Oracle® Fusion Middleware

Developer's Guide for Oracle Application Integration Architecture Foundation Pack

11*g* Release 1 (11.1.1.6.3)

E17364-08

August 2012

Documentation for developers that describes how to use Oracle Application Integration Architecture (AIA) Foundation Pack to conceptualize AIA projects. Describes how to use Project Lifecycle Workbench, Service Constructor, and deployment plans to implement AIA solutions. Describes how to implement new services that extend pre-built integrations. Use the information in this guide to help ensure that the AIA solutions you develop can be upgraded and supported.

ORACLE®

Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack, 11*g* Release 1 (11.1.1.6.3)

E17364-08

Primary Author: Ramakanth Kotha

Contributing Author: Rosemarie Hall

Contents

3 Working with Project Lifecycle Workbench

4 Working with Service Constructor

5 Harvesting Oracle AIA Content

6 Working with Project Lifecycle Workbench Bills of Material

7 Working with Project Lifecycle Workbench Seed Data

8 Generating Deployment Plans and Deploying Artifacts

9 Generating a Deployment Plan for ODI

10 Developing and Deploying Custom XPath Functions

11 Configuring and Using Oracle Enterprise Repository as the Oracle AIA SOA Repository

12 Annotating Composites

13 Designing and Developing Enterprise Business Services

14 Designing Application Business Connector Services

15 Constructing the ABCS

16 Completing ABCS Development

19 Introduction to B2B Integration Using AIA

20 Developing and Implementing Outbound B2B Integration Flows

21 Developing and Implementing Inbound B2B Integration Flows

22 Describing the Event Aggregation Programming Model

23 Establishing Resource Connectivity

24 Using Oracle Data Integrator for Bulk Processing

25 Working with Message Transformations

27 Working with AIA Design Patterns

28 Working with Security

29 Best Practices for Designing and Building End-to-End Integration Flows

30 Tuning Integration Flows

31 Oracle AIA Naming Standards for AIA Development

A Delivered Oracle AIA XPath Functions

List of Examples

List of Figures

List of Tables

Preface

Welcome to the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack 11g Release 1 (11.1.1.6.3).*

Audience

This guide is intended for developers to conceptualize AIA projects and use the AIA Project Lifecycle Management, Service Constructor to implement AIA solutions and implement additional functionalities in the form of new services extending the AIA Pre-Built Integrations functionalities.

This guide ensures that the AIA solution that you build and implement can be upgraded, supported, and maintained.

Oracle AIA Guides

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

- *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack 11g Release 1 (11.1.1.6.3)*
- *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack 11g Release 1 (11.1.1.6.3)*
- *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack 11g Release 1 (11.1.1.6.3)*
- *Oracle Fusion Middleware Reference Process Models User's Guide for Oracle Application Integration Architecture Foundation Pack 11g Release 1 (11.1.1.6.3)*
- *Oracle Fusion Middleware Migration Guide for Oracle Application Integration Architecture 11g Release 1 (11.1.1.6.3)*
- *Oracle Fusion Middleware Product-to-Guide Index for Oracle Application Integration Architecture Foundation Pack 11g Release 1 (11.1.1.6.3)*

Related Documents

For more information, see the following documents in the Oracle SOA Suite and Oracle Business Process Management Suite 11g Release 1 (11.1.1.6.3) documentation set:

- *Oracle Fusion Middleware Installation Guide for Oracle SOA Suite and Oracle Business Process Management Suite 11g Release 1 (11.1.1.6.3)*

- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite 11g Release 1 (11.1.1.6.3)*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite 11g Release 1 (11.1.1.6.3)*
- *Oracle Fusion Middleware Security and Administrator's Guide for Web Services 11g Release 1 (11.1.1.6.3)*
- *Oracle Fusion Middleware User Guide for Oracle Enterprise Repository 11g Release 1 (11.1.1.6.3)*

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.

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 11.1.1.6.x

The following table lists the sections that have been added or changed for Release 11.1.1.6.x. The last two columns denote for which release of 11.1.1.6.x was the feature added.

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

Sections	Changes Made	11.1.1.6.0	11.1.1.6.1	11.1.1.6.3
Chapter 2, "Building AIA Integration Flows"	The term Task was described as a Business Task. This is now changed to Task. Sequence of the chapter is changed. Earlier it was chapter 20 now it is chapter 2.	X		
Chapter 10, "Developing and Deploying Custom XPath Functions"	This is a new chapter. This chapter discusses how to implement a function in Java as a Java XPath Class, deploy an XPath/XSL function in JDeveloper and deploy an XPath/XSL function in application servers.	X		
Chapter 22, "Describing the Event Aggregation Programming Model"	This is a new chapter. This chapter provides an overview of the Event Aggregation programming model and discusses how to implement it.	X		
Chapter 16 Completing ABCS Development	New section on "Resequencing in Oracle Mediator" was added.			
Section 16.9, "Resequencing in Oracle Mediator"	This section provides an overview of the Requencing feature in Oracle Mediator and discusses how to set up and use it in the Oracle AIA Asynchronous message exchange pattern.	X		
Chapter 16 Completing ABCS Development	New section on Layered Customizations is added.			
Section 16.10, "Developing Layered Customizations"	This section provides an overview of Layered Customizations and how to develop layered customizations for out-of-box pre-built integrations delivered by Oracle.		X	

1

Getting Started with the AIA Development Guide

This chapter provides an overview of types and styles of integrations addressed by Oracle Application Integration Architecture (AIA) and details how to use this guide.

This chapter includes the following sections:

- Section 1.1, "Types of Integrations Addressed by AIA"
- Section 1.2, "Integration Styles Addressed by AIA"
- Section 1.3, "How to Use the AIA Development Guide"

1.1 Types of Integrations Addressed by AIA

AIA addresses two types of integrations:

- **Functional integration**

  Functional integration weaves various functionalities of different participating applications, exposed as services, as processes to accomplish tasks that span multiple applications in any enterprise.

- **Data integration**

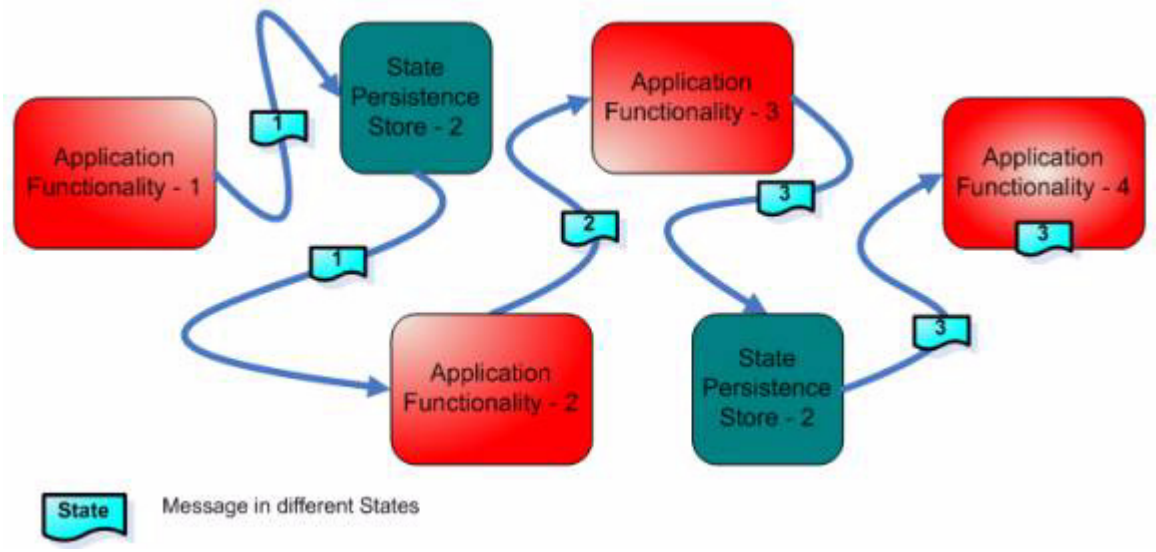
  Data integration connects applications at the data level and makes the same data available to multiple applications. This type of integration relies on database technologies and is ideal when a minimum amount of business logic is reused and large amounts of data transactions are involved across applications. This type of integration is suitable for batch data uploads or bulk data sync requirements.

1.2 Integration Styles Addressed by AIA

AIA provides reference architecture for a variety of situations. Depending on the size and complexity of integration projects, the integration style adopted for implementing *integration flows* varies. The number of participating applications and their role in *integration flows* contribute to the integration style adopted.

The *integration flow* as shown in Figure 1–1 represents the journey of a message from a business event triggering source to one or more target milestones, after passing through possible intermediary milestones. At each milestone, the message is stored in a different state.

Figure 1–1 Illustration of the Integration Flow



The integration flow represents the run-time path of a message. It is not a design time artifact.

AIA addresses the following integration styles:

- Integration through native application interfaces using the Oracle Applications Technology Infrastructure.
- Integration styles with integration framework
  - Direct integration through application web services using Oracle SOA Suite.
  - Integration through packaged canonical and standardized interfaces using Oracle Foundation Pack.
- Integration styles for bulk data processing
  - Real-time data integration flow
  - Batch data integration flow

1.3 How to Use the AIA Development Guide

The sales process provides detailed information about the value of AIA offerings. The value presented is perceived in the context of a business problem for which a solution is being sought.

The detailed analysis of the business problem and documenting of related business requirements leads to a Functional Design Document (FDD), which provides:

- A detailed description of the business case
- Various use cases detailing the various usage scenarios including the exception cases with expected actions by various actors
- Details about all the participating applications - commercial, off-the-shelf with versions and homegrown
- Details about the triggering business events
- Details about the functional flow

- Details about business objects to be used
- Actions to be performed on the various business objects
- Details about performance and scalability requirements

The AIA Development Guide assumes:

- A Functional Design Document is available.
- Access to AIA software.
- Access to all AIA-provided documents.
- You have read the AIA Concepts and Technologies Guide

The AIA Development Guide is logically divided as follows:

- Overview of all tasks for building the AIA integration flow
- Details of development of various AIA artifacts
- Activities around interaction between AIA artifacts and external artifacts
- Discussion of various design patterns, best practices, and tuning for run-time performance

Start with Chapter 2, "Building AIA Integration Flows" and move to relevant chapters in the Development Guide as needed.

2

Building AIA Integration Flows

This chapter introduces AIA integration flows and describes how to set up development and test environments. It discusses the role of the AIA Project Lifecycle Workbench and provides an overview of choosing of integration styles and AIA patterns. Finally, it provides a high-level overview of the development tasks for AIA artifacts and how to test integration flow.

This chapter includes the following sections:

2.1 How to Set Up Development and Test Environments

For AIA development and testing, the setup of development and test environments consists of the following activities:

- Set up Oracle JDeveloper for AIA.

  JDeveloper is the integrated development tool of choice.

  See Section 2.1.1, "How to Set Up JDeveloper for AIA Development."

- Set up Oracle Fusion Middleware for AIA.

  The Oracle Fusion Middleware environment is for deploying the AIA service and artifacts and for running through all Quality Assurance test cases.

  See Section 2.1.2, "How to Set Up the Oracle Fusion Middleware Environment for AIA Development."

- Set up AIA Workstation.

  AIA Workstation is the designated system where the AIA Foundation Pack is set up.

  See Section 2.1.2, "How to Set Up the Oracle Fusion Middleware Environment for AIA Development."

2.1.1 How to Set Up JDeveloper for AIA Development

Task 1 Set up JDeveloper for AIA development

1. Download JDeveloper 11.1.1.4.0 or above and install from http://www.oracle.com/technology/software/products/jdev/htdocs/soft11.html.
2. Update JDeveloper with the SOA Suite Composite Editor plug-in from the Update center.
   a. Go to **Help** > **Check for Updates**.
   b. Select **Oracle Fusion Middleware Products** and search.
   c. Select **SOA Composite Editor**, and download and apply it.
3. Update JDeveloper with AIA Service Constructor plug-in from the Update center.
   a. Go to **Help** > **Check for Updates**.
   b. Select **Oracle Fusion Middleware Products** and search.
   c. Select **AIA Service Constructor**, and download and apply it.
4. Apply Freemarker 2.3.15 or higher needed by AIA Service Constructor.
   a. Download **Freemarker template engine** from http://www.freemarker.org.
   b. Put **freemarker.jar** (extracted from the downloaded zip file) in the jdeveloper/jdev/lib folder.

Task 2 Create a connection to the SOA Suite server

1. Create a connection to the SOA Suite server set up for AIA Development and Testing using these details after Oracle Fusion Middleware is set up.

   For more information see Section 2.1.2, "How to Set Up the Oracle Fusion Middleware Environment for AIA Development".

   a. Connection name: <*give a connection name for your application*>.
   b. Connection Type: use the default value - *WebLogic 10.3*.
   c. Provide the username and password of your WebLogic server.

      WebLogic Host name: <your server host name>.
   d. Port: <*your server port number*>.
   e. SSl port: <*your server ssl port*>.
   f. WLS Domain: <*provide your domain name where SOA server is installed*>.

Task 3 Create the database connection for SOA-MDS connection.

Create a connection to SOA MDS set up after Oracle Fusion Middleware is set up.

For more information see Section 2.1.2, "How to Set Up the Oracle Fusion Middleware Environment for AIA Development".

1. From the Resource Palette, click **New**.
2. Select the **New Connections** and then select a **Database**.
3. Create a database connection using MDS DB credentials. Contact your administrator for details.

Task 4 Create the SOA-MDS connection

1. From the Resource Palette, click **New**.
2. Select the **New Connections** and then select **SOA-MDS.**
3. Provide a connection name and select connection type as *DB based MDS*.
4. Select the DB connection you created in the previous step.
5. Select the MDS partition as *SOA-Infra* and **Save**.

Task 5 Set up the Harvester utility

1. Download **AIAHarvester.zip** from $AIA_HOME/ Infrastructure/LifeCycle.
2. Unzip to a local folder. Use this to harvest the composites.

For more information about harvesting AIA content, see Chapter 5, "Harvesting Oracle AIA Content.".

2.1.2 How to Set Up the Oracle Fusion Middleware Environment for AIA Development

The Oracle Fusion Middleware components used in the development environment depend on the topology of the development environment:

- Oracle SOA Suite

  Deploy and test AIA Services.

  See Section 2.1.2.1, "Set Up Oracle SOA Suite."

- Oracle Enterprise Repository (optional)

  Test harvesting of AIA services and ensure that all required service annotations are in place.

  See Section 2.1.2.2, "Set Up Oracle Enterprise Repository."

- Oracle Service Registry (optional)

  Test publishing of AIA services.

  See Section 2.1.2.3, "Set Up Oracle Service Registry."

- Oracle Business Process Publisher (optional)

  Analyze the delivered AIA Reference Process Models.

  See Section 2.1.2.4, "Set Up Oracle Business Process Publisher."

2.1.2.1 Set Up Oracle SOA Suite

1. Download Oracle SOA Suite 11gR1PS3 and install from http://www.oracle.com/technology/products/soa/soasuite/collateral/downloads.html.
2. Deploy AIA Foundation Pack artifacts from AIA Workstation.

   For more information, see Section 2.1.3.17, "How to Deploy AIA Foundation Pack Artifacts to Oracle SOA Suite Server".

2.1.2.2 Set Up Oracle Enterprise Repository

1. Download Oracle Enterprise Repository and install from http://www.oracle.com/technology/products/soa/repository/index.html.
2. Import AIA SOA portfolio Oracle Enterprise Repository Solution Pack.

For more information, see Chapter 11, "Configuring and Using Oracle Enterprise Repository as the Oracle AIA SOA Repository".

2.1.2.3 Set Up Oracle Service Registry

Download Oracle Service Registry and install from http://www.oracle.com/technology/products/soa/registry/index.html.

2.1.2.4 Set Up Oracle Business Process Publisher

Oracle Business Process Publisher is installed as part of the AIA Foundation Pack install.

2.1.3 How to Set Up AIA Workstation

AIA Workstation is a dedicated system set up to drive AIA development. Install the AIA Foundation Pack on this system and set up the AIA Project Lifecycle Workbench from the AIA Foundation Pack.

This section includes the following topics:

2.1.3.1 Prerequisites

- Install 11*g* WebLogic Server with Oracle Application Development Framework.
- Install 11*g* Database.
- Install JDeveloper.

Hardware should be at least 2 CPU and 4 GB of RAM with a supported Operating System.

2.1.3.2 How to Install AIA Foundation Pack

To install AIA Foundation Pack:

1. Install the AIA Foundation Pack

   For more information, see "Installing and Deploying Using AIA Foundation Pack Installer" in the *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack*.

2. Select the **Copy AIA Software** option.
3. The system creates **$AIA_HOME** and copies all the software to it.

   When new artifacts are created as a part of development, copy them to a relevant folder in $AIA_HOME. If a separate source control is maintained for development, then content from the builds generated is expected to be copied to $AIA_HOME.

   This enables content to take advantage of the features provided in AIA Project Lifecycle Workbench.

2.1.3.3 Updating SOA MDS with AIA MetaData

The content under $AIA_HOME/AIAMetaData provided as part of AIA Foundation Pack is uploaded to SOA-MDS. This content includes all the schemas, WSDLs, XSLs, domain value map (DVM) and Cross Reference meta information, default fault policies, AIAConfigurationProperties.xml, and AIAEHNotification.xml.

> **Note:** After any extensions or customizations on the artifacts in $AIA_HOME/AIAMetaData, copy back and upload the artifacts to SOA-MDS > apps/AIAMetaData.

When new artifacts similar to the ones in $AIA_HOME/AIAMetaData, are created as part of development, copy them to a relevant folder in $AIA_HOME/AIAMetaData to facilitate their upload to SOA-MDS > apps/AIAMetaData.

If a separate source control is maintained for all development, the content from the builds generated are expected to be copied to $AIA_HOME/AIAMetaData to facilitate uploading to SOA-MDS > apps/AIAMetaData.

2.1.3.4 Using MDS in AIA

Oracle Metadata Services (MDS) repository contains metadata for deployed J2EE applications, including SOA Suite on WLS.

Under a partition SOA-MDS created specifically for SOA, all SOA Composites (including AIA composites) are also stored upon deployment.

Under the same partition, the contents of $AIA_HOME/AIAMetaData are uploaded to SOA-MDS > apps/AIAMetaData.

The content and details of each set of metadata and how it is used by AIA is provided below. Also described is the process of creating new content or changing existing content.

Uploading the $AIA_HOME/AIAMetaData content to MDS is also described in Section 2.1.3.15, "Updating MDS".

2.1.3.5 Content of $AIA_HOME/AIAMetaData

AIA MetaData ($AIA_HOME/AIAMetaData) includes the following content:

AIAComponents - Presents the various schemas and WSDLs referred to by various services. The structure is as follows:

- **ApplicationConnectorServiceLibrary**: Abstract WSDLs of various Application Business Connector Services (ABCSs)
- **ApplicationObjectLibrary**: WSDLs of services exposed by applications and schemas of application business objects
- **B2BObjectLibrary**: Business-to-business (B2B) schemas
- **B2BServiceLibrary**: Abstract WSDLs of various B2B Connector Services (B2BCSs) and B2B Infrastructure Services
- **BusinessProcessServiceLibrary**: Abstract WSDLs of Composite Business Processes (CBPs) and Enterprise Business Flows (EBFs)
- **EnterpriseBusinessServiceLibrary**: Abstract WSDLs of Enterprise Business Services (EBSs)
- **EnterpriseObjectLibrary**: Schemas of the Oracle AIA Canonical Model
- **ExtensionServiceLibrary**: Concrete WSDLs pointing to mirror servlet
- **InfrastructureServiceLibrary:** Abstract WSDLs of infrastructure services
- **Transformations**: XSLs shared among various services
- **UtilityArtifacts**: Utility schemas and WSDLs

config: AIAConfigurationProperties.xml and AIAEHNotification.xml

dvm: Domain Value Maps

faultPolicies: Default policies applicable to all the services

xref: Metadata for Cross References

2.1.3.6 Working with AIA Components Content in $AIA_HOME/AIAMetaData

The AIA Components consist of Schemas, WSDLs, and XSLs shared among various AIA artifacts at run time. Usage and purpose of each of these files is dealt with in detail in AIA artifact-specific chapters of this guide.

AIA Components Folder Structure

All the abstract WSDLs of various application connector services and adapter services are stored here. The folder structure convention followed is: ApplicationConnectorServiceLibrary.

AIAMetaData\AIAComponents\ApplicationConnectorServiceLibrary\<Application Name>\<Version Number>\<Service Type>

- Possible values for <Version Number> are V1, V2, and so on.
- Possible values for <Service Type> are:

  - RequesterABCS
  - ProviderABCS
  - AdapterServices
- Possible values for <Application Name> are:
  - PeopleSoft
  - BRM
  - UCM
  - SAP
  - PIM
  - OracleRetail
  - Logistics
  - JDEE1
  - CRMOD
  - Agile
  - Ebiz
  - Siebel

Note: The <Application Name> specified here is used in AIA Project Lifecycle Workbench in the definition of the bill of materials for deployment. It should match the <productCode> list of values in the Workbench.

Examples:

AIAMetaData/AIAComponents/ApplicationConnectorServiceLibrary/Siebel/V1/RequestorABCS

AIAMetaData/AIAComponents/ApplicationConnectorServiceLibrary/Siebel/V1/ProviderABCS

ApplicationObjectLibrary

All the WSDLs of the services exposed by the participating applications and the referenced schemas are stored in:

$AIA_HOME/AIAMetaData/AIAComponents/ApplicationObjectLibrary

Applications consume AIA requester service WSDLs. To avoid any need for transformation in the participating applications, the AIA requester services' WSDLs are developed referencing the external facing business object schemas of the participating applications. These schemas are also stored in:

$AIA_HOME/AIAMetaData/AIAComponents/ApplicationObjectLibrary.

The folder structure convention followed is:

ApplicationObjectLibrary/<Application Name>/<Version Number>/schemas

ApplicationObjectLibrary/<Application Name>/<Version Number>/wsdls

The possible values for <Application Name> and <Version Number> are as described in the previous section.

Note: The <Application Name> specified here is used in AIA Project Lifecycle Workbench in the definition of the bill of materials for deployment.

Examples:

AIAMetaData/AIAComponents/ApplicationObjectLibrary/Siebel/V1/schemas

AIAMetaData/AIAComponents/ApplicationObjectLibrary/Siebel/V1/wsdls

B2BServiceLibrary

All of the abstract WSDLs of B2B Connector Services (B2BCSs) are stored in this location.

Requester B2BCS WSDLs are stored under $AIA_HOME/AIAMetaData/AIAComponents/B2BServiceLibrary/Connectors/wsdls.

The folder structure convention followed is: B2BServiceLibrary/Connectors/wsdls/<B2BStandard>/RequesterB2BCS/<Connector Version>/*.wsdl.

Provider B2BCS WSDLs are stored under $AIA_HOME/AIAMetaData/AIAComponents/B2BServiceLibrary/Connectors/wsdls.

The folder structure convention followed is: B2BServiceLibrary/Connectors/wsdls/<B2BStandard>/ProviderB2BCS/<Connector Version>/*.wsdl.

Other abstract WSDLs of reusable infrastructure services are stored under $AIA_HOME/AIAMetaData/AIAComponents/B2BServiceLibrary/Infrastructure/<ServiceVersion>/.

BusinessProcessServiceLibrary

All the abstract WSDLs of Composite Business Processes and Enterprise Business Flows are stored in:

$AIA_HOME/AIAMetaData/AIAComponents/ BusinessProcessServiceLibrary

The folder structure convention followed is:

BusinessProcessServiceLibrary/<Service Type>

The possible values for <Service Type> are CBP and EBF.

Example:

AIAMetaData/AIAComponents/BusinessProcessServiceLibrary/CBP

AIAMetaData/AIAComponents/BusinessProcessServiceLibrary/EBF

EnterpriseBusinessServiceLibrary

Part of Oracle AIA Canonical Model

All the abstract WSDLs of Enterprise Business Services are stored in:

$AIA_HOME/AIAMetaData/AIAComponents/EnterpriseBusinessServiceLibrary

EnterpriseObjectLibrary

Part of Oracle AIA Canonical Model

All the schema modules of the Enterprise Object Library are stored in:

$AIA_HOME/AIAMetaData/AIAComponents/EnterpriseObjectLibrary

ExtensionServiceLibrary

All the concrete WSDLs pointing to mirror servlet are stored in:

$AIA_HOME/AIAMetaData/AIAComponents/ExtensionServiceLibrary

The folder structure convention followed is:

ExtensionServiceLibrary/<Application Name>

The possible values for <Application Name> are described in theAIA Components Folder Structure section.

> **Note:** The <Application Name> specified here is used in AIA Project Lifecycle Workbench in the definition of the bill of materials for deployment.

Examples:

AIAMetaData/AIAComponents/ExtensionServiceLibrary/Siebel

InfrastructureServiceLibrary

All the abstract WSDLs of infrastructure services are stored in:

$AIA_HOME/AIAMetaData/AIAComponents/InfrastructureServiceLibrary

The folder structure convention followed is**:**

InfrastructureServiceLibrary/<Version Number>

Example:

AIAMetaData/AIAComponents/InfrastructureServiceLibrary/V1

Transformations

All the XSLs shared among various AIA services are stored in:

$AIA_HOME/AIAMetaData/AIAComponents/Transformations

The folder structure convention followed is:

Transformations/<Application Name>/<Version>

The possible values for <Application Name> and <Version Number> are described in the AIA Components Folder Structure section.

> **Note:** The <Application Name> specified here is used in AIA Project Lifecycle Workbench in the definition of the bill of materials for deployment.

Example:

AIAMetaData/AIAComponents/Transformations/Siebel/V1

UtilityArtifacts

All the Utility schemas and WSDLs are stored in:

$AIA_HOME/AIAMetaData/AIAComponents/UtilityArtifacts

The folder structure convention followed is:

UtilityArtifacts/schemas

UtilityArtifacts/wsdls

2.1.3.7 How to Change an Existing File

To change an existing file:

1. From JDeveloper, open the relevant file by browsing to AIAWorkstation/$AIA_HOME/AIAMetaData/AIAComponents/<Path to file>
2. Make modifications. Review the upgrade safe extensibility guidelines provided in AIA artifact-specific chapters of the guide.
3. Save.
4. Upload to SOA-MDS > apps/AIAMetaData/AIAComponents. Refer to Section 2.1.3.15, "Updating MDS".

2.1.3.8 How to Create File

To create a file:

1. In JDeveloper, create a file following the design and development guidelines provided in AIA artifact-specific chapters of the guide.
2. Copy the file to AIAWorkstation/$AIA_HOME/AIAMetaData/AIAComponents/<Path to file>.

   Note: Place the file in this folder.

3. Upload to SOA-MDS > apps/AIAMetaData/AIAComponents. Refer to Section 2.1.3.15, "Updating MDS".

Accessing the Files in the AIA Components Folder

Use the following protocol to access the AIA Components content from all the AIA service artifacts at design time and run time:

oramds:/apps/AIAMetaData/AIAComponents/<Resource Path Name>

Example:

oramds:/apps/AIAMetaData/AIAComponents/ApplicationObjectLibrary/SampleSEBL/schemas/CmuAccsyncAccountIo.xsd

Note: All of the files in the AIA Components folder use relative paths to refer to other files in the AIA Components folder, if needed.

The WSDLs in the Enterprise Business Service Library use relative paths to refer to schemas in the Enterprise Object Library.

2.1.3.9 How to Work with AIAConfigurationProperties.xml in $AIA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/config

AIA provides external configuration properties to influence the run-time behavior of system, infrastructure components, and services. These properties are provided as name-value pairs at the system, module, and service levels in AIAConfigurationProperties.xml.

The AIAConfigurationProperties.xml supports two types of configurations:

- System level, including module level

  Contains system-level configuration name-value pairs and module-level configuration name-value pairs within the system level.

- Service level

  Contains service-specific configuration name-value pairs.

The following XPath functions are provided to access the configuration name-value pairs in the **AIAConfigurationProperties.xml**:

- *aiacfg:getSystemProperty* (propertyName as string, needAnException as boolean) returns propertyValue as string
- *aiacfg:getSystemModuleProperty* (moduleName as string, propertyName as string, needAnException as boolean) returns propertyValue as string
- *aiacfg:getServiceProperty* (EBOName as string, serviceName as string, propertyName as string, needAnException as boolean) returns propertyValue as string

"*getSystemModuleProperty()*" and "*getServiceProperty()*" functions first look for the appropriate module and service property and if it is not found, they then look for a system property with the same property name.

In all three functions, if a matching property is not found, the result depends upon the value of the needAnException argument. If need AnException is true, then a PropertyNotFound Exception is thrown; otherwise, an empty string is returned.

2.1.3.10 How to Add a New Property to AIAConfigurationProperties.xml

To add a new property to AIAConfigurationProperties.xml:

1. From JDeveloper, open the file *AIAConfigurationProperties.xml* by browsing to AIAWorkstation/$AIA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
2. Make modifications, as needed in AIA artifact development.
3. Save.
4. Upload to SOA-MDS > apps/AIAMetaData/config. Refer to Section 2.1.3.15, "Updating MDS".
5. From the AIA Home Page, click Go in the Setup area. Select the Configuration tab to access the Configuration page. Click Reload to refresh the AIA Configuration cache.

2.1.3.11 How to Work with AIAEHNotification.xml in $AIA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/config

This file is the template of the email notification sent as a part of the Error Handling Framework.

To modify the AIAEHNotification.xml:

1. From JDeveloper, open the file *AIAEHNotification.xml* by browsing to AIAWorkstation/$AIA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
2. Modify as needed.
3. Save.
4. Upload to SOA-MDS > apps/AIAMetaData/config. Refer to Section 2.1.3.15, "Updating MDS".

2.1.3.12 How to Work with Domain Value Maps in $AIA_HOME/AIAMetaData/dvm

The Domain Value Maps utility is a feature of Oracle SOA Suite. It supports the creation of static value maps and provides the custom XPath function:

```
dvm:lookupValue("oramds:/apps/AIAMetaData/dvm/<DVM Map Name>",<Source
Column>,<Source Column Value>,<Target Column>,"")
```

For more information about DVMs, see Section 25.4, "Working with DVMs and Cross-References."

The DVMs are used by all the AIA Pre-Built Integrations and are shipped as part of AIA Foundation Pack. They are stored in $AIA_HOME/AIAMetaData/dvm.

To modify the Domain Value Maps:

1. From JDeveloper, open the DVM file by browsing to AIAWorkstation/$AIA_HOME/AIAMetaData/dvm.
2. Modify as needed.
3. Save.
4. Upload to SOA-MDS > apps/AIAMetaData/dvm. Refer to Section 2.1.3.15, "Updating MDS".

2.1.3.13 How to Work with Cross Reference (Xref) in $AIA_HOME/AIAMetaData/xref

The Cross Reference utility is a feature of Oracle SOA Suite. It supports the creation of dynamic values.

For more information about cross references, see "Working with Cross References" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite 11g*.

The Cross Reference meta information as used by all the AIA Process Integration Packs are shipped as part of AIA Foundation Pack and are stored in $AIA_HOME/AIAMetaData/xref.

To modify the Cross Reference metadata:

1. From JDeveloper, open the Cross Reference file by browsing to AIAWorkstation/$AIA_HOME/AIAMetaData/xref.
2. Modify as needed.
3. Save.
4. Upload to SOA-MDS > apps/AIAMetaData/xref. Refer to Section 2.1.3.15, "Updating MDS".

2.1.3.14 How to Work with Fault Policies in $AIA_HOME/AIAMetaData/faultPolicies/V1

The default fault policy file "fault-bindings.xml" is shipped as part of AIA Foundation Pack and is stored in $AIA_HOME/AIAMetaData/faultPolicies/V1.

To modify the "fault-bindings.xml":

1. From JDeveloper, open the fault-bindings.xml file by browsing to AIAWorkstation/$AIA_HOME/AIAMetaData/faultPolicies/V1.
2. Modify as needed.
3. Save.
4. Upload to SOA-MDS > apps/AIAMetaData/faultPolicies/V1. Refer to Section 2.1.3.15, "Updating MDS".

2.1.3.15 Updating MDS

> **Note:** Repeat this procedure **every time** a file is added to MDS.

To update SOA-MDS > apps/AIAMetaData:

1. Browse to the folder at $AIA_HOME/aia_instances/$INSTANCE_NAME/bin.
2. Source the file aiaenv.sh by executing the following command:

   ```
   source aiaenv.sh
   ```

3. Browse to the folder at $AIA_HOME/aia_instances/$INSTANCE_NAME/config and open the deployment plan file, UpdateMetaDataDP.xml.
4. Update the file UpdateMetaDataDP.xml by inserting include tags for each resource group that you want to add to the MDS:
   a. To upload all the files under "AIAMetaData", add the following:

      ```
      <include name ="**"/>
      ```

   b. To upload the files copied to "AIAComponents/ApplicationObjectLibrary/SEBL/schemas" folder, add the following:

      ```
      <include name ="AIAComponents/ApplicationObjectLibrary/SEBL/schemas/**"/>
      ```

   > **Note:** In the include tag, the folder path must be relative to the folder AIAMetaData.

5. Browse to AIA_HOME/Infrastructure/Install/config. Execute the script UpdateMetaData.xml by typing the command:

   ```
   ant -f UpdateMetaData.xml
   ```

Note: MetaData gets deleted when you uninstall Foundation Pack. For specific instruction on how to clean the MDS after you uninstall Foundation Pack, see *Cleaning the MDS* in *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack*.

2.1.3.16 How to Set Up AIA Project Lifecycle Workbench

The AIA Project Lifecycle Workbench application is set up on the AIA Workstation. All the members of the AIA implementation team developing services use the AIA Project Lifecycle Workbench.

For more information about setting up AIA Project Lifecycle Workbench, see *Oracle Fusion Middleware Installation Guide for Oracle Application Integration Architecture Foundation Pack*, "Post Install Configurations," Setting up AIA Roles.

2.1.3.17 How to Deploy AIA Foundation Pack Artifacts to Oracle SOA Suite Server

For more information about deploying AIA Foundation Pack artifacts to the Oracle Fusion Middleware-SOA server set up for AIA development, see *Oracle Fusion Middleware Installation Guide for Oracle Application Integration Architecture Foundation Pack*, "Installing and Deploying Using AIA Foundation Pack Installer."

2.1.3.18 How to Deploy AIA Service Artifacts to Oracle SOA Suite Server

Define an AIA project in the AIA Project Lifecycle Workbench and capture meta information for all of the AIA service artifacts to be developed. The system uses this information to generate a deployment plan specific to the AIA project.

For more information about deploying all the AIA Project artifacts using a deployment plan generated from AIA Project Lifecycle Workbench to the Oracle Fusion Middleware-SOA server set up for AIA development, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts.".

2.2 Role of AIA Project Lifecycle Workbench

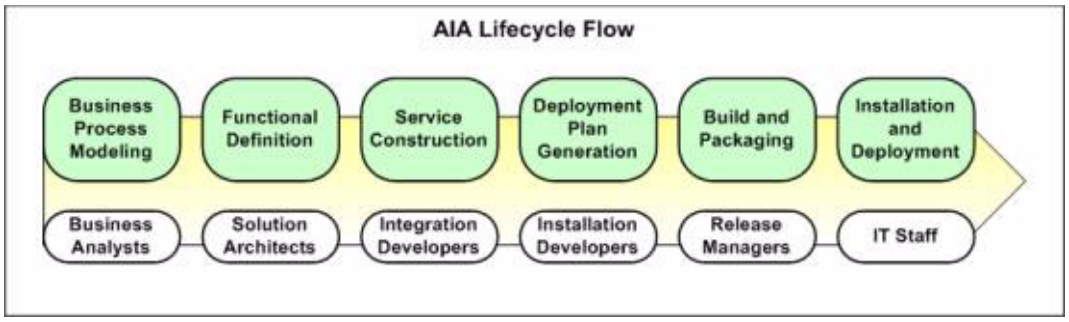
An AIA project is about engineering service-oriented business processes. The outcome of business process analysis and reengineering is taken as input, leading to conceptualization of service, planning, development, deployment and support paradigms of a typical SOA project.

The AIA Project Lifecycle flow consists of the following phases:

- Business Process Modeling and Analysis
- Business Process Decomposition and Service Conception
- Service Design and Construction
- Installation and Deployment

  Figure 2–1 illustrates the AIA project lifecycle flow showing the six phases and the actors involved in each phase.

Figure 2–1 AIA Project Lifecycle Flow: Phases and Actors



2.2.1 Introduction to the Tools Used

The following tools support the entire lifecycle of the an AIA project:

- Oracle Business Process Publisher
- Oracle Enterprise Repository
- AIA Service Constructor plugin in JDeveloper
- Oracle Enterprise Repository and AIA Harvester
- AIA Deployment Plan Generator
- AIA Project Lifecycle Workbench

Oracle Business Process Publisher

Use the Oracle Business Process Publisher to analyze the Reference Process Models delivered as part of Foundation Pack.

For more information about Reference Process Models, see the *Oracle Fusion Middleware Reference Process Models User's Guide for Oracle Application Integration Architecture Foundation Pack*.

Oracle Enterprise Repository

Oracle Enterprise Repository is the design-time repository for governing design-time and run-time assets to achieve reuse and sharing across the distributed development community. Oracle Enterprise Repository provides:

- **Visibility -** From design time to run time, captures and maintains metadata, relationships, categories, and so on.
- Control - Manages development lifecycle transition from concept, through implementation, to release.
- Evolution - Empowers customers to evolve their business and integration processes.

For more information about Oracle Enterprise Repository, see Chapter 11, "Configuring and Using Oracle Enterprise Repository as the Oracle AIA SOA Repository."

AIA Service Constructor

The AIA Service Constructor is an Oracle JDeveloper plug-in to generate composites conforming to AIA guidelines and naming standards. It also provides guidance for annotating the artifacts to support governance.

For more information about the AIA Service Constructor, see Chapter 4, "Working with Service Constructor."

AIA Harvester Tool

The AIA Harvester parses the AIA service artifacts and captures metadata into the AIA Project Lifecycle Workbench database and the Oracle Enterprise Repository, if used. The system uses this information to generate deployment plans.

AIA leverages Oracle Enterprise Repository to achieve SOA visibility. Oracle Enterprise Repository is an optional component to AIA installation and execution.

AIA Harvester is built on top of the Oracle Enterprise Repository Harvester Extension Framework. It introspects SOA artifacts and publishes their ensuing metadata into the Project Lifecycle Workbench back end or Oracle Enterprise Repository (optional), or both, to aid governance and downstream automation. In AIA, the AIA Harvester is provided in the form of command-line utility. Users may download and set up on their local development environment.

For more information about AIA Harvester, see Chapter 5, "Harvesting Oracle AIA Content."

AIA Project Lifecycle Workbench

The AIA Project Lifecycle Workbench is installed on the AIA Workstation and helps drive the AIA Project Lifecycle flow.

For more information about the AIA Project Lifecycle Workbench, see Chapter 3, "Working with Project Lifecycle Workbench."

Deployment Plan Generator

The deployment plan has all the details for the artifacts in an AIA project to be deployed. The AIA Deployment Driver takes the deployment plan as input and deploys all the artifacts on Oracle Fusion Middleware servers and updates the end point information back into Oracle Enterprise Repository.

It also facilitates publishing this information into Oracle Service Registry.

For more information about the Deployment Plan Generator, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

2.2.2 Introduction to the Business Process Modeling and Analysis Phase

Business Process Models are blueprints of the work accomplished in an enterprise to add value and deliver to customers. The models represent the ideal way work should be carried out with optimal cost at highest possible efficiencies.

The analysis of these models and categorizing them into business processes, business activities, and tasks is done by **Business Analysts**. The result of this analysis is a set of reusable business activities and tasks, which can be woven into different business processes.

In this phase, Business Analysts identify and catalog the set of reusable business activities and tasks for the business processes. Business Analysts also establish the Key Performance Indicators (KPIs) that are to be met. They use the Oracle BPA Suite to analyze the AIA Reference Process Models and identify the business processes, business activities, and tasks to be implemented.

Business Analysts define the project and provide details of solutions needed.

For more information about reference process models in Oracle AIA, see the "*Oracle Fusion Middleware Reference Process Models User's Guide for Oracle Application Integration Architecture Foundation Pack*".

For more information about the AIA Project Lifecycle Workbench, see Chapter 3, "Working with Project Lifecycle Workbench."

2.2.3 Introduction to the Business Process Decomposition and Service Conception Phase

Each of the business activities and tasks can be described in terms of business entities involved and the action being performed with them. The origins of the service definition are in the categorization of reusable business activities and tasks.

- Examples of **business entities** are *Customer*, *Product*, *Order*, and so on.
- Examples of **tasks** are *Create Customer*, *Update Order*, *Enter Service Request*, *Request Account Balance*, and so on.
- Examples of **business activities** are *Do Customer Credit Check*, *Fulfill Order*, *Process Trouble Ticket*, and so on.

The EBSs defined as AIA artifacts represent the business activities and tasks. The metadata about the EBSs are available in Oracle Enterprise Repository after loading the AIA Solution Pack.

Various business activities and tasks defined in AIA Reference Process Models provide links to the corresponding EBSs in the Oracle Enterprise Repository.

Solution Architects refine the projects defined by business analysts, identify the services available, create definitions for new services to be developed, and engage with developers to drive the design of new services.

For more information about the Project Lifecycle Workbench, see Chapter 3, "Working with Project Lifecycle Workbench."

2.2.4 Introduction to the Service Design and Construction Phase

During the service construction phase, **Developers** extend available AIA services, if needed, or design and develop new services.

The tasks of this phase are supported by the following tools:

- The service solution components defined in the **Project Lifecycle Workbench** provide guidance to developers about the fine-grained services they must create to fulfill the functionality of a task.

  For more information about Project Lifecycle Workbench, see Chapter 3, "Working with Project Lifecycle Workbench."

- The **Service Constructor** enables developers to automatically generate AIA-compliant SCA composites for ABCSs.

  For more information about Service Constructor, see Chapter 4, "Working with Service Constructor."

- The **Composite Application Validation System (CAVS)** supports process and service testing.

  For more information about CAVS, see "Preparing to Use the Composite Application Validation System" in the *Oracle Fusion Middleware Infrastructure*

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

- The **error handling framework** augments SCA exception management and retries.

  For more information about error handling, see "Setting Up Error Handling." in the *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

2.2.5 Introduction to the Deployment Plan Generation Phase

After composites are constructed, installation developers select desired composites that collectively perform a required functionality. An AIA project is a collection of functionally related composites. Each composite is self-contained in that its deployment information (such as adapters, queues, schemas, and JDNI resources) is fully specified by its developer during the preceding service construction stage.

The installation developer creates an AIA project specific deployment plan by aggregating deployment information from all composites that form an AIA project. The developer may choose to use the Deployment Plan Generator to automatically generate process deployment plans.

For more information about error handling and logging, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

2.2.6 Introduction to the Install and Deploy Phase

The AIA Installer helps to deploy an AIA project release into your development environment. The AIA Installer makes all AIA composites and other artifacts available to target environments. It deploys the composites and other artifacts specified in your selected deployment plan. It also publishes assets to a SOA repository and registry, respectively.

For more information about the AIA Installer, see the *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack*.

2.3 AIA Artifacts in Various Integration Styles

AIA Architecture recommends variety of integration styles and AIA patterns to enable the flight of a message in an **Integration Flow**.

AIA implementation teams should evaluate the following points to choose an integration style and methodology to follow when creating various AIA artifacts.

Business Scenario - The **Functional Design Document (FDD)** describes the Business Scenario and provides:

- A detailed description of the business case.
- Various use cases detailing the various usage scenarios, including exception cases with expected actions by various actors.
- Details about all the participating applications - commercial, off-the-shelf with versions and homegrown.
- Details about the triggering business events.
- Details about the functional flow.
- Details about performance and scalability requirements.

- Details about business objects to be used.
- Actions to be performed on the various business objects.
- Oracle tools and technologies to be leveraged

The AIA artifacts that are required for the collaboration between the applications or functions are dependent on the integration style adopted for an *Integration Flow*. These artifacts include ABCSs, EBSs, EBFs, CBPs, and various adapter services.

The AIA Project Lifecycle Workbench is used to define an AIA Project, which contains definitions of all the artifacts to be designed, developed, tested, packaged, and delivered.

2.3.1 Integration Through Native Application Interfaces Using the Oracle Applications Technology Infrastructure

In this style, messages flow from the requester application to the providing application. The mode of connectivity could be SOAP/HTTP, queues, topics, or native adapters.

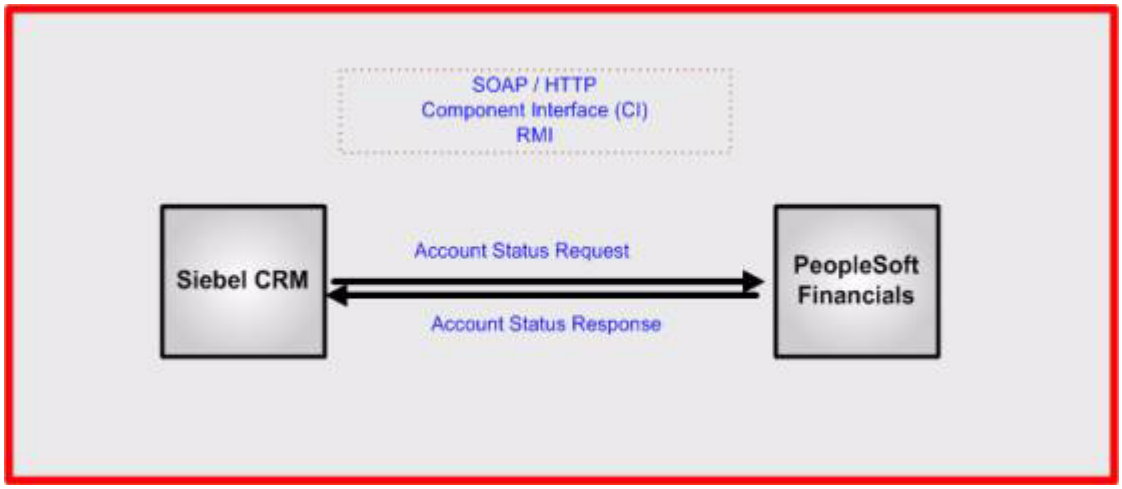
No middleware is involved in this integration.

The requester application must establish the connectivity with the provider applications. The requester application is responsible for sending the request in the format mandated by provider's API and interpreting the response sent by the provider. Requester and provider applications are responsible for the authentication and authorization of requests.

The integration flow consists of individual application functions interacting directly. All capabilities required to make this interaction possible must be available or made available in the individual applications.

Figure 2–2 illustrates how a requester application interacts directly with a provider application.

Figure 2–2 Example of a Requester Application Interacting Directly with a Provider Application



In more complex situations, when the integration flow consists of multiple steps involving interactions with multiple applications, leverage the workflow-like capability in one or more applications.

No AIA artifacts are built in this case. Establish direct connectivity between applications.

For more information about different modes of connectivity, see Chapter 23, "Establishing Resource Connectivity."

2.3.2 Understanding Integration Styles with Integration Framework

In all the integration styles with integration framework, middleware technologies are leveraged. The applications push the messages to the middleware, and the middleware pushes the messages to the target applications.

These integration styles have integration framework:

- Integration flow leveraging provider services
- Integration flow leveraging provider services with canonical model-based virtualization

The AIA service artifacts needed for implementing an integration flow in any of the integration styles with integration framework depends on:

- Complexity of data exchange
  - No processing logic
  - With processing logic
- Message exchange pattern
  - Synchronous request-response
  - Asynchronous one-way (fire-and-forget) - Need for guaranteed delivery assumed
  - Asynchronous request-delayed response - Need for guaranteed delivery and correlation of response assumed

These AIA service artifacts are defined in AIA Architecture:

- CBPs
- EBSs
- EBFs
- ABCSs
- Adapter Services
  - JMS Producer and JMS Consumer
  - JCA Adapter Service

For more information about different message exchange patterns, see Chapter 13, "Designing and Developing Enterprise Business Services" and Chapter 27, "Working with AIA Design Patterns."

For more information about different AIA service artifacts and Oracle SOA Suite components used to implement them, see the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*.

Note: In the following sections, for each integration style with integration framework, the AIA artifacts to be developed are recommended for a combination of situations.

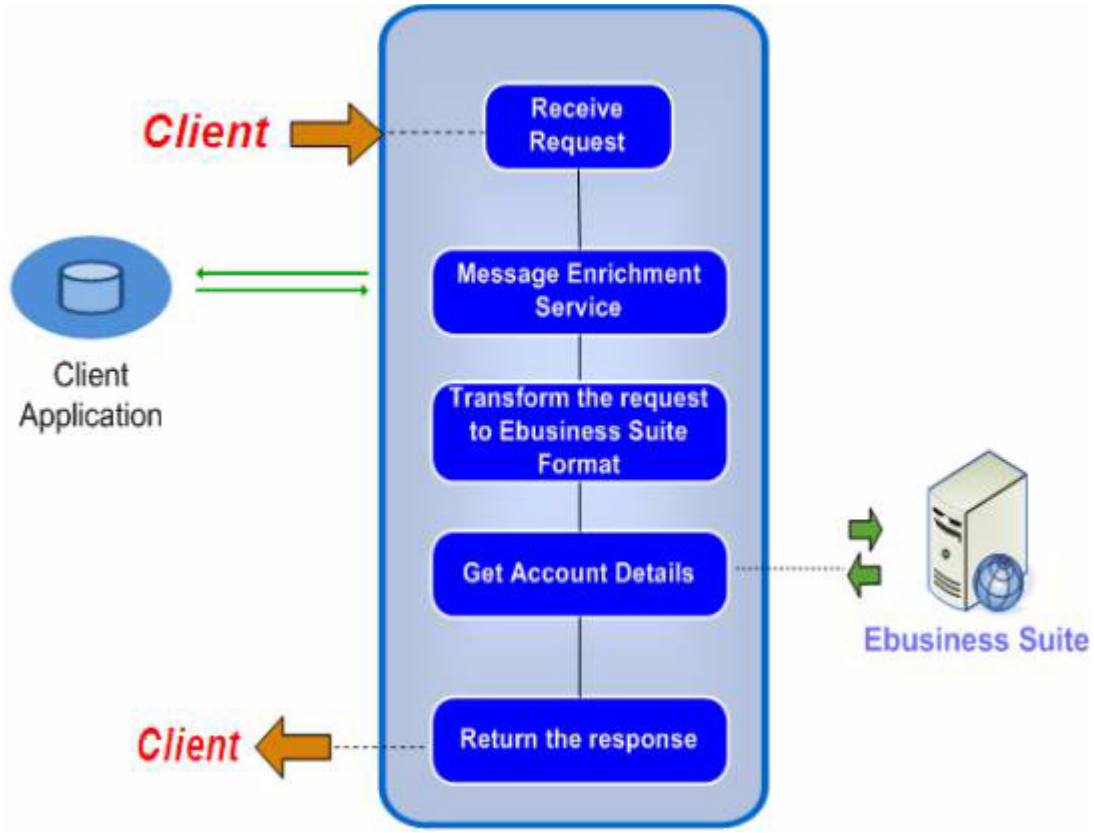
2.3.2.1 Integration Flow with Requester Application Services

The requester application invokes a single AIA service on the middleware. The request presented by the requester application is fulfilled by the AIA service by invoking suitable APIs in the provider applications. The AIA service can accept a message in the requester application format. The AIA service presents the request to the provider applications in the format mandated by the provider's API. The AIA service accepts the response in the provider application format, if needed. The AIA service is responsible for the authentication and authorization of the requests.

The integration flow consists of this single AIA service artifact deployed on the middleware managing all interactions with all participating applications.

Figure 2–3 illustrates how a service deployed on the middleware enables integration between the requester and the provider application.

Figure 2–3 Example of Integration Flow with Native Application Services



For more complex situations in which the integration flow consists of multiple steps involving interactions with multiple applications, the AIA service implements a workflow-like capability and manages all interactions with all the participating applications.

The AIA service artifacts to be developed depend on the complexity of data exchange and various message exchange patterns.

Table 2–1 lists suitable AIA artifacts for a combination of situations.

Table 2–1 AIA Artifacts for Integration Flows with Requester Application Services

Message Pattern	No Processing Logic	With Processing Logic
Synchronous Request Response	EBS	CBP
Asynchronous One-Way	EBS	CBP
Asynchronous Request-Delayed Response	EBS - Request EBS - Response	CBP

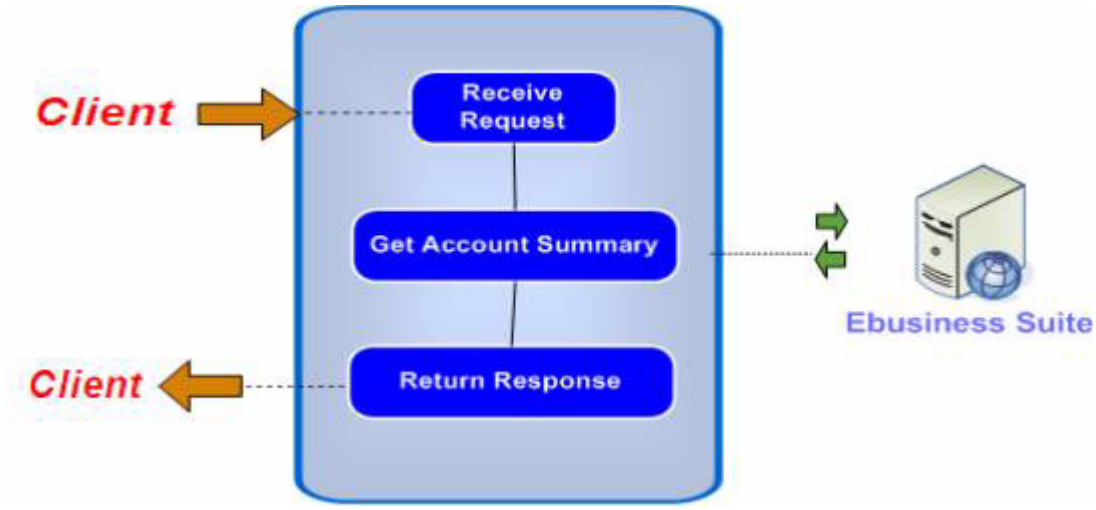
2.3.2.2 Direct Integration Through Application Web Services

A provider application-specific AIA service, exposing a coarse-grained functionality of the provider application leveraging one or more APIs, is created with a suitable provider application-specific interface. Several business initiators can invoke this AIA service. If the business initiators cannot present the request in the format understood by the provider application-specific AIA service, a requester application-specific AIA service is used to transform the business initiator request to the provider application format. The requester application-specific AIA service is responsible for authenticating and authorizing the requests. The provider application-specific AIA service propagates the authentication and authorization information of the requests to the provider application.

The integration flow would consist of a requester application-specific AIA service artifact deployed on the middleware managing all interactions with all provider application-specific AIA services.

Figure 2–4 illustrates how a service deployed on the middleware enables the integration between the requester and the provider application.

Figure 2–4 Example of Integration Flow Leveraging Provider Services



For more complex situations in which the integration flow involves interactions with multiple applications, the requester application-specific AIA service implements a workflow-like capability and manages all interactions with all the provider application-specific AIA services.

The AIA service artifacts to be developed depend on the complexity of data exchange and various message exchange patterns.

Table 2–2 lists suitable AIA artifacts for a combination of situations.

Table 2–2 AIA Artifacts for Leveraging Provider Services

Message Pattern	No Processing Logic	With Processing Logic
Synchronous Request Response	Requester ABCS Provider ABCS	Requester ABCS Provider ABCS
Asynchronous One-Way	Requester ABCS Provider ABCS	Requester ABCS Provider ABCS
Asynchronous Request-Delayed Response	Requester ABCS Provider ABCS	Requester ABCS Provider ABCS

2.3.2.3 Integration Through Packaged Canonical and Standardized Interfaces

Loose coupling through a canonical (application-independent) model is a sign of a true SOA. Participating applications in loosely coupled integrations communicate through a virtualization layer. Instead of direct mappings between data models, transformations are used to map to the canonical data model. While this allows for greater reusability, the transformations both increase the message size and consume more computing resources. For functional integrations, this integration pattern is ideal since the reusability gained is worth the slight overhead cost.

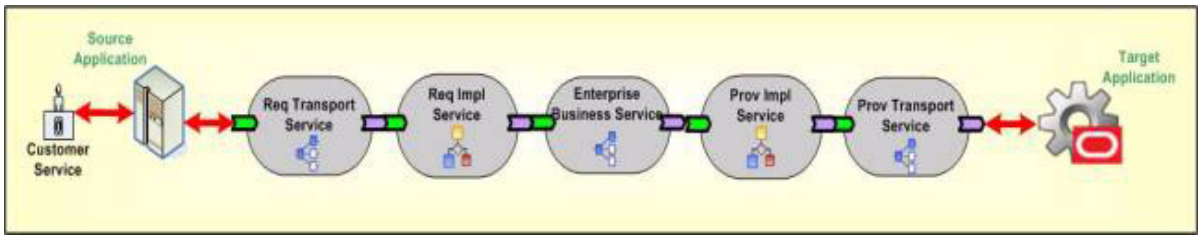
In this case, an EBS based on the Enterprise Business Objects (EBOs) and Enterprise Business Messages (EBMs) is created as a mediator service.

A provider service, exposing a coarse-grained functionality of the provider application leveraging one or more APIs, is created with the same EBM interface as the EBS operation interface.

If the business initiators cannot present the request in the format understood by the EBS operation interface, a requester service is used to transform the business initiator request into the provider service format.

Figure 2–5 illustrates how the request sent by the source application is processed by the target application with the help of the EBS and a set of intermediary services. The request and provider transport services are optional and are needed only in case of non-SOAP-based transports.

Figure 2–5 Example Showing Canonical Model-based Virtualization



For more complex situations in which the integration flow involves interactions with multiple applications, the requester application-specific AIA service presents its request to the mediator AIA service. The mediator AIA service triggers an AIA service, which implements a workflow-like capability and manages all interactions with all the provider application-specific AIA services through mediator AIA services. In this case, the mediator AIA service interface chosen is assumed to be accepted as the common interface. Thus, all requester application-specific AIA services invoke this mediator AIA service, and all the provider application-specific AIA services implement this

common interface. The AIA service artifacts to be developed depend on the complexity of data exchange and various message exchange patterns.

Table 2–3 lists suitable AIA artifacts for a combination of situations.

