instances."

Figure 12–2 Instances Page



2. Use the Export feature to export instances in XML format. You can use XML-based reporting tools to generate reports of test and simulator executions using these XML files.

Select one or more instances and click the **Export** button to initiate the export.

For more information about the structure of the Definitions.xml file created by the CAVS export instance feature, see Appendix A, "XML Structures of Exportable CAVS Definitions and Instances."

3. Click **Save** to create and save the definitions to a location on your local system. The default file name for the exported definition(s) is Instances.xml.

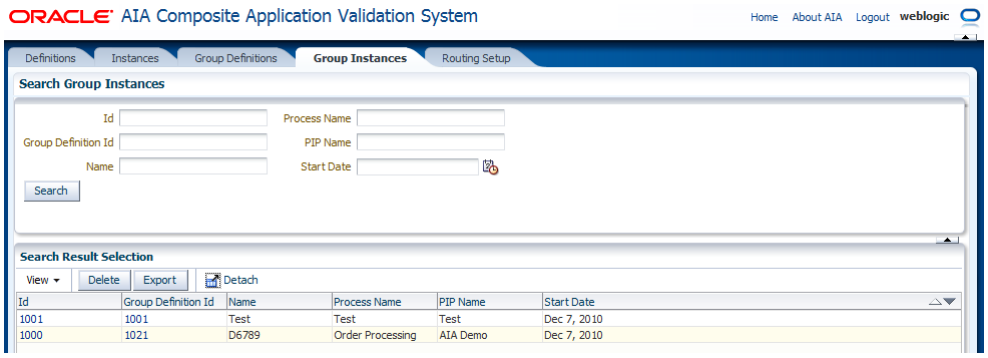
12.3 How to Export Group Instances

To export group instances:

1. Access the AIA Home Page. In the Composite Application Validation System area, click the **Go** button. Select the **Group Instances** tab. The Group Instances page displays, as shown in Figure 12–3.

   For more information about the Group Instances page, see Section 10.1, "How to View Group Instances."

Figure 12–3 Group Instances Page



2. Select one or more group instances that you want to export and click the **Export** button to execute the download.

   For more information about the structure of the Definitions.xml file created by the CAVS export definition feature, see Appendix A, "XML Structures of Exportable CAVS Definitions and Instances."

13

Introduction to Oracle AIA Error Handling

This chapter provides an introduction to the Oracle AIA Error Handling Framework. It also provides overviews about business faults, BPEL and Mediator system faults, and Oracle B2B errors.

The Oracle AIA Error Handling Framework provides error handling and logging components to support the needs of integration services operating in an Oracle Application Integration Architecture (AIA) ecosystem.

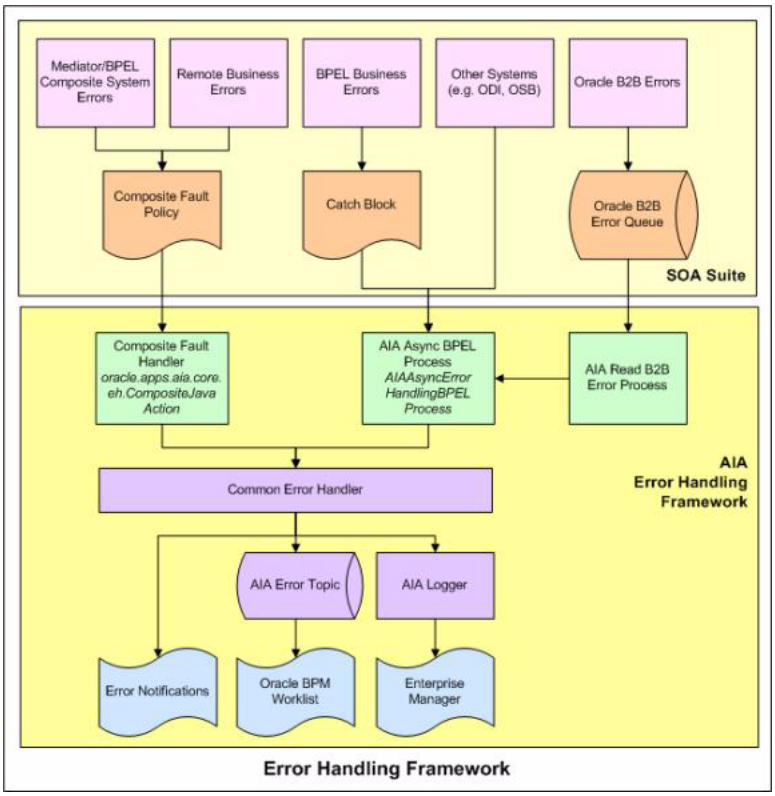
This chapter includes the following sections:

13.1 Introduction to the Error Handling Framework

This section includes the following topic: Section 13.1.1, "Fault Categories."

Figure 13–1 provides a high-level overview of the Error Handling Framework.

Figure 13–1 Key Features of Error Handling Framework Components



The AIA Error Handling Framework provides the following key features for integration services operating in an AIA ecosystem.

Unified Error Handling Approach

- Works across technologies, including BPEL and Mediator components, business-to-business (B2B), and ODI.
- Works across categories of faults, including business and system, runtime, and technical faults.
- Works across integration patterns.
- Adopts the Oracle SOA Suite 11g tech stack.

Error Notifications

- Error notifications are emailed to suitable actor roles, such as integration administrators, and FYI roles, such as customer service representatives.
- Provides visibility into error context.
  - Drill-down to the Oracle Enterprise Manager Console Flow Trace page from the error notification email.
  - View errors in the context of an AIA flow trace.
- Enables customization of error notification content.
  - Add key fields to the error notification body.
  - Add or remove fields from error notification content.
  - Issue error notifications to suitable Actor and FYI roles.

    - Provide a link to Oracle BPM Worklist for error details, if desired.
- Enables error notification throttling.
    - Control the number of error notifications issued for a specific error.
    - Regulate the issuance of error notifications by time interval and number of errors.

Oracle BPM Worklist Integration

- Centralized user interface to access error details that are assigned for resolution or for informational purposes.
- Accessible to administrators and end-users.
- Decoupled from the Error Notification Framework.
    - Oracle BPM Worklist is not tied to error notifications.
    - Oracle BPM Worklist can be used as an optional component.

Error Logging

- Logs messages non-intrusively in a consistent schema.
- Logs can be searched, sorted, and filtered using Oracle Enterprise Manager.

B2B Error Handling

- Errors in the Oracle B2B component of Oracle Fusion Middleware are routed to the AIA Error Handling Framework.
- The AIA fault definition captures B2B-specific details of a failed AIA flow.

Extensible Framework

- Ability to extend error handling capabilities.
- Automated error actions.
- Ability to automatically acts upon the errored object to provide automated retry actions, error notifications, and logging.
- Error actions, including retry, rethrow, replay, abort, Java action, and human-intervention.

13.1.1 Fault Categories

There are two categories of faults:

- **Business faults**

  Business faults are generated when there is a problem with the information being processed. For example, a credit card number is invalid.

  Error actions for business faults that are internal to BPEL are configured in catch blocks. These are business faults that are thrown by a throw activity. Error notifications and logging for these business faults are handled by AIAAsyncErrorHandlingBPELProcess.

  Error actions for business faults from external applications and services are configured using the Composite Fault Policy Framework. These are business errors that are returned by an invoked service or application when using a BPEL invoke activity. Error notifications and logging for these business faults are handled by oracle.apps.aia.core.eh.CompositeJavaAction.

- **System faults**

  System faults occur as a result of problems within the running of the BPEL process or Mediator service component. For example, data cannot be copied properly because the variable name is incorrect or because of transformation errors.

  Error actions for system faults are configured using the Composite Fault Policy Framework. Error notifications and logging for system faults are handled by oracle.apps.aia.core.eh.CompositeJavaAction.

13.2 Introduction to Error Handling for Business Faults

This section discusses error handling for two types of business faults:

- Local business faults
- Remote business faults

Local Business Faults

If a BPEL process or Mediator component needs to issue a business error, such as a validation error, the process must be developed to issue the error explicitly, catch it in a catch block, and invoke the AIAAsyncErrorHandlingBPELProcess. The input to the process is a fault message in the AIA fault message schema. This is also true for business errors for Oracle Data Integrator, Oracle Service Bus, third-party B2B, and other external systems that want to leverage the AIA Error Handling and Logging framework.

Remote Business Faults

If an invoked service or application responds to a request with a business fault, the Oracle SOA Suite captures these types of errors using the Composite Fault Policy Framework. The AIA Error Handling framework provides a custom Java action, oracle.apps.aia.core.eh.CompositeJavaAction, which can be configured as the Java action for all policies.

By configuring fault policies to include this Java action, the AIA Error Handling framework can perform all necessary error logging and notifications.

For more information, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

13.3 Introduction to Error Handling for BPEL and Mediator System Faults

These types of errors are captured using the Composite Fault Policy Framework. The AIA Error Handling framework provides a custom Java action, oracle.apps.aia.core.eh.CompositeJavaAction, which can be configured as the Java action for all policies.

By configuring fault policies to include this Java action, the AIA Error Handling framework can perform all necessary error logging and notifications.

For more information, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

For more information about the Composite Fault Policy framework, see "Using Fault Handling in a BPEL Process" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

13.4 Introduction to Error Handling for Oracle B2B Errors

The Oracle AIA Error Handling Framework is automatically triggered when there is an error in Oracle B2B.

Oracle B2B can encounter errors while exchanging B2B documents with trading partners. Some common reasons for errors in the Oracle B2B layer include the following scenarios:

- Failure of document schema validation in the B2B layer.
- Incorrect or missing trading partner agreements in Oracle B2B.
- Incorrect or missing document-type definitions in Oracle B2B.
- Network errors or unavailability of a trading partner system.
- Authentication failures, for example invalid digital certificates, and so forth.

> **Note:** Business process failures, such as an order being rejected by the trading partner if the ordered item is not in stock, are not considered to be Oracle B2B errors. Response or acknowledgment messages from trading partner applications containing these failures are treated as independent flows.

When Oracle B2B encounters these system errors, its default behavior is to publish the error to the Oracle Advanced Queuing (AQ) queue defined in the Oracle B2B infrastructure schema.

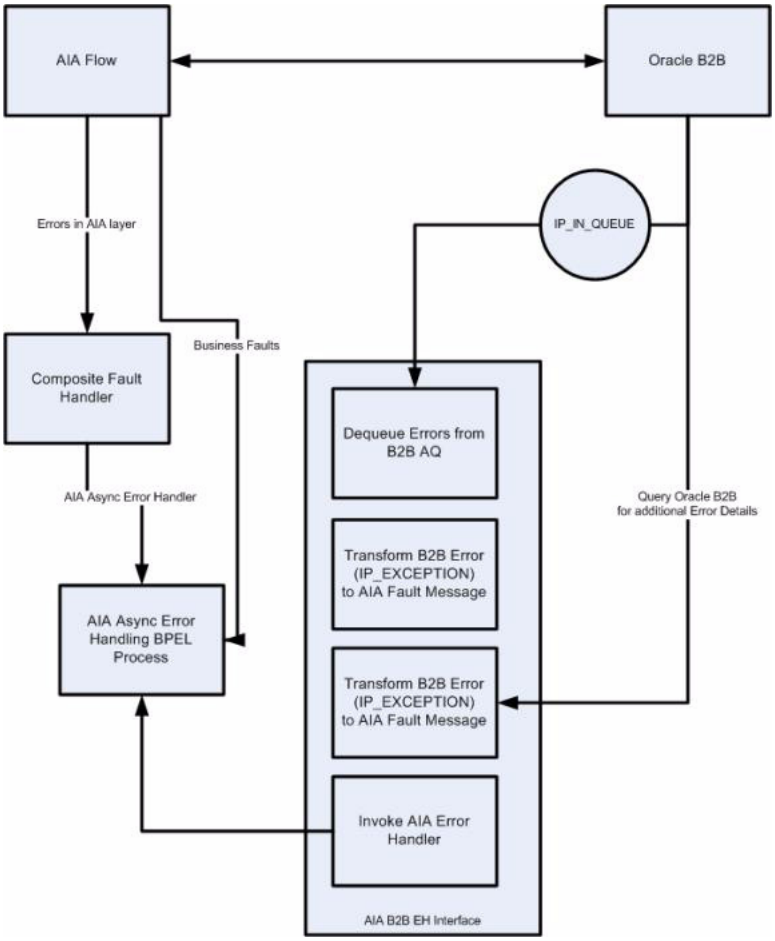
The details of the AQ to which Oracle B2B posts errors are covered in Table 13–1.

Table 13–1 Oracle B2B Error AQ Details

Queue Name	IP_IN_QUEUE
Database Schema	SH_SOAINFRA
Queue Consumer	b2berroruser
Data Source	jdbc/SOADatasource

Figure 13–2 illustrates the way in which the AIA error handling framework captures B2B errors:

Figure 13–2 Error Handling Framework Support for Capturing B2B Errors



These errors can be viewed in error reports available in the Oracle B2B console.

For more information about viewing Oracle B2B error reports, see Chapter 19, "Accessing Oracle B2B Errors."

For Oracle B2B inbound and outbound flows, when an error occurs within the Oracle B2B server and not in the AIA layer, the AIA fault has the capacity to capture only the B2B-specific details.

Enterprise Business Message (EBM) details will not be available. However, in the case of an error in an outbound flow, AIA is able to track the EBMID and include that information in the fault.

14

Setting Up Error Handling

This chapter provides an overview and discusses how to create user roles, associate email addresses to user roles, configure notification details, and how to set up error handling configuration details.

This chapter includes the following sections:

14.1 Introduction to Setting Up Error Handling

Setting up error handling involves configuring the following items:

- Error notification enablement

  Error notification functionality is enabled by default.

  For more information about disabling error notification functionality, see Section 15.4, "Disabling Error Notifications."

- Oracle BPM Worklist enablement

  Oracle BPM Worklist functionality is disabled by default.

  For more information about enabling Oracle BPM Worklist functionality, see Section 16.2, "How to Enable the Oracle BPM Worklist."

- Error handling user roles

  Create user roles in WebLogic Server Administration Console to receive error notifications and Oracle BPM Worklist task assignments.

  For more information, see Section 14.2, "How to Create Error Handling User Roles."

- Error handling user role email addresses

  Use Oracle User Messaging Service to associate email addresses with error handling user roles. Error notifications will be sent to the email addresses specified.

  For more information, see Section 14.3, "How to Associate Email Addresses with Error Handling User Roles."

- Notification configuration details

  Configure details that enable error notification emails to be sent.

  For more information, see Section 14.4, "How to Configure Notification Details."

- Error handling configuration details

  Define and modify error handling configuration details, including Error Notification and Oracle Worklist roles and responsibilities for processes operating in an Oracle Application Integration Architecture (AIA) ecosystem.

  For more information, see Section 14.5, "How to Set Up AIA Error Handling Configuration Details."

- Error handling responsibilities

  If you do not want to assign Actor and FYI user roles for specific error scenarios, you can assign default Actor and FYI user roles in AIAConfigurationProperties.xml.

  For more information, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

Figure 14–1 illustrates the way in which error handling setup elements enable Error Notification functionality.

Figure 14–1 Error Handling Setup Elements That Enable Error Notification Functionality

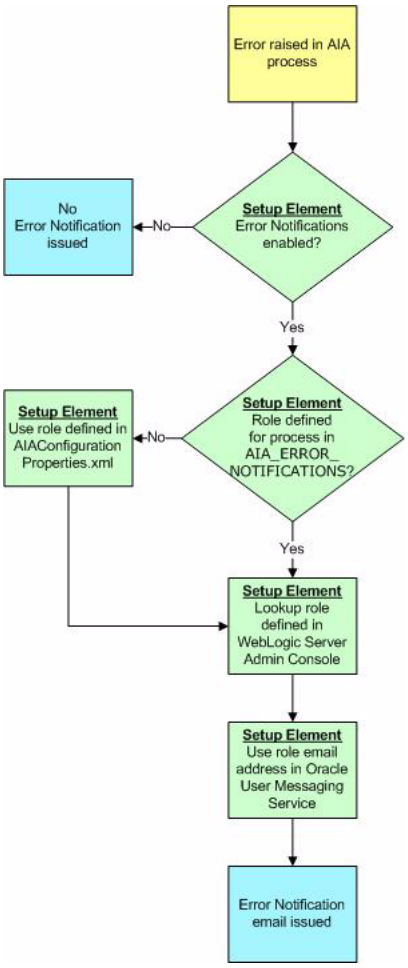
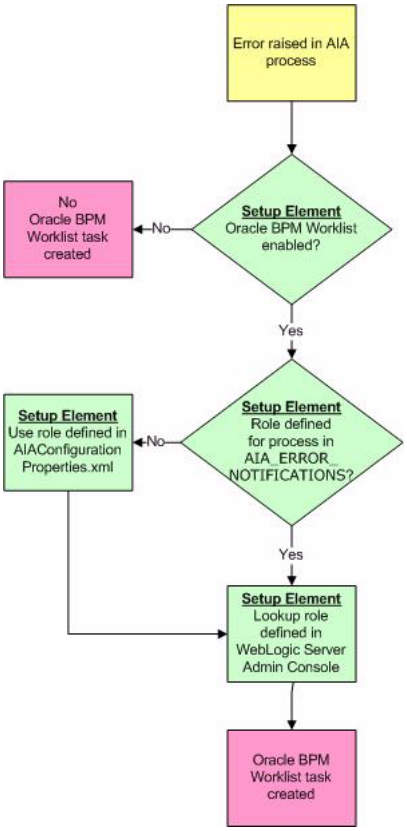


Figure 14–2 illustrates the way in which error handling setup elements enable Oracle Worklist functionality.

Figure 14–2 Error Handling Setup Elements That Enable Oracle Worklist Functionality



14.2 How to Create Error Handling User Roles

To create error handling user roles:

1. Access the Oracle WebLogic Server Administration Console: http://<host>:<port>/console.
2. In the Domain Structure menu, click **Security Realms**.
3. On the Summary of Security Realms page, select **myrealm**.
4. On the Settings for myrealm page, select the **Users and Groups** tab.
5. Select the **Users** tab.
6. Create and modify user roles for use with the Error Handling Framework. For error handling notification and worklist functionality to work as designed, ensure that you are using user roles and not groups.

For more information about setting up user roles, see "Using the Administration Console to Manage Users, Groups, and Roles" in *Oracle Fusion Middleware Securing Resources Using Roles and Policies for Oracle WebLogic Server.*

Note: Any user roles you create in the WebLogic Server Administration Console are stored in the Oracle WebLogic Server's embedded LDAP server. You may integrate a third-party LDAP solution to the embedded LDAP server.

For more information about Oracle WebLogic Server s embedded LDAP server, see "Security Provider Databases" in *Oracle Fusion Middleware Understanding Security for Oracle WebLogic Server.*

14.3 How to Associate Email Addresses with Error Handling User Roles

To associate email address with error handling user roles:

1. Access the My Messaging Channels page in the Oracle User Messaging Service standalone user interface:
   http://<soa-host>:<soa-port>/sdpmessaging/userprefs-ui.

   For more information about creating, updating, and deleting a message channel, see "How to Manage Messaging Channels" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

2. Associate an email address with an error handling user role.

   For more information about creating user roles, see Section 14.2, "How to Create Error Handling User Roles."

3. Ensure that the messaging channel name you enter corresponds to an error handling user role name you have created according to information in Section 14.2, "How to Create Error Handling User Roles."

14.4 How to Configure Notification Details

To configure notification details:

1. Set up workflow notification properties in the Oracle Enterprise Manager.

   For more information about how to set up these properties, see "Configuring Human Workflow Notification Properties" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite.*

2. Configure an email messaging channel. This enables the messaging service to resolve the email address when trying to send a notification to a user.

   For more information about how to configure an email messaging channel, see the Oracle WebLogic Communication Services Developer's Guide.

3. Set the sender address for email notifications to a valid email address. Set this value in the `FROM.EMAIL.ID` property in the Error Handling Module section of the AIAConfigurationProperties.xml file. For example:

   ```
   <Property name="FROM.EMAIL.ID">Email:AIA-Error-Handling@oracle.com</Property>
   ```

   For more information about requirements for working with AIAConfigurationProperties.xml, see "How to Set Up AIA Workstation" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

14.5 How to Set Up AIA Error Handling Configuration Details

This section includes the following topic: Section 14.5.1, "What You Need to Know about Setting Up Error Handling Configurations."

To set up AIA error handling configuration details:

1. Access the Error Notifications page. To access the page,

   Access the AIA Home Page. In the Setup area, click the **Go** button. Select the **Error Notification** tab. The Error Notification page displays as shown in Figure 14–3 and Figure 14–4.

Figure 14–3 Error Notification Page (1 of 2)

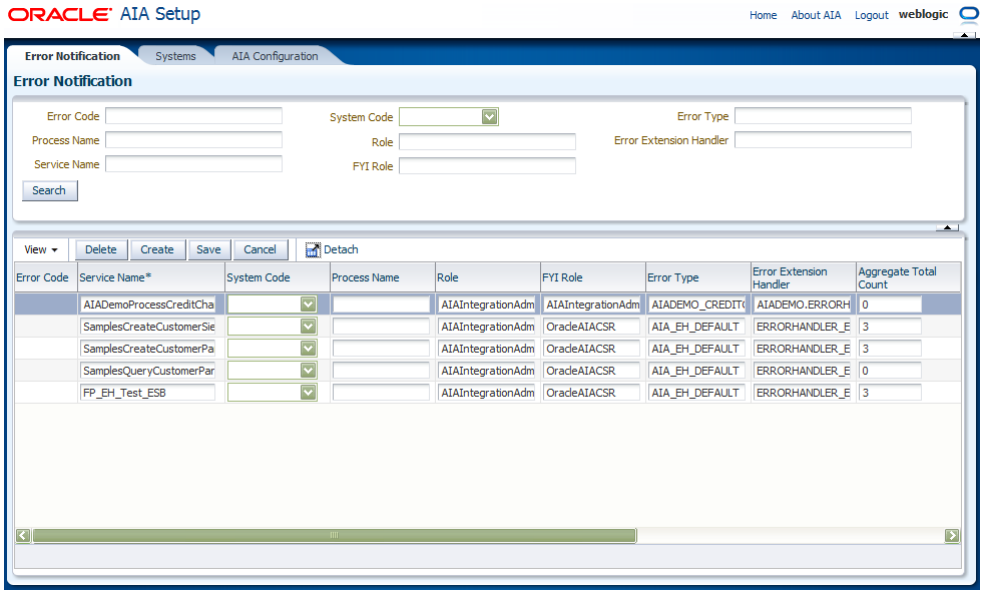
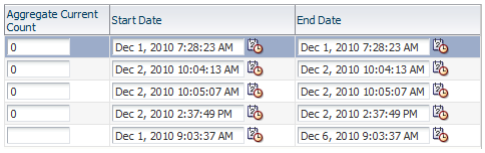


Figure 14–4 Error Notification Page (2 of 2)



2. Use the page elements to define and modify error handling configuration details for processes operating in an Oracle AIA ecosystem, including Error Notification and Oracle Worklist roles and Error Notification throttling parameters.

The error handling configurations you define on the Error Notifications page are stored in the AIA_ERROR_NOTIFICATIONS table.

Note: For a given process, if no entry is found in the AIA_ERROR_NOTIFICATIONS table, the Actor and FYI roles specified in AIAConfigurationProperties.xml are used for Error Notifications and Oracle Worklist assignments, if enabled. By default, the Actor role is set to AIAIntegrationAdmin. Therefore, you are not required to populate the AIA_ERROR_NOTIFICATIONS table unless there is an explicit need.

For more information, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

Descriptions of key elements on the Error Notification page are discussed in Table 14–1.

Table 14–1 Error Notification Page Elements

Element	Description
ErrorCode	For BPEL and Mediator process system error notifications, this is the fault code. For business errors using catch blocks, this is the business error code you are catching. This is user-defined, for example, *OUT_OF_INV*. The sample error code for system faults is `{http://schemas.oracle.com/bpel/extension}remoteFault` and `{http://schemas.oracle.com/bpel/extension}bindingFault`.
SystemCode	This is the system code of the participating application.
ProcessName	This is the business process in which the service is participating.
ServiceName	For BPEL and Mediator services, this is the name of the service that experiences the error for which you are defining error notification details. For example, *SampleBPELProcess*.
NotificationRole	If you have enabled Error Notifications, specify the user role that you want to receive Actor error notifications for a process. If you have enabled Oracle Worklist functionality, specify the role to which you want to assign Actor tasks for a process. The Actor role is responsible for taking action to correct the error that generated the notification. For Error Notifications or Oracle Worklist functionality, ensure that the role you specify here has a corresponding entry in the Oracle WebLogic Server Administration Console user store. For more information, see Section 14.2, "How to Create Error Handling User Roles." For Error Notifications functionality, ensure that the user role has an email address defined in the Oracle WebLogic User Messaging Service. For more information, see Section 14.3, "How to Associate Email Addresses with Error Handling User Roles."