Table 2–3 AIA Artifacts for Integration Flows with Multiple Application Interactions

Message Pattern	No Processing Logic	With Processing Logic
Synchronous Request Response	Requester ABCS EBS Provider ABCS	Requester ABCS EBS Provider ABCS
Asynchronous One-Way	Requester ABCS EBS Provider ABCS	CBP Requester ABCS EBS EBF Provider ABCS
Asynchronous Request-Delayed Response	Requester ABCS EBS Provider ABCS	CBP Requester ABCS EBS EBF Provider ABCS

2.3.3 Bulk Data Processing

Bulk data processing involves a large batch of discrete records or records with very large data. The methodology is point-to-point integration specializing in the high-performance movement of data with a trade-off of reusability.

Bulk data processing is about persistent data replicated across multiple applications. The integrations fall into four styles:

- Initial data loads
- High-volume transactions with Xref table
- Intermittent high-volume transactions
- High-volume transactions without Xref table

For complete details about using bulk data processing with AIA, see Chapter 24, "Using Oracle Data Integrator for Bulk Processing."

2.3.4 Integration Style Choice Matrix

In any AIA implementation, a variety of integration flows conforming to different integration styles coexist. Integration flows represent processing of messages by AIA services deployed to deliver the business functionality desired.

The choice of an integration style influences the design of AIA services. Conversely, decisions about the design of AIA services lead to a particular integration style.

Various aspects of AIA service design are:

- **Service Granularity**

  Decide on the functionality to be implemented in a service either based on the business logic needed for a business activity or task (the result of business process analysis) or based on the functions exposed by a provider application.

Service granularity is **Coarse-Grained** if the business logic conforms to the requirements of a business activity or task (result of business process analysis of AIA Reference Process Models). In this case, the service is implemented by invoking multiple low APIs of the provider application.

Service granularity is **Granular** if the business logic conforms to the low-level functions exposed by the provider application.

- **Service Reusability**

  Service reuse is **High** for different Integration Flows if its design is modular and built for the requirements of various business use cases. Conformance to the requirements of a business activity or task (the result of business process analysis of AIA Reference Process Models) leads to modular design.

  If the service is built to meet the requirements of a particular business use case, then the logic is monolithic and reuse is **Low**.

- **Service Virtualization**

  This aspect provides for separation of concerns and independence of requester applications and provider applications from changes. The choices are **Yes** or **No**.

- **Service Interoperability**

  The capability of diverse service implementations to interoperate is dependent on their conformance to WS-I standards, a common message meta model, and a robust versioning strategy. The AIA EBS WSDLs and AIA EBOs and EBMs provide this capability as delivered. For others, special effort is needed to ensure interoperability.

Table 2–4 summarizes the granularity, reusability, virtualization, and interoperability for the various integration styles.

Table 2–4 AIA Service Design Summary

Integration Styles	Granularity	Reusability	Virtualization	Interoperability
Integration Flow with Native Application APIs	No AIA services on middleware; direct Application to Application interaction	No AIA services on middleware; direct Application to Application interaction	No AIA services on middleware; direct Application to Application interaction	No AIA services on middleware; direct Application to Application interaction
Integration Flow with Requester Application Services	Coarse Grained with High Performance overhead	Low	No	Special Effort
Integration Flow leveraging Provider Services	Granular	High	No	Special Effort
Integration Flow leveraging Provider Services with Canonical-Model based Virtualization	Coarse Grained	High	Yes	As Delivered

2.4 Development Tasks for AIA Artifacts

This section discusses the development tasks for AIA artifacts and describes how to:

- Identify the EBO
- Design an integration flow

- Identify and create the EBS
- Construct the ABCS
- Enable the participating applications
- Identify and create the EBF

2.4.1 Identifying the EBO

The FDD should discuss:

- The conceptual canonical model or EBOs to be used and the actions that must be performed on this entity.
- Identification of the EBO from the Foundation Pack or construction of EBO. The EBO should incorporate all of the requirements needed by this integration.
- Identification of the EBM from the Foundation Pack or construction of EBM, if required.

For more information, see the *Enterprise Object Library Extensibility Guide* on My Oracle Support in note ID 983958.1.

2.4.2 Designing an Oracle AIA Integration Flow

The design of an Integration Flow is detailed in a technical design document (TDD). The criteria for completion of a TDD are dependent on the project and the accompanying deliverables.

The TDD lays out complete details on the flow logic, integration artifacts, object/element mapping, DVM types and values, error handling, and installation specifics. It also includes an outline of the unit test plans that a developer uses to validate run-time operation.

The Integration Flow is not a run-time executable artifact, so a message exchange pattern cannot be attributed to the Integration Flow. The design of the AIA service artifacts listed previously must include a careful analysis of the business scenario, which leads to a decision on the message exchange patterns for each AIA service artifact.

For more information about message exchange pattern decisions, see Section 13.2.3, "Establishing the MEP of a New Process EBS", Section 14.3, "Identifying the MEP", and Section 18.2.2, "How to Identify the Message Pattern for an EBF".

The tasks needed to enable the Oracle AIA service artifacts to participate in various message exchange patterns are discussed in the chapters detailing the development of these artifacts.

To design an Integration Flow:

1. Analyze the participating Application Business Messages (ABMs) and map them to the EBM.

   For more information, see "Understanding EBOs and EBMs" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*.

2. Identify the EBSs.

   See Chapter 13, "Designing and Developing Enterprise Business Services."

3. Identify the EBFs.

See Chapter 18, "Designing and Constructing Enterprise Business Flows."

4. Identify the ABCSs.

   See Chapter 14, "Designing Application Business Connector Services."

5. Identify and decide on the participating application connectivity methodology.

   See Chapter 13, "Designing and Developing Enterprise Business Services."

   For more information, see "Understanding Interaction Patterns" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*.

6. Identify and decide on the security model.

   See Chapter 28, "Working with Security."

   For more information, see "Understanding Security" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

7. Identify and decide on the performance and scalability needs.

8. Identify and decide on the deployment strategy.

An Oracle AIA Integrating Scenario is a logical collection of AIA service artifacts, including:

- EBSs
- EBFs
- ABCSs

Since an AIA service artifact can be part of multiple AIA integration flows, go through the Oracle Enterprise Repository and identify any service artifact that can be reused. The AIA service artifacts are built with reusability in mind.

For each of the artifacts, follow these reusability guidelines:

- Reusability guidelines for EBSs
- Reusability guidelines for EBFs
- Reusability guidelines for ABCSs

To develop an AIA Integration Flow:

1. Identify and create the EBS, if needed.

   See Chapter 13, "Designing and Developing Enterprise Business Services."

2. Enable the participating applications.

   See Section 2.4.5, "Enabling and Registering Participating Applications".

3. Construct the ABCS.

   See Chapter 15, "Constructing the ABCS."

4. Construct the EBF.

   See Chapter 18, "Designing and Constructing Enterprise Business Flows."

   For more information about standard naming conventions, see Chapter 31, "Oracle AIA Naming Standards for AIA Development."

2.4.3 Identifying and Creating the EBS

Each EBO has an EBS to expose the "Create, Read, Update, Delete" (CRUD) operations and any other supporting operations. Each action present in the associated EBM is implemented as a service operation in EBS. Creation of the EBS and the implementation of all of the service operations is a critical task in the implementation of the end-to-end integration flow.

> **Note:** The operations in the entity EBS should only act on the relevant business object and not any other business object. Operations that act on multiple business object should reside in the process EBS.

When the EBS exists, check whether all of the actions necessary for acting on the EBO pertaining to this Integration Flow were implemented as service operations. If they were not, change the existing EBS to add the additional service operations.

This procedure lists the high-level tasks to construct an entity based EBS. Chapter 13, "Designing and Developing Enterprise Business Services" provides complete information on how to complete each of these tasks:

To construct an entity-based EBS:

1. Define and create the contract for the EBS.
2. Construct the EBS.
3. Register relevant provider services for each of the operations.
4. Add routing rules for new or existing operations.
5. Enable each of the service operations for the CAVS.

For more information, see Chapter 13, "Designing and Developing Enterprise Business Services"

2.4.4 Constructing the ABCSs

For more information, see Chapter 15, "Constructing the ABCS" and Chapter 16, "Completing ABCS Development."

2.4.5 Enabling and Registering Participating Applications

This section discusses the steps required to enable the participating applications in your AIA Foundation Pack ecosystem. It also discusses the options available for managing participating applications to the AIA registry.

2.4.5.1 Enabling Participating Applications

To enable participating applications:

1. Identify the service APIs that must be invoked.
2. Provide the WSDL to the participating applications.
3. Construct adapter services, if required.

2.4.5.2 Managing the Oracle AIA System Registry

This section discusses how to manage the Oracle Application Integration Architecture (AIA) system registry.

2.4.5.2.1 Understanding the Oracle AIA System Registry

The purpose of the system registry is to identify and capture information about the requester and provider systems that are participating in your Oracle AIA ecosystem. The system registry captures system attributes that are relatively static.

While the system registry for your Oracle AIA ecosystem is initially loaded during the Oracle AIA installation process, you can use the user interface to manage information in the registry.

A benefit of storing a system registry is that certain system attributes can be easily defaulted to enterprise business message headers. Requester and target systems may have multiple instances, each with different attributes. The system registry enables you to capture attributes for each instance so that they can be identified in subsequent processing.

Foundation Pack provides two ways to manage data in your system registry:

- Systems page
- SystemRegistration.xml configuration file

2.4.5.2.2 How to Manage the System Registry Using the Systems Page

Goal:

Use the Systems page to manage the participating applications stored in the Oracle AIA system registry.

Actor:

System administrator (AIAApplicationUser role)

To manage your Oracle AIA system registry using the System page:

1. Access the AIA Home Page. Click **Go** in the **Setup** area.
2. Select the **Systems** tab as shown in Figure 2–6.

Figure 2–6 Systems page (1 of 2)

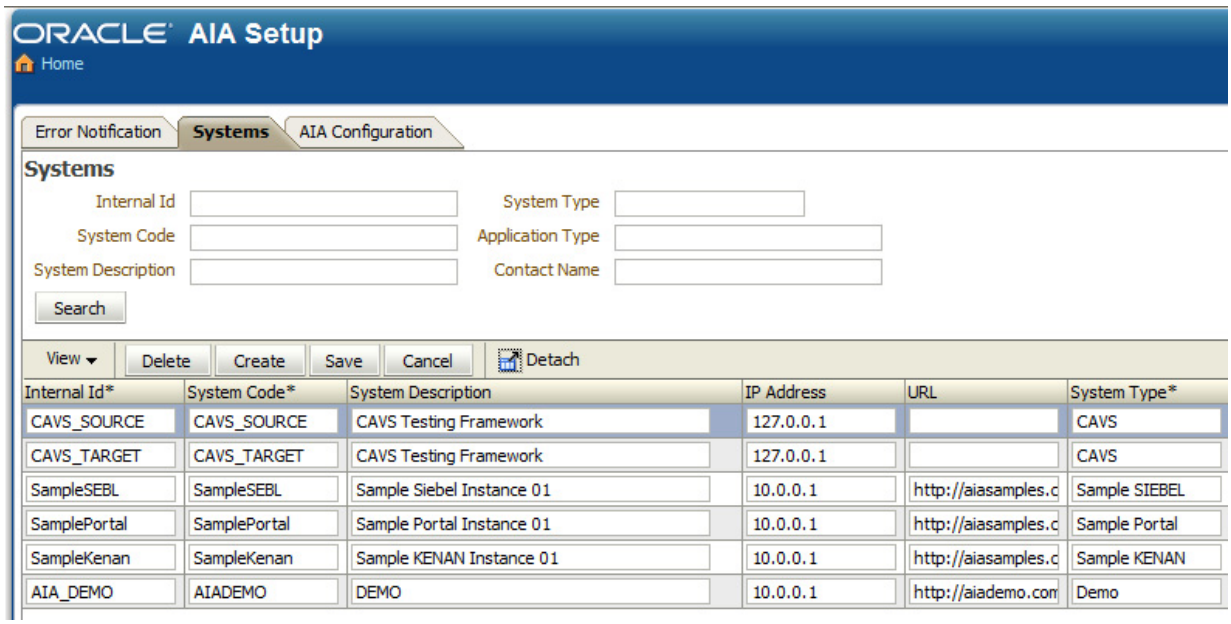
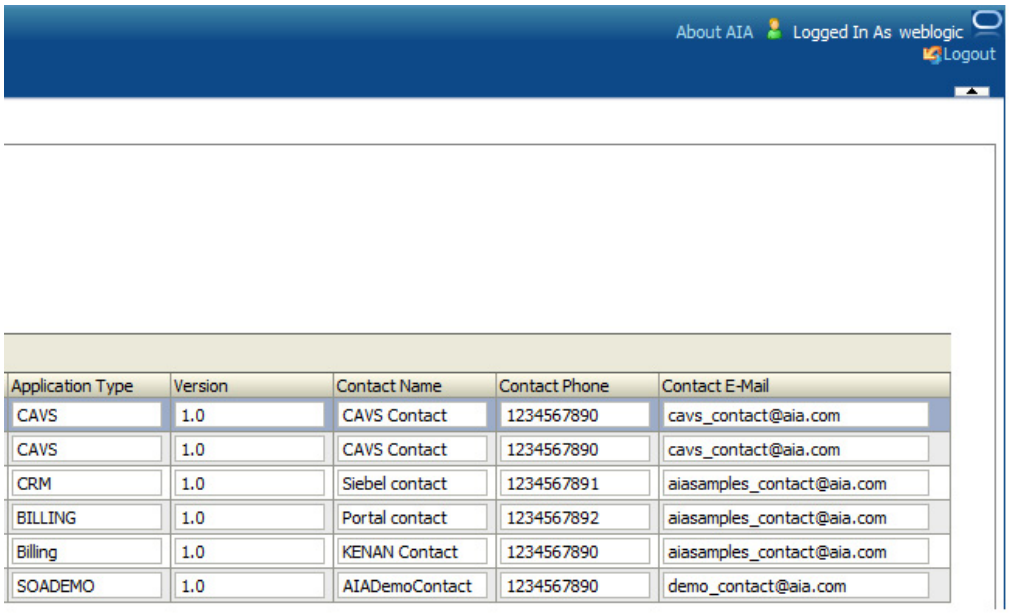


Figure 2–7 Systems page (2 of 2)



3. Enter your search criteria and click **Search** to execute a search for a particular participating application system.
4. Use the page elements listed in Table 2–5 to define participating application system registry entries.

Table 2–5 Systems Page Elements

Page Element	Description
Delete	Select a system row and click **Delete** to execute the deletion.
Create	Click to add a system row that you can use to add a participating application to the system registry.
Save	Click to save all entries on the page.
Internal Id	System instance name assigned by an implementer, for example `ORCL 01`. This value would be used in transformation and domain value map column names to indicate a system instance. For example, the value can be used in transformations to identify the requester or provider system.
System Code	This is a required value. This value populated to the Enterprise Business Message (EBM) header.
System Description	Long description of the requester or provider system instance identified in the **System Code** field. This value populated to the EBM header.
IP Address	IP address of the participating application by which one can access the participating application endpoint. This value is populated to the EBM Header.
URL	This is the host name and port combination specified in the URL format by which one can access the participating application. Typically, this is in the form of `http://hostname:port/`. This value is populated to the EBM Header. This URL is not used by process integrations to access the participating application's web services, but rather this value is stored for informational purposes. Process integration developers determine which URL to use as the entry point for a participating application.

Table 2–5 (Cont.) Systems Page Elements

Page Element	Description
System Type	Provides the system type. For example, `Oracle EBIZ`. This is a required value.
Application Type	Application being run within the specified system. For example, a system type of `Oracle EBIZ` may have an application type value of `FMS`. This value populated to the EBM header.
Version	Provides the version of the application being run by the system. This value populated to the EBM header.
Contact Name	Provides the name of the contact responsible for the system. This value populated to the EBM header.
Contact Phone	Provides the phone number of the contact responsible for the system. This value populated to the EBM header
Contact E-Mail	Provides the email address of the contact responsible for the system. This value is populated to the EBM header.

2.4.5.2.3 How to Manage the System Registry Using the SystemRegistration.xml Configuration File

Goal:

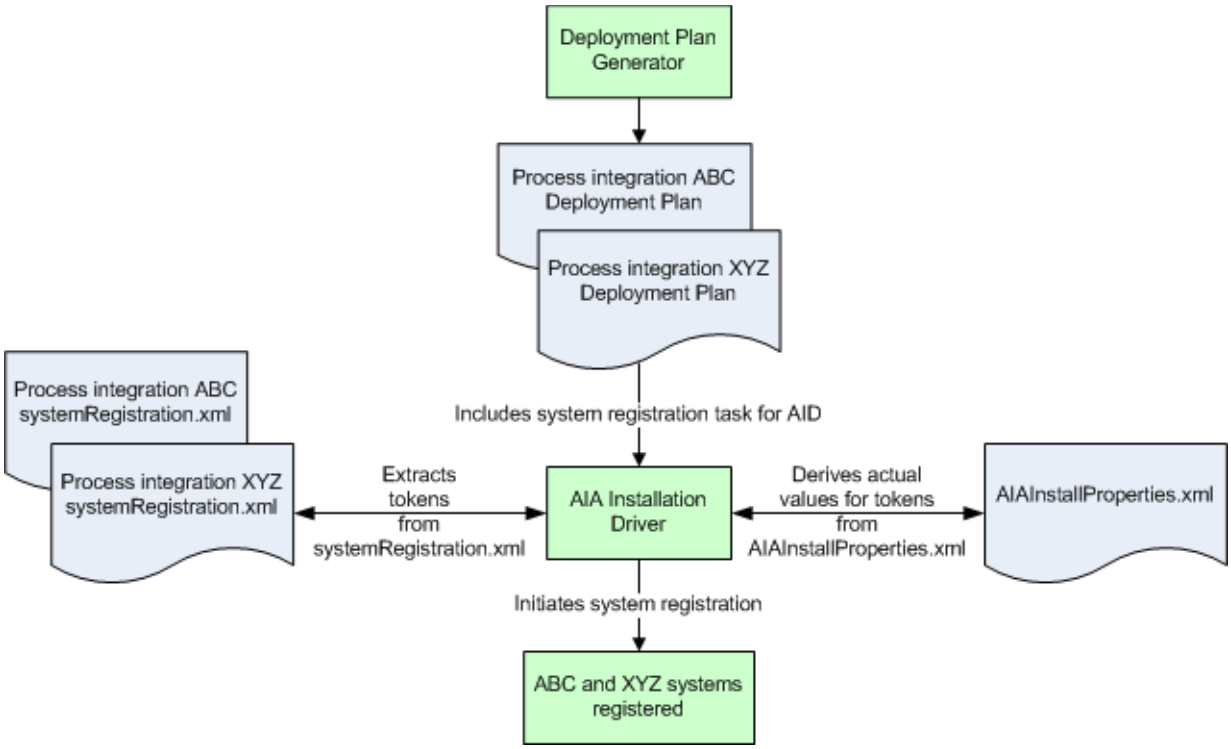
Create a systemRegistration.xml configuration file for a Project Lifecycle Workbench process integration project. Create one systemRegistration.xml file per project. The systemRegistration.xml file is used by the Deployment Plan Generator and AIA Deployment Driver to populate the participating applications used by the process integration to the Oracle AIA system registry.

This is a progammatic approach to completing the same task that can be completed using the Systems page.

For more information about the Systems page, see Section 2.4.5.2.2, "How to Manage the System Registry Using the Systems Page".

Figure 2–8 illustrates how the values defined in systemRegistration.xml files for each process integration are used to populate the Oracle AIA system registry.

Figure 2–8 Flow of System Registration Data



Role:

Integration developer

To manage your Oracle AIA system registry using the SystemRegistration.xml configuration file:

1. Create a `systemRegistration.xml` file for your process integration project.
2. To add participating application values to the system registry, use the `<create>` element provided in the sample XML structure shown in Example 2–1.

Example 2–1 AIASystemRegistration.xml

```
<AIASystemRegistration>
   <create>
   MERGE INTO AIA_SYSTEMS sysreg USING dual on (dual.dummy is
 not null and sysreg.SYSTEM_CODE='SampleSEBLDPT003' ) WHEN NOT
 MATCHED THEN INSERT (SYSTEM_ID,SYSTEM_INTERNAL_ID,SYSTEM_CODE,
SYSTEM_DESC,SYSTEM_IP_ADDR,SYSTEM_URL,SYSTEM_TYPE,APPLICATION_
TYPE,APPLICATION_VERSION,CONTACT_NAME,CONTACT_PHONE,CONTACT_
EMAIL) VALUES (AIA_SYSTEMS_S.nextval,'SampleSEBLDPT003','SampleSEBLDPT003',
'Sample Siebel Instance01','10.0.0.1','http://
${participatingapplications.SampleSiebel.server.
soaserverhostname}:${participatingapplications.SampleSiebel.
server.soaserverport}
/ecommunications_enu'
,'Sample SIEBEL','CRM','${pips.BaseSample.version}
','Siebel contact','1234567891','aiasamples_contact@aia.com')
      /
   MERGE INTO AIA_SYSTEMS sysreg USING dual on (dual.dummy is
 not null and sysreg.SYSTEM_CODE='SamplePortalDPT003' )
WHEN NOT MATCHED THEN INSERT (SYSTEM_ID,SYSTEM_INTERNAL_
```

```
ID,SYSTEM_CODE,SYSTEM_DESC,SYSTEM_IP_ADDR,SYSTEM_URL,SYSTEM_
TYPE,APPLICATION_TYPE,APPLICATION_VERSION,CONTACT_NAME,CONTACT_
PHONE,CONTACT_EMAIL) VALUES (AIA_SYSTEMSS.nextval,
'SamplePortalDPT003','SamplePortalDPT003',
'Sample Portal Instance 01','10.0.0.1',
'http://${participatingapplications.SamplePortal.server.
soaserverhostname}:${participatingapplications.SamplePortal.
server.soaserverport}
','Sample Portal','BILLING','${pips.BaseSample.version}'
,'Portal contact','1234567892','aiasamples_contact@aia.com')
    /
   </create>
</AIASystemRegistration>
```

The tokens you create in the `<create>` element should be based on the structure of the `AIAInstallProperties.xml` file.

For example, the following token, `${participatingapplications.SamplePortal.server.soaserverhostname}`, was derived from the structure in the `AIAInstallProperties.xml` file shown in Example 2–2:

Example 2–2 AIAInstallProperties.xml

```
<properties>
   <participatingapplications>
      <SamplePortal>
         <server>
            <soaserverhostname>adc2180948.us.oracle.com
            </soaserverhostname>
         </server>
      ...
   </participatingapplications>
</properties>
```

Actual values for the tokens in bold are derived from `$AIA_INSTANCE/config/AIAInstallProperties.xml` during process integration installation using a deployment plan.

3. To delete participating application values from the AIA application registry, use the `<delete>` element provided in the sample XML structure shown in Example 2–3 and replace the system code values in bold with the system codes of the participating applications.

Example 2–3 <delete> element in AIASystemRegistration.xml

```
<AIASystemRegistration>
   <delete>
      delete from AIA_SYSTEMS where SYSTEM_CODE='SampleSEBLDPT005'
       /
      delete from AIA_SYSTEMS where SYSTEM_CODE='SamplePortalDPT005'
       /
   </delete>
</AIASystemRegistration>
```

4. Save the `systemRegistration.xml` file to `$AIA_HOME/pips/<projectCode>/DatabaseObjects/`.

If you are using the Project Lifecycle Workbench in your development methodology, the <projectCode> folder name is the project code value assigned to your project in Project Lifecycle Workbench.

If you are not using the Project Lifecycle Workbench, assign your own project code value to the folder name.

The Deployment Plan Generator always generates the system registration task in the deployment plan. The system registration task extracts configuration information from systemRegistration.xml, replaces tokens with actual values from AIAInstallProperties.xml, and executes an SQL statement to populate the system registry.

If no systemRegistration.xml file is present for the Deployment Plan Generator to leverage, manually remove the system registration task from the deployment plan, or the deployment fails.

Example 2–4 shows a generated deployment plan that includes the system registration task to register systems for projectCode ABCProcessInt:

Example 2–4 System Registration Task in Generated Deployment Plan

```
<DeploymentPlan>
   <Configurations>
      ...
      <SystemRegistration database="fp.db.aia" resourceFileName="$AIA_
HOME/pips/ABCProcessInt/DatabaseObjects/systemRegistration.xml" delimiter="/"
action="create"/>
      ...
   </Configurations>
</DeploymentPlan>
```

The AIA Deployment Driver reads the deployment plan, replaces the tokens from the systemRegistration.xml with actual values from AIAInstallProperties.xml, and executes the system registration SQL statement to populate the system registry.

Example 2–5 shows the SQL content extracted by the AIA Deployment Driver:

Example 2–5 SQL Content Extracted by AIA Deployment Driver

```
MERGE INTO AIA_SYSTEMS sysreg
USING dual
on (dual.dummy is not null and sysreg.SYSTEM_CODE='SampleSEBL' )
WHEN NOT MATCHED THEN INSERT
(SYSTEM_ID,SYSTEM_INTERNAL_ID,SYSTEM_CODE,SYSTEM_DESC,SYSTEM_IP_ADDR,SYSTEM_
URL,SYSTEM_TYPE,APPLICATION_TYPE,APPLICATION_VERSION,CONTACT_NAME,CONTACT_
PHONE,CONTACT_EMAIL)
VALUES (AIA_SYSTEMS_S.nextval,'Siebel','SampleSEBL','Sample Siebel Instance
01','10.0.0.1','http://siebel.us.oracle.com:7001/ecommunications_enu',
SIEBEL','CRM','1.0, 'Siebel contact','1234567891','Siebelcontact@Siebel.com')
/
```

If you are deploying multiple Project Lifecycle Workbench process integration projects, multiple systemRegistration.xml files get added. Check whether participating applications are added to system registry entries before deploying them. Duplicate requests cause an SQL exception due to primary key and unique value constraints enabled on the system registry table.

Check them by including the text in bold in the SQL content above. If a participating application value is found to be a duplicate, it is not created. If it is unique, it is created.

The AIA Deployment Driver extracted this example SQL content based on replacing tokens from systemRegistration.xml with Example 2–6 data provided in `$AIA_INSTANCE/config/AIAInstallProperties.xml`:

Example 2–6 Sample SQL Content Extracted by AIA Deployment Driver

```
<?xml version="1.0" encoding="UTF-8"?>
   <properties>
      <participatingapplications>
         <siebel>
             <http>
                <host>siebel.us.oracle.com</host>
                <port>7001</port>
             </http>
             <internal>
                <id>Siebel</id>
             </internal>
             <version>1.0</version>
          <siebel>
            ...
      </participatingapplications>
   </properties>
```

2.4.6 Identifying and Creating the EBF

Each identified EBF has a corresponding business process EBS. The operations within this EBS are the entry points to EBFs.

For more information, see Chapter 18, "Designing and Constructing Enterprise Business Flows."

2.5 Testing an Oracle AIA Integration Flow

The CAVS supports end-to-end testing by stubbing out the applications.

For more information about CAVS, see "Working with the CAVS." in the *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

3

Working with Project Lifecycle Workbench

This chapter provides an overview of Project Lifecycle Workbench and discusses how to add lookup values for workbench, how to work with lifecycle workbench projects, and lifecycle workbench service solution components.

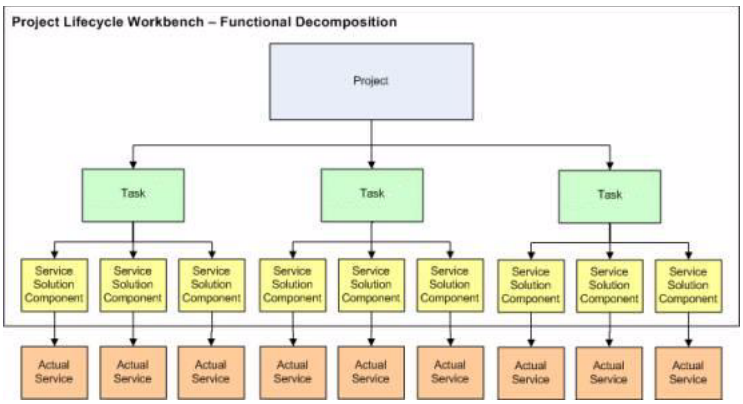
This chapter includes the following sections:

3.1 Introduction to Project Lifecycle Workbench

As a part of the Oracle Application Integration Architecture (AIA) development methodology and lifecycle, functional designs are created to specify requirements and functionality that must be implemented for an integration project.

Users, such as solution architects and functional product managers, can use Project Lifecycle Workbench to perform functional decompositions to break down overall projects into tasks, each of which is implemented by a collection of composites and services, as shown in Figure 3–1.

Figure 3–1 Functional Decomposition of a Project into Actual Implemented Services



Project Lifecycle Workbench complements any existing design and analysis process with its various modeling and functional analysis mechanisms. The functional

decomposition, the final output of the design and analysis process, is captured and persisted in Project Lifecycle Workbench.

The primary building blocks that enable functional decomposition in Project Lifecycle Workbench are:

- **Project**

  A project is a named effort to deliver a solution conforming to the AIA methodology and guidelines. The components of a project must be packaged and deployed on a middleware solution.

- **Task**

  A task represents a reusable unit of work that is arrived at based on analysis of reference process models. These could be business processes, business activities, or tasks and are loosely added to a project in the Project Lifecycle Workbench as tasks.

- **Service Solution Component**

  A service solution component is a chunk of functionality within the scope of a task that is implemented as an AIA service artifact. For example, a service solution component corresponds to an Application Business Connector Service (ABCS), Enterprise Business Service (EBS), or Enterprise Business Flow (EBF). A service solution component conveys the functionality that the AIA service artifact must fulfill.

For more information about the overall Oracle AIA Project Lifecycle, see Chapter 2, "Building AIA Integration Flows."

3.2 Adding Project Lifecycle Workbench Lookup Values

This section includes the following topic: Chapter 3.2.1, "How to Add Lookup Values."

3.2.1 How to Add Lookup Values

Objective

Project Lifecycle Workbench is delivered with the following sets of lookup values:

- Industry code, for use with projects
- Status, for use with projects
- Product code, for use with service solution components
- Service type, for use with service solution components
- Scope, for use with tasks

These lookup values are available for assignment to projects, tasks, or service solution components. If a required lookup value is not available, use this procedure to add it.

For more information about working with projects, see Section 3.3, "Working with Project Lifecycle Workbench Projects."

For more information about working with service solution components, see Section 3.4, "Working with Project Lifecycle Workbench Service Solution Components."

Prerequisites and Recommendations

Obtain confirmation from the Project Lifecycle Workbench user community that the lookup value to be added is necessary and valid.

Actor

System administrator

To add your own lookup values:

1. Run an SQL statement, as shown in Example 3–1, to insert the required lookup value into the AIA_LOOKUPS_B table.

Example 3–1 SQL Statement Used to Insert Lookup Values into AIA_LOOKUPS_B

```
INSERT INTO AIA_LOOKUPS_B (LOOKUP_TYPE, LOOKUP_CODE, PARENT_LOOKUP_CODE,
CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATE_DATE, OBJECT_
VERSION_NUMBER)
SELECT '<lookup value type>', '<lookup value code>', null,
SYS_CONTEXT('USERENV', 'AUTHENTICATED_IDENTITY'), sysdate, SYS_CONTEXT
('USERENV', 'AUTHENTICATED_IDENTITY'), sysdate, 1
FROM DUAL
WHERE NOT EXISTS (SELECT 1
FROM AIA_LOOKUPS_B
WHERE LOOKUP_TYPE = '<lookup value type>'
AND LOOKUP_CODE = '<lookup value code>');
```

   a. Replace the bold **<lookup value type>** variable with the type of lookup code that you want to use: INDUSTRY, PROJECT_STATUS, SERVICE_TYPE, TASK_SCOPE, or PRODUCT_CODE.

   b. Replace the bold **<lookup value code>** variable with the lookup value code that you want to add. This value has a 30-character limit.

2. Run an SQL statement, as shown in Example 3–2, to insert the required lookup value code, meaning, and description into the AIA_LOOKUPS_TL table. After running this SQL statement, a message displays to notify you of the number of rows that were created. If this value is correct, enter *commit*.

Example 3–2 SQL Statement Used to Insert Lookup Values into AIA_LOOKUPS_TL

```
INSERT INTO AIA_LOOKUPS_TL (LOOKUP_TYPE, LOOKUP_CODE, MEANING, DESCRIPTION,
LANGUAGE, SOURCE_LANG, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATE
_DATE, OBJECT_VERSION_NUMBER)
SELECT '<lookup value type>', '<lookup value code>',
'<lookup value in UI>', '<lookup value description>', SYS_CONTEXT
('USERENV', 'LANG'), SYS_CONTEXT('USERENV', 'LANG'), SYS_CONTEXT('USERENV',
 'AUTHENTICATED_IDENTITY'), sysdate, SYS_CONTEXT('USERENV', 'AUTHENTICATED_
IDENTITY'), sysdate, 1
FROM DUAL
WHERE NOT EXISTS (SELECT 1
FROM AIA_LOOKUPS_TL
WHERE LOOKUP_TYPE = '<lookup value type>'
AND LOOKUP_CODE = '<lookup value code>');
```

   a. Replace the bold **<lookup value type>** variable with the type of lookup code that you used in step 1.

   b. Replace the bold **<lookup value code>** variable with the lookup value code that you added in step 1. This value has a 30-character limit.

c. Replace the bold **<lookup value in UI>** variable with the actual lookup value that you want to appear in the UI. This value has an 80-character limit.

d. Replace the bold **<lookup value description>** variable with a description of the lookup value. This is optional. This value has a 240-character limit.

3. Project Lifecycle Workbench lookup values are configured with the following language settings: LANGUAGE = US and SOURCE_LANG = US. By default, an AIA Installer-based Foundation Pack installation has the supporting US language installed.

   If your implementation uses a value other than LANGUAGE = US, run an SQL statement, as shown in Example 3–3, to populate the AIA_LOOKUPS_TL table with the internationalized text for the lookup value that you added in step 2.

Example 3–3 SQL Statement to Insert Internationalized Text for Lookup Values into AIA_LOOKUPS_TL

```
INSERT INTO AIA_LOOKUPS_TL (LOOKUP_TYPE, LOOKUP_CODE, MEANING, DESCRIPTION,
LANGUAGE, SOURCE_LANG, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATE_
DATE, OBJECT_VERSION_NUMBER)
SELECT <lookup value type>', '<industry code>',
'<industry lookup value in UI>', '<industry description>', <language code>
, 'US', SYS_CONTEXT('USERENV'
, 'AUTHENTICATED_IDENTITY'), sysdate, SYS_CONTEXT('USERENV', 'AUTHENTICATED_
IDENTITY'), sysdate, 1
FROM DUAL
WHERE NOT EXISTS (SELECT 1
FROM AIA_LOOKUPS_TL
WHERE LOOKUP_TYPE = '<lookup value type>'
AND LANGUAGE = '<language code>'
AND LOOKUP_CODE = '<industry code>');
```

a. Replace the bold **<language code>** values with the actual language code.

   For more information, see "Setting Up a Globalization Support Environment" in *Oracle Database Globalization Support Guide*.

b. Replace the bold **<lookup value type>** variable with the type of lookup code that you used in step 1.

c. Replace the bold **<industry code>** variable with the lookup value code that you added in step 1. This value has a 30-character limit.

d. Replace the bold **<industry lookup value in UI>** variable with the internationalized industry value that you want to appear in the UI. This value has an 80-character limit.

e. Replace the bold **<industry description>** variable with an internationalized description of the industry. This is optional. This value has a 240-character limit.

3.3 Working with Project Lifecycle Workbench Projects

This section includes the following topics:

- Section 3.3.1, "How to Define Project Lifecycle Workbench Projects"
- Section 3.3.2, "How to Update Project Lifecycle Workbench Projects"
- Section 3.3.3, "How to Access Project Lifecycle Workbench Projects"

- Section 3.3.4, "How to Edit a Locked Project"
- Section 3.3.5, "How to Delete Project Lifecycle Workbench Projects"

3.3.1 How to Define Project Lifecycle Workbench Projects

Objective

Define a project that you want to functionally decompose into tasks and service solution components.

Prerequisites and Recommendations

Ensure that all required lookup values are available. If not, contact your system administrator to add required values.

For more information, see Section 3.2.1, "How to Add Lookup Values."

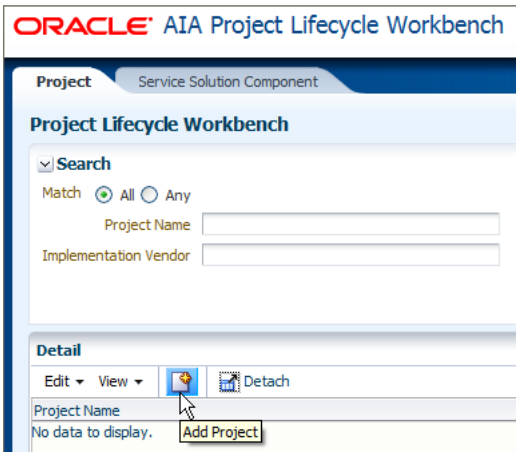
Actors

- Functional product manager
- Solution architect

To define a project:

1. Access the AIA Home page. Click **Go** in the Project Lifecycle Workbench area. Select the **Project** tab. Click the **Add Project** button, as shown in Figure 3–2.

Figure 3–2 Add Project Button



2. The Add Project page displays.

   In the **Project Name** field, enter a descriptive project name.

   In the **Project Code** field, enter a code (one word with no spaces) for your project. The project code value appears as a root-level attribute of the bill of material generated for the project. Per Pre-Built Integrations development best practices, this project code value must be used by development as the project file directory name in their source control system for downstream automation features, such as the Deployment Plan Generator, to work as designed.

   In the **Implementation Vendor** field, enter the vendor that is responsible for implementing the project, whether it is Oracle or another vendor.

In the **Industry** drop-down list box, select the industry that is targeted by the project. If the required industry value is not available, users should be able to contact their system administrator to have an approved industry added.

For more information about how to add industry codes lookup values, see Section 3.2.1, "How to Add Lookup Values."

In the **Type** field, select the type of project that you are defining.

In the **Status** field, select the project status.

In the **Version** field, enter the AIA release to which this project belongs.

Upon adding a project, the system assigns a Project UID (unique ID) value. Upon saving the project, the value is stored in the AIA_OER_PROCESSES table. This ID can be used in downstream processing. For example, you can choose to export BOM and functional decomposition data by project unique ID value.

For more information about exporting BOM and functional decomposition data, see Section 7.3, "How to Export Seed Data."

3. Click **Save** to save your work and continue entering more information about the project.
4. To enter a project description, click the **Description** link. In the text field, enter information about the project, such as the features and requirements of the project.
5. To enter project assumptions, click the **Assumption** link. In the text field, describe any assumptions involved in implementing the project.
6. To define the tasks into which you want to decompose the project, click the **Tasks** link.

   To add a task, click the **Add Task** button. Enter a task name. Expand the task to display scope options and to enter a description.

   Use the **In Scope** and **Out of Scope** options to indicate whether the task is in scope to be implemented as a part of the project. As development of this project progresses, a task may move in and out of scope. Only tasks with the In Scope value selected are included in a generated project bill of material (BOM).

   For more information, see Chapter 6, "Working with Project Lifecycle Workbench Bills of Material."

   To delete a task, click the **Delete Task** button.
7. To provide links to supporting project documentation, click the **Document Links** link.

   To add a document link, click the **Add Document Link** button. For each document, enter a document name and URL location.

   To delete a document link, click the **Delete Document Link** button.
8. Click **Save and Return** to save your work and access the Project page.

   For more information about the Project page, see Section 3.3.3, "How to Access Project Lifecycle Workbench Projects."

3.3.2 How to Update Project Lifecycle Workbench Projects

Objective

Update an existing project for which a BOM has not been generated.

For information about how to update a project for which a BOM has been generated, see Section 3.3.4, "How to Edit a Locked Project."

Actors

- Functional product manager
- Solution architect

To update a project:

1. Access the AIA Home Page. Click **Go** in the Project Lifecycle Workbench area. Select the **Project** tab.
2. Locate the project and click the **Update Project** button.
3. Make necessary changes in the **Update Project** page.

For more information about the elements available on the Update Project page, see Section 3.3.1, "How to Define Project Lifecycle Workbench Projects."

3.3.3 How to Access Project Lifecycle Workbench Projects

Objective

Access project summaries, access pages that enable you to create projects, update projects, and generate BOMs for projects.

Actors

Anyone

To access projects:

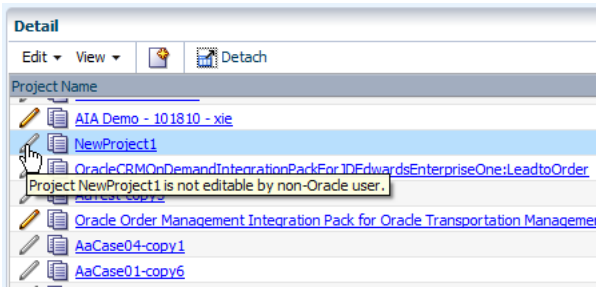
1. Access the AIA Home Page. Click **Go** in the Project Lifecycle Workbench area. Select the **Project** tab.
2. Use the query criteria in the **Search** area to locate the project with which you want to work.
3. The **Detail** area displays project summaries and provides access to the following details and functionality. Hover over a project name link to display project details.
   - Click the **Add Project** button to access the Add Project page.

     For more information, see Section 3.3.1, "How to Define Project Lifecycle Workbench Projects."
   - Click the **Update Project** button to access the Update Project page.

     For more information, see Section 3.3.2, "How to Update Project Lifecycle Workbench Projects."

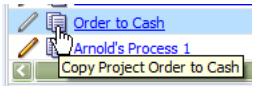
     The **Update Project** button is inactive if the Implementation Vendor value for the project is **Oracle** and you are logged into the system using a non-Oracle user ID, as shown in Figure 3–3.

Figure 3–3 Inactive Update Project Button



- To use an Oracle-delivered project as the basis for a new custom project while using a non-Oracle user ID, copy the project. To copy the project, click the **Copy Project** button for the project, as shown in Figure 3–4.

Figure 3–4 Copy Project Button



The Copy Project page displays.

Update project values to reflect the characteristics of your new project. Provide a unique project name in the **Project Name** field.

All project data is copied to the new project. You can revise the functional decomposition data by adding and dropping tasks and service solution components or by directly adding, dropping, and revising associated composites in the replicated project's BOM.

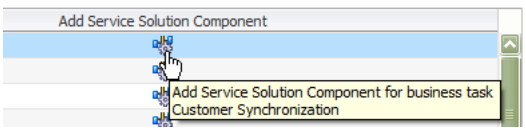
After the project data is copied, the **Edit** button becomes active for the project on the Project tab. This functionality ensures that customized projects are not affected by upgrades to Oracle-delivered projects.

- The Bill Of Material column provides access to controls that enable you to work with BOMs.

  For information about generating, editing, and viewing BOMs, see Chapter 6, "Working with Project Lifecycle Workbench Bills of Material."

4. The Tasks & Document Links area provides access to the following details and functionality.
   - Select the **Tasks** tab to expand a task and display the task's description and associated service solution components.
   - Click the **Add Service Solution Component** button to access the Create Service Solution Component page, as shown in Figure 3–5.

Figure 3–5 Add Service Solution Component Button



For more information, see Section 3.4.1, "How to Define Project Lifecycle Workbench Service Solution Components."

- Click the **Document Links** tab to access documents associated with the task.

3.3.4 How to Edit a Locked Project

After a BOM has been generated for a project, the project is locked and cannot be edited without manual intervention.

To edit a locked project, make a copy of the locked project, delete the locked project, edit the copied project as necessary, and regenerate the BOM.

Data shown in screenshots is for illustrative purposes only.

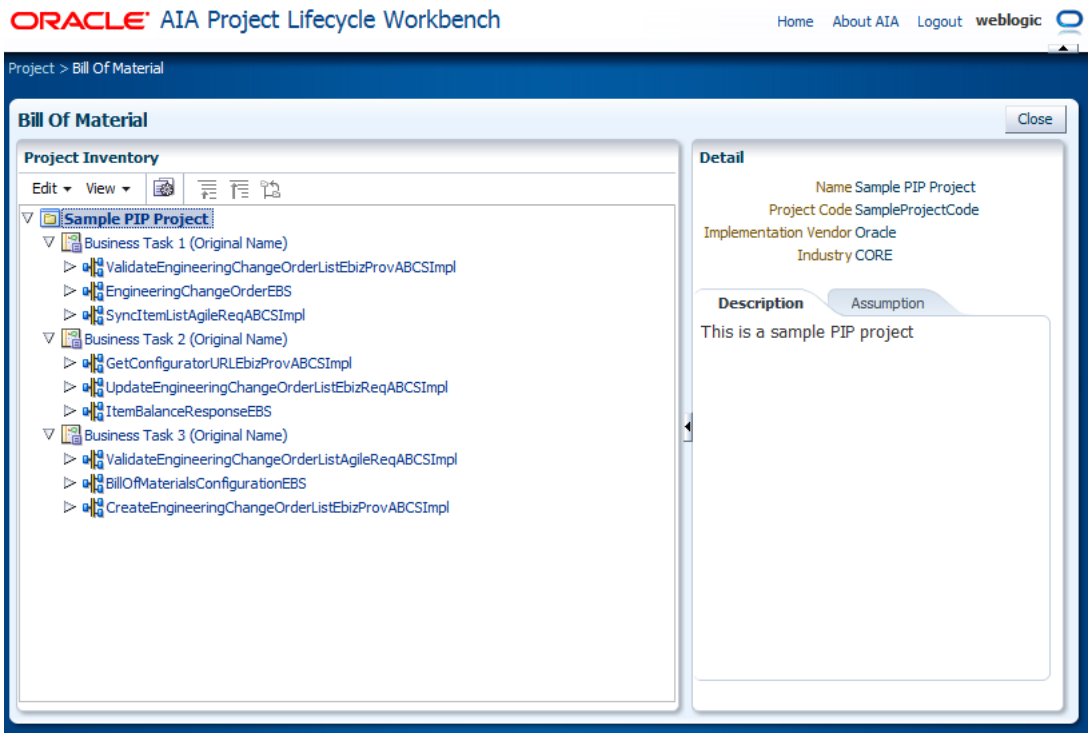
To edit a locked project:

1. Access the AIA Home page. Click **Go** in the Project Lifecycle Workbench area. Select the **Project** tab. Locate the project that requires editing.

   The presence of the Edit Bill of Material link tells us that the BOM has been generated for this project. Before BOM generation, the Generate Bill of Material link would have been displayed instead.

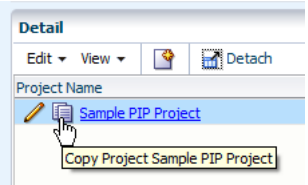
2. Click the **Edit Bill of Material** link to access the BOM. In taking a closer look at the BOM, notice that some harvested composites have been associated with the project's tasks, as shown in Figure 3–6. In this example, each task contains 3 composites, resulting in a total of 9 composites in the BOM.

Figure 3–6 Nine Composites in the BOM Generated from the Project



3. Create a copy of the project that requires editing by selecting the project and clicking the **Copy Project** button, as shown in Figure 3–7.

Figure 3–7 Copy the Project



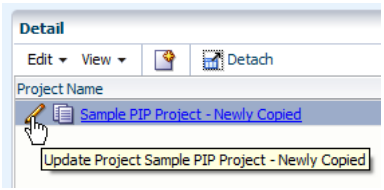
When you create a copy of the project, you are prompted to rename the project. Change the name in Project Name field.

4. Before making any revisions on the copied project, delete the original project.

   For information about deleting projects, see Section 3.3.5, "How to Delete Project Lifecycle Workbench Projects."

5. Edit the copied project as necessary, including defining its final project name. Select the copied project to be edited and click the Update Project button, as shown in Figure 3–8.

Figure 3–8 Select to Update the Copied Project



You can edit every field, such as the project name, task names, various descriptions, assumptions, and add and remove tasks.

You may also rename tasks according to Pre-Built Integrations naming standards.

6. Regenerate the BOM for the copied and edited project by clicking the **Generate Bill Of Material** link.

   When prompted to generate the BOM, click **Generate**.

7. Oracle AIA recommends that you also:

   - Export the regenerated BOM.xml from the Project Lifecycle Workbench.
   - Regenerate <ProjectCode>DeploymentPlan.xml.
   - Test the regenerated <ProjectCode>DeploymentPlan.xml.
   - Source control the regenerated <ProjectCode>DeploymentPlan.xml back to the ADE.
   - Repackage <ProjectCode>DeploymentPlan.xml.

3.3.5 How to Delete Project Lifecycle Workbench Projects

When you delete a project, the following aspects of the project are deleted:

- The project itself.
- All tasks associated with the project.
- All service solution components associations with deleted tasks.

You delete a project by running the DeleteProject.sql script located in $AIA_HOME/Infrastructure/LifeCycle/sql directory against the Project Lifecycle Workbench database instance in which the project to be deleted resides. The SQL script requires the Project Unique ID of the project to be deleted as input.

To delete a Project Lifecycle Workbench project:

1. Identify the Project Unique ID assigned to the project by hovering over the project on the **Project** tab.

   To access the Project tab, access the AIA Home page. Click **Go** in the Project Lifecycle Workbench area. Select the **Project** tab.

2. Run DeleteProject.sql, located in <FMW_HOME>/AIAHOME/Infrastructure/LifeCycle/sql, against the Project Lifecycle Workbench database instance in which the project to be deleted resides.

3. When prompted, provide the Project Unique ID value.

4. Validate that the project was successfully deleted by checking the database and the Project Lifecycle Workbench.

3.4 Working with Project Lifecycle Workbench Service Solution Components

This section includes the following topics:

- Section 3.4.1, "How to Define Project Lifecycle Workbench Service Solution Components"
- Section 3.4.2, "How to Update Project Lifecycle Workbench Service Solution Components"
- Section 3.4.3, "How to Access Service Solution Components"

3.4.1 How to Define Project Lifecycle Workbench Service Solution Components

Objective

Define the service solution components that constitute a task in a Project Lifecycle Workbench project. Each service solution component that you define should correspond to a single composite or service that is built or reused. These composites and services collectively accomplish the functionality and logic required by their parent task.

Prerequisites and Recommendations

- Ensure that the project and tasks to which you want to assign service solution components exist.

  For more information, see Section 3.3, "Working with Project Lifecycle Workbench Projects."

- Ensure that all required lookup values are available. If not, contact your system administrator to add required values.

  For more information, see Section 3.2.1, "How to Add Lookup Values."

Actors

- Developers

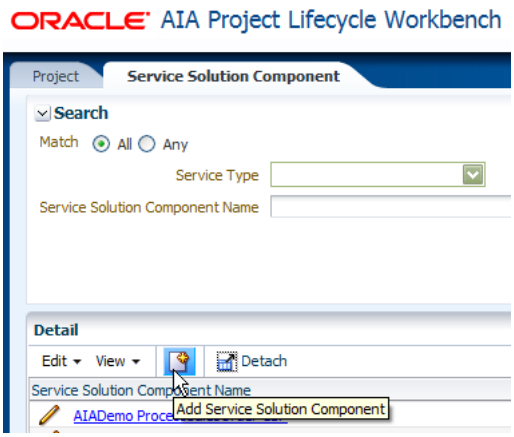
- Functional product managers
- Solution architects

Developers work with functional product managers and solution architects to validate the project design and decomposition. The developers ultimately take on the task of implementing the service solution component as composites

To define service solution components for Project Lifecycle Workbench project tasks:

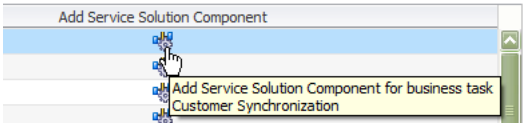
1. Access the Add Service Solution Component page in one of two ways:
   - Access the AIA Home page. Click **Go** in the Project Lifecycle Workbench area. Select the **Service Solution Component** tab. Click the **Add Service Solution Component** button, as shown in Figure 3–9.

Figure 3–9 Add Service Solution Component Button on the Service Solution Component Tab



   - Access the AIA Home page. Click **Go** in the Project Lifecycle Workbench area. Select the **Project** tab. Click the project link for the project for which you want to define service solution components. Tasks assigned to the selected project display on the Tasks tab. Locate the task for which you want to define service solution components. Click the **Add Service Solution Component** button, as shown in Figure 3–10.

Figure 3–10 Add Service Solution Component Button on the Tasks Tab on the Project Tab



   The Add Service Solution Component page displays.

2. Select a **Service Type** value: **Requestor ABCS**, **Provider ABCS**, **Enterprise Business Flow**, **Enterprise Business Service**, **Composite Business Process**, or **Others**.

The actual composite or service that is built from this service solution component definition inherits the service type value. Developers can override this inherited attribute when working in Service Constructor.

For more information, see Chapter 4, "Working with Service Constructor."

For example, selecting a Requestor ABCS service type results in a requester Application Business Connector Service being queued for construction later in the AIA lifecycle flow.

If you select Requestor ABCS or Provider ABCS, an autogenerated globally unique identifier (GUID) is assigned to the service solution component. The composite built for this service solution component is tagged with the GUID when it is created by the Service Constructor. When the implemented composite is harvested for display in the Oracle Enterprise Repository, the GUID on the composite is used to associate the composite back with its originating service solution component in the Project Lifecycle Workbench.

The **Product Code** field displays. Select the product code that corresponds to the participating application for which the service solution composite is being defined. For example, if you are defining a requester service solution component, the product code that you select should correspond to the requesting participating application.

The Deployment Plan Generator uses the product code value in carrying out its logic.

For more information about the Deployment Plan Generator, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

Available product code values are derived from the Application Object Library directory names in your source control system. If the required product code is not available, users should be able to contact their system administrator to have a product code added.

For more information about making product code values available to the Project Lifecycle Workbench, see Section 3.2.1, "How to Add Lookup Values."

Note: For service solution components with Service Type field values set to Provider ABCS or Requestor ABCS, the Service Constructor can function based on the information captured on this page.

For more information, see Chapter 4, "Working with Service Constructor."

3. In the **Name** field, enter a descriptive name for the service solution component. This is not meant to be the actual service implementation name.
4. To associate the service solution component with projects and tasks, click the **Project and Tasks** link.

   If access this page from the Task tab, the task that you have selected to access this page displays relevant project and task values.

   If access this page from the Service Solution Component page, you can associate the service solution component with one or more project and task combinations.
5. To provide a description of the service solution component, click the **Description** link. In the text field, enter information about the service solution component, such as the functionality that it intends to provide and implementation assumptions.

6. To search for and add existing composites that you are qualified to reuse to your service solution component, click the **Reuse Artifacts** link.
7. If you do not know the name of the composite you want to use, you can search for it in the Oracle Enterprise Repository or another solution that your organization may use to store information about existing composites.
   a. If an Oracle Enterprise Repository instance has been defined as a part of your AIA Installer installation, the **Link to OER** button display.

      Click the **Link to OER** button to access the Oracle Enterprise Repository and search for existing AIA interface assets that you can reuse for your integration project. If you find a match in Oracle Enterprise Repository, make a note of its service name value.

   b. If an Oracle Enterprise Repository instance exists, but was not defined as a part of your AIA Installer installation, access it manually to search for the service name value.

      For more information about loading AIA interfaces into Oracle Enterprise Repository, see Section 11.1, "Introduction to Using Oracle Enterprise Repository as the Oracle AIA SOA Repository."

   c. If an Oracle Enterprise Repository instance does not exist, you cannot use it to browse for and research which existing services are available for reuse. However, your organization may have an alternate means of sharing information about existing services. If applicable, you can use these means to search for the service name value.

   Note: Use the service name to search for the existing composite in Project Lifecycle Workbench. These service names likely correspond to composite names in Project Lifecycle Workbench.

   If no composite exists that is appropriate for reuse for your project, proceed to step 8.
8. Click the **Add Service** button to add a new row to the page.

   If you know the name of the composite you want to add, enter its name in the **Composite Name** field.

   To search for the composite, click the **Composite** button to access the Search and Select: Composite Name page, where you can use the service name value you obtained in step 6 to search for the reusable composite in the Project Lifecycle Workbench database. Select the existing composite and click **OK** to add it to the service solution component.

   If no composite exists that is appropriate for reuse for your project, proceed to the next step.
9. Click **Save and Return** to save your work and access the Service Solution Component page.

   For more information about the Service Solution Component page, see Section 3.4.3, "How to Access Service Solution Components."

3.4.2 How to Update Project Lifecycle Workbench Service Solution Components

Objective

Update service solution components.

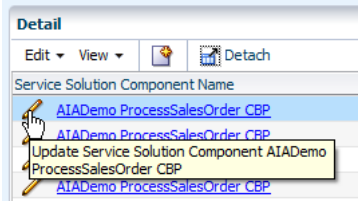
Actors

- Functional product manager
- Developers
- Solution architect

To update service solution components:

1. Access the AIA Home Page. Click **Go** in the Project Lifecycle Workbench area. Select the **Service Solution Component** tab.
2. Locate the service solution component click the **Update Service Solution Component** button, as shown in Figure 3–11.

Figure 3–11 Update Service Solution Component Button



The Update Service Solution Component page display.

For more information about the elements available on the Update Service Solution Component page, see Section 3.4.1, "How to Define Project Lifecycle Workbench Service Solution Components."

3.4.3 How to Access Service Solution Components

Objective

Access service solution component summaries. Access pages that enable you to create and update service solution components.

Actors

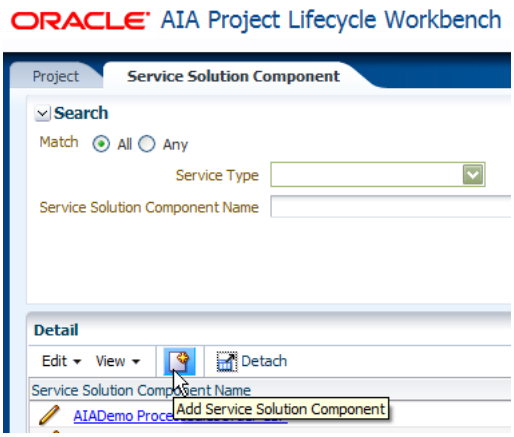
Anyone

To access service solution component summaries:

1. Access the AIA Home Page. Click **Go** in the Project Lifecycle Workbench area. Select the **Service Solution Component** tab.
2. Use the query criteria in the **Search** area to locate the service solution component with which you want to work.
3. The Detail area displays service solution component summaries and provides access to the following details and functionality.