Table 14–1 (Cont.) Error Notification Page Elements

Element	Description
FyiNotificationRole	If you have enabled Error Notifications, specify the user role that you want to receive FYI error notifications for a process. If you have enabled Oracle BPM Worklist functionality, specify the role to which you want to assign FYI tasks for a process. This is the role that will be given information about the error, but will not be responsible for taking any actions to correct the error that generated the notification. For Error Notifications or Oracle BPM Worklist functionality, ensure that the role you specify here has a corresponding entry in your implementation s user management store. By default, the AIA user management store is WebLogic s embedded LDAP server. For more information, see Section 14.2, "How to Create Error Handling User Roles." For Error Notifications functionality, ensure that the user role has an email address defined in Oracle User Messaging Service preferences. For more information, see Section 14.3, "How to Associate Email Addresses with Error Handling User Roles."
ErrorType	The default value is AIA_EH_DEFAULT. Use this value if you want to use the AIA default error listener as the consuming component for this error notification. Enter a unique value here if you are using extended error handling functionality. For more information about extending error handling, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.* If you want to use default and extended error handling functionality in a single error notification definition, add multiple Error Type values separated by commas. For example, *AIA_EH_DEFAULT, ORDER_FO*, where *AIA_EH_DEFAULT* is the default Oracle AIA follow-through action, and *ORDER_FO* identifies the custom JMSCorrelationID for the extended error handling implementation. The listeners and associated actions for both of these error types will be executed at run time.
ErrorExtHandler (error extension handler)	The default value is ERRORHANDLER_EXT. Use this value for the error notification if you are not using an extended handler and the fault message will be generated based on the default fault message schema. For more information about extending fault messages, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.* If you are using an extended handler to extend the fault message for this error notification, enter a unique value to identify the extension handler that will be used to enrich the fault message.
AggrCountTot (aggregation count total)	Error notification throttling must be enabled before this field value can be used to control the issuance of error notifications. For more information, see Section 15.2.2, "How to Enable Error Notification Throttling." Enter the total number of error notifications you want the system to suppress during a specific time interval for the given error scenario. The count is valid only during the specified time interval. An error notification email is issued for the first error during the time interval. After reaching the count value, the count is reset to 0 and another error notification email is issued.

Table 14–1 (Cont.) Error Notification Page Elements

Element	Description
StDatetime/EndDatetime	Error notification throttling must be enabled before these field values can be used to control the issuance of error notifications. For more information, see Section 15.2.2, "How to Enable Error Notification Throttling." Enter the start and end date-and-time intervals to which you want the count value to apply. For example, if you set the **AggrCountTot** field value to *100*, the start date and time to *30-Oct-2009 18:00:00*, and the end date and time to *01-Nov-2009 17:00:00*, one error notification email will be sent out on the first occurrence of an error in the time interval. When the count value entered in the **AggrCountTot** field is reached, the count is reset to 0 and another error notification email is issued. The date and time values used to track the time interval are derived from the database. The date and time displayed in the fields are derived from your browser time. Hover over the field values to view the database time.

14.5.1 What You Need to Know about Setting Up Error Handling Configurations

The Error Handling Framework uses runtime values and the data you enter on this page to execute the following hierarchical logic to determine the appropriate Error Notification and Oracle BPM Worklist assignment roles for an error.

1. If all four runtime values (SYSTEM CODE, ERROR CODE, SERVICE NAME, and PROCESS_NAME) are available and they map to an entry in this table, use the specified roles.
2. If the ERROR CODE, SERVICE_NAME, and PROCESS_NAME are available and map to an entry in this table, use the specified roles.
3. If the SERVICE_NAME and PROCESS_NAME are available and map to an entry in this table, use the specified roles.
4. If the SERVICE_NAME is available and maps to an entry in this table, use the specified roles.
5. If none of these values are available, the default values are fetched from the AIAConfigurationProperties.xml file.

15 Using Error Notifications

This chapter provides an overview of error notifications and describes how to set up notification throttling, how to customize notification emails, and how to disable error notifications.

This chapter includes the following sections:

- Section 15.1, "Introduction to Error Notifications"
- Section 15.2, "Setting Up Error Notification Throttling"
- Section 15.3, "Customizing Error Notification Emails"
- Section 15.4, "Disabling Error Notifications"

15.1 Introduction to Error Notifications

By default, error notification functionality is enabled. However, there are setup steps that must be completed.

For more information about setting up error notifications, see Chapter 14, "Setting Up Error Handling."

Error notification functionality generates and delivers email notifications to configured user roles.

For more information about configuring user roles for error notifications, see Section 14.2, "How to Create Error Handling User Roles" and Section 14.3, "How to Associate Email Addresses with Error Handling User Roles."

You can define a user role to receive error notifications for a specific error scenario on the Error Notifications page.

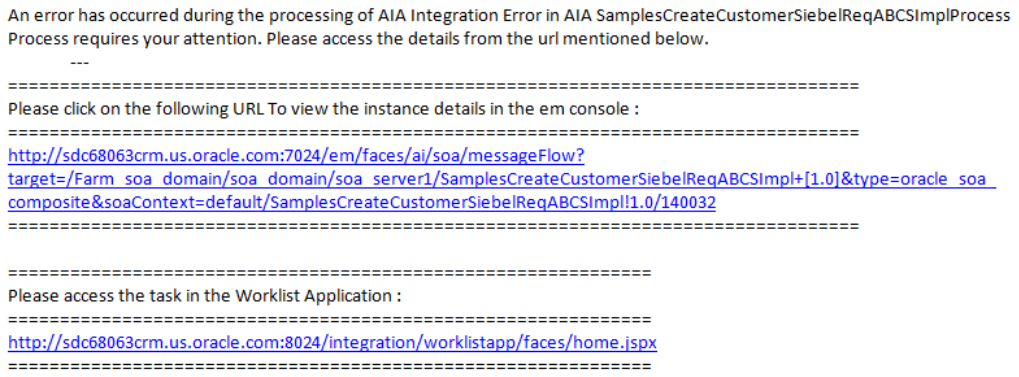
For more information, see Section 14.5, "How to Set Up AIA Error Handling Configuration Details."

You can control the number of error notifications issued for an error scenario over a specific interval of time using error notification throttling functionality.

For more information, see Section 15.2, "Setting Up Error Notification Throttling."

Figure 15–1 provides a sample error notification email.

Figure 15–1 Sample Error Notification Email


An error has occurred during the processing of AIA Integration Error in AIA SamplesCreateCustomerSiebelReqABCSImplProcess
Process requires your attention. Please access the details from the url mentioned below.

==========
Please click on the following URL To view the instance details in the em console :
==========
http://sdc68063crm.us.oracle.com:7024/em/faces/ai/soa/messageFlow?
target=/Farm_soa_domain/soa_domain/soa_server1/SamplesCreateCustomerSiebelReqABCSImpl+[1.0]&type=oracle_soa_
composite&soaContext=default/SamplesCreateCustomerSiebelReqABCSImpl!1.0/140032
==========

==========
Please access the task in the Worklist Application :
==========
http://sdc68063crm.us.oracle.com:8024/integration/worklistapp/faces/home.jspx
==========

As delivered, error notification emails contain a link to the Oracle Enterprise Manager Console, where recipients can view error information in the context of its flow trace. If the Oracle BPM Worklist is also enabled, error notification emails also contain a link to the Oracle BPM Worklist.

For more information about enabling the Oracle BPM Worklist, see Section 16.2, "How to Enable the Oracle BPM Worklist."

Error notification actor and FYI emails are generated based on content and configurations in AIAEHNotifications.xml file located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config. You can customize email content by editing this file.

For more information about customizing error notifications, see Section 15.3, "Customizing Error Notification Emails."

15.2 Setting Up Error Notification Throttling

This section includes the following topics:

- Section 15.2.1, "Introduction to Error Notification Throttling"
- Section 15.2.2, "How to Enable Error Notification Throttling"
- Section 15.2.3, "How to Configure Error Notification Throttling Parameters"

15.2.1 Introduction to Error Notification Throttling

The Error Handling Framework is capable of sending out an error notification email each time an error scenario arises, however you may choose to use error notification throttling functionality to control the number of error notification emails sent during a specific time interval for a specific error scenario.

For example, if you know that a particular high-volume transaction will be down for an extended period, you can configure notification throttling settings to control the number of error notification emails that are sent for the error scenario during the transaction s down-time. This will help you avoid the onslaught of error notification emails that would have been triggered by the down-time had throttling configurations not been set.

15.2.2 How to Enable Error Notification Throttling

Objective

Enable error notification throttling. By default, error notification throttling is disabled.

Prerequisites

Ensure that error notification functionality is enabled. By default, error notifications are enabled. To verify that this functionality is enabled, access AIAConfigurationProperties.xml located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config. Ensure that the **EH.INVOKE.NOTIF** property value is set to *true*.

Actor

Integration administrator

To enable error notification throttling:

1. Access AIAConfigurationProperties.xml in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
2. Set **Property Name = EH.AGGR.NOTIFY** to *true*.

   If error notification throttling functionality is disabled by setting this property value to *false*, an error notification email is issued each time an error scenario arises.
3. Reload updates to the AIAConfigurationProperties.xml file.

   For more information about reloading updates to AIAConfigurationProperties.xml, see "Building AIA Integration Flows" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

15.2.3 How to Configure Error Notification Throttling Parameters

Objective

Configure the parameters by which you want error notification throttling to occur for an error scenario.

Prerequisites

- Ensure that error notification functionality is enabled. By default, error notifications are enabled. To verify that this functionality is enabled, access AIAConfigurationProperties.xml located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config. Ensure that the **EH.INVOKE.NOTIFY** property value is set to *true*.
- Ensure that error notification throttling is enabled.

  For more information, see Section 15.2.2, "How to Enable Error Notification Throttling."

Actor

Integration administrator

To configure the parameters by which you want error notification throttling to occur:

1. Access the AIA Home Page. In the Setup area, click the **Go** button. Select the **Error Notification** tab. The Error Notification page displays as shown in Figure 15–2 and Figure 15–3.

Figure 15–2 Error Notification Page (1 of 2)

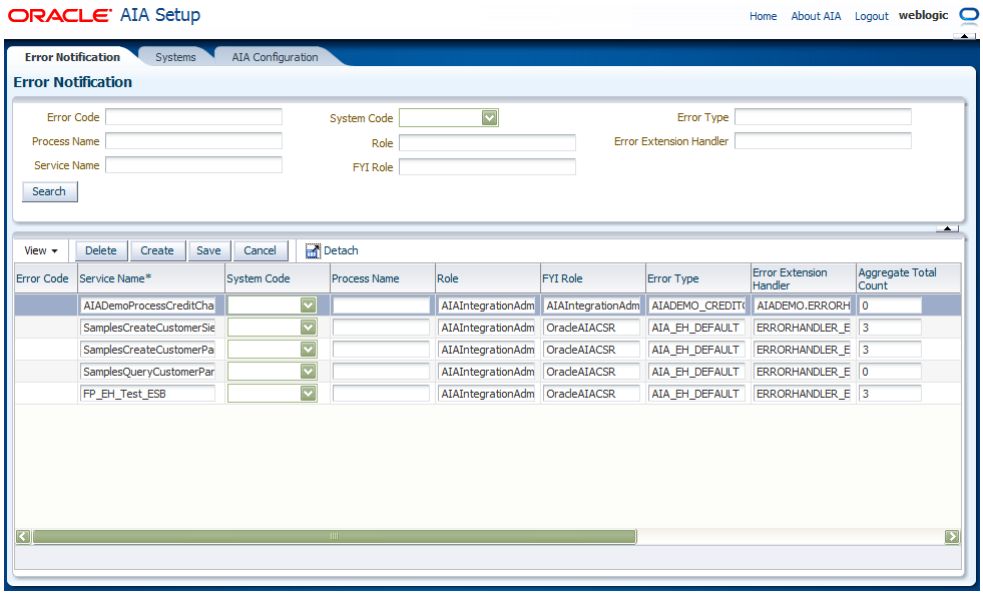
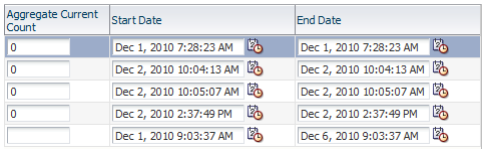


Figure 15–3 Error Notification Page (2 of 2)



2. For a given error scenario, defined by a set of ErrorCode, SystemId, ProcessName, and ServiceName values, enter **AggrCountTot**, **StDatetime**, and **EndDatetime** values.

   - In the **AggrCountTot** (aggregation count total) field, enter the total number of error notifications you want the system to suppress during a specific time interval for the given error scenario. The count is valid only during the specified time interval.

     An error notification email is issued for the first error during the time interval. After reaching the count value, the count is reset to 0 and another error notification email is issued.

   - In the **StDatetime** and **EndDatetime** fields, enter the start and end date-and-time intervals to which you want the total count value to apply.

     For example, if you set the **AggrCountTot** field value to *100*, the start date and time to *30-Oct-2009 18:00:00*, and the end date and time to *01-Nov-2009 17:00:00*, one error notification email will be sent out on the first occurrence of an error in the time interval. When the count value entered in the

AggrCountTot field is reached, the count is reset to 0 and another error notification email is issued.

The date and time values used to track the time interval are derived from the database. The date and time displayed in the fields are derived from your browser time. Hover over the field values to view the database time.

For more information about the options on the Error Notifications page, see Section 14.5, "How to Set Up AIA Error Handling Configuration Details."

15.3 Customizing Error Notification Emails

This section includes the following topics:

- Section 15.3.1, "Introduction to Error Notification Customization"
- Section 15.3.2, "How to Customize the Subject Line of Error Notification Emails"
- Section 15.3.3, "How to Customize the Body Text of Error Notification Emails"
- Section 15.3.4, "How to Customize Additional URLs Provided in Error Notification Email Body Text"

Note: These customizations will apply to all emails issued by error notification functionality.

15.3.1 Introduction to Error Notification Customization

You can customize the subject line and body text of emails issued by error notification functionality by editing the AIAEHNotifications.xml file located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config. The text of the file is shown in Example 15–1.

Example 15–1 AIAEHNotifications.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<AIAEHNotification xmlns="http://schemas.oracle.com/aia/notify" version="1.0">
  <EMAIL>
    <SUBJECT>Error in AIA#@#XPATH.{/default:Fault/default:FaultNotification/default:
     FaultingService/default:ID}#@#Process</SUBJECT>
    <BODY>An error has occurred during the processing of AIA Integration Error in AIA#@#XPATH.
{/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
Process requires your attention. Please access the details from the url mentioned below.</BODY>
  </EMAIL>
  <FYI_EMAIL>
    <SUBJECT>Error in AIA#@#XPATH.{/default:Fault/default:FaultNotification/default:
     FaultingService/default:ID}#@# Process FYI</SUBJECT>
    <BODY>An error has occurred during the processing of AIA Integration Error in AIA#@#XPATH.
     {/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
     Process requires your attention. Please access the details from the url mentioned
     below.</BODY>
  </FYI_EMAIL>
  <URL>
    ================================================================================
    Please click on the following URL To view the instance details in the em console :
    ================================================================================
    @ http://$adminHost:$adminPort/em/faces/ai/soa/messageFlow?target=/Farm_$domainName/
    $domainName/$targetServer/#@#PROPS.{compositeName}#@#+[#@#PROPS.{composite
    Revision}#@#]%26type=oracle_soa_composite%26soaContext=#@#PROPS.{compositeDN}#@#/#@#PROPS.
    {compositeInstanceID}#@#
```

```
    ================================================================================
  </URL>
  <EXT_URL>
    ============================================================
    Please access the task in the Worklist Application :
    ============================================================
    @ http://$managedHost:$managedPort/integration/worklistapp/faces/home.jspx
    ============================================================
  </EXT_URL>
</AIAEHNotification>
```

All elements can be customized. All elements shown are required for error notifications to work as designed, even if you choose to leave some of them blank.

15.3.1.1 EMAIL Element

Example 15–2 is an example of how you can customize the `EMAIL` element in AIAEHNotifications.xml to provide content that appears in error notification emails to Actor roles.

Example 15–2 EMAIL Element in AIAEHNotifications.xml

```
<EMAIL>
  <SUBJECT>Error in AIA #@#XPATH.{/default:Fault/default:FaultNotification/
   default:FaultingService/default:ID}#@#Process
  </SUBJECT>
  <BODY>An error has occurred during the processing of AIA Integration Error in
   AIA#@#XPATH.{/default:Fault/default:FaultNotification/default:Faulting
   Service/default:ID}#@#Process requires your attention. Please access the
   details from the url mentioned below.
  </BODY>
</EMAIL>
```

The `SUBJECT` element provides the subject line of the error notification email. As delivered, the subject line is set to reference the ID of the service that experienced the error.

The `BODY` element provides the body text of the error notification email. As delivered, the body text is set to reference the ID of the service that experienced the error.

15.3.1.2 FYI_EMAIL Element

Example 15–3 is an example of how you can customize the `FYI_EMAIL` element in AIAEHNotifications.xml to provide content that appears in error notification emails to FYI roles.

Example 15–3 FYI_EMAIL Element in AIAEHNotifications.xml

```
<FYI_EMAIL>
  <SUBJECT>Error in AIA #@#XPATH.{/default:Fault/default:FaultNotification/
   default:FaultingService/default:ID}#@#Process FYI
  </SUBJECT>
  <BODY>An error has occurred during the processing of AIA Integration Error in
   AIA #@#XPATH.{/default:Fault/default:FaultNotification/default:Faulting
   Service/default:ID}#@#Process requires your attention. Please access the
   details from the url mentioned below.
  </BODY>
</FYI_EMAIL>
```

The SUBJECT element provides the subject line of the error notification email. As delivered, the subject line is set to reference the ID of the service that experienced the error.

The BODY element provides the body text of the error notification email. As delivered, the body text is set to reference the ID of the service that experienced the error.

15.3.1.3 URL Element

As delivered, the URL element in AIAEHNotification.xml is used to provide a link to the composite instance flow trace details in the Oracle Enterprise Manager Console for your AIA implementation. You can customize this element to suit your implementation needs.

$hostname, $adminport, and $domain tokens shown in Example 15–4 are populated with implementation-specific values by the Oracle AIA Installer upon installation of Foundation Pack.

Example 15–4 URL Element in AIAEHNotification.xml

```
<URL>
  ================================================================================
  Please click on the following URL to view instancedetails in the em console:
  ================================================================================
  @ http://$adminHost:$adminPort/em/faces/ai/soa/messageFlow?target=/Farm_
   $domainName/$domainName/$targetServer/#@#PROPS.{compositeName}
   #@#+[#@#PROPS.{composite Revision}#@#]%26type=oracle_soa_
   composite%26soaContext=#@#PROPS.{compositeDN}#@#/#@#PROPS.{composite
   InstanceID}#@#
  ================================================================================
</URL>
```

15.3.1.4 EXT_URL Element

As delivered, the EXT_URL (external system URL) element in AIAEHNotifications.xml is used to provide a link to the Oracle BPM Worklist application, where, if enabled for AIA, the user can view their assigned AIA error-related tasks. You can customize this element to suit your implementation s needs.

$hostname and $port tokens shown in Example 15–5 are populated with implementation-specific values by the Oracle AIA Installer upon installation of Foundation Pack.

Example 15–5 EXT_URL Element in AIAEHNotifications.xml

```
<EXT_URL>
  ================================================================
  Please access the task in the Worklist Application :
  ================================================================
  @ http://$managedHost:$managedPort/integration/worklistapp/faces/home.jspx
  ================================================================
</EXT_URL>
```

For more information about enabling Oracle BPM Worklist functionality, see Section 16.2, "How to Enable the Oracle BPM Worklist."

15.3.2 How to Customize the Subject Line of Error Notification Emails

Objective

Customize the subject line of error notification emails.

Prerequisites

Ensure that error notification functionality is enabled. By default, error notifications are enabled. To verify that this functionality is enabled, access AIAConfigurationProperties.xml located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config. Ensure that the **EH.INVOKE.NOTIFY** property value is set to *true*.

Actor

Integration administrator

To customize the subject line of error notification emails:

1. Access the AIAEHNotifications.xml file located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
2. To customize the subject line used in error notification emails to Actor roles, edit the values in the `<EMAIL><SUBJECT>` element. To customize the subject line used in error notification emails to FYI roles, edit the values in the `<FYI_EMAIL><SUBJECT>` element.

   You can customize the subject line text to include multibyte characters for translation to other languages. The encoding used is UTF-8. After customized, the file must be uploaded to Oracle Meta Data Services (MDS).

   For more information about uploading content to MDS, see "Updating MDS" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*
3. To customize the AIA fault message schema value being displayed in the subject line, edit the XPATH value to use a different token. The token notation should use this format: `#@#XPATH.{ACTUAL_XPATH_VALUE}#@#`. Error notification functionality will parse this file and replace the tokens with dynamic content. Enter as many or as few tokens as needed.

   Example 15–6 is an example of how to customize the subject line of error notification emails.

Example 15–6 Customizing the Subject Line of Error Notification Emails

```
<?xml version="1.0" encoding="UTF-8"?>
<AIAEHNotification xmlns="http://schemas.oracle.com/aia/notify" version="1.0">
   <EMAIL>
      <SUBJECT>Error in AIA
      #@#XPATH.{/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
       Process</SUBJECT>
      <BODY>An error has occurred during the processing of AIA Integration Error in AIA#@#XPATH.
       {/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
        Process requires your attention. Please access the details from the url mentioned
       below.</BODY>
   </EMAIL>
   <FYI_EMAIL>
      <SUBJECT>Error in AIA
       #@#XPATH.{/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
       Process FYI</SUBJECT>
```

```
      <BODY>An error has occurred during the processing of AIA Integration Error in AIA#@#XPATH.
       {/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
       Process requires your attention. Please access the details from the url mentioned
       below.</BODY>
   </FYI_EMAIL>
   <URL>
    ================================================================================
    Please click on the following URL To view the instancedetails in the em console :
    ================================================================================
    @ http://$adminHost:$adminPort/em/faces/ai/soa/messageFlow?target=/Farm_
    $domainName/$domainName/$targetServer/#@#PROPS.
    {compositeName}#@#+[#@#PROPS.{composite Revision}#@#]%26type=oracle_soa_
    composite%26soaContext=#@#PROPS.
    {compositeDN}#@#/#@#PROPS.{compositeInstanceID}#@#
    ================================================================================
   </URL>
   <EXT_URL>
    ============================================================
    Please access the task in the Worklist Application :
    ============================================================
    @ http://$managedHost:$managedPort/integration/worklistapp/faces/home.jspx
    ============================================================
   </EXT_URL>
</AIAEHNotification>
```

For more information about the AIA fault message schema, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

4. If you have implemented fault message schema extensions, you can customize the subject line to use these schema values as well.

   For more information about extending the fault schema, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

5. Reload updates to the AIAEHNotifications.xml file.

   For more information about reloading updates to AIAEHNotifications.xml, see "Building AIA Integration Flows" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

15.3.3 How to Customize the Body Text of Error Notification Emails

Objective

Customize the body text of error notification emails.

Prerequisites

Ensure that error notification functionality is enabled. By default, error notifications are enabled. To verify that this functionality is enabled, access AIAConfigurationProperties.xml located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config. Ensure that the **EH.INVOKE.NOTIFY** property value is set to *true*.

Actor

Integration administrator

To customize the body text of error notification emails:

1. Access the AIAEHNotifications.xml file located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
2. To customize the body text used in error notification emails to Actor roles, edit the values in the `<EMAIL><BODY>` element. To customize the body text used in error notification emails to FYI roles, edit the values in the `<FYI_EMAIL><BODY>` element.

   You can customize the body text to include multibyte characters for translation to other languages. The encoding used is UTF-8. After customized, the file must be uploaded to Oracle Meta Data Services (MDS).

   For more information about uploading content to MDS, see "Updating MDS" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*
3. To customize the AIA fault message schema values being displayed in the body text, edit the XPATH value to use a different token. The token notation should use this format: `#@#XPATH.{ACTUAL_XPATH_VALUE}#@#`. Error notification functionality will parse this file and replace the tokens with dynamic content. Enter as many or as few tokens as needed.

   Example 15–7 is an example of how to customize the body text of an error notification email.

Example 15–7 Customizing the Body Text of Error Notification Emails

```
<?xml version="1.0" encoding="UTF-8"?>
<AIAEHNotification xmlns="http://schemas.oracle.com/aia/notify" version="1.0">
   <EMAIL>
      <SUBJECT>Error in AIA
       #@#XPATH.{/default:Fault/default:FaultNotification/default:FaultingService/default:ID}
       #@#Process</SUBJECT>
      <BODY>An error has occurred during the processing of AIA Integration Error in AIA
      #@#XPATH.{/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
       Process requires your attention. Please access the details from the url mentioned
       below.</BODY>
   </EMAIL>
   <FYI_EMAIL>
      <SUBJECT>Error in AIA
       #@#XPATH.{/default:Fault/default:FaultNotification/default:FaultingService/default:ID}
       #@#ProcessFYI</SUBJECT>
      <BODY>An error has occurred during the processing of AIA Integration Error in AIA
      #@#XPATH.{/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
       Process requires your attention. Please access the details from the url mentioned
       below.</BODY>
   </FYI_EMAIL>
   <URL>
    ================================================================================
    Please click on the following URL To view the instancedetails in the em console :
    ================================================================================
    @ http://$adminHost:$adminPort/em/faces/ai/soa/messageFlow?target=/Farm_
    $domainName/$domainName/$targetServer/#@#PROPS.{compositeName}#@#+[#@#PROPS.{composite
    Revision}#@#]%26type=oracle_soa_composite%26soaContext=#@#PROPS.{compositeDN}#@#/#@#PROPS.
    {compositeInstanceID}#@#
    ================================================================================
   </URL>
   <EXT_URL>
    ============================================================
    Please access the task in the Worklist Application :
```