- Click the **Add Service Solution Component** button, as shown in Figure 3–12, to access the Add Service Solution Component page, where you can define service solution components.

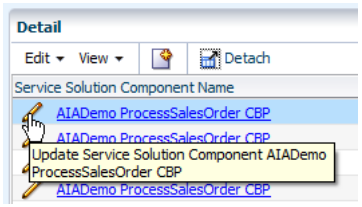
Figure 3–12 Add Service Solution Component Button



For more information, see Section 3.4.1, "How to Define Project Lifecycle Workbench Service Solution Components."

- Click the **Update Service Solution Component** button, as shown in Figure 3–13, to access the Update Service Solution Component page, where you can update a service solution component.

Figure 3–13 Update Service Solution Component Button



For more information, see Section 3.4.2, "How to Update Project Lifecycle Workbench Service Solution Components."

- Hover over a service solution component name link to display component details. Expand a service solution component to display further details.

4

Working with Service Constructor

This chapter provides an overview of Service Constructor and discusses how to use Service Constructor to create new service solution components.

This chapter includes the following sections:

- Section 4.1, "Introducing Service Constructor"
- Section 4.2, "Using Service Constructor to Create New Service Solution Components"

4.1 Introducing Service Constructor

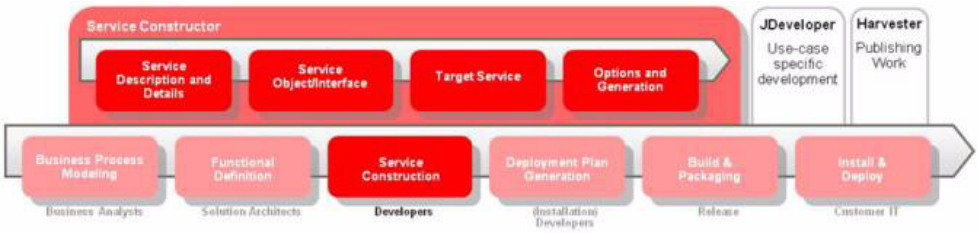
This section provides an overview of Service Constructor and includes the following topic: Section 4.1.1, "Required Software for Using Service Constructor."

Service Constructor is an extension to Oracle JDeveloper that developers can use to easily create new Oracle Application Integration Architecture (AIA) service component projects that conform to the AIA programming models and naming and architectural recommendations. Presently, Service Constructor supports the following service components:

- Requester Application Business Connector Services (ABCSs)
- Provider ABCSs

As a part of the AIA project lifecycle, Service Constructor can be used in the service construction phase, as shown in Figure 4–1.

Figure 4–1 Overview of the Service Construction Phase of the AIA Project Lifecycle



Service Constructor guides the developer through four of the six sub phases of developing an ABCS. The four sub phases are:

- Define service description and details

  Define high-level information about the service solution component and the project and task to which it belongs.

- Define the service object (or interface)

  Define the type of message that the ABCS should receive and reply.

- Define the target service

  Define the services that should be invoked by the ABCS, and the message it should pass to those services.

- Define additional options and generation point

  Define options such as error-handling and extension run-time location, and the launching point to generate an Oracle JDeveloper project for continued development.

After the developer completes the Service Constructor interview, a complete Oracle JDeveloper project is produced that the developer continues to develop incorporating use case-specific requirements.

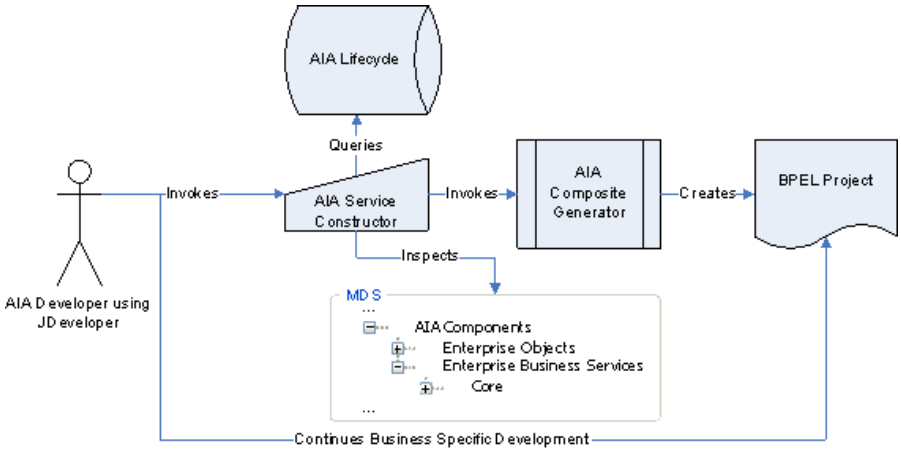
Service Constructor enhances developer productivity by providing a more user-friendly interface to the Composite Generator (formerly the Artifact Generator) and auto-inspection of services and project selection from the Project Lifecycle Workbench.

For more information about Project Lifecycle Workbench, see Chapter 3, "Working with Project Lifecycle Workbench."

Service Constructor relieves developers of the time involved in performing repeatable mundane tasks, enabling them to place greater focus on value-added business scenario-specific tasks and thus reducing ABCS development time.

The overall flow for creating a new AIA service component is illustrated in Figure 4–2.

Figure 4–2 Flow for Creating a New AIA Service Component



4.1.1 Required Software for Using Service Constructor

Service Constructor requires that presence of the following Oracle and third-party components:

- Oracle JDeveloper 11g
- Oracle SOA Composite Editor and Service Constructor

  Both are available through the Oracle JDeveloper update center. In Oracle JDeveloper, select **Help, Check For Updates**.

- Freemarker 2.3.15 or higher

  Visit the Freemarker site at `http://www.freemarker.org` to obtain the Freemarker template engine. Place freemarker.jar (contained in the downloaded zip file) in your <location where AIA Service Constructor was downloaded>/jdev/lib folder found under your Oracle JDeveloper installation.

4.2 Using Service Constructor to Create New Service Solution Components

Developers and solution architects regularly collaborate through multiple channels and use documents such as the functional design document to define and communicate the services that must be developed. To begin the development process, the developer should open an existing application workspace or create an application workspace in Oracle JDeveloper.

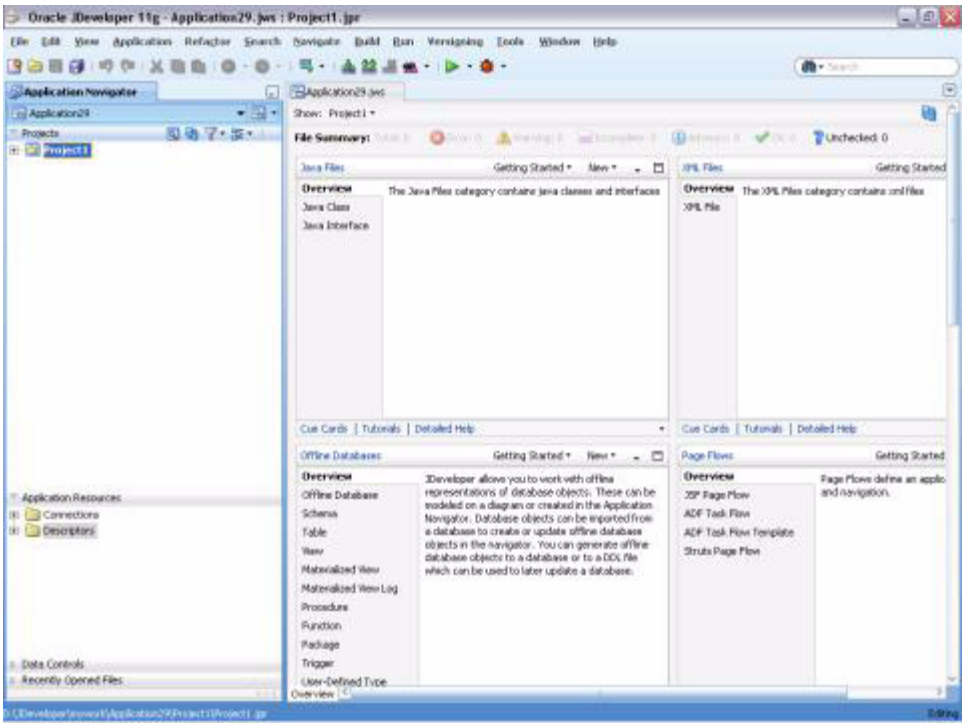
This section discusses the following topics:

4.2.1 Creating a New Service Solution Component Project

To create a service solution component project with Service Constructor:

1. In Oracle JDeveloper, open an existing application workspace or create a generic application, as shown in Figure 4–3.

Figure 4–3 Typical Opening View of a Generic Application in Oracle JDeveloper

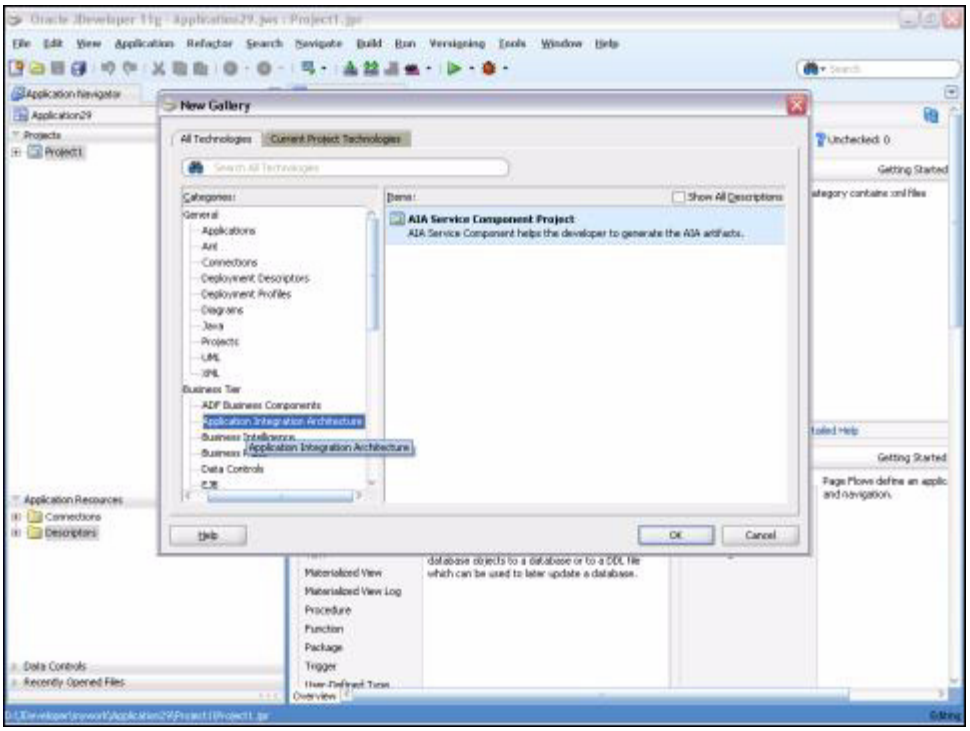


The developer should then create a project within the Workspace to develop a Project Lifecycle Workbench service solution component.

2. To begin the process of creating a service solution component project, click the **New** button. The New Gallery dialog box displays, as shown in Figure 4–4.

   Select the **All Technologies** tab. Select the **Business Tier, Application Integration Architecture** or **General, Projects** tree item. Select the **AIA Service Component Project** list item. Click **OK**.

Figure 4–4 All Technologies Tab



3. The Service Constructor **Welcome** screen displays, as shown in Figure 4–5. Select the **Skip This Page Next Time** option to bypass this page in subsequent visits. Click **Next**. The Service Description page displays.

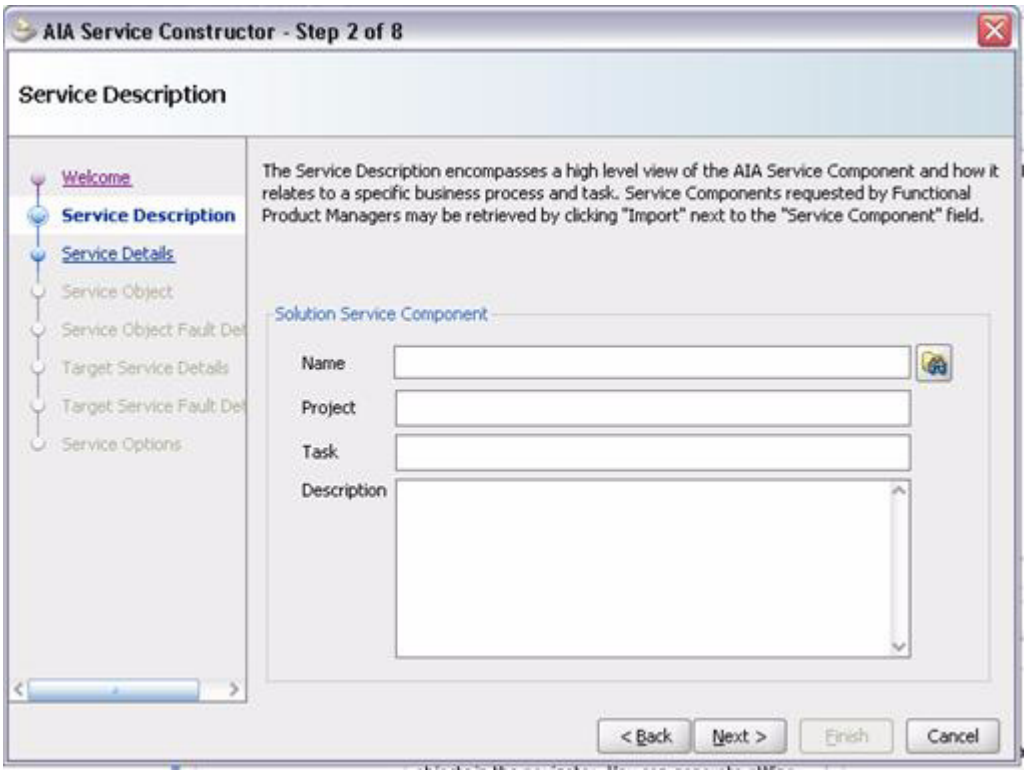
Figure 4–5 Welcome to the AIA Service Constructor

4.2.2 Describing the Service

To define the service description:

1. On the Service Description page, as shown in Figure 4–6, click the **Import** button to access the AIA Resource Browser.

Figure 4–6 Import the Service Description



Use AIA Resource Browser to select the name of an existing service solution component request entered in Project Lifecycle Workbench by a solution architect or functional product manager.

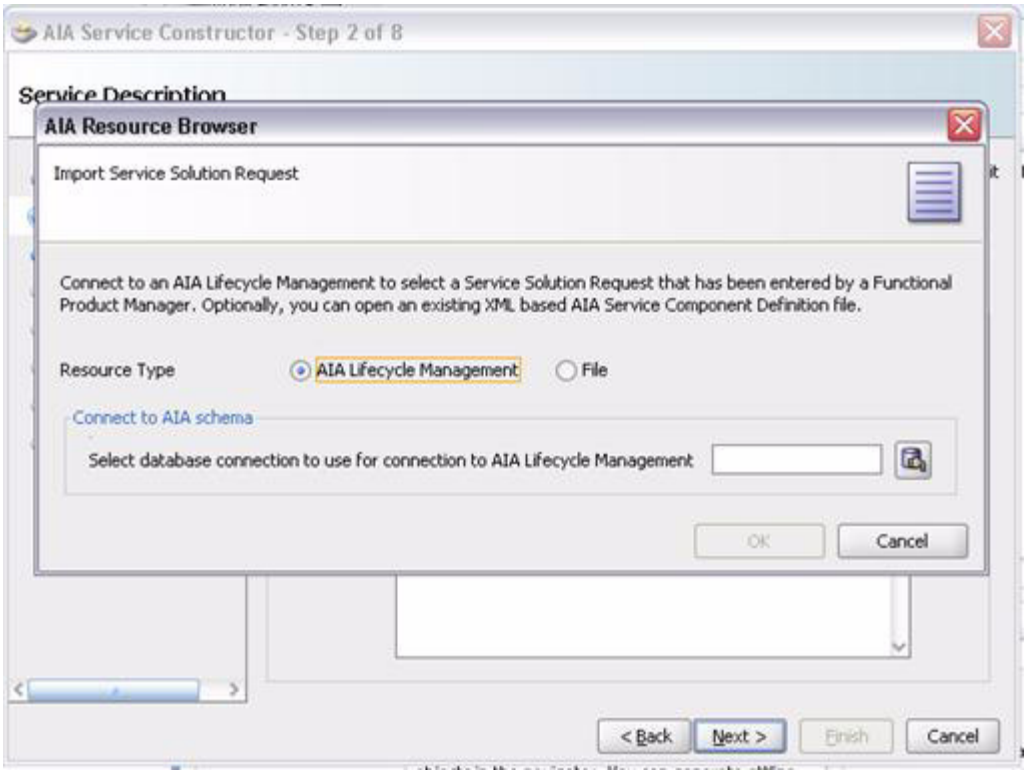
This service solution component request represents a request for creation of a new provider or requester Application Business Connector Service (ABCS).

If the service solution component request is not available from Project Lifecycle Workbench, you can enter the service solution component information directly in Service Constructor. However, Oracle AIA suggests that the functional definition be entered and available from Project Lifecycle Workbench before you start development of a service solution component.

For more information about creating service solution component requests in Project Lifecycle Workbench, see Section 3.4, "Working with Project Lifecycle Workbench Service Solution Components."

2. AIA Resource Browser displays. Select an option for importing a service solution component request, **AIA Lifecycle Management** or **File**, as shown in Figure 4–7.

Figure 4–7 AIA Resource Browser



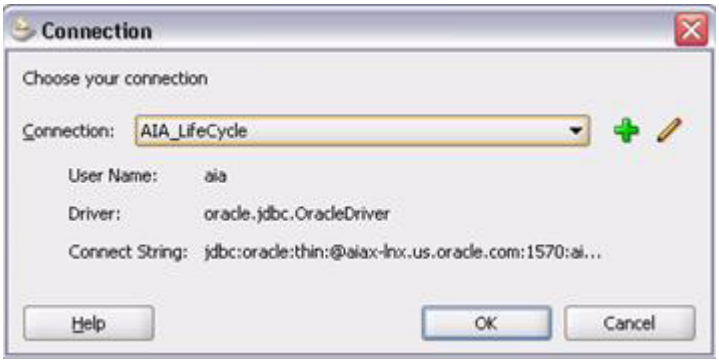
Most users select **AIA Lifecycle Management** to connect to Project Lifecycle Workbench to retrieve the service solution component functional definition entered by a solution architect or functional product manager.

Select the **File** option to create a service solution component using a previously created XML input file for the AIA 2.x Artifact Generator or the AIA Composite Generator.

This flow described here illustrates the retrieval of a service solution component request from Project Lifecycle Workbench.

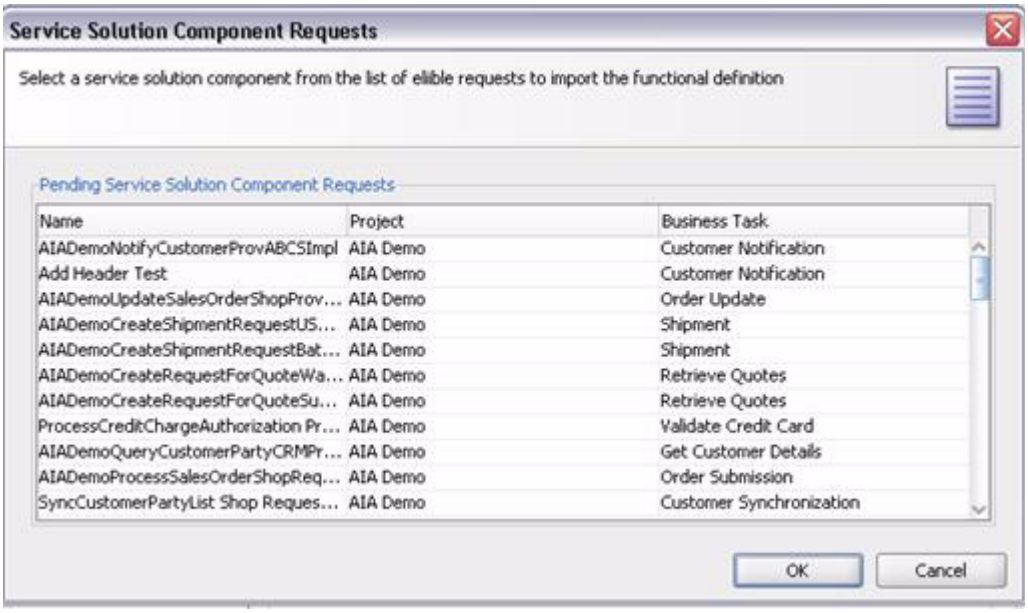
3. Click the **Database Connection** button to access the Connection dialog box, as shown in Figure 4–8, where you can select a connection to Project Lifecycle Workbench. Click **OK**.

Figure 4–8 Connection



4. The Service Solution Component Requests dialog box displays, as shown in Figure 4–9. Select a service solution component from the list of eligible requests. Click **OK**.

Figure 4–9 Service Solution Component Requests



5. The Service Description page is populated with the name, project, task, and description of the select service solution component.

   Following the AIA lifecycle flow, these values were defined by a solution architect performing a functional decomposition of the service in Project Lifecycle Workbench.

6. If necessary, the developer can add to, update, or re-create the information populated to the Service Description shown in Figure 4–10. Available fields are discussed in Table 4–1. Any additions or changes are later harvested into Oracle Enterprise Repository and Project Lifecycle Workbench.

Figure 4–10 Service Description Populated from Project Lifecycle Workbench Service Solution Component Request

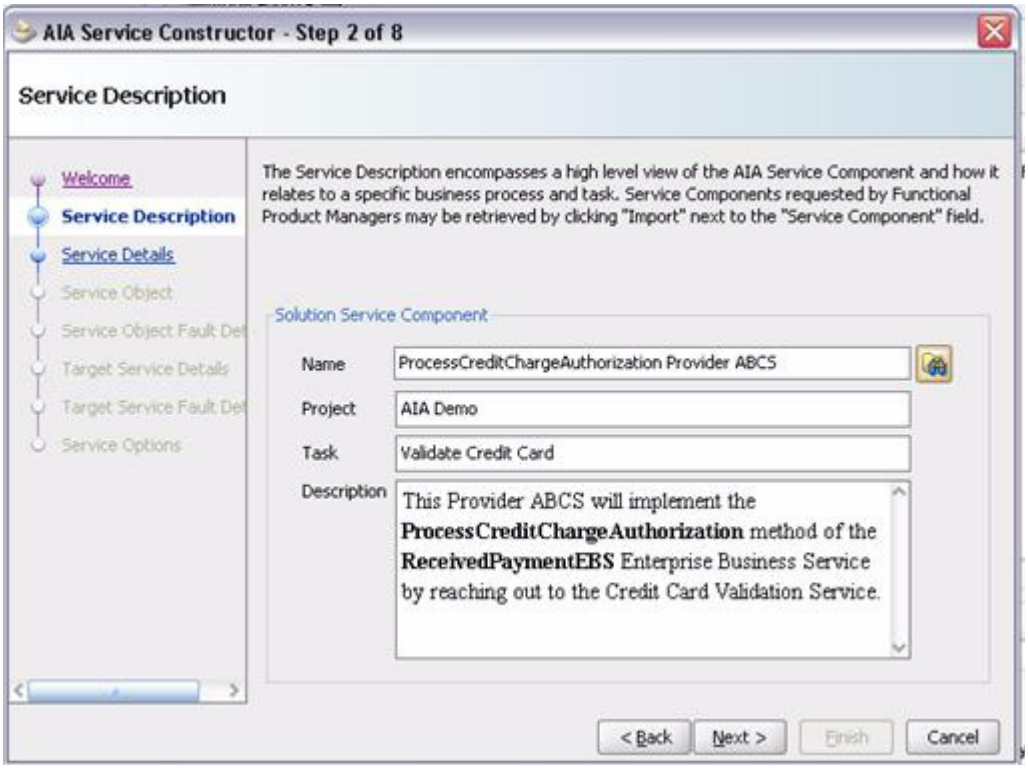


Table 4–1 Service Description Elements

Element	Description
Name	Name of the service solution component.
Project	Name of the project that contains the task that contains the service solution component.
Task	Task that contains the service solution component request.
Description	Details about the requested service solution component, including its purpose.

Click **Next**.

7. The Service Details page displays. Define information about the associated application, and the type of service that is being created.

   Product Code, Industry, and Service Type values typically come from Project Lifecycle Workbench, if an existing service solution component request was selected to access this page.

8. If necessary, the developer can add to, update, or re-create the information populated to the Service Details page, as shown in Figure 4–11. Available fields are discussed in Table 4–2. Any additions or changes are later harvested into Oracle Enterprise Repository and Project Lifecycle Workbench.

Figure 4–11 Service Details

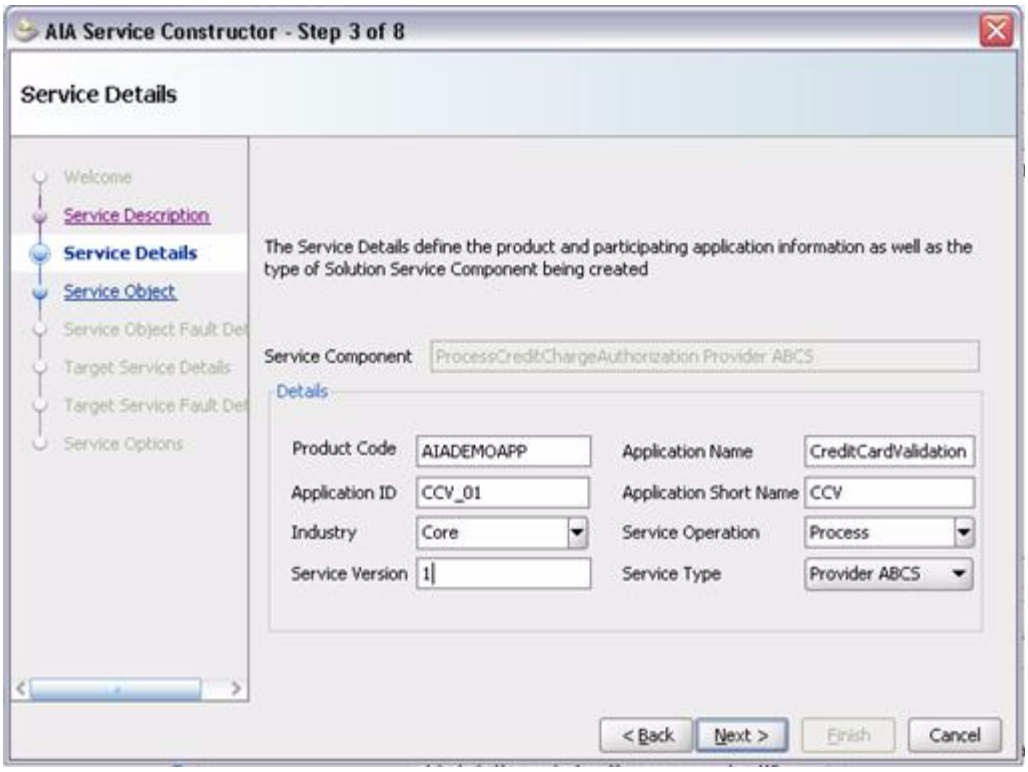


Table 4–2 Service Detail Elements

Element	Description
Service Component	Displays the name of the selected service solution component.
Product Code	Defines the associated application.
Application Name	User-friendly name assigned to the application for which the connector service is being created. This is defined by the developer.
Application ID	Internal code for use in operations, such as cross-referencing. This is defined by the developer.
Application Short Name	Application code, which is also used in naming the connector service. This is defined by the developer.
Industry	Industry to which the connector service applies, such as Insurance, Banking, Utilities, and so on. Select **Core**, if the connector service is not associated with a particular industry.
Service Operation	Defines the verb of the service, such as Create, Update, Delete, Sync, Validate, Process, or Query. This is defined by the developer.
Service Version	Version of the service. This is defined by the developer.
Service Type	Type of service being created: Provider ABCS or Requester ABCS.

Click **Next** to access the Service Object page where you define the inbound message that the service receives. Service objects are also typically referred to as service interfaces. This document refers to them as service objects. Service Constructor leverages service inspection to learn about the message.

4.2.3 Defining the Service Object

After defining the service description, the developer should define the service object, also known as the service interface, to be handled by the ABCS. This is the information about the type of message that is received, which is different based on the type of ABCS.

Requester ABCSs connect participating applications to an Enterprise Business Service (EBS) to enable the application to invoke the EBS. These services receive an Application Business Message (ABM) and produce an Enterprise Business Message (EBM).

Provider ABCSs, however, connect applications to an EBS to enable the EBS to invoke a service provided by the participating applications.

This section includes the following topics:

- Section 4.2.3.1, "Defining the Service Object for a Requester ABCS"
- Section 4.2.3.2, "Defining the Service Object for a Provider ABCS"

4.2.3.1 Defining the Service Object for a Requester ABCS

Define the service object for a requester ABCS on the Service Object page accessed for a service type of Requester ABCS, as shown in Figure 4–12. Available fields are discussed in Table 4–3.

Figure 4–12 Service Object Definition for a Requester ABCS

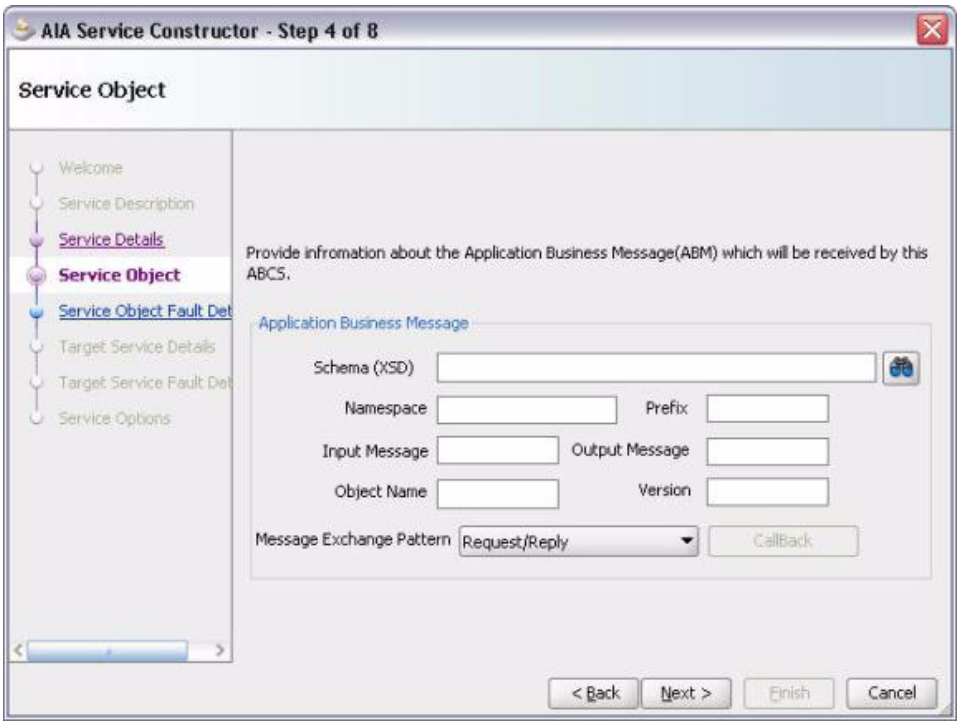


Table 4–3 Service Object Elements for a Requester ABCS

Element	Description
Schema (XSD)	Path to the XML schema definition (XSD) for the message that is received by the service.

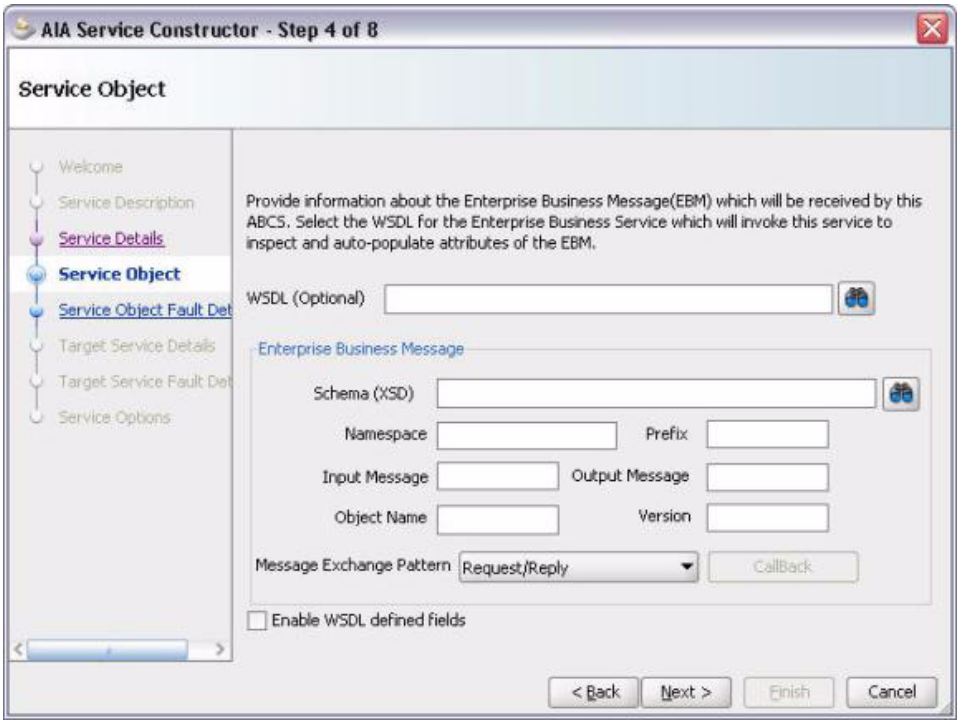
Table 4–3 (Cont.) Service Object Elements for a Requester ABCS

Element	Description
Namespace	A reference in the form of a URL that uniquely identifies the attributes of the service object.
Prefix	Used to associate attributes with this service.
Input Message	Name of the message that is received by the service.
Output Message	Name of message that is returned by the service. Typically, this is used in the request-reply message exchange pattern.
Object Name	Name of the object (noun) that is received by the service.
Version	Version of the object received by the service.
Message Exchange Pattern	Message exchange pattern implemented by the service, such as Request/Reply, Request/Delayed Response, or Fire and Forget.
CallBack	If the service is implementing a Request/Delayed Response message exchange pattern, the CallBack button displays.

4.2.3.2 Defining the Service Object for a Provider ABCS

Define the service object for a provider ABCS on the Service Object page, as shown in Figure 4–13.

Figure 4–13 Service Object Definition for a Provider ABCS



Available fields are discussed in Table 4–4. The service object for a provider ABCS requires the same types of attributes contained in a requester ABCS, however the message being received is an EBM.

Because all AIA EBSs are available through Oracle Metadata Services (MDS), the WSDLs for the services are also available. Service Constructor can inspect a service and learn about the corresponding message.

To inspect a service and learn about the corresponding message:

1. On the Service Object page accessed for a service type of Provider ABCS, click the **Select Service Operation** button to access the Select Service Operation dialog box, where you can select the WSDL, or operation, that invokes the service.

   With the services and schemas being served from MDS, the SOA Resource Browser is typically used to select the appropriate WSDL.

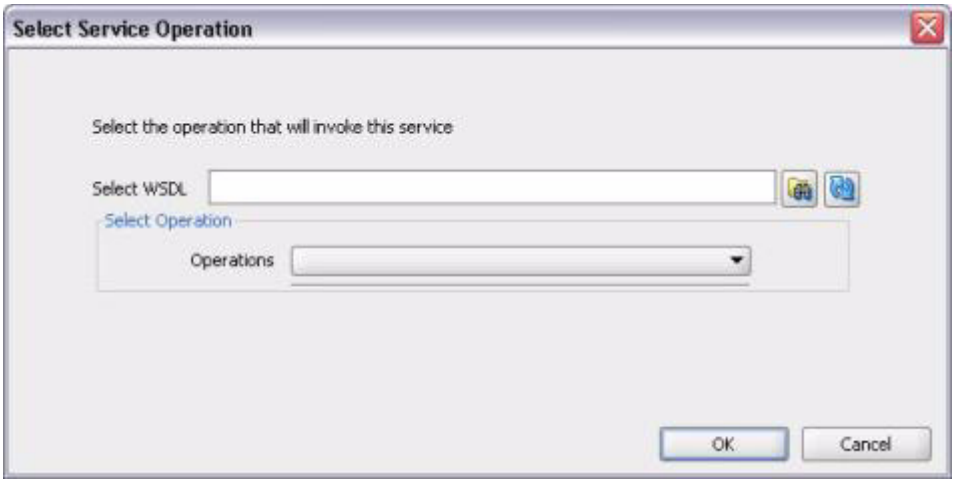
   For more information about creating MDS connections, see Section 2.1.1, "How to Set Up JDeveloper for AIA Development."

   Alternatively, if the WSDL is external and accessible as a URL, such as with a remote service provider, the URL can be pasted directly into the **Select WSDL** field. Then, click the **Reload** button to inspect the WSDL.

   For the purposes of this flow, we will use the SOA Resource Browser.

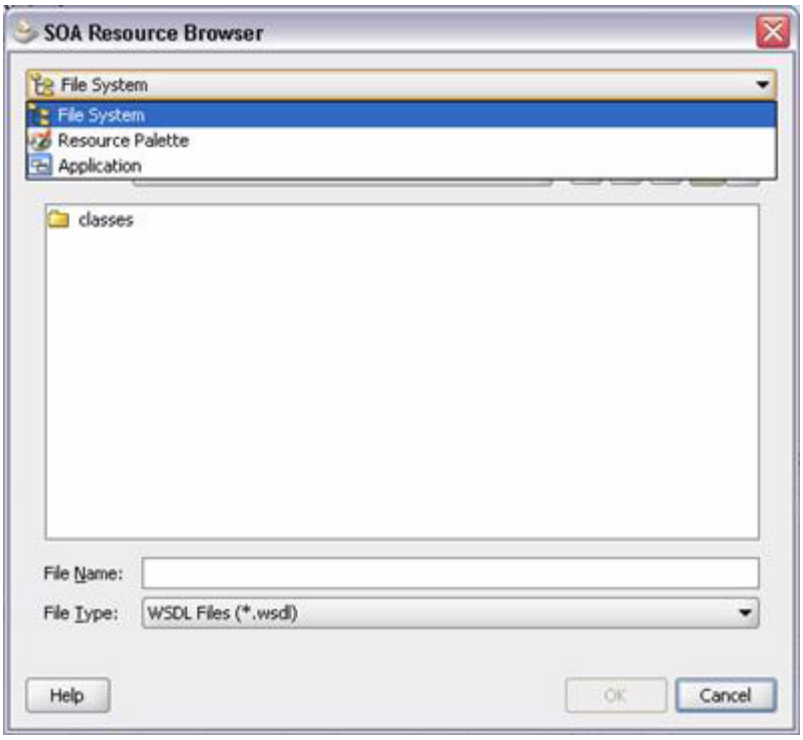
2. In the Select Service Operation dialog box, as shown in Figure 4–14, click the **Resource Browser** button to select the WSDL.

Figure 4–14 Select Service Operation



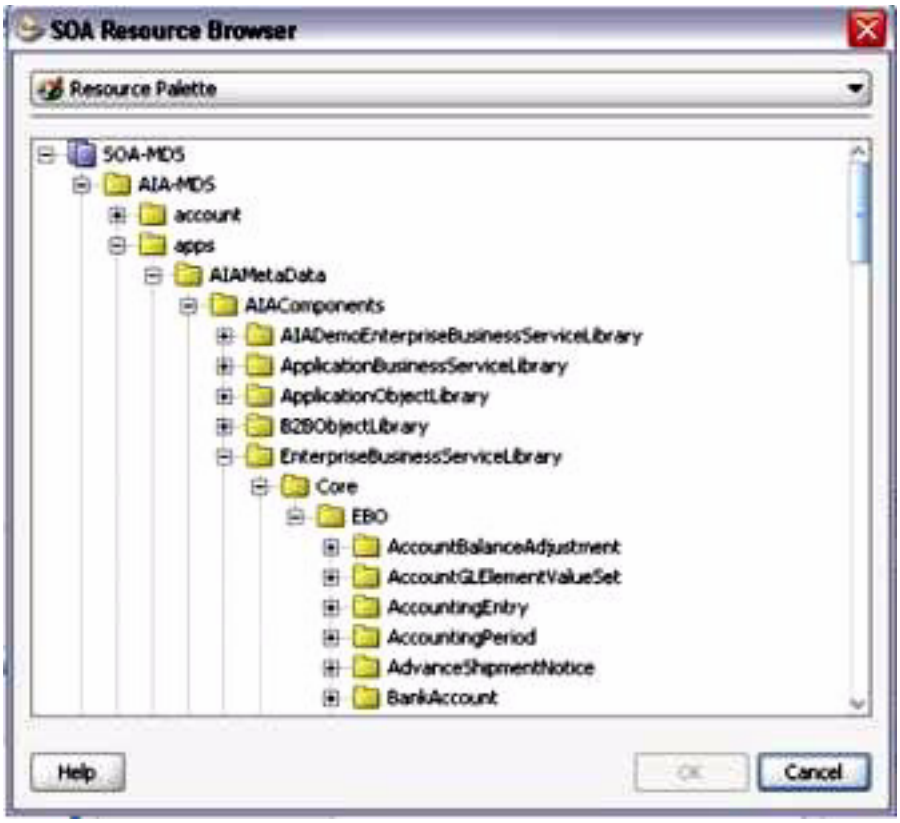
3. The SOA Resource Browser dialog box displays. In the **File System** drop-down list box, select **Resource Palette**, as shown in Figure 4–15.

Figure 4–15 SOA Resource Browser



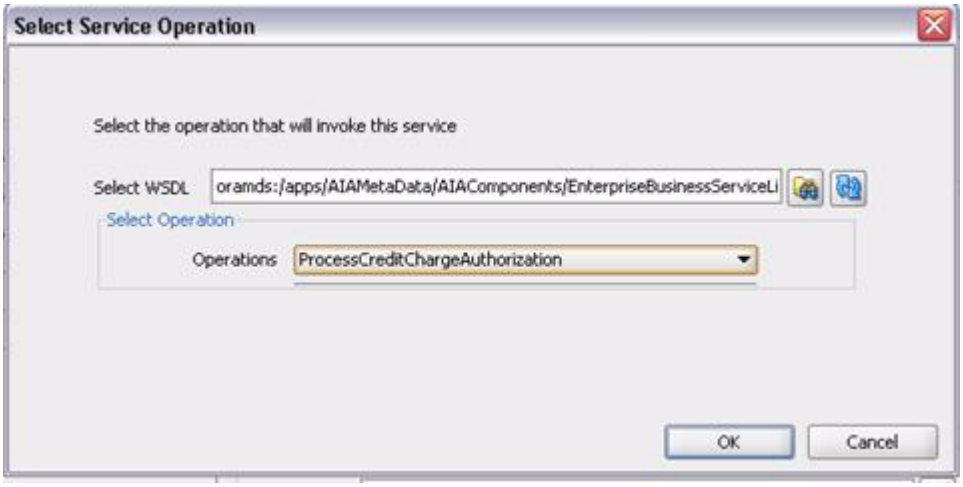
4. Expand the **SOA-MDS** tree item, as shown in Figure 4–16. Navigate to and select the WSDL for the service that invokes the service component you are creating. Click **OK**.

Figure 4–16 SOA Resource Browser - WSDL Selection



5. The Selective Service Operation - Operation Selected page displays, as shown in Figure 4–17. In the **Operations** drop-down list box, select an operation. Click **OK**.

Figure 4–17 Select Service Operation - Operation Selected



Service Constructor inspects the selected service and, in most cases, supplies most of the attributes by default. The developer only needs to define the object name and version.

If the developer must access any of the fields that were automatically populated, such as to make a correction, select the **Enable WSDL defined fields** option.

6. The Service Object page is populated with the WSDL information, as shown in Figure 4–18. Available elements are discussed in Table 4–4.

Figure 4–18 Service Object Definition for a Provider ABCS - WSDL Information Populated

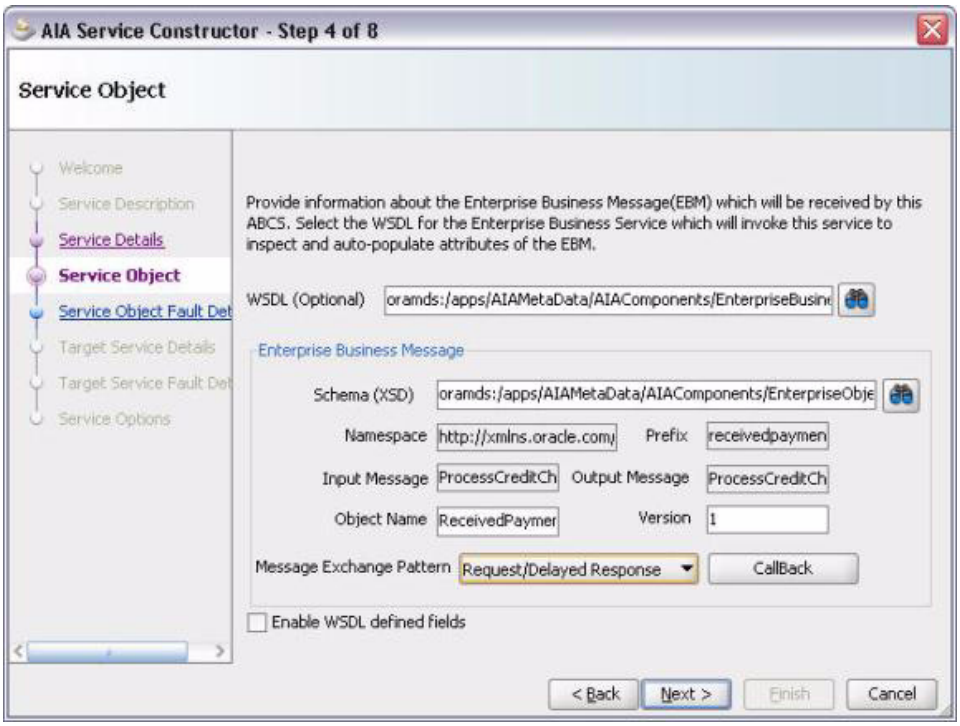


Table 4–4 Service Object Elements for a Provider ABCS

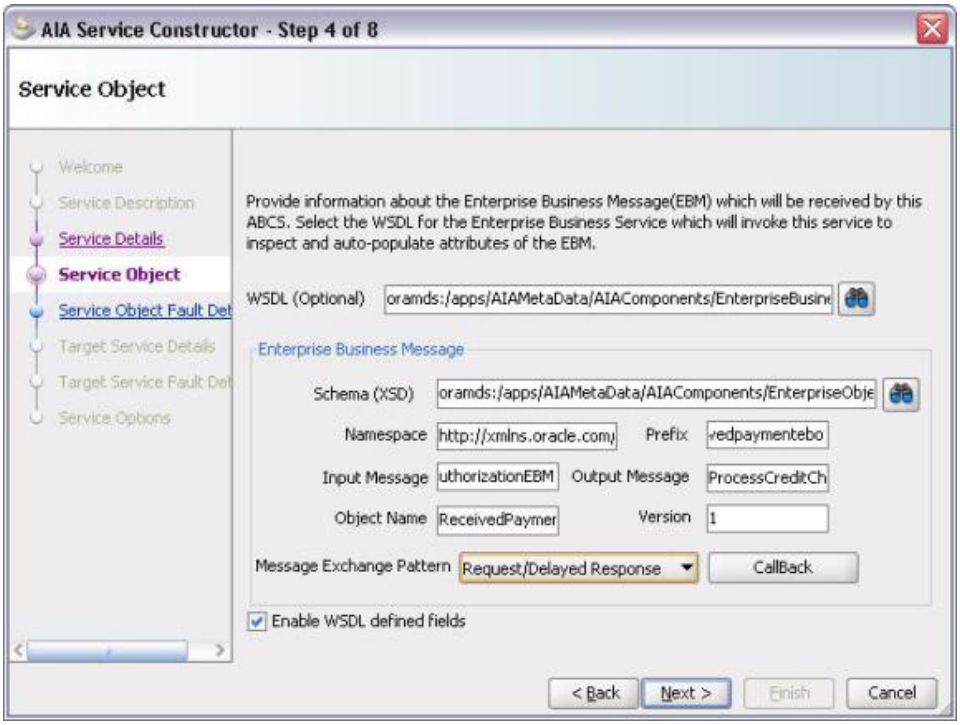
Element	Description
WSDL (optional)	The calling service WSDL is used by the Service Constructor to inspect and automatically supply most of the values by default.
Schema (XSD)	Schema definition of the message type being used by the calling service to call the service interface. This value is typically supplied automatically by the service/operation inspection. The browser button is available if the developer needs to browse for the appropriate object in MDS.
Namespace	Namespace of the underlying object. This value is typically supplied automatically by the service/operation inspection.
Prefix	Prefix to be used as a reference for the message that is received from the calling service. This value is typically supplied automatically by the service/operation inspection.
Input Message	Message that is received from the calling service. This value is typically supplied automatically by the service/operation inspection.
Output Message	Message with which this service may respond to the calling service. This value is typically supplied automatically by the service/operation inspection.
Object Name	Name of the underlying object used in the service invocation. This value is typically supplied automatically by the service/operation inspection.
Version	Version of the underlying object. This value is typically supplied automatically by the service/operation inspection.

Table 4–4 (Cont.) Service Object Elements for a Provider ABCS

Element	Description
Message Exchange Pattern	Message exchange pattern being used. This value is typically supplied automatically by the service/operation inspection based on a series of rules.
Enable WSDL defined fields	In most cases, most or all of the attributes are supplied automatically by the service/operation inspection. If an attribute is not supplied automatically, is supplied incorrectly, or the developer must change any of the values, select this option to make all fields editable.

7. If the service being created is following a Request/Delayed Response message exchange pattern, the **CallBack** button displays on the Service Object page, as shown in Figure 4–19.

Figure 4–19 CallBack Button



Click the **CallBack** button to access the Call Back dialog box, as shown in Figure 4–20. Define the target service that should be invoked by the service being created here as a delayed response. Available fields are discussed in Table 4–5.

Figure 4–20 Call Back Details

Table 4–5 Call Back Elements

Element	Description
WSDL	Click the **WSDL Inspector** button to view a list of the operations. Select the WSDL that invokes the service. You can also use services that are not in MDS or in the local file system, but which are accessible through a URL. If the URL to the service is in the buffer, paste it into the **Select WSDL** field. After the WSDL is selected, click the **Reload** button.
Service NameSpace	Namespace of the service. This value is typically supplied automatically by the service/operation inspection.
Service NameSpace Prefix	Prefix used to reference the namespace. This value is typically supplied automatically by the service/operation inspection.
Input Message	Message this service receives. This value is typically supplied automatically by the service/operation inspection.
Input Message Element	Message element as a part of the message this service receives. This value is typically supplied automatically by the service/operation inspection.
BPEL Scope	Invocation scope defined in BPEL. This value is typically supplied automatically by the service/operation inspection.
Schema (XSD)	Underlying schema definition of the service. This value is typically supplied automatically by the service/operation inspection.
Object Name	Name of the object. This value is typically defined by the developer.
Version	Version of the schema definition. This value is typically defined by the developer.
NameSpace	Namespace of the underlying schema definition. This value is typically supplied automatically by the service/operation inspection.
Prefix	Prefix by which the namespace is referenced. This value is typically supplied automatically by the service/operation inspection.

8. Click **OK** in the Call Back dialog box.
9. Click **Next** on the Service Object page.
10. The Service Object Fault Details page displays as shown in Figure 4–21. Define the fault details for the service interface. Available fields are discussed in Table 4–6.

Figure 4–21 Service Object Fault Details

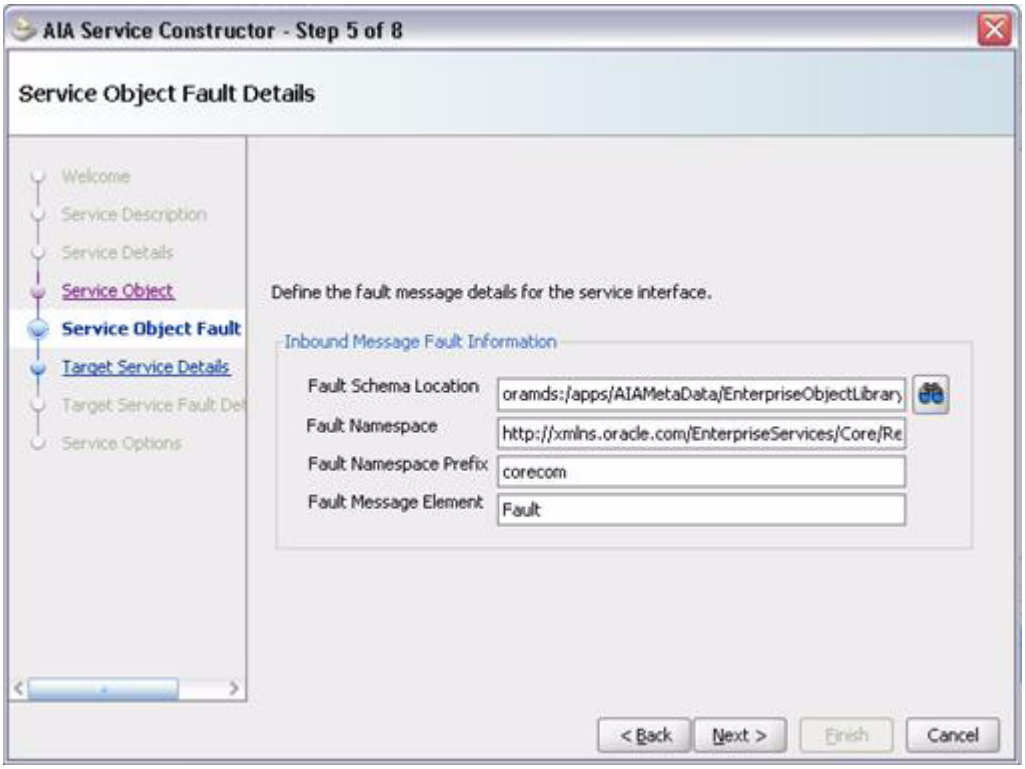


Table 4–6 Service Object Fault Detail Elements

Element	Description
Fault Schema Location	Schema definition for the fault message. This value is typically supplied automatically by the service/operation inspection based on common error definition.
Fault Namespace	Namespace for the fault message. This value is typically supplied automatically by the service/operation inspection.
Fault Namespace Prefix	Prefix to be used as a reference for the fault message. This value is typically supplied automatically by the service/operation inspection.
Fault Message Element	Element definition of the fault message. This value is typically supplied automatically by the service/operation inspection.

11. Click **Next** to access the Target Service Details page.

4.2.4 Defining the Target Services

After the service description and service object are complete, define the target service invocations.

On the Target Service Details page, as shown in Figure 4–22, select the target services that should be invoked. Available fields are discussed in Table 4–7. Here the WSDL is

used to not only inspect and supply the underlying object attributes, but also to be included in part of the service definition.

Note: The Service Constructor supports only services that have an exposed interface and cannot directly invoke JCA adapters, such as the File or DB adapter.

Figure 4–22 Target Service Details

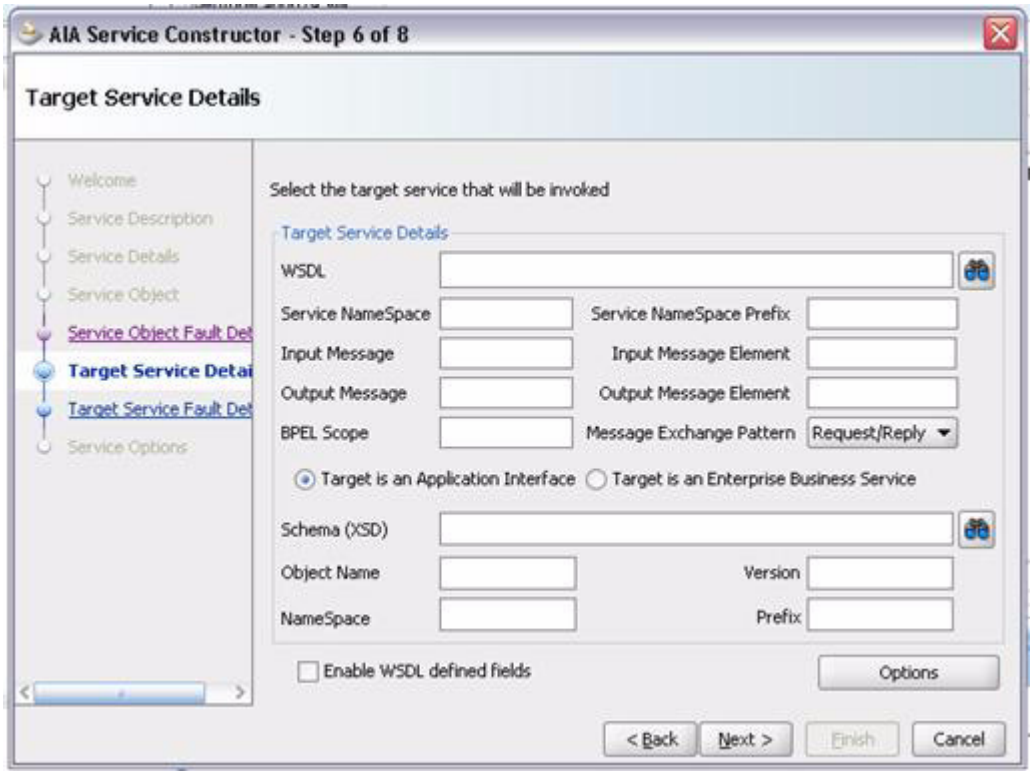


Table 4–7 Target Service Detail Elements

Element	Description
WSDL	Click the **WSDL Inspector** button to view a list of the operations. Select the WSDL that invokes the service. You can also use services that are not in MDS or in the local file system, but are accessible through a URL. If the URL to the service is in the buffer, paste it into the **Select WSDL** field. After the WSDL is selected, click the **Reload** button.
Service NameSpace	Namespace of the service. This value is typically supplied automatically by the service/operation inspection.
Service NameSpace Prefix	Prefix used to reference the namespace. This value is typically supplied automatically by the service/operation inspection.
Input Message	Message that this service receives. This value is typically supplied automatically by the service/operation inspection.
Input Message Element	Message element as a part of the message that this service receives. This value is typically supplied automatically by the service/operation inspection.

Table 4–7 (Cont.) Target Service Detail Elements

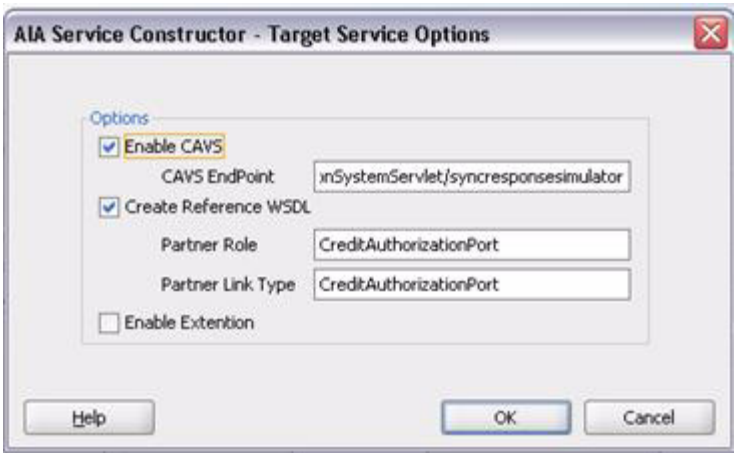
Element	Description
Output Message	Message that this service may return. This value is typically supplied automatically by the service/operation inspection.
Output Message Element	Message element as a part of the message that this service may return. This value is typically supplied automatically by the service/operation inspection.
BPEL Scope	Invocation scope that is defined in BPEL. This value is typically supplied automatically by the service/operation inspection.
MessageExchange Pattern	Message exchange pattern of this service. Service Constructor attempts to supply this value by default based on the input/output messages of the service. The choices are Request/Response, Request/Delayed Response, and FireAndForget.
Target is an Application Interface or Target is an Enterprise Business Service	Defines whether invoking an application interface (or service) or an EBS. Service Constructor attempts to supply this value by default based on the directory structure of the service.
Schema (XSD)	Underlying schema definition of the service. This value is typically supplied automatically by the service/operation inspection.
Object Name	Name of the object. This value is typically defined by the developer.
Version	Version of the schema definition. This value is typically defined by the developer.
NameSpace	Namespace of the underlying schema definition. This value is typically supplied automatically by the service/operation inspection.
Prefix	Prefix by which the namespace is referenced. This value is typically supplied automatically by the service/operation inspection.
Enable WSDL defined fields	This option is deselected by default. Most of the fields are supplied automatically so they are set to uneditable. If for some reason the developer must make a change to one of the preceding fields, selecting this option makes them editable.

To define the target services:

1. On the Target Service Details page, click the **Options** button to access the Target Service Options dialog box, as shown in Figure 4–23.

   Target service options such as the Composite Application Validation System (CAVS) and the reference WSDL creation are enabled by default. If an extension service is used, enable it here. Click **OK**.

Figure 4–23 Target Service Options



2. On the Target Service Details page, click **Next** to access the Target Service Fault Details page, as shown in Figure 4–24, where you can define the fault details for the target service. Available fields are discussed in Table 4–8.

Figure 4–24 Target Service Fault Details

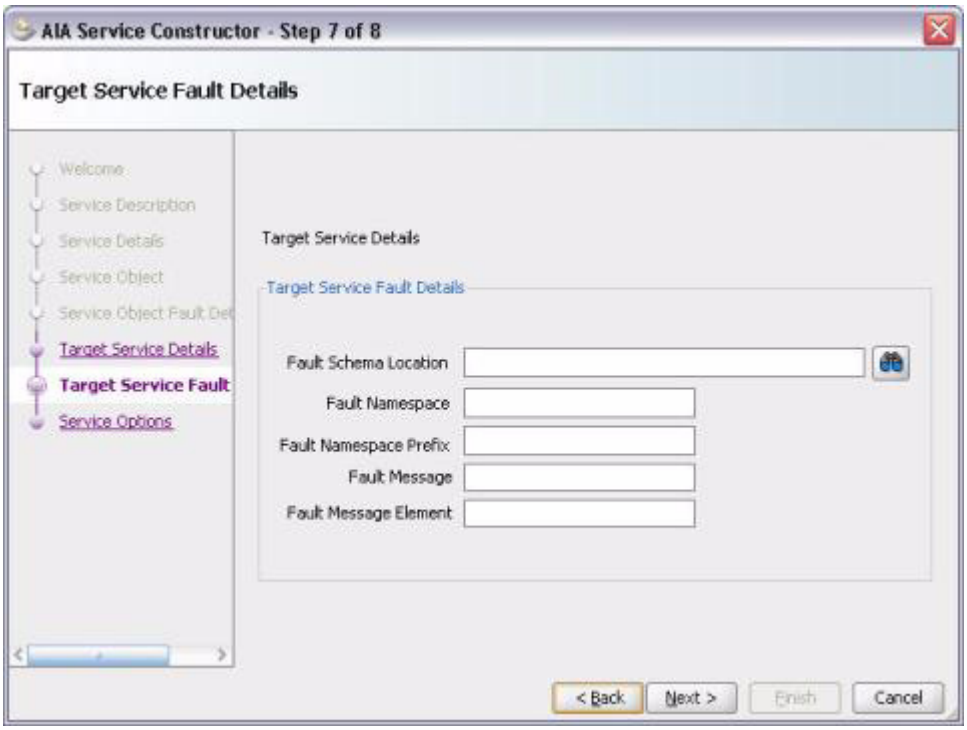


Table 4–8 Target Service Fault Detail Elements

Element	Description
Fault Schema Location	Defines the schema of the fault message that may be returned. This value is typically supplied automatically by the service/operation inspection.

Table 4–8 (Cont.) Target Service Fault Detail Elements

Element	Description
Fault Namespace	Defines the name space of the fault schema. This value is typically supplied automatically by the service/operation inspection.
Fault Namespace Prefix	Defines the prefix by which elements are attributed to the namespace. This value is typically supplied automatically by the service/operation inspection.
Fault Message	Message that may be returned in a fault. This value is typically supplied automatically by the service/operation inspection.
Fault Message Element	Element within the message that may be returned in a fault. This value is typically supplied automatically by the service/operation inspection.

3. Click **Next** to access the Service Options page.

4.2.5 Defining Service Options

To define service options:

1. On the Service Options page, as shown in Figure 4–25, enter service option details. Available fields are discussed in Table 4–9.

Figure 4–25 Service Options

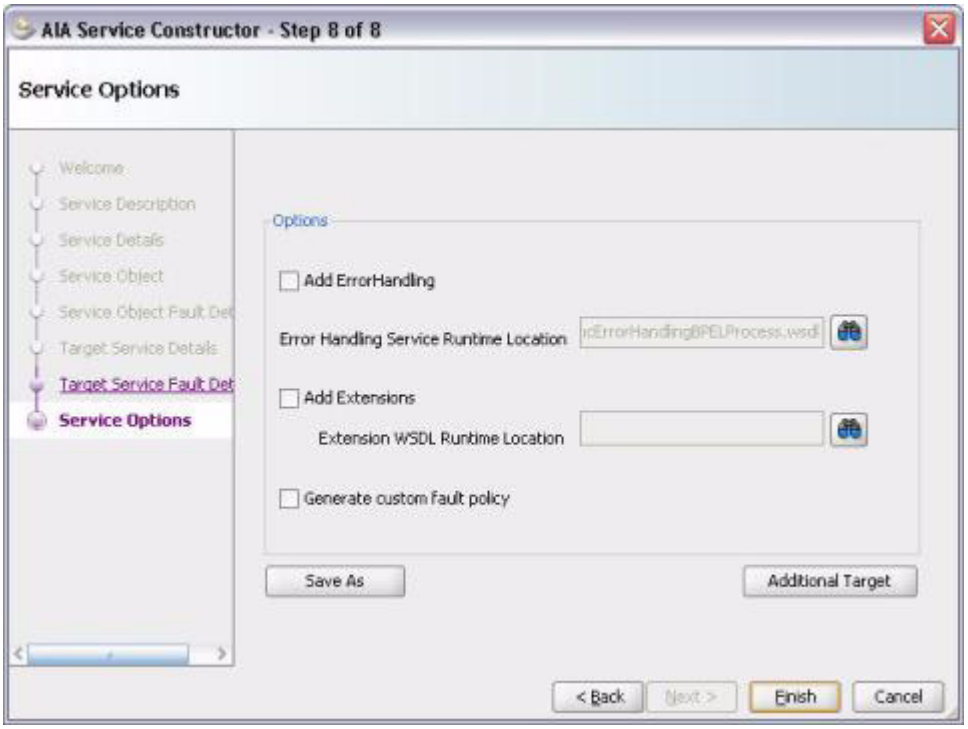


Table 4–9 Service Option Elements

Element	Description
Add ErrorHandling	Option to add error handling. The default value is *TRUE*.

Table 4–9 (Cont.) Service Option Elements

Element	Description
Error Handling Service Runtime Location	Designate the runtime location of the AIAAsyncErrorHandlingBPELProcess service. The default location is the common oramds:/ location. If you select the AIAAsyncErrorHandlingBPELProcess service WSDL location in the MDS infrastructure library, perform the post-service-generation task described following this table. If you select the actual runtime endpoint location value of AIAAsyncErrorHandlingBPELProcess, no post-service-generation task is required.
Add Extensions	Option to add extensions. If the Enable Extension option is enabled under the Target Service options, then this option is selected by default and is required.
Extension WSDL Runtime Location	WSDL of extension service.
Generate custom fault policy	Select to be able to create a custom fault policy.
Additional Target	Click to add an additional target. Certain participating applications require calls to multiple service interfaces to complete the necessary steps for message transformation (message enrichment, and so on).

If you have selected the AIAAsyncErrorHandlingBPELProcess service WSDL location in the MDS infrastructure library as the Error Handling Service Runtime Location value, perform the following post-service-generation task:

a. Access the generated composite to update the location attribute of the binding.ws element within the reference element for AIAAsyncErrorHandlingBPELProcess.

b. Update the location with the actual AIAAsyncErrorHandlingBPELProcess runtime location. For example, the location indicated in bold in Example 4–1, must be updated.

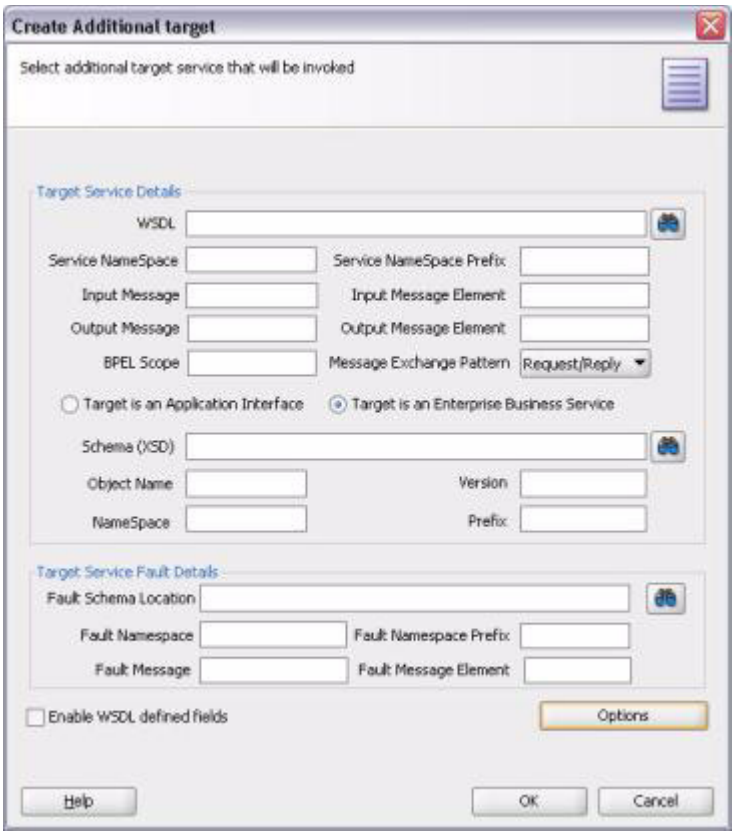
Example 4–1 Reference Element for AIAAsyncErrorHandlingBPELProcess

```
<reference ui:wsdlLocation="oramds:/apps/AIAMetaData/AIAComponents/
InfrastructureServiceLibrary/V1/wsdls/AIAAsyncErrorHandlingBPELProcess.wsdl"
name="AIAAsyncErrorHandlingBPELProcess">
...
   binding.ws port="http://xmlns.oracle.com/AIAAsyncErrorHandlingBPEL
Process#wsdl.endpoint(AIAAsyncErrorHandlingBPELProcess/AIAAsyncError
HandlingBPELProcess)" location="oramds:/apps/AIAMetaData/AIAComponents/
InfrastructureServiceLibrary/V1/wsdls/AIAAsyncErrorHandlingBPELProcess.wsdl"
xmlns:ns="http://xmlns.oracle.com/sca/1.0"/>
...
</reference>
```

2. If an additional target is needed, click the **Additional Target** button on the Service Option page to access the Create Additional Target dialog box, as shown in Figure 4–26, where you can add another invocation for an application that cannot support the EBM through only one invocation.

For example, take a scenario in which updating a Person in a participating application may require an invocation to both a Person and an Address service.

Figure 4–26 Create Additional Target



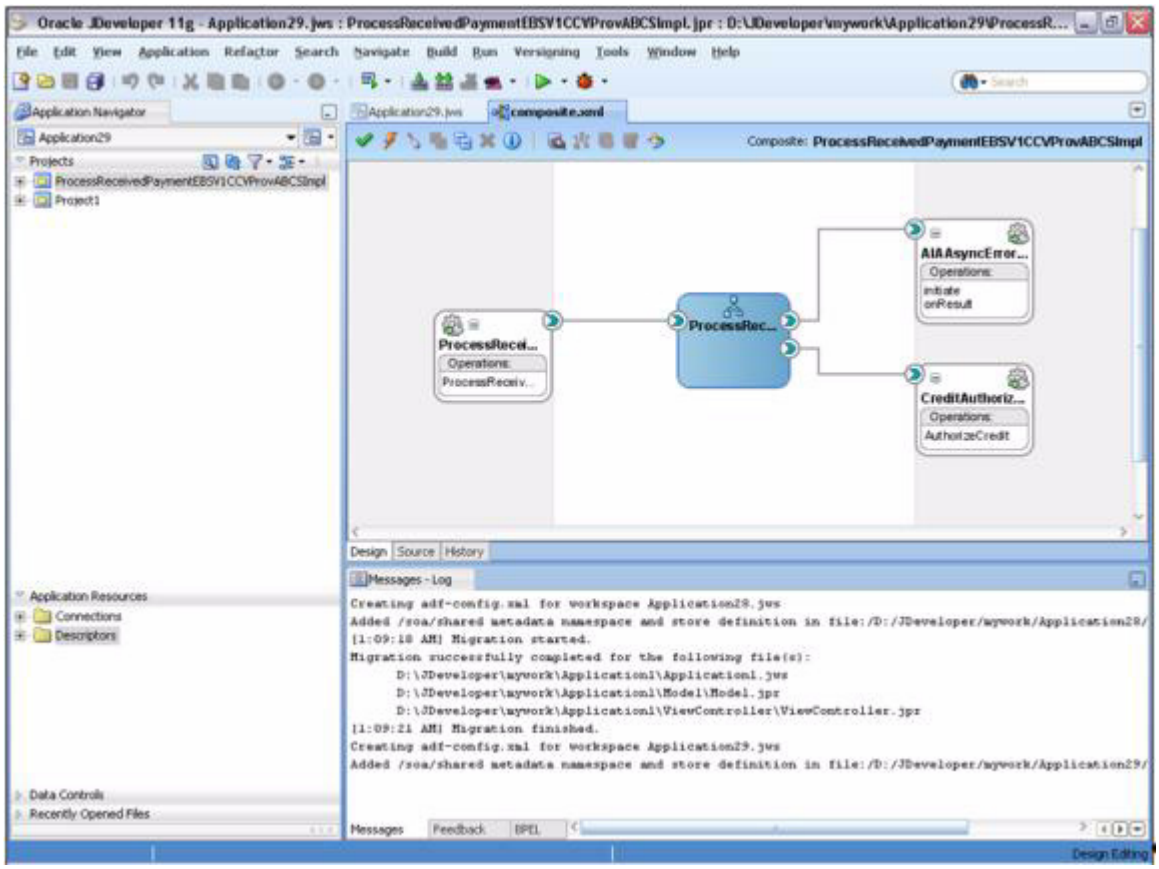
This dialog box contains the same attributes and features as the primary target dialog box, but consolidates the service details, fault details, and options.

The details for the additional target are completed in the same manner discussed for the Target Service Details page, discussed in Section 4.2.4, "Defining the Target Services".

3. Optionally, click the **Save As** button to save a service solution component request input file. Saving the input file allows it to be used with the command line-based Composite Generator, if desired, or for troubleshooting of the Service Constructor.
4. Click **Finish**.

   The ABCS composite has been created, as shown in Figure 4–27, and is ready for further development.

Figure 4–27 Example of the ABCS Composite



After service construction is completed, you can choose to perform harvesting of your design-time composites into the Project Lifecycle Workbench and optionally, to Oracle Enterprise Repository.

For more information, see Section 5.2, "Harvesting Design-Time Composites into Project Lifecycle Workbench and Oracle Enterprise Repository."

5 Harvesting Oracle AIA Content

This chapter discusses how to set up Oracle AIA content harvesting, harvest design-time composites into Project Lifecycle Workbench and Oracle Enterprise Repository (OER), harvest interfaces to Oracle Repository in bulk, harvest deployed composites into OER and introduce OER to AIA after you have finished installing AIA.

This chapter includes the following sections:

Note: All script execution examples provided in this chapter are Linux-specific. For Windows script execution, replace ".sh" with ".bat."

5.1 How to Set Up Oracle AIA Content Harvesting

Objective

Set up your environment to enable harvesting of Oracle Application Integration Architecture (AIA) content. This setup is a prerequisite to performing any Oracle Enterprise Repository harvesting covered in this chapter.

Actor

System administrator

To set up Oracle AIA content harvesting:

1. Import the Oracle Enterprise Repository solution pack file (<BEA_HOME>/repository111/core/tools/solutions /11.1.1.4.0-OER-Harvester-Solution-Pack.zip) from the Oracle Enterprise Repository installation home. This is a setup step required by Oracle Enterprise Repository.
2. Import the AIA asset definition file (AIAAssetTypeDef.zip) located in $AIA_HOME/Infrastructure/LifeCycle/config to the Oracle Enterprise Repository server. This is a setup step required by Oracle AIA.

3. Enable remote Java Database Connectivity (JDBC) in WebLogic Server. Ensure that the remote JDBC is enabled.
   a. Go to the domain (for example: soadomain), and open the file setDomainEnv.sh.
   b. Search for WLS_JDBC_REMOTE_ENABLED="-Dweblogic.jdbc.remoteEnabled= false" and set the value to *true*.
   c. Restart the Admin server.

For more information about performing these imports, see "Importing Items into Oracle Enterprise Repository" in *Oracle Fusion Middleware User Guide for Oracle Enterprise Repository.*

5.2 Harvesting Design-Time Composites into Project Lifecycle Workbench and Oracle Enterprise Repository

This section includes the following topics:

- Section 5.2.1, "Introduction to Harvesting Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository"
- Section 5.2.2, "How to Set Up Environments to Enable Design-Time Harvesting"
- Section 5.2.3, "How to Harvest Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository"

5.2.1 Introduction to Harvesting Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository

After you have unit-tested, source-controlled, and completed your composite implementation, you can harvest these design-time composites into the Project Lifecycle Workspace and, optionally, Oracle Enterprise Repository.

When you harvest into the Project Lifecycle Workbench, annotations in composite XML files are published to Project Lifecycle Workbench. These annotations published to Project Lifecycle Workbench are instrumental in facilitating downstream automation, such as bill of material (BOM) generation and deployment plan generation. Annotations and harvesting are required to enable this downstream automation.

If downstream automation is not required, you can choose not to annotate and harvest. When you reach a point in the lifecycle flow at which the result of annotations and harvesting are used, such as BOM and deployment plan generation, you can manually complete the BOM through the Project Lifecycle Workbench UI or manually write your own ANT script to generate a deployment plan.

For more information about BOM generation, see Chapter 6, "Working with Project Lifecycle Workbench Bills of Material."

For more information about deployment plan generation, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

When you harvest into Oracle Enterprise Repository, annotations on Application Business Connector Service (ABCS) WSDL files, Enterprise Business Service (EBS) WSDL files, Enterprise Business Object (EBO) XSD files, and Enterprise Business Message (EBM) XSD files are published to Oracle Enterprise Repository. Harvesting to Oracle Enterprise Repository is optional.

For more information about viewing harvested AIA artifacts in Oracle Enterprise Repository, see Chapter 11, "Configuring and Using Oracle Enterprise Repository as the Oracle AIA SOA Repository."

Harvested composites may be annotated based on AIA annotation specifications or may be generic, nonannotated composites provided straight out of Oracle JDeveloper. If composites are not annotated correctly, accuracy of downstream data, BOM generation, and deployment plan generation may suffer.

For more information about AIA annotation specifications, see Chapter 12, "Annotating Composites."

Oracle AIA recommends that you harvest your composite into both Project Lifecycle Workspace and Oracle Enterprise Repository if an Oracle Enterprise Repository instance is available for your implementation.

Oracle AIA also recommends that you harvest composites upon completion of the composite implementation when the composite is in a stable and tested condition.

5.2.2 How to Set Up Environments to Enable Design-Time Harvesting

This section includes the following topics:

- Section 5.2.2.1, "Setting Up for Design-Time Harvesting Using a Non-Foundation Pack Environment"
- Section 5.2.2.2, "Setting Up for Design-Time Harvesting Using a Foundation Pack Environment"

Objective

Set up environments so that design-time harvesting can be done through command line. Perform this setup in preparation for harvesting design-time composites into the Project Lifecycle Workspace and Oracle Enterprise Repository.

Prerequisites and Recommendations

Oracle Meta Data Services (MDS) repository must be set up and populated with the prerequisite and dependent artifacts for the composites to be harvested, such as Enterprise Object Library and Application Object Library artifacts.

For more information about setting up and populating MDS, see Chapter 2, "Building AIA Integration Flows."

To avoid a weblogic.common.resourcepool.ResourceLimitException error when running AIAHarvester, ensure that SOA-MDS is configured according to a recommended set of minimum basic configurations.

For information about the minimum basic configurations, see Section 30.2.3, "Configuring Performance Related Database Initialization Parameters."

Actor

Individual developers

5.2.2.1 Setting Up for Design-Time Harvesting Using a Non-Foundation Pack Environment

To set up environments to enable design-time command line-based harvesting using a non-Foundation Pack environment:

1. Download AIAHarvester.zip.

AIAHarvester.zip contains all components necessary to perform a harvest against Project Lifecycle Workbench and Oracle Enterprise Repository.

AIAHarvester.zip no longer includes adf-config.xml. This file is no longer needed by AIAHarvester. If your installation was upgraded from a previous release, you may still have this file but no further action is required.

2. Unzip AIAHarvester.zip in any location but maintain the unzipped structure.

> **Note:** The following steps assume that tasks are being performed within the AIAHarvester directory.

3. Update the values shown in bold in the ./AIAInstallProperties.xml file, as shown in Example 5–1.

   Foundation Pack uses Java Database Connectivity (JDBC) to determine the Project Lifecycle Workbench database. Ensure that the `<jdbc-url>jdbc:oracle:thin:@localhost:1521:XE</jdbc-url>` value points to the Project Lifecycle Workbench database where you want AIAHarvester results to be stored.

Example 5–1 AIAInstallProperties.xml File Adjustments

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<properties>
    <aiainstance>
        <aiaHome>AIA_HOME</aiaHome>
        <name>AIA_INSTANCE_NAME</name>
        <javahome>JAVA_HOME</javahome>
        <remote_install>true</remote_install>
        <domain_root>SERVER_DOMAIN_ROOT</domain_root>
        <mwHome>MW_HOME</mwHome>
        <soaHome>SOA_HOME</soaHome>
        <aiainstalltype>standard</aiainstalltype>
        <isencrypted>true</isencrypted>
    </aiainstance>
    <aialifecycle>
        <jdbc-url>jdbc:oracle:thin:@localhost:1521:XE</jdbc-url>
        <username>FPINST_AIALIFECYCLE</username>
        <password>[C@8b567c</password>
        <createschema>true</createschema>
        <sysusername>sys</sysusername>
        <syspassword>[C@15b0e2c</syspassword>
        <role>SYSDBA</role>
        <defaulttablespace>SYSTEM</defaulttablespace>
        <temptablespace>TEMP</temptablespace>
        <isRac>false</isRac>
        <racCount>1</racCount>
        <racInstances>rac0</racInstances>
        <rac>
            <serviceName>RAC_DATABASE_SERVICENAME</serviceName>
            <rac0>
                <instanceName>RAC_INSTANCE_NAME</instanceName>
                <host>INSTANCE_HOST</host>
                <port>INSTANCE_PORT</port>
            </rac0>
        </rac>
    </aialifecycle>
...
```

4. Ensure the accuracy of information in the ./META-INF/adf-config.xml file. Under normal circumstances, you should not have to update it.

   Specifically, ensure that the file includes the `jdbc/mds/MDS_LocalTxDataSource` value that appears in bold in Example 5–2.

Example 5–2 adf-config.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<adf-config xmlns="http://xmlns.oracle.com/adf/config" xmlns:adf="http://xmlns.
 oracle.com/adf/config/properties" xmlns:sec="http://xmlns.oracle.com/adf/
 security/config">
    <adf:adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/
     properties">
        <adf-property name="adfAppUID" value="OER.oracle.apps.aia.oer"/>
    </adf:adf-properties-child>
    <adf-mds-config xmlns="http://xmlns.oracle.com/adf/mds/config">
        <mds-config xmlns="http://xmlns.oracle.com/mds/config">
            <persistence-config>
                <metadata-namespaces>
                    <namespace path="/soa/b2b" metadata-store-usage=
                     "soa-infra-store"/>
                    <namespace path="/deployed-composites" metadata-store-usage=
                     "soa-infra-store"/>
                    <namespace path="/soa/shared" metadata-store-usage=
                     "soa-infra-store"/>
                    <namespace path="/apps" metadata-store-usage=
                     "soa-infra-store"/>
                </metadata-namespaces>
                <metadata-store-usages>
                    <metadata-store-usage id="soa-infra-store" deploy-target=
                     "true">
                        <metadata-store class-name="oracle.mds.persistence.
                         stores.db.DBMetadataStore">
                        <property value="soa-infra" name="partition-name"/>
                        <property value="jdbc/mds/MDS_LocalTxDataSource" name=
                         "jndi-datasource"/>
                        </metadata-store>
                    </metadata-store-usage>
                </metadata-store-usages>

                <auto-purge seconds-to-live="300"/>
            </persistence-config>

        </mds-config>
    </adf-mds-config>
</adf-config>
```

5. Encode the password for the Project Lifecycle Workbench database by running AIALifeCycleEncode.sh: `./AIALifeCycleEncode.sh -user <username>`

   When prompted, enter the password.

6. Encode the password for the application server by running AIALifeCycleEncode.sh: `./AIALifeCycleEncode.sh -user <username> -type jndi`

   Replace `<username>` with the WebLogic username value. When prompted, enter the password.

7. Decide upon the type of harvesting:
   - Oracle Enterprise Repository and Project Lifecycle Workbench database harvesting
   - Project Lifecycle Workbench database harvesting only
8. Generate the HarvesterSettings.xml file.

   If you prefer to harvest using command-line options instead of the HarvesterSettings.xml file, you can skip this step and can proceed to Section 5.2.3.3, "How to Harvest Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository Using Command Line Options."
   - Adjust the harvesterSettings_prep.xml file based on the choice that you made in the previous step.
   - Run an encryption script on the harvesterSettings_prep.xml to produce the final HarvesterSettings.xml file, which you will use as the input file to the harvester script that harvests your composites: `./Harvester/encrypt.sh <source_file> <target_file>`.

     For example: `./Harvester/encrypt.sh harvesterSettings_prep.xml HarvesterSettings.xml`

harvesterSettings_prep.xml and HarvesterSettings.xml file names are used for illustrative purposes only. The source and target file names can be set to any file name.

5.2.2.2 Setting Up for Design-Time Harvesting Using a Foundation Pack Environment

To set up environments to enable design-time command line-based harvesting using a Foundation Pack environment:

1. Run `source $AIA_INSTANCE/bin/aiaenv.sh`.

   Note: The encrypted database access information is stored in $AIA_INSTANCE/Infrastructure/LifeCycle/AIAHarvester/config/.configuration/Properties.

   If you subsequently change the database user name, for example, re-encrypt the database access information by running `$AIA_HOME/Infrastructure/LifeCycle/AIAHarvester/AIALifeCycleEncode.sh`.

   ---

2. Prepare HarvesterSettings.xml, which contains the file system directory path to the newly completed composite.xml.
3. If you are harvesting to Oracle Enterprise Repository, run the following encryption script to produce the final HarvesterSettings.xml file, which you will use as the input file to the harvester script that harvests your composites: `$AIA_HOME/Infrastructure/LifeCycle/AIAHarvester/Harvester/encrypt.sh <source> <target>`

5.2.3 How to Harvest Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository

This section discusses:

- Section 5.2.3.1, "How to Harvest Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository Using HarvesterSettings.xml."
- Section 5.2.3.2, "How to Harvest Design-Time Composites into Project Lifecycle Workbench Only Using HarvesterSettings.xml."
- Section 5.2.3.3, "How to Harvest Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository Using Command Line Options."

Objective

Harvest design-time composites into Project Lifecycle Workspace and Oracle Enterprise Repository either by using the harvester script and the HarvesterSettings.xml file or by using command-line options.

Prerequisites and Recommendations

- Complete the steps covered in Section 5.1, "How to Set Up Oracle AIA Content Harvesting."
- Complete the steps covered in Section 5.2.2, "How to Set Up Environments to Enable Design-Time Harvesting."
- Ensure that the composite to be harvested is fully annotated according to AIA specifications. This is the ideal scenario.

  These annotations include those in ABCS WSDL files and composite XML files. AIA EBO and EBM XSD files are delivered with annotations.

  Annotations are not required, however downstream automation features will not work as designed if they are not annotated properly.

For more information about AIA annotation specifications, see Chapter 12, "Annotating Composites."

Actor

Individual developers

5.2.3.1 How to Harvest Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository Using HarvesterSettings.xml

To harvest design-time composites into Project Lifecycle Workspace and Oracle Enterprise Repository using HarvesterSettings.xml:

1. Ensure that your HarvesterSettings.xml file is accurately configured. Sample file content is provided in Example 5–3.

   In particular, ensure that the content shown:

   - Provides Oracle Enterprise Repository server information and credentials.
   - Uses `<fileQuery>`.
   - Specifies the file system path to the composite.xml for a given composite project.

Example 5–3 Sample HarvesterSettings.xml to Harvest Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:harvesterSettings xmlns:tns="http://www.oracle.com/oer/integration/
 harvester" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.oracle.com/oer/integration/harvester
```

```
 Harvester_Settings.xsd">
   <!--Description to set on created Assets in OER.-->
   <harvesterDescription>Oracle Enterprise Repository Harvester
   </harvesterDescription>

   <!--Registration status to set on created Assets in OER. The Valid
    Registration states are 1) Unsubmitted 2)Submitted - Pending Review
    3)Submitted - Under Review  4)Registered -->
   <registrationStatus>Registered</registrationStatus>
   <!--Namespace to set on created Assets in OER.  If left empty, this is set
    based on information from SOA Suite and OSB projects when available. That's
    generally the best practice, so override this with caution.-->
   <namespace/>


   <!--Connection info to OER-->
   <!-uri is the OER server-->
   <!- admin is username to login OER in cleartext-->
   <!-password is encrypted-->
   <repository>
      <uri>http://ple-jgau.us.oracle.com:7101/oer</uri>
      <credentials>
         <user>admin</user>
         <password>v2_1.qRhDTl1LdPo=</password><!--run encrypt.bat to encrypt
          this-->
      </credentials>
      <timeout>30000</timeout>
   </repository>
   <!--Query: the files to harvest-->
   <query>
   <!--To specify design-time files to harvest: Uncomment the section below and
    specify the file(s) you want to harvest. Or specify on the command-line
through
    the -file parameter.-->

   <!--To specify run-time files to harvest: Uncomment this and specify the
    file(s) you want to harvest.  Or specify on the command-line through the -file
    parameter. The serverType must be one of: SOASuite, OSB, or WLS. Run
    encrypt.bat to encrypt the password.-->
   <fileQuery>
      <files>/slot/ems3470/oracle/AIAHOME_1022/samples/AIASamples/BaseSample
       /SamplesCreateCustomerPartyPortalProvABCSImpl/composite.xml</files>
      <fileType>.xml</fileType>

   </fileQuery>

   </query>

   <!--Predefined Policy Location: If harvesting SOA Suite projects from the
    command line, uncomment the section below and set it to point to your
    installation of JDeveloper-->
   <introspection>
      <reader>com.oracle.oer.sync.plugin.reader.file.FileReader</reader>
      <writer>com.oracle.oer.sync.plugin.writer.oer.OERWriter</writer>
   </introspection>
</tns:harvesterSettings>
```

2. Access a command line utility and run the AIAHarvest.sh harvester script. Oracle AIA recommends using the `-partial true` option most of the time.

For example, for harvesting using a non-Foundation Pack environment, run `AIAHarvest.sh -partial true -settings HarvesterSettings.xml`

For example, for harvesting using a Foundation Pack environment, run `$AIA_HOME/Infrastructure/LifeCycle/AIAHarvester/AIAHarvest.sh -partial true -settings <file path to destHarvesterSettings.xml>`

5.2.3.2 How to Harvest Design-Time Composites into Project Lifecycle Workbench Only Using HarvesterSettings.xml

To harvest design-time composites into Project Lifecycle Workspace using HarvesterSettings.xml:

1. Ensure that your HarvesterSettings.xml file is accurately configured. Sample file content is provided in Example 5–4.

   In particular, ensure that the sample file content:

   a. Uses `<fileQuery>`.

   b. Uses `-mode AIA` as a command-line parameter.

   c. Does not include Oracle Enterprise Repository server information.

Example 5–4 Sample HarvesterSettings.xml Used to Harvest Design-Time Composites into Project Lifecycle Workbench Only

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:harvesterSettings xmlns:tns="http://www.oracle.com/oer/integration/
 harvester" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.oracle.com/oer/integration/harvester
 Harvester_Settings.xsd ">
   <!--Description to set on created Assets in OER.-->
   <harvesterDescription>Oracle Enterprise Repository
    Harvester</harvesterDescription>

   <!--Registration status to set on created Assets in OER. The Valid
    Registration states are 1) Unsubmitted 2)Submitted - Pending Review
    3)Submitted - Under Review 4)Registered -->
   <registrationStatus>Registered</registrationStatus>
   <!--Namespace to set on created Assets in OER.  If left empty, this is set
    based on information from SOA Suite and OSB projects when available.
    That's generally the best practice, so override this with caution.-->
   <namespace/>

   <!--  comment out the OER server section since we do not perform OER
    harvesting
   <repository>
      <uri>http://ple-jgau.us.oracle.com:7101/oer</uri>
      <credentials>
         <user>admin</user>
         <password>v2_1.qRhDTl1LdPo=</password><!--run encrypt.bat to encrypt
          this-->
      </credentials>
      <timeout>30000</timeout>
   </repository>

   <!--Query: the files to harvest-->
   <query>
      <fileQuery>
```

```
        <files> /slot/ems3470/oracle/AIAHOME_1022/samples/AIASamples/
         BaseSample/SamplesCreateCustomerPartyPortalProvABCSImpl/composite.xml
        </files>
        <fileType>.xml</fileType>
      </fileQuery>

   </query>

   <introspection>
      <reader>com.oracle.oer.sync.plugin.reader.file.FileReader</reader>
      <writer>com.oracle.oer.sync.plugin.writer.oer.OERWriter</writer>
   </introspection>
</tns:harvesterSettings>
```

2. Access a command line utility and execute the harvester script: `AIAHarvest.sh -mode AIA`.

   For example, for harvesting using a non-Foundation Pack environment:
   `AIAHarvest.sh -partial true -mode AIA -settings HarvesterSettings.xml`

   For example, for harvesting using a Foundation Pack environment:
   `$AIA_HOME/Infrastructure/LifeCycle/AIAHarvester/AIAHarvest.sh -partial true -settings` *<file path to harvesterSettings.xml>* `-mode AIA`

5.2.3.3 How to Harvest Design-Time Composites into Project Lifecycle Workspace and Oracle Enterprise Repository Using Command Line Options

To harvest design-time composites into Project Lifecycle Workspace and Oracle Enterprise Repository using command line options:

1. Compose your AIAHarvest.sh script command statement using the available command line options covered in Table 5–1.

Table 5–1 Harvester Command-Line Options

Harvester Command-Line Options	Description
`-settings <file name>`	Configuration settings XML file name
`-url <URL>`	Oracle Enterprise Repository URL
`-user <OER user name>`	Oracle Enterprise Repository user name
`-password <OER password>`	Oracle Enterprise Repository password
`-partial <true | false>`	Provides support for partial introspection. To continue harvesting even if encountering errors, enter *true*. The default value is *false*. In other words, if this option is set to *true*, processing continues even if an imported/included file cannot be accessed. USE WITH CARE.
`-artifact_store <store>`	Name of Oracle Enterprise Repository Artifact Store to look in. If specified, the `-file` argument will be resolved relative to the artifact store URL
`-file <filename or URL>`	File or directory to be harvested. This can be a file name or URL to the file.

Table 5–1 (Cont.) Harvester Command-Line Options

Harvester Command-Line Options	Description
`-file_type <type>`	File type of the file to be harvested. If not specified, it is derived from the file extension.
`-remote_url <URL>`	Running server from which to harvest the remote project, instead of from a file.
`-remote_username <username>`	Username to connect to the remote server
`-remote_password <password>`	Password to connect to the remote server
`-remote_server_type <type>`	Type of remote server: SOASuite, OSB, or WLS,
`-remote_project <type>`	Name of remote project to harvest from the remote server. If omitted, all the projects on the server are harvested.
`-deployment_status <status>`	Deployment status to set on created assets: design-time or run-time.
`-version <version>`	Print version information.
`-help`	Lists Harvester command line options.
`-mode <OER | AIA>`	Enter AIA to run the Project Lifecycle Workbench harvest. Enter OER to run the Oracle Enterprise Repository harvest. Do not provide a value to run both the Project Lifecycle Workbench and Oracle Enterprise Repository harvests.

2. Access a command line utility and issue your harvester command.

   For example: `AIAHarvest -settings <file name>`

5.3 Harvesting Interfaces to Oracle Enterprise Repository in Bulk

This section includes the following topics:

- Section 5.3.1, "How to Set Up Environments to Harvest Interfaces to Oracle Enterprise Repository in Bulk."
- Section 5.3.2, "How to Harvest Interfaces to Oracle Enterprise Repository in Bulk."

5.3.1 How to Set Up Environments to Harvest Interfaces to Oracle Enterprise Repository in Bulk

This section includes the following topics:

- Section 5.3.1.1, "Setting Up to Harvest Interfaces Using a Non-Foundation Pack Environment"
- Section 5.3.1.2, "Setting Up to Harvest Interfaces Using a Foundation Pack Environment"

Objective

Set up environments to be able to harvest release-time interfaces to Oracle Enterprise Repository.

Prerequisites and Recommendations

Oracle AIA Foundation Pack has been installed.

Actor

- Developer
- System administrator

5.3.1.1 Setting Up to Harvest Interfaces Using a Non-Foundation Pack Environment

If you are performing the interface harvest to Oracle Enterprise Repository using an environment other than the one on which Foundation Pack has been installed, perform the following procedure.

To set up to harvest interfaces using a non-Foundation Pack environment:

1. Download AIAHarvester.zip.
2. Unzip the AIAHarvester.zip in any location but maintain the unzipped structure.

> **Note:** The following steps assume that tasks are being performed within the AIAHarvester directory.

3. Adjust the values in bold in ./AIAInstallProperties.xml, as shown in Example 5–5.

   Foundation Pack uses Java Database Connectivity (JDBC) to determine the Project Lifecycle Workbench database. Ensure that the `<jdbc-url>jdbc:oracle:thin:@localhost:1521:XE</jdbc-url>` value points to the Project Lifecycle Workbench database where you want AIAHarvester results to be stored.

Example 5–5 AIAInstallProperties.xml File Adjustments

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<properties>
    <aiainstance>
        <aiaHome>AIA_HOME</aiaHome>
        <name>AIA_INSTANCE_NAME</name>
        <javahome>JAVA_HOME</javahome>
        <remote_install>true</remote_install>
        <domain_root>SERVER_DOMAIN_ROOT</domain_root>
        <mwHome>MW_HOME</mwHome>
        <soaHome>SOA_HOME</soaHome>
        <aiainstalltype>standard</aiainstalltype>
        <isencrypted>true</isencrypted>
    </aiainstance>
    <aialifecycle>
        <jdbc-url>jdbc:oracle:thin:@localhost:1521:XE</jdbc-url>
        <username>FPINST_AIALIFECYCLE</username>
        <password>[C@8b567c</password>
        <createschema>true</createschema>
        <sysusername>sys</sysusername>
        <syspassword>[C@15b0e2c</syspassword>
        <role>SYSDBA</role>
        <defaulttablespace>SYSTEM</defaulttablespace>
        <temptablespace>TEMP</temptablespace>
        <isRac>false</isRac>
        <racCount>1</racCount>
        <racInstances>rac0</racInstances>
```

```
        <rac>
            <serviceName>RAC_DATABASE_SERVICENAME</serviceName>
            <rac0>
                <instanceName>RAC_INSTANCE_NAME</instanceName>
                <host>INSTANCE_HOST</host>
                <port>INSTANCE_PORT</port>
            </rac0>
        </rac>
    </aialifecycle>
...
```

4. Ensure the accuracy of information in the ./META-INF/adf-config.xml file. Under normal circumstances, you should not have to update it.

   Specifically, ensure that the file includes the `jdbc/mds/MDS_LocalTxDataSource` value shown in bold in Example 5–6.

Example 5–6 Sample adf-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<adf-config xmlns="http://xmlns.oracle.com/adf/config" xmlns:adf=
 "http://xmlns.oracle.com/adf/config/properties" xmlns:sec="http://
 xmlns.oracle.com/adf/security/config">
   <adf:adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/
    properties">
       <adf-property name="adfAppUID" value="OER.oracle.apps.aia.oer"/>
   </adf:adf-properties-child>
   <adf-mds-config xmlns="http://xmlns.oracle.com/adf/mds/config">
      <mds-config xmlns="http://xmlns.oracle.com/mds/config">
         <persistence-config>
            <metadata-namespaces>
               <namespace path="/soa/b2b" metadata-store-usage=
                "soa-infra-store"/>
               <namespace path="/deployed-composites" metadata-store-usage=
                "soa-infra-store"/>
               <namespace path="/soa/shared" metadata-store-usage=
                "soa-infra-store"/>
               <namespace path="/apps" metadata-store-usage="soa-infra-store"/>
            </metadata-namespaces>
            <metadata-store-usages>
               <metadata-store-usage id="soa-infra-store" deploy-target="true">
                  <metadata-store class-name="oracle.mds.persistence.stores.db.
                   DBMetadataStore">
                     <property value="soa-infra" name="partition-name"/>
                     <property value="jdbc/mds/MDS_LocalTxDataSource" name=
                      "jndi-datasource"/>
                  </metadata-store>
               </metadata-store-usage>
            </metadata-store-usages>

            <auto-purge seconds-to-live="300"/>
         </persistence-config>

      </mds-config>
   </adf-mds-config>
   <sec:adf-security-child xmlns="http://xmlns.oracle.com/adf/security/config">
      <CredentialStoreContext credentialStoreClass="oracle.adf.share.security.
       providers.jps.CSFCredentialStore" credentialStoreLocation="../../
       src/META-INF/jps-config.xml"/>
   </sec:adf-security-child>
</adf-config>
```

5. Locate the encrypted HarvesterSettings_#.xml files in the Harvester directory. One file is provided for each delivered AIA prebuilt interface. Modify each of the HarvesterSettings_#.xml files to point to the Oracle Enterprise Repository server for your particular environment and to include relevant user name and password values.

   The HarvesterSettings_#.xml files always uses the `oramds://` protocol, as shown in Example 5–7.

Example 5–7 oramds:// Protocol Used in HarvesterSettings_#.xml

```
<fileQuery>
   <files>oramds://apps/AIAMetaData/AIAComponents/EnterpriseBusinessService
    Library/Industry/BankingAndWealthManagement/EBO/SpecificationValueSet/V1/
    BankingAndWealthManagementSpecificationValueSetEBSV1.wsdl</files>
   <fileType>.wsdl</fileType>
</fileQuery>
```

   To publish your own interface artifacts (namely, WSDL and XSD files) in bulk, compose your own HarvesterSettings_#.xml files using the delivered files as samples. Specify your artifact paths and file types in the `<fileQuery>` element.

   Oracle AIA recommends using the MDS protocol (you must have populated MDS with interface artifacts); however, you can also use the fileSystem protocol. Depending on whether you are using MDS or fileSystem, provide the file path or oramds path to the WSDL or XSD files.

   For more information about populating MDS with interface artifacts, see Chapter 2, "Building AIA Integration Flows."

6. Run an encryption script on each HarvesterSettings_#.xml file to produce the final HarvesterSettings.xml files, which you will use as the input files to the harvester script that harvests your interfaces in bulk to Oracle Enterprise Repository: `./Harvester/encrypt.sh <source_file> <target_file>`

   For example: `./Harvester/encrypt.sh harvesterSettings_prep.xml HarvesterSettings.xml`

   The harvesterSettings_prep.xml and HarvesterSettings.xml file names are used for illustrative purposes only. The source and target file names can be set to any file name.

5.3.1.2 Setting Up to Harvest Interfaces Using a Foundation Pack Environment

If you are performing the interface harvest to Oracle Enterprise Repository using the environment on which Foundation Pack has been installed, this set up has been performed by the AIA Installer.

To set up to harvest interfaces using a Foundation Pack environment:

1. Run the following script: `source ${AIA_INSTANCE}/bin/aiaenv.sh`
2. Ensure that the HarvesterSettings_#.xml files are located in the $AIA_INSTANCE/Infrastructure/LifeCycle/Install/FPHarvest directory. One file is provided for each delivered AIA prebuilt interface.
3. Modify each of the HarvesterSettings_#.xml files to point to the Oracle Enterprise Repository server for your particular environment and to include relevant user name and password values in clear text.

4. Encrypt the HarvesterSettings_#.xml files in the $AIA_INSTANCE/Infrastructure/LifeCycle/Install/FPHarvest directory.

   For example:
   ```
   ${AIA_HOME}/Infrastructure/LifeCycle/AIAHarvester/Harvester/encrypt.sh <source_file> <target_file>
   ```

5.3.2 How to Harvest Interfaces to Oracle Enterprise Repository in Bulk

Objective

Harvest interfaces to Oracle Enterprise Repository in bulk.

Prerequisites and Recommendations

- Complete the steps covered in Section 5.1, "How to Set Up Oracle AIA Content Harvesting."
- Complete the steps covered in Section 5.3.1, "How to Set Up Environments to Harvest Interfaces to Oracle Enterprise Repository in Bulk."

Actor

- Development
- System administrator

To harvest interfaces to Oracle Enterprise Repository in bulk:

1. Access a command line utility and run the AIAHarvest.sh harvester script located in ${AIA_HOME}/Infrastructure/LifeCycle/AIAHarvester/. Oracle AIA recommends using the `-partial true` option most of the time.

   For example:

   - `AIAHarvest.sh -partial true -settings HarvesterSettings1.xml`
   - `AIAHarvest.sh -partial true -settings HarvesterSettings2.xml`
   - `AIAHarvest.sh -partial true -settings HarvesterSettings3.xml`

2. Access Oracle Enterprise Repository to ensure that your interfaces are visible.

   For more information, see Chapter 11, "Configuring and Using Oracle Enterprise Repository as the Oracle AIA SOA Repository."

5.4 Harvesting Deployed Composites into Oracle Enterprise Repository

This section includes the following topics:

- Section 5.4.1, "How to Set Up Environments to Harvest Deployed Composites into Oracle Enterprise Repository."
- Section 5.4.2, "How to Harvest Deployed Composites into Oracle Enterprise Repository."

Run-time publishing to Oracle Service Registry is delegated to Oracle Enterprise Repository.

For more information about publishing to Oracle Service Registry from Oracle Enterprise Repository, see "Configuring Oracle Enterprise Repository to Exchange Metadata with the Oracle Service Registry" in *Oracle Fusion Middleware Configuration Guide for Oracle Enterprise Repository.*

5.4.1 How to Set Up Environments to Harvest Deployed Composites into Oracle Enterprise Repository

Objective

Set up environments to harvest deployed composites into Oracle Enterprise Repository.

Prerequisites and Recommendations

- The Oracle AIA Installer installation has been run and completed.
- The composites that you want to harvest into Oracle Enterprise Repository have been deployed into the run-time SOA engine.

Actors

System administrator

Specifically, this is the actor who installs and deploys the AIA Foundation Pack and Pre-Built Integrations.

To set up environments to harvest deployed composites into Oracle Enterprise Repository:

Run an AIA command to set up the environment: `Source ${AIA_INSTANCE}/bin/aiaenv.sh`

5.4.2 How to Harvest Deployed Composites into Oracle Enterprise Repository

This section includes the following topics:

- Section 5.4.2.1, "Harvesting Pre-Built Integration-Delivered Deployed Composites to Oracle Enterprise Repository"
- Section 5.4.2.2, "Harvesting Custom-Built Deployed Composites to Oracle Enterprise Repository"

Objective

As a part of your Oracle AIA installation, you have deployed composites into the Oracle SOA engine, whether they are AIA-delivered or custom built. Post installation, you can choose to publish these deployed composites to the Oracle Enterprise Repository using a command-line script.

Note: This post- installation harvest of deployed composites into Oracle Enterprise Repository is optional and is not performed by the AIA Installer.

Publishing your deployed composites to Oracle Enterprise Repository ensures that it accurately reflects your SOA run-time composite information, including end points and so forth.

Prerequisites and Recommendations

- Complete the steps covered in Section 5.1, "How to Set Up Oracle AIA Content Harvesting."
- Complete the steps covered in Section 5.4.1, "How to Set Up Environments to Harvest Deployed Composites into Oracle Enterprise Repository."

Actors

System administrator

Specifically, this is the actor who installs and deploys the AIA Foundation Pack and Pre-Built Integrations.

5.4.2.1 Harvesting Pre-Built Integration-Delivered Deployed Composites to Oracle Enterprise Repository

If you are harvesting deployed composites that were delivered as part of an AIA Pre-Built Integration, perform the following procedure to harvest them to Oracle Enterprise Repository.

To harvest Pre-Built Integration-delivered deployed composites into Oracle Enterprise Repository:

1. Navigate to $<AIA_HOME>/Infrastructure/LifeCycle/Install/Pre-Built IntegrationHarvest.
2. Run the following command: `ant -f PIPCompositeHarvest.xml`
3. Access Oracle Enterprise Repository to confirm that all expected composites are present.

   For more information, see Section 11.3, "How to Access Oracle AIA Content in Oracle Enterprise Repository."

5.4.2.2 Harvesting Custom-Built Deployed Composites to Oracle Enterprise Repository

If you are harvesting custom-built deployed composites, perform the following procedure to harvest them to Oracle Enterprise Repository.

For example, if you have installed Foundation Pack and implemented and deployed your own composites, you can choose to harvest these deployed composites to Oracle Enterprise Repository.

To harvest custom-built deployed composites into Oracle Enterprise Repository:

1. Edit the HarvesterSettings.xml file, as shown in Example 5–8, to provide content in bold specific to your harvesting requirements.
   - Provide Oracle Enterprise Repository information.
   - Use `<remoteQuery>` and list all deployed composite names to be harvested.
   - As part of the `<remoteQuery>` syntax, provide and encrypt SOA server information.