```
    ============================================================
    @ http://$managedHost:$managedPort/integration/worklistapp/faces/home.jspx
    ============================================================
   </EXT_URL>
</AIAEHNotification>
```

For more information about the AIA fault message schema, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

4. If you have implemented fault message schema extensions, you can customize the body text to use these schema values as well.

   For more information about extending the fault schema, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

5. Reload updates to the AIAEHNotifications.xml file.

   For more information about reloading updates to AIAEHNotifications.xml, see "Building AIA Integration Flows" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

15.3.4 How to Customize Additional URLs Provided in Error Notification Email Body Text

Objective

Customize additional URLs provided in error notification email body text.

Prerequisites

- Ensure that error notification functionality is enabled. By default, error notifications are enabled. To verify that this functionality is enabled, access AIAConfigurationProperties.xml located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config. Ensure that the **EH.INVOKE.NOTIFY** property value is set to *true*.
- As delivered, error notification email body text includes a link to the Oracle BPM Worklist. To enable users to access AIA-related error tasks in the Oracle BPM Worklist, ensure that Oracle BPM Worklist functionality is enabled.

For more information about enabling Oracle BPM Worklist to work with AIA error handling, see Section 16.2, "How to Enable the Oracle BPM Worklist."

Actor

Integration administrator

To customize application links in body text of error notification emails:

1. Access the AIAEHNotifications.xml file located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
2. To customize the URLs provided in the error notification email body text, edit the values in the `<URL>` and `<EXT_URL>` elements.
3. As delivered, the `<URL>` element provides a link to flow trace details for the composite instance in the Oracle Enterprise Manager Console for your AIA

implementation. The flow trace provides details about all of the services, references, and components across composites that are participating in the flow.

For more information about viewing flow trace details in Oracle Enterprise Manager, see "Viewing the Audit Trail and Process Flow of a BPEL Process Service Component" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite.*

Figure 15–4 shows a sample error notification email text, which provides a link to flow trace details in the Oracle Enterprise Manager console.

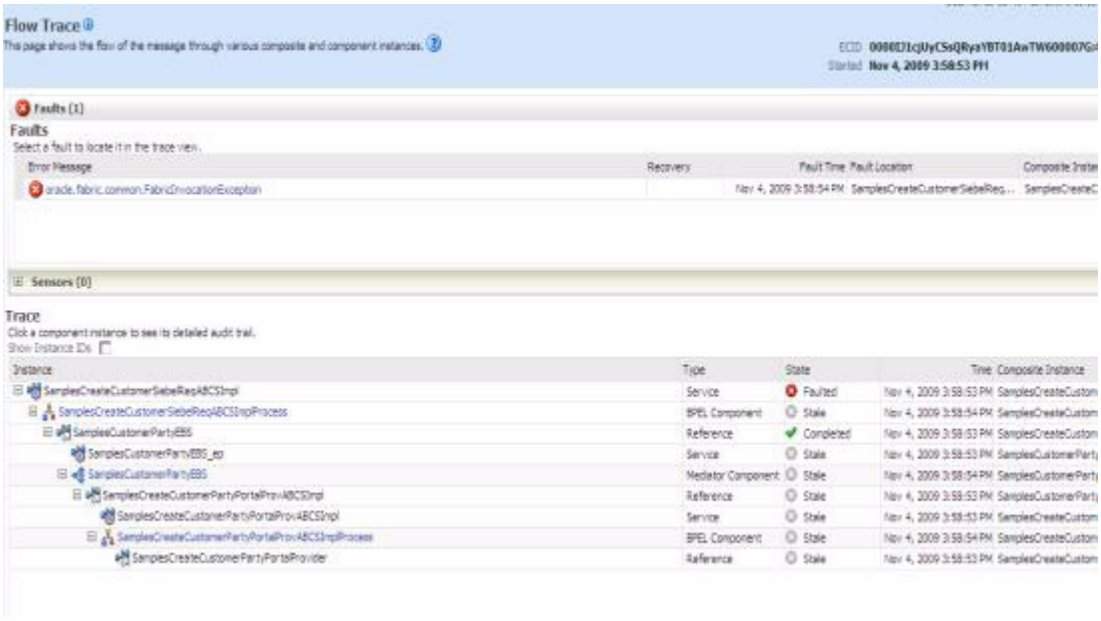
Figure 15–4 Sample Error Notification Email Text Providing a Link to Flow Trace Details in the Oracle Enterprise Manager Console

```
An error has occurred during the processing of AIA Integration Error in AIA SamplesCreateCustomerSiebelReqABCSImplProcess
Process requires your attention. Please access the details from the url mentioned below.
        ---
================================================================================
Please click on the following URL To view the instance details in the em console :
================================================================================
http://sdc68063crm.us.oracle.com:7024/em/faces/ai/soa/messageFlow?
target=/Farm_soa_domain/soa_domain/soa_server1/SamplesCreateCustomerSiebelReqABCSImpl+[1.0]&type=oracle_soa_
composite&soaContext=default/SamplesCreateCustomerSiebelReqABCSImpl!1.0/140032
================================================================================

============================================================
Please access the task in the Worklist Application :
============================================================
http://sdc68063crm.us.oracle.com:8024/integration/worklistapp/faces/home.jspx
============================================================
```

Figure 15–5 is an example of fault details being displayed on the Oracle Enterprise Console Flow Trace page.

Figure 15–5 Fault Details on the Enterprise Manager Console Flow Trace Page



We deliver the following parameters that enable this drill-down into the Oracle Enterprise Manager Console:

- `#@#PROPS.{compositeName}#@#`

- #@#PROPS.{compositeRevision}#@#
- #@#PROPS.{compositeDN}#@#
- #@#PROPS.{compositeInstanceID}#@#

For system errors configured in fault policy files, these parameters will be automatically derived to build the URL for inclusion in the error notification email. Specifically, by default, remote and binding faults are configured in the fault policy file.

For business errors, you must configure impacted processes to populate the fault message with the execution context ID (ECID). Error notification functionality will derive these parameters to build the URL based on this ECID value.

For more information about programming guidelines to populate fault messages with ECID value, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

4. As delivered, the <EXT_URL> element provides a link to the Oracle BPM Worklist, as shown in Example 15–8.

   If you are not using Oracle BPM Worklist as a part of your AIA implementation and do not want error notification emails to include this link to Oracle BPM Worklist, access AIAConfigurationProperties.xml located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config and set the **EH.INVOKE.HWF** property to *false*. This setting will remove any content expressed in the <EXT_URL> element, including the Oracle BPM Worklist default link, from error notification emails.

Example 15–8 Customizing Application Links in Body Text of Error Notification Emails

```
<?xml version="1.0" encoding="UTF-8"?>
<AIAEHNotification xmlns="http://schemas.oracle.com/aia/notify" version="1.0">
   <EMAIL>
      <SUBJECT>Error in AIA #@#XPATH.{/default:Fault/default:FaultNotification/default:
       FaultingService/default:ID}#@# Process</SUBJECT>
      <BODY>An error has occurred during the processing of AIA Integration Error in AIA #@#XPATH.
       {/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
       Process requires your attention. Please access the details from the url mentioned
       below.</BODY>
   </EMAIL>
   <FYI_EMAIL>
      <SUBJECT>Error in AIA #@#XPATH.{/default:Fault/default:FaultNotification/default:
       FaultingService/default:ID}#@#Process FYI</SUBJECT>
      <BODY>An error has occurred during the processing of AIAIntegration Error in AIA #@#XPATH.
       {/default:Fault/default:FaultNotification/default:FaultingService/default:ID}#@#
       Process requires your attention. Please access the details from the url mentioned
       below.</BODY>
   </FYI_EMAIL>
   <URL>
    ================================================================================
    Please click on the following URL To view the instance details in the em console :
    ================================================================================
    @ http://$adminHost:$adminPort/em/faces/ai/soa/messageFlow?target=/Farm_
    $domainName/$domainName/$targetServer/#@#PROPS.{compositeName}#@#+[#@#PROPS.
    {composite Revision}#@#]%26type=oracle_soa_composite%26soaContext=#@#PROPS.
    {compositeDN}#@#/#@#PROPS.{compositeInstanceID}#@#
    ================================================================================
   </URL>
   <EXT_URL>
```

```
    ============================================================
    Please access the task in the Worklist Application :
    ============================================================
    @ http://$managedHost:$managedPort/integration/worklistapp/faces/home.jspx
    ============================================================
   </EXT_URL>
</AIAEHNotification>
```

5. Reload updates to the AIAEHNotifications.xml file.

   For more information about reloading updates to AIAEHNotifications.xml, see "Building AIA Integration Flows" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

15.4 Disabling Error Notifications

By default, error notification functionality is enabled. You can disable this functionality in AIAConfigurationProperties.xml.

To disable error notifications:

1. Access AIAConfigurationProperties.xml located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
2. Set the **EH.INVOKE.NOTIFY** property value to *false*.
3. Reload updates to AIAConfigurationProperties.xml.

   For more information about reloading updates to AIAEHNotifications.xml, see "Building AIA Integration Flows" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

When error notification functionality is disabled, the Error Handling Framework does not issue error notification emails, but continues to log errors and assemble fault messages in the AIA Error Topic.

While error notifications are disabled, the AIA fault message remains available for input in the AIA Error Topic. This enables the Error Handling Framework to support a fully customized error handling solution.

16

Using the Oracle BPM Worklist

This chapter provides an overview of the Oracle BPM Worklist and describes how to enable and use the BPM Worklist.

The Oracle BPM Worklist application is a user interface (UI) that Actor roles, such as integration administrators, and FYI roles, such as customer service representatives (CSRs), can use to access details about AIA ecosystem service errors that have been assigned to them for resolution or for informational purposes only.

This chapter includes the following sections:

- Section 16.1, "Introduction to the Oracle BPM Worklist"
- Section 16.2, "How to Enable the Oracle BPM Worklist"
- Section 16.3, "How to Use the Oracle BPM Worklist"

16.1 Introduction to the Oracle BPM Worklist

The Oracle BPM Worklist application can be used to provide an error console for the Oracle Application Integration Architecture (AIA). You can enable this functionality in AIAConfigurationProperties.xml.

> **Note:** Oracle BPM Worklist functionality will be accessible even if you have not enabled it to work with Oracle AIA. However, the Oracle BPM Worklist will not include any AIA-specific error tasks.

For more information about enabling Oracle BPM Worklist functionality, see Section 16.2, "How to Enable the Oracle BPM Worklist."

The Oracle BPM Worklist application is a user interface (UI) that Actor roles, such as integration administrators, and FYI roles, such as customer service representatives (CSRs), can use to access details about AIA ecosystem service errors that have been assigned to them for resolution or for informational purposes only. Users will not receive email notifications regarding Oracle BPM Worklist task assignments unless error notifications are enabled.

For more information, see Chapter 15, "Using Error Notifications."

Based on their roles, users will be able to interact with the following types of tasks in the Oracle BPM Worklist:

- Single-approver task

  Actor roles, such as integration administrators, are assigned single-approver tasks in the Oracle BPM Worklist. Typically, this role is responsible for taking action to

resolve the error and must update the error task with activity and status details. Therefore, for Actor roles, the Oracle BPM Worklist provides an editable UI.

- FYI task

  FYI roles, such as customer service representatives, are assigned FYI tasks in the Oracle BPM Worklist. Typically, this role only needs a view of information about the status of the errored end-to-end transaction. Therefore, for FYI roles, the Oracle BPM Worklist provides a display-only UI. The FYI role is not responsible for taking any particular action to resolve the error.

The Oracle BPM Worklist provides the following error details that can assist in the troubleshooting process:

- EBMID
- EBMName
- EBOName
- Verb Code
- Business Scope Reference ID
- Business Scope Reference Instance ID
- Enterprise Service Name
- Enterprise Service Operation Name
- Sender Reference ID
- Sender Message ID
- Sender Reference Transaction Code
- Sender Object Identification ID
- Context ID
- EBOID
- Reporting Date Time
- Corrective Action
- Fault Message Code
- Fault Message Text
- Severity
- Stack
- Faulting Service ID
- Faulting Service Implementation Code
- Faulting Service Instance ID
- B2B Fault Element
- B2BMReference/B2BMID
- B2BMReference/B2BDocumentType/DocumentTypeCode
- B2BMReference/B2BDocumentType/DocumentTypeVersion
- B2BMReference/SenderTradingPartner/TradingPartnerID
- B2BMReference/ReceiverTradingPartner/TradingPartnerID

16.2 How to Enable the Oracle BPM Worklist

By default, Oracle BPM Worklist functionality is disabled. You can enable this functionality in AIAConfigurationProperties.xml.

To enable the Oracle BPM Worklist:

1. Access AIAConfigurationProperties.xml located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
2. Set the **EH.INVOKE.HWF** property value to *true*.
3. Reload updates to the AIAConfigurationProperties.xml file.

For more information about reloading updates to AIAConfigurationProperties.xml, see "Building AIA Integration Flows" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

The AIAReadJMSNotification BPEL process will now listen to the AIA Error Topic Java message service (JMS) topic, which is populated by the Error Handling Framework. Relevant errors will be aggregated by the AIAReadJMSNotification BPEL process and displayed in the Oracle BPM Worklist.

If error notification is also enabled, error notification emails will contain a link to the Oracle BPM Worklist.

For more information about error notifications, see Chapter 15, "Using Error Notifications."

16.3 How to Use the Oracle BPM Worklist

After you have been assigned an AIA error task that you need to view or act upon to resolve, you can use the details provided by the Oracle BPM Worklist to troubleshoot the error.

Access the Oracle BPM Worklist: http://<host>:<SOA server port>/integration/worklistapp.

Your assigned tasks display on the My Tasks page. You can filter your assigned tasks using various criteria and search for assigned tasks by title, priority, and status. Click an assigned task to access complete task details.

For more information, see "Using Oracle BPM Worklist" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

FYI user roles can view a task in read-only mode in the Oracle BPM Worklist.

Actor user roles can work on a task by acquiring the task. They can also enter comments against the task and update the task status. For example, when the error has been resolved, the user can set the task action to **COMPLETED**. Setting this value in the Actions field completes the task.

For more information about message resubmission, see Chapter 17, "Using the AIA Message Resubmission Utility."

17

Using the AIA Message Resubmission Utility

This chapter provides an overview and describes how to use the AIA Message Resubmission Utility. The AIA Message Resubmission Utility enables users to resubmit error messages based on these integration milestones: Queue, Topic, Resequencer, or AQ.

This chapter includes the following sections:

17.1 Introduction to the AIA Message Resubmission Utility

To use the AIA Message Resubmission Utility, you must implement error handling and recovery for the asynchronous message exchange pattern.

For more information, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

According to this implementation method, when a message cannot be delivered to a service or component in the flow of a global transaction, the message is rolled back to the appropriate source milestone. This source milestone corresponds to a Queue or a Topic, a Resequencer, or AQ. It is here that the message will be persisted until it can be resubmitted for delivery to the service or component.

At the same time, a fault is raised by the Error Handling framework and, if enabled, error notifications and Oracle BPM Worklist tasks regarding the fault are created to alert administrators.

For more information about the Oracle BPM Worklist, see Chapter 16, "Using the Oracle BPM Worklist."

For more information about error notifications, see Chapter 15, "Using Error Notifications."

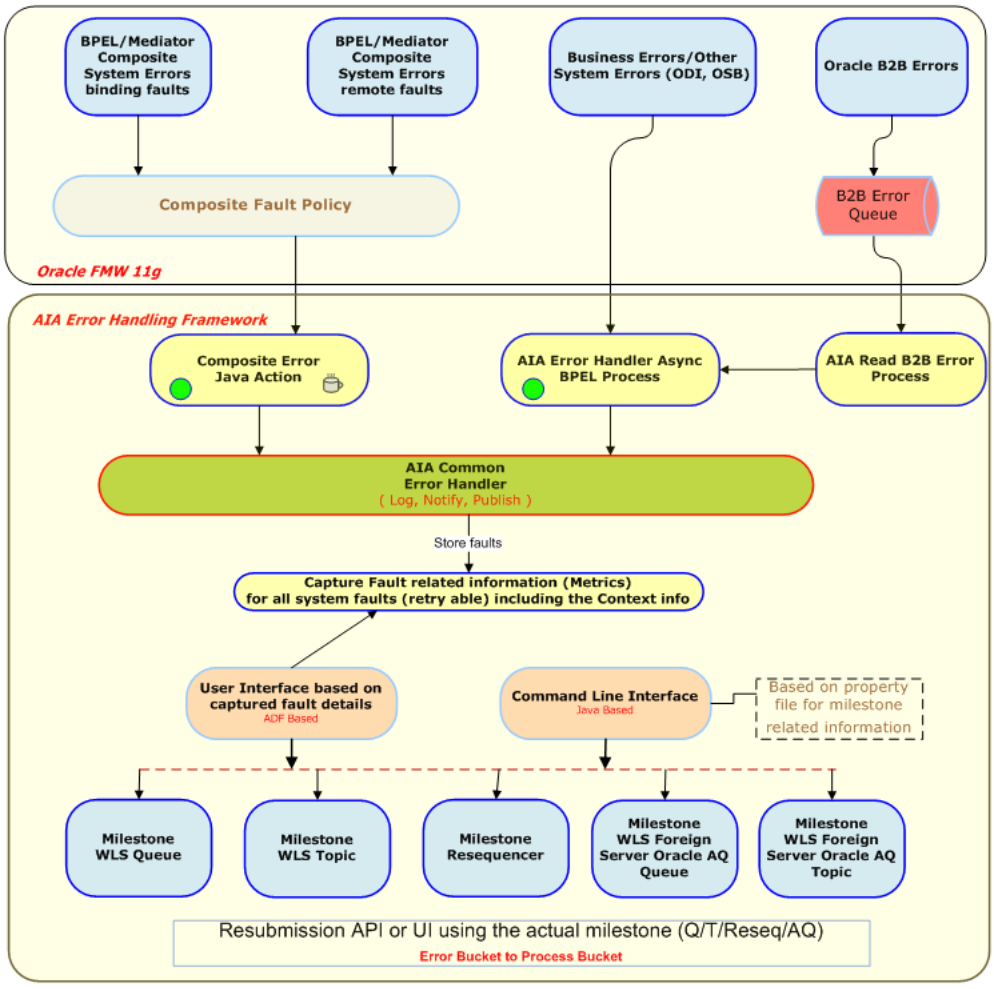
Once notified, the most natural course of action is for the administrator to bring up the failed service or component. After the service or component is back up and running, the administrator can use the AIA message resubmission utility to recover the faulted message from the source milestone. The AIA message resubmission utility changes the state of the faulted message to the **Ready** state, enabling it to be picked up by the consumer process.

Messages can be resubmitted by user interface or by command line in these ways:

- UI-based AIA Message Resubmission. For more information, see Section 17.2, "Using the AIA Message Resubmission Utility User Interface".
- Command line AIA Message Resubmission:
  - Oracle AQ store-based resubmission. For more information, see Section 17.3.1, "AQ Store Based Resubmission".
  - WLS JMS store-based (can be configured with file system or database). For more information, see Section 17.3.2, "WLS JMS based Resubmission".
  - Resequencer-based. For more information, see Section 17.3.3, "Resequencer Based Resubmission".

Figure 17–1 displays the architecture of the AIA message resubmission utility.

Figure 17–1 AIA Message Resubmission Utility Architecture



17.2 Using the AIA Message Resubmission Utility User Interface

This section discusses how to use the message resubmission utility user interface (UI) to resubmit faulted messages. The UI is integrated into the AIA Home page.

The AIA message resubmission utility figures out the milestone that is involved in an integration flow, based on the AIA Fault (canonical) and then resubmits any failed messages by connecting to the actual milestone whether it is a WLS Queue, Topic, Resequencer, or AQ.

The AIA message resubmission UI enables you query the error queue and filter for failed messages of interest, helping you to choose candidates for resubmission.

You can search for faults based on filters like Execution Context ID, Message ID, ErrorCode, Composite Name, Resource Type and other AIA context-related parameters. The ability to search and filter based on ErrorCode, for example, distinguishes system faults from business faults.

The UI also enables bulk resubmission of messages, and lets you quickly re-start a set of integration flows and track their status.

You can still use the command line utility for error resubmission. For more information, see Section 17.3, "Using the Command Line AIA Message Resubmission Utility".

To search for and resubmit faults:

1. Access the Oracle Application Integration Architecture (AIA) Home Page. In the AIA Message Resubmission Utility area, click the **Go** button. The Search Error Messages page displays, as shown in Figure 17–2 and Figure 17–3.

Figure 17–2 Search Error Messages Page (1 of 2)

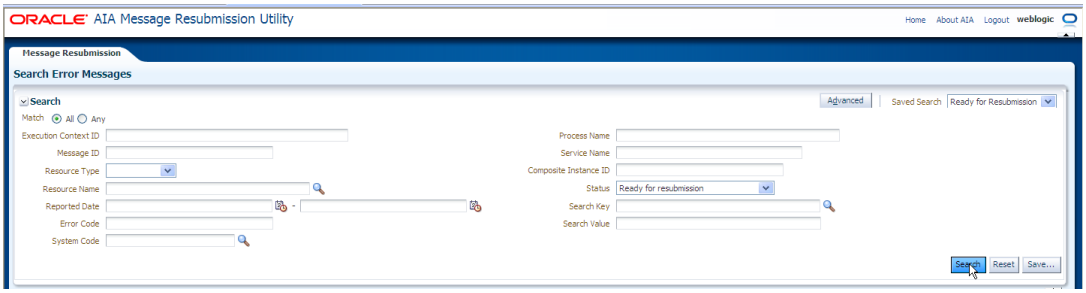
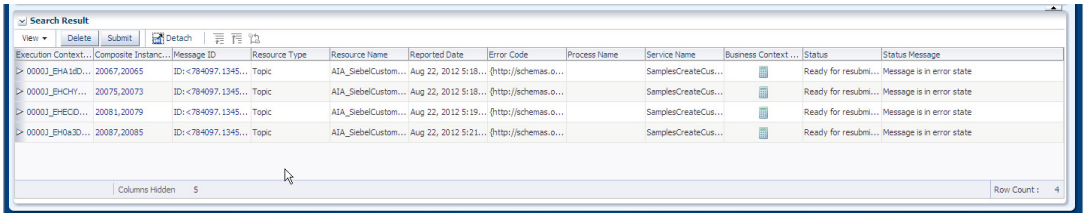


Figure 17–3 Search Error Messages Page (2 of 2)



2. On the Search page, use the page elements discussed in Table 17–1 to search for faults.

> **Tip:** You can submit a single row or multiple rows at a time. You can also submit multiple rows of the same resource type at a time. To select multiple rows, press the Ctrl key and click your mouse.

Table 17–1 Search Error Messages Page Elements

Element	Description
Match	Click the All or Any radio button if you want to match all or any of the search criteria.
Advanced	Click Advanced to open the advanced search window, shown in Figure 17–6.
Saved Search	You can save your own search criteria at any time, enabling you to quickly search on status and other criteria, such as failed messages, messages in process, ready for resubmission, or resubmitted messages.
Execution Context ID	A unique identifier used to correlate individual events as being part of the same request execution flow.
Message ID	A string value that uniquely identifies each message sent by a WLS JMS provider/AQ/Resequencer.
Resource Type	Choose from the list of available resource types: Queue, Topic or Resequencer.
Resource Name	Click the Resource Name lookup icon to see all of the available Resource Names and Resource Types. The dialog box shows the milestones that are configured in WLS and are automatically fetched, making it easy to choose the correct resource.
Reported Date	The reported date of an error message. You can filter error messages from a specific date using this field.
Error Code	For BPEL and Mediator process system error notifications, this is the fault code. For business errors using catch blocks, this is the business error code you are catching. This is user-defined, for example, OUT_OF_INV.
System Code	This is the system code of the participating application.
Process Name	This is the business process in which the service is participating.
Service Name	For BPEL and Mediator services, this is the name of the service that experiences the error for which you are defining error notification details. For example, SampleBPELProcess.
Composite Instance ID	Click the Composite Instance ID link to open the Enterprise Manager (EM) flow trace page. The EM drill down page displays the integration flow stack and all of the composites involved in the process orchestration.
Status	Statuses are: Ready for resubmission, Message is in process of resubmission, Resubmitted, Failed.
Search Key	Click the Search Key lookup icon to see all of the available application context keys. The application context correlation fields are automatically fetched and displayed. The context information comes from the source participating applications, for example, Siebel where an order is placed and an OrderID is available. This field tracks and correlates the faulted message in the integration layer.
Search Value	This is the value for a selected application context key. For example, "1001"(value of OrderID).
Search	Click Search to see results in the Search Result page, shown in Figure 17–3.
Reset	Click Reset to clear all search fields.
Save	Click Save to save your search criteria.

Search Results

Use the page elements on the Search Results page to view, submit, or delete messages. Available elements are discussed in Table 17–2.

Table 17–2 Search Results Page Elements

Element	Description
View	Click **View** and choose **Columns, Manage Columns** to add or remove columns to your Search Result view.
Delete	Select a fault and click the **Delete** button.
Submit	Select a fault and click the **Submit** button.
Detach	Click **Detach** to separate search the criteria pane and the Search result pane.
Go Up or Go To Top	Click the **Go Up** or **Go to Top** buttons to move through the list.
Show as Top button	Click a message and then click **Show as Top** to move the message to the top of the list.
Last Modified Date	The the last modified date of an error message.
Message Order	The order of an error message that is logged (the sequence number).
Error Stamping	This is the Error Type value used to stamp the JMSCorrelationID. The JMSCorrelationID is used by the custom error listener to identify fault messages that require its custom error handling. For example: AIA_EH_DEFAULT, ORDER_FO
Composite Version	The version of the composite that is deployed and active.
Service Engine	The type of service engine. It can be a BPEL process, human workflow, a decision service, Oracle mediator, or spring that executes the business logic of their respective components within the SOA composite application.
Composite Instance ID	Click the Composite ID field to see the trace flow in Oracle Enterprise Manager (EM). You must be logged in to EM to see the trace flow.
Message ID	Click the Message ID link to open a brand new page.
Business Context	Click the Business Context button to open the search details dialog which provides a search key and search value.
Status	Statuses are: Ready for resubmission, Message is in process of resubmission, Resubmitted, Failed.
Status Message	The Status Message is displayed based on the status of the error message. Ready for resubmission - Message is in error state. Resubmitted - Message is resubmitted to main queue. Failed - Exception stack trace.

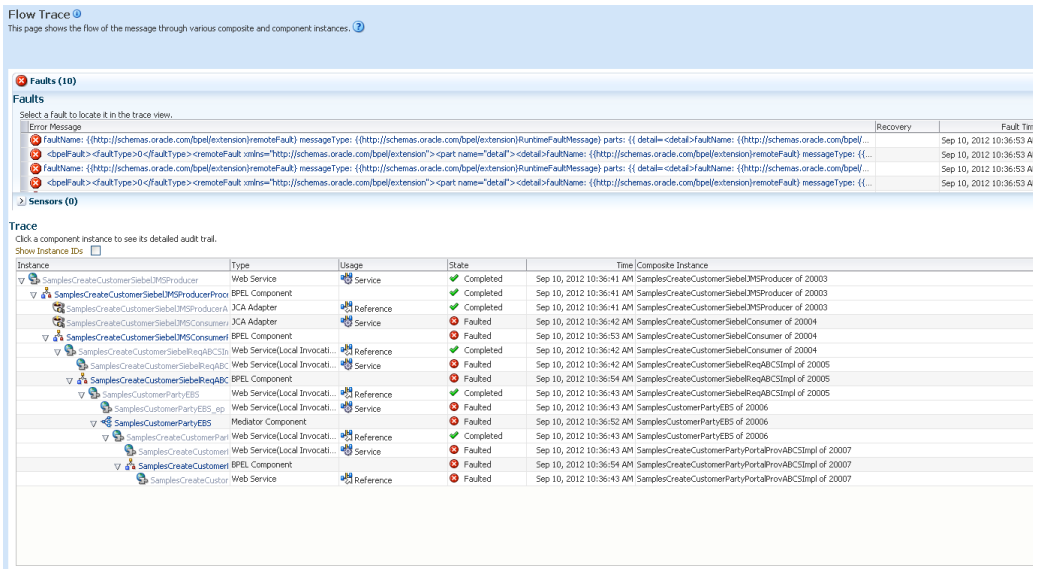
Finding Messages of Interest

You can use the **Oracle EM Flow Trace** page to review the integration flow stack and all of the composites involved in a process orchestration, enabling you to narrow down messages of interest.

You must already be logged in to Oracle EM.

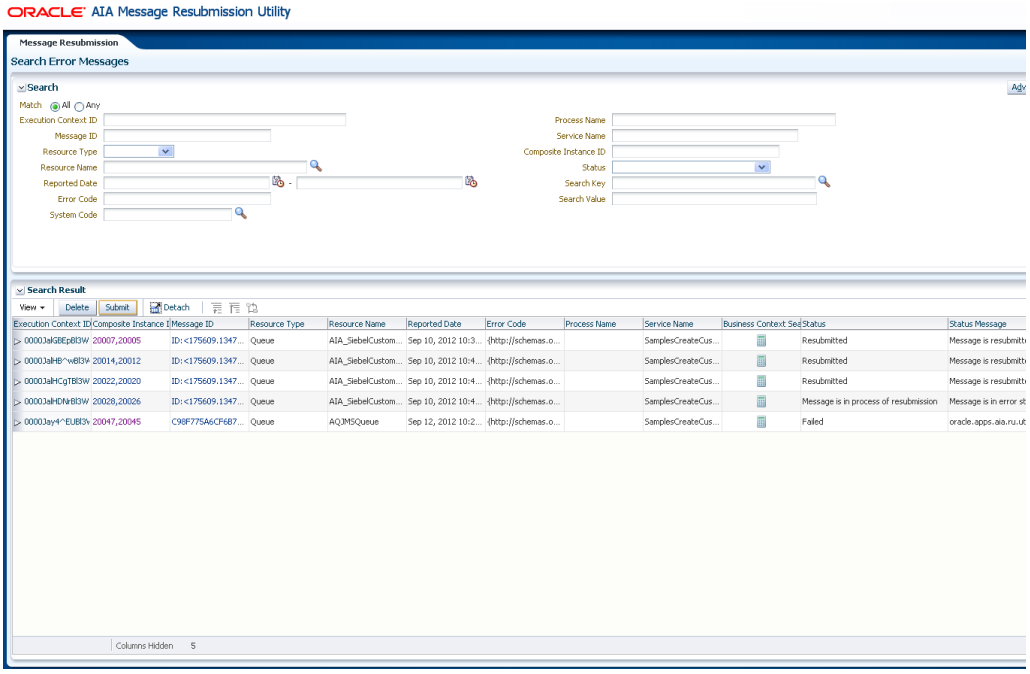
1. Click the **Composite Instance ID** link to open the Flow Trace page in EM. This page shows the flow of the message through various composite and component instances as in Figure 17–4.

Figure 17–4 EM Flow Trace Page



2. After determining a message of interest, close this window. In the **AIA Message Resubmission Utility Search Results** page, click on the message and then click the **Submit** button. The Status changes to In Progress as in Figure 17–5.

Figure 17–5 Search Result Page

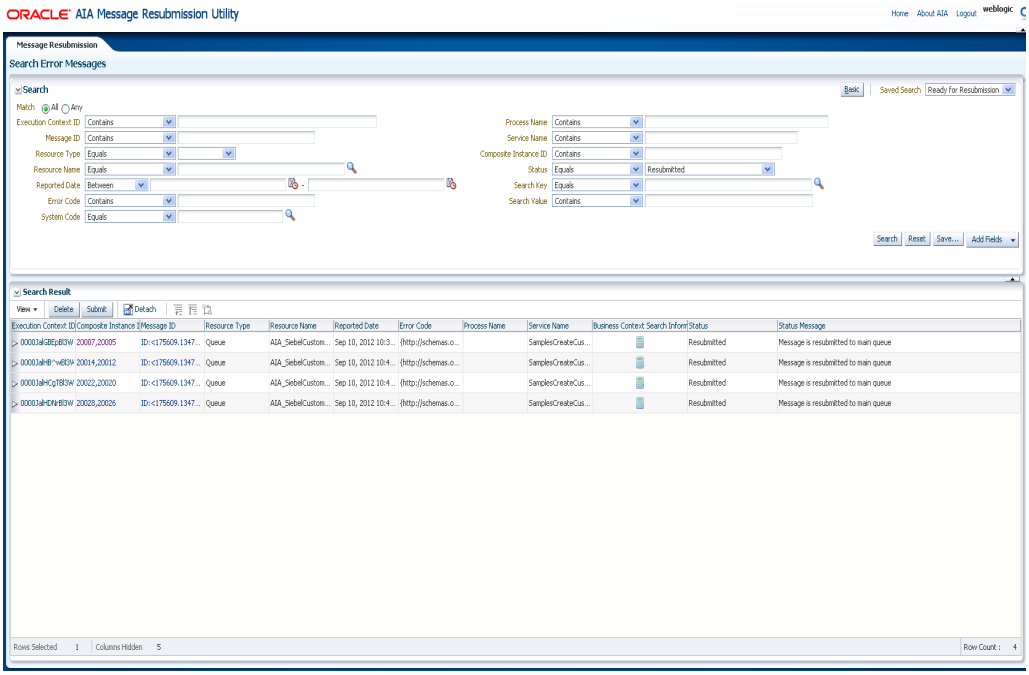


3. After the message has been submitted, click **Composite Instance ID** again to see the EM page where you can see that resubmission has happened--notice that a new set of activities were triggered in the flow trace.

Using the Advanced Search:

1. Click the Advanced button to open the **Search Error Messages** window, shown in Figure 17–6. You can search on different criteria and click **Search** to initiate a search. Click **Reset** to clear the criteria. Click **Add Fields** to add search fields.

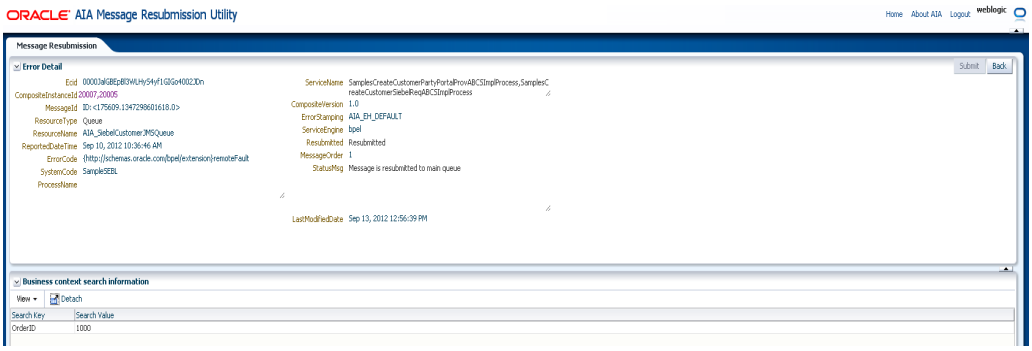
Figure 17–6 Advanced Search



Viewing Message Detail:

1. From the **Search Result** page, click the **Message ID** to open the specific message and get more detail, as in Figure 17–7.

Figure 17–7 Message ID Detail



17.3 Using the Command Line AIA Message Resubmission Utility

Messages can be resubmitted by user interface or by command line.

For more information about using the UI, see Section 17.2, "Using the AIA Message Resubmission Utility User Interface".

17.3.1 AQ Store Based Resubmission

1. For message resubmission scenarios that involve Oracle Advanced Queue, Topic, internally the MSG_RESUBMIT stored procedure is used. This procedure assumes that the message type is SYS.AQ$_JMS_MESSAGE.

   If the message type being used is not SYS.AQ$_JMS_MESSAGE, change the data type for the MSG variable in the MSG_RESUBMIT stored procedure and then recompile the procedure. You can then use the Message Resubmission Utility for resubmission based on message ID.

   For more information about configuring a queue with AQ to support resubmission, see "Configure AQ JMS Foreign Server Destinations" in *Oracle Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server*.

2. Access the Oracle AIA log file, <DOMAIN_HOME>/servers/<SOA Server Name>/logs/aia-error.log to look up the following values included in the IntermediateMessageHop element for the message that requires resubmission:

   - SenderResourceTypeCode
   - SenderResourceID
   - SenderMessageID

   For more information about these values in the context of the Oracle AIA fault message schema, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

   Alternatively, you can also look up the aia-error.log in the Oracle Enterprise Manager.

   a. Under **WebLogic Domain, <domain name>**, right-click the manage server entry (usually **soa_server1**).

   b. Navigate to **Logs, View Log Messages**. On the Log Message page, provide search criteria (optional) and click the **Search** button.

   For more information about viewing the Oracle AIA log in Oracle Enterprise Manager, see Chapter 18, "Using Trace and Error Logs."

3. AIAResubmissionUtility is available under $AIA_HOME/util, so browse to $AIA_HOME/util/AIAResubmissionUtility. Set all the required values in ResubmissionParams.properties file.

   Note: The *messageID* should be in normal format, and should not start with ID : < >

Example 17–1 Sample ResubmissionParams.properties for AQ based Resubmission