Example 5–8 Sample HarvesterSettings.xml Used to Harvest Custom-Built Deployed Composites into Oracle Enterprise Repository:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:harvesterSettings xmlns:tns="http://www.oracle.com/oer/integration/
 harvester"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"xsi:
```

```
schemaLocation="http://www.oracle.com/oer/integration/harvester
Harvester_Settings.xsd">
  <!--Description to set on created Assets in OER.-->
  <harvesterDescription>Oracle Enterprise Repository Harvester
  </harvesterDescription>
  <!--Registration status to set on created Assets in OER. The Valid
   Registration states are 1) Unsubmitted 2)Submitted - Pending Review
   3)Submitted - Under Review  4)Registered -->
  <registrationStatus>Registered</registrationStatus>
  <!--Namespace to set on created Assets in OER.  If left empty, this is set
   based on information from SOA Suite and OSB projects when available.
   That's generally the best practice, so override this with caution.-->
  <namespace/>


  <!--Connection info to OER-->
  <!-uri is the OER server-->
  <!- admin is username to login OER in cleartext -->
  <!-password is encrypted -->
  <repository>
     <uri>http://ple-jgau.us.oracle.com:7101/oer</uri>
     <credentials>
        <user>admin</user>
        <password>v2_1.qRhDTl1LdPo=</password><!--run encrypt.bat to
         encrypt this-->
     </credentials>
      <timeout>30000</timeout>
  </repository>
  <!--Query: the files to harvest-->
  <query>
   <!--To specify design-time files to harvest: Uncomment the section below and
    specify the file(s) you want to harvest.  Or specify on the command-line
    through the -file parameter.-->

   <!--To specify run-time files to harvest: Uncomment this and specify the
    file(s) you want to harvest.  Or specify on the command-line through the
-file
    parameter. The serverType must be one of: SOASuite, OSB, or WLS. Run
  encrypt.bat to encrypt the password.-->
     <remoteQuery>
        <serverType>SOASuite</serverType>

         <!-This is the composite project deployed into the SOA server-->
        <projectName>AIADemoQueryCustomerPartyCRMProvABCSImpl
        </projectName>
         <!-another example below, we specifically give the
         deployment revision number rev1.0. It is appended after the
          composite project name. User may find out this revision number of
          browsing MDS through their jDev
         <projectName>AIADemoQueryCustomerPartyCRMProvABCSImpl_rev1.0
         </projectName>-->
         <!-uri is the SOA server information -->
         <!-user is the username to log in to the SOA server -->
         <!-password is used to log in to the SOA server, it is encrypted -->
        <uri>http://10.146.91.163:8001/</uri>
        <credentials>
           <user>weblogic</user>
           <password>v2_1.G+NTr3az8thaGGJBn0vwPg==</password>
        </credentials>
     </remoteQuery>
```

```
    </query>

    <!--Predefined Policy Location: If harvesting SOA Suite projects from
     the command line, uncomment the section below and set it to point to your
     installation of JDeveloper-->
   <introspection>
     <reader>com.oracle.oer.sync.plugin.reader.file.FileReader</reader>
     <writer>com.oracle.oer.sync.plugin.writer.oer.OERWriter</writer>
   </introspection>
</tns:harvesterSettings>
```

2. Access a command line utility and run the AIAHarvest.sh harvester script. Oracle AIA recommends using the `-partial true` option.

   For example: `AIAHarvest.sh -partial true -mode OER -settings HarvesterSettings.xml`

5.5 Introducing Oracle Enterprise Repository After AIA Installation

Oracle Enterprise Repository is a product separate from AIA and is an optional component in the context of AIA installation and execution. Therefore, Oracle Enterprise Repository possibly may not be present at the time of AIA installation.

To adopt Oracle Enterprise Repository after AIA has been installed, perform the following procedure.

To introduce Oracle Enterprise Repository after AIA installation:

1. Install Oracle Enterprise Repository according to Oracle Enterprise Repository guidelines.
2. Import the Oracle Enterprise Repository solution pack and the AIA asset definition file.

   For more information about performing these imports, see Section 5.1, "How to Set Up Oracle AIA Content Harvesting."
3. Install the AIA solution pack for your release. This AIA solution pack is made available as a post-release patchset.

   For more information about the AIA solution pack, see Section 11.1, "Introduction to Using Oracle Enterprise Repository as the Oracle AIA SOA Repository."
4. Deploy the AIALifeCycleArtifactsLink.war redirect servlet to facilitate the Business Publisher-to-Oracle Enterprise Repository redirect feature.

   For more information, see "Configuring the AIA Redirect Servlet" in *Oracle Application Integration Architecture Foundation Pack: Installation Guide.*
5. Run the following command line script to register Oracle Enterprise Repository with AIA: `AIAOerEncode.sh -url <OER Server URL> -user <OER Username>`

   When prompted, enter the Oracle Enterprise Repository password.

6

Working with Project Lifecycle Workbench Bills of Material

This chapter provides an overview of Bills of Materials and how to work with Bill of Materials for an AIA Lifecycle Project.

This chapter includes the following sections:

6.1 Introduction to Bills of Material

Following the Oracle Application Integration Architecture (AIA) development lifecycle, required functionality for a Pre-Built Integration is implemented as a series of composites.

For more information about using the Service Constructor to generate these composites, see Chapter 4, "Working with Service Constructor."

After these design-time composites are complete, they are harvested into the Project Lifecycle Workbench.

For more information about harvesting design-time composites into the Project Lifecycle Workbench, see Section 5.2, "Harvesting Design-Time Composites into Project Lifecycle Workbench and Oracle Enterprise Repository".

At this point, you can use the Project Lifecycle Workbench to create a bill of material (BOM) for a Pre-Built Integration. The BOM specifies the inventory of composites that constitute the Pre-Built Integration. Using this feature can help expedite deployment plan generation for a Pre-Built Integration project by enabling the automatic or semiautomatic generation of the Pre-Built Integration's BOM.

For more information about generating a deployment plan using the Deployment Plan Generator, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts"

The BOM options that appear for a user in the Project Lifecycle Workbench depend on the Project Lifecycle Workbench role assigned to their WebLogic user.

For more information about setting up Project Lifecycle Workbench roles, see "Setting up AIA Roles" in *Oracle Application Integration Architecture Foundation Pack: Installation Guide.*

AIALifecycleUser and AIALifecycleDeveloper roles are provided with the following BOM options on the Project page:

- If the BOM has not yet been generated, the No Bill Of Material Exists message displays.
- If the BOM has been generated, the View Bill Of Material link displays.

  The user can click the link to view the BOM.

For more information about viewing BOMs, see Section 6.4, "How to View a Bill of Material for an AIA Lifecycle Project."

The AIALifecycleInstallDeveloper role is provided with the following BOM options on the Project page:

- If the BOM has not yet been generated, the Generate Bill Of Material link displays.

  The user can click the link to preview or generate the BOM.

  For more information about generating BOMs, see Section 6.2, "How to Generate a Bill of Material for an AIA Lifecycle Project."
- If BOM has been generated, the Edit Bill Of Material link displays.

  The user can click the link to edit the BOM.

  For more information about editing BOMs, see Section 6.3, "How to Edit a Bill of Material for an AIA Lifecycle Project."

Automatic generation of the BOM is possible if you have implemented AIA-recommended annotations when constructing composites and have harvested completed design-time composites to Project Lifecycle Workbench.

Generation of the BOM becomes semiautomatic if you have not implemented AIA-recommended annotations when constructing composites and have not harvested completed design-time composites to Project Lifecycle Workbench. In this case, manual intervention is necessary to generate the BOM.

For more information about AIA-recommended annotations, see Section 5.2, "Harvesting Design-Time Composites into Project Lifecycle Workbench and Oracle Enterprise Repository."

Along with prebuilt AIA Pre-Built Integrations, Oracle delivers the BOMs for these Pre-Built Integrations. You can import this BOM seed data into your system so that you can use the Project Lifecycle Workbench to revise the BOMs in accordance with any modifications you make to the prebuilt Pre-Built Integrations.

For more information about importing BOM seed data, see Section 7.4, "How to Import Seed Data."

6.2 How to Generate a Bill of Material for an AIA Lifecycle Project

Objective

Generate a BOM from an AIA Lifecycle project. The BOM captures all in-scope tasks, their ensuing composites and metadata.

The generated BOM output, an XML file, contains all composite information, which collectively makes up a project and is useful for configuring a deployment plan using the Deployment Plan Generator.

For more information about generating a deployment plan using the Deployment Plan Generator, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

Prerequisites and Recommendations

In the AIA development lifecycle, as with any software development cycle, deployment plan generation occurs near the end of the development cycle. Do not generate BOM and generate deployment plan based on the BOM until you are certain that you are no longer required to alter the functional decomposition and project definition.

Likewise, the composites for the Pre-Built Integration should have been implemented and, ideally, harvested into the Project Lifecycle Workbench before approaching BOM generation.

For more information about using the Service Constructor to generate these composites, see Chapter 4, "Working with Service Constructor."

For more information about harvesting composites into the Project Lifecycle Workbench, see Section 5.2, "Harvesting Design-Time Composites into Project Lifecycle Workbench and Oracle Enterprise Repository."

Note: If you have not annotated composites and you have not performed harvesting, you can still use BOM generation functionality by generating the BOM for your project and manually building BOM with annotations using the instructions in this chapter.

Actors

Only users assigned the AIALifecycleInstallDeveloper role can generate and modify a BOM for a Pre-Built Integration project. This user should be a developer who is responsible for producing the deployment plan for all executables in the project for which the BOM is being generated. This developer will therefore also have a solid overall picture of the project.

To generate a BOM for an AIA Lifecycle project:

1. Access the AIA Home Page. Click **Go** in the Project Lifecycle Workbench area. Select the **Project** tab.
2. Click the **Generate Bill Of Material** link.

   This option is available only if your user has the AIALifecycleInstallationDeveloper role assigned and the BOM has not been generated.

   a. If the BOM has been generated and you have been assigned the AIALifecycleInstallationDeveloper role, you will see the **Edit Bill of Material** link. Click it to view and edit and generated BOM.

      For more information about editing BOMs, see Section 6.3, "How to Edit a Bill of Material for an AIA Lifecycle Project."

   b. If the BOM has been generated and you have not been assigned the AIALifecycleInstallationDeveloper role, you will see the **View Bill Of Material** link. Click it to view the BOM on the Bill Of Material page.

      For more information about viewing BOMs, see Section 6.4, "How to View a Bill of Material for an AIA Lifecycle Project."

3. The application prompts you with two choices:

   a. Click **Generate** to generate the BOM for the Pre-Built Integration project and access it on the Bill Of Material page.

Click this option only if the functional decomposition and project definition have truly been completed.

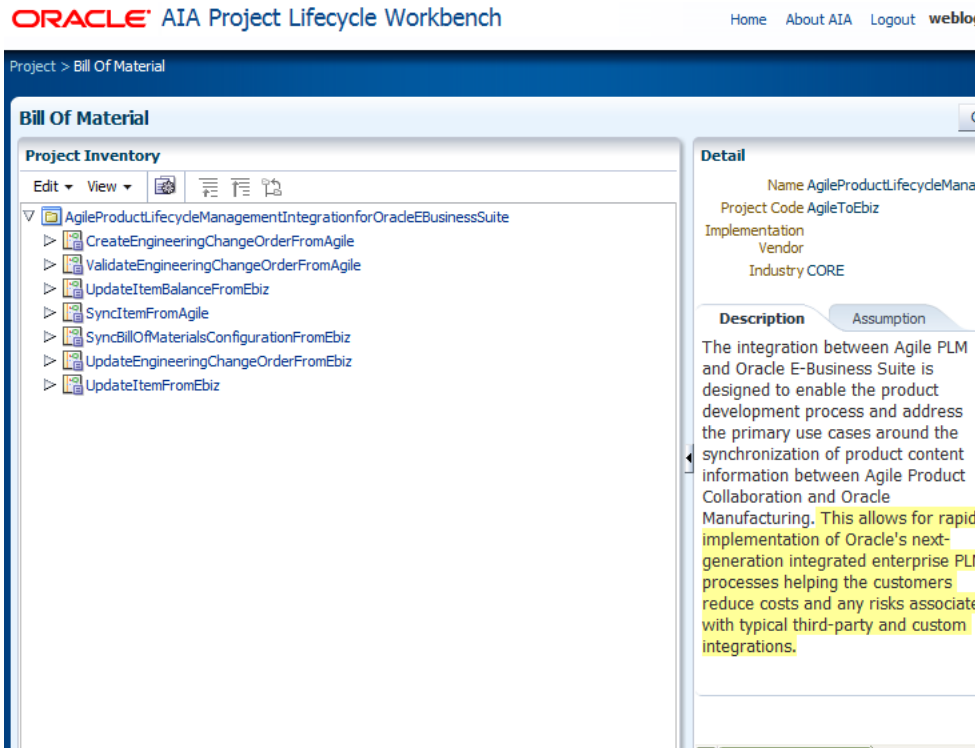
Note: After you generate the BOM for a project, any changes that are made to the BOM on the Bill Of Material page are not reflected in the original project definition and decomposition in the Project Lifecycle Workbench.

---

b. Click **Preview** to view the BOM. This option is available only before the BOM has been generated.

4. The generated BOM displays on the Bill Of Material page. Edit the BOM as necessary and click the **Export BOM** button, as shown in Figure 6–1 to export the BOM as an XML file to be supplied as input to the Deployment Plan Generator to expedite deployment plan generation.

Figure 6–1 Export BOM Button



The recommended location for the saved BOM XML file is AIA_HOME/aia_instances/<instance_name>/config.

The recommended file name for the saved BOM XML file is <Custom_Pre-Built Integration_Name>BOM.xml.

For more information about supplying the BOM XML file to the Deployment Plan Generator, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

For more information about editing BOM, see Section 6.3, "How to Edit a Bill of Material for an AIA Lifecycle Project."

6.3 How to Edit a Bill of Material for an AIA Lifecycle Project

Objective

Edit a BOM for an AIA Lifecycle Project.

Prerequisites and Recommendations

The BOM must have been generated.

For more information about generating BOMs, see Section 6.2, "How to Generate a Bill of Material for an AIA Lifecycle Project."

> **Note:** Any changes that you make to the BOM on the Bill Of Material page are not reflected in the original project definition and decomposition in the Project Lifecycle Workbench.

Actors

Only users assigned the AIALifecycleInstallationDeveloper role can edit a BOM for a Pre-Built Integration project. This user should be a developer who is responsible for producing the deployment plan for all executables in the project for which the BOM is being generated. This developer will therefore also have a solid overall picture of the project.

To edit a BOM for an AIA Lifecycle Project:

1. Access the AIA Home Page. Click **Go** in the Project Lifecycle Workbench area. Select the **Project** tab.
2. Click the **Edit Bill Of Material** link.

   This option is available only if your user has the AIALifecycleInstallationDeveloper role assigned and the BOM has been generated.

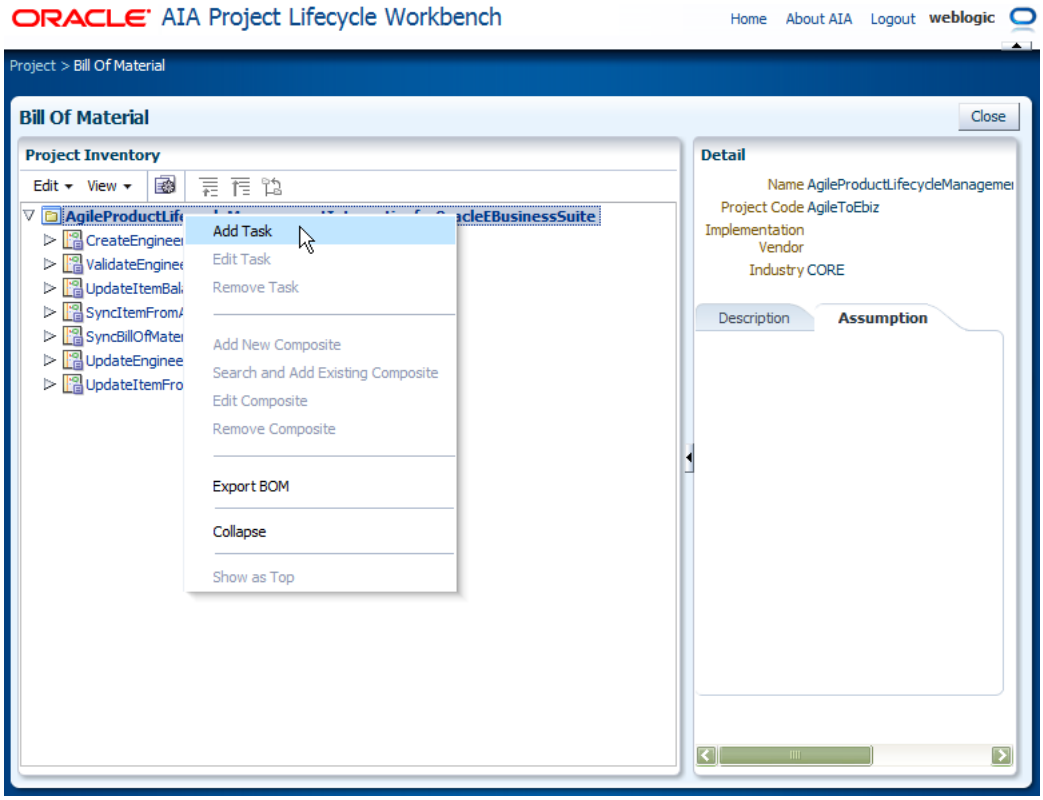
   - If the BOM has not been generated and you have been assigned the AIALifecycleInstallationDeveloper role, you will see the **Generate Bill Of Material** link.

     For more information about generating BOMs, see Section 6.2, "How to Generate a Bill of Material for an AIA Lifecycle Project."

   - If the BOM has been generated and you have not been assigned the AIALifecycleInstallationDeveloper role, you will see the **View Bill Of Material** link. Click it to view the BOM on the Bill Of Material page.

     For more information about viewing BOMs, see Section 6.4, "How to View a Bill of Material for an AIA Lifecycle Project."
3. The BOM displays on the Bill Of Material page.
4. Right-click the project root node as shown in Figure 6–2 to view permitted actions.

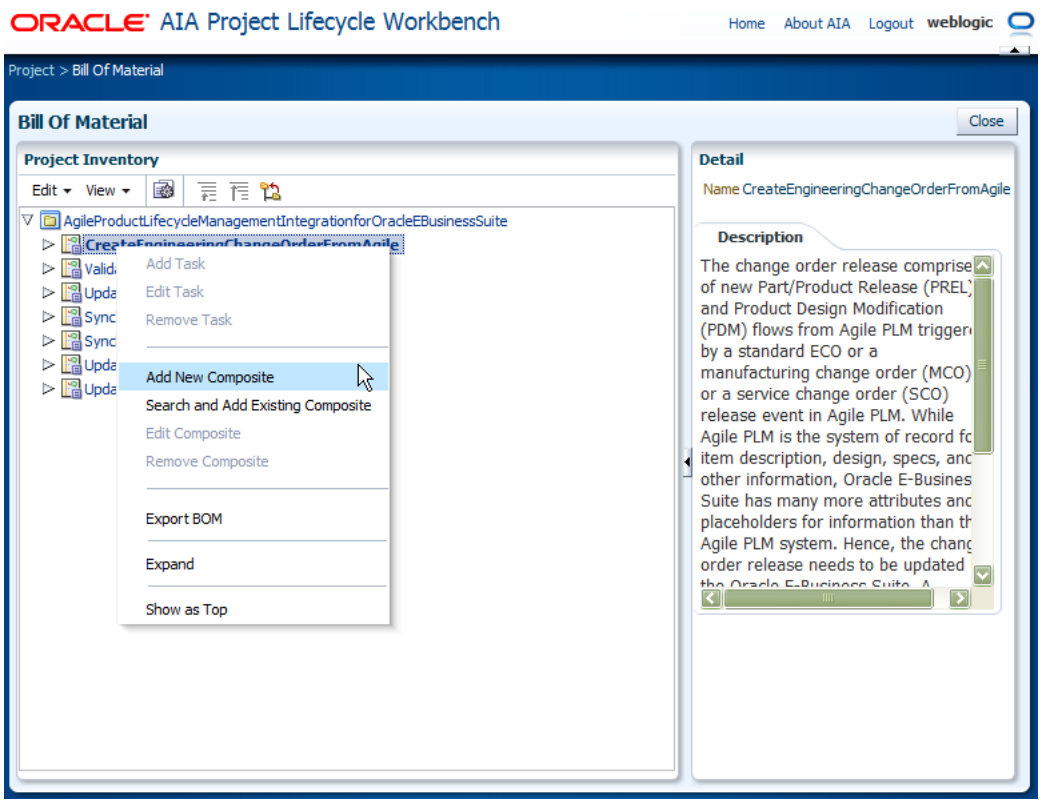
Figure 6–2 Project Root Node Actions on the Bill Of Materials Page



Select **Add Task**.

5. Right-click the task branch node as shown in Figure 6–3 to view permitted actions.

Figure 6–3 Task Branch Node Actions on the Bill Of Materials Page



A task represents a chunk of functionality. The composites associated with a task collectively realize the business logic embodied by the task.

- The Edit option is available for a task branch node if the task was added to the project root node of the generated BOM. The Edit option is not available if the task was added to the project before BOM generation through the functional decomposition pages.
- The Remove option is available for a task branch node if the task was added to the project root node of the generated BOM. The Remove option is not available if the task was added to the project before BOM generation through the functional decomposition pages.
- Select **Add New Composite** to manually define a composite and its annotations and to associate it with the task. Enter composite details in the **Detail** area that displays.

Note: The newly added composite and its ensuing annotations are not harvested from composite source code. Instead, they are manually added through the BOM UI and are stored only in the Project Lifecycle Workbench.

You may subsequently re-harvest the added composite directly from source code. At this point, the annotations previously entered through the UI are overwritten by the harvested data. The Add New Composite option is always available.

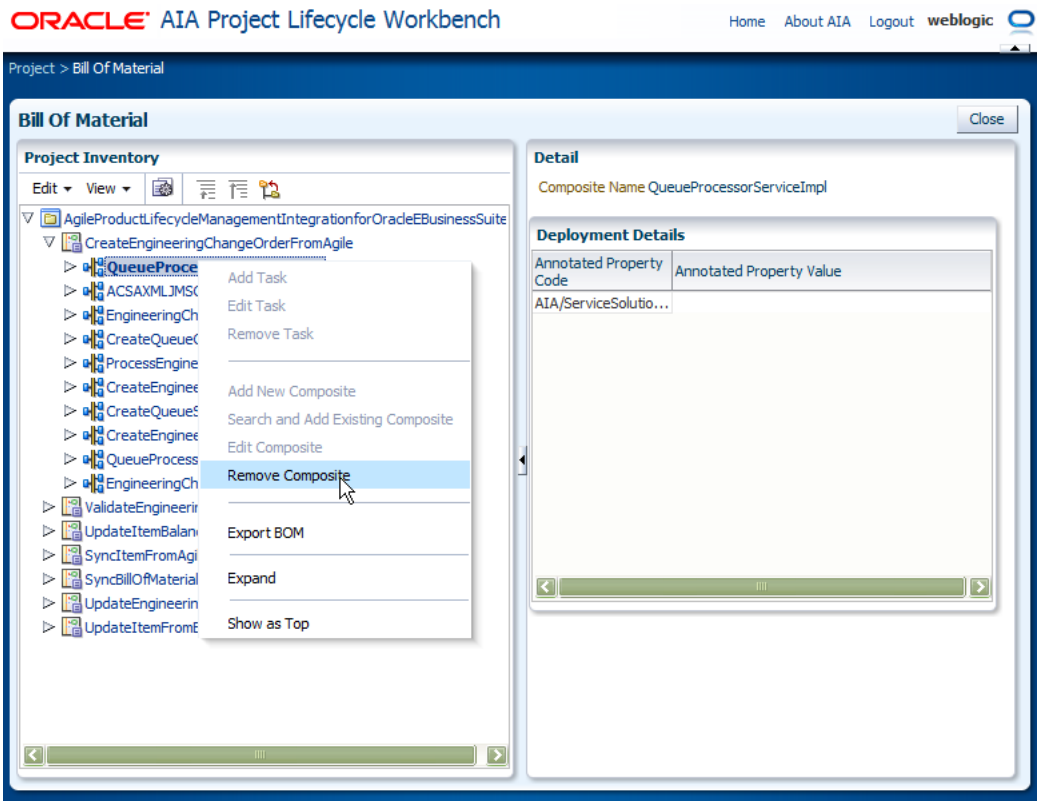
- Select **Search and Add Existing Composite** to access the Search and Add Existing Composite page, where you can search for and locate existing composites and associate one or more with the task.

  For existing composites to be available for selection, they must have been harvested by running AIAHarvester.sh. Upon selecting one or more existing composites, any available composite details and annotations appear in the Detail area that appears. The Search and Add Existing Composite option is always available.

For more information about running AIAHarvester.sh, see Section 5.2, "Harvesting Design-Time Composites into Project Lifecycle Workbench and Oracle Enterprise Repository."

6. Right-click the composite leaf node to view permitted actions, as shown in Figure 6–4.

Figure 6–4 Composite Leaf Node Actions on the Bill Of Materials Page



Multiple composites collectively provide functionality encompassed by a task. A composite can be defined and included in a BOM in either of two ways:

- Harvested composites

  A composite can be harvested upon completion of its implementation by running the AIAHarvester.sh command. A harvested composite can then be added to a generated BOM by selecting the **Search and Add Existing Composite** option on the task branch node.

  For more information about running AIAHarvester.sh, see Section 5.2, "Harvesting Design-Time Composites into Project Lifecycle Workbench and Oracle Enterprise Repository."

- Manually defined composites

  The composite was not harvested, but rather was manually defined as an associated composite by selecting the **Add New Composite** option on the task branch node.

If you have defined a composite manually, select the composite leaf node to display the Detail area where you can edit composite-level annotations by specifying a series of XPath values in the BOM. The internal structure of the composite (services and references, for example) is not shown in the BOM. All annotations are flat for manually defined composites.

For both harvested and manually defined composites, select the **Remove** option to disassociate the composite from its parent task branch node. This removes the composite definition and association from the BOM. For harvested composites, this does not remove the composite from the Project Lifecycle Workbench database or the Oracle Enterprise Repository. For manually defined composites, this does not remove the composite from the Project Lifecycle Workbench database.

7. After you have finalized all details conveyed in the BOM, click **Export BOM** to export the BOM as an XML file to be supplied as input to the Deployment Plan Generator to expedite deployment plan generation.

   The recommended location for the saved BOM XML file is AIA_HOME/aia_instances/<instance_name>/config.

   The recommended file name for the saved BOM XML file is <Custom_Pre-Built Integration_Name>BOM.xml.

   For more information about supplying the BOM XML file to the Deployment Plan Generator, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

6.4 How to View a Bill of Material for an AIA Lifecycle Project

Objective

View a BOM from a Project Lifecycle Workbench project. The BOM captures project details, and the tasks defined as being in-scope for the project.

Prerequisites and Recommendations

The BOM to be viewed must have been generated.

Actors

The ability to view a BOM using the View Bill Of Material link is available only to a user with the AIALifeCycleUser or AIALifeCycleDeveloper role assigned.

Users with the AIALifeCycleInstallDeveloper role assigned will not see the View Bill Of Material link. Instead, they see the Generate Bill Of Material link or Edit Bill Of Material link. If the Generate Bill Of Material link displays, a user can click the link and select the **Preview** option to view the BOM. If the Edit Bill Of Material link displays, the user can click it to view and edit the BOM.

For more information about generating and editing the BOM as a user with the AIALifeCycleInstallDeveloper role assigned, see Section 6.2, "How to Generate a Bill of Material for an AIA Lifecycle Project" and Section 6.3, "How to Edit a Bill of Material for an AIA Lifecycle Project."

To view a BOM for an AIA Lifecycle project:

1. Access the AIA Home Page. Click **Go** in the Project Lifecycle Workbench area. Select the **Project** tab.
2. Click the **View Bill Of Material** link to view the BOM on the Bill Of Material page. The **View Bill Of Material** link does not display if the BOM has not yet been generated. The **No Bill Of Material Exists** message displays instead.

   Depending on the lifecycle phase of the BOM, the Bill Of Material page may or may not display information.

   For example, if the functional decomposition and service solution component definition are complete, but no composites have been implemented and nothing has been harvested, not much information appears on the page.

   However, if functional decomposition, service solution component definition, actual composite implementation, and composite harvesting have been completed, the BOM tree displays to enable you to click through and view the functional decomposition.
3. Select nodes in the tree to display any available details and annotations for the selected node.
   - Project root node: Reflects the name of the project from which the BOM was generated.
   - Task branch nodes: Reflect the tasks that form the project from which the BOM was generated.
   - Composite leaf nodes: Reflect the composites that collectively implement the functionality of the tasks.

     A composite consists of child elements, such as services, references, and so forth. As such, leaf nodes do not display child elements if annotations are not present for their child elements.

7

Working with Project Lifecycle Workbench Seed Data

This chapter provides an overview of Project Lifecycle Workbench Seed Data and describes how to set up an environment to export or import see data.

This chapter includes the following sections:

- Section 7.1, "Introducing Project Lifecycle Workbench Seed Data"
- Section 7.2, "How to Set Up an Environment to Export or Import Seed Data for the First Time"
- Section 7.3, "How to Export Seed Data"
- Section 7.4, "How to Import Seed Data"

7.1 Introducing Project Lifecycle Workbench Seed Data

In Project Lifecycle Workbench, seed data can be categorized into two areas:

- Bill of material (BOM) data

  Bill of material seed data contains information about the SOA composites in a Project Lifecycle Workbench project. You can produce a BOM.xml file from the Project Lifecycle Workbench user interface (UI) based on seed data residing in the Project Lifecycle Workbench backend.

  For more information about generating BOMs, see Section 6.2, "How to Generate a Bill of Material for an AIA Lifecycle Project."

- Functional decomposition data

  Functional decomposition is a task performed by functional experts or solution architects that involves an overall analysis of an process integration's business logic. The Project Lifecycle Workbench can be used to facilitate this task.

  Functional decomposition data includes a set of requirements and intentions that have been defined in Project Lifecycle Workbench to articulate the business logic of a given process integration. This seed data is composed of projects, tasks, and service solution components.

  For example, a Project Lifecycle Workbench service solution component is a product of functional decomposition that represents an intent to fulfill a chunk of business logic in the form of a service. At the time of its creation in Project Lifecycle Workbench, the service solution component may or may not correspond to an existing executable service.

You can use the seed data created by your organizations work in Project Lifecycle Workbench to capture and migrate data from one Project Lifecycle Workbench to another.

Oracle delivers BOM and functional decomposition seed data with its Pre-Built Integration products. You can clone this Oracle-delivered data and modify the projects, adding custom logic and artifacts. Cloning the delivered seed data ensures that subsequent AIA upgrades and patches do not tamper with your customized data.

After cloning the data, you can do the following types of updates to the data:

- Modify an existing composite and its associated annotations.
- Add a composite to the cloned BOM, if the composite has been developed based on the AIA programming models and annotation standards.
- Remove a composite from the cloned BOM.

With this seed data, you can go through your development cycle, modifying Pre-Built Integration project definitions and composites, generate BOMs, and from the BOMs, generate deployment plans.

For more information about generating deployment plans, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

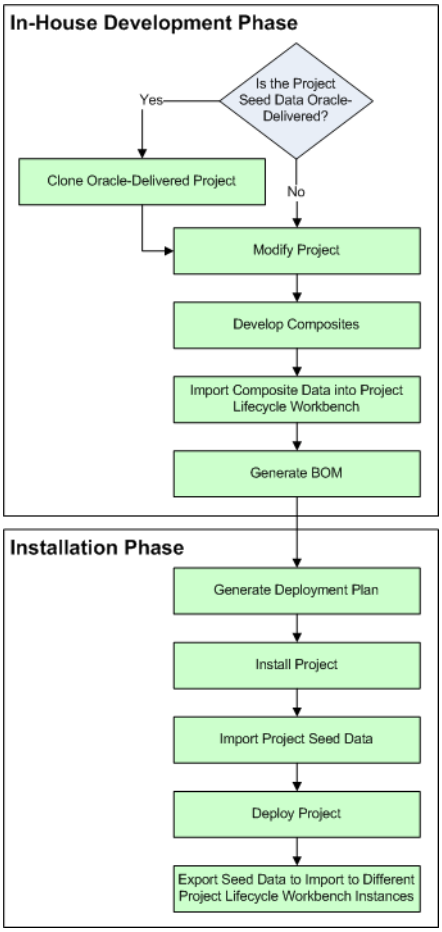
Oracle-delivered seed data can be imported into a Project Lifecycle Workbench instance as a part of the overall Pre-Built Integration installation process, or the data can be imported using a standalone post-installation script, depending on the needs of your implementation.

To determine the phase in which you should import your seed data, consider that deployment plans are only meaningful when the underlying executables, such as composite source code, are available for installation, deployment, and execution. For this reason, determine the phase in which to import seed data by determining when the executable source code will be ready for deployment on the target file system.

For more information about how to load Project Lifecycle Workbench seed data post-installation, see Section 7.4, "How to Import Seed Data"

Figure 7–1 provides a sample usage flow for the import and export of Project Lifecycle Workbench seed data.

Figure 7–1 Sample Usage Flow for Import and Export of Project Lifecycle Workbench Seed Data



Project Lifecycle Workbench does not capture non-native artifacts. Deployment of Oracle-delivered non-native artifacts can be manually specified using SupplementaryDP.xml. Deployment of custom non-native artifacts can be manually specified using CustomSupplementaryDP.xml.

For more information about how to deploy non-native artifacts using SupplementaryDP.xml and CustomSupplementaryDP.xml, see Section 8.2.2, "Extending Non-Native Artifacts."

The Project Lifecycle Workbench seed data schema is provided in Example 7–1.

Example 7–1 Project Lifecycle Workbench Seed Data Schema

```
<?xml version="1.0" encoding="windows-1252"?>
<!-- edited with XMLSpy v2006 rel. 3 sp2 (http://www.altova.com) by Siebel Systems (Siebel
Systems) -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://xml.oracle.com
 /AIA/PLWBOM/V1" targetNamespace="http://xml.oracle.com/AIA/PLWBOM/V1"
 elementFormDefault="qualified">
   <xsd:element name="PLW">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Process">
               <xsd:annotation>
                  <xsd:documentation>Project LifeCycle WorkBench Seed Data
```

```
         Schema</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="ProcessDetail" type="ProcessType"/>
            <xsd:element name="ProcessUuid" type="xsd:string"/>
            <xsd:element name="ParentProcessUuid" type="xsd:string"/>
            <xsd:element name="Task" minOccurs="0" maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="TaskDetail" type="TaskType"/>
                     <xsd:element name="Scope" type="xsd:string"/>
                     <xsd:element name="SvccompId" type="xsd:string" minOccurs="0"
                      maxOccurs="unbounded"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
            <xsd:element name="ProcessBom" minOccurs="0">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="ProcessBomDetail" type="ProcessType"/>
                     <xsd:element name="BomStatus" type="xsd:string"/>
                     <xsd:element name="TaskBom" minOccurs="0" maxOccurs="unbounded">
                        <xsd:complexType>
                           <xsd:sequence>
                              <xsd:element name="TaskBomDetail" type="TaskType"/>
                              <xsd:element name="TaskBomSource" type="xsd:string"/>
                              <xsd:element name="SvccompBomId" type="xsd:string"
                               minOccurs="0" maxOccurs="unbounded"/>
                           </xsd:sequence>
                        </xsd:complexType>
                     </xsd:element>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
            <xsd:element name="SvccompBom" minOccurs="0" maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="SvccompBomDetail" type="SvccompType"/>
                     <xsd:element name="SvccompBomId" type="xsd:string"/>
                     <xsd:element name="CompositeId" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Svccomp" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="SvccompDetail" type="SvccompType"/>
            <xsd:element name="ReusedServiceObjectType" type="xsd:string"
           minOccurs="0"/>
            <xsd:element name="ReusedServiceObjectID" type="xsd:string"
           minOccurs="0"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Composite" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
```

```
                <xsd:sequence>
                   <xsd:element name="CompositeDetail">
                      <xsd:complexType>
                         <xsd:complexContent>
                            <xsd:extension base="CompositeType"/>
                         </xsd:complexContent>
                      </xsd:complexType>
                   </xsd:element>
                   <xsd:element name="CompositeXML" type="xsd:hexBinary" minOccurs="0"/>
                   <xsd:element name="CompositeRevID" type="xsd:hexBinary" minOccurs="0"/>
                   <xsd:element name="CompositeServiceElem" minOccurs="0"
                    maxOccurs="unbounded">
                      <xsd:complexType>
                         <xsd:sequence>
                            <xsd:element name="ServiceElementID" type="xsd:string"/>
                            <xsd:element name="CompositeServiceType" type="xsd:string"/>
                            <xsd:element name="CompositeServiceName" type="xsd:string"/>
                            <xsd:element name="ServiceElementXML" type="xsd:hexBinary"/>
                            <xsd:element name="Audit" type="CommonAuditType"/>
                            <xsd:element name="DeployAttributeValue"
                             type="DeployAttributeValueType" minOccurs="0"
                             maxOccurs="unbounded"/>
                         </xsd:sequence>
                      </xsd:complexType>
                   </xsd:element>
                   <xsd:element name="DeployAttributeValue" type="DeployAttributeValueType"
                    minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
             </xsd:complexType>
          </xsd:element>
          <xsd:element name="ServiceElement" minOccurs="0" maxOccurs="unbounded">
             <xsd:complexType>
                <xsd:sequence>
                   <xsd:element name="ServiceElementDetail" type="ServiceElementType"/>
                </xsd:sequence>
             </xsd:complexType>
          </xsd:element>
       </xsd:sequence>
 <xsd:attribute name="processCode"/>
 <xsd:attribute name="version"/>
 <xsd:attribute name="createDate"/>
 <xsd:attribute name="uuid"/>
</xsd:complexType>
 </xsd:element>
 <xsd:complexType name="CommonAuditType">
    <xsd:sequence>
       <xsd:element name="CreatedBy" type="xsd:string"/>
       <xsd:element name="CreationDate" type="xsd:string"/>
       <xsd:element name="LastUpdatedBy"/>
       <xsd:element name="LastUpdateDate"/>
       <xsd:element name="ObjectVersionNumber"/>
    </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="ProcessType">
    <xsd:sequence>
       <xsd:element name="ProcessName" type="xsd:string"/>
       <xsd:element name="ProcessDesc" type="xsd:hexBinary"/>
       <xsd:element name="ProcessAssumption" type="xsd:hexBinary"/>
       <xsd:element name="ProcessSource" type="xsd:string"/>
       <xsd:element name="ProcessIndustry" type="xsd:string"/>
```

```
         <xsd:element name="ProcessCode" type="xsd:string"/>
         <xsd:element name="ProcessType" type="xsd:string"/>
         <xsd:element name="Status" type="xsd:string"/>
         <xsd:element name="Version" type="xsd:string"/>
         <xsd:element name="Audit" type="CommonAuditType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="TaskType">
      <xsd:sequence>
         <xsd:element name="TaskID" type="xsd:string"/>
         <xsd:element name="TaskName" type="xsd:string"/>
         <xsd:element name="TaskDesc" type="xsd:hexBinary"/>
         <xsd:element name="TaskUuid" type="xsd:string"/>
         <xsd:element name="Audit" type="CommonAuditType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="SvccompType">
      <xsd:sequence>
         <xsd:element name="SvccompID" type="xsd:string"/>
         <xsd:element name="SvccompType" type="xsd:string"/>
         <xsd:element name="SvccompName" type="xsd:string"/>
         <xsd:element name="SvccompDesc" type="xsd:hexBinary"/>
         <xsd:element name="SvccompAssumption" type="xsd:hexBinary"/>
         <xsd:element name="ProductPillar" type="xsd:string"/>
         <xsd:element name="ProductFamily" type="xsd:string"/>
         <xsd:element name="ProductCode" type="xsd:string"/>
         <xsd:element name="SvccompGUID" type="xsd:string"/>
         <xsd:element name="Audit" type="CommonAuditType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="CompositeType">
      <xsd:sequence>
         <xsd:element name="CompositeID" type="xsd:string"/>
         <xsd:element name="CompositeName" type="xsd:string"/>
         <xsd:element name="Source" type="xsd:string"/>
         <xsd:element name="Audit" type="CommonAuditType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="ServiceElementType">
      <xsd:sequence>
         <xsd:element name="ServiceElementID" type="xsd:string"/>
         <xsd:element name="ServiceElementType" type="xsd:string"/>
         <xsd:element name="ServiceName" type="xsd:string"/>
         <xsd:element name="OperationName" type="xsd:string"/>
         <xsd:element name="Namespace" type="xsd:string"/>
         <xsd:element name="ProductPillar" type="xsd:string"/>
         <xsd:element name="ProductFamily" type="xsd:string"/>
         <xsd:element name="ProductCode" type="xsd:string"/>
         <xsd:element name="Audit" type="CommonAuditType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="DeployAttributeValueType">
      <xsd:sequence>
         <xsd:element name="GroupNumber" type="xsd:string"/>
         <xsd:element name="DeployAttributeValue" type="xsd:string"/>
         <xsd:element name="DeployAttributeCode" type="xsd:string"/>
         <xsd:element name="Audit" type="CommonAuditType"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

7.2 How to Set Up an Environment to Export or Import Seed Data for the First Time

Objective

Perform set up tasks to enable the export and import of Project Lifecycle Workbench seed data. Perform this procedure for each separate environment, whether it is being exported from or imported to. Perform this procedure only once for each environment, before the environment is accessed for export and import for the first time.

Prerequisites and Recommendations

None

Actor

- Integration developers
- System administrators

To set up a Linux environment to export or import seed data for the first time:

1. Start a server session.
2. Navigate to aiaenv.sh located in $AIA_INSTANCE/bin/aiaenv.sh.
3. At the command prompt, execute `source aiaenv.sh`.

To set up a Windows environment to export or import seed data for the first time:

1. Access a command prompt.
2. Access the drive on which AIAHome is installed.
3. Navigate to aiaenv.bat located in <AIA_HOME>\aia_instances\AB\bin.
4. At the command prompt, run `aiaenv.bat`.

7.3 How to Export Seed Data

Objective

Export Project Lifecycle Workbench seed data.

Prerequisites

Complete the steps covered in Section 7.2, "How to Set Up an Environment to Export or Import Seed Data for the First Time".

Actors

- System administrators
- Integration developers

To export Project Lifecycle Workbench seed data:

1. Access a command window on the server from which you want to export seed data.
2. Navigate to the $AIA_HOME/Infrastructure/LifeCycle/PLWImExport folder.

3. Run PLWExport to perform the seed data export. Output files are saved to the PLWImExport folder. One output XML file is generated per project.

PLWExport provides a set of export commands, as discussed in Table 7–1.

Table 7–1 PLWExport Commands

Command	Description
`-p <project code>`	Provide `-p` or `-u` in the PLWExport command.
`-v <version>`	For optional use with `-p`. If a version is not specified and there are multiple projects with the same project code, but different versions, the projects are displayed. If a version is not specified and there is only one project that matches the project code criteria, the seed data for the project is exported.
`-u <UUID value>`	Provide `-u` or `-p` in the PLWExport command. A UUID value is assigned to a Project Lifecycle Workbench project as the Project Unique ID value upon saving the project for the first time. The ID displays on the Add Project page and Update Project page and is stored in the AIA_OER_PROCESSES table. For more information about the project unique ID value as it displays in the Project Lifecycle Workbench UI, see Section 3.3, "Working with Project Lifecycle Workbench Projects."
`-s` (scope)	Use as follows: ■ `-s all` Exports BOM and functional decomposition data. ■ `-s fd` Export functional decomposition data only. If not specified, PLWExport uses `-s all`. To export BOM data, also export functional decomposition data, `-s all`. You cannot export only BOM data.
`-o <output file name>`	If not specified, PLWExport uses a file name of <project code>_<UUID>.xml.

Caution: If a project is exported with no `-o` command, and then is exported again with no `-o` command, the export file generated by the first export is overwritten by the export file generated by the subsequent export.

Following are sets of sample commands for performing exports using a variety of export criteria.

Export BOM and functional decomposition data by project code and version to a named output file:

- For Linux: `PLWExport.sh -p <Project Code> -v <version> -s all -o <output file name>`
- For Windows: `PLWExport.bat -p <Project Code> -v <version> -s all -o <output file name>`

Export functional decomposition data by project code and version to a named output file:

- For Linux: `PLWExport.sh -p <Project Code> -v <version> -s fd -o <output file name>`

  For Windows: `PLWExport.bat -p <Project Code> -v <version> -s fd -o <output file name>`

Export BOM and functional decomposition data by project code to a named output file:

- For Linux: `PLWExport.sh -p <Project Code> -s all -o <output file name>`

  For Windows: `PLWExport.bat -p <Project Code> -s all -o <output file name>`

Export functional decomposition data by project code to a named output file:

- For Linux: `PLWExport.sh -p <Project Code> -s fd -o <output file name>`

  For Windows: `PLWExport.bat -p <Project Code> -s fd -o <output file name>`

Export BOM and functional decomposition data by UUID to a named output file:

- For Linux: `PLWExport.sh -u <UUID value> -s all -o <output file name>`

  For Windows: `PLWExport.bat -u <UUID value> -s all -o <output file name>`

Export functional decomposition data by UUID to a named output file:

- For Linux: `PLWExport.sh -u <UUID value> -s fd -o <output file name>`

  For Windows: `PLWExport.bat -u <UUID value> -s fd -o <output file name>`

7.4 How to Import Seed Data

Objective

Import Project Lifecycle Workbench seed data.

Prerequisites

- Complete the steps covered in Section 7.2, "How to Set Up an Environment to Export or Import Seed Data for the First Time."
- Ensure that the seed data file to be imported exists.

  For information about how to export seed data, see Section 7.3, "How to Export Seed Data."

Actors

- Integration developers
- System administrators

To import Project Lifecycle Workbench seed data:

1. Access a command window on the server into which you want to import seed data.
2. Navigate to the PLWImExport folder: $AIA_HOME/Infrastructure/LifeCycle/PLWImExport.
3. Run PLWImport to perform the seed data import.

   PLWImport provides a set of import commands, as discussed in Table 7–2.

Table 7–2 PLWImport Commands

Command	Description
`-f <file name or file path>`	Use to import a single file. If you provide a file name, PLWImport looks for the file name in the directory from which the PLWImport was invoked and attempts to import. For example, in Linux: `./PLWImport.sh -f import1.xml` If you provide a file path, PLWImport looks for the file name in that specified directory and attempts to import. You can provide a full file path that provides a path to the file from the root of the file system. For example, in Linux: `./PLWImport.sh -f /home/user/plw/data/import1.xml` You can provide a relative file path that provides a path to the file relative to the directory from which PLWImport was invoked. For example, in Linux: `./PLWImport.sh -f data/import1.xml`
`-d <directory path>`	Use to import multiple files stored in a directory. If you provide a directory path, PLWImport looks for the specified directory and attempts to import using all files in the directory. You can provide a full directory path that provides a directory path from the root of the file system. For example, in Linux: `./PLWImport.sh -d /home/user/plw/data/batch` You can provide a relative directory path that provides a directory path relative to the directory from which PLWImport was invoked. For example, in Linux: `./PLWImport.sh -d data/batch` or `./PLWImport.sh -d batch`

   Provide `-f` or `-d` in the PLWImport command, else PLWImport displays an error message as these commands are mutually exclusive.

4. All imported seed data includes associated UUIDs.

   If a UUID associated with imported data exists in the target Project Lifecycle Workbench instance, the PLWImport updates data associated with the existing UUID.

   If the UUID is unique in the target instance, PLWImport inserts a record for the new UUID.

5. The results of the import are stored in the plwimporter.log file located in the PLWImExport folder.
6. After the PLWImport script has run successfully, verify that the imported data is accessible in Project Lifecycle Workbench.

   For more information about Project Lifecycle Workbench, see Section 3.3, "Working with Project Lifecycle Workbench Projects."

8

Generating Deployment Plans and Deploying Artifacts

This chapter discusses the approach to create Deployment Plans for SOA artifacts extended using AIA Extension mechanism. AIA Deployment Plan Generator utility helps generate Deployment Plans for extended artifacts and AIA Installation Driver (AID) helps deploy them. The chapter also outlines the approach to deploy non-native SOA artifacts. The Pre-Built Integrations or direct integrations delivered by Oracle AIA use the features outlined in this chapter to deploy both native SOA artifacts and non-native artifacts.

This chapter includes the following sections:

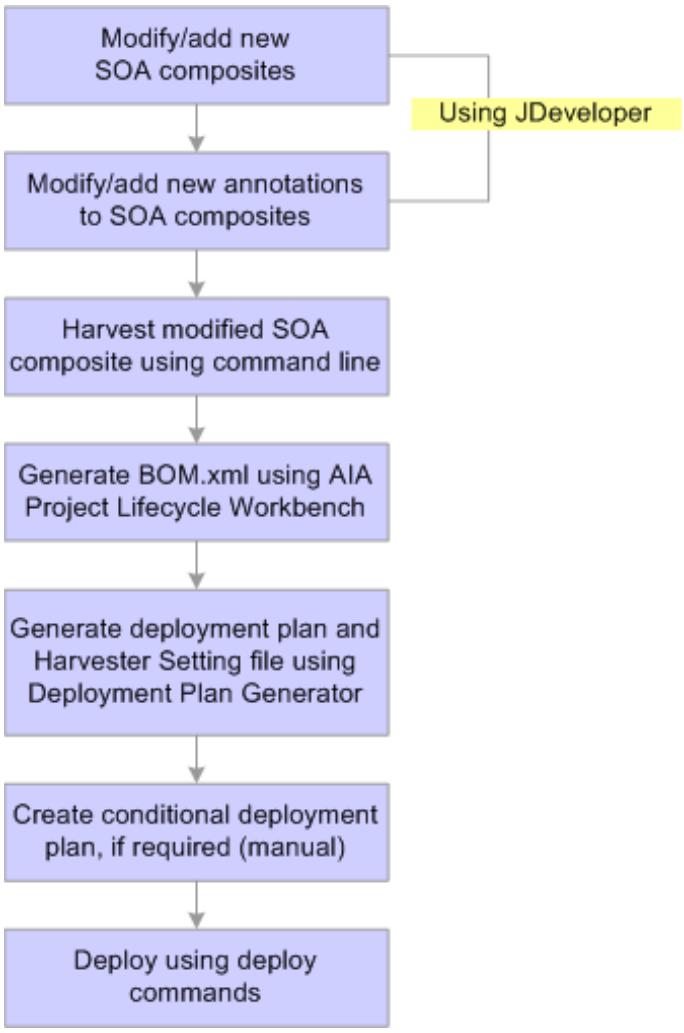
8.1 Introduction

AIA Project Lifecycle Workbench contains the integration specific details of the Pre-Built Integration or direct integrations, like the tasks involved and the service components involved in each task. Project Lifecycle Workbench supports only Oracle SOA artifacts which are created using FMW technologies such as BPEL or Mediator. These are called **native artifacts** and they are supported by AIA Foundation Pack tools such as Project Lifecycle Workbench, Harvester, Deployment Generator, and AID. When a composite is harvested, the annotation details of the composite are stored in the lifecycle database. The Bill of Material (BOM) of a Pre-Built Integration can be exported as BOM.xml using the **Generate BOM** feature in AIA Lifecycle Workbench. The existing artifacts can be modified and new natively supported artifacts can be added using AIA Lifecycle Workbench and a BOM.xml file can be generated.

The **BOM.xml** file consists of a list of artifacts that constitute a project and their ensuing annotations. Deployment Plan Generator reads the annotations and other information in BOM.xml file as input and generates deployment plan for the selected services. The deployment plan is used to perform the required configurations and deploy the services to the Fusion Middleware server using AID.

Figure 8–1 illustrates the flow for extending and deploying native artifacts.

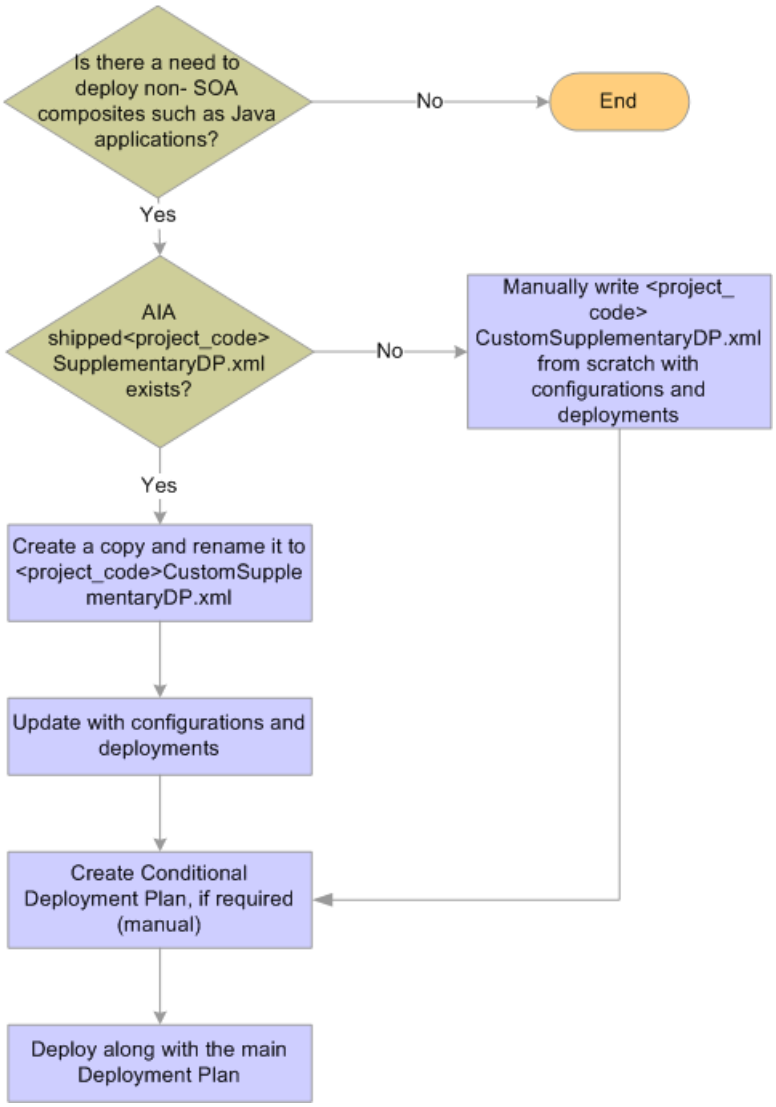
Figure 8–1 Extending and Deploying Native Artifacts



You may be required to deploy artifact types that are not supported by the Project Lifecycle Workbench and AIA Harvester. For instance, your integration may require the use of Java Web Services or interaction with third party services. In such cases, the deployment of those artifacts is supported by invoking custom scripts such as Shell scripts or ANT scripts from the Deployment plan. AID supports deployment of these supplementary artifacts. However, modify and add new non-native artifacts outside AIA Lifecycle Workbench.

Figure 8–2 illustrates the flow for extending and deploying non-native artifacts.

Figure 8–2 Extending and Deploying Non-native Artifacts



The following files are used in the deployment plan generation and deployment of AIA artifacts:

- **<PIP_Name>DP.xml:** This deployment plan file orchestrates the deployment of natively supported services and the configuration required for those services. Oracle AIA delivered Pre-Built Integrations usually have their deployment plans under *AIA_HOME/PIPS/<pip name>/DeploymentPlans*. Oracle AIA Foundation Pack customers are also advised to save their Deployment plans under the corresponding folder in the *AIA_HOME/PIPS/* folder.
- **<PIP_Name>SupplementaryDP.xml:** Supplementary Deployment Plan contains configuration required for deployment of non native services. A Pre-Built Integration installation may or may not constitute the deployment of non-native artifacts. If it does, then the corresponding Pre-Built Integration supplies a Supplementary deployment plan at *AIA_HOME/PIPS/<pip name>/DeploymentPlans*.
- **<PIP_Name>HS.xml:**HS is a Harvester Settings file that harvests the deployed services to OER for a Pre-Built Integration project from the SOA run time. When a

Pre-Built Integration is installed, the file can be located in *AIA_HOME/PIPS/<pip name>/HarvesterSettings* folder.

For more information about the Harvester settings file, see Chapter 5, "Harvesting Oracle AIA Content"

You can also generate deployment plans for ODI and include them in your deployment plans. For information on generating a deployment plan for ODI refer to Chapter 9, "Generating a Deployment Plan for ODI"

Note: You can deploy the artifacts only after installing Foundation Pack. For information about installing Foundation Pack, see *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack.*

8.2 Extending Deployment Plans

Oracle AIA Foundation Pack provides a comprehensive mechanism to extend the Oracle shipped deployment plans. Most often, integration scenarios require modifications to AIA-shipped integrations or Pre-Built Integrations before they are deployed to customer environments. The modifications include changes to Pre-Built Integration functionality, and they often manifest on the changes to Pre-Built Integration deployment configurations. This section discusses how to modify or extend both native and non-native artifacts that are shipped with Oracle AIA.

8.2.1 Extending Native Artifacts

If you must modify the native artifacts or add new natively supported artifacts, modify existing annotations or add new annotations, generate a BOM.xml using Project Lifecycle Workbench. Use the BOM.xml to run Deployment Plan Generator to generate a new deployment plan. As this is the process employed by Oracle AIA, this file will be equivalent to <PIP_Name>DP.xml. However, save it at a different file path with a different name.

Note: You can modify and add new native artifacts and generate a deployment plan to include both. However, Oracle AIA strongly recommends that you generate a new deployment plan on a different name so that these extensions do not get affected when upgrades are done or patches are applied. For example, <PIP_Name>CustomDP.xml

---

For more information about modifying AIA Artifacts or making custom extensions to AIA Artifacts and generating the BOM XML input file, seeChapter 6, "Working with Project Lifecycle Workbench Bills of Material"

8.2.2 Extending Non-Native Artifacts

To modify non-native artifacts or add new non-native artifacts, copy-paste AIA-shipped **<PIP_Name>SupplementaryDP.xml** with a different name, for example <PIP_Name>CustomSupplementaryDP.xml and modify the artifacts manually. This ensures that your customizations do not get overwritten during AIA upgrades or patch updates. If the Pre-Built Integration does not supply a SupplementaryDP.xml file, create one and add the artifacts to be modified manually.

As non-native artifacts cannot be harvested using the Project Lifecycle Workbench UI, you cannot generate a deployment plan using DPG. Deploy non-native artifacts in the <PIP_Name>CustomSupplementaryDP.xml using the deployment command and also undeploy the artifacts manually.

Note: You cannot modify and add new non-native artifacts in the same AIA-shipped <PIP_Name>SupplemenatryDP.xml.

8.3 Generating Deployment Plans

This section discusses the input required for Deployment Plan Generator, the process for generating the deployment plan, and the output generated

8.3.1 Input for Deployment Plan Generator

Deployment Plan Generator takes the following four command line inputs.

- **BOM.xml:** The BOM file is exported from the Project Lifecycle Workbench UI for the projects that are selected. BOM.xml contains the annotations to the services that are specified by users. These annotations are read by the Deployment Plan Generator to generate the deployment plan for the selected services. This deployment plan is used to configure the required configuration and deploy the services to the FMW server.
- **ODIBOM.xml:** This ODIBOM file is to be generated manually. It should contain the list of artifacts to be imported to ODI along with the list of tokens to be replaced and encrypted. These annotations, in addition to some other information, are read by Deployment Plan Generator to generate the deployment plan for ODI.
- **File path for the Deployment Plan to be generated:** The naming convention for the deployment plan is *<projectCode>DP.xml*. The Deployment Plan Generator gives a warning message if the given input argument for deployment plan does not follow the naming standards.
- **File path of the HarvesterSettings.xml to be generated:** The naming convention for the HarvesterSettings is *<projectCode>HS.xml*. The Deployment Plan Generator gives a warning message if the given input argument for Harvester Settings does not follow the naming standards. Set the path for the file to *AIA_HOME/pips/<projectCode>/HarvesterSettings* as the deployment plan refers to this file during execution.

8.3.2 Executing Deployment Plan Generator

To execute Deployment Plan Generator:

1. Set environment variables by running the commands specific to your platforms:
   - For UNIX/Linux based systems, run source <AIA_HOME>/aia_instances/<instance_name>/bin/aiaenv.sh
   - For Microsoft Windows, run <AIA_HOME>/aia_instances/<instance_name>/bin/aiaenv.bat
2. Run the following command for executing the Deployment Plan Generator. This can be executed from any location.

```
ant -f ${AIA_HOME}/Infrastructure/Install/AID/AIADeploymentPlanGenerator.xml
-Dinput=<full file path of lifecycle BOM.xml> -DDeploymentPlan=<output file
```

```
path of the generated deploymentplan. For example ${AIA_
HOME}/PIPS/<pipName>/DeploymentPlans/<pipName>DP.xml>
-DHarvesterSettings=<output file path of the generated HarvesterSettings file.
For example ${AIA_HOME}/PIPS/<pipName>/HarvesterSettings/<pipName>HS.xml>
[-DODIinput=<ODI BOM file name along with absolute path of the file>]
```

Note: AIA recommends that the output for the run-time Harvester Settings file be AIA_HOME/pips/<PIP_Name>/HarvesterSettings as deployment plan refers to this file during execution.

Provide at least BOM.xml or ODIBOM.xml as input. If you are providing both files as input, ensure that the ProjectCode is same for both the BOM files. If you are providing only ODIBOM.xml as input, then skip the HarvesterSettings.xml filepath input.

8.3.3 Output by Deployment Plan Generator

Deployment Plan Generator produces the following output:

- **Deployment Plan:** The deployment plan that is generated is very specific to the BOM.xml and ODIBOM.xml which are given as input. The services which are specified in BOM.xml can be deployed and configured by using this deployment plan. The artifacts list provided in the ODIBOM.xml is processed and imported into the ODI with the deployment plan. Deployment Plan Generator generates a combined deployment plan if both the BOM and ODIBOM are provided. This deployment plan is executed by feeding it to AID.
- **Undeployment Plan:** The undeployment plan is generated at the same location as the deployment plan. The naming convention for the undeployment plan is *<projectCode>UndeployDP.xml*. This contains undeploy tasks for all the services that are deployed and the Configurations done as part of Deployment Plan. There is no undeployment plan for ODI artifacts. The undeployment plan is also executed using the AID.
- **Harvester Settings File:** A Deployment Plan Generator utility generates a HarvesterSettings.xml file for harvesting the deployed services to OER. The list of services to be harvested are taken from the BOM.xml.

8.4 Generating Conditional Deployment Plans

You can set conditions to your deployment plans for the following scenarios:

- Choose artifacts to be deployed depending on the version of participating application.
- Define conditional logics depending on deployments and sequences for two different projects. For example, project A may lay out certain artifacts which must be conditionally kept or removed while deploying project B.

Define the conditions in a deployment policy file. The deployment policy file supports all the artifact types in configurations and deployments except the adapter types (JmsAdapter, DbAdapter, AqAdapter, OracleAppsAdapter). However, for composites define an additional condition.

The deployment policy file is passed as an additional input to AID through -DDeploymentPolicyFile variable. When AID begins execution, it checks if -DDeploymentPolicyFile variable is passed as a command line input. When it finds the variable, AID creates a copy of the original deployment plan with the name

<Original_DP_Name>-Conditional.xml. AID checks whether there are changes required to the original deployment plan based on inputs from the deployment policy file, picks up the modified deployment plan and executes them during installation.

Create the deployment policy file manually. The sample deployment plan shown in Example 8–1 helps you understand deployment policy file.

Example 8–1 Sample Deployment Plan File

```
<DeploymentPlan component="AIADemo" version="3.0">
       <PreInstallScript>
       </PreInstallScript>
       <Configurations>
              <Datasource name="FODDS" jndiLocation="jdbc/fodDS" wlserver="fp"

database="participatingapplications.AIADemo.db.aiademoparticipatingapp"
action="create"
                          xa-enabled="true"/>
              <JMSResource wlserver="fp" jmsResourceType="ConnectionFactory"
                           jmsResourceName="AIADemoCF"
                           jmsresourcejndi="jms/aia/AIADemoCF"
                           action="create" jmsModuleName="AIAJMSModule"
                           jmsSubDeploymentName="AIASubDeployment"/>
      </Configurations>
       <Deployments>
              <Application name="WebServices_WebLogicFusionOrderDemo_
CreditCardAuthorization"
                           filelocation="${AIA_
HOME}/samples/AIADemo/Services/CreditCardAuthorization/deploy/WebServices_
WebLogicFusionOrderDemo_CreditCardAuthorization.war"
                           wlserver="fp" failonerror="true" action="deploy"/>
              <Composite compositeName="AIADemoUpdateSalesOrderEBS"
                         compositedir="${AIA_
HOME}/samples/AIADemo/EBS/AIADemoUpdateSalesOrderEBS"
                         revision="1.0" wlserver="fp" action="deploy"/>
              <Composite
compositeName="AIADemoSyncCustomerPartyListCRMProvDBAdapter"
                         compositedir="${AIA_
HOME}/samples/AIADemo/Adapters/AIADemoSyncCustomerPartyListCRMProvDBAdapter"
                         revision="1.0" wlserver="fp" action="deploy"/>
       </Deployments>
</DeploymentPlan>
```

If this is your deployment plan file, create a deployment policy file as illustrated in Example 8–2.

Example 8–2 Deployment Policy File

```
<DeploymentPlan component="AIADemo" version="3.0">
<Conditions component="AIADemo" version="3.0">
   <if>
      <equals arg1="${pip.foo.boo}" arg2="11.5.7"/>
      <then>
          <UpdateDP artifacttype="Datasource" name="FODDS" action="no-action" />
          <UpdateDP artifacttype="JMSResource" name="AIADemoCF"
action="no-action" />
          <UpdateDP artifacttype="Composite" name="AIADemoUpdateSalesOrderEBS"
action="no-action" />
      </then>
   </if>
```

```
</Conditions>
</DeploymentPlan>
```

When AID executes the policy file, the contents between <Conditions> tag are written to a temporary build file and executed. Ensure that the contents inside conditions are valid ant tasks. A policy file can have only one conditions tag and all the conditions should be written inside this tag.

8.4.1 Understanding the Deployment Policy File

UpdateDP task is implemented as a taskdef in AID. The taskdef updates the action attribute of configuration and deployment tasks. In the UpdateDP task:

- **artifactype** refers to the valid task names in deployment plan.
- **name** refers to the name of the artifact created in the tag.
- **action** refers to the new action value to which the value in deployment plan should be updated.

For a Composite tag, create an additional attribute:

- dir="<new directory path>", this attribute updates the compositedir attribute in the corresponding composite tag of deployment plan. Only one of the action and dir attributes can be specified for the composite artifact type.

8.4.2 Executing the Deployment Plan

AIA executes the deployment plan for UpdateDP in the following sequence:

1. Reads the deployment plan from "DeploymentPlan" environment variable.
   a. If *artifacttype != 'Composite*', verifies whether dir attribute is passed as an input. If it is, AID throws incompatible attributes exception and shows usage.
   b. If *artifacttype = 'Composite*', verifies whether both 'dir' and 'action' attributes are passed as an input. If they are, AID throws incompatible attributes exception and shows usage.
2. Reads name and action attributes, and updates deployment plan tasks accordingly.
3. Writes the file back to disk.

For the Macrodef that you do not want deployed by AID, set the action attribute in the listed macros to "no-action" action. The AID then skips the macro execution and proceeds to the next task. This is like removing the corresponding task from the deploymentPlan. "no-action" value for the action attribute is supported by the following Macrodefs:

- **Configurations**
  - Datasource
  - JMSResource
  - JMSConnectionFactory
- For Deployments you can add the "no-action" attribute value for Composite macrodef.

8.5 Deploying Artifacts

> **Note:** To facilitate durability across upgrades and patch updates, place custom modified files in a different directory path from AIA-shipped <PIP_Name>DP.xml and <PIP_Name>SupplementaryDP.xml.

The deployment of the artifacts is done by AID. AID takes the deployment plan and AIAInstallProperties.xml file as input. Based on the tags specified in the deployment plan, AID configures and deploys the artifacts onto the server.

AID supports three types of deployment plans:

- Main Deployment Plan
- Supplementary Deployment Plan
- Custom Deployment Plan

Main Deployment Plan is auto generated by the Deployment Plan Generator whereas Supplementary Deployment Plan and Custom Deployment Plan are handcoded. Support to add custom deployment tags to the main deployment plan is available through Pre-Install and Post-Install sections in the Deployment plan. However, the problem with using these sections is that the deployment plan may not be upgrade-safe. To mitigate the issue, supplementary and custom deployment plans are introduced. The supplementary deployment plan is used mostly by the internal Pre-Built Integration development team. Use custom deployment plan to meet the requirement of non-native artifact deployment plan.

The execution sequence of deployment plans followed by AID is Main Deployment Plan -> Supplementary Deployment Plan -> Custom Deployment Plan. Here Supplementary Deployment Plan and Custom Deployment Plan are optional.

The following sections show the deployment commands for various deployment scenarios.

8.5.1 Deploying AIA Shipped Native Artifacts and Non-native Artifacts

This scenario does not involve any customizations. The following command takes the main deployment plan and the supplementary deployment plan which are shipped with the Pre-Built Integration installer as input.

```
ant -f <AIA_HOME>\Infrastructure\Install\AID\AIAInstallDriver.xml
-DPropertiesFile=<AIA_HOME>\aia_instances\<AIA_Instance_
name>\config\AIAInstallProperties.xml  -DDeploymentPlan=<AIA_HOME>\pips\<PIP_
name>\DeploymentPlans\<PIP_name>DP.xml  -DSupplementaryDeploymentPlan =<AIA_
HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_name>SupplementaryDP.xml
-DDeploymentPolicyFile=<AIA_HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_
name>ConditionalPolicy.xml
```

8.5.2 Deploying Modified AIA-shipped Artifacts

This section discusses how to deploy modified AIA-shipped native and non-native artifacts.

8.5.2.1 Deploying Modified Native Artifacts and Original Non-native Artifacts

For modified native artifacts scenario, re-harvest the modified artifacts and regenerate the deployment plan, and name it *<PIP_Name>CustomDP.xml*. This has to be passed as the main deployment plan instead of the shipped deployment plan. The AID command is:

```
ant -f <AIA_HOME>\Infrastructure\Install\AID\AIAInstallDriver.xml
-DPropertiesFile=<AIA_HOME>\aia_instances\<AIA_Instance_
name>\config\AIAInstallProperties.xml  -DDeploymentPlan=<AIA_HOME>\pips\<PIP_
name>\DeploymentPlans\<PIP_name>CustomDP.xml  -DSupplementaryDeploymentPlan
=<AIA_HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_name>SupplementaryDP.xml
-DDeploymentPolicyFile=<AIA_HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_
name>ConditionalPolicy.xml
```

8.5.2.2 Deploying Original Native Artifacts and Modified Non-native Artifacts

For the original native artifacts and modified non-native artifacts scenario, copy the contents of the shipped supplementary DP to a new file, name it *<PIP_Name>CustomSupplementaryDP.xml*, and modify the new file with the customizations. This is passed as the supplementary deployment plan instead of the shipped supplementary DP. The AID command is:

```
ant -f <AIA_HOME>\Infrastructure\Install\AID\AIAInstallDriver.xml
-DPropertiesFile=<AIA_HOME>\aia_instances\<AIA_Instance_
name>\config\AIAInstallProperties.xml  -DDeploymentPlan=<AIA_HOME>\pips\<PIP_
name>\DeploymentPlans\<PIP_name>DP.xml  -DSupplementaryDeploymentPlan =<AIA_
HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_name>CustomSupplementaryDP.xml
-DDeploymentPolicyFile=<AIA_HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_
name>ConditionalPolicy.xml
```

8.5.3 Deploying New or Custom Built Artifacts

This section discusses how to deploy newly added native and non-native artifacts.

8.5.3.1 Deploying Newly-added Native Artifacts and Original Non-native Artifacts

If you are introducing new native artifacts, harvest the new artifacts and regenerate the deployment plan for the new artifacts along with the shipped ones, and name it *<PIP_Name>CustomDP.xml*. Pass this as the main deployment plan instead of the shipped deployment plan. The AID command is:

```
ant -f <AIA_HOME>\Infrastructure\Install\AID\AIAInstallDriver.xml
-DPropertiesFile=<AIA_HOME>\aia_instances\<AIA_Instance_
name>\config\AIAInstallProperties.xml  -DDeploymentPlan=<AIA_HOME>\pips\<PIP_
name>\DeploymentPlans\<PIP_name>CustomDP.xml  -DSupplementaryDeploymentPlan
=<AIA_HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_name>SupplementaryDP.xml
-DDeploymentPolicyFile=<AIA_HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_
name>ConditionalPolicy.xml -l <AIA_HOME>\pips\<PIP_
name>\DeploymentPlans\<PIPDeploymentPlanName>.log
```

8.5.3.2 Deploying Newly Added Non-native Artifacts

For new non-native artifacts scenario, add customizations to <PIP_Name>CustomDP.xml which is an empty DP shipped with the Pre-Built Integration. This Custom DP is in the same location as the main DP. Pass this as Custom Deployment Plan to AID. The AID command is:

```
ant -f <AIA_HOME>\Infrastructure\Install\AID\AIAInstallDriver.xml
-DPropertiesFile=<AIA_HOME>\aia_instances\<AIA_Instance_
name>\config\AIAInstallProperties.xml  -DDeploymentPlan=<AIA_HOME>\pips\<PIP_
name>\DeploymentPlans\<PIP_name>DP.xml  -DSupplementaryDeploymentPlan =<AIA_
HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_name>SupplementaryDP.xml
-DCustomDeploymentPlan=<AIA_HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_
name>CustomDP.xml> -DDeploymentPolicyFile=<AIA_HOME>\pips\<PIP_
name>\DeploymentPlans\<PIP_name>ConditionalPolicy.xml
```

The PropertiesFile, AIAInstallProperties.xml, contains the details of the AIA environment and is located here:
<AIA_HOME>/aia_instances/<instance_name>/config.

8.6 Undeploying Services

The undeployment plan is generated at the same location as the deployment plan with the name *<PIP_Name>UndeployDP.xml*. The undeployment plan is generated only for native artifacts modified through the Project Lifecycle Workbench. This contains undeploy tasks for all the services deployed and the configurations done as part of Deployment Plan. The undeployment plan is executed using the AID.

The undeployment command is similar to the deployment plan command except for the Deployment Plan input argument and an additional argument "Uninstall".

For example, if you have used the following command to deploy modified native artifacts:

```
ant -f <AIA_HOME>\Infrastructure\Install\AID\AIAInstallDriver.xml
-DPropertiesFile=<AIA_HOME>\aia_instances\<AIA_Instance_
name>\config\AIAInstallProperties.xml  -DDeploymentPlan=<AIA_HOME>\pips\<PIP_
name>\DeploymentPlans\<PIP_name>DP.xml  -DSupplementaryDeploymentPlan =<AIA_
HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_name>SupplementaryDP.xml
-DDeploymentPolicyFile=<AIA_HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_
name>ConditionalPolicy.xml
```

then the undeployment command will be:

```
ant -f <AIA_HOME>\Infrastructure\Install\AID\AIAInstallDriver.xml
-DPropertiesFile=<AIA_HOME>\aia_instances\<AIA_Instance_
name>\config\AIAInstallProperties.xml Uninstall -DDeploymentPlan=<AIA_
HOME>\pips\<PIP_name>\DeploymentPlans\<PIP_name>UndeployDP.xml
```

However, for non-native artifacts, generate the undeployment plan manually. Take a copy of the supplementary deployment plan and name it *<PIP_Name>UndeploySupplementaryDP.xml* or *<PIP_Name>UndeployCustomSupplementaryDP.xml* depending on the supplementary deployment plan name. In the new deployment plan change the action attributes of all the tasks from "`deploy`" to "`undeploy`" or from "`create`" to "`delete`".

9

Generating a Deployment Plan for ODI

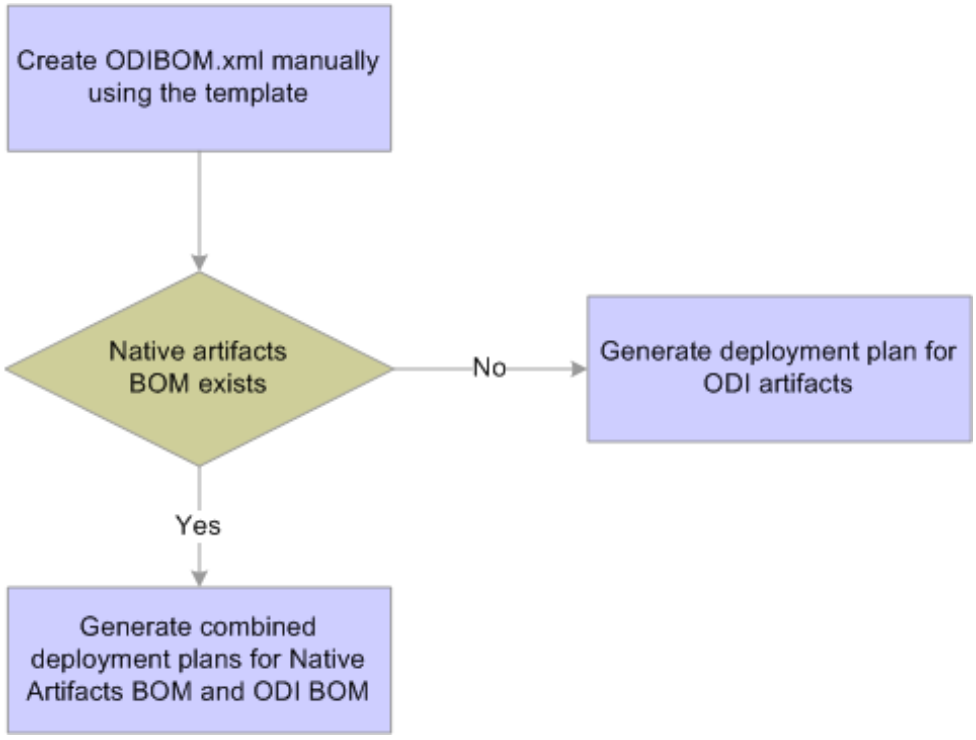
This chapter discusses the approach to create Deployment Plans for Oracle Data Integrator (ODI) artifacts extended using AIA Extension mechanism. AIA Deployment Plan Generator utility helps generate Deployment Plans for extended artifacts and AIA Installation Driver (AID) helps deploy them.

This chapter includes the following sections:

9.1 Introduction to Generating a Deployment Plan for ODI

The deployment plan is used to perform the required configurations and deploy the services to the Fusion Middleware server using the AIA Installation Driver. Figure 9–1 illustrates the flow for generating deployment plans for ODI.

Figure 9–1 Generate Deployment Plans for ODI



For more information about deployment plans, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

9.2 Generating the BOM for ODI

ODI BOM is a hand-coded xml file with the list of artifacts to be imported to ODI along with the list of tokens to be replaced and encrypted. This ODI BOM is provided as input to the Deployment Plan Generator to generate the ODI deployment plan.

9.2.1 Understanding the ODIBOM.xml File

The ODIBOM.xml file consists of the following information:

1. Passwords to be encrypted in the ODI artifacts.
2. Tokens to be replaced during deployment time in the ODI artifacts.
3. The path elements of the master and work repository files to be imported into ODI.

Example 9–1 shows the structure of a sample ODI BOM.

Example 9–1 Sample ODI BOM

```
<?xml version="1.0" encoding="UTF-8"?>

<PipOdi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
projectCode="ODISample">
<!-- WORK & MASTER REPOSITORIES  -->
 <Repositories>
  <MasterRepositoryLocation>${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master</MasterRepositoryLocation>
```

```
  <WorkRepositoryLocation>${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/work</WorkRepositoryLocation>
</Repositories>

<CopyDvmstoODIPath>
    <fileset dir="${AIA_
HOME}/PIPS/Industry/Communications/DIS/Collections/setupxmls/">
        <include name="COLLECTION_ACTIONNAME.xml"/>
    </fileset>
</CopyDvmstoODIPath>
<ODIReplaceTokens>
    <property>siebel.db.username</property>
    <property>brm.username</property>
    <property>ebiz.db.username</property>
</ODIReplaceTokens>
<ODIEncryptPasswords>
    <property>siebel.db.password</property>
    <property>brm.password</property>
    <property>ebiz.db.pwd</property>
</ODIEncryptPasswords>
<!-- Exported ODI files -->
 <!--  Master Repository objects -->
 <MSTREP_Grp>
<path id="masterFiles" >
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONT_Global.xml"/>
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONT_RETL_TO_PSFT.xml"/>
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONT_RETL_TO_PSFT_DB2.xml"/>
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONN_PEOPLESOFTDS.xml"/>
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONN_PSFT_DB2_DS.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/master
/LSC_Peoplesoft.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/master
/LSC_PSFT_DB2_LogicalSchema.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/master
/LSC_Retail.xml"/>
</path>
 </MSTREP_Grp>

 <!-- Work Repository files-->
 <WRKREP_Grp>
<path id="workFiles" >
   <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work/MFOL_
OracleRetailToPeopleSoft.xml"/>
   <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work/MFOL_
ReIMToPeopleSoftAccountingEntry.xml"/>
   <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work/MFOL_
ReIMToPSFTInvoice.xml"/>
   <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work/MFOL_
RMSToPeopleSoftAccountingEntry.xml"/>
   <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work/CONN_
PSFT_DB2_DS.xml"/>
   <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work/PROJ_
ReIMToPeopleSoftInvoiceProject.xml"/>
   <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work/PROJ_
ReIMToPSFTAcctEntryProject.xml"/>
```

```
   <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work/PROJ_
RMSToPSFTAcctEntryProject.xml"/>
</path>
 </WRKREP_Grp>
</PipOdi>
```

9.2.2 Understanding the Sections in the BOM.xml

This section discusses how to use the sections in the BOM.xml.

9.2.2.1 ODIReplaceTokens

The ODIReplaceTokens section lists the tokens that are to be replaced in ODI artifacts. In AIA 2.x versions, all tokens for artifacts are available as $<token name>$. Change this to <token name>. Token name should be same as the xpath of the property in AIAInstallProperties.xml. Example 9–2 shows an example for a property.

Example 9–2 Structure of a Property in AIAInstallProperties.xml

```
<pips>
 <siebel>
  <db>
   <jdbc-url>xyz</jdbc-url>
  </db>
  </siebel>
</pips>
```

The above property value in the ODI artifacts should be tokenized with the xpath of the property in AIAInstallProperties.xml, which is, `pips.siebel.db.jdbc-url`.

9.2.2.2 ODIEncryptPasswords

The ODIEncryptPasswords section lists the passwords that are to be encrypted in ODI artifacts. The password property should be same as the xpath of the corresponding property in AIAInstallProperties.xml. Example 9–3 shows an example for the structure of a password field.

Example 9–3 Structure of a Password Field in AIAInstallProperties.xml

```
<pips>
 <siebel>
  <db>
   <password>xyz</password>
  </db>
  </siebel>
</pips>
```

9.2.2.3 CopyDvmstoODIPath

The CopyDvmstoODIPath section lists the DVMs that are to be copied to the odi.dvm.path location. List all of the files using the filename tag within the fileset tag. You can have multiple fileset tags within the CopyDvmstoODIPath element.

9.2.2.4 MSTREP_Grp

The MSTREP_Grp section lists the files that are to be imported to the master repository. List all the files using the pathelements inside path tag. They should be listed in the same order in which they must be imported to ODI.

9.2.2.5 WRKREP_Grp

The WRKREP_Grp section lists the files that are to be imported to the work repository. List all the files using the pathelements inside path tag. They should be listed in the same order in which they must be imported to ODI.

9.3 Generating a Deployment Plan for ODI

The command to generate a deployment plan for ODI is similar to the command to generate a deployment plan for the Project Lifecycle Workbench BOM. You must add -DODIinput variable.

To generate a deployment plan for ODI, execute the following steps:

1. Set environment variables running the command specific to your platform.
   - *source <AIA_HOME>/aia_instances/<instance_name>/bin/aiaenv.sh* for Linux or UNIX based systems.
   - *<AIA_HOME>/aia_instances/<instance_name>/bin/aiaenv.bat* for Microsoft Windows.
2. Run the following command for executing Deployment Plan Generator. This can be executed from any location:

```
ant -f
<AIA_HOME>/Infrastructure/Install/AID/AIADeploymentPlanGenerator
.xml -DODIinput=<ODI BOM file name along with absolute path of
the file> -DDeploymentPlan=<Destination path of the generated
DeploymentPlan>
```

To generate a combined Deployment Plan for Project Lifecycle Workbench BOM and ODI BOM, execute the following command:

```
ant -f
<AIA_HOME>/Infrastructure/Install/AID/AIADeploymentPlanGenerator
.xml -Dinput=<BOM file name along with absolute path of the
file> -DODIinput=<ODI BOM file name along with absolute path of
the file> -DDeploymentPlan=<output path for the Deployment plan
along with file name> -DHarvesterSettings=<output path for the
runtime Harvester setting file along with file name>
```

> **Note:** While generating a combined Deployment Plan, ensure that the `<PIP_Name>` attribute in the root tags of both BOMs is the same. The Deployment Plan Generator generates a single deployment plan for both the Project Lifecycle Workbench BOM and ODI BOM.

9.3.1 Understanding the ODI Deployment Plan

When you generate a deployment plan for the sample ODIBOM.xml file discussed in Section 9.2, "Generating the BOM for ODI" the deployment plan shown in Example 9–4 is generated.

Example 9–4 Sample Deployment Plan

```
<DeploymentPlan component="ODISample" version="3.0">
<Configurations>
   <replace token="siebel.db.username"   value="${siebel.db.username}"
dir="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/master" />
```

```
   <replace token="siebel.db.username"   value="${siebel.db.username}"
dir="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work" />
   <replace token="brm.username"   value="${brm.username}"   dir="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master" />
   <replace token="brm.username"   value="${brm.username}"   dir="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/work" />
   <replace token="ebiz.db.username"   value="${ebiz.db.username}"   dir"${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master" />
   <replace token="ebiz.db.username"   value="${ebiz.db.username}"   dir"${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/work" />
<UpdateOdiParams projectCode="ODISample" />
   <OdiEncrypt property="siebel.db.password"
masterRepositoryLoc="${AIAHome}/PIPS/Core/DIS/RetailToPSFTFin/master"
workRepositoryLoc="${AIAHome}/PIPS/Core/DIS/RetailToPSFTFin/work" />
   <OdiEncrypt property="ebiz.db.password"
masterRepositoryLoc="${AIAHome}/PIPS/Core/DIS/RetailToPSFTFin/master"
workRepositoryLoc="${AIAHome}/PIPS/Core/DIS/RetailToPSFTFin/work" />
   <OdiEncrypt property="brm.password"
masterRepositoryLoc="${AIAHome}/PIPS/Core/DIS/RetailToPSFTFin/master"
workRepositoryLoc="${AIAHome}/PIPS/Core/DIS/RetailToPSFTFin/work" />
<copy todir="${odi.dvm.path}">
   <fileset dir="${AIA_
HOME}/PIPS/Industry/Communications/DIS/Collections/setupxmls/">
        <include name="COLLECTION_ACTIONNAME.xml"/>
    </fileset>
</copy>

</Configurations>
<Deployments>
<OdiImportObject import_mode="SYNONYM_INSERT_UPDATE" />
   <path id="masterFiles" >
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONT_Global.xml"/>
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONT_RETL_TO_PSFT.xml"/>
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONT_RETL_TO_PSFT_DB2.xml"/>
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONN_PEOPLESOFTDS.xml"/>
     <pathelement location="${AIA_
HOME}/PIPS/Core/DIS/RetailToPSFTFin/master/CONN_PSFT_DB2_DS.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/master
/LSC_Peoplesoft.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/master
/LSC_PSFT_DB2_LogicalSchema.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/master
/LSC_Retail.xml"/>
   </path>
</OdiImportObject>
<OdiImportObject workrepname="${pips.ODISample.odi.workrep.name}"  import_
mode="SYNONYM_INSERT_UPDATE" />
   <path id="workFiles" >
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work
/MFOL_OracleRetailToPeopleSoft.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work
/MFOL_ReIMToPeopleSoftAccountingEntry.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work
/MFOL_ReIMToPSFTInvoice.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work
/MFOL_RMSToPeopleSoftAccountingEntry.xml"/>
```

```
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work
/CONN_PSFT_DB2_DS.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work
/PROJ_ReIMToPeopleSoftInvoiceProject.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work
/PROJ_ReIMToPSFTAcctEntryProject.xml"/>
     <pathelement location="${AIA_HOME}/PIPS/Core/DIS/RetailToPSFTFin/work
/PROJ_RMSToPSFTAcctEntryProject.xml"/>
   </path>
</OdiImportObject>
</Deployments>
```

The macros in the generated deployment plan handle the following tasks.

9.3.1.1 OdiImportObject

The OdiImportObject macrodef shown in Example 9–5 imports the contents of an export file into a repository. This macrodef internally invokes the OdiImportObject utility provided as a part of Oracle Data Integrator tools.

Example 9–5 Sample Syntax of OdiImportObject

```
<OdiImportObject  workrepname="<work_rep_name>"  import_mode="SYNONYM_INSERT_
UPDATE" >
  <path id="workFiles" >
   <pathelement location="${AIAHome]/PIPS/Core/DIS/RetailToPSFTFin/src/work/MFOL_
OracleRetailToPeopleSoft.xml"/>
   <pathelement location="${AIAHome]/PIPS/Core/DIS/RetailToPSFTFin/src/work/MFOL_
ReIMToPeopleSoftAccountingEntry.xml"/>
  </path>
</OdiImportObject>
```

import_mode is optional. If you do not include import_mode, the value is set to `SYNONYM_INSERT_UPDATE`. Include all the names to be listed using the pathelement tags. For the master repository files, `workrepname` attribute should not be present.

9.3.1.2 OdiEncrypt

The OdiEncrypt macrodef encrypts the passwords and replaces the password tokens in the ODI artifacts with the encrypted values. When AIA Install Driver is launched it decrypts all of the values in AIAInstallProperties.xml and loads them into memory. Using OdiEncrypt macrodef, the values in memory are encrypted and the files in master and work repository directories are updated with the new values.

Syntax of OdiEncrypt:

```
<OdiEncrypt property="<property name in
AIAInstallProperties.xml>" masterRepositoryLoc=""
workRepositoryLoc="" />
```

9.3.1.3 UpdateOdiParams

UpdateOdiParams macrodef updates the odiparams.sh(bat) script under $ODI_HOME/agent/bin with the current ODI server details. It picks up the required properties from AIAInstallProperties.xml. The password fields are encrypted using the $ODI_HOME/agent/bin/encode.sh(bat) script. The syntax of UpdateOdiParams will be as below:

```
<UpdateOdiParams projectCode="ODISample"/>
```

The replace tokens, UpdateOdiParams, ODIEncrypt and Dvm Copy tags are generated in the Configurations section. The ODIImportObject tags that are used for ODI artifacts import are generated in the Deployments section.

After the deployment plan is generated, deploy the services using commands specified in Section 8.5, "Deploying Artifacts"

10

Developing and Deploying Custom XPath Functions

This chapter discusses how to implement a function in Java as a Java XPath Class, deploy an XPath/XSL function in JDeveloper and deploy an XPath/XSL function in application servers.

This chapter includes the following sections:

10.1 Implementing a Function in Java as a Java XPath Class

Since the custom function can be invoked either as an XPath or XSL function, the implementation class must take care of both. For this reason, the hosting class must have at least:

- A public static method with the same name as the function name.

  This method is used by the XSL function invocation. Currently, optional parameters on custom functions are not supported when registering with JDeveloper. For this reason, it is recommended to avoid functions with optional parameters.

- A public static inner class which implements:

  oracle.fabric.common.xml.xpath.IXPathFunction

  This interface has the method:

  Object call (IXPathContext context, List args) throws XPathFunctionException that gets called by service engines when the XPath function is invoked. When the function is called, the runtime engine passes the XPathContext and list of arguments passed to the function. IXPathContext enables you to get access to the BPEL variable values within the execution context. The implementation of this method should naturally delegate to the public static method representing the function.

  Here is a custom function implementation to get the current date. The implementation takes care of supporting both XSL and XPath. The function also has an optional parameter to show how overloaded methods are implemented:

Example 10–1 Custom Function Implementation to Get the Current Date

```
package oracle.apps.aia.core;•
import java.text.SimpleDateFormat;
import java.util.Date; import
java.util.List;
import oracle.fabric.common.xml.xpath.IXPathContext;
import oracle.fabric.common.xml.xpath.IXPathFunction;
import oracle.fabric.common.xml.xpath.XPathFunctionException;
public class DateUtils
{
public static class GetCurrentDateFunction implements IXPathFunction
{
public Object call(IXPathContext context, List args)
throws XPathFunctionException
{
if (args.size() == 1)
{
String mask = (String) args.get(0);
if(mask==null || mask.equals(""))
mask = "yyyy-MM-dd";
return getCurrentDate(mask);
}
throw new XPathFunctionException("must pass one argument.");
}
}
public static String getCurrentDate(String formatMask)
{
SimpleDateFormat df = new SimpleDateFormat(formatMask)
String s = df. format (new Date());
return s;
}
. . .
}
```

10.1.1 Naming Standards for Custom XPath Functions

These naming standards are enforced:

- Function name
- Function implementation class
- Function implementation method
- Function inner class
- Function namespace
- Function namespace prefix

10.1.1.1 Function Name

The function name must follow the standard Java method naming standards where the name should be Lower-Camel-Case.

Example: `getCurrentDate`, `getEBMHeaderValue`, ...

10.1.1.2 Function Implementation Class

You can have one or more function implemented in the same class. The class name must follow the Java class naming standard where it should be Upper-Camel-Case and convey the functionality of the functions it implements.

Example: `EBMHeaderUtils`, `RoutingUtils`, ...

10.1.1.3 Function Implementation Method

The implementation class will have a method for each XPath function. The method should be named the same as the function name. The parameter's data types should match the function parameter data types. If there are optional parameters in the function, you should have a different overloaded method to handle the extra optional parameters.

Example: `getCurrentDate`, `getEBMHeaderValue`, ...

10.1.1.4 Function Inner Class

There should be an inner-class for each XPath function named exactly as the Upper-Camel-Case of the function with a 'Function' suffix. The inner class is only needed to access the XPath function from BPEL. The inner class will have to implement the com.oracle.bpel.xml.XPath.IXPathFunction interface.

Example: `GetESBHeaderValueFunction`, ...

10.1.1.5 Function Namespace

For the function to appear in both the BPEL expression builder wizard and the XSL Mapper, the namespace must start with:
http://www.oracle.com/XSL/Transform/java/ then followed by the fully qualified name of the Java class, which implements the function.

Example:

http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.ebmheader.EBMHeaderUtils

http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.order.route.RoutingUtils

10.1.1.6 Function Namespace Prefix

The namespace prefix must be a readable abbreviation based on functionality.

Example: ebh for `GetEBMHeaderValueFunction`.

10.1.2 Supported Data Types

The XPath 1.0 and XSL 1.0 data types are supported as return or input parameters to the function:

Table 10–1 Supported Data Types for XPath Functions

XPath 1.0/XSL 1.0	Java
Node-set	XMLNodeList
Boolean	boolean
String	String
Number	Int, float, double,...
Tree	XMLDocumentFragment

10.2 Deploying the XPath/XSL Function in JDeveloper

Custom functions should be registered in JDeveloper to be able to show them with BPEL expression builder and in the XSL Mapper. To do that, provide a User Defined Extension Functions config file [ext-soa-xpath-functions-config.xml] and register it with JDeveloper through FilePreferencesXSL Map.

This config file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<soa-xpath-functions xmlns=http://xmlns.oracle.com/soa/config/xpath
xmlns:utl="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.DateUtils
"
<function name="utl:getCurrentDate">
 <className>oracle.xpath.CustomFunction</className>
 <return type="string"/>
 <params>
  <param name="formatMask" type="string"/>
 <params>
</function>
<function ...>
 . . .
</function>
</functions>
```

The implementation classes must be zipped/jared and dropped at:. $SOA_HOME/soa/modules/oracle.soa.ext_11.x.x/classes folder. This way, these classes will be placed in the classpath of WebLogic server.

10.3 Deploying the XPath/XSL Function in the Application Server

The Java XPath function should be registered with the application server. This is done by adding an entry in the ext-soa-xpath-functions-config.xml file, which is found at: $SOA_HOME/soa/modules/oracle.soa.ext_11.x.x/classes\META-INF. This file contains the list and descriptions of all XPath functions defined in the system.

Each function entry in this file has this information:

- <function>: Has the name attribute that defines XPath function name.
- <classname>: Defines the XPath implementation class name. In this case, it is the inner class name.
- <return>: Used to define return type.
- <params>:
  - All the parameters required by the function.
  - <param name="formatMask" type="string"/>
  - For each of the parameter passed specify the name and the data type of the input.

```
<function name="utl:getCurrentDate">
 <className>oracle.xpath.CustomFunction</className>
 <return type="string"/>
 <params>
  <param name="formatMask" type="string"/>
 <params>
</function>
```

The implementation classes must be dropped at: $SOA_HOME/soa/modules/oracle.soa.ext_11.x.x/classes folder. This way, these classes will be placed in the classpath of the SOA container and available to the classloader.

11

Configuring and Using Oracle Enterprise Repository as the Oracle AIA SOA Repository

This chapter discusses how to use Oracle Enterprise Repository (OER) as Oracle AIA SOA Repository, provide EBO and EBM documentation links in OER and access Oracle AIA content in OER.

This chapter includes the following sections:

11.1 Introduction to Using Oracle Enterprise Repository as the Oracle AIA SOA Repository

Oracle Application Integration Architecture (AIA) leverages Oracle Enterprise Repository as its SOA repository solution, providing a centrally managed user interface for discovering and learning about the SOA assets in your Oracle AIA ecosystem.

Specifically, all prebuilt AIA design-time interfaces, including Enterprise Business Service (EBS) WSDL files, Application Business Connector Service (ABCS) WSDL files, Enterprise Business Object (EBO) XSD files, Enterprise Business Message (EBM) XSD files, and their underlying artifacts, relationships, and metadata are delivered through Oracle Enterprise Repository.

Beyond this, you also can publish run-time, deployed composites into Oracle Enterprise Repository. As such, Oracle Enterprise Repository can provide visibility and coverage across the span of the SOA design-time and run-time lifecycles.

Solution Packs

Solution packs are an Oracle Enterprise Repository mechanism used to deliver content in bulk. The prebuilt AIA SOA portfolio is shipped as a solution pack. You can import the solution pack into your on-premise Oracle Enterprise Repository instances to view, access, and evaluate the AIA SOA portfolio.

The importing of an AIA solution pack to Oracle Enterprise Repository is a task that is independent of the installation of AIA. If you intend to use Oracle Enterprise

Repository in your enterprise, you can import an AIA solution pack before or after your AIA installation. In fact, you can even import an AIA solution pack without purchasing or installing AIA.

The importing of an AIA solution pack does require that you perform the prerequisite steps documented in Section 5.1, "How to Set Up Oracle AIA Content Harvesting."

For more information about performing these imports, see "Importing Items into Oracle Enterprise Repository" in *Oracle Fusion Middleware User Guide for Oracle Enterprise Repository.*

Note: While the AIA solution pack import is in progress, ensure that no other activities occur on the target Oracle Enterprise Repository instance.

After AIA design-time artifacts and deployed composites have been published into Oracle Enterprise Repository, your enterprise can search for, browse, and view them in Oracle Enterprise Repository. By keeping Oracle Enterprise Repository synchronized with your AIA ecosystem, you ensure that it can serve as the system of record for your business services and their topologies.

Potential users of Oracle Enterprise Repository as a SOA repository are active across the span of the SOA development lifecycle and include functional and business analysts, architects, developers, system integrators, and system administrators.

For example, a business analyst working on requirements for building a particular business process can use Oracle Enterprise Repository to determine which business capabilities are available in a particular integration area and then determine which additional capabilities must be built. The capabilities are delivered in the form of application-independent services and objects.

Additionally, solution architects can use Oracle Enterprise Repository during their functional analysis (using Project Lifecycle Workbench) to evaluate the potential for service reuse.

For more information about viewing Oracle AIA artifacts and composites in Oracle Enterprise Repository, see Section 11.3, "How to Access Oracle AIA Content in Oracle Enterprise Repository."

For more information about Project Lifecycle Workbench, see Chapter 3, "Working with Project Lifecycle Workbench."

11.2 How to Provide EBO and EBM Documentation Links in Oracle Enterprise Repository

Objective

AIA delivers HTML documentation for each EBO and EBM, which you can link to from respective EBO and EBM entry detail pages in Oracle Enterprise Repository.

To do this, set up an Oracle Enterprise Repository artifact store to enable Oracle Enterprise Repository to provide links to AIA EBO and EBM HTML documentation provided on an AIA web server.

If you do not perform this setup, the AIA Reference Doc link appears on EBO and EBM detail pages in the Oracle Enterprise Repository, but does not lead anywhere.

Actor

System administrator

How to provide EBO and EBM HTML documentation links in Oracle Enterprise Repository

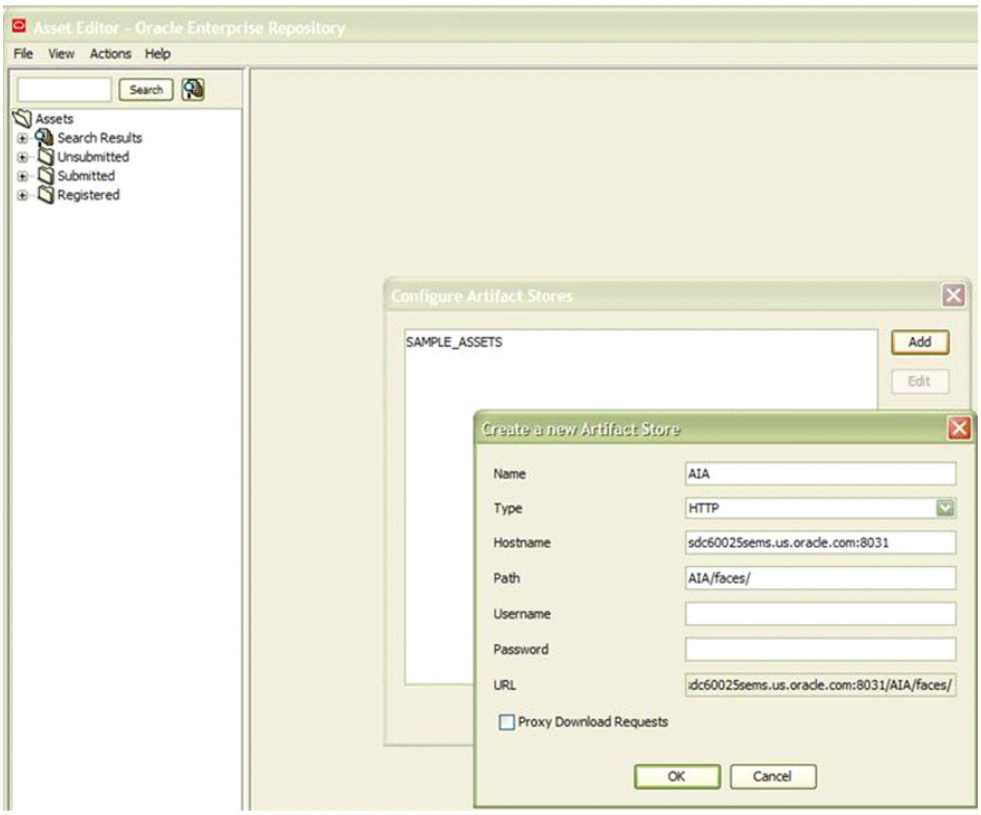
1. Access the Oracle Enterprise Repository user interface (UI): http://<host>:<port>/oer. Click the **Edit/Manage Assets** link in the Assets menu to launch the Asset Editor.
2. In the Asset Editor, navigate to **Actions, Configure Artifact Stores**.

   Click **Edit** to add a new AIA artifact store. The artifact store must be named **AIA.**

   You must manually add this artifact store when setting up the Oracle Enterprise Repository for the first time.
3. In the Create a new Artifact Store dialog box, as shown in Figure 11–1, define the **Hostname** value for the AIA artifact store, which is the <host>:<port> value for the AIA web server installation. Define the **Path** value as */AIA/faces*.

   The Hostname and Path values express the AIA location at which EBO and EBM HTML documentation can be accessed through the HTTP protocol.

Figure 11–1 Edit Artifact Store Dialog Box



4. Click **OK**.

11.3 How to Access Oracle AIA Content in Oracle Enterprise Repository

Objective

Access AIA content in the Oracle Enterprise Repository.

Prerequisites and Recommendations

- Read relevant Oracle Enterprise Repository documentation and understand Oracle Enterprise Repository asset models and graphs.

  For more information, see *Oracle Fusion Middleware User Guide for Oracle Enterprise Repository.*

- Complete the steps covered in Section 5.1, "How to Set Up Oracle AIA Content Harvesting."
- For prebuilt design-time interfaces delivered by AIA, import the solution pack into the Oracle Enterprise Repository.

  For more information, see Section 11.1, "Introduction to Using Oracle Enterprise Repository as the Oracle AIA SOA Repository."

- For custom-built individual composites that have not been deployed, run the AIA Harvester to publish them into Oracle Enterprise Repository.

  For more information, see Section 5.2, "Harvesting Design-Time Composites into Project Lifecycle Workbench and Oracle Enterprise Repository."

- For custom-built interfaces, run the AIA Harvester to publish them into Oracle Enterprise Repository.

  For more information, see Section 5.3, "Harvesting Interfaces to Oracle Enterprise Repository in Bulk."

- For deployed composites (part of deployed Process Integration Packs), run the AIA post installation script to publish the run-time composite into Oracle Enterprise Repository.

  For more information, see Section 5.4, "Harvesting Deployed Composites into Oracle Enterprise Repository."

- To provide links to EBO and EBM HTML documentation from EBO and EBM detail pages, complete the steps in Section 11.2, "How to Provide EBO and EBM Documentation Links in Oracle Enterprise Repository."

Actors

- Business analysts
- Solution architects
- Developers
- Functional analysts
- Integration architects
- Release engineers
- System administrators
- System integrators

To access Oracle AIA design-time artifacts and deployed composites in Oracle Enterprise Repository:

1. Access the Oracle Enterprise Repository to which your AIA design-time artifacts and deployed composites have been published.

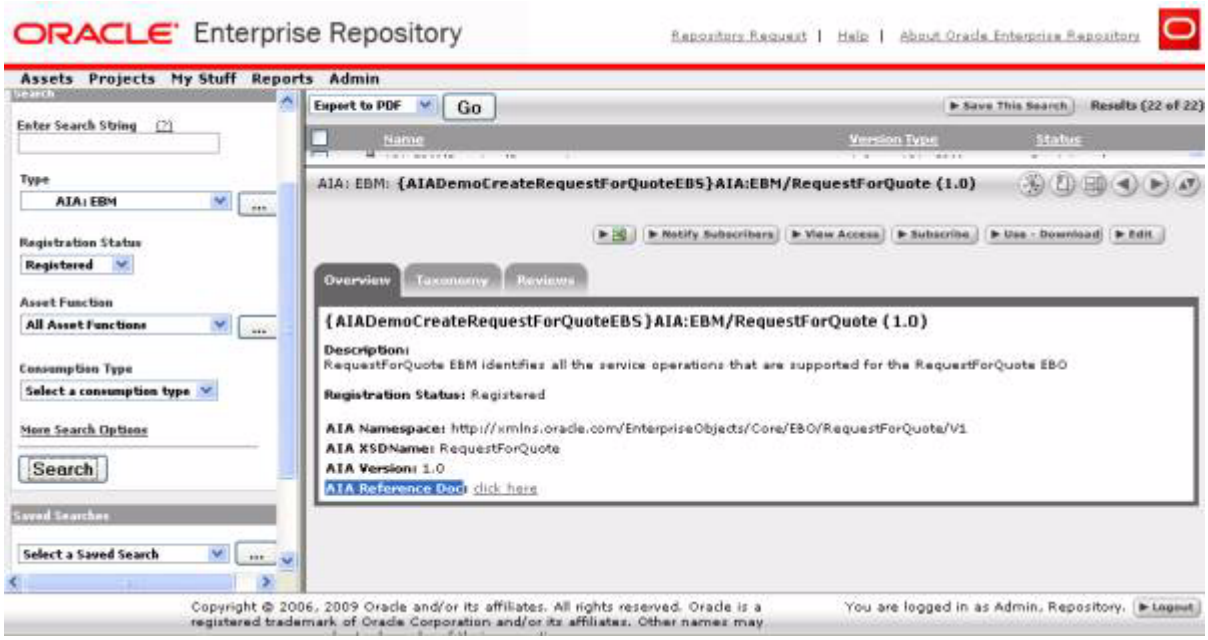
   Use Oracle Enterprise Repository search and browse functionality to locate AIA content.

   For more information about using Oracle Enterprise Repository, see *Oracle Fusion Middleware User Guide for Oracle Enterprise Repository.*

2. To narrow your search to AIA asset types, select an **AIA: <XYZ>** value in the **Type** drop-down list box in the Search menu. For example, by selecting **AIA: EBO**, you can narrow your search to AIA EBOs only.

   When viewing EBOs and EBMs in Oracle Enterprise Repository, you can click the **AIA Reference Doc** link as shown in Figure 11–2 to access HTML documentation about the content.

Figure 11–2 AIA Reference Doc Link



For more information about providing links to EBO and EBM HTML documentation from EBO and EBM detail pages in Oracle Enterprise Repository, see Section 11.2, "How to Provide EBO and EBM Documentation Links in Oracle Enterprise Repository."

12

Annotating Composites

This chapter describes annotations and discusses how to annotate Service Element and Reference Element in Requester ABCS Composite, Provider ABCS Composite, Transport Adapter Composite, Enterprise Business Flow Composite, and Composite Business Process Composite. It also discusses valid values for annotation elements.

This chapter includes the following sections:

12.1 Why Annotate a SOA Composite?

AIA recommends annotations in the composite XML file to provide detailed information about:

- AIA artifacts and their relationships to other AIA artifacts.
- Composite-level descriptor properties that are used to configure the component at deployment and run time.

AIA architecture categorizes SOA composites further as adapter services, requester services, provider services, and so on based on their usage. The meta information of these AIA services is used in maintaining Oracle Enterprise Repository (OER) assets of

AIA asset types and linking them to OER assets with native asset types; this is accomplished with the help of the AIA Harvester, which harvests the SOA composites.

12.1.1 What Elements of a Composite Must be Annotated?

You must provide annotations in the composites for the exposed services and for the referenced services, according to AIA guidelines, and you must insert these comments at development time.

In line with SOA modeling and development practices, these composites are expected to be harvested multiple times during the development cycle, from conception till deployment to production environment.

Embed annotations in the <*svcdoc:AIA*> element and place the <svcdoc:AIA> element itself inside the xml comments tags <!-- and -- > as shown below.

The first annotation element that should occur in every annotated composite.xml file is <svcdoc:ServiceSolutionComponentAssociation>. This element, shown in Example 12–1, describes the globally unique identifier (GUID) used for artifacts generation and should be placed under the root element <composite>. The value for this element is not provided by the composite developer and is left blank. The GUID is used to associate the PM-identified Solution Service Component with the implemented Application Business Connector Service (ABCS) Composite. This association enables autopopulation into the Bill Of Material user interface (UI). The GUID is first auto generated in the Solution Service Component UI and it is persisted in the AIA Project Lifecycle Workbench database.

Example 12–1 First Annotation Element for Every composite.xml File

```
<composite name="SamplesCreateCustomerPartyPortalProvABCSImpl">
.......................
<!--<svcdoc:AIA>
            <svcdoc:ServiceSolutionComponentAssociation>
                        <svcdoc:GUID></svcdoc:GUID>
             </svcdoc:ServiceSolutionComponentAssociation>
 </svcdoc:AIA>-->
.................................
</composite>
```

You must annotate the Service and Reference elements of the Composite.xml in the manner presented in Example 12–2 and Example 12–3.

Example 12–2 A Skeletal Service Element in a composite.xml with Annotations

```
<service ui:wsdlLocation .......>
<interface.wsdl ....... />
        <binding.ws ........... />
         <!-- <svcdoc:AIA>
                      <svcdoc:Service>
                       . . . . . . . . . . . . .
                    </svcdoc:Service>
          </svcdoc:AIA> -->
        </service>
```

Example 12–3 A Skeletal Reference Element in a composite.xml with Annotations

```
<reference ui:wsdlLocation ........">
            <interface.wsdl ............./>
            <binding.ws ................/>
```

```
        <!-- <svcdoc:AIA>
                    <svcdoc:Reference>
                       ..................
                    </svcdoc:Reference>
                </svcdoc:AIA> -->

</reference>
```

As shown in Example 12–2 and Example 12–3 the root of the annotation element of the composite is <svcdoc:AIA>. Provide annotations for the Service and Reference elements of the Composite under xml comment tags using the annotation elements *<svcdoc:Service>* and *<svcdoc:Reference>*, respectively.

Figure 12–1 illustrates the annotations for Reference and Service elements.

Figure 12–1 Example of Annotations

12.1.2 Understanding the Service Annotation Element

The annotation element *<svcdoc:Service>* describes the details of the exposed service as denoted by the Service element of the Composite.xml. At a broader level, three annotation elements provide details about a service interface, its implementation, and the transport details (only if it is a transport adapter composite or if the composite contains as an adapter component).

These annotation elements are:

- **InterfaceDetails**

  This element provides information about the service interface, such as the name of the service, the name of the operation defined on the service, and the type of the service artifact.

- **ImplementationsDetails**

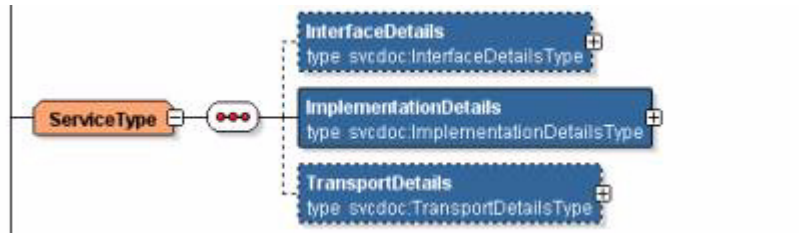
  This element provides information about the application with which the ABCS is interacting.

- **TransportDetails**

  This element provides information about the transport adapters and their details, if the composite has any transport adapters.

Figure 12–2 illustrates the annotations for interface, implementation, and transport details.

Figure 12–2 Example Of Annotation Elements



Note: Only when the Composite is that of an Adapter service are the Transport details provided.

For more information, see Section 12.1.4, "Understanding the TransportDetails Element".

12.1.2.1 InterfaceDetails

This element identifies the interface that is being implemented by the AIA service in consideration. The AIA service that implements an interface definition has these details annotated in its composite.xml. This element should provide details such as:

- The name of the service
- The name of the service operation
- The type of the service artifact.

The element **InterfaceDetails** is explained in detail in Table 12–1.

Table 12–1 Interface Details Elements

Element	Description
ServiceName	Identifies the name of the AIA service whose interface is being implemented by the composite under consideration. The Deployment Plan Generator uses this value.
Namespace	The namespace of the service whose interface is being implemented as defined in its WSDL.
ArtifactType	The type of service artifact, for example, an ABCS, and so on. The valid values are given here: Valid Values for Annotation Elements.
ServiceOperation/Name	The element that holds the complete service operation name in the format of verb + entity. The name of the operation of the AIA service whose interface is being implemented by the composite under consideration.
AdditionalServiceInformation	This element can be used to provide the additional service information like the supported application versions along with the base version or any other service related information Example:- This service supports create customer or billing profile ABM coming out of Siebel 8.0.x and 8.2 versions

Example 12–4 shows a sample of the interface details annotation element.

Example 12–4 Interface Details Annotation Element Example

```
<svcdoc:InterfaceDetails>
<svcdoc:ServiceName>CustomerPartyEBS</svcdoc:ServiceName>
```

```
<svcdoc:Namespace>http://xmlns.oracle.com/EnterpriseServices/Core/CustomerParty
/V2</svcdoc:Namespace>
         <svcdoc:ArtifactType>EnterpriseBusinessService</svcdoc:ArtifactType>
       <svcdoc:ServiceOperation>
       <svcdoc:Name>CreateCustomerPartyList</svcdoc:Name>
       </svcdoc:ServiceOperation>
<AdditionalServiceInformation>This service supports create customer or billing
profile ABM coming out of Siebel 8.0.x and 8.2
versions</AdditionalServiceInformation>
</svcdoc:InterfaceDetails>
```

Service Interface details are optional. The AIA artifacts that implement an interface definition have these details annotated. For example, a provider ABCS implements an interface defined by an Enterprise Business Service (EBS) operation, whereas a requester ABCS does not. So, annotations about the service interface will only be captured in the provider ABCS to identify which interface that particular ABCS is implementing.

Therefore, specify Service Interface details in the Composite.xml only if they are applicable.

12.1.2.2 ImplementationDetails

The annotation element *ImplementationDetails* describes the application for which the AIA service is being implemented. Table 12–2 lists the elements and provides a description for each element.

Table 12–2 ImplementationDetails Elements

Element	Description
ApplicationName	The name of the participating application with which the service is interacting.
B2BDocument	For requester or provider B2B Connector Services, this element contains the name of the B2B document type that is supported by the B2B Connector Service. An example of a B2B document type is an 850 (EDI ORDER). The value in this field should also match the DocumentType name in Oracle B2B.
B2BDocumentVersion	Contains the version of the B2B document type that is supported by the B2B Connector Service, for example, 4010. The value in this field should also match the DocumentRevision in Oracle B2B.
B2BStandard	Contains the name of the B2B standard/protocol that is supported by the B2B Connector Service, for example, EDI_X12, OAG, or RosettaNet.
B2BStandardVersion	Contains the version of the B2B standard/protocol that is supported by the B2B Connector Service, for example, 1.0.
BaseVersion	The version of the participating application with which the service is interacting.
DevelopedBy	The name of the Business Unit that developed the service. A possible value is ABSG.
OracleCertified	Indicates whether it is tested by Oracle or not.
ArtifactType	Describes the type of the service artifact, for example, whether it is an ABCS, and so on. The valid values are given here: Valid Values for Annotation Elements.
AdditionalServiceInformation	Provide any additional service information.

Table 12–2 (Cont.) ImplementationDetails Elements

Element	Description
ServiceOperation/Name	Holds the complete service operation name in the format of verb + entity. The name of the operation defined in the WSDL of the AIA service that is being implemented.

Example 12–5 shows a sample of the ImplementationDetails annotation element.

Example 12–5 ImplementationDetails Annotation Element Example

```
<svcdoc:ApplicationName>BRM</svcdoc:ApplicationName>
<svcdoc:ImplementationDetails>
                <svcdoc:BaseVersion>7.3</svcdoc:BaseVersion>
                 <svcdoc:DevelopedBy>Oracle</svcdoc:DevelopedBy>
                 <svcdoc:OracleCertified>Yes</svcdoc:OracleCertified>