```
jms.app.admin.hostName=example.oracle.com
jms.app.admin.port=7001
jms.app.soa.url=t3://example.oracle.com:8001
```

```
jms.app.userName=weblogic
jms.app.password=password
isCluster=true
jms.resourceCFJndi=jms/aia/aiaResourceCF
jms.errorResourceCFJndi=jms/aia/aiaErrorQueueCF
resourceType=1
resourceName=AIA_SiebelCustomerJMSQueue
messageID=7109EDC5FFD9BA25E04014908FC62C90
forceResubmit=false
```

4. For Windows, execute *$AIA_INSTANCE\bin\aiaenv.bat*.

   For Linux, source *$AIA_INSTANCE/bin/aiaenv.sh*.

5. Navigate to $AIA_HOME/util/AIAMessageResubmissionUtil and execute the following:

   ```
   ant -f MessageResubmit.xml -logfile $AIA_
   HOME/util/AIAResubmissionUtility/MessageResubmit.log
   ```

   The MessageResubmit.xml script references the edited ResubmissionParams.properties file. Based on the results of the command line execution, the status for a specific message will be set in the back end. Statuses are: Ready for resubmission, Message is in process of resubmission, Resubmitted, Failed.

   Note: All of these properties are self explanatory in the ResubmissionParams.properties file. For security reasons the "Password" property should be deleted from the properties file after the execution of the command line AIA Message Resubmission Utility.

   Multiple message IDs of a particular composite can be set for the *messageID* property with comma "," as a delimiter. For example: 7109EDC5FFD9BA25E0401, 9EDC5FFD9BA25E04014908F.

17.3.2 WLS JMS based Resubmission

1. Access the Oracle AIA log file, <DOMAIN_HOME>/servers/<SOA Server Name>/logs/aia-error.log, to look up the following values included in the IntermediateMessageHop element for the message that requires resubmission:

   - SenderResourceTypeCode
   - SenderResourceID
   - SenderMessageID

   For more information about these values in the context of the Oracle AIA fault message schema, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

   Alternatively, you can also look up the aia-error.log in the Oracle Enterprise Manager.

   a. Under **WebLogic Domain, <domain name>**, right-click the manage server entry (usually **soa_server1**).

   b. Navigate to **Logs, View Log Messages**. On the Log Message page, provide search criteria (optional) and click the **Search** button.

For more information about viewing the Oracle AIA log in Oracle Enterprise Manager, see Chapter 18, "Using Trace and Error Logs."

2. You may optionally define jms.resourceCFJndi= and jms.errorResourceCFJndi= property values in the ResubmissionParams.properties file. The default values for the error jndi are fetched based on the resourceName that is provided in the properties file by using the mbean infrastructure. The connection factories are derived based on the naming standards. See the second list item below.

   - The `jms.resourceCFJndi= property` defines a resource-specific ConnectionFactory that will be used to connect to the resource error queue. In this context, a resource is a JMS queue or topic. This property cannot have multiple values, even if you have multiple connection factories. You must specify one ConnectionFactory to be used by the resubmission script.
   - If the resource name is `AIASamples_Queue` and the JNDI is `jndi/aia/AIASamples_Queue`, the ConnectionFactory property value would be `jndi/aia/AIASamples_QueueCF`.
   - The `jms.errorResourceCFJndi= property` defines a generic ConnectionFactory that will be used to connect to all resource error queues that are not explicitly defined using the `jms.resourceCFJndi= property`. If you do not define this value, it automatically uses `jms/aia/aiaErrorQueueCF` which is created during AIA Foundation Pack installation.

3. AIAResubmissionUtility is available under $AIA_HOME/util, so browse to $AIA_HOME/util/AIAResubmissionUtility. Set all the required values in ResubmissionParams.properties file.

   Note: The *messageID* should be in the format ID : < >

   Multiple message IDs of a particular composite can be set for the *messageID* property with comma "," as a delimiter. For example: ID:<7109EDC5FFD9BA25E0401>, ID:<9EDC5FFD9BA25E04014908F>.

Example 17–2 Sample ResubmissionParams.properties for WLS JMS based Resubmission

```
jms.app.admin.hostName=example.oracle.com
jms.app.admin.port=7001
jms.app.soa.url=t3://example.oracle.com:8001
jms.app.userName=weblogic
jms.app.password=password
isCluster=true
jms.resourceCFJndi=jms/aia/aiaResourceCF
jms.errorResourceCFJndi=jms/aia/aiaErrorQueueCF
resourceType=1
resourceName=AIA_SiebelCustomerJMSQueue
messageID=ID:<983029.1264581138423.0>
forceResubmit=false
```

4. For Windows, execute `$AIA_INSTANCE\bin\aiaenv.bat`.

   For Linux, source `$AIA_INSTANCE/bin/aiaenv.sh`.

5. Navigate to $AIA_HOME/util/AIAMessageResubmissionUtil and execute the following:

```
ant -f MessageResubmit.xml -logfile $AIA_
HOME/util/AIAResubmissionUtility/MessageResubmit.log
```

The MessageResubmit.xml script references the edited ResubmissionParams.properties file. Based on the results of the command line execution, the status for a specific message will be set in the back end. Statuses are: Ready for resubmission, Message is in process of resubmission, Resubmitted, Failed.

Note: All of these properties are self explanatory in the ResubmissionParams.properties file. For security reasons the "Password" property should be deleted from the properties file after the execution of the command line AIA Message Resubmission Utility.

17.3.3 Resequencer Based Resubmission

1. Faults/rejected messages which are marked as Recovery Needed in the EM Console can only be resubmitted using AIAResubmissionUtility. To get the list of faulted/rejected messages in the EM Console, navigate to SOA, click on corresponding domain and navigate to Faults and Rejected messages tab in the middle pane.
2. Make a list of all faulted instances (which are marked as Recovery Needed).
3. Get the composite name and messageID/GroupID (xpath of the messageID is defined at the design time of the resequencer) of the faulted instances.
4. AIAResubmissionUtility is available under $AIA_HOME/util. Navigate to $AIA_HOME/util/AIAResubmissionUtility and set all the required values in ResubmissionParams.properties file.

Note: The *resourceName* should be in the format default/<compositeName>!<version>.

Multiple message IDs of a particular composite can be set for the *messageID* property with comma "," as a delimiter. For example: 7109EDC5FFD9BA25E0401, 9EDC5FFD9BA25E04014908F.

Example 17–3 Sample ResubmissionParams.properties for Resequencer based Resubmission

```
jms.app.admin.hostName=example.oracle.com
jms.app.admin.port=7001
jms.app.soa.url=t3://example.oracle.com:8001
jms.app.userName=weblogic
jms.app.password=password
isCluster=true
jms.resourceCFJndi=jms/aia/aiaResourceCF
jms.errorResourceCFJndi=jms/aia/aiaErrorQueueCF
resourceType=3
resourceName=default/JMSConsumer!1.0
messageID=7109EDC5FFD9BA25E0
forceResubmit=false
```

5. For Windows, execute `$AIA_INSTANCE\bin\aiaenv.bat`.

   For Linux, source `$AIA_INSTANCE/bin/aiaenv.sh`.

6. Navigate to $AIA_HOME/util/AIAMessageResubmissionUtil and execute the following:

```
ant –f MessageResubmit.xml -logfile $AIA_
HOME/util/AIAResubmissionUtility/MessageResubmit.log
```

The MessageResubmit.xml script references the edited ResubmissionParams.properties file. Based on the results of the command line execution, the status for a specific message will be set in the back end. Statuses are: Ready for resubmission, Message is in process of resubmission, Resubmitted, Failed.

Note: All of these properties are self explanatory in the ResubmissionParams.properties file. For security reasons the "Password" property should be deleted from the properties file after the execution of the command line AIA Message Resubmission Utility.

18

Using Trace and Error Logs

This chapter provides an overview and describes how to enable trace logging, how to set trace log levels, and how to access trace and error logs.

The Oracle Application Integration Architecture (AIA) enables you to generate trace and error log files that provide a detailed view of services running in your AIA ecosystem.

This chapter includes the following sections:

18.1 Introduction to Trace and Error Logging

The Oracle Application Integration Architecture (AIA) enables you to generate trace and error log files that provide a detailed view of services running in your AIA ecosystem. These logs can be especially informative when troubleshooting service processing issues.

- Trace

  Trace logs capture chronological recordings of a service's general activities. The trace log is created by configuring the service to make an explicit call using the trace logging custom XPath or Java API.

  For more information, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

- Error

  Error logs capture a recording of errors that occur during a service's activities. No specific configurations are required to make BPEL and Mediator services eligible for error logging. The Error Handling Framework is designed to trigger an error logging event for errors occurring in any of the Oracle AIA services, whether they are BPEL- or Mediator-based. The Error Handling Framework does this logging non-intrusively.

18.2 How to Enable Trace Logging

Trace logging is enabled using configurations in the AIAConfigurationProperties.xml file located in <AIA_HOME>/aia_instances/$INSTANCE_NAME/AIAMetaData/config.

Logging can be set at the system or service level. The logging property set at the service level overrides the property set at the system level.

To enable trace logging for the entire system:

1. Access the AIAConfigurationProperties.xml file.
2. Set the **TRACE.LOG.ENABLED** property at the system level to *TRUE*.

To enable trace logging for an individual service:

1. Access the AIAConfigurationProperties.xml file.
2. Set the **TRACE.LOG.ENABLED** property for the service to *TRUE*.
3. Reload updates to AIAConfigurationProperties.xml.

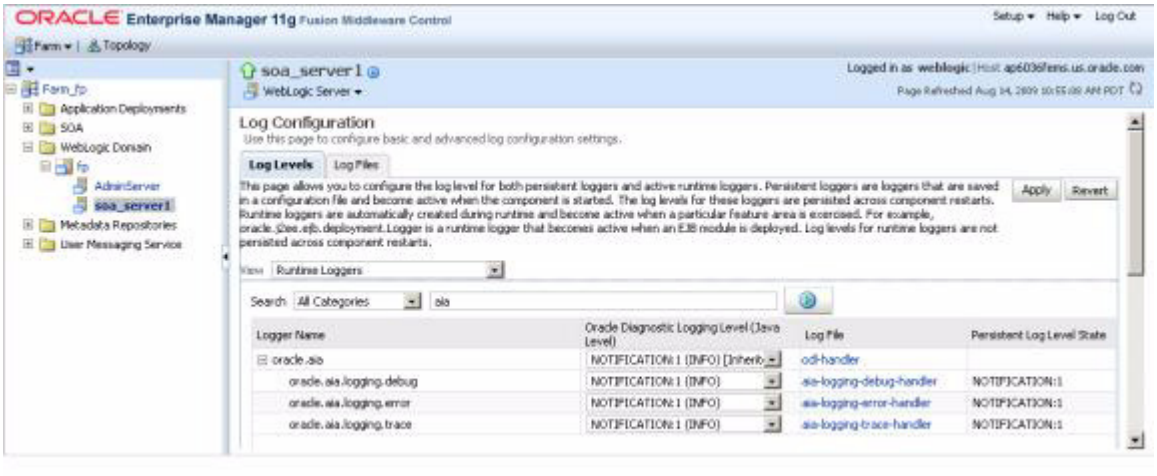
For more information about reloading updates to AIAConfigurationProperties.xml, see "Building AIA Integration Flows" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

18.3 How to Set Trace Log Levels

To set trace log levels:

1. Access the Oracle Enterprise Manager console (http://<host>:<port>/em).
2. Expand the **WebLogic** domain and navigate to your domain. Right-click on your domain and select **Logs, Log Configuration**.
3. Select the **Log Levels** tab on the Log Configuration page, as shown in Figure 18–1.

Figure 18–1 Log Configuration Page



4. In the **View** drop-down list box, select **Runtime Loggers**.
5. In the **Search** drop-down list box, select **All Categories**. Enter *aia* in the **Search** field and execute the search.

6. Locate **Logger Name** value **oracle.aia -> oracle.aia.logging.trace** and set the **Oracle Diagnostic Logging Level (Java Level)** field value accordingly. The type and amount of information written to trace log files is determined by the message type and log level specified.
7. Select from one of the values listed in Table 18–1, ordered from highest to lowest severity. The lower the severity level, the more information is written to the log file.

Table 18–1 Trace Log Levels of Severity

Severity Level	Description
INCIDENT_ ERROR:1 (SEVERE+100)	A serious problem, such as one from which you cannot recover. The problem may be caused by a bug in the product and should be reported to Oracle Support.
ERROR:1 (SEVERE)	A serious problem that requires immediate attention from the administrator and is not caused by a bug in the product.
WARNING:1 (WARNING)	A potential problem, such as invalid parameter values or a specified file that does not exist, that should be reviewed by the administrator.
NOTIFICATION:1 (INFO)	A major lifecycle event such as the activation or deactivation of a primary subcomponent or feature.
NOTIFICATION:16 (CONFIG)	A finer level of granularity for reporting normal events.
TRACE:1 (FINE)	Trace or debug information for events that are meaningful to end-users of the product, such as public API entry or exit points.
TRACE:16 (FINER)	Detailed trace or debug information that can help Oracle Support diagnose problems with a particular subsystem.
TRACE:32 (FINEST)	Very detailed trace or debug information that can help Oracle Support diagnose problems with a particular subsystem.

18.4 How to Access Trace and Error Logs

This section includes the following topics:

- Section 18.4.1, "Accessing Oracle AIA Logs in the Oracle Enterprise Manager Console"
- Section 18.4.2, "Searching for Oracle AIA Log Messages"
- Section 18.4.3, "Accessing Oracle AIA Log XML Files"

18.4.1 Accessing Oracle AIA Logs in the Oracle Enterprise Manager Console

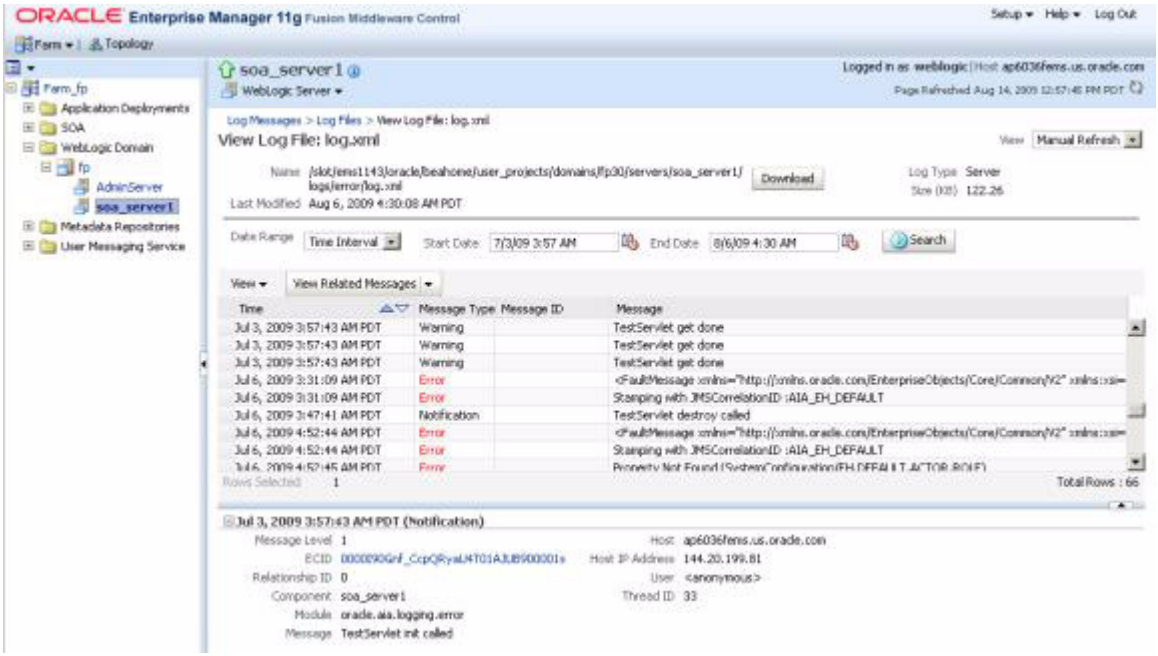
Log files can be accessed using the Oracle Enterprise Manager user interface, in much the same way that standard log files generated by various components of the Oracle SOA Suite can be handled in Oracle Enterprise Manager. Using Oracle Enterprise Manager as the user interface for the logs enables searching, sorting, and filtering of logs.

To access Oracle AIA trace and error log files:

1. Access the Oracle Enterprise Manager console (`http://<host>:<port>/em`).
2. Expand the **WebLogic** domain and navigate to your domain. Right-click on your domain and select **Logs, View Log Messages**.

3. Click the **Target Log Files** button. The error log file, aia-error.log, can be found under ${domain.home}/servers/${weblogic.Name}/logs. The trace log file, aia-trace.log, can be found under ${domain.home}/servers/${weblogic.Name}/logs.
4. To view a log file, select the file row and click the **View Log File** button, as shown in Figure 18–2.

Figure 18–2 View Log File Page



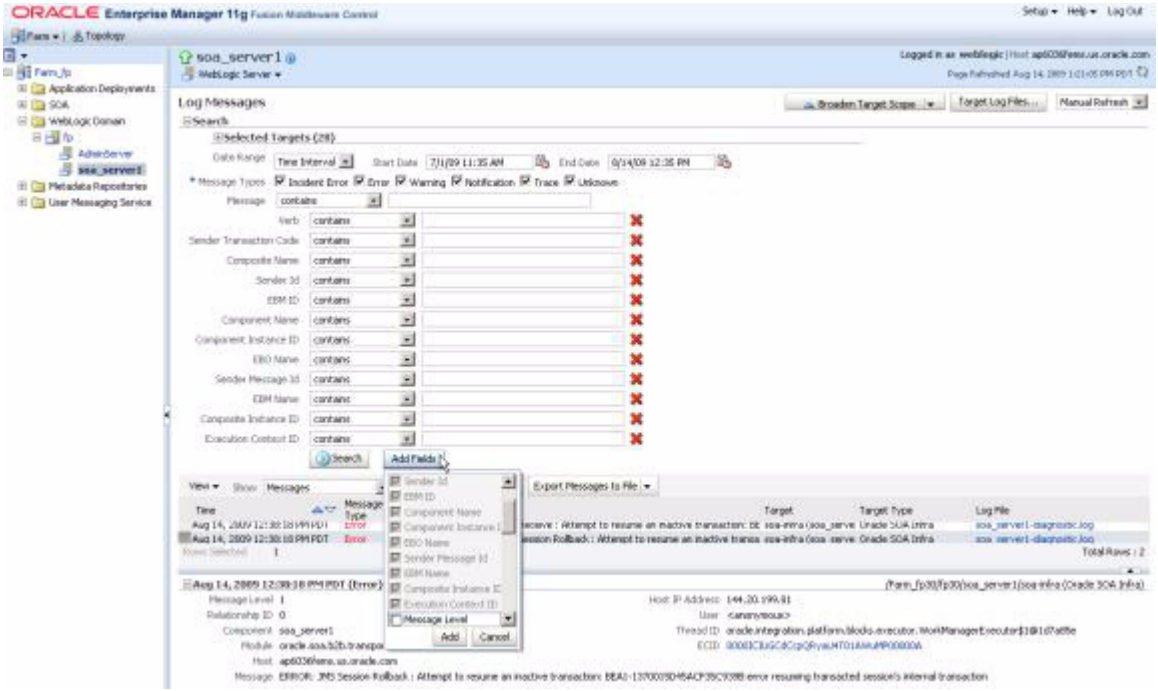
5. To download a log file, select the file row and click the **Download** button.

18.4.2 Searching for Oracle AIA Log Messages

To search for Oracle AIA trace and error log messages:

1. Access the Oracle Enterprise Manager console (`http://<host>:<port>/em`).
2. Expand the **WebLogic** domain and navigate to your domain. Right-click on your domain and select **Logs, View Log Messages**.
3. Search for specific log messages using the search parameters available in the Search area on the Log Messages page, as shown in Figure 18–3.

Figure 18–3 Log Messages Page



18.4.3 Accessing Oracle AIA Log XML Files

You can access Oracle AIA trace and error log XML files directly in the following directories:

- ${domain.home}/servers/${weblogic.Name}/logs/aia-error.log
- ${domain.home}/servers/${weblogic.Name}/logs/aia-trace.log

19

Accessing Oracle B2B Errors

This chapter describes how to access Oracle B2B error reports.

This chapter includes the following section: Section 19.1, "Accessing B2B Error Reports."

19.1 Accessing B2B Error Reports

To access error reports about failed business-to-business (B2B) transactions in the Oracle B2B console:

1. Access the Oracle B2B console: http://<soa-host>:<port>/b2b.
2. Log in using the B2B administrator user name and password.
3. Select the **Reports** tab.
4. Select the **Errors** tab.
5. Enter search criteria or leave fields blank. Click **Search**. after you have located the failed message, you can use the B2B error monitor to:
   - View details about the error.
   - View the payload of the failed message.
   - Retry processing of the failed message.

> **Note:** Business process failures, such as an order being rejected by the trading partner if the ordered item is not in stock, are not considered to be Oracle B2B errors. Response or acknowledgment messages containing these failures are treated as independent flows

For more information about Oracle B2B errors in Oracle Application Integration Architecture (AIA), see "Introduction to Error Handling for Oracle B2B Errors" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

20

Using the Code Compliance Inspector

This chapter provides an overview and describes how to use the Code Compliance Inspector (CCI). Adherence to open standards and the enforcement of good coding practices are key principles of SOA governance. CCI enables developers, architects, and QA engineers to develop integration projects that are in compliance with open standards and Oracle Application Integration Architecture (AIA) recommended best practices for design and coding.

This chapter includes the following sections:

20.1 Overview

The Code Compliance Inspector uses a pre-defined set of assertions that are based on AIA Integration Developer guidelines and the Web Services Interoperability Organization Basic Profile (WS-I BP) to check SOA projects for design consistency and good coding and documentation practices. CCI qualifies code as Compliant, Conformant, or Fully Conformant to be in sync The Open Group Architecture Framework (TOGAF) standard guidelines based on the pass criteria of the highest priority assertions.

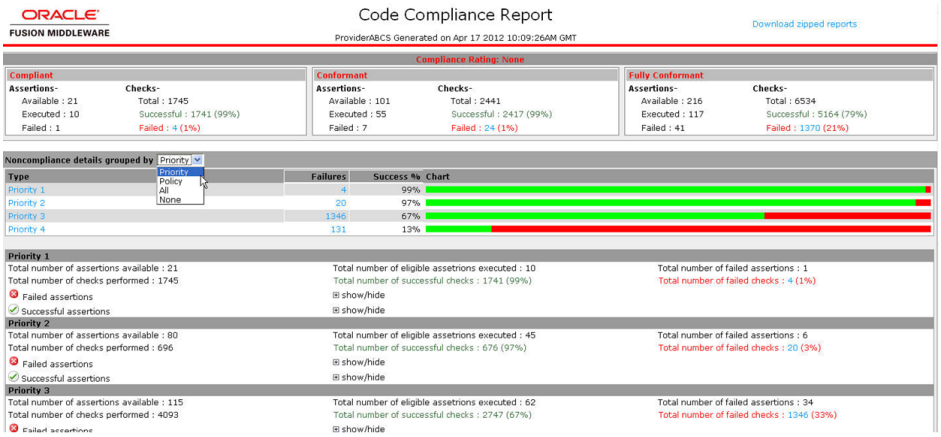
CCI is available as a JDeveloper extension and as a command-line utility. Developers will typically use the JDeveloper extension and will continuously check compliance on JDeveloper projects as they develop. The command line utility can be used to check compliance for large, multi-composite projects and produces a report that details the level of compliance, pass and failure percentages, and groups results by Priority and Policies.

CCI provides optional integration to Oracle Enterprise Repository (OER). When OER is present, CCI can synchronize results to the repository, enabling users to access the report from the OER console. Integrating compliance data into OER provides repository users with information about whether composites are compliant into the repository reports and individual asset metadata.

For more information about the optional integration with OER, see "Integration with Code Compliance Inspector" in the Oracle Fusion Middleware Integration Guide for Oracle Enterprise Repository.

Figure 20–1 shows an example of the overall Code Compliance Report.

Figure 20–1 Overall Code Compliance Report



For more information about AIA integration development guidelines, see the Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.

20.1.1 Understanding the Terminology

Table 20–1 describes the labels and concepts for Code Compliance Inspector. For more information about new terminology, see Appendix B.1, "New Terminology".

Table 20–1 Terminology

New Name	Description
Assertion	Assertions can be defined once within the AssertionCatalog.xml files and then used within one or more Policies in the Policies.xml files. For a list of the delivered assertions, see Section B.2, "Delivered Assertions".
AssertionSet	AssertionSet is informational only and is not surfaced in the JDeveloper Extension or in the Code Compliance By Policy Report.
AssertionCatalog.xml	There are two files: AssertionCatalog-AIA-<version>.xml and AssertionCatalog-WS-I-<version>.xml.
Category	Category is part of an Assertion's definition in AssertionCatalog.xml. Category is a tag within the definition of an Assertion that is largely just informational. For this release, existing Categories will become the Policy names; meaning that all Assertions tagged with a particular category will appear in the Policies.xml files using a Policy name that matches the Category.
Policy	Policies can be reused into more coarse-grained policy buckets using the <depends name=> tag.

Table 20–1 (Cont.) Terminology

New Name	Description
Polices.xml	There are two files: Policies-AIA-<version>.xml and Policies-WS-I-<version>.xml. Within the Policies.xml provided by Oracle, the policy name must match the Catalog names used within the Assertions in the AssertionCatalog.xml. Customers can modify the Policies.xml including renaming and reorganizing the Policies & Assertions.
Priority	Priorities are defined as: ■ Priority 1 assertions are the basic assertions that an integration project has to satisfy 100% to be qualified as a Compliant. ■ Priority 2 assertions are more stringent on certain design time patterns and an integration project that meets these assertions is qualified as Conformant. ■ Priority 3 assertions are the most stringent at the lowest levels of the technology, and an integration project that meets at least a certain threshold of these assertions is qualified as Fully Conformant. ■ Priority 4 assertions are recently introduced assertions that can be qualified as P3 or P2 or P1 assertions. For this release, these assertions do not play a role in the qualification of an integration project.

For more information, see Appendix B.1, "New Terminology".

20.1.2 Understanding the Delivered Catalogs

Code Compliance Inspector delivers two sets of Policies with CCI. The following Catalogs contain the Policies and Assertions that are used to check compliance for the AIA design and coding best practices:

- AssertionsCatalog-AIA-<version>.xml
- Policies-AIA-<version>.xml

The following Catalogs contain the Policies and Assertions that are used to check compliance for the WS-I Basic Profile.

- AssertionsCatalog-WS-I-<version>.xml
- Policies-WS-I-<version>.xml

The AIA catalogs and policies are for AIA customers. The WS-I catalogs and policies are for non-AIA composites. Assertions and policies are stored in these XML files. A list of pre-defined assertions can be viewed in the packaged Assertion Catalog XML files available under the ComplianceInspector/config directory.

The <version> placeholder represents the directory where assertions and policies are located. Delivered catalogs are named 10.x or 11.x, replacing <version>.

You can create custom policies and assertions in an upgrade safe manner. For more information, see Section 20.6, "Writing Custom Assertions for Code Compliance Inspector".

20.2 Invoking Code Compliance Inspector from JDeveloper

The External Tools menu option in JDeveloper provides a dialog that enables you to execute tools by adding command line executables and parameters. By adding menu and context menu options you can activate your extension.

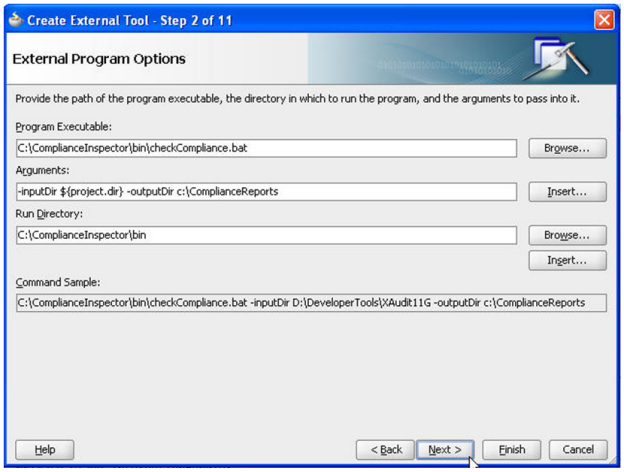
To invoke CCI from JDeveloper:

1. Select **External Tools** from the Tools menu in JDeveloper.
2. Click **New** to create a link to the external tool (in this case, Code Compliance Inspector).
3. Select **External Program** as the Tool Type, and click **Next.**
4. Click the **Browse** button, next to the **Program Executable** input field. Navigate to the location where you unzipped Code Compliance Inspector.zip, then browse to 'ComplianceInspector/bin' directory. Select the Code Compliance Inspector.bat file to open.

   In the **Arguments** input field, give necessary arguments you need to run Code Compliance Inspector.

   For example: -inputDir ${project.dir} -outputDir d:\PAoutput, as shown in Figure 20–2.

Figure 20–2 Create External Tool Page



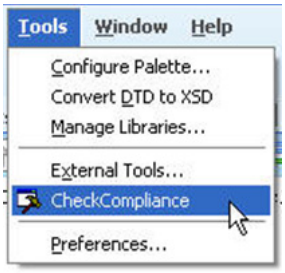
Note: **${project.dir}** is a predefined variable in JDeveloper that refers to the currently opened project directory. You can give any other absolute path instead.

For more information about the Code Compliance Inspector arguments list, see Section 20.4, "Invoking Code Compliance Inspector from a Command Line".

---

5. Click **Finish,** and then click **OK.**
6. Verify that the **CheckCompliance** option appears in the **Tools** menu as in Figure 20–3.

Figure 20–3 Tools menu



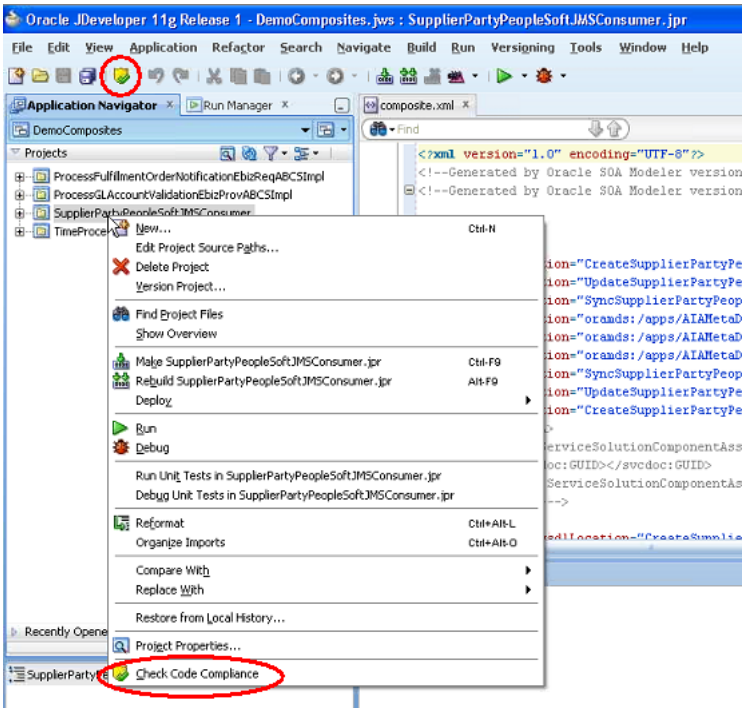
Now you can run the Code Compliance Inspector from JDeveloper for the current project.

20.3 Running Code Compliance Inspector in JDeveloper

To run Code Compliance Inspector:

1. After installing the extension, click the Tool button or right click on any project to run CCI. The menu is shown in Figure 20–4

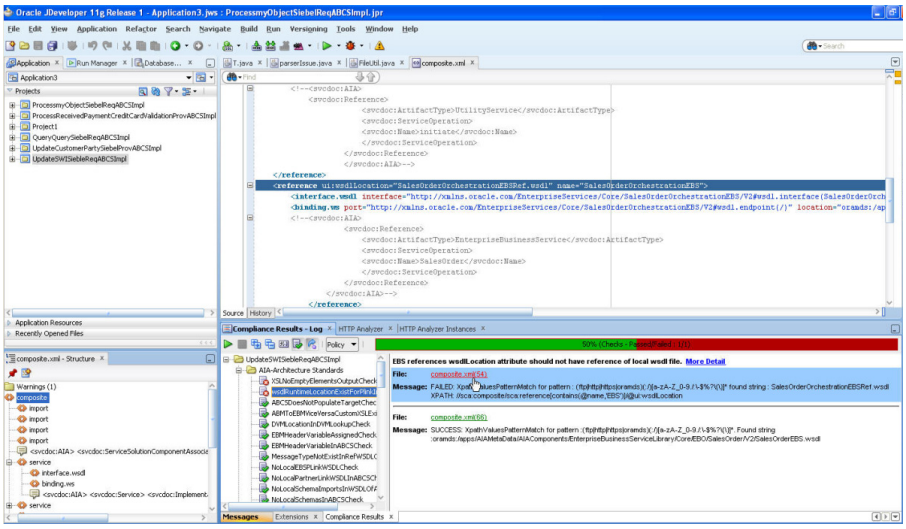
Figure 20–4 Using the menu to run CCI



Finding and Fixing Compliance Issues

Figure 20–5 shows your projects and the compliance issues that have passed and failed.

Figure 20–5 Code Compliance Page Elements



Use this page to find and fix compliance problems. Table 20–2 describes the page elements.

Table 20–2 Code Compliance Page Elements

Element	Description
Projects List	Your projects are listed in the tree. Click to view in the xml tab.
Composite.xml tab	Shows the XML for the selected project.
Composite.xml - Structure	This tab shows the structure of the composite: services, components, or references.
Compliance Results - Log	Use this tab to correct and recheck compliance issues. The buttons and fields are documented below.
Start	Click to restart after correcting.
Stop	Click to stop compliance checking in between processes.
Expand	Click to show all policies or priorities in the left-hand tree.
Collapse	Click to hide all policies or priorities in the left-hand tree.
Regular Expression Tool	Use the Regular Expression Tool button to check the value of a regular expression. Click on the assertion name in the tree and then click More Detail. Copy the XPath expression and paste it into the Regular Expression field, enter the Matching information, and click Validate. You can use this tool instead of running the whole CCI if you want.
Show/Hide Success Detail	Toggles to show or hide successful or failed and not executed processes.
Show/Hide Not Executed	Toggle on or off to show only non executed processes.
Policy drop-down	Select to view by Policy or Priority.

Table 20–2 (Cont.) Code Compliance Page Elements

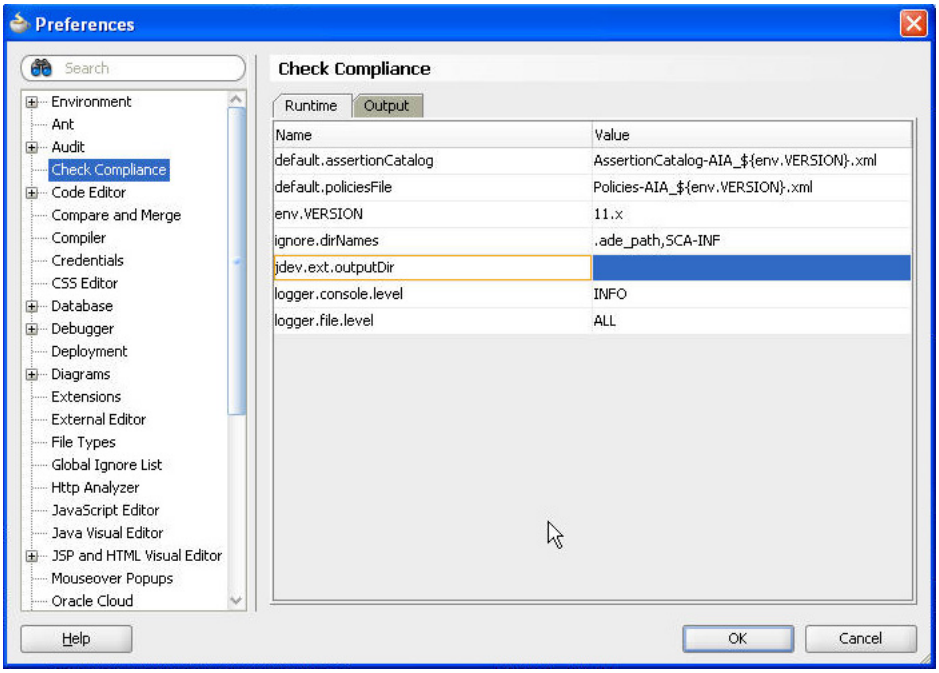
Element	Description
Failures, Success, Skipped, Total	CCI displays the counts and a bar chart. Skipped means that the assertion was not executed due any of the following reasons: 1. Target artifacts are not available. 2. Assertion definition is incorrect.

Setting defaults:

Use Preferences Figure 20–6 to set default versions, directory locations, and so forth.

1. Click **Check Compliance** in the left-hand tree to view the **Runtime** tab.

Figure 20–6 Setting Default Values



Runtime page elements are defined in Table 20–3:

Table 20–3 Runtime Page Elements

Element	Description
default.assertionCatalog	The value of this property is used as the default value for the -assertionFile switch. You can override the default value by passing the -assertionFile switch during execution of CCI.
default.policiesFile	The value of this property is used as the default value for the -testFile switch. You can override the default value by passing -testFile during execution of CCI.
env.VERSION	This is the version against which the tool is going to run. The value of this property can be overridden by introducing environment variable with name VERSION.

Table 20–3 (Cont.) Runtime Page Elements

Element	Description
filepath.hyperlink.enable	The value of this property determines if the file paths in all reports will appear as links for each file or if they will appear as plain text. If set to true, the file paths will appear as links. If set to false, the file paths will appear as plain text.
ignore.dirNames	This property is a comma separated list of directories that should not be included in the validating process. You can provide a directory name as a regular expression.
jdev.ext.outputDir	Determines where CCI will put the reports. When the extension is first installed, the value here is blank. After CCI is first run, this value will be replaced with the user's working directory. CCI takes the value of the 'ide.work.dir' property and creates a directory called 'CCI_Report' under it. This will be the default value for 'jdev.ext.outputDir'.
logger.console.level	The value of this property is used as the default value for the console handler. CCI uses the console handler to show the execution status messages to users.
logger.file.level	The value of this property is used as the default value for the file handler. CCI uses the file handler to write log information into the checkCompliance.log file. You can limit the logging information by setting various logger levels.

For more information about these fields and to make other configurations, see Section 20.5, "Configuring Code Compliance Inspector".

2. Click the **Output** tab. This tab enables you to set the output results of the html files generated by CCI. Select the Value check box to generate files, or clear the check box if you do not want to generate files. Page elements are defined in Table 20–4.

Figure 20–7 Output Page Elements

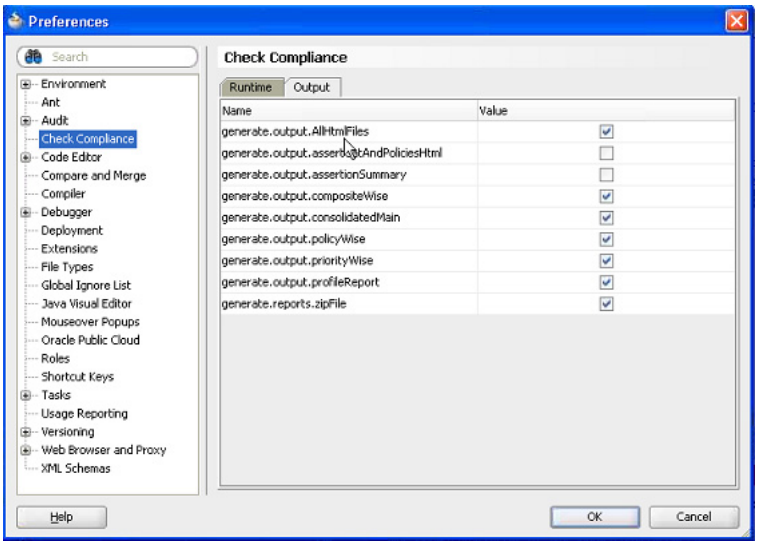


Table 20–4 Output Page Elements

Element	Description
generate.output.AllHtmlFile generate.output.assertCatAndPoliciesHtml generate.assertion.summary generate.output.compositeWise generate.output.consolidatedMain generate.output.policyWise generate.output.priorityWise generate.output.profileReport generate.reports.zipFile	If the value property is true, then only that report will be generated. If the value is set to true, then only HTML reports will be generated. If the value is set to false, then no HTML report will be generated.

For more information about these fields and to make other configurations, see Section 20.5, "Configuring Code Compliance Inspector".

20.3.1 Understanding the Reports

These are the html reports that CCI generates:

- AssertionCatalog.html and Policies.html
- ComplianceReportByPolicies.html
- ComplianceReportByPriority.html
- ComplianceReportCompositeIndex.html
- ContextProfilesReport.html

After you run CCI in a command line against a single integration project (PIP) or a directory of composites, the directory structure looks like Figure 20–8:

Figure 20–8 Sample Output Directory Structure for One PIP or Composites

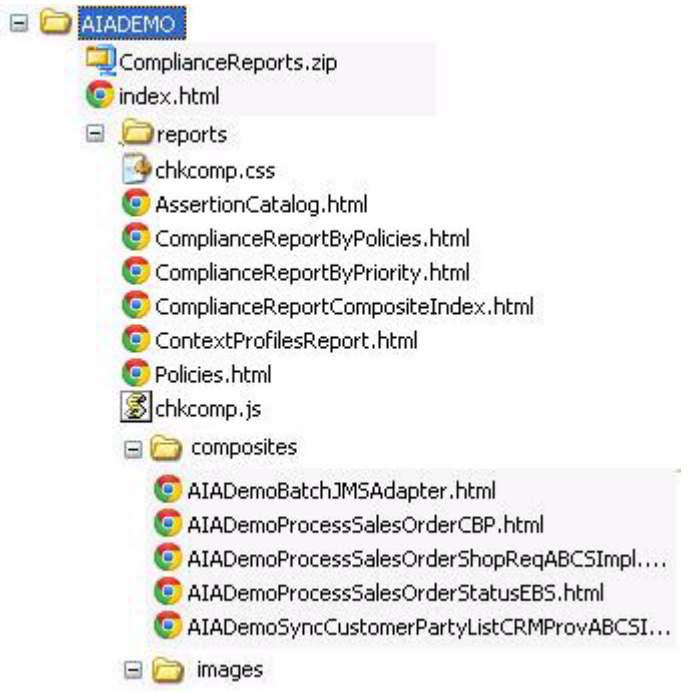
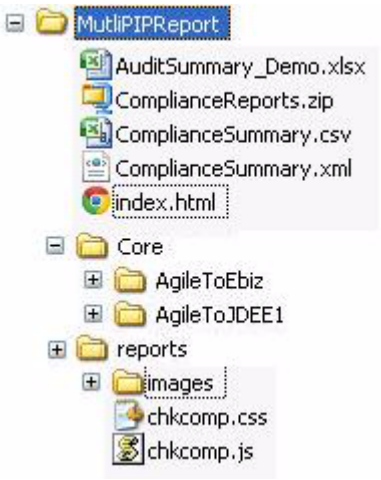


Figure 20–9 shows the directory structure that CCI generates after you run it against multiple integration projects (multiple PIPs):

Figure 20–9 Sample Output Directory Structure for Multiple PIPs



In both examples, index.html is the main consolidated report.

For more information about running CCI from the command line, see Section 20.4, "Invoking Code Compliance Inspector from a Command Line".

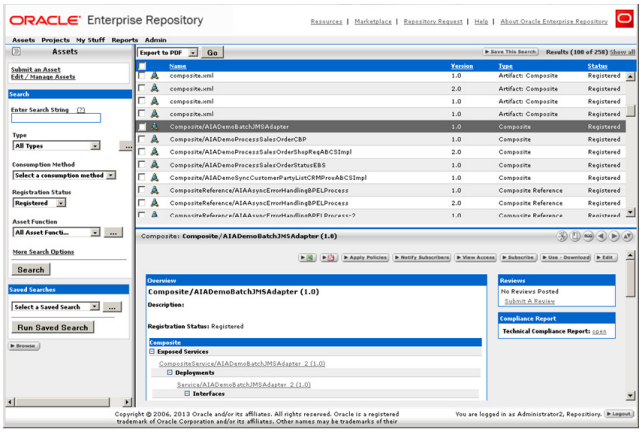
20.3.1.1 Sharing the Reports

You can post report files to a server and share the compliance results with colleagues. Copy the report files to an HTTP server and notify others of its location.

20.3.1.2 Integration with Oracle Enterprise Repository

Individual composites can be viewed in the OER asset detail page, as shown in Figure 20–10.

Figure 20–10 Viewing reports in OER



For more information about the optional integration with OER, see "Integration with Code Compliance Inspector" in the Oracle Fusion Middleware Integration Guide for Oracle Enterprise Repository.

20.3.1.3 Generating a Trend Analysis Chart

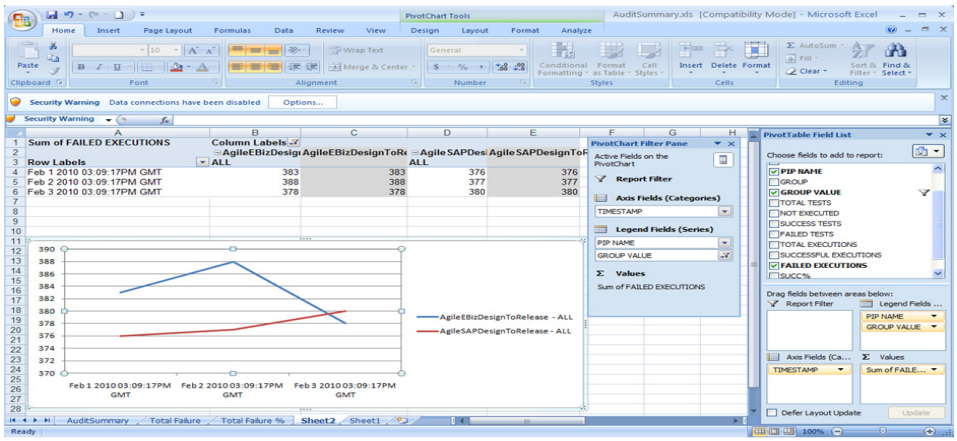
You can generate a trend analysis chart from AuditSummary.csv. The trend analysis chart shows how a selected integration project has adhered to standards over various timelines.

To generate a trend analysis chart:

1. Create a new Excel spreadsheet.
2. Import the AuditSummary.csv into cell A1 of the newly created spreadsheet.
3. Insert a new pivot chart from the Insert menu, and then select the range of source data for which you want generate the chart in the popup wizard. (Note that you can select all of the imported data from step 2, and can filter it later while presenting the charts.)
4. In the Create Pivot table with pivot chart wizard choose the New Worksheet option to generate the chart and table in the new spreadsheet.
5. You can see the PivotTable field List panel along with the empty pivot table and chart sheet. Drag and drop the DATE field to the Axis field, the PIPNAME, and GROUP VALUE field to the Legend field and the FAILED EXECUTIONS Field to the Values from the Pivot Table Field List Panel.
6. Select the GROUP VALUE Filter and select ALL in the list.

   Now you can see the trend chart as shown in Figure 20–11.

Figure 20–11 Trend Chart in Excel



Note: You can create different types of charts. For more information, see the help in Excel.

20.4 Invoking Code Compliance Inspector from a Command Line

The CCI can be invoked from a command line or from JDeveloper.

Customers who have Oracle Enterprise Repository can also use CCI. For more information, see "Integration with Code Compliance Inspector" in the Oracle Fusion Middleware Integration Guide for Oracle Enterprise Repository.

Invoke CCI with the checkCompliance.sh on Linux or checkCompliance.bat command on Windows using the following switches:

- ***-inputDir*** {Absolute path to the folder that contains composite(s)}

  This is a mandatory switch indicating where the input directory is located. If the -inputMetaFile switch is not specified, this input is not necessarily representative of a single integration project. If the -inputMetaFile switch is provided, this specifies the integration project root directory (the source folder containing the integration project folder from AIA_HOME).

- ***-outputDir*** {Output folder where the compliance report will be generated}

  This is a mandatory switch to indicate where the output reports will be stored. For example:

  If your composites live here: /tmp/cci/composites/AIADemo

  And you pass the output directory as: /tmp

  Then CCI will put the produced files here: /tmp/AIADemo

- ***-policiesFile*** {Policies file name}

  Use this optional switch to indicate which policies file CCI should run against, for example, Policies-AIA_11.x.xml. The file should be available under ComplianceInspector/lib or ComplianceInspector/config (Tool class path) or embedded in compliance.policy.engine.jar.

- ***-policy*** {Policy name}

  This is an optional switch to specify the policy to execute. If not given, the default policy from Policies.xml will be executed.

- ***-assertion*** {Assertion name}

  Use this optional switch to indicate which assertion CCI should run against. This will run the tool for a specific assertion that you have defined, for example, ABCSTargetNameSpacesCheck.

- ***-inputMetaFile*** {absolute path to the integration project MetaFile}

  Use this optional switch if you want to run reports for a specific integration project. The input metafile contains paths pointing to the specific directories that Code Compliance Inspector needs to scan so that the output results are specific for the integration project. This file contains the names of all of the services that are used in a given integration project. When this option is specified, the -inputDir switch will point to the integration project root directory since the input metafile contains the directory path relative to this root. Here are some examples:

  -inputDir=$AIA_HOME/aia -inputMetaFile <dir path of the file>/GenerateScriptInput.xml

  -inputDir=$AIA_HOME/aia -inputMetaFile <dir path of the file>/MyPIPDP.xml

- ***-inputMetaFile ALL***

  Use this optional switch if you want to run reports for all of the integration projects. When this option is specified, the -inputDir switch will point to the integration project root directory since the input metafile contains the directory path relative to this root.

  For example: -inputDir=$AIA_HOME/aia -inputMetaFile ALL

  Note: The -inputDir parameter should be the parent folder of the location specified in //Service/@location element in the GenerateInputScript.xml or //Composite/@compositedir in DP.xml. The output directory contains ComplianceSummary.csv and consolidated index.html. The integration project specific output will be segregated according to Industry/Core PIPS.

  Core PIPS will be placed under the <outputDir>/Core/<PIPName>.

  Industry PIPS will be placed under <outputDir>/Industry/<IndustryName>/<PIPName>.

  For 10g PIPs, the GenerateScriptInput.xml must be placed under the PIPS/Core/Setup/[PIP Name]/Install for Core PIPs and PIPS/Industry/[Industry Name]/Setup/[PIP Name]/Install for Industry specific PIPs for Code Compliance Inspector to process PIP specific reports.

  For 11g PIPs, [PIPName]DP.xml must be placed under the pips/[PIP Name/DeploymentPlans for Code Compliance Inspector to process PIP specific reports.

- ***-version***

  The -version flag tells you which version of CCI (CCI build date and time) you are using. This is an optional argument that displays the version information.

Examples of Invoking Code Compliance Inspector from a Command Line

Here are some examples for invoking the Code Compliance Inspector from a command line:

- Windows: checkCompliance.bat -inputDir D:\AIA\demo -outputDir D:\ComplianceOut
- Linux: sh checkCompliance.sh -inputDir /AIA/demo -outputDir /ComplianceOut

With inputMetaFile for a specific integration project:

- Windows: checkCompliance.bat -inputDir D:\AIAPIP\aia -inputMetaFile <dir path of the file>/GenerateScriptInput.xml -outputDir D:\ComplianceOut
- Linux: sh checkCompliance.sh -inputDir $AIA_HOME/aia -inputMetaFile <dir path of the file>/GenerateScriptInput.xml -outputDir /ComplianceOut

With inputMetaFile for all of the integration projects:

- Windows: checkCompliance.bat -inputDir D:\AIAPIP\aia -inputMetaFile ALL -outputDir d:\ComplianceOut
- Linux: sh checkCompliance.sh -inputDir $AIA_HOME/aia -inputMetaFile ALL -outputDir d:\ComplianceOut

20.5 Configuring Code Compliance Inspector

You can change the default configuration of the Code Compliance Inspector using the AuditorRuntime.properties file. This file can be found in the ComplianceInspector/config directory.

For example, you can generate specific html reports using the "generate" configuration properties in Table 20–5. Additionally, if you wanted to exclude directories from the process, then you can use the audit.ignore property.

Note: If any property value contains ${another-property-name}, then this value will be substituted by another property, mentioned inside ${}, of the same file.

Table 20–5 Property Names and Values

Property Name	Property Values (examples)	Description
env.VERSION	2.5	This is the version against which the tool is going to run. The value of this property can be overridden by introducing environment variable with name VERSION.
default.policiesFile	Policies-AIA_.${env.VERSION}.xml	The value of this property is used as the default value for the -testFile switch. Users can override the default value by passing -testFile during execution of Code Compliance Inspector.
default.assertionCatalog	AssertionCatalog-AIA_ ${env.VERSION}.xml	The value of this property is used as the default value for the -assertionFile switch. Users can override the default value by passing the -assertionFile switch during execution of Code Compliance Inspector.
logger.console.level	INFO	The value of this property is used as the default value for the console handler. Code Compliance Inspector uses the console handler to show the execution status messages to users.

Table 20–5 (Cont.) Property Names and Values

Property Name	Property Values (examples)	Description
logger.file.level	INFO	The value of this property is used as the default value for the file handler. Code Compliance Inspector uses the file handler to write log information into the pipaudit.log file. Users can limit the logging information by setting various logger levels.
metafile.patterns	GenerateInputScript.xml, DeploymentPlan.xml	Code Compliance Inspector uses this property when users pass the switch -inputMetaFile ALL. Code Compliance Inspector will search metaFiles in given input directory (using the -inputDir switch) based on the property value.
metafile.components.xpaths	//Service/Location	The input metafile contains paths pointing to the specific directories that Code Compliance Inspector needs to scan so that the output results are specific to the PIP. In order to get a list of directories, Code Compliance Inspector uses the property value as the XPath to execute in the meta file.
metafile.logicalname.xpaths	//PIPName|//ComponentName	In order to get the PIP Name, Code Compliance Inspector uses property value as the XPath to execute in the meta file.
metafile.locationName	DeploymentPlans	This is the directory name, where Code Compliance Inspector will look for metaFiles. Install for 2.x and DeploymentPlans for 10g.
generate.output.consolidatedMain	true/false	If the value is set to true, then HTML reports will be generated. If the value is set to false, then no HTML reports will be generated.
generate.output.profileReport		
generate.output.assertCatAndPoliciesHtml		
generate.output.priorityWise		
generate.output.policyWise		
generate.output.compositeWise		
audit.ignore.dirNames	ade_path,SCA-INF	This property is a comma separated list of directory names, which should not be included in the audit process. Users can give dirname as a regular expression too.
generate.reports.zipFile	true/false	This flag decides whether Code Compliance Inspector generates the zip file for reports in output directory or not.

Note: In order for the **Download zipped reports** link to work, make sure that ComplianceReports.zip file is found in the same directory where index.html file is.

20.5.1 Considerations

Code Compliance Inspector reads the target namespace of a BPEL or ESB process and uses it as metadata to derive AIA-related information like application name, service name, service operation, industry, version, and so forth. So if the process target namespace has not been coded as per the standards, checks will not work correctly.

Running Code Compliance Inspector with inputDir as a mapped network drive (for example, a ClearCase mapped drive) can cause performance issues. The currently recommended method is to run Code Compliance Inspector against a local source folder.

20.6 Writing Custom Assertions for Code Compliance Inspector

Code Compliance Inspector uses the AssertionCatalog-<version>.xml file that contains assertions in a "native" assertion language, in XML format. The following sections describe assertions, assertion executors, policies, and assertion and policies files.

To create a custom assertion and to execute it, follow these steps:

1. Understand the structure of the assertion.
2. Select the appropriate assertion executor and operation.
3. Write assertion definitions and parameters required by the executor operation.
4. Include the newly created assertion in a policy in the Policies.xml.

20.6.1 Understanding the Assertion Structure

Figure 20–12 shows the structure of an assertion catalog file. An assertion catalog file primarily consists of AssertionSets. These AssertionSets in turn contain different assertions and common properties associated with the assertions.

Figure 20–12 Assertion Structure

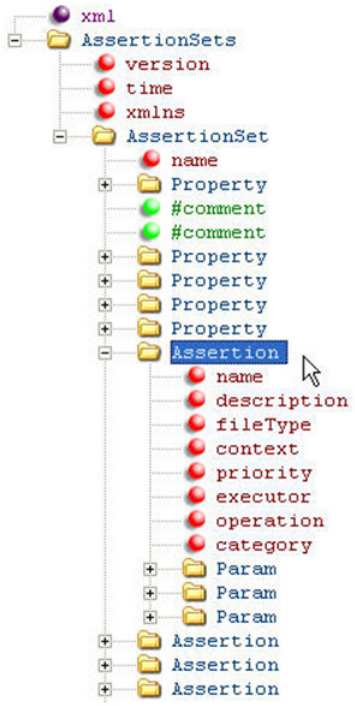


Figure 20–12 is an example of the assertion structure at a high level. To see the elements in more detail, open the AssertionCatalog-<version>.xml file using an XML editor and it should look like Figure 20–13. The elements are described in the tables that follow.

Figure 20–13 Assertion Structure Detail

Table 20–6 and Table 20–7 show the elements and attributes that make up AssertionSets and Assertions:

Table 20–6 AssertionSet Table

Element	Description
name	The name of the assertion set. This is helpful if you want to group the assertions in different categories.
Property	Property consists of two attributes: name and value. These name value pairs are used to define properties that are shared by a few assertions within the AssertionSet. One such example is the XPath prefixes. There are many prefixes that are shared by all of the assertions written for XPathExecutor. So these prefixes form a property at the AssertionSet level that is used by all assertions.
Assertion	AssertionSet can contain various assertions. The Assertion Table describes various elements and attributes that make an Assertion.

Table 20–7 Assertion Table

Element	Description
name	The name of the assertion. Note that since we do not use any ID, **the assertion name must be a unique name.** This acts as an identifier for an assertion.

Table 20–7 (Cont.) Assertion Table

Element	Description
description	The description of an assertion is a plain text description of what the assertion checks. This helps an end user understand what is actually checked and what needs to be done in order to achieve compliance.
executor	The assertion engine executes the assertions using executors. In simple terms, executors provide the base infrastructure for an assertion developer to write assertions. Various operations can be performed on executors. For example, XpathExecutor provides the assertion writer with different operations that he can perform on an XPath. For more information, see Appendix B, "Code Compliance Inspector: New Terminology & Available Assertion Executors" which provides detailed information about the executors shipped with Code Compliance Inspector.
operation	The name of any one the executor operations supported by the executors that are shipped with Code Compliance Inspector. Each executor has a set of operations. For more information, see Appendix B, "Code Compliance Inspector: New Terminology & Available Assertion Executors" which provides detailed information about the executors and operations shipped with Code Compliance Inspector.
fileType	Every assertion works on either a file or a directory. fileType attribute gives a user flexibility to perform compliance checks on specific types of files. For example, fileType="*.wsdl" means that only files with .wsdl extensions are picked for the execution of a particular assertion. You can also only select folders with the help of a special character "*'. So FileType="*" selects only those folders. Note that FileType="*" is dependent on the executor. For example, you cannot use FileType="*" with XpathExecutor since you cannot perform XPath operations on a folder whereas you can use it with FSExecutor to perform file related operations.
category	Category is an attribute used to categorize the compliance results. When there are a lot of assertions, it is easy to maintain them if they are grouped. Category is used to group a set of assertions. It is a free text attribute. The compliance results can be viewed based on a category.
priority	Priority is another attribute to group the output of audit results. Numbers can be entered here since sorting is done on results. The priority of the assertion is based on its importance. For example, priority="1" is critical-all composites must comply with this assertion and produce only Compliant nodes in the _ Audit.xml file.
context	Context is used for additional filtering of the matching files found in the input. For example, as an assertion writer you want to audit only utility BPEL processes for a certain assertion. Assuming that all utility processes have 'util' in the process name, you can specify filetype="*.bpel" and context="util". This selects only the utility bpel processes for auditing.

Note: Code Compliance Inspector executes a new assertion when the Policies-<version>.xml file has an 'Assertion' element with 'name' attribute set to be a new Assertion name. The 'Assertion' element is a child of the 'Policy' element, which is a placeholder for multiple assertions grouped together. While testing a new assertion, you can use UnitPolicy before a new test to find a place under the appropriate Policy.

For more information, see Section 20.6.6, "Writing Custom Assertions and Policies in a Policy File".

20.6.1.1 Assertion Parameters

Param is the element used to hold the assertion parameters. These are the parameters that are passed on to the executor during the execution of the Assertion. Any substitutions are performed before the parameters are sent for assertion execution.

```
<Param name=" " default=" " value=" "/>
```

Table 20–8 Assertion Parameters

Parameter	Description
name	The name of the parameter.
default	The default value that needs to be passed.
value	Variable or expression, the result of which will give the value of this parameter. If null, then the value in the "default" attribute will be considered to be an actual value.

20.6.2 Selecting an Appropriate Executor

An assertion executor is the underlying infrastructure provided by the Code Compliance Inspector to write new assertions. All of the assertions are executed by one of the executors that Code Compliance Inspector provides. These executors provide a common mechanism to execute checks. They expose different operations (as methods) that can be executed when supplied with a set of arguments (the XpathExecutor performs different XPath related operations on an XML file). For example, you can check if a value at a particular XPath location matches an input string and so on. Currently you can only write assertions using existing executors.

For more information, see Appendix B, "Code Compliance Inspector: New Terminology & Available Assertion Executors" to select the appropriate operations that will help you write your own assertion.

20.6.3 Understanding Profile Assertions

Profile Assertions are used to derive information that can be used by assertions during runtime. Profile Assertions are used to derive properties and associate them to files. These properties can then be used in assertions. Here is an example that shows how to write a Profile Assertion and what it contains.

The XML snippet in Figure 20–14 illustrates a sample profile assertion.

Figure 20–14 XML Snippet

```
<ProfileAssertion name="EBSMediaterCompsiteProfileHarvester" description="Compliance Inspector
recognizes Mediator projects based on composite.xml existance and attaches project profile to all the files which
are children of composite.xml's parent folder." executor="XMLProfiler" fileType="composite.xml"
category="ComplianceInspector-Bootstrap" priority="0" include="{ComponentName:.*EBS.*}">
   <Property name="ServiceNameId" xpath="/sca:composite/@name"/>
   <Property name="ComponentName" value="${ServiceNameId}"/>
   <Property name="ComponentVersion" xpath="/sca:composite/@revision"/>
   <Property name="ServiceImplFile" xpath="/sca:composite/sca:component[contains(@name,'EBS')]
   /sca:implementation.mediator/@src"/>
   <Property name="ServiceType" value="EBS"/>
   <Property name="ServiceTypeResp" xpath="substt()"/>
   <Property name="assignTo.1" value="..*.*" visible="false"/>
</ProfileAssertion>
```

Things to note about profile assertions:

- They are executed before any other assertions (with tag "Assertion") are executed.
- They derive property values and associate these properties to a set of files. Files to be associated can be controlled using a property named "assignTo1."
- All profile assertions must be included in an assertionset named "Bootstrap." Code Compliance Inspector executes all the profile assertions under the assertionset "Bootstrap" before executing any of the assertions. This can be compared to the functioning of a bootstrap class loader in Java.
- Executors used in profile assertions are different from the ones used in assertions.
- Apart from using the derived properties in assertions, these properties can also be used to analyze information about projects. All of the profile properties are written to a XML file named {Input dir name}_PDesc.xml which is generated in same folder that contains compliance reports. You can find out more information about your file or apply some custom style sheets to analyze the information.

Table 20–9 shows the elements and attributes that make profile assertions:

Table 20–9 Elements and Attributes for Profile Assertions

Elements and Attributes	Description
name	This is the name of the profile assertion. Note that since we do not use any ID, the profile assertion name must be a unique name. This acts as an identifier for a profile assertion.
description	The description of an assertion is a plain text description of what the assertion checks. This helps an assertion writer understand the purpose of the profile assertion.
executor	Profile Assertions are executed by executors. These are different from the assertion executors. For example, XMLProfiler helps us derive all profiling information from XML files using simple XPaths. Currently, the only available executor is XMLProfiler.
fileType	Every profile assertion works on either a file or a directory. The fileType attribute gives users the flexibility to derive information from specific types of files. For example, fileType="*.wsdl" means that only files with a .wsdl extension are picked for the execution of a particular profile assertion. Note that since XMLProfiler is the only available executor, even though the fileType can be of any extension, it must ultimately be an XML file.

Table 20–9 (Cont.) Elements and Attributes for Profile Assertions

Elements and Attributes	Description
category	Category is only used for categorizing the profile assertions.
priority	If different profiles act on the same file and have the same parameter name, then the value will be derived for the assertion that has a higher priority. Ideally this should be avoided. It's better to have different parameter names in different profiles.
include	The include attribute is used for filtering along with the fileType attribute. You can use the parameter values derived from one of the parameters in the profile assertion for filtering. The format to use is {paramName:paramValue}. For example, include="{ServiceType:ABCSImpl}" will make sure that the profiler derives the parameter ServiceType first from the list of parameters, and only if the value is ABCSImpl, it will further process the profile assertion.
exclude	The exclude attribute is used to filter files that are not to be included. Suppose you want to exclude all 'ABCSImpl' files, then exclude="{ServiceType:ABCSImpl}" will make sure that the profiler derives the parameter ServiceType first from the list of parameters and if the value matches 'ABCSImpl', it will not further process the profile assertion.

20.6.4 Profile Assertion Properties

Property is the element used to hold the profile assertion parameters. These are the parameters that are derived by the profile assertion executor and made available to assertions.