<svcdoc:ArtifactType>ProviderABCSImplementation</svcdoc:ArtifactType>
                  <svcdoc:ServiceOperation>
                  <svcdoc:Name>CreateCustomer</svcdoc:Name>
                  </svcdoc:ServiceOperation>
   </svcdoc:ImplementationDetails>
```

12.1.3 Understanding the Reference Annotation Element

The annotation element *<svcdoc:Reference>* describes the Reference element of the composite. Figure 12–3 illustrates the Reference annotation element. The child elements of this element and their purpose are described below.

Figure 12–3 Reference Annotation Element Example

These annotation elements are:

- **ArtifactType**

  Describes the type of AIA artifact used at the Reference service.

- **ServiceOperation/Name**

  Holds the complete service operation name in the format of verb + entity, as defined on the Reference service.

- **TransportDetails**

  Provides information about the transport adapters and their details, if the composite has any transport adapters. See Section 12.1.4, "Understanding the TransportDetails Element" for details about the TransportDetails element.

Note: Only when the Composite is for an Adapter service would TransportDetails be provided.

- **AdditionalServiceInformation**

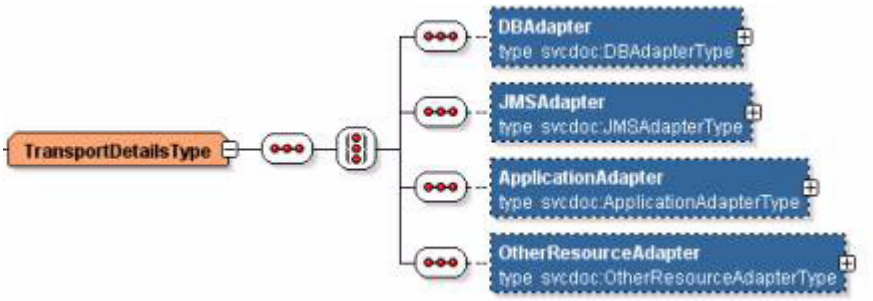
  This element can be used to provide the additional service information like the supported application versions along with the base version or any other service related information. This provides more information about the service in the OER. Example:- This service supports EBiz 11.x and 12.x application adapters

12.1.4 Understanding the TransportDetails Element

Use the annotation element **TransportDetails** to provide details about a transport adapter in a composite if a nonSOAP transport is used to interface with an application, either using inbound or outbound connectivity.

The structure of the element TransportDetails is shown in Figure 12–4.

Figure 12–4 Example of the TransportDetails Element



12.1.4.1 Annotating DBAdapter

If you give ApplicationName details for DBAdapter and set the ResourceType value to Business Event, then the deployment plan generator executes the SQL scripts on the participating application layer and the data source is created for the database in the participating application. If not, then Deployment Plan Generator executes the SQL scripts on the middleware layer, that is, on the same server where Pre-Built Integration or project is to be deployed.

Table 12–3 lists the elements found in the **DBAdapter** element and provides a description for each.

Table 12–3 DBAdapter Elements

Element	Description
ResourceProvider	Holds data about the Database Provider name. Examples: Oracle 10g, Oracle 11g, MS-SQL 9, MySQL 9. Based on this tag, the Deployment Plan Generator identifies and generate the appropriate configurations for the database.
ConnectionFactory	Holds the name of the connection factory. Using this value, the AIA installation driver creates data sources during deployment. The DB Adapter.rar file is updated during deployment. For example: eis/DB/AIASamplesDB

Table 12–3 (Cont.) DBAdapter Elements

Element	Description
ApplicationName	The name of the source or target application, depending on whether it is an inbound or outbound adapter. For example: EBizDB, SiebelDB
BaseVersion	The version of the application database with which the service is interacting.
XAEnabled	Used by the Deployment Plan Generator to decide if data source is to be configured using XA connection or not. Possible values are *true* and *false*. Based on the whether the tag is true or false, the AIA installation driver selects the appropriate JDBC driver and other configurations.
ResourceTargetIdentifier	Identifier that is used as the element name in the AIAInstallProperties.xml to retrieve database information while running the deployment plan. The xpath to retrieve database details is presented in DP as xpath, for example, pips.<projectCode_fromBOM>.db.<ResourceTargetIdentifier>. The AIA installation driver uses this element during deployment. Examples: EBIZ1, EBIZ2, JMSUSER1, JMSUSER2
Resource Type	Identifies what type of resource must be configured.
ResourceName	Holds data about database table name. The Deployment Plan Generator uses this element to create and configure data sources.
ResourceFileName	Holds data about SQL file name of the database table creation script. The Deployment Plan Generator uses this element to create and configure data sources.
AdditionalResourceInformation	Provide additional resource information such as the supported adapter versions or specifications or any other resource related information.

12.1.4.2 Annotating JMSAdapter

Table 12–4 lists the elements found in **JMSAdapter** and provides a description for each.

Table 12–4 JMSAdapter Elements

Element	Description
ResourceProvider	Holds data about JMS Provider name. Examples: WLSJMS, AQJMS, Tibco, MQ-Series, and SonicMQ. Based on this tag, the Deployment Plan Generator identifies whether it is AQJMS or WLSJMS and generate the appropriate configurations for AQ or WLS.
ConnectionFactory	Holds the name of the connection factory. Using this value, the AIA installation driver creates JMS modules during deployment. The JMS Adapter.rar file is updated during deployment. Example: eis/jms/AIA<Application Name>CF.
XAEnabled	The Deployment Plan Generator uses this element to decide whether the data source should be configured using XA connection. Possible values are *true* and *false*. Based on the whether this tag is true or false, the AIA installation driver selects the appropriate JDBC driver and other configurations.
ResourceTargetIdentifier	Can be either blank or filled in. If blank, AIAJMSModule is used to create resources on WLS. If filled in, AIAJDBCJMSModule is used to create resources on WLS.

Table 12–4 (Cont.) JMSAdapter Elements

Element	Description
ResourceType	Identifies what type of the JMS resource must be configured. The value can be *Queue* or *Topic*.
ResourceName	Holds data about the Queue or Topic name. The AIA installation driver uses this element to create and configure data sources.
ResourceFileName	Holds data about the SQL file name of the database table creation script. The AIA installation driver uses the SQL file to create and configure data sources.
AdditionalResourceInformation	Provide the additional resource information such as the supported adapter versions or specifications or any other resource related information. This provides more information about the service in the OER.

12.1.4.3 Annotating AQJMS Adapter

Table 12–5 lists the elements found in **AQJMSAdapter** and provides a description for each.

Table 12–5 AQJMSAdapter Elements

Element	Description
ResourceProvider	Holds data about AQJMS Provider name. Examples: AQJMS. Based on this tag, the Deployment Plan Generator generates the appropriate configurations for AQJMS.
ConnectionFactory	Holds the name of the connection factory. Using this value, the AIA installation driver creates AQJMS modules during deployment. The AQ Adapter .rar file is updated during deployment. Example: eis/aqjms/ AIA<Application Name>CF.
XAEnabled	The Deployment Plan Generator uses this element to decide whether the data source should be configured using XA connection. Possible values are *true* and *false*. Based on the whether this tag is true or false, the AIA installation driver selects the appropriate JDBC driver and other configurations.
JMSForeignServerName	Value used to create JMSForeignServer under JMSModule
ResourceTargetIdentifier	Identifier used as the element name in the AIAInstallProperties.xml to retrieve database information while running the deployment plan. The xpath to retrieve database details is presented in DP as xpath, for example, pips.<projectCode_fromBOM>.db.<ResourceTargetIdentifier>.
ResourceType	Identifies what type of the AQ resource must be configured. The value can be *Queue* or *Topic*.
ResourceName	Holds data about the Queue or Topic name. The Deployment Plan Generator uses this element to create and configure data sources.
ResourceFileName	Holds data about the SQL file name of the database table creation script. The AIA installation driver uses to create and configure data sources.
AdditionalResourceInformation	Provide additional Resource information such as the supported adapter versions or specifications or any other resource related information. This provides more information about the service in the OER.

12.1.4.4 Annotating Other Resources

To use other resources like File or FTP adapters or any other resource, you must provide the following annotations. The Deployment Plan Generator does not have support for all the resource adapters. For more information on supported resource adapters, refer to the Foundation Pack release notes.

Table 12–6 lists the elements used to annotate other resources and provides a description for each.

Table 12–6 Elements for Other Resources

Element	Description
ResourceProvider	Holds data about Resource Provider name like FILE or FTP or AQ.
ConnectionFactory	Holds the name of the connection factory. Using this value, the AIA installation driver can create JNDI reference to the target URI.
BaseVersion	The version of the application with which the service is interacting. This provides more information about the service in the OER.
XAEnabled	The Deployment Plan Generator uses this element to decide whether the data source should be configured using XA connection. Possible values are *true* and *false*. Based on the whether this tag is true or false, the AIA installation driver selects the appropriate JDBC driver and other configurations.
ResourceTargetIdentifier	The value of the application product code name. The AIA installation driver uses this element during deployment. If ResourceTargetIdentifier is used for AQAdapter, then it is the identifier that is used as the element name in the AIAInstallProperties.xml to retrieve database information while running the deployment plan. The xpath to retrieve database details is presented in DP as xpath, for example, pips.<projectCode_fromBOM>.db.<ResourceTargetIdentifier>.
ResourceName	Holds data about the Queue or Topic name. The AIA installation driver use this element to create and configure data sources.
ResourceType	Identifies what type of the resource must be configured.
ResourceFileName	Holds data about the SQL file name of the database table creation script or Shall scripts to create the file folders at a specified location. The AIA installation driver uses to create and configure these sources.
AdditionalResourceInformation	Provide the additional Resource information like the supported adapter versions or specifications or any other resource related information. This provides more information about the service in the OER.
ResourceLocation	Provide the information about the location or relative path of the file. This is used to create the source or target folders at appropriate local or remote locations.

12.1.4.5 Annotating Application Adapter

If you give ApplicationName details for EBiz Adapter and set the ResourceType value to Business Event, then the deployment plan generator executes the SQL scripts on the edge application layer and the data source is created for the database in the participating application. If not, then Deployment Plan Generator executes the SQL scripts on the middleware layer, that is, on the same server where Pre-Built Integration/project is to be deployed.

Table 12–7 lists the elements found in **ApplicationAdapter** and provides a description for each.

Table 12–7 Application Adapter Elements

Element	Description
ResourceProvider	Holds data about EBiz Provider name.
ConnectionFactory	Holds the name of the connection factory. Using this value, the AIA installation driver creates EBiz modules during deployment. The EBiz Adapter .rar file is updated during deployment.
Application Name	The name of the source or target application, depending on whether it is an inbound or outbound adapter.
BaseVersion	The version of the application with which the service is interacting. This provides more information about the service in the OER.
XAEnabled	The Deployment Plan Generator uses this element to decide whether the data source should be configured using XA connection. Possible values are *true* and *false*. Based on the whether this tag is true or false, the AIA installation driver selects the appropriate JDBC driver and other configurations.
ResourceTargetIdentifier	The value of the application product code name. The AIA installation driver uses this element during deployment. If ResourceType is blank, database information is retrieved from pip.<projectCode_fromBOM>.db.ResourceTargetIdentifier element in the AIAInstallProperties.xml during execution of deployment plan. If ResourceType is BusinessEvent or Procedure, database details are retrieved from participatingapplications.<ApplicationName>.db
ResourceName	Holds data about the Queue or Topic name. The AIA installation driver uses this element to create and configure data sources.
ResourceType	Identifies what type of the AQ resource must be configured. The value can be *Queue* or *Topic*. If values are *BusinessEvent* or *Procedure*, dp retrieves database information from under participatingapplications.<ApplicationName>.db in AIAInstallProperties.xml
ResourceFileName	Holds data about the SQL file name of the database table creation script. The AIA installation driver uses to create and configure data sources.
AdditionalResourceInformation	Provide the additional Resource information such as the supported adapter versions or specifications or any other resource related information. This provides more information about the service in the OER.

12.2 How to Annotate the Service Element in a Requester ABCS Composite

To annotate the service element in a requester ABCS composite:

The details of the source participating application must be furnished in this element.

Annotating the Service element in the composite is explained using the Example 12–6:

Example 12–6 Service Element in Requester ABCS Composite Annotation Example

```
<service ui:wsdlLocation……..">
        <interface.wsdl ……./>
        <binding.ws …….../>
         <!--<svcdoc:AIA>
                   <svcdoc:Service>
```

```
                        <svcdoc:ImplementationDetails>

<svcdoc:ApplicationName>SampleSEBL</svcdoc:ApplicationName>
                            <svcdoc:BaseVersion>1.0</svcdoc:BaseVersion>
                            <svcdoc:DevelopedBy>Oracle</svcdoc:DevelopedBy>
                            <svcdoc:OracleCertified>Yes</svcdoc:OracleCertified>

<svcdoc:ArtifactType>RequesterABCSImplementation</svcdoc:ArtifactType>
                            <svcdoc:ServiceOperation>
                                <svcdoc:Name>CreateCustomer</svcdoc:Name>
                            </svcdoc:ServiceOperation>
                        </svcdoc:ImplementationDetails>
                    </svcdoc:Service>

          </svcdoc:AIA>-->
    </service>
```

In Example 12–6, the value of the element, **ArtifactType,** is provided as *RequesterABCSImplementation* because the composite represents a Requester ABCS.

The value of the element, **ServiceOperation/Name**, should be same as the value defined for the operation in the WSDL of the ABCS.

12.3 How to Annotate the Reference Element in a Requester ABCS Composite

To annotate the Reference Element in a requester ABCS composite:

1. Annotate the Reference Element in the composite, as shown in Example 12–7, providing the details of the Service being invoked.

Example 12–7 Reference Element in Requester ABCS Composite Annotation Example

```
<reference ui:wsdlLocation.....">
        <interface.wsdl .....>/>
        <binding.ws......./>
        <!--<svcdoc:AIA>
                    <svcdoc:Reference>

<svcdoc:ArtifactType>EnterpriseBusinessService</svcdoc:ArtifactType>
                        <svcdoc:ServiceOperation>
                        <svcdoc:Name>CreateCustomerPartyList</svcdoc:Name>
                        </svcdoc:ServiceOperation>
                    </svcdoc:Reference>
                </svcdoc:AIA>-->
</reference>
```

- In Example 12–7, the value of the element, **ArtifactType**, is provided as EnterpriseBusinessService because it references an external service in the composite.
- In cases when the external service is an infrastructure utility, such as AIAAsyncErrorHandlingBPELProcess, then the value should be *UtilityService*.
- The value of the element, **ServiceOperation/Name**, should be the same as the value defined for the operation in the WSDL of the service being referenced.

12.4 How to Annotate the Service Element in a Provider ABCS Composite

To annotate the Service Element in a provider ABCS composite:

1. Furnish the details of the interface that this ABCS is implementing.
2. Match the values for elements **ServiceName**, **Namespace**, and **ServiceOperation/name** with the corresponding values defined in the interface service's WSDL target participating application as shown in Example 12–8.

Example 12–8 Service Element in a Provider ABCS Composite Annotation Example

```
<service ui:wsdlLocation=.....>
        <interface......./>
        <binding.ws ..../>
        <!--<svcdoc:AIA>
                    <svcdoc:Service>
                      <svcdoc:InterfaceDetails>

<svcdoc:ServiceName>CustomerPartyEBS</svcdoc:ServiceName>

<svcdoc:Namespace>http://xmlns.oracle.com/EnterpriseServices/Core/CustomerParty
/V2</svcdoc:Namespace>

<svcdoc:ArtifactType>EnterpriseBusinessService</svcdoc:ArtifactType>
                         <svcdoc:ServiceOperation>
                            <svcdoc:Name>CreateCustomerPartyList</svcdoc:Name>
                         </svcdoc:ServiceOperation>
                       </svcdoc:InterfaceDetails>
                       <svcdoc:ImplementationDetails>

<svcdoc:ApplicationName>SamplePortal</svcdoc:ApplicationName>
                         <svcdoc:BaseVersion>1.0</svcdoc:BaseVersion>
                         <svcdoc:DevelopedBy>Oracle</svcdoc:DevelopedBy>
                         <svcdoc:OracleCertified>Yes</svcdoc:OracleCertified>

<svcdoc:ArtifactType>ProviderABCSImplementation</svcdoc:ArtifactType>
                         <svcdoc:ServiceOperation>
                            <svcdoc:Name>CreateCustomer</svcdoc:Name>
                         </svcdoc:ServiceOperation>
                       </svcdoc:ImplementationDetails>
                    </svcdoc:Service>

          </svcdoc:AIA>-->
    </service>
```

- In Example 12–8, the value of the element, **InterfaceDetails/ArtifactType**, is provided as *EnterpriseBusinessService* because it defines the interface that is being implemented by the Provider ABCS.
- **ImplementationDetails/ArtifactType**, is provided as *ProviderABCSImplementation* because the composite represents a provider ABCS.
- The value of the element, **ServiceOperation/Name**, should be same as the value defined for the operation in the WSDL of the ABCS.

12.5 How to Annotate the Reference Element in a Provider ABCS

Annotate the Reference element in the composite, as shown in Example 12–9, Example 12–10, and Example 12–11 providing the details of the Service. The service

being invoked can be a participating application, an adapter service, or a utility service.

Example 12–9 Reference Element in Provider ABCS Invoking Participating Web Service Annotation Example

```
                  <!--<svcdoc:AIA> ;
                         <svcdoc:Reference>

<svcdoc:ArtifactType>UtilityService</svcdoc:ArtifactType>
                               <svcdoc:ServiceOperation>
                                   <svcdoc:Name>initiate</svcdoc:Name>
                               </svcdoc:ServiceOperation>
                            </svcdoc:Reference>
              </svcdoc:AIA>-->
<!--<svcdoc:AIA>
              <svcdoc:Reference>

<svcdoc:ArtifactType>ApplicationWebService</svcdoc:ArtifactType>
                        <svcdoc:ServiceOperation>
                            <svcdoc:Name>insert</svcdoc:Name>
                        </svcdoc:ServiceOperation>
                     </svcdoc:Reference>
         </svcdoc:AIA>-->
```

Example 12–10 Reference Element in Provider ABCS Invoking Utility Service Annotation Example

```
<!--<svcdoc:AIA>
<svcdoc:Reference>

<svcdoc:ArtifactType>UtilityService</svcdoc:ArtifactType>
                               <svcdoc:ServiceOperation>
                                   <svcdoc:Name>initiate</svcdoc:Name>
                               </svcdoc:ServiceOperation>
                            </svcdoc:Reference>
                     </svcdoc:AIA>-->
```

Example 12–11 Reference Element in Provider ABCS Invoking Non-SOAP Service Annotation Example

```
<!--<svcdoc:AIA>
<svcdoc:Reference>

<svcdoc:ArtifactType>TransportAdapter</svcdoc:ArtifactType>

<svcdoc:ServiceOperation>

<svcdoc:Name>process</svcdoc:Name>

</svcdoc:ServiceOperation>
                                                    </svcdoc:Reference>
                   </svcdoc:AIA>-->
```

12.6 How to Annotate the Transport Adapter Composite

To annotate the Transport Adapter composite:

1. For a Transport Adapter composite, populate the element, **TransportDetails**, under:
   - **Service** if nonSOAP transport is used to interface with this service.
   - **Reference** if the service uses nonSOAP transport to interface with participating applications /external systems.
2. In both cases, the values for the element **ArtifactType** are provided as *TransportAdapter*.

   The artifact type **TransportAdapter** indicates that the service is responsible for transforming nonSOAP requests into SOAP requests and vice versa.

   Example 12–12 illustrates how transport details are populated in the section **Service**.

Example 12–12 Transport Adapter Composite Annotation Example

```
<!--<svcdoc:AIA>
<svcdoc:Service>
      <svcdoc:ImplementationDetails>
<svcdoc:ApplicationName>SamplePortal</svcdoc:ApplicationName>
          <svcdoc:BaseVersion>1.0</svcdoc:BaseVersion>

<svcdoc:DevelopedBy>ABSG</svcdoc:DevelopedBy>

<svcdoc:OracleCertified>Yes</svcdoc:OracleCertified>

<svcdoc:ArtifactType>TransportAdapter</svcdoc:ArtifactType>
          <svcdoc:ServiceOperation>
              <svcdoc:Name>process</svcdoc:Name>
          </svcdoc:ServiceOperation>
      </svcdoc:ImplementationDetails>
   </svcdoc:Service>
</svcdoc:AIA>-->
```

Example 12–13 depicts how transport details are populated in the section **Reference.**

Example 12–13 Transport Details Populated in Reference Element Annotation Example

```
<!--<svcdoc:AIA>
<svcdoc:Reference>

<svcdoc:ArtifactType>TransportAdapter</svcdoc:ArtifactType>
            <svcdoc:ServiceOperation>
                <svcdoc:Name>insert</svcdoc:Name>
            </svcdoc:ServiceOperation>
            <svcdoc:TransportDetails>
                <svcdoc:DBAdapter>

<svcdoc:ResourceProvider>OracleDB</svcdoc:ResourceProvider>

<svcdoc:ConnectionFactory>eis/db/AIASamplesDB</svcdoc:ConnectionFactory>

<svcdoc:ApplicationName>SamplePortal</svcdoc:ApplicationName>
                        <svcdoc:XAEnabled>True</svcdoc:XAEnabled>
```

```
<svcdoc:ResourceTargetIdentifier>AIASamplesDB</svcdoc:ResourceTargetIdentifier>
            <svcdoc:ResourceName>AIAS_PORTAL_ACC_CONTACT</svcdoc:ResourceName>

<svcdoc:ResourceName>AIAS_PORTAL_ACCOUNT</svcdoc:ResourceName>

<svcdoc:ResourceFileName>AIAS_PORTAL_ACCOUNT_CONTACT.sql
           </svcdoc:ResourceFileName>
         </svcdoc:DBAdapter>
       </svcdoc:TransportDetails>
     </svcdoc:Reference>
</svcdoc:AIA>-->
```

In Example 12–13:

- The element <*svcdoc:ResourceTargetIdentifier*> denotes the database schema used for the project.
- The element <*svcdoc:ResourceName*> denotes the database table used. This element may be repeated for each database table when multiple tables are used.
- The element <*svcdoc:ResourceFileName*> should have all the SQLs and sequences required included in the file. Also, it should provide the correct order for the SQLs, in which SQLs must be executed.

Example 12–14 shows the code when a JMS Adapter is used.

Example 12–14 Transport Details Populated in Reference Element Using JMS Adapter Annotation Example

```
<!--<svcdoc:AIA>
<svcdoc:Reference>

<svcdoc:ArtifactType>TransportAdapter</svcdoc:ArtifactType>
           <svcdoc:ServiceOperation>
               <svcdoc:Name>Produce</svcdoc:Name>
           </svcdoc:ServiceOperation>
           <svcdoc:TransportDetails>
           <svcdoc:JMSAdapter>

<svcdoc:ResourceProvider>WLSJMS</svcdoc:ResourceProvider>

<svcdoc:ConnectionFactory>eis/jms/AIASamplesCF</svcdoc:ConnectionFactory>
           <svcdoc:XAEnabled>True</svcdoc:XAEnabled>

<svcdoc:ResourceTargetIdentifier>JMSUSER1</svcdoc:ResourceTargetIdentifier>
     <svcdoc:ResourceType>Queue</svcdoc:ResourceType>
     <svcdoc:ResourceName>AIA_SiebelCustomerJMSQueue</svcdoc:ResourceName>
<svcdoc:ResourceFileName>AIASiebelCustomerJMSQueue.sql</svcdoc:ResourceFileName>
     </svcdoc:JMSAdapter>
   </svcdoc:TransportDetails>
  </svcdoc:Reference>
</svcdoc:AIA>-->
```

12.7 How to Annotate the Service Element in Enterprise Business Flow Composite

To annotate the Service Element in Enterprise Business Flow composite:

1. Furnish the details of the EBS interface that the Enterprise Business Flow (EBF) implements.

2. Match the values for elements - **ServiceName, Namespace, ServiceOperation/name** with the corresponding values defined in the interface service's WSDL as shown below.

   In Example 12–15 the value of the element:

   - **InterfaceDetails/ArtifactType** is provided as *EnterpriseBusinessService* because it defines the interface that is being implemented by the EBF.
   - **ImplementationDetails/ArtifactType** is provided as *EnterpriseBusinessFlow* because the composite represents an EBF.
   - **ImplementationDetails/ApplicationName** is provided as *AIA* because the composite is participating application-agnostic.

The value of the element, **ServiceOperation/Name,** should be same as the value defined for the operation in the EBS WSDL.

Example 12–15 Service Element in EBF Composite Annotation Example

```
<service ui:wsdlLocation=.....>
<interface......./>
<binding.ws ......./>
<!--<svcdoc:AIA>
<svcdoc:Service>
<svcdoc:InterfaceDetails>

<svcdoc:ServiceName>DoCreditCheckCustomerPartyEBS</svcdoc:ServiceName>

<svcdoc:Namespace>http://xmlns.oracle.com/EnterpriseServices/Core/CustomerParty
/V2</svcdoc:Namespace>

<svcdoc:ArtifactType>EnterpriseBusinessService</svcdoc:ArtifactType>
<svcdoc:ServiceOperation>

<svcdoc:Name>DoCreditCheckCustomerParty</svcdoc:Name>
</svcdoc:ServiceOperation>
</svcdoc:InterfaceDetails>
<svcdoc:ImplementationDetails>
<svcdoc:ApplicationName>AIA</svcdoc:ApplicationName>
<svcdoc:BaseVersion>1.0</svcdoc:BaseVersion>
<svcdoc:DevelopedBy>Oracle</svcdoc:DevelopedBy>
<svcdoc:OracleCertified>Yes</svcdoc:OracleCertified>

<svcdoc:ArtifactType>EnterpriseBusinessFlow</svcdoc:ArtifactType>
<svcdoc:ServiceOperation>
<svcdoc:Name>DoCreditCheck</svcdoc:Name>
</svcdoc:ServiceOperation>
</svcdoc:ImplementationDetails>
</svcdoc:Service>

</svcdoc:AIA>-->
</service>
```

12.8 How to Annotate the Reference Element in Enterprise Business Flow Composite

To annotate the Reference Element in an EBF composite:

Annotate the Reference element in the composite, as shown in Example 12–16, providing the details of the Service being invoked.

Example 12–16 Reference Element in EBF Composite Annotation Example

```
<reference ui:wsdlLocation.....">
        <interface.wsdl ......./>
        <binding.ws......./>
        <!--<svcdoc:AIA>
                    <svcdoc:Reference>

<svcdoc:ArtifactType>EnterpriseBusinessService</svcdoc:ArtifactType>
                        <svcdoc:ServiceOperation>

<svcdoc:Name>GetCreditScoreCustomerPartyList</svcdoc:Name>
                        </svcdoc:ServiceOperation>
                    </svcdoc:Reference>
                </svcdoc:AIA>-->
</reference>
```

- In Example 12–16, the value of the element, **ArtifactType**, is provided as *EnterpriseBusinessService* because it is the referenced external service in the composite.
- When the external service is an infrastructure utility, such as *AIAAsyncErrorHandlingBPELProcess*, then the value should be **UtilityService**.
- The value of the element, **ServiceOperation/Name**, should be same as the value defined for the operation in the WSDL of the service to which it refers.

12.9 How to Annotate the Service Element in Composite Business Process Composite

To annotate the Service Element in a composite business process composite:

1. Furnish the details of the interface that the EBF is implementing.

   This could be an application specific interface, a UI service interface, or canonical interface.

2. Match the values for elements **ServiceName, Namespace,** and **ServiceOperation/name** with the corresponding values defined in the WSDL of the interface service shown below.

   In Example 12–17, the value of the element:

   - **InterfaceDetails/ArtifactType** is provided as **UIService** because it defines the interface that is being implemented by the Composite Business Process (CBP).
   - **ImplementationDetails/ArtifactType** is provided as *CompositeBusinessProcess* because the composite represents a CBP.
   - **ImplementationDetails/ApplicationName** is provided as *AIA* because the composite, in this case, is participating application-agnostic.

3. The value of the element, **ServiceOperation/Name**, should be same as the value defined for the operation in the UI service WSDL.

Example 12–17 Service Element in Composite Business Process Composite Annotation Example

```
<service ui:wsdlLocation=.....>
        <interface......./>
        <binding.ws ..... ../>
        <!--<svcdoc:AIA>
                    <svcdoc:Service>
                      <svcdoc:InterfaceDetails>

<svcdoc:ServiceName>TelcoResolveCustomerComplaintCBP</svcdoc:ServiceName>

<svcdoc:Namespace>http://xmlns.oracle.com/UIServices/Industry/CustomerParty/V1
</svcdoc:Namespace>
                         <svcdoc:ArtifactType>UIService</svcdoc:ArtifactType>
                         <svcdoc:ServiceOperation>

<svcdoc:Name>TelcoResolveCustomerComplaint</svcdoc:Name>
                         </svcdoc:ServiceOperation>
                      </svcdoc:InterfaceDetails>
                      <svcdoc:ImplementationDetails>
                         <svcdoc:ApplicationName>AIA</svcdoc:ApplicationName>
                         <svcdoc:BaseVersion>1.0</svcdoc:BaseVersion>
                         <svcdoc:DevelopedBy>Oracle</svcdoc:DevelopedBy>
                         <svcdoc:OracleCertified>Yes</svcdoc:OracleCertified>

<svcdoc:ArtifactType>CompositeBusinessProcess</svcdoc:ArtifactType>
                         <svcdoc:ServiceOperation>

<svcdoc:Name>TelcoResolveCustomerComplaint</svcdoc:Name>
                         </svcdoc:ServiceOperation>
                      </svcdoc:ImplementationDetails>
                   </svcdoc:Service>

          </svcdoc:AIA>-->
    </service>
```

12.10 How to Annotate the Reference Element in Composite Business Process Composite

To annotate the Reference Element in a Composite Business Process composite:

Annotate the Reference element in the composite, as shown in tExample 12–18 by providing the details of the Service being invoked.

Example 12–18 Reference Element in Composite Business Process Composite Annotation Example

```
<reference ui:wsdlLocation.....">
        <interface.wsdl ..... ../>
        <binding.ws..... ../>
        <!--<svcdoc:AIA>
                    <svcdoc:Reference>
```

```
<svcdoc:ArtifactType>EnterpriseBusinessService</svcdoc:ArtifactType>
                        <svcdoc:ServiceOperation>
                        <svcdoc:Name>GetComplaintDetails</svcdoc:Name>
                        </svcdoc:ServiceOperation>
                    </svcdoc:Reference>
                </svcdoc:AIA>-->
</reference>
```

- In Example 12–18, the value of the element, **ArtifactType**, is provided as *EnterpriseBusinessService* because it is the referenced external service in the composite.
- When the external service is an infrastructure utility, such as, AIAAsyncErrorHandlingBPELProcess, then the value should be **UtilityService**.
- The value of the element, **ServiceOperation/Name**, should be same as the value defined for the operation in the WSDL of the service being referred to.

12.11 Valid Values for Annotation Elements

This section lists the valid values for the annotation elements:

- ArtifactType
- ApplicationName

12.11.1 Valid Values for the Element ArtifactType

- RequesterABCSImplementation
- RequesterABCSExtension
- ProviderABCSImplementation
- ProviderABCSExtension
- EnterpriseBusinessService
- EnterpriseBusinessFlow
- EnterpriseBusinessFlowExtension
- CompositeBusinessProcess
- CompositeBusinessProcessExtension
- ApplicationService
- ExternalService
- UtilityService
- TransportAdapter
- VersionAdapter
- Other

12.11.2 Valid Values for the Element ApplicationName

- AIA
- PeopleSoft

- BRM
- FAH
- UCM
- SAP
- PIM
- OracleRetail
- Logistics
- JDEE1
- CRMOD
- Agile
- Ebiz
- Siebel
- OUCCB - Oracle Utilities Customer Care and Billing
- OUWAM - Oracle Utilities Work and Asset Management
- OUMWM - Oracle Utilities Mobile Workforce Management

13

Designing and Developing Enterprise Business Services

This chapter provides an overview of Enterprise Business Services (EBS) and describes how to design EBS, construct the WSDL for the process EBS, work with message routing, build EBS using Oracle Mediator, implement the Fire-and Forget Message Exchange Pattern, implement Synchronous Request-Response Message Exchange Pattern, and implement the Asynchronous Request-Delayed Response Message Exchange Pattern.

This chapter includes the following sections:

13.1 Introduction to Enterprise Business Services

EBSs are the foundation blocks in Oracle Application Integration Architecture (AIA). An EBS represents the application or implementation-independent Web service definition for performing a task, and the architecture facilitates distributed processing using EBS. Since an EBS is self-contained, it can be used independently of any other services. In addition, it can be used within another EBS.

For more information about EBS, see "Understanding Enterprise Business Services" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*.

You must construct an EBS when the business process integration is between multiple source applications and target applications using the canonical model.

The purpose of the EBS is to:

- Provide the mediation between the requesting services and providing services.

- Provide different operations invoked from a requester Application Business Connector Service (ABCS), an EBS, or an Enterprise Business Flow (EBF).
- Route an operation to a suitable EBS, EBF, or provider ABCS based on the evaluation of the various routing rules for an operation.

Oracle AIA leverages Mediator technology available in Oracle SOA Suite to build the EBS.

The EBS is implemented as a Mediator routing service. A Mediator service has an elaborate mechanism to hold multiple operations of the EBS, create routing rules for each operation, perform XSLT transformation, and define endpoints for each routing rule.

For more information about using Oracle Mediator, see "Using the Oracle Mediator Service Component" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

You can model EBS operations either as synchronous or asynchronous message exchange patterns (MEPs).

13.1.1 Understanding EBS Types

The types of EBS are:

- **Business Activities**

  They represent an atomic business unit of work that has a set of steps involving system-to-system interaction. They are exposed as mediator services with implementations through ABCSs or EBFs.

- **Tasks**

  They provide an aggregated, real-time view of enterprise data. They are primarily **Create, Read, Update, Delete (CRUD)** operations acting on the Enterprise Business Object (EBO) and its business components. They are exposed as mediator services with implementations through ABCS. They eliminate point-to-point links at the data level and direct dependency on data models of data sources.

For more information about EBS types, see "Understanding Enterprise Business Services" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*.

13.1.2 Working with the Enterprise Business Service Library

AIA Foundation Pack is shipped with an EBS library. The EBS library consists of the service definitions delivered for the all of the EBOs present in the Enterprise Object Library. These are shipped as WSDL files. The service operations present in the EBS typically are the CRUD operations and some operations specific to entities.

All the operations of the EBS WSDLs, of type Data Services, in the Enterprise Business Service Library are modeled as asynchronous one-way services. The only exceptions are the **Query** operations and **Validate** operations. These are modeled as synchronous request-response operations with a named fault.

- You can review the sample WSDLs provided in the AIA Foundation Pack under the AIAComponents/EnterpriseServiceLibrary folder.
- Review each of the WSDLs, the operation's description, and the metadata before deciding to create either a new service or an operation.

- Any new EBS that you create must be of type **Activity Service**, with operations put in to meet the requirements of integration solution being developed.
- The new EBS should be put in a different WSDL and not added to the entity EBS WSDLs.

13.2 Designing the EBS

This section includes the following topics:

13.2.1 Understanding Design Guidelines

The methodology for designing and implementing an EBS is a contract-first methodology, that is, the contract is defined and created before the EBS is implemented. The contract for an EBS is defined as a WSDL document.

For the task EBS, use the WSDLs from the Enterprise Service Library of the Foundation Pack.

For the business activity EBS, use the TemplateEBS.wsdl available in the Foundation Pack and create a process EBS WSDL. Customers wanting to create an EBS for an EBO that was not delivered as part of Enterprise Object Library can use TemplateEBS.wsdl as a model to create the new WSDL.

Service operations supporting the synchronous request-response MEP are defined in one port type. The operation should have input, output, and fault message defined.

Service operations supporting fire-and-forget MEP should be defined in one port type and the operation must have an input message defined.

Service operations supporting an asynchronous request-delayed response pattern should have two operations, one for sending the request message and another for processing the response message. Each of these two operations must have an input message. Two different portTypes exist, one for each operation. The service operation for processing the response message should reside in a port type that has **Response** as the suffix.

The EBS WSDLs should have two kinds of portTypes:

- A portType for all operations used for modeling synchronous request-response operations and request-only operations. The name does not specify the Request.
- A portType for asynchronous response operations. The name specifies Response.

You should create two Mediator routing services for each of the portTypes.

13.2.2 Understanding Design Considerations

When designing EBS, consider the following:

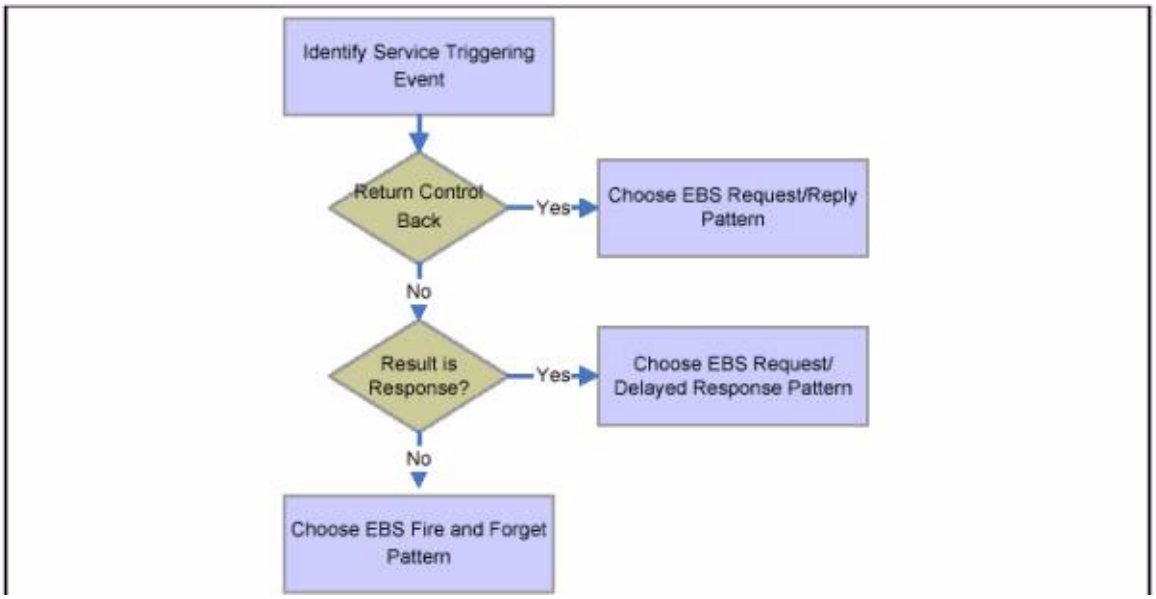
- What is the service for?
  - Your enterprise should have only one EBS for a particular business function, and it should be independent of any application.
  - The service can implement multiple business operations defined in EBS WSDL.
  - It should have implementations for all the business operations for a business object.
  - The service should also contain response operations, which are required for the asynchronous request-response pattern.
- How will the service be invoked?
  - The service is invoked using HTTP transport as a Web service.
  - When the calling client is co-located with this service, then the call can be configured to be a local invocation using optimized binding to improve performance and propagate transactions.
- What invokes the service?
  - The service can be invoked by an application if the application can send an Enterprise Business Message (EBM) and can perform all the functions of requester ABCS.
  - When the application does not have the capability to invoke this service directly, then the requester ABCS invokes this service.
  - An EBF flow that orchestrates across multiple business objects can also invoke this service.
- How do you interact with the application?
  - If the application can receive the EBM message, the EBS can call the application using HTTP as Web service call.
  - If not, then you must create and call a provider ABCS.
- What type of interaction is between client and EBS?
  - The client can call EBS using any of the three MEPs.

13.2.3 Establishing the MEP of a New Process EBS

Since the MEPs for the entity EBS WSDLs in the EBS library are predefined, you must design the process EBS WSDLs. The EBS is modeled to have multiple operations. Each operation leads to the execution of the EBS for a particular business scenario requirement and is granular in nature. Thus, each operation can be modeled to support a different interaction style or pattern.

Figure 13–1 illustrates the decision points in establishing the EBS pattern.

Figure 13–1 Identifying the Interaction Pattern for EBS Operations



For more information about EBS types, see "Understanding Enterprise Business Services" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*.

13.2.4 How to Establish the MEP for a New Process EBS

To establish the MEP for a new process EBS:

1. Identify the triggering event for the EBS operation based on the understanding of the business process requirement from the functional design.
2. If the control is to be blocked until a response is returned to the point of invocation, then choose the EBS request-reply pattern.

   This is a synchronous call. In this case, the EBS operation has input and output messages with a named fault.
3. If, after the EBS is invoked, the triggering point does not wait for the response and continues, this invocation of the EBS would be an asynchronous call.
4. Next, check whether the execution of the EBS results in a response.

   Should the request and the response be correlated? If the answer is yes, then this is a delayed response. Use the EBS request-delayed response pattern. In this case, the EBS has two portTypes, each of which accepts an input message only and each of them belongs to a different port.

   If the answer is no, then choose the EBS fire-and-forget pattern. In this case, the EBS operation has an input message only.

13.2.5 How to Handle Errors

The EBS should be configured to rethrow the errors back to the invoking client. Ensure that the application error handling capabilities are in line with the integration platform error handling capabilities.

For more information about error handling, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging".

13.2.6 How to Secure the EBS

When the invoking client (requester ABCS or application) is remote, the EBS must be enabled with security as described in the security chapter.

For more information about security, see Chapter 28, "Working with Security."

For more information about EBS, see "Understanding Enterprise Business Services" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*.

13.2.7 How to Configure Transactions

Based on the SOA transaction semantics in Oracle Fusion Middleware, you can design and configure transactions across ABCS, EBS, and EBF.

For more information, see Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite.

13.2.8 How to Guarantee Delivery

For details about how to guarantee message delivery, see Section 16.7, "Guaranteed Message Delivery".

13.2.9 How to Define the EBS Service Contract

To define the EBS service contract:

1. Identify the EBS and the operations it must support.
2. Identify the interaction patterns for each of the operations in EBS.
3. Identify the EBMs to be used for the requests and responses (if any) pertaining to each of the operations.

13.3 Constructing the WSDL for the Process EBS

This section includes the following topics:

- Section 13.3.1, "Introduction to WSDL Construction for the Activity Service EBS"
- Section 13.3.2, "How to Complete the <definitions> Section"
- Section 13.3.3, "How to Define Message Structures"
- Section 13.3.4, "How to Check for WS-I Basic Profile Conformance"

These sections describe how to fill in sections of the WSDL.

13.3.1 Introduction to WSDL Construction for the Activity Service EBS

When constructing the WSDL, consider that:

- Oracle's JDeveloper and text editing tools can be used to create the WSDL.
- The EBM schema module for the EBO must be referenced in the WSDL.
- The WSDL should reference the schema modules hosted on the central location.

- Annotations should be done based on the recommendations provided in Chapter 12, "Annotating Composites".
- You should adhere to the naming conventions for creating service, target namespace, messages, operations, port type, and so on as described in Chapter 31, "Oracle AIA Naming Standards for AIA Development".

13.3.2 How to Complete the <definitions> Section

To complete the <definitions> section:

1. Name

   The service contains operations related to creation and maintenance, and actions to be performed on an EBO.

2. Namespace

   Mark this namespace as the target namespace.

3. Namespace prefixes

   Define a namespace prefix for the newly identified namespace:

   - *wsdl* is the namespace prefix for the default namespace.
   - *xsd* is the namespace prefix for XMLSchema.
   - *soap* is the namespace prefix for the SOAP namespace.

   A namespace prefix must be defined for the EBO. The namespace prefix should be the same as the EBS WSDL in the delivered EBOs.

13.3.3 How to Define Message Structures

To define message structures in the WSDL types section:

1. Import the appropriate schema that has all of the relevant EBMs defined.
2. Use **xsd:import** to import the elements from a schema.

   Ensure that the namespace attribute for the xsd:import element is the same as the target namespace for the schema document you import.

   Also, ensure that the namespace prefix is declared for that namespace so that the elements in the imported schema can be referenced using the namespace prefix.

3. The attribute targetNamespace for the xsd:schema element should be the same as the targetNamespace for this WSDL.

Message Definitions

For every operation, you must create a message for sending requests and another message (optionally) for receiving responses, depending on the pattern you selected.

- The message for sending the requests should be the same as the operation followed by ReqMsg.
- The response message should be the same as the operation followed by RespMsg.

PortType Definition

You should define a portType name for each operation, and it should be the same as the service name. You defined the message names specified for input and output

elements in the message definitions section based on the pattern. When services are deployed, Mediator appends *Service* to the port type to form the service name that goes under the service section in the concrete WSDL.

Annotating Service Interface

Sections of the WSDL allow documentation where the details of the sections can be annotated. WSDL authors must annotate the sections thoroughly, and in a manner similar to the way existing EBS WSDLs are annotated. The annotations serve as the source of truth for populating the metadata in the Oracle Enterprise Repository.

13.3.4 How to Check for WS-I Basic Profile Conformance

The WS-I Basic Profile consists of a set of nonproprietary Web services specifications, along with clarifications, refinements, interpretations, and amplifications of those specifications that promote interoperability.

Conformance to the Profile is defined by adherence to the set of requirements for a specific target, within the scope of the Profile.

13.4 Working with Message Routing

This section includes the following topics:

- Section 13.4.1, "Creating Routing Rules"
- Section 13.4.2, "Routing at the EBS"
- Section 13.4.3, "Guidelines for EBS Routing Rules"
- Section 13.4.4, "How to Identify the Target System at EBS"

13.4.1 Creating Routing Rules

The routing rules in the EBS routing service operations are used to decide to which target end point the incoming message should be routed.

Follow these guidelines when creating routing rules:

- Routing rules must first be defined functionally and always with a specific integration topology in mind.
- In most cases, routing logic should be performed in the routing rules of the EBS.

However, all routing rules in the EBS should check for and respect existing target system IDs that are stamped in the header. EBS rules should not assume the target system ID is populated.

- Requester ABCS should not determine target systems or stamp target system IDs in the EBM header.
- For any EBS operation, each possible target application system instance requires a routing rule.

  For example, if two Siebel provider application system instances exist, SEBL_01 and SEBL_02, then each must have a routing rule even though both rules target the same Siebel provider ABCS.

  Alternatively, if functional requirements dictate that only a single instance of the application type can receive the message at run time, then a single rule could be used and an XSLT would be invoked to stamp the ID of the one instance to be used at run time.

When an EBS operation has multiple provider application system instances of the same application type (such as SEBL_01 and SEBL_02), the routing rules for each instance must have an XSLT to stamp the appropriate system instance ID in the EBM header so that the provider ABCS that is shared between the multiple instances can identify which instance to invoke and cross-reference.

- If an EBS operation is a synchronous request-reply pattern or asynchronous request-delayed response pattern, then the routing rules must be mutually exclusive given the actual topology of the Oracle AIA system.
- Routing rules are delivered with Pre-Built Integrations as part of Mediator routing services.

  These rules are designed to work for the delivered topology. If you implement any changes to the delivered topology, such as adding an additional system instance, then you must implement your own complete set of routing rules.

  The standard routing rule clause structure is:

  (cavs_check) and (ruleset_check) and ((target_system_identified_check) or ((target_system_absent_check) and (topology_specific_clauses))

Table 13–1 lists the routing rule clauses and the related XPath expressions.

Table 13–1 Routing Rule Clauses

Clause	XPath expression
`cavs_check) =`	`MessageProcessingInstruction/EnvironmentCode='PRODUCTION' or not(MessageProcessingInstruction/EnvironmentCode/text()))`
`ruleset_check) =`	`TBD`
`target_system_identified_check) =`	`EBMHeader/Target/ApplicationTypeCode='SIEBEL'`
`target_system_absent_check) =`	`not(EBMHeader/Target/ID/text())`
`O2C2 OOTB (topology_specific_clauses) =`	`aia:getSystemType(EBMHeader/Sender/ID)!='SIEBEL'`

Table 13–2 shows some routing rules delivered as part of the Integrated Supply Chain Management Pre-Built Integration.

Table 13–2 Delivered Routing Rules

Target	Siebel provider ABCS
`XPath Filter:`	`(MessageProcessingInstruction/EnvironmentCode='PRODUCTION' or not(MessageProcessingInstruction/EnvironmentCode/text()))and (EBMHeader/Target/ApplicationTypeCode='SIEBEL' or ( not(EBMHeader/Target/ID/text()) and aia:getSystemType(EBMHeader/Sender/ID)!='SIEBEL' ) )`
Transformation:	None
Explanation:	MessageProcessingInstruction/EnvironmentCode='PRODUCTION' or is missing entirely and either Target application type is specified as Siebel, or else no Target is specified and the Sender application type is not Siebel.

Table 13–2 (Cont.) Delivered Routing Rules

Target	Siebel provider ABCS
Target:	Oracle EBusiness provider ABCS
XPath Filter:	(MessageProcessingInstruction/EnvironmentCode='PRODUCTION' or not(MessageProcessingInstruction/EnvironmentCode/text())) and (EBMHeader/Target/ApplicationTypeCode='EBIZ' or (not(EBMHeader/Target/ID/text()) and aia:getSystemType(EBMHeader/Sender/ID)!='EBIZ'))
Transformation:	None
Explanation:	MessageProcessingInstruction/EnvironmentCode='PRODUCTION' or is missing entirely and either Target application type is specified as EBiz, or else no Target is specified and the Sender application type is not EBiz.
Target:	CAVS
XPath Filter:	MessageProcessingInstruction/EnvironmentCode='CAVS'
Transformation:	None
Explanation:	MessageProcessingInstruction/EnvironmentCode='CAVS'

13.4.2 Routing at the EBS

Routing rules are specified for each operation defined on EBS services. The system uses routing rules to determine where to route the incoming EBM, either to an EBF, EBS, ABCS, or the Composite Application Validation System (CAVS). Routing rules are specified as XPath expressions in the filter of the Mediator routing rule.

Routing rules must be mutually exclusive since all the rules are evaluated and messages are routed to an end point based on the rule evaluation.

13.4.3 Guidelines for EBS Routing Rules

At a minimum, each EBS operation should have these rules:

- One routing rule for CAVS enabling.

  This rule should check whether the EBM Header > MessageProcessingInstruction > EnvironmentCode is set to *CAVS*.

- One or more routing rules to connect to the provider ABCS or EBF.

  The filter expression specified in these routing rules must ensure that the message is not a test message.

For each ABCS or EBF, one routing rule exists. The conditions can be one of these:

- Target system ABCS populated in the EBM header

  Example of a filter expression for a SalesOrderEBS routing rule for determining the target system ABCS:

  In this case, the filter has an expression to check whether the target system ABCS in the EBM header was prepopulated and the Override Routing Indicator is set to *False*.

```
/sordebo:QuerySalesOrderEBM/ns5:EBMHeader/ns5:Target/ns5:ID = "SEBL78_01" and
```

```
/sordebo: Query SalesOrderEBM/ns5:EBMHeader/ns5:Target/ns5: Override
RoutingIndicator = "false"
```

- Content-based routing

  In this case, the content of the EBM is evaluated to determine the target system ABCS. The filter expression should ensure that the target system information was not prepopulated in the EBM header.

  Example expression:

```
starts-with(/sordebo: CreateSalesOrderEBM
/sordebo: DataArea/sordebo: CreateSalesOrder/sordebo: SalesOrderLine
/sordebo:SalesOrderLineSchedule/ns5:ShipToPartyReference
/ns5:LocationReference
/ns5:Address/ns5:CountrySubDivisionCode,'9') and
/sordebo:CreateSalesOrderEBM/ns5:EBMHeader /ns5:Target/ns5:ID = ""
```

- Parallel Routing

  EBS should use parallel routing when you want each target service to invoke in its own transaction and retry the target service.

  For more information, see Chapter 14, "Designing Application Business Connector Services."

- Enable error handling and logging

  EBS should handle errors to allow clients or administrators to resubmit or retrigger processes through a central error handler.

  For more information, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging".

13.4.4 How to Identify the Target System at EBS

The EBS can route the request from the Requester ABCS, an EBF, or another EBS to one of the many provider ABCS available. The target system must be identified only for **Create** operations. For all other operations, the Xref function **lookupPopulatedColumns** is used to identify the systems in which the data synchronization of entity has been done. A combination of steps listed below leads to the correct ABCS.

To identify the target system at EBS:

1. Using content-based routing.

   The routing rule in the EBS is based on the content of the message and is used to decide the target ABCS. This is used only for the **Create** operation. This is the case when the target system information in the EBM header is empty and has not been set as yet.

   After the target system is determined, it is set in the EBM header. This information is used for all subsequent services - like other EBS or ABCS.

2. Using the target system information in the EBM header.

   If the target system information is set in the EBM, this is used for routing to the correct ABCS.

3. From the XREF for operations other than **Create**.

   For all operations other than Create, the target system has been determined and the cross-reference IDs have been set. In this case, use the XREF function

lookupPopulatedColumns to identify the systems in which the data synchronization of the entity has been done.

13.5 Building EBS Using Oracle Mediator

You can use the Oracle Mediator component to build a component in an EBS composite.

Although Oracle AIA allows any technology to be used for developing an EBS service, use Mediator in most situations.

13.5.1 How to Develop the Oracle Mediator Service

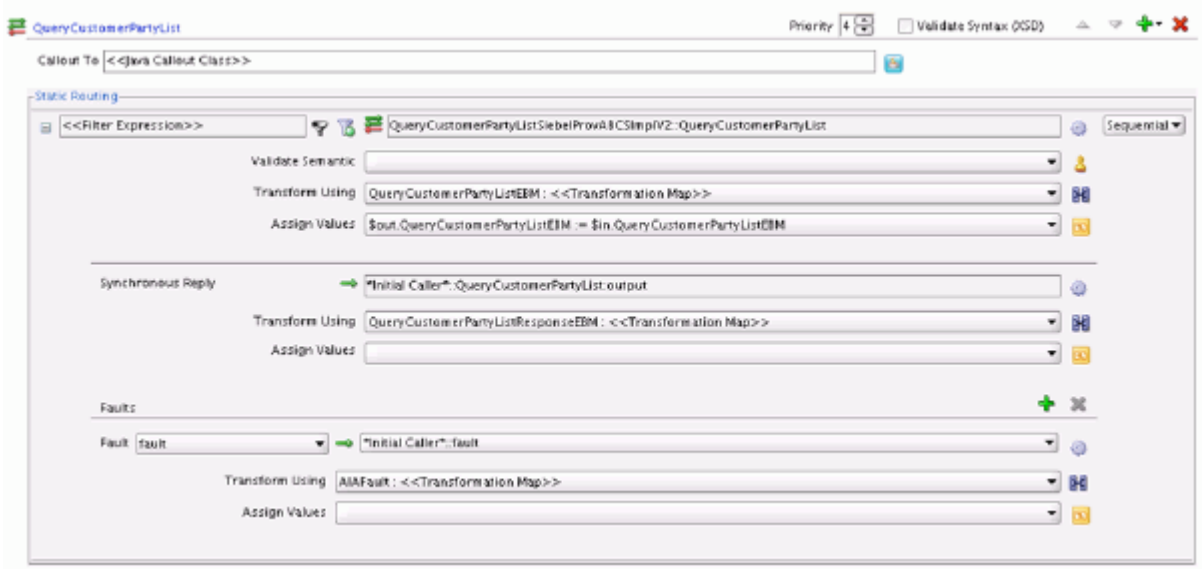
To develop the Oracle Mediator Service:

1. In JDeveloper, create an SOA composite project.
2. Open the **composite.xml** in Design mode.
3. Add a Mediator component in the components swim lane.
4. Add a Web service as external reference service in the references swim lane.

   This reference service could represent a provider ABCS.

   You should use a concrete WSDL of the provider ABCS. That is, you should pre-deploy the service that is being referred to as an external reference.
5. Wire the Mediator component to the external reference component created in step 4.
6. Open the Mediator component and configure routing rules.
7. Create an assign by copying the input variable to the output variable as shown in Figure 13–2.

Figure 13–2 Creating an Assign



13.6 Implementing the Fire-and-Forget Message Exchange Pattern

To implement both asynchronous MEPs (fire-and-forget and request-delayed response), you must create EBS WSDLs and then create one or two Mediator routing services, depending upon the MEP. Next, implement the requester and provider services, adhering to the guidelines laid out for the respective MEPs.

The requesting service can be a requester ABCS (BPEL process), EBF (BPEL process), or a participating application.

The providing service can be a provider ABCS (BPEL Process), EBF (BPEL process), or a participating application.

This section includes the following topics:

- Section 13.6.1, "How to Implement Fire-and-Forget Pattern with EBS One-Way Calls"
- Section 13.6.2, "Creating EBS WSDLs"
- Section 13.6.3, "Creating Mediator Routing Services for Asynchronous Fire-and-Forget Patterns with a One-Way Call EBS"
- Section 13.6.4, "Asynchronous Fire-and-Forget MEP Error Handling Using Compensatory Operations"
- Section 13.6.5, "How to Invoke the Compensate Operation of EBS"
- Section 13.6.6, "How to Enable Routing Rules in Compensate Operation Routing Service"

13.6.1 How to Implement Fire-and-Forget Pattern with EBS One-Way Calls

The initiator for a fire-and-forget pattern is a requesting service not waiting for or expecting a response. The requesting service can be a participating application, a requester ABCS Impl, or an EBF. In each of these cases, the request payload must be an EBM request.

For more information about enabling the ABCS (both requester and provider), see Chapter 14, "Designing Application Business Connector Services", Chapter 15, "Constructing the ABCS", and Chapter 16, "Completing ABCS Development".

For more information about enabling the EBF, see Chapter 18, "Designing and Constructing Enterprise Business Flows".

To implement fire-and-forget pattern with EBS one-way calls:

1. Create EBS WSDLs.
2. Create a Mediator routing service for asynchronous fire-and-forget patterns with one-way call EBS.
3. Route the request from the requesting service to correct providing service in the routing service of the one-way call operation of the request EBS.
4. Implement error handling for logging and notification based on fault policies.

Note: These steps are in addition to the regular steps required for the requesting service and the providing service.

13.6.2 Creating EBS WSDLs

For the entity EBS, use the WSDLs from the Enterprise Service Library of the Foundation Pack.

For the process EBS, use the TemplateEBS.wsdl available in the Foundation Pack and create a Process EBS wsdl.

- Service operations supporting a synchronous request-response MEP must be defined in one port type and the operation must have input, output, and fault message defined.
- Service operations supporting a fire-and-forget MEP must be defined in one port type and the operation must have an input message.
- Service operations supporting asynchronous request-response pattern must have two operations, one operation for sending the request message and another operation for processing the response message.

  Each of these two operations must have an input message alone. You should have two different portTypes, one for each operation. The service operation for processing the response message must reside in a portType having *Response* as the suffix.
- The EBS WSDLs must have two portTypes:
  - PortType for all operations used for modeling synchronous request, response operations and request-only operations. **The name must not specify** ***Request*****.**
  - PortType for asynchronous response operations. **The name must specify** ***Response*****.**
- Two Mediator routing services must be created for each of the portTypes.

13.6.3 Creating Mediator Routing Services for Asynchronous Fire-and-Forget Patterns with a One-Way Call EBS

To create Mediator routing services for asynchronous fire-and-forget patterns with a one-way call EBS:

1. Create Mediator projects.
2. Create routing services.
3. Create routing rules.
4. Implement error handling.

13.6.3.1 How to Create Mediator Projects for the Asynchronous Fire-and-Forget MEP

To create Mediator projects for the asynchronous fire-and-forget MEP:

1. Create two Mediator projects, one for each of the portTypes in the EBS WSDL.

   If all of the service operations for an EBS have either a synchronous request-response or fire-and-forget pattern, then all of these operations must reside in only one portType so there would be only one Mediator routing service.

   If the EBS has at least one asynchronous request-response operation, then there should be two port types - two Mediator Routing Services and two Mediator projects (one for each of the routing services).

2. Follow the naming conventions detailed in Appendix: Oracle AIA Naming Standards.

   Examples of typical names for the Mediator projects:

   - **CustomerPartyEBSV2** (This has a routing service with all operations for synchronous request-response and request-only.)
   - **CustomerPartyEBSResponseV2** (This has a routing service with all operations for asynchronous request-response.)

13.6.3.2 How to Create Routing Services for Asynchronous Fire-and-Forget MEP

To create routing services for asynchronous fire-and-forget MEP:

1. Put the EBS WSDL in the Mediator project folder.
2. Create a routing service and name according to the naming convention detailed in Appendix: Oracle AIA Naming Standards**.**
3. Select the WSDL.

   The WSDL must be parsed and the portType name filled in the portType field of the routing service.
4. Select the portType matching with the routing service. Save the routing service.

   The routing service created for a portType must have all the operations specified in that portType in the EBS WSDL.

13.6.3.3 How to Create Routing Rules for Asynchronous Fire-and-Forget MEP

The routing rules for the request EBS are the same as those for the synchronous request-response section.

For more information, see Section 13.7.3, "How to Create Routing Services for the Synchronous Request-Response MEP".

13.6.3.4 How to Implement Error Handling for Asynchronous Fire-and-Forget MEP

For more information, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging".

13.6.4 Asynchronous Fire-and-Forget MEP Error Handling Using Compensatory Operations

To offset the effects of errors in the provider services, operations can be added to the EBS to trigger compensation in the requesting services for one-way calls. This can be achieved by having compensatory operations in the EBS.

Compensatory operations, modeled as one-way calls are separate operations. For each request-only operation in the request portType, there must be an operation for triggering compensation.

For example: *CompensateCreateCustomer*, *CompensateCreateOrder*.

Compensatory operations are invoked in cases where a business exception is likely to result in a irrecoverable error. The conventional retry and resubmit is not possible and the correction is required to be made in the requesting service.

In this situation, you must implement suitable compensatory taking advantage of the participating applications compensatory action web services or APIs.

13.6.5 How to Invoke the Compensate Operation of EBS

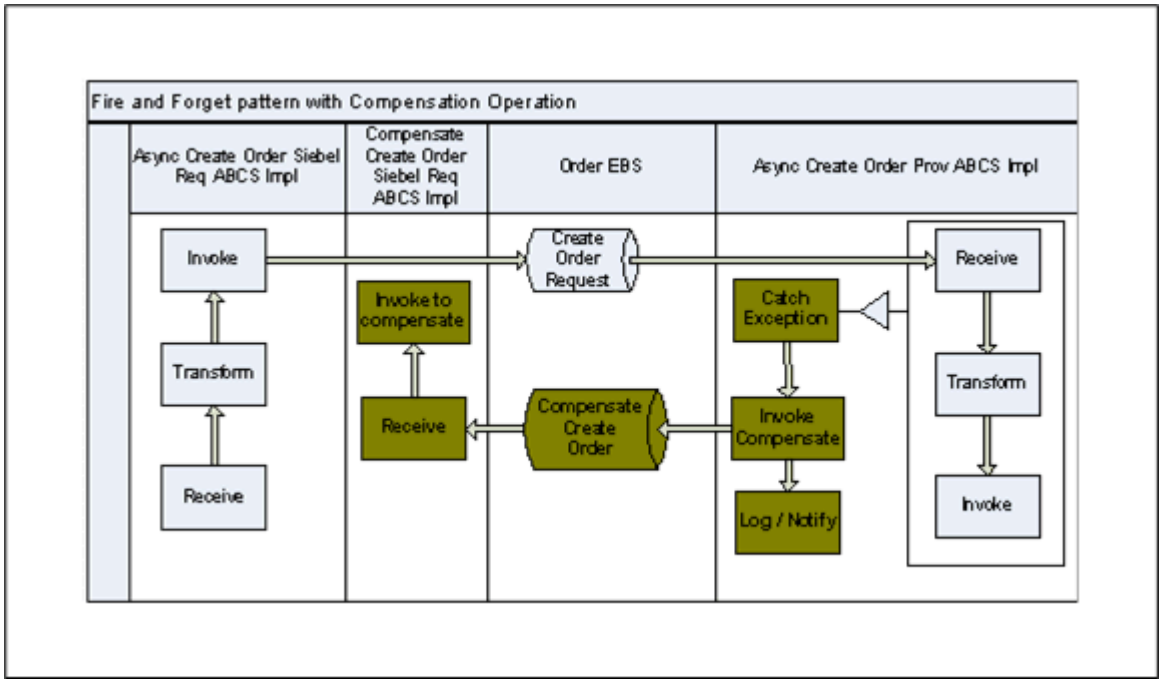
In error handling, you must ensure that compensatory actions are taken for some errors so the compensate operation of the EBS is invoked from the providing service. The compensate operation of the EBS routes to the correct compensating service.

To invoke the compensate operation of the EBS:

1. For an error in the providing service, raise an exception and catch it in the catch block.
2. In the catch block, construct the request EBM along with the fault component in the EBM header.
3. Create a transform step and select the input variable representing the request EBM and the compensate variable, also representing the request EBM.
4. When an exception is generated, put the exception details in a variable and pass that as input to the compensation XSLT.
5. Map the following to the compensate variable:
   - Standard EBM header content from the request EBM
   - Data area from the request EBM
   - Fault message
6. Set the **InvokeCompensate** step to invoke the corresponding compensate operation in the request EBS routing service.
7. Route the compensate request to a suitable compensating service.

   Figure 13–3 illustrates the fire-and-forget pattern with the compensation operation.

Figure 13–3 Fire-and-Forget Pattern with Compensation Operation



13.6.6 How to Enable Routing Rules in Compensate Operation Routing Service

There must be two routing rules.

To enable routing rules in compensate operation routing service:

1. Routing rule for the compensate operation of EBS.

   The information populated in the `<EBO Name>ResponseEBM\corecom:EBMHeader\Sender\ WSAddress/WSAddress/wsa:FaultTo/wsa:ServiceName` in the requesting service is used to route the request for compensation to the correct compensating service in the compensate operation of the EBS.

   Put this routing rule in the compensate operation of the EBS:

   ```
   <EBO Name>ResponseEBM\corecom:EBMHeader\Sender\ WSAddress/
   WSAddress/wsa:FaultTo/wsa:ServiceName = <Compensating Service Name>
   ```

2. Routing rule for CAVS.

   If the test case created in CAVS is of type asynchronous delayed-response, then the response message can come to the CAVS endpoint and be correlated back to make the test pass/fail. For this to happen, an explicit invoke to the CAVS system endpoint must exist:
   http://host:port/AIAValidationSystemServlet/asyncresponserecipient

13.7 Implementing the Synchronous Request-Response Message Exchange Pattern

The initiator for a synchronous request-reply pattern is a requesting service waiting for and expecting a response. The requesting service can be a participating application, requester ABCS Impl, or an EBF. In each of these cases, the request payload would be an EBM request and the response payload would be an EBM response.

This section includes the following topics:

- Section 13.7.1, "How to Implement Synchronous Request-Reply Message Exchange Patterns in EBS"
- Section 13.7.2, "How to Create Mediator Projects for the Synchronous Request-Response MEP"
- Section 13.7.3, "How to Create Routing Services for the Synchronous Request-Response MEP"
- Section 13.7.4, "How to Implement Error Handling for the Synchronous Request-Response MEP"

For more information about enabling the ABCS (both requester and provider) see Chapter 14, "Designing Application Business Connector Services", Chapter 15, "Constructing the ABCS", and Chapter 16, "Completing ABCS Development".

For more information about enabling the EBF, see Chapter 18, "Designing and Constructing Enterprise Business Flows".

13.7.1 How to Implement Synchronous Request-Reply Message Exchange Patterns in EBS

To implement synchronous request-reply MEP in EBS:

1. Create Mediator projects with routing services.
2. Create routing rules to route the request from the requesting service to the correct providing service in the routing service of the EBS.
3. Implement error handling for logging and notification based on fault policies.

13.7.2 How to Create Mediator Projects for the Synchronous Request-Response MEP

To create Mediator projects for the synchronous request-response MEP:

Follow these guidelines when creating Mediator projects:

1. Create two Mediator projects, one for each of the portTypes in the EBS WSDL.

   If all of the services operations for an EBS have either synchronous request-response or fire-and-forget pattern, then all of these operations must reside in only one portType, so only one Mediator Routing Service should exist.

   If the EBS has at least one asynchronous request-response operation, then two portTypes should exist, two Mediator routing services and two Mediator projects (one for each routing service).

2. Follow the naming convention detailed in Appendix: Oracle AIA Naming Standards.

   Examples of typical names for the Mediator projects are:

   - *CustomerPartyEBSV2* (This example has a routing service with all operations for synchronous request-response and request-only.)
   - *CustomerPartyEBSResponseV2* (This example has a routing service with all operations for asynchronous request-response.)

13.7.3 How to Create Routing Services for the Synchronous Request-Response MEP

To create routing services for the synchronous request-response MEP:

1. In JDeveloper, put the EBS WSDL in the Mediator project folder.
2. Create a routing service and name according to the naming convention detailed in Chapter 31, "Oracle AIA Naming Standards for AIA Development".
3. Select the WSDL. The WSDL must be parsed and the portType name filled in the portType field of the routing service.
4. Select the portType matching with the routing service. Save the routing service.

   The routing service created for a portType must have all the operations specified in that portType in the EBS WSDL.

13.7.4 How to Implement Error Handling for the Synchronous Request-Response MEP

For more information, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging".

13.8 Implementing the Asynchronous Request-Delayed Response Message Exchange Pattern

The initiator of the request-delayed response pattern is a requesting service that invokes the request EBS and waits for a response. The requesting service can be a participating application, requester ABCS Impl, or an EBF. In each of these cases, the request payload is an EBM request and the response payload is an EBM response.

For an error in the providing service, a response message with error information is constructed and returned to the requesting service for action.

This section includes the following topics:

- Section 13.8.1, "How to Implement the Request-Delayed Response Pattern with the Two One-Way Calls of the EBS"
- Section 13.8.2, "Creating Mediator Routing Services for Asynchronous Request-Delayed Response Patterns with Two One-Way Call EBS"
- Section 13.8.3, "Asynchronous Request-Delayed Response MEP Error Handling"

For more information about enabling the ABCS (both requester and provider) see Chapter 14, "Designing Application Business Connector Services", Chapter 15, "Constructing the ABCS", and Chapter 16, "Completing ABCS Development".

For more information about enabling the EBF, see Chapter 18, "Designing and Constructing Enterprise Business Flows".

13.8.1 How to Implement the Request-Delayed Response Pattern with the Two One-Way Calls of the EBS

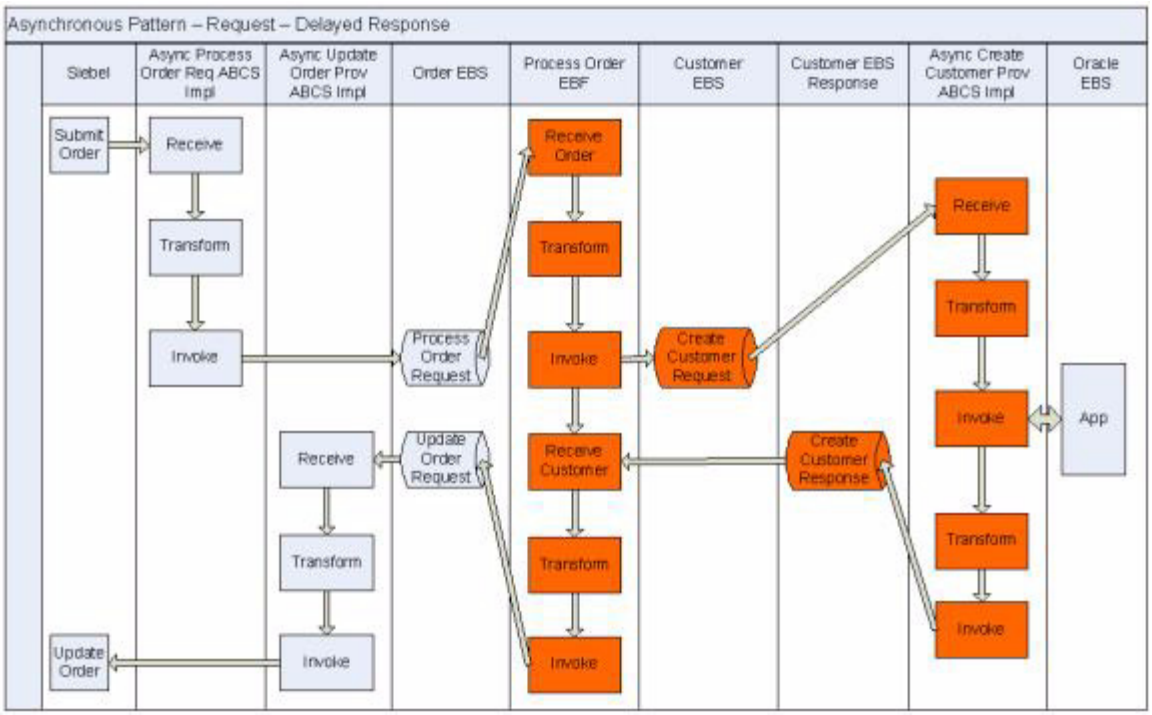
To implement the request-delayed response pattern with the two one-way calls of the EBS:

> **Note:** Perform these steps in addition to the regular steps required for the requesting service and the providing service.

1. Create EBS WSDLs.
2. Create Mediator routing services for asynchronous request-delayed response patterns with two one-way call EBSs.
3. Route the request from the requesting service to the correct providing service in the routing service of the one-way call operation of the request EBS.
4. Implement error handling for logging and notification based on fault policies.
5. Route the response message in the EBS response to the correct requesting service.

   Figure 13–4 illustrates the request-delayed response pattern:

Figure 13–4 Request-Delayed Response Pattern



13.8.1.1 How to Create the EBS WSDLs for the Request-Delayed Response MEP

To create the EBS WSDLs for the request-delayed response MEP:

1. For the entity EBS, use the WSDLs from the Enterprise Service Library of the Foundation Pack.
2. For the process EBS, use the TemplateEBS.wsdl available in the Foundation Pack and create a Process EBS WSDL.
3. The EBS WSDLs must have two portTypes:
   - PortType for all operations used for modeling synchronous request-response operations and request-only operations.

     The name must not specify the *Request*. Service operations supporting the synchronous request-response MEP must be defined in one portType, and the operation must have input, output, and fault message defined.
   - PortType for asynchronous response operations.

     The name must specify *Response*. Service operations supporting asynchronous request-response pattern must have two operations, one operation for sending the request message and another for processing the response message.
4. Two Mediator routing services must be created for each of the portTypes.

13.8.2 Creating Mediator Routing Services for Asynchronous Request-Delayed Response Patterns with Two One-Way Call EBS

To create Mediator routing services:

1. Create Mediator projects.

2. Create routing services.
3. Create routing rules.

13.8.2.1 How to Create Mediator Projects for the Request-Delayed Response MEP

To create Mediator projects for the request-delayed response MEP:

1. Create two Mediator projects, one for each of the portTypes in the EBS WSDL.

   If all of the services operations for an EBS have either synchronous request-response or fire-and-forget pattern, then all of these operations must reside in only one portType, so only one Mediator routing service is used.

   If the EBS has at least one asynchronous request-response operation, then two portTypes must exist, two Mediator Routing Services and two Mediator Projects (one for each routing service).

2. Follow the naming convention detailed in Chapter 31, "Oracle AIA Naming Standards for AIA Development".

   Examples of typical names for the Mediator projects are:

   - *CustomerPartyEBSV2* (This example has a routing service with all operations for synchronous request-response and request-only.)
   - *CustomerPartyEBSResponseV2* (This example has a routing service with all operations for asynchronous request-response.)

13.8.2.2 How to Create Routing Services

To create routing services for the request-delayed response MEP:

1. Put the EBS WSDL created into the Mediator project folder.
2. Create a routing service and name according to the naming convention detailed in Chapter 31, "Oracle AIA Naming Standards for AIA Development".
3. Select the WSDL.

   The WSDL must be parsed and the portType name filled in the portType field of the routing service.

4. Select the portType matching with the routing service. Save the routing service.

   The routing service created for a portType must have all the operations, including compensate operations, specified in that portType in the EBS WSDL.

> **Note:** These guidelines are in addition to the implementation of asynchronous message exchanging patterns.

13.8.2.3 How to Create Routing Rules

For the asynchronous request-delayed response EBS, routing rules must be created for both request and response.

Routing Rules for Request EBS

The routing rules for request EBS are the same as those explained in the synchronous request-response section.

For more information, see Section 13.7.3, "How to Create Routing Services for the Synchronous Request-Response MEP".

Routing Rules for Response EBS

There must be two routing rules.

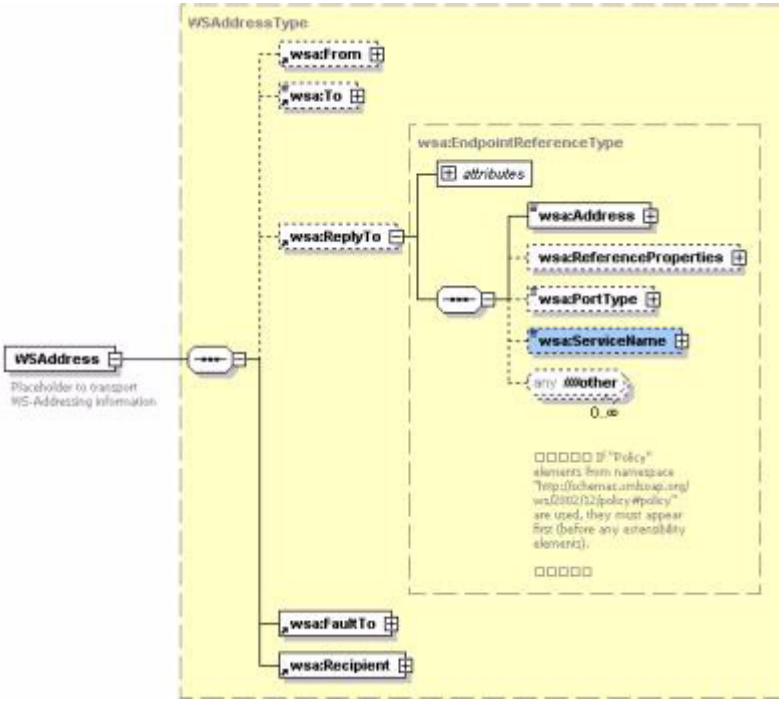
To create routing rules for the response EBS:

1. Routing to the correct requesting service.

   When multiple requesting services from multiple participating applications are invoking a request EBS and are waiting for a delayed response, then you must route the response to the correct requesting service.

   As illustrated in Figure 13–5, set the `EBMHeader/Sender/WSAddress/wsa:ReplyTo/wsa:ServiceName` to the name of the requesting service name in the requesting service-Application Business Message (ABM) to EBM transformation-before invoking the request EBS.

Figure 13–5 Structure of the WSAddress Type Element



In the providing service, this information is transferred from the request EBM to the response EBM. This information is used in the response EBS by putting a routing rule in the filter as:

```
<EBO Name>ResponseEBM\corecom:EBMHeader\Sender\
WSAddress/wsa:ReplyTo/wsa:ServiceName = <Requesting Service Name>
```

The target endpoint for the evaluation of this rule should be set to the requesting service.

For every requesting service of the request EBS that is waiting for a response EBS to send back a response, specify a routing rule as shown above.

2. CAVS routing rule.

The CAVS routing rules are the same as that explained in the Sync Request-Response section.

For more information about enabling the ABCS (both requester and provider) see Chapter 14, "Designing Application Business Connector Services", Chapter 15, "Constructing the ABCS", and Chapter 16, "Completing ABCS Development".

Error Handling Implementation

For more information, see Section 26.5, "Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery."

13.8.3 Asynchronous Request-Delayed Response MEP Error Handling

In the asynchronous request-delayed response MEP, the requesting service is in a suspended mode, waiting for a response. If there is an error in the providing service, the response to the requesting service includes the details of the error.

To handle errors in the asynchronous request-delayed response MEP:

1. In case of an error in the providing service, raise an exception and catch it in the catch block.
2. In the catch block, construct the response EBM along with fault component in the EBM header.
3. Create a transform step and select the input variable representing the request EBM and the output variable representing the response EBM.
4. Pass the fault message generated from the exception as a variable into the 'input variable to fault variable' XSLT.
5. Map to the output variable:
   - Standard EBM header content from the request EBM including the correlation information
   - Fault message
6. Set the **Invoke** step to invoke the response operation in the response EBS routing service.
7. Route the response from the providing service to the correct requesting service.

For more information about enabling the ABCS (both requester and provider), see Chapter 14, "Designing Application Business Connector Services", Chapter 15, "Constructing the ABCS", and Chapter 16, "Completing ABCS Development".

14

Designing Application Business Connector Services

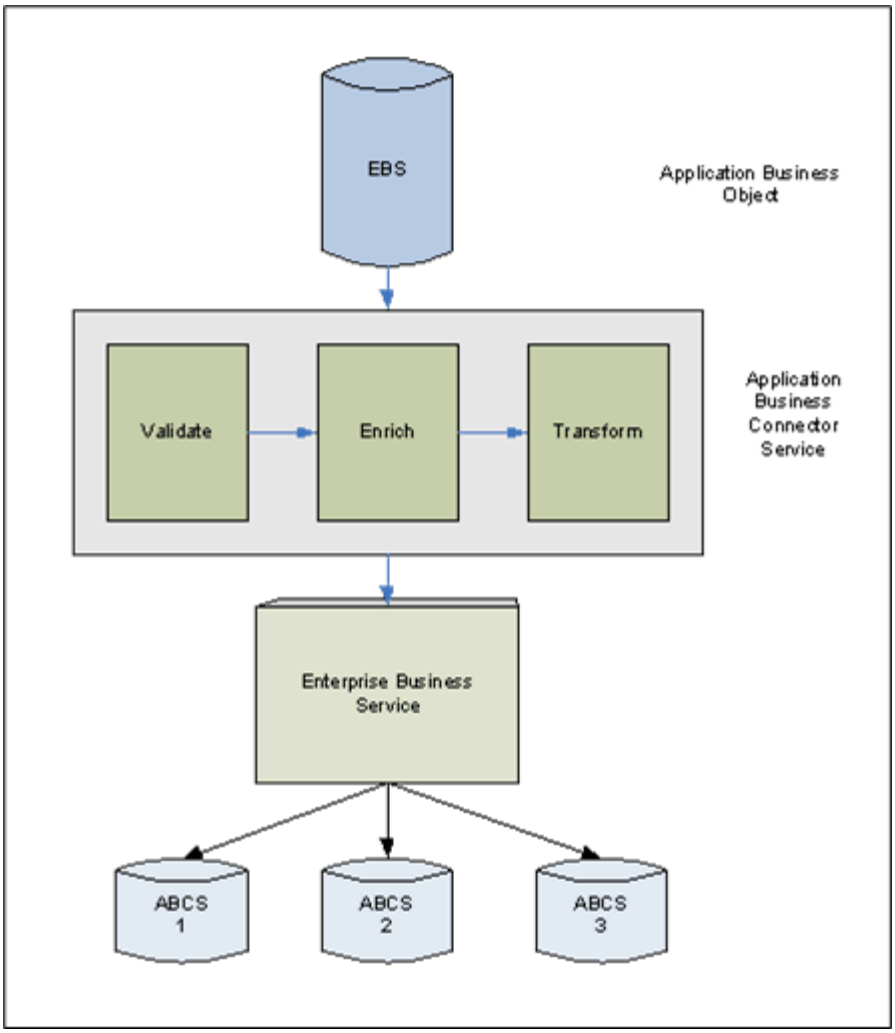
This chapter provides an overview of Application Business Connector Services (ABCS) describes how to define the ABCS contract, identify the MEP and discusses technology options for outbound interaction with the application and how to use BPEL for building ABCS.

This chapter includes the following sections:

14.1 Introduction to ABCS

The role of the ABCS is to expose the business functions provided by the participating application in a representation that is agreeable to Enterprise Business Services (EBSs). It can also play another role in which it serves as a glue to allow the participating application to invoke the EBSs. Figure 14–1 illustrates a scenario in which the E-Business Suite application is using an ABCS to invoke an EBS. In this scenario, the ABCS is playing a requester role. The same diagram also illustrates how the EBS invokes one of three ABCS to perform a specific task. Here, the three ABCS are playing provider roles.

Figure 14–1 ABCS in Oracle AIA Architecture



The ABCS enables participating applications to become service providers and service consumers. It also enables applications having nonstandard connectivity to expose their functionality as web services.

For more information about ABCS, see "Understanding Application Business Connector Services" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*..

14.1.1 ABCS Types

The two types of ABCSs are:

- Requester ABCS
- Provider ABCS

14.1.1.1 Requester ABCS

The requester ABCS is provided by the requesting application to interface with an EBS for performing a single task. The service interfaces between Requesting/Source application and EBS. The role of the requester ABCS is to enable the participating application to invoke the EBS either to access data or to perform a transactional task.

The requester ABCS is a service provided by the requesting application to invoke the EBS. It receives the Application Business Message (ABM) as payload and optionally returns the ABM as the response.

The ABM can be enriched by having additional conversations with the requester application. The ABM is transformed into an Enterprise Business Message (EBM) using XSL. Responses received from the EBS (EBM) are transformed into the ABM.

Non-SOAP payloads are handled by having transport adapters in between requester application and requester ABCS. Requester applications must pass system information for the ABCS to have subsequent conversations.

14.1.1.2 Provider ABCS

The provider ABCS is supplied by the provider application to interface with an EBS for performing a single task. The service interfaces between the EBS and provider or target application.

The provider ABCS:

- Is a service supplied by the provider application to interface with the provider application
- Receives EBM as payload
- Optionally returns the EBM as the response

The EBM is transformed into an ABM using XSL. The ABM can be enriched by having additional conversations with the provider application. Responses received from the application (ABM) are transformed into the EBM.

Non-SOAP connectivity with the provider application is handled by having transport adapters in between the provider ABCS and the provider application.

For more information about ABCS, see "Understanding Application Business Connector Services" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*.

14.1.2 Designing ABCS - Key Tasks

An ABCS architect or developer must perform the following tasks as part of the design activity:

- Understand the ABCS role
- Identify the business events triggering the invocation of ABCS
- Identify the interactions between application and the ABCS and vice versa
- Identify the appropriate technologies to be used to facilitate the interaction
- Define the ABCS contract

The following sections discuss these tasks:

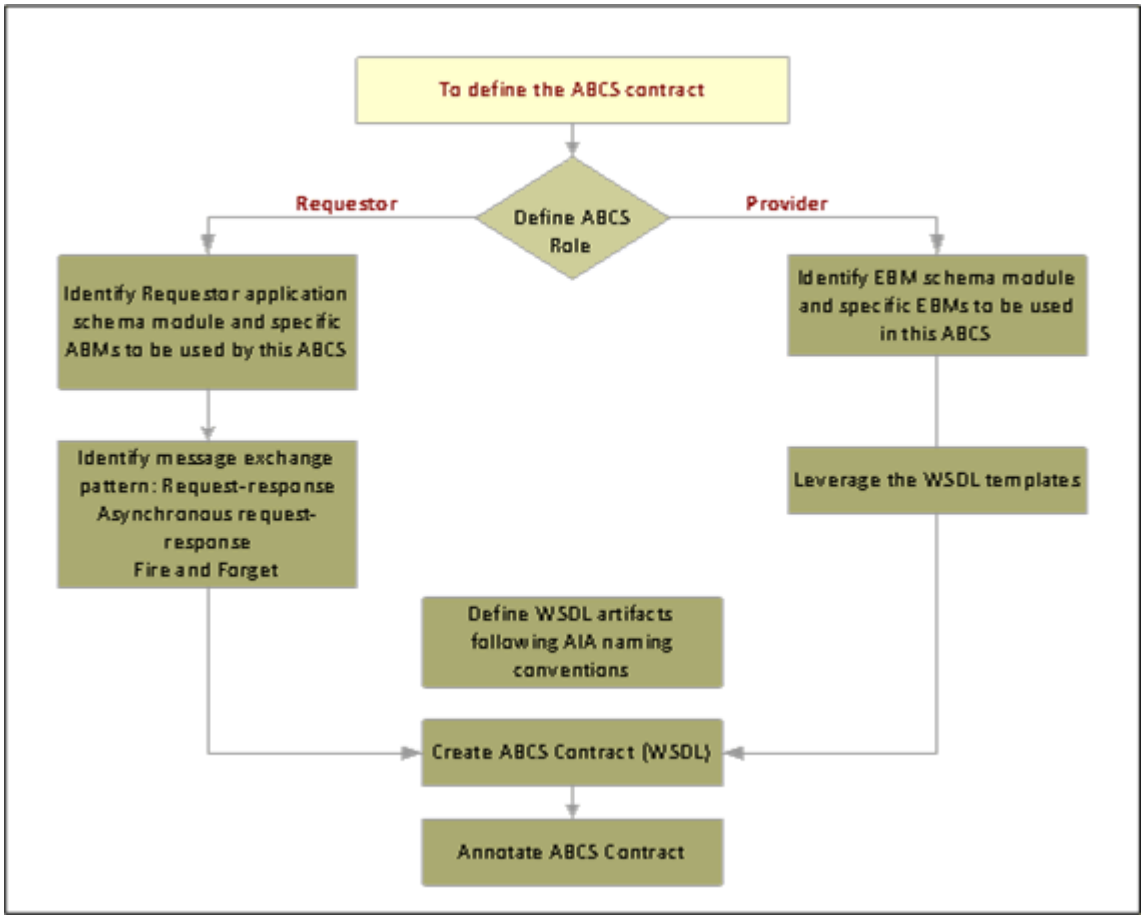
- Section 14.2, "Defining the ABCS Contract"
- Section 14.3, "Identifying the MEP"
- Section 14.4, "Technology Options"

14.2 Defining the ABCS Contract

The ABCS contract (WSDL) specifies how a caller interacts with its Business Connector Service. The caller could either be a participating application or another service such as EBS.

Figure 14–2 illustrates the decision flow for defining the contract (WSDL) for the ABCS.

Figure 14–2 Decision Flow for Defining the ABCS Contract



14.2.1 Defining the Role of the ABCS

This section discusses design decisions that you must make to enable the ABCS to participate in either a requester or provider role.

14.2.1.1 Designing an ABCS to Participate in a Requester Role

To determine how to enable the ABCS to participate in a requester role:

1. Determine how the ABCS is invoked or triggered. This analysis tells you how the ABCS is invoked by the application. Determine:
   a. Whether the application can make a SOAP-based invocation of ABCS.
   b. Whether the application service can invoke the ABCS using JCA adapter.

c. Whether the ABCS must subscribe to messages published by the application due to occurrence of business events.

2. Determine what interaction pattern is needed.

   The following questions help the architect identify the optimal message exchange pattern.

   a. What business events cause the invocation of requester ABCS?

   b. Is the application expecting a response from the requester ABCS?

   c. Should the application wait until the response is received for performing the next task?

   d. Should the application wait until the request is fully processed by the provider application?

   For more information about choosing a MEP, see Section 14.3.2, "Choosing the Appropriate MEP."

3. Determine what Enterprise Business Services or operations must be invoked.

4. Does the message sent by the requester application have all the information necessary for requester ABCS to construct the EBM?

   a. If not, does it have to reach out to applications to get the additional information?

   b. If yes, what technologies are available? What application interfaces are to be used to get the additional information?

   For more information, see Section 14.4, "Technology Options".

5. Can the application possibly publish a message with incomplete information?

   a. Should the ABCS wait until the message with complete information is published?

      In such a scenario, an aggregator should collect and store individual messages until a complete set of related messages has been received

   b. If yes, what is the application service and how does it communicate with it?

14.2.1.2 Designing an ABCS to Participate in a Provider Role

To determine how to enable the ABCS to participate in a provider role:

1. Decide whether the application can directly consume the information sent by EBS.

2. Determine how the ABCS interacts with provider applications.

   See Chapter 23, "Establishing Resource Connectivity."

3. Define the tasks that must be done by this ABCS.

4. Determine what message exchange pattern (MEP) is used to communicate with the application.

5. Determine whether data validation must be done on the EBM sent by the EBS before sending the request to the provider application.

6. Decide whether you need guaranteed delivery of messages.

7. Determine whether it is a single message containing data for multiple instances.

8. Decide whether the entire message should be processed as a unit of work.

9. Decide what must be done when a subset of instances present in that message encounter failure.

14.2.2 Constructing ABM Schemas

The ABM schemas are created by the participating applications. Consider the following points when defining the ABMs:

- Construct messages that are durable and long-lasting because frequent changing of the ABM definition puts duress on ABCS development.
- Ensure reusability of business components across multiple ABMs to promote reusability of transformation maps.
- Design ABMs to pass the following system data:
  - System Instance - connection information that helps identify the system ID registered in the system registry
  - Locale - language code in which the request is sent
  - Sender information - contextual details pertaining to the origination of the message
- Design ABMs that are extensible

14.2.3 Analyzing the Participating Application Integration Capabilities

To analyze the participating application integration capabilities:

After identifying the participating applications, you must analyze the integration capabilities of each participating application and how each application exposes its data.

1. Work with the application providers (developers) to decide the best way to interact with the application to attain the needed functionality.

   The interactions can be inbound or outbound. The interaction mechanism can be one of these:

   - Web Service using SOAP/HTTP
   - Events/Queues
   - JCA Adapter
   - Database Adapter

   For more information about various types of connectivity to different resources, see Chapter 23, "Establishing Resource Connectivity."

2. Obtain relevant details from the applications in situations in which the interactions between the participating applications and the ABCS happen through a message queue adapter, database adapter, or JCA adapter.

   These details are used in configuring the adapters. The configuration results in the WSDL creation. Care must be taken to ensure that the environment-specific details are configured in relevant resource files on the server and not directly in the WSDLs.

3. Identify the available functionality within the participating applications and the mapping between EBMs and application business objects (map operations).

   A one-to-one, one-to-many, or many-to-one mapping between the two may exist.

4. Tabulate the mapping between the Enterprise Business Object (EBO) attributes/elements and the application objects attributes/elements in a spreadsheet.

   Remember these points when creating the transformation maps:

5. For an ABM being mapped to an EBM, ensure that an attempt is made to populate all of the data elements present in the ABM to the relevant elements in the EBM.

   In situations in which the ABM does not have all of the content to supply all of the data elements in the EBM, you must work with the respective requester application teams to identify the application services that could be used to enter the content in the EBM.

   For example, a Siebel application invoking the CreateOrder ABCs might supply only the Siebel Customer ID in the ABM that is being passed as the payload. However, the EBM to be passed as a payload to the CreateOrder EBS operation expects all the information regarding the Customer. So the CreateOrder ABCS is responsible for invoking a Siebel service to retrieve complete Customer details from the Siebel application and populate the EBM. Although the integration platform provides support for this pattern, we highly recommend that you work with the participating application to ensure that the payload being passed to the ABCS is as close to the shape of the EBM as possible to minimize the chattiness.

14.3 Identifying the MEP

The possible interaction types between an ABCS and its client are:

- Synchronous request-response
- Asynchronous request only
- Asynchronous request-delayed response

MEPs vary in functionality and usage depending on the role of the ABCS.

For more information about MEPs, see "Understanding Interaction Patterns" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*.

14.3.1 Introduction to MEPs

Synchronous Request-Response Pattern

- A requester sends a request to a provider, who processes the request and sends back a response.
- The requester is in a suspended mode until the response is received from the provider.
- Results are in real-time response or error feedback.

Asynchronous Request Only

- The message exchange is one-way, with a requester sending a request to a provider. No response comes from provider to requester.
- The requester is free, after request submission, to resume normal execution.

Asynchronous Request - Delayed Response

- This involves two one-way invocations.

- First, a one-way message is sent from requester to provider. At a later time, another one-way message is returned from the provider to the original requester.
- Two distinct one-way messages are correlated.

For more information about MEPs, see "Understanding Interaction Patterns" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*.

14.3.2 Choosing the Appropriate MEP

- Identify how the participating applications interact with the ABCS for inbound and outbound interactions.
- Identify whether the requester application must be in a call-waiting state until it has received the response from the provider application.

  Waiting until the response is received from the provider application forces the requester application to invoke the ABCS in a synchronous manner. For example, a CRM application submitting a request to retrieve account details from a billing application would fit this scenario. In this case, the requester cannot perform any task until the response is received.

  However, sending a request from the CRM application to create a customer in a billing application can be done in an asynchronous manner. After publishing an outbound request, the CRM application can move to the next activity without having to wait until the provider application has successfully created a customer. In this case, the CRM application invokes this outbound request in an asynchronous manner.

Applications can leverage different kinds of integration technologies for synchronous versus asynchronous invocations. You must select the most appropriate technology based on the situation.

For more information about how AIA services interact with applications, see Chapter 23, "Establishing Resource Connectivity."

In a single flow, even though the requester ABCS MEP is influenced by the MEP for the EBS operation to be invoked, the requester application's interaction capabilities influence the MEP for requester ABCS. For provider ABCS, the MEP is predominantly influenced by the MEP of the EBS operation.

14.3.2.1 When to Use the Synchronous Request-Response MEP

Most of the cases involving Query and Validate need this pattern because the participating application that initiated the request is waiting for the response. For all the other cases, the suitability of this pattern is determined by the demands of the response time and possibility of meeting those requirements based on the amount of processing to be done.

The MEP between requester ABCS and EBS is synchronous. The MEP between provider ABCS and participating application servicing the request can be either synchronous or asynchronous.

For more information about Query, see Section 15.8.7, "Query".

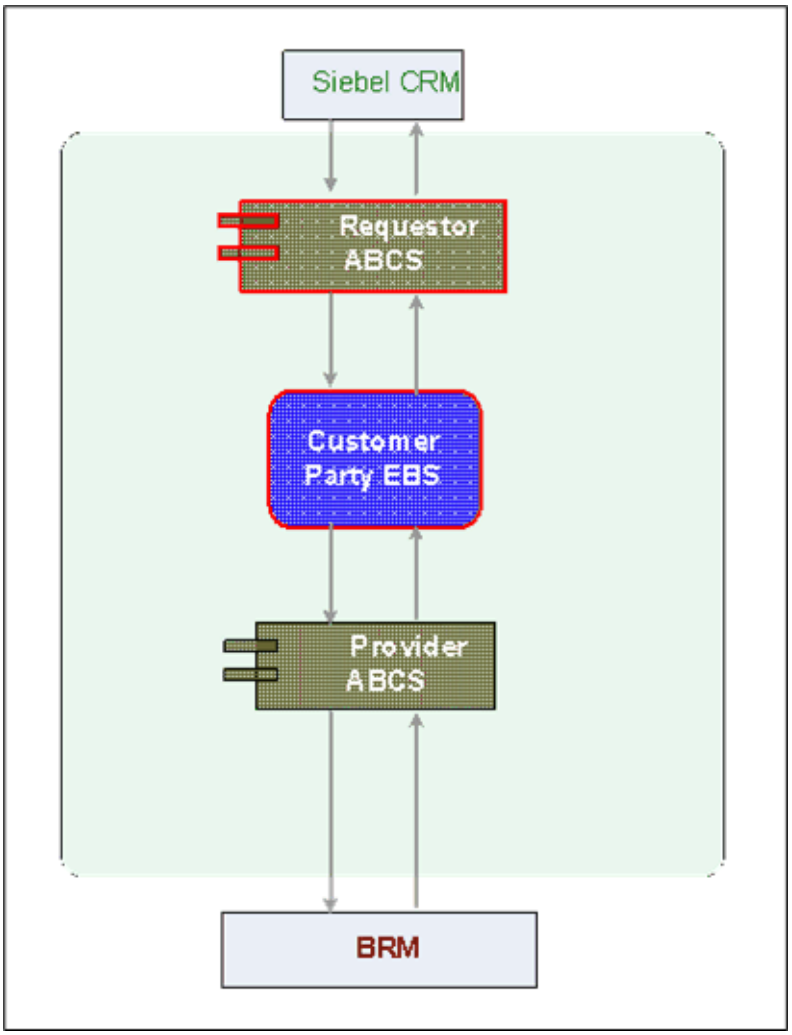
For more information about Validate, see Section 15.8.5, "Validate".

Use Case

CRM Application requesting account details about a customer from the billing system

Figure 14–3 illustrates the two-way communication between AIA services, requester ABCS, and CustomerPartyEBS.

Figure 14–3 GetAccountBalanceSummary Integration Scenario



14.3.2.2 When to Use the Asynchronous Request Only (Fire-and Forget) MEP

Use this pattern when:

- The requester ABCS receives the request message from the participating application and ends with invoking of the EBS. In this case, no response comes from the EBS to the requester ABCS and no response is made to the participating application that initiated the request.
- The provider ABCS receives the request message from the EBS and ends with invoking the provider participating application API. No response is made to the EBS that initiated the request.

Use Case

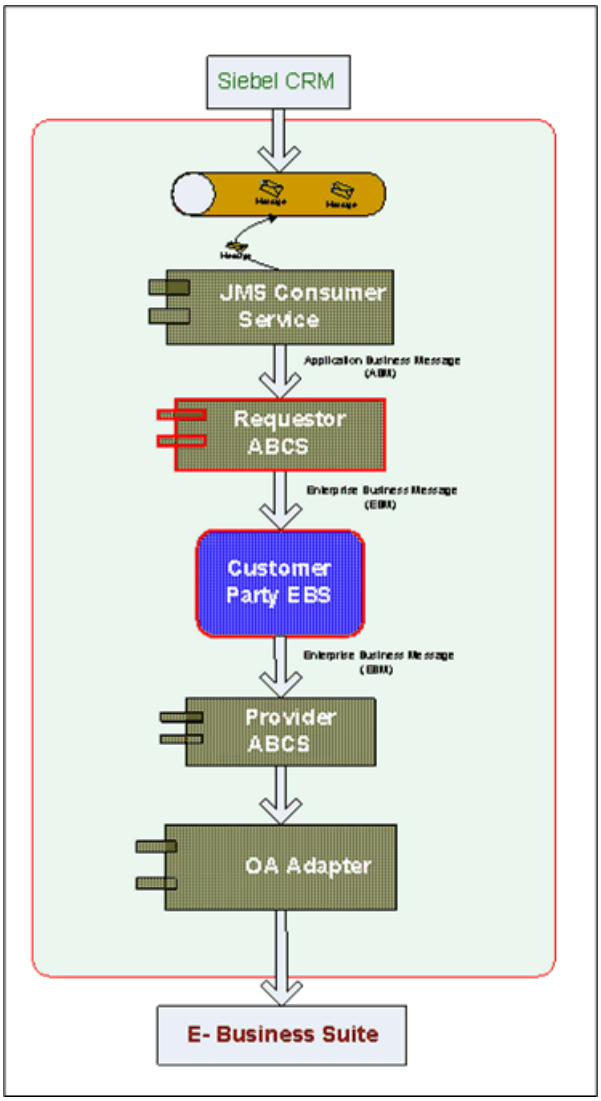
A CRM application submitting a request to the BRM application to create CustomerParty

From the perspective of the participating application, Siebel, a request message to the BRM application to create a CustomerParty is enqueued for processing at a later time.

The Siebel application is free to resume its processing immediately. The request message is dequeued from the queue and is processed in an asynchronous fashion.

Figure 14–4 illustrates the one-way invocation between AIA services, Requester ABCS, and CustomerPartyEBS.

Figure 14–4 Example of One-Way Invocation



14.3.2.3 When to Use the Asynchronous Request Delayed Response MEP

Use this pattern when:

- The requester ABCS receives the request message, processes the message, and finally updates the participating application that initiated the request by invoking an API exposed by the participating application. The participating application is not waiting for the response.
- The provider ABCS receives the request message from the EBS Request mediator service, processes the message, and finally responds to the requesting service-requester ABCS or Enterprise Business Flow (EBF)-by invoking the response operation of the EBS Response mediator service. The EBS Mediator service is not waiting for the response.

The MEP between requester ABCS and EBS can be synchronous or asynchronous, and between the provider ABCS and the provider participating application can also be synchronous or asynchronous.

14.4 Technology Options

This section discusses technology options for:

- Section 14.4.1, "Outbound Interaction with the Application"

  For inbound interactions see Section 15.9, "Invoking the ABCS".

- Section 14.4.2, "Using BPEL for Building ABCS"

For more information about resource connectivity, see Chapter 23, "Establishing Resource Connectivity."

14.4.1 Outbound Interaction with the Application

Communication between ABCSs and the application is governed by the application capabilities.

Communication technologies include:

- JCA Adapters
- Standard web service interfaces
- Adapters
- JMS queues

14.4.1.1 When to Use JCA Adapters for Outbound Interactions

JCA Adapters, when available and authorized for use, should be the first choice for developers to connect to applications.

JCA Adapters:

- Natively invoke participating applications in a co-located fashion.
- Standardize the programming interface.
- Standardize the quality of services (QoS) implementation.
- Provide support for tasks associated with connection management, transaction management, and scalability in combination with the run-time environment.

For more information about establishing outbound connectivity, see Chapter 23, "Establishing Resource Connectivity."

14.4.1.2 When to Use Standard Web Service Interfaces (SOAP/HTTP, XML/HTTP) for Outbound Interactions

This is the most common means of communicating with participating applications.

Most applications expose functionality in the form of a web service. The most common transport used is SOAP over HTTP, however some applications may expose direct XML over HTTP.

14.4.1.3 When to Use JCA Adapters, (Database, File, JMS, or AQJMS), for Outbound Interactions

In situations in which the message submission is decoupled from the processing of the message, usage of queues is suggested. When JMS queues are used, they must be accessed using JMS Adapter. The message is enqueued to the JMS queue by the ABCS and the application would dequeue the message.

> **Tip:** AIA highly recommends that participating applications making an outbound interaction in an asynchronous mode use message queues to publish the requests. AIA recommends this approach because it allows for high scalability and a better end-user experience. Similarly, the ABCS making an outbound interaction to the applications must assess the tasks to be performed by the application. Use of message queues to communicate with the applications in situations in which applications need longer periods to perform the tasks.

Participating applications making an outbound interaction in a synchronous mode alone should send the requests using SOAP/HTTP.

For more information about JMS Adapters, see *Oracle Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server.*

14.4.2 Using BPEL for Building ABCS

AIA recommends using a BPEL component to build an ABCS composite.

BPEL is the most effective solution when:

- You have a long-lived process

  In scenarios in which you have a long-lived process that may take hours or even days, BPEL is the best technology choice. In this case, BPEL persists (dehydrates) the process at certain hook points and then bring the process to life when needed.

- You need complex orchestration capabilities

  In scenarios that require complex orchestration, parallel processing, and multiple conversations with participating applications, and with integrated services such as BPA, BAM, Workflow, and Rules Engine, BPEL is the most effective solution.

- You have content augmentation and validation that cannot be done using XSLT

  In cases in which the decision-making and validation is not simple enough to perform using XSLT, other means are needed such as the standard BPEL procedural constructs or even calling out to the Rules Engine. BPEL enables you to easily perform fairly complicated decision trees and route the results to different partners.

- You need aggregation or disaggregation

  In scenarios in which the ABCS must make multiple calls to the application to process a message or to collect data and then consolidate it in one message, BPEL is the suitable technology. BPEL constructs enable you to split or consolidate payloads and aggregate or disaggregate calls. The BPEL process is also responsible for maintaining the state and handling transactions in between invocations.

- You must augment the participating application functionality.

For more information, see *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

15

Constructing the ABCS

This chapter describes how to construct an Application Business Connector Service (ABCS). It lists the prerequisites that are necessary before you set out to construct an ABCS and briefly introduces you to the concepts of SOA composite application. It provides information about constructing an ABCS using AIA Service Constructor and constructing one using Oracle JDeveloper and lists the tasks that a developer should manually complete after creating an ABCS composite using AIA Service Constructor. Regardless of whether the ABCS is developed either using AIA Service Constructor or entirely using JDeveloper, you should review the relevant sections depending upon the capabilities to be implemented in the ABCS.

This chapter includes the following sections:

15.1 Constructing an ABCS

Table 15–1 lists the common tasks for constructing an ABCS and provides links to detailed information:

Table 15–1 Summary of Tasks for Constructing an ABCS

If	Refer to
You are implementing the Fire-and-Forget Message Exchange Pattern (MEP)	Section 14.3.2.2, "When to Use the Asynchronous Request Only (Fire-and Forget) MEP" Section 15.4, "Implementing the Fire-and-Forget MEP" To implement the MEP in EBS, see Section 13.6, "Implementing the Fire-and-Forget Message Exchange Pattern"

Table 15–1 (Cont.) Summary of Tasks for Constructing an ABCS

If	Refer to
You are implementing the Asynchronous Request Delayed Response MEP	Section 14.3.2.3, "When to Use the Asynchronous Request Delayed Response MEP" Section 15.5, "Implementing the Asynchronous Request Delayed Response MEP" To implement the MEP in EBS, see Section 13.8, "Implementing the Asynchronous Request-Delayed Response Message Exchange Pattern"
You are implementing the Synchronous Request - Response MEP	Section 14.3.2.1, "When to Use the Synchronous Request-Response MEP" Section 15.7, "Implementing Synchronous Request-Response Message Exchange Scenario" To implement the MEP in EBS, see Section 13.7, "Implementing the Synchronous Request-Response Message Exchange Pattern"
Your ABCS is invoking an Enterprise Business Service (EBS) operation	Chapter 13, "Designing and Developing Enterprise Business Services," for the guidelines for working with operations of an EBS
You need details about transforming a message from one format to another	Chapter 25, "Working with Message Transformations."
You need information about how ABCS can be connected with the participating applications both inbound and outbound	Chapter 23, "Establishing Resource Connectivity."
You must know how clients such as applications and other services can invoke the ABCS you are developing	Section 15.9, "Invoking the ABCS"
You are securing an ABCS	Section 16.5, "Securing the ABCS"
You are enabling ABCS to handle errors and faults	Section 16.2, "Handling Errors and Faults"
You want to allow ABCS to be extended by customers	Section 16.1, "Developing Extensible ABCS"
You need guidelines on Composite Application Validation System (CAVS)-enabling the ABCS	Section 16.4, "Developing ABCS for CAVS Enablement"
You need guidelines on how to annotate the SOA composites to facilitate the harvesting of metadata into Oracle Enterprise Repository and for deployment of the service	Chapter 12, "Annotating Composites."

Table 15–1 (Cont.) Summary of Tasks for Constructing an ABCS

If	Refer to
You want to harvest design-time composites into Oracle Enterprise Repository	Section 5.2, "Harvesting Design-Time Composites into Project Lifecycle Workbench and Oracle Enterprise Repository"
You want to deploy the ABCS you are developing using AIA Installation Driver	Chapter 2, "Building AIA Integration Flows." Chapter 3, "Working with Project Lifecycle Workbench."
You want to access AIA Configuration Properties from within an ABCS	Section 2.1.3.9, "How to Work with AIAConfigurationProperties.xml in $AIA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/config."

15.1.1 Prerequisites

Before you begin constructing an ABCS, ensure that:

- The relevant AIA Workstation with Oracle AIA Foundation Pack is installed and the development SOA server is up and running.

  Refer to Section 2.1.3, "How to Set Up AIA Workstation."

- The application entities' schemas (Application Business Message [ABM] schemas) are accessible from Metadata Service (MDS) repository. They should not be part of each ABCS project.

  Refer to Section 2.1.3.4, "Using MDS in AIA."

- Enterprise Object Library containing Enterprise Business Objects (EBOs) and Enterprise Business Messages (EBMs) are accessible in the MDS Repository. EBOs and EBMs should not be part of each ABCS project.

  Refer to Section 2.1.3.4, "Using MDS in AIA."

- All the abstract WSDLs of EBS or participating applications should be accessible from the MDS Repository.

  Tip: The abstract WSDL of an ABCS that is being developed should be accessed from MDS. The exceptions to this rule are:

  - The EBS references WSDLs that define PartnerLink types
  - Participating applications reference WSDLs that define PartnerLink types
  - The adapter WSDLs that are generated by JDeveloper
  - Abstract WSDLs of the services defined at ABCS extension points

  When AIA Service Constructor is used for constructing ABCS, abstract WSDLs of the ABCS generated by AIA Service Constructor must be pushed into MDS.

  For more information, see Section 2.1.3.4, "Using MDS in AIA."

- Requester and provider participating applications have been identified.
- EBOs and EBMs have been identified.

- Functionality mapping between EBS and participating applications is complete. This mapping should include:
  - Data element spreadsheet mapping between the EBO and participating application messages.
  - Operations mapping between the EBS and participating applications.
- The style of interaction (MEP: request-response, fire and forget, asynchronous request, and delayed response) is defined.

  See Section 14.3.2, "Choosing the Appropriate MEP".
- The communication protocols with the participating applications are identified.

  For more information about how an ABCS interacts with participating applications, see the sections in Chapter 23, "Establishing Resource Connectivity."
- Any participating application-specific requirements such as error handling or security have been identified.

15.1.2 ABCS as a Composite Application

AIA services (ABCSs and EBSs) can be developed as a composite application built using SCA building blocks. Four SCA elements apply to these services:

- Service

  Represents an entry point into the SCA component or a composite.
- Reference

  Represents a pointer to an external service outside the scope of the given SCA component.
- Implementation type

  Defines the type of implementation code for a given SCA component that is used for coding the business logic. Example implementation types include BPEL and Mediator.
- Wire

  Represents the mechanism by which two components can be connected to each other. Usually a reference of one component is connected to a service exposed by another component.

An SCA component may expose one or more services, may use one or more references, may have one or more properties defining some specific characteristics, and may be connected to one or more components using SCA wiring.

The building blocks of SCA can be combined and assembled to create composite applications. A composite application can be thought of as a composition of related SCA components that collectively provide a coarse-grained business function.

15.1.3 How Many Components Must Be Built

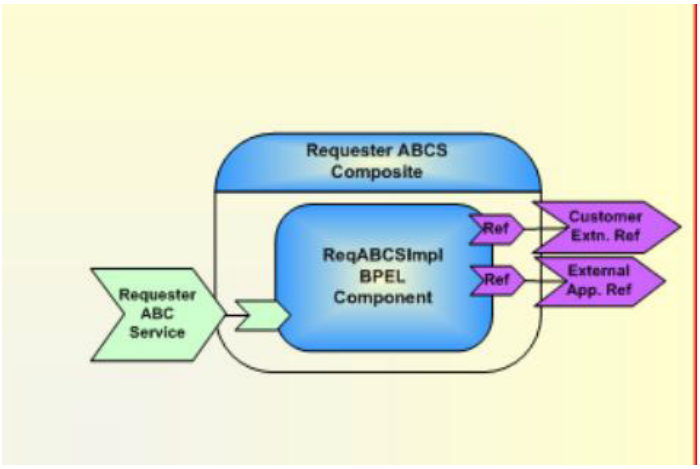
A composite application can be thought of as a composition of related SCA components that collectively provide a coarse-grained business function. A composite may contain certain components that might be required only for the internal implementation of the composite and for which interfaces might not be required outside the scope of the composite in context.

Components may exist that can be used just for the internal implementation of the composite and are thus not required to expose services, references, or both.

A requester ABCS composite as shown in Figure 15–1 can be built using a single component that is exposed as service.

The component can use one reference representing an EBS or more references representing the outbound interactions with other applications.

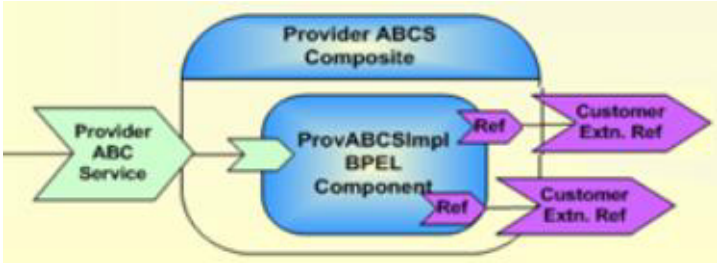
Figure 15–1 Example of a Requester ABCS Composite



Similarly, a Provider ABCS composite as shown in Figure 15–2 can be built using a single component that is exposed as service.

The component can use one or more references representing the outbound interactions with the applications.

Figure 15–2 Example of a Provider ABCS Composite



15.2 Constructing an ABCS Using Service Constructor

AIA Service Constructor is an application that helps jump-start ABCS development by pregenerating AIA artifacts complying with architectural recommendations. It generates artifacts according to the AIA architecture naming recommendations and relieves developers of performing repeatable mundane tasks, making them focus more on value-added business scenario-specific tasks.

For more information about the Service Constructor, see Chapter 4, "Working with Service Constructor."

This section includes the following topics:

- Section 15.2.1, "How to Create the ABCS in the Service Constructor"

- Section 15.2.2, "How to Complete ABCS Development for the AIA Service Constructor"

15.2.1 How to Create the ABCS in the Service Constructor

See Chapter 4, "Working with Service Constructor."

15.2.2 How to Complete ABCS Development for the AIA Service Constructor

After the ABCS is created in the Service Constructor, the following post-ABCS construction tasks must be performed manually.

Complete the following manual post-ABCS construction tasks:

> **Tip:** Ensure that the required AOL schemas are available in the MDS on the server.

1. Move abstract WSDLs of Application Business Service Connector to the MDS.

   Refer to Section 2.1.3.4, "Using MDS in AIA."

2. Configure the JDeveloper to access the MDS repository.

   Refer to Section 2.1.3.4, "Using MDS in AIA."

To complete ABCS development for the AIA Service Constructor:

1. Rearrange service invocations, if necessary.

   The AIA Service Constructor generates relevant definitions for all external service invocations in a sequence in the order that you specify while using the wizard. If your ABCS requires that the invocations be done either in parallel or based on certain other conditions, rearrange the services as needed using Oracle BPEL Designer.

2. Complete the business payload transformation.

   The AIA Service Constructor generates transformations only if the target document is an Enterprise Business Message and generates root element business transformations only. It generates the transformation mapping pertaining to most of the header elements, and the XSL code pertaining to the business payload that is common to all of the ABCSs.

   Use Oracle BPEL Designer to develop the transformation code for transforming remaining parts of the message document to complete the business payload and header transformation.

3. Configure correlation properties for asynchronous application service invocation.

   The AIA Service Constructor does not generate correlation properties for any application target service that is being invoked using an asynchronous request-response MEP.

   Use Oracle BPEL Designer to configure the correlation upon the invoke activity and subsequent receive activity.

4. Create invocations for other operations when a service is called multiple times.

   If a service is called multiple times, the AIA Service Constructor does not generate code for multiple invocations. Copy and paste the invocation code generated for the first service invocation from the same process and rename operations and variables.

5. Populate attributes of binding.ws element of the reference services in the composite.xml file.

   Tip: Always use the abstract WSDL to refer to the services that must be invoked. The reference binding component should always use the abstract WSDL.

   Do not use the concrete WSDLs directly in your consuming artifacts.

   Do not reference the deployed concrete WSDL of a Service Provider

   The following paragraphs describe how you can accomplish this.

   For a better understanding of why this is recommended, see Section 29.1.6, "Separation of Concerns."

   In the composite.xml file, under the element <reference>, the AIA Service Constructor generates empty attributes, port, and location for the element <binding.ws>.

   Populate the attributes with relevant values as specified in the following list.

   a. When the referenced service is BPEL-based, the binding.ws element should be populated as shown in Example 15–1.

Example 15–1 Binding.ws element for BPEL-based Service

```
<binding.ws port="[Namespace of the Service as defined in the
wsdl]#wsdl.endpoint([Name of the Service as given in the WSDL]/[ Name of the
Porttype as given in the WSDL>)"  location="< URL of the concrete WSDL]"/>
```

Tip: The name of the Service is the value of the attribute **definitions/name** in the abstract WSDL.

This follows from naming conventions for the Service name in the ABCS Composite. The name of the service is <name of the composite>, which in turn is the value of the attribute 'name', of the element 'definitions', in the WSDL.

The URL of the concrete WSDL uses this format as shown in Example 15–2:

Note: Populating the 'location' attribute of the element binding.ws with the URL of a run-time WSDL, does not amount to violating the principle of 'designing the service using only abstract WSDLs'. The reason is that this concrete WSDL is not accessed either at composite's design-time or at composite's deploy-time. This WSDL is only accessed at composite's run time.

---

http://{host}:{port}/soa-infra