```
<Property name=" " file=" " value=" " xpath=""/>
```

Table 20–10 provides Property parameters.

Table 20–10 Property Names

Property Name	Description
Name	The name of the parameter.
Xpath	XPath in case the value is derived from an XPath using an XML file. For example: <Property name="ServiceNameId" xpath="//BPELSuitcase/BPELProcess/@id"/>
File	The file from which the parameter must be derived. This can use a derived value as well. Examples: <Property name="ServiceImplFile" xpath="//BPELSuitcase/BPELProcess/@src"/> <Property name="ReceivePLinkName" file="${ServiceImplFile}" xpath="//bpel:receive[@createInstance='yes']/@partnerLink"/>

Table 20–10 (Cont.) Property Names

Property Name	Description
Value	The value for a property can be derived without the use of an XPath. It can be a hard coded value or a derivation using previously derived values. Example: <Property name="ServiceImplFilePath" value="${pwd}/${ServiceImplFile}"/>

Certain parameters are available to every profile assertion by default. Table 20–11 shows the available parameters:

Table 20–11 Available Parameters

Parameter	Description
Pwd	Present working directory for the file selected to be profiled. Similar to pwd in unix.
Pwf	Present working file for the file selected to be profiled. Similar to pwf in unix.

20.6.5 Using Properties from Profile Assertions in an Assertion

As described in previous sections, properties can be assigned to files using profile assertions. This section describes how to use them in an assertion.

Here is a sample assertion that uses the ServiceType property derived from the profile assertion from the above section (Profile : ESBServiceProfileHarvester).

The XML snippet in Figure 20–15 shows a sample assertion that uses a derived property.

Figure 20–15 XML Snippet

```
<Assertion name='MaxExtPointsInSyncMEPCheck' description='All BPEL processes which follow SYNC Request Response pattern should not have more than 6 extension points'
  fileType='*.bpel' context='ABCSImpl,{MEP:SYNC_REQ_RESPONSE}' priority='3' executor='XPathExecutor' operation='xpathNodeCountLessThanCheck' category='AIA - Extensibility
  Standards'>
  <Param name='xpath.namespace.prefixes' value='${xpathPrefixes}' />
  <Param name='xpath.match.regxpattern' value='${syncMinExtension}' default='7' />
  <Param name='xpath.search' default='//bpel:invoke[(contains(@portType,'Extension') or contains(@portType,'Ext')) and (contains(@partnerLink,'Extension') or contains
    (@partnerLink,'Ext'))]' />
</Assertion>
```

The assertion above tries to use the context="ABCSImpl,{MEP:SYNC_REQ_ RESPONSE}"

Review this part in the example: {MEP:SYNC_REQ_RESPONSE}

The first part 'ABCSImpl' tells the assertion to filter out all of the *.bpel files that have ABCSImpl in the name.)

MEP is the property name and SYNC_REQ_RESPONSE is the property value to check for.

So during the assertion execution, only those .esbsvc files that have a MEP of value "SYNC_REQ_RESPONSE" are picked up for audit. The primary intention is a second

level filtering since sometimes you are not be able to write assertions purely relying on the naming of the files and filetypes.

20.6.6 Writing Custom Assertions and Policies in a Policy File

An assertion in a Policies-<version>.xml file is an invocation of an assertion from an assertion catalog by passing parameters to override it if required.

The simplest form of an assertion looks like this:

```
<Assertion name="NoHardWiringUnamePwdInEndpointURICheck"/>
```

The value of the attribute "name" must be the same name as that given in the "name" attribute of an assertion in the AssertionCatalog.xml file. Here no parameter values are passed from the assertion in the Policies.xml file and the default values from the assertion are taken.

Here is an example where the assertion in the Policies.xml file passes parameters to override for execution of an assertion.

Consider the sample assertion below:

```
<Assertion name="DocumentMinLengthCheck " description=" " executor=" " fileType="
" category=" " priority=" " context=" ">
 <Param name=" minLength" default=" 20" value=" "/>
</Assertion>
```

An assertion in policies.xml file can be written against the assertion in assertion catalog as:

```
<Assertion name="DocumentMinLengthCheck ">
<Param name=" minLength" value=" 30" />
<\Assertion>
```

During the execution of the above test, the default value '20' for parameter minLength is overridden by '30'.

A policy is an element used to group assertions. The compliance results of the Code Compliance Inspector are also grouped by policies along with priority and category. So the grouping of assertions with the help of policies is helpful in terms of prioritizing the compliance results.

Here is an example of a policy:

```
<Policy name="FaultPolicyRelatedSuite">
    <Assertion  name="FaultPolicyEnabledforABCSAndEBFCheck"/>
    <Assertion  name="FaultPolicyFileExistsInABCSAndEBFCheck"/>
</Policy>
```

The attribute 'name' has to have a unique value. For example, the policy name has to be unique in a Policies-<version>.xml file.

A policy can also invoke other policies. This is used for the purpose of grouping. If you have already grouped assertions under policy elements according to some criterion and now you want a wrapper policy that holds these policies under one parent policy, here is how to do that:

```
<Policy name="AllPolicy">
  <depends policyName="ABCSsecuritySuite"/>
  <depends policyName="SeedDataAndConfigSuite"/>
  <depends policyName="ESBProjectContentSuite"/>
  <depends policyName="BPELProjectContentSuite"/>
    <Assertion name="FaultPolicyEnabledforABCSAndEBFCheck"/>
```

```
    <Assertion name="FaultPolicyFileExistsInABCSAndEBFCheck"/>
</Policy>
```

For the reference mechanism stated above, the element 'depends' is used. The attribute 'policyName' in the element must contain the name of the policy.

Also note that the policy 'AllPolicy' contains both 'depends' and 'Test' elements. This signifies that the policy can invoke other policies as well as tests. If you execute Code Compliance Inspector passing -policy AllPolicy, this command will execute all policies and tests from the above example.

20.6.7 Understanding the Custom AssertionCatalog File

An AssertionCatalog file is an XML file found either in the Code Compliance Inspector's classpath (ComplianceInspector/config) or embedded in the compliance.policy.engine.jar. The jar can be found at the following location: [CodeComplianceInspector install folder]/ ComplianceInspector/lib.

The AssertionCatalog XML file found in the ComplianceInspector/config takes precedence over the one found in the jar file. The assertion file delivered by Oracle is called AssertionsCatalog.<version>.xml. This delivered AssertionCatalog.xml file can have a corresponding optional (Custom_ <<base assertion catalog file name>>) custom assertion XML file found in the ComplianceInspector/config. Additionally, users can create an assertion xml by copying and renaming the Custom_ AssertionCatalog-AIA_10.x.xml found in the ComplianceInspector/samples folder.

The AssertionCatalog.xml file contains Oracle delivered assertions that are executed by the Code Compliance Inspector. A new assertion or an override for existing assertions can be added to the Custom assertion xml file by inserting a new "Assertion" node as a child node of the "AssertionSet" element that with attribute name value same as original assertionset name. For example:

<AssertionSet name="OraleAIAAssertions"
xmlns="http://www.oracle.com/aia/PIPvalidator">

After changes to the assertion file have been made, a new Custom_ AssertionsCatalog-<version>.xml file has to be placed at the following location:

[CodeComplianceInspector install folder]/ ComplianceInspector/config.

In order for a new assertion added to the Custom_AssertionsCatalog-<version>.xml to be executed, a corresponding Assertion needs to be added in the Custom_ Policies-<version>. xml file. For more information about how to add a new test to a customer policy xml file, see Section 20.6.8, "Understanding the Custom Policies XML File".

There are three ways to override a delivered assertion:

1. Add the assertion to a custom assertion XML file found in the ComplianceInspector/config and modify the values for the parameters in the assertion. In this case, there is no need to have a test added to the custom policy XML file because there is already one in the delivered policy XML file found in the jar. By default, tests from the delivered policy XML file do not pass override parameters.
2. Add the test to a custom policy XML file found in the ComplianceInspector/config for an assertion you are going to override. Pass parameters with override values from the test for those parameters that have

variables defined in the assertion. In this case, there is no need to have an assertion added to the custom assertion XML file.

3. As a less practical way of doing it, it is possible to have a combination of the above two methods where some parameters are overridden by passing them from the test in the custom policy XML file while others are overridden by directly changing them in the assertion added to the custom assertion XML file.

20.6.8 Understanding the Custom Policies XML File

A Policies.xml file is an XML file found either in the Code Compliance Inspector's classpath (ComplianceInspector/config) or embedded in the compliance.policy.engine.jar. The jar can be found at the following location: [CodeComplianceInspector install folder]/ComplianceInspector/lib. The Policies.xml file found in the ComplianceInspector/config takes precedence over the one found in the jar file. Oracle delivers a Policies-<version>.xml file for each major SOA release, for example 10g, 11g.

The delivered Policies.xml file contains a top level element called 'Validator.' This also has an attribute called 'default'. The value in the default attribute dictates which policy must be invoked by default when Code Compliance Inspector is run. This value can be overridden by the user while running Code Compliance Inspector by giving the '-policy' option and the name of the policy. Here is an example:

```
<Validator xmlns="http://www.oracle.com/aia/PIPvalidator" default="all">
```

The delivered Policy.xml file can have a corresponding optional (Custom_<<base policy file name>>) custom policy file found in the ComplianceInspector/config. Users can create a customer policy by copying and renaming Custom_Policies-AIA_10.x.xml found in the ComplianceInspector/samples.

The Custom_policies xml file contains an overriding or new set of Assertions that need to be executed by Code Compliance Inspector in addition to those already found in the delivered Policies.xml. A new Assertion can be added by adding an Assertion node under an existing Policy. This way, the new Assertion will be executed under this policy in addition to existing assertions from this policy found in the delivered Policies.xml file.

In summary, the delivered Policies-AIA_11.x.xml file contains assertions that are executed when Code Compliance Inspector runs. An assertion from Policies-AIA_11.x.xml invokes an assertion from AssertionCatalog-AIA_11.x.xml. Parameters can be passed from an Assertion in Policies file to an Assertion in AssertionCatalog file if required. So, only those assertions from AssertionCatalog-AIA_11.x.xml for which there is a corresponding assertion in the Policies-AIA_11.x.xml are executed. Note that the same assertion can be invoked by multiple policies. This can be used if there is a need to pass different values to the same parameters of the same assertion. This will override the default values for these parameters.

20.6.9 Delivered Assertions & Policies

Oracle delivers dedicated Policies XML and Assertion Catalog XML files for every release. You can download release-specific Policies XML and Assertion Catalog XML files from the ComplianceInspector/lib directory. The dedicated release-specific files can be passed to Code Compliance Inspector using the -policiesFile command line option, accepting the policies xml file name as an argument. When you pass the -policiesFile, Code Compliance Inspector will try to identify the assertion catalog file name based on the policies file naming convention.

To avoid any annotations and merge/patch problems, we recommend that you only make changes in custom Policies XML and/or Assertion Catalog XML files that the engine will read as overriding files. Code Compliance Inspector will automatically identify custom files based on file naming patterns from the same location where Oracle delivered base test and assertion files are found, for example, in ComplianceInspector/config.

Some examples for policy and assertion file name matching patterns are:

```
Policies-AIA_11.x.xml (Custom_ Policies-AIA_11.x.xml)
AssertionCatalog-AIA_11.x.xml (Custom_ AssertionCatalog-AIA_11.x.xml)
```

For examples of adding and modifying assertions and policies, see the Custom_ AssertionCatalog-AIA_10.x.xml and Custom_ Policies-AIA_10.x.xml files found in the ComplianceInspector/sample folder.

20.6.10 Adding and Modifying Assertions in a Custom Assertion File

To add a new assertion for execution by Code Compliance Inspector, add it in the Custom Assertion file. A corresponding assertion needs to be added in the Custom Policies XML file in order for a new assertion to be executed.

To customize the Oracle delivered assertion, copy this assertion from the base Assertion file and paste it in the Custom Assertion file. Edit the parameters of the assertion as needed in the Custom Assertion file.

To remove an assertion from execution by Code Compliance Inspector, copy the assertion from the base Assertion file and paste it in the Custom Assertion file. Replace the value for the executor attribute of the pasted assertion to, for example, NAExecutor.

20.6.11 Adding and Modifying Assertions in a Custom Policies XML File

To add a new assertion under the existing policy, add an assertion node under the existing policy element in Policies XML file. To customize an Oracle delivered assertion, copy the assertion from the base Policies XML file and paste it in the Custom Policies XML file. Edit parameters as needed in the Custom Policies XML file.

To remove a test from execution by Code Compliance Inspector, copy the test from the base Policies XML file and paste it in the Custom Policies XML file. Add an active attribute with the value set to false, for example:

```
<assertion name="ABCSTargetNsCheck" active="false">.
```

The Assertions.html and Policy.html files found in the <<output directory>>/reports folder will have a different text style to display a new or customized assertion executed by Code Compliance Inspector.

20.6.12 Executing Newly Created and Customized Assertions

In order to execute newly created or customized assertions, make sure that the base and custom Assertion and Policies XML files are found in the ComplianceInspector/config folder. Code Compliance Inspector will automatically detect base and custom files and execute new assertions in addition to those available in the base Assertion file as well as customized assertions.

20.7 Executing Code Compliance Inspector for Integration Projects

This section describes how to host technical compliance reports in a central location for all of the integration projects in a release vehicle. Currently these files can only be executed on Linux.

File name: $CCI_Home/ComplianceInspector/bin/aia_setEnv.sh

This file is used to set up environment variable values to execute the Code Compliance Inspector. Table 20–12 describes the variables.

Table 20–12 Variables

Variable	Description
VERSION	This variable is used to specify the release number. For example: 2.5 or 3.1.
INPUTDIR	This variable is used to specify the path of the root directory where the AIA source is available. Example: /scratch/aia/AIASource/RV2.5/aia
REPORT_STAGING_HOME	This variable is used to specify the directory location where reports will be generated by the tool. Example: /scratch/aia/AIAReports_stage
REPORT_PRODUC_HOME	This variable is used to specify the directory where reports are copied to be viewed by report viewers. Example: /scratch/aia/AIAReports
AIA_HOME	This variable is used to specify the directory where the Code Compliance Inspector is installed. Example: /scratch/aia/

A

XML Structures of Exportable CAVS Definitions and Instances

This appendix provides XML structures of exportable Composite Application Validation System (CAVS) definitions and instances.

This appendix includes the following sections:

- Section A.1, "Definition.xml"
- Section A.2, "Instance.xml"

A.1 Definition.xml

The structure of the Definitions.xml file created by the CAVS definition export feature is shown in Example A–1.

This export feature should be used to migrate definitions between instances running on the same version of Oracle Application Integration Architecture (AIA) Foundation Pack.

Use this structure as a reference if you receive a validation error when importing definitions.

Edit this structure to create new definitions for importing to an Foundation Pack instance.

For more information about the definition export and import feature, see Chapter 12, "Exporting and Importing CAVS Definitions and Instances."

Example A–1 Definition.xml

```
<DefinitionsList>

<!-- The section below is for one test/simulator definition. This includes all definition details
as well as XPATH conditions set by the user.
For each definition the section below will be repeated -->

<DefinitionsViewRORow>
   <DefinitionId>[Definition ID that was set in the previous environment. During import, the
    target system will generate a new ID for this field]</DefinitionId>
   <Type>[Test|Simulator]</Type>
   <Description>[String. Description of the test or simulator]</Description>
   <State>[Locked|Unlocked]</State>
   <ServiceType>[Synchronous|Notify|Asynchronous two way]</ServiceType>
   <UrlEndpoint>[URL]</UrlEndpoint>
   <SoapAction>[String. Valid soap action from the wsdl of the above URL]</SoapAction>
   <SoapTransportType>[HTTP]</SoapTransportType>
```

```
   <MessageRequest>[SOAP Message. Request message along with CAVS SOAP
    envelopes]</MessageRequest>
   <MessageResponse>[SOAP Message. Response message along with CAVS SOAP
    envelopes]</MessageResponse>
   <Delay>[Integer greater than -1. Only in the case of ServiceType Asynchronous two way]</Delay>
   <ServiceName>[String]</ServiceName>
   <ServiceVersion>[String]</ServiceVersion>
   <ProcessName>[String]</ProcessName>
   <PipName>[String]</PipName>
   <AuditedOn>[YYYY-MM-DD HH:MM:SS.M]</AuditedOn>
   <AuditedBy>[oc4jadmin]</AuditedBy>
   <!-- Namespace details from the request/response message. There can be more than one occurrence
    of the section below -->
   <NsXpathsForDefinitionsRO>
      <DefinitionNsXpathsViewRORow>
         <DefinitionId>[Definition ID mentioned above]</DefinitionId>
         <NamespaceAlias>[String. namespace alias]</NamespaceAlias>
         <Namespace>[valid namespace URL]</Namespace>
      </DefinitionNsXpathsViewRORow>
   </NsXpathsForDefinitionsRO>
   <!-- XPATH variables and values. There can be more than one occurrence of the section below
    -->
   <XpathsForDefinitionsRO>
      <DefinitionXpathsViewRORow>
         <DefinitionId>[Definition ID mentioned above]</DefinitionId>
         <XpathSeqId>[Non negative Integer]</XpathSeqId>
         <Xpath>[XPATH expression]</Xpath>
         <IsNodeText>[0|1.Applicable only for Simulator Definitions]</IsNodeText>
         <IsNodeKey>[0|1. Applicable only for Simulator Definitions]</IsNodeKey>
         <Condition>[OK|EQ|NE|LT|GE|LE|Not Null]</Condition>
         <IsSystemGenerated>[0|1]</IsSystemGenerated>
      </DefinitionXpathsViewRORow>
   </XpathsForDefinitionsRO>
</DefinitionsViewRORow>

<!-- The section below is for one group test definition. This includes all definition details as
well as references to Test definitions that are mentioned above.
For each such group definition the section below will be repeated -->

<GroupDefinitions>
   <!-- There can be more than one occurrences of the section below -->
<GroupDefinitionsViewRORow>
   <GroupDefinitionId>[Group Definition ID that was set in the previous environment. During import
    the target system will generate a new ID for this field]
</GroupDefinitionId>
   <Description>[String]</Description>
   <ProcessName>[String]</ProcessName>
   <PipName>[String]</PipName>
   <GDDefinitionsViewRO>
      <!-- There can be more than one occurrences of the section below -->
      <GDDefinitionsViewRORow>
         <GroupDefinitionId>[Group Definition ID set above]</GroupDefinitionId>
            <SeqId>[Non negative Integer]</SeqId>
            <DefinitionId>[One of the Definition ID set in the DefinitionsViewRORow
             section]</DefinitionId>
            <DefinitionSeqId>[Non negative Integer]</DefinitionSeqId>
            <ServiceType>[Synchronous|Notify|Asynchronous two way]</ServiceType>
            <SoapTransportType>[HTTP]</SoapTransportType>
         </GDDefinitionsViewRORow>
      </GDDefinitionsViewRO>
```

```
   </GroupDefinitionsViewRORow>
</GroupDefinitions>
</DefinitionsList>
```

A.2 Instance.xml

The structure of the `Instance.xml` file created by the CAVS instance export feature is shown in Example A–2.

This export feature can be used to export a test or group instance in XML format that can be used with XML reporting tools to generate reports of test executions.

For more information about the instance export feature, see Chapter 12, "Exporting and Importing CAVS Definitions and Instances."

Example A–2 Instance.xml

```
<InstancesList><?xml version = '1.0' encoding = 'UTF-8'?>
<InstancesViewRORow>
<!-- There would be more occurrences of this if more instances are exported--!>
   <InstanceId>[Instance ID that was assigned by the environment in which the instance was
    run]</InstanceId>
   <Type>[Test|Simulator|Group</Type>
   <Status>[Status of the instances being exported] </Status>
   <StartedOn>[Date and time at which the instance started]</StartedOn>
   <EndedOn>[Date and time at which the instance ended]</EndedOn>
   <IsStaled>[0|1]</IsStaled>
   <DefinitionId>[Definition ID of the definition that generated the instance]</DefinitionId>
   <Description>[Description of the definition ID that generated the instance]</Description>
   <ServiceType>Synchronous|Asynchronous two-way|Asynchronous (notify)</ServiceType>
   <SoapAction>[String. Valid SOAP action for the WSDL defined for the definition
    ID]</SoapAction>
   <SoapTransportType>HTTP</SoapTransportType>
   <MessageRequest>actual request message</MessageRequest>
   <MessageResponse>actual response message</MessageResponse>
   <DefinitionsViewRO>
      <DefinitionsViewRORow>
         <DefinitionId>[Definition ID mentioned above]</DefinitionId>
         <Type>[Type mentioned above] </Type>
         <Description>[Description mentioned above]</Description>
         <State>[Locked|Unlocked]</State>
        <ServiceType>[Service Type mentioned above] </ServiceType>
        <SoapAction>[SOAP Action mentioned above] </SoapAction>
        <SoapTransportType>HTTP</SoapTransportType>
        <MessageRequest>[Request message defined in the corresponding Test or Simulator
         definition]</MessageRequest>
        <MessageResponse>[Response message defined in the corresponding Test or Simulator
         definition]</MessageResponse>
        <AuditedOn>[YYYY-MM-DD HH:MM:SS.M]</AuditedOn>
        <AuditedBy>[oc4jadmin]</AuditedBy>
      </DefinitionsViewRORow>
   </DefinitionsViewRO>
   <InstanceXpathsViewRO>
      <InstanceXpathsViewRORow>
         <InstanceId>[Instance ID assigned to the instance}</InstanceId>
         <XpathSeqId>[Non-negative integer] </XpathSeqId>
         <Status>[Status of the instance] </Status>
         <Xpath>/soap:Envelope</Xpath>
         <IsNodeKey>[0|1. Applicable only for Simulator Definitions]</IsNodeKey>
```

```
         <Condition>[OK|EQ|NE|LT|GE|LE|Not Null]</Condition>
      </InstanceXpathsViewRORow>
   </InstanceXpathsViewRO>
   <InstanceNsXpathsViewRO>
      <InstanceNsXpathsViewRORow>
         <InstanceId>[Instance ID assigned to the instance] </InstanceId>
         <NamespaceAlias>[String]</NamespaceAlias>
         <Namespace>[Valid namespace URL]</Namespace>
      </InstanceNsXpathsViewRORow>
   </InstanceNsXpathsViewRO>
</InstancesViewRORow></InstancesList>
```

B

Code Compliance Inspector: New Terminology & Available Assertion Executors

This appendix describes the new terminology, delivered assertions, and the available assertion executors for the Code Compliance Inspector tool.

There are two executors for CCI:

- **XPathExecutor**: contains all of the XPath related operations. For example, "XpathExistsCheck" and "XpathNodeCountEqualCheck"
- **FSExecutor**: contains all of the file system related operations. For example, "FileExistCheck" and "FilesMatchPatternCheck"

This appendix includes the following sections:

B.1 New Terminology

The Code Compliance Inspector was previously called the Process Integration Pack (PIP) Auditor. The new labels and concepts for Code Compliance Inspector are described in Table B–1.

Table B–1 New terminology

Old Name	New Name	Description
PIP Auditor	Code Compliance Inspector	Run the Code Compliance Inspector to check for good coding practices.
Rule	Assertion	Assertion replaces the <Rule> tag within the old Rules.xml. Assertions can be defined once within the AssertionCatalog.xml files and then used within one or more Policies in the Policies.xml files.
RuleSet	AssertionSet	AssertionSet replaces the <Ruleset> tag within the old Rules.xml. AssertionSet is informational only and is not surfaced in the JDeveloper Extension or in the Code Compliance By Policy Report.

Table B–1 (Cont.) New terminology

Old Name	New Name	Description
Rules.xml	AssertionCatalog.xml	There are two files: AssertionCatalog-AIA-<version>.xml and AssertionCatalog-WS-I-<version>.xml.
Category	Category	Category is part of an Assertion's definition in AssertionCatalog.xml. Category is a tag within the definition of an Assertion, that is largely just informational. For this release, existing categories will become the Policy names; meaning that all Assertions tagged with a particular category will appear in the Policies.xml files using a Policy name that matches the Category.
TestSuite	Policy	Replaces the <TestSuite name= > tag in the old TestSuite.xml with <Policy name= >; Policies can be reused into more coarse-grained policy buckets using the <depends name=> tag.
Test	Assertion	Replaces the <Test rulename= > tag in the old TestSuite.xml.
TestSuite.xml	Polices.xml	There are two files: Policies-AIA-<version>.xml and Policies-WS-I-<version>.xml. Within the Policies.xml provided by Oracle, the policy name should match the Catalog names used within the Assertions in the AssertionCatalog.xml. Customers can modify the Policies.xml including renaming and reorganizing the Policies & Assertions.
pipaudit	CheckCompliance	Command line utility.
Priority	Priority	This term remains the same. Priorities are defined as: ■ Priority 1 assertions are the basic assertions that an integration project has to satisfy 100% to be qualified as a Compliant. ■ Priority 2 assertions are more stringent on certain design time patterns and an integration project that meets these assertions is qualified as Conformant. ■ Priority 3 assertions are the most stringent at the lowest levels of the technology, and an integration project that meets at least a certain threshold of these assertions is qualified as Fully Conformant. ■ Priority 4 assertions are recently introduced assertions that can be qualified as P3 or P2 or P1 assertions. For this release, these assertions do not play a role in the qualification of an integration project. Priority levels are additive in nature, so passing Priority 2 (conformant) means that you need to pass Priority 1 (compliance) as well. A detailed list of pre-defined assertions can be viewed in the packaged Assertion Catalog XML files available under the ComplianceInspector/config directory.

B.2 Delivered Assertions

The following tables show the delivered assertions. These pre-defined assertions can be viewed in the packaged Assertion Catalog XML files located in the ComplianceInspector/config directory.

Table B–2 Category :Coding Standards

Assertion	Priority	Description
NoTargetSysIdHardWiringInDVMLookupCheck	1	The Target SystemId must not be hardwired in DVM lookups when used in XSL. The syntax used should be: orcl:lookupValue('DVM_NAME',$DVMSourceCol,XPATH,$DVMTargetCol,'') where DVM_NAME can contain alphanumeric and underscore characters and XPATH can contain any XPATH expression. Note: the targetId column can be one of the following: 1) Any Xpath. 2) 'COMMON' 3) a variable that does not contain a hard coded string value. 4) Xpath function.
NoTargetSysIdHardWiringInXREFLookupCheck	1	The Target SystemId must not be hardwired in XREF lookups when used in XSL. The syntax used should be: xref:lookupXRef('XREF_NAME',$XREFSourceCol,XPATH,$XREFTargetCol,true() | false()) where XREF_NAME can contain alphanumeric and underscore characters and XPATH can contain any XPATH expression. Note: the targetId column can be one of the following: 1) Any Xpath. 2) 'COMMON' 3) a variable that does not contain a hard coded string value. 4) Xpath function.
TXNEnableInASyncDelayed	2	This check is for an Async Delayed Response service. It must participate in a global transaction.
TXNEnableInFireForget	2	This check is for a Fire and Forget service. It must participate in a global transaction.
TonkenizedReferencesInXSLCheck	2	All http URLs that point to an http server location must be tokenized with ${hostname} and ${port}.
NameSpacePrefixesNamingInBPELCheck	3	In a BPEL process, namespace prefixes must follow naming standard guidelines. The ns1, ns2 prefixes which are generated by default are not allowed.

Table B–3 Category :Error Handling Standards

Assertion	Priority	Description
CatchBlockBindingFaultExistInBPELCheck	2	Catch block has to be defined for BindingFault in BPEL process.
CatchBlockRemoteFaultExistInBPELCheck	2	Catch block has to be defined for RemoteFault in BPEL process.

Table B–4 Category :Loose Coupling Standards

Assertion	Priority	Description
TonkenizedReferencesInBpelCheck	2	All http URLs that point to an http server location must be tokenized with ${hostname} and ${port}.
TonkenizedReferencesInCompositeCheck	2	All http URLs that point to an http server location must be tokenized with ${hostname} and ${port}.
TonkenizedReferencesInWSDLCheck	2	All http URLs that point to an http server location must be tokenized with ${hostname} and ${port}.

Table B–5 Category :Naming Standards

Assertion	Priority	Description
BPELAssignActivityNamingCheck	3	Change the default JDeveloper generated name for the 'assign' activity.
BPELInvokeActivityNamingCheck	3	Change the default JDeveloper generated name for the 'invoke' activity.
BPELReceiveActivityCheck	3	Change the default JDeveloper generated name for the 'receive' activity.
BPELReplyActivityNamingCheck	3	Change the default JDeveloper generated name for the 'reply' activity.
BPELScopeActivityCheck	3	Change the default JDeveloper generated name for the 'scope' activity.

Table B–6 Category :Performance Standards

Assertion	Priority	Description
CompletionPersistPolicyCheck	1	The bpel.config.completionPersistPolicy property configures how the instance data is saved. For synch transient processes, the value for this property should be 'faulted.' Only the faulted instances will be saved.
SynchAuditLogLevelCheck	1	The bpel.config.auditLavel property configures how the BPEL service engine will capture audit details. For Synch Transient processes, the value for this property should be 'faulted.'

Table B–7 Category :Reusability Standards

Assertion	Priority	Description
NoLocalSchemasInBPELCheck	2	The BPEL Process folder must not contain any schema files. All Utility schemas must be accessed from a web server.
NoLocalAdaptersInBPELCheck	3	Adapters should be defined as ESB services. This helps in Endpoint Virtualization. Also, BPEL processes gain homogeneity, focusing on business problems rather than protocol transformations.
NoSchemaElementsDefinedInWSDLCheck	3	WSDLs should use schema imports. All schema elements must be defined in XSD.

Table B–8 Category :Security Standards

Assertion	Priority	Description
NoPlinkusageForSettingWSSecPropCheck	1	BPEL processes must not use property for passing the username and password for ws-security. OWSM should be used for all web services invocation authentication purposes.
NoUnamePwdInDVMCheck	1	DVM stores must not store credentials. They should not contain UserName and Password values. OWSM should be used for all web services invocation authentication purposes.

Table B–9 Category :WS-I BP Standards

Assertion	Priority	Description
SchemaImportUsedforXSDOnlyCheck	3	WSDL import elements must not be used to import other kinds of XML schemas. WSDL imports must only import WSDLs. This check is to ensure compatibility with the WS interoperability basic profile 1.0.
SchemaImportsOnlyInsideSchemaCheck	3	XML Schema 'import' statements must be within the xsd:schema element of the types element.
SchemaNodeOnlyInsideWsdlTypesCheck	3	XML Schema elements must be within the xsd:types element of the types element.
SchemaTargetNamespaceExistCheck	3	All xsd:schema elements contained in a wsdl:types element must have a targetNamespace attribute with a valid and non-null value, unless the xsd:schema element has xsd:import and/or xsd:annotation as its only child element(s).
SchemaTargetNamespaceMatchingCheck	3	WSDLs must not import WSDLs that have a different targetNamespace in the definition. This assertion assumes that inputDir contains the AIAMetaData directory.
SchemaXSDFileRootSchemaCheck	3	XSD files must import XSD files that have schema as a root node in the location of xsd:import. This assertion assumes that inputDir contains the AIAMetaData directory .

Table B–9 (Cont.) Category :WS-I BP Standards

Assertion	Priority	Description
UTFEncodingUsedinSchemaCheck	3	Schema definitions must use UTF-8 or UTF-16 encoding. UTF encoding can be specified in the processing instruction of an XML. The assertion looks for the existence of UTF in the processing instructions. This check is to ensure compatibility with the WS interoperability basic profile 1.0.
UTFEncodingUsedinWSDLCheck	3	WSDL description must use UTF-8 or UTF-16 encoding. UTF encoding can be specified in the processing instruction of an xml. The assertion looks for the existence of UTF in the processing instruction. This check is to ensure compatibility with the WS interoperability basic profile 1.0.
WSDLDocumentationIsFirstChildCheck	3	The wsdl:documentation element may be present as the first child element of wsdl:import, wsdl:part and wsdl:definitions in addition to the elements cited in the WSDL1.1 specification.
WSDLFileRootDefinitionsCheck	3	WSDLs must import WSDL files that have definitions as a root node in the location of wsdl:import. This assertion assumes that inputDir contains the AIAMetaData directory.
WSDLImportLocationNotEmptyCheck	3	The location attribute of all wsdl:import elements must be non-empty.
WSDLImportNoRelativeURIInNSCheck	3	The namespace attribute of wsdl:import must not be a relative URI. The URI should be an absolute URI as per URI standards. This check is to ensure compatibility with the WS interoperability basic profile 1.0.
WSDLImportOnlyPrecededByDocCheck	3	All WSDL import elements must only be preceded by WSDL documentation element in a WSDL file. This check is to ensure compatibility with the WS interoperability basic profile 1.0.
WSDLImportUsedforWSDLOnlyCheck	3	WSDL import element must not be used to import other kinds of XML schemas. WSDL import must only import WSDLs. This check is to ensure compatibility with WS interoperability basic profile 1.0.
WSDLImportsOnlyInsideDefinitionCheck	3	All WSDL 'import' statements must be within wsdl:definition elements.
WSDLOperationMustHaveInputCheck	3	Solicit-Response and Notification type operations must not be used in a wsdl:portType definition. For example, output messages should always be after input messages.
WSDLOperationNameMustBeUniqueCheck	3	All wsdl:portType elements must have operations with distinct values for their name attributes(overloading).
WSDLPartMustNotUseElementAndTypeCheck	3	A wsdl:message element must not specify both 'type' and 'element' attributes on the same wsdl:part element.
WSDLTargetNamespaceMatchingCheck	3	WSDLs must not import other WSDLs that have different targetNamespace in definitions. This assertion assumes that inputDir contains the AIAMetaData directory.

Table B–9 (Cont.) Category :WS-I BP Standards

Assertion	Priority	Description
WSDLTypesOnlyPrecededByDocAndImportCheck	3	All WSDL types elements must only be preceded by WSDL documentation element or wsdl import in a WSDL file. This check is to ensure compatibility with the WS interoperability basic profile 1.0.
XMLversionUsageInSchemaCheck	3	XSD files must use XML version 1.0. The XML version can be specified in the processing instructions of an XML. The assertion looks for the existence of version in the processing instructions. This check is to ensure compatibility with WS interoperability basic profile 1.0.
XMLversionUsageInWSDLCheck	3	WSDL files must use XML version 1.0. The XML version can be specified in the processing instructions of an XML. The assertion looks for the existence of version in the processing instructions. This check is to ensure compatibility with WS interoperability basic profile 1.0.

B.3 Assertion Parameters for the XPathExecutor

The following tables describe the mandatory and optional parameters for the XPathExecutor.

B.3.1 Mandatory Parameters List

Table B–10 describes the mandatory parameters.

Table B–10 Mandatory Parameters for XPathExecutor

Param Name	Description	Default Value	Example	Failure Situation
Xpath.search	XPath to be executed on a particular document	No default	//xsl:variable/@name	XPath execution failure exception
xpath.namespace.prefixes	The list of delimited values of namespace prefixes that are used for a particular XPath expression execution.	No default	'bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"; xsl="http://www.w3.org/1999/XSL/Transform"; aiacfg="http://xmlns.oracle.com/aia/core/config/V1"; wsdl="http://schemas.xmlsoap.org/wsdl/"; xsd="http://www.w3.org/2001/XMLSchema"; xsd="http://www.w3.org/2001/XMLSchema";	XPath execution failure exception

Table B–10 (Cont.) Mandatory Parameters for XPathExecutor

Param Name	Description	Default Value	Example	Failure Situation
assertCondition	If the Assertion level attribute "operation" is not present then this param value will be the valid executor operation name. Otherwise, this can be used as a sub-operation value.	No default	XpathValuesPatternMatchCheck	Unsupported operation exception
xpath.match.regxpattern	Assertion value	No default	[a-zA-Z_0-9]* The regular expression says that a variable name can contain only alphanumeric characters with underscores.	Error depending on the comparison type

B.3.2 Optional Parameters List

Table B–11 describes the optional parameters.

Table B–11 Optional Parameters for XPathExecutor

Param Name	Description	Default Value	Example	Failure Situation
xpath.notexist.ignore	Every XPath operation assumes that the comparison between the XPath execution result and the assertValue can only be made if the output NodeList returned after evaluating the Xpath contains at least one node (the default behavior except for the 'XpathExists' and 'XpathNotExists' operations). The default behavior of non-compliance is reported if the XPath does not return any nodes.)	False	We have a test saying "all compensate activities in BPEL should start with a prefix of compensate". Now if we do not have any compensate activities in a BPEL file, Code Compliance Inspector reports a non-compliance. If we specify xpath.notexist.ignore="true", then the test would be considered a success by Code Compliance Inspector.	--
xml.external.filename	There are situations when the assertValue is more than just a mere String. The value can be an entire XML fragment, which contains regEx patterns. In this case, a compare file argument specifies the file and the XPath specifies the fragment within the file.	No default	Used to compare standard code snippets. <Param name="xml.external.filename" value="${faultXML}" default="AIAStdCode.xml"/>	--

Table B–11 (Cont.) Optional Parameters for XPathExecutor

Param Name	Description	Default Value	Example	Failure Situation
xml.external.search.xpath	Xml.external.search.xpath is always used with xml.external.filename. As mentioned above, this XPath helps us identify the XML fragment for comparison.	--	<Param name="xml.external.search.xpath" value="//EBMHeaderPopulation/corecom:EBMHeader/P4/corecom:MessageProcessingInstruction"/> This is used with the xml.external.filename param above. When we apply the XPath (Xml.external.search.xpath) on the file (Xml.external.filename) we get an XML fragment for comparison. In the above example, the MessageProcessingInstruction returned by evaluating XPath on selected files is evaluated against MessageProcessingInstruction returned by evaluating Xml.external.search.xpath on the file Xml.external.filename.	--
xml.node.match.mode	Special operations, if any, to be executed to derive the assertValue. Otherwise, the default behavior is executed which is stated in the default section	Default: String It converts the NodeList from Resultant XPath to string (ConvertNodeListToStirng)	Currently, the only supported type is length. By default, when nothing is specified, it converts the resultant NodeList to String.	--

B.4 Assertion Parameters for the FSExecutor

The following tables describe the mandatory and optional parameters for the FSExecutor.

B.4.1 Mandatory Parameters List

Table B–12 describes the mandatory parameters.

Table B–12 Mandatory Parameters for the FSExecutor

Param Name	Description	Default Value	Example	Failure Situation
assertCondition	Any one of the operations supported by the FSExecutor. See the Available Operations for FSExecutor section.	No default	FileNotExistCheck	Unsupported Operation exception
filename.search.regxpattern	Regular expression for selecting matching files.	No default	a-zA-Z_0-9_ /]*(EBF)((V)[0-9]*)??.wsdl	Invalid Regular expression

B.4.2 Optional Parameters List

Table B–13 describes the optional parameters.

Table B–13 Optional Parameters for the FSExecutor

Param Name	Description	Default Value	Example	Failure Situation
filename.match.regxpattern	Assertion value (String)	No default	InputFilePattern_(c|C)ustom.xsl	--
filecontent.exclude.regxpattern	Exclude of file for which content matched with this pattern (Regular Expression)	No default	.*(c|C)ustom.xsl	--
filename.exclude.regxpattern	Exclude of file for which filename matched with this pattern (Regular Expression)	No default	.*(JMSProducer|OutboundHeader).*.wsdl	--

B.5 Available Operations for the XPathExecutor

The following tables describe the available operations for the XPathExecutor.

XpathListExistCheck

Checks for the existence of the nodes in the given XPath List. Every XPath in the list should have at least one node. This is very similar to the XpathExist operation except that we can check for multiple XPaths.

Table B–14 XpathListExistCheck

Operation	Description	Comments
Xpath.search	XPath where zero nodes are expected	-
Xpath.namespace.prefixes	-	-
Xpath.1.search	-	/A/B
Xpath.2.search	-	And so on... We can check for 'n' XPaths in this manner (xpathn)
xml.local.imports.resovable	-	-
xml.remote.imports.resovable	-	-
local.metadir	-	-

XpathNotExistsCheck

Checks for the existence of the nodes in the given XPath List. Every XPath in the list should have at least one node. This is very similar to the XpathExist operation except that we can check for multiple XPaths.

Table B–15 XpathNotExistsCheck

Operation	Description	Comments
Xpath.search	XPath where zero nodes are expected	-
Xpath.namespace.prefixes	-	-
xml.local.imports.resovable	-	Same as the one in XpathExists.
xml.remote.imports.resovable	-	-
local.metadir	-	-

XpathNodeCountLessThanCheck

Checks if the number of nodes found at the XPath is less than the assert value.

Table B–16 XpathNodeCountLessThanCheck

Operation	Description	Comments
Xpath.search	XPath for which node count is checked.	-
Xpath.namespace.prefixes	-	-
xpath.match.regxpattern	Maximum number of nodes that can be present for the XPath. Note that if your intended value is 'n' then the assert value is its 'n+1'.	All BPEL processes, which follow SYNC Request Response pattern, should not have more than 6 extension points. So assert value would be '7'.

XpathNodeCountGreaterThanCheck

Checks if the number of nodes found at the XPath is less than the assert value.

Table B–17 XpathNodeCountGreaterThanCheck

Operation	Description	Comments
Xpath.search	XPath for which node count is checked.	-
Xpath.namespace.prefixes	-	-
xpath.match.regxpattern	Minimum number of nodes that can be present for the XPath. Note that if your intended value is 'n' then the assert value is its 'n-1'.	All BPEL processes, which follow SYNC Request Response pattern, should have minimum of 4 extension points. So assert value would be '3'.

XpathValuesLessThanCheck

Checks if the value in the XPath is less than the assert value.

Table B–18 XpathValuesLessThanCheck

Operation	Description	Comments
Xpath.search -	XPath for which node count is checked.	-
Xpath.namespace.prefixes	-	-
xpath.match.regxpattern	Maximum number that the value from the XPath can have. Note that if your intended value is 'n' then the assert value is its 'n+1'.	All BPEL processes, which follow SYNC Request Response pattern, should not have more than 6 extension points. So assert value would be '7'.

XpathValuesLessThanEqualCheck

Checks if the value in the XPath is less than or equal to the assert value.

Table B–19 XpathValuesLessThanEqualCheck -

Operation	Description	Comments
Xpath.search	XPath for which node count is checked.	-
Xpath.namespace.prefixes	-	-
xpath.match.regxpattern	Maximum number that the value from XPath can have.	-

XpathValuesGreaterThanCheck

Checks if the value in the XPath is greater than the assert value.

Table B–20 XpathValuesGreaterThanCheck

Operation	Description	Comments
Xpath.search	XPath for which node count is checked.	-
Xpath.namespace.prefixes	-	-
xpath.match.regxpattern	Minimum number of nodes that can be present for the XPath. Note that if your intended value is 'n' then the assert value is its 'n-1'.	All BPEL processes, which follow SYNC Request Response pattern, should have minimum of 4 extension points. So assert value would be '3'.

XpathValuesGreaterThanEqualCheck

Checks if the value in the XPath is greater than or equal to the assert value.

Table B–21 XpathValuesGreaterThanEqualCheck

Operation	Description	Comments
Xpath.search	XPath for which node count is checked.	-
Xpath.namespace.prefixes	-	-
xpath.match.regxpattern	Minimum number of nodes that can be present for the XPath.	-

CompareNodeWithRegExXMLCheck

Checks if the node returned by the XPath matches the XML snippet from a file. Note that it is a regular expression comparison and the snippet can contain regular expressions.

Table B–22 CompareNodeWithRegExXMLCheck

Operation	Description	Comments
Xpath.search	XPath for the node, which has to be checked.	-
xpath.namespace.prefixes	-	-
xml.external.filename	The XML file where the snippet for comparison lies.	"ABCS WSDL should be documented as per AIA Documentation standards." The file AIAStdCode.xml for example contains all the XML snippets. So this file is the xml.external.filename
xml.external.search.xpath	//ABCSwsdlDoc/wsdl:documentation	XPath to derive the XML snippet from the xml.external.filename XML file. This XPath will separate out just the documentation snippet from the XML.

CompareNodeListWithRegExXMLCheck

Checks if every node from the NodeList returned by the XPath matches the XML snippet from a file. Note that it is a regular expression comparison and the snippet can contain regular expressions. This can be used when multiple nodes from a file have to be checked against the same XML snippet.

Table B–23 CompareNodeListWithRegExXMLCheck

Operation	Description	Comments
Xpath.search	XPath for the NodeList, which has to be checked.	Every node from NodeList returned from this XPath should be compliant to the XML snippet derived using xml.external.filename and xml.external.search.xpath.
xpath.namespace.prefixes	-	-
xml.external.filename	The XML file where the snippet for comparison lies.	"Catch blocks are defined as per AIA Error Handling Guidelines." The file AIAStdCode.xml for example contains all the XML snippets. So this file is the xml.external.filename
xml.external.search.xpath	XPath to derive the XML snippet from the xml.external.filename XML file.	//catch This XPath will separate out just the documentation snippet from the XML.

XpathValuesEqualCheck

Checks if the string value of every node from the NodeList returned by the XPath matches the string value specified in the assert value.

Table B–24 XpathValuesEqualCheck

Operation	Description	Comments
Xpath.search	XPath for the NodeList, which has to be checked.	Every node from NodeList returned from this XPath should be compliant to the assert value.
xpath.namespace.prefixes	-	-
xpath.match.regxpattern	String value to check against.	-
xpath.notexist.ignore	-	-

XpathValuesNotEqualCheck

Checks if the string value of every node from the NodeList returned by the XPath does not match the string value specified in the assert value.

Table B–25 XpathValuesNotEqualCheck

Operation	Description	Comments
Xpath.search	XPath for the NodeList, which has to be checked.	Every node from NodeList returned from this XPath should be compliant to the assert value.

Table B–25 (Cont.) XpathValuesNotEqualCheck

Operation	Description	Comments
xpath.namespace.prefixes	-	-
xpath.match.regxpattern	String value to check against.	-
xpath.notexist.ignore	-	-

XpathValuesPatternMatchCheck

Checks if the string value of every node from the NodeList returned by the XPath matches the regular expression pattern specified in the assert value.

Table B–26 XpathValuesPatternMatchCheck

Operation	Description	Comments
Xpath.search	XPath for the NodeList, which has to be checked.	Every node from NodeList returned from this XPath should be compliant to the assert value.
xpath.namespace.prefixes	-	-
xpath.match.regxpattern	Regular expression pattern to check against.	"All Assign activities in a BPEL process should start with a prefix of Assign followed by activity name". The pattern would look like: (Assign){1}(_)??(([a-zA-Z])([a-zA-Z_0-9])*)
xpath.notexist.ignore	-	-

XpathValuesNotMatchPatternCheck

Checks if the string value of every node from the NodeList returned by the XPath does not match the regular expression pattern specified in the assert value. This does the exact opposite check of XpathValuesPatternMatch.

Table B–27 XpathValuesNotMatchPatternCheck

Operation	Description	Comments
Xpath.search	XPath for the NodeList, which has to be checked.	Every node from NodeList returned from this XPath should be compliant to the assert value.
xpath.namespace.prefixes	-	-
xpath.match.regxpattern	Regular expression pattern to check against.	"Target node should not be populated during ABM to EBM transformation in Requester ABCSImpl." The following pattern would ensure that no hard coding of target ID is present. :([a-zA-Z_0-9\s]*)
xpath.notexist.ignore	-	-

XpathValueNotContainsCheck

Checks if the string value of every node from the NodeList returned by the XPath does not contain the string specified in the assert value.

Table B–28 XpathValueNotContainsCheck

Operation	Description	Comments
Xpath.search	XPath for the NodeList, which has to be checked.	Every node from NodeList returned from this XPath should be compliant to the assert value.
xpath.namespace.prefixes	-	-
xpath.match.regxpattern	String value to check against.	"DVM stores should have no credentials stored.." The following pattern would ensure that no tokens that are generally used to store credentials are used in DVMs.:UserName;Password;uname; pwd;username;password
xpath.notexist.ignore	-	-

XpathValueContainsCheck

Checks if the string value of every node from the NodeList returned by the XPath contains the string specified in the assert value.

Table B–29 XpathValueContainsCheck

Operation	Description	Comments
Xpath.search	XPath for the NodeList, which has to be checked.	Every node from NodeList returned from this XPath should be compliant to the assert value.
xpath.namespace.prefixes	-	-
xpath.match.regxpattern	String value to check against.	-
xpath.notexist.ignore	-	-

ExistsRegExXMLCheck

Iterates through the children of the node specified by xml.external.filename and xml.external.search.xpath. Checks if every node in NodeList returned by executing xml.external.search.xpath on xml.external.filename, exists in the NodeList returned by executing xpath on the policies file. Note that CompareNodeWithRegExXML checks against the xml.external.filename. The behavior is reverse in this operation. This operation iterates through all the children of the node from xml.external.filename and makes sure each one of them is present in the NodeList from XPath.

Table B–30 ExistsRegExXMLCheck

Operation	Description	Comments
Xpath.search	XPath for the NodeList, which has to be checked.	Every node from NodeList returned from this XPath should contain all the children of the node derived from xml.external.filename and xml.external.search.xpath.
xpath.namespace.prefixes	-	-
xml.external.filename	String value to check against.	"Ensure MessageProcessingInstruction is populated fully in ReqABM_to_EBM xsl".
xml.external.search.xpath	-	-
xml.node.match.mode	-	1: (NODE_MUST) This is the default option. All the elements are considered for comparison. Any missing elements are reported for non-compliance. 2: (NODE_IGNORE). If xml.node.match.mode is specified as 2, then missing nodes are not considered for comparison. For example, consider an XML structure with A as a parent and B and C as children (<A><B/><C/></A>). When xml.node.match.mode=1 and if node B or C is absent all together, then non-compliance is reported. If we want to change this default behavior to report a compliance, then we should specify xml.node.match.mode=2. Note that if a node is present then it should conform to the regular expression specified. 3: (NODE_OPTIONAL). This lets us pick and choose what differences we would want to ignore. We can add an attribute minoccurs="0" in any element that we would want to skip comparison when not found. For example, consider an XML structure where A is a parent element and has 2 children B and C (<A><B/><C/></A>). If we want a scenario where missing B's should be reported as compliance where as missing C's should be reported as non-compliance then this is how we can achieve it through xml.node.match.mode: <A><B minoccurs="0"/><C/></A>
xpath.notexist.ignore	-	-

NotExistsRegExXMLCheck

Iterates through children of node specified by xml.external.filename and xml.external.search.xpath. Checks if every node in NodeList returned by executing xml.external.search.xpath on xml.external.filename, does not exist in the NodeList returned by executing the XPath on the policies file. Note that this does the exact reverse of ExistsRegExXML.

Table B–31 NotExistsRegExXMLCheck

Operation	Description	Comments
Xpath.search	XPath for the NodeList, which has to be checked.	Every node from NodeList returned from this XPath should contain all the children of the node derived from xml.external.filename and xml.external.search.xpath.
xpath.namespace.prefixes	-	-

Table B–31 (Cont.) NotExistsRegExXMLCheck

Operation	Description	Comments
xml.external.filename	String value to check against.	-
xml.external.search.xpath	-	Example of this would be say we want to make sure double notifications are not sent as part of error handling. So we could check for the non-existence of certain error handling code snippets in some of the catch blocks.
xml.node.match.mode	-	See the table description for ExistsRegExXML for more information.
xpath.error.path	-	This would be helpful if we would want to show the user the node, which is not supposed to exist. For example, if we want to show the user the catch block that contains the redundant call, this is how we can do it: <Param name="xpath.error.path" default="@faultName"/>
xpath.notexist.ignore	-	-

B.6 Available Operations for the FSExecutor

The following tables describe the available operations for the FSExecutor

FileExistCheck

Checks if a file of particular pattern exists in the selected directory.

Table B–32 FileExistCheck

Operation	Description	Comments
filename.search.regxpattern	Pattern of the file to be selected.	If you want to check for the existence of a config file, for example, AIAConfigurations.xml in every ABCS integration project, then you can select FileType="*" and context="ABCS" and then provide "AIAConfigurations.xml" in this param value.

FileNotExistCheck

Checks if a file of particular pattern does not exist in the selected directory.

Table B–33 FileNotExistCheck

Operation	Description	Comments
filename.search.regxpattern	Pattern of the file to be selected.	If you want to check for the non-existence of a local schema in every ABCS integration project, then you can select FileType="*" and context="ABCS" and then provide "*.xsd" in this param value.

FilesMatchPatternCheck

Checks if the selected file name matches a particular pattern.

Table B–34 FilesMatchPatternCheck

Operation	Description	Comments
filename.search.regxpattern	Pattern of the file to be selected.	If you want to check for the existence of an extension WSDL in every ABCS integration project, then you can select FileType="*" and context="ABCS" and then provide ".*(ABCSImpl)((V)[0-9]*)??.wsdl" in this param value.
filename.match.regxpattern	The pattern the file name should be checked against.	If you want to assert that the extension file selected matches a particular naming pattern, for example: ".*(ABCSImpl)Extension.wsdl"

Index

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

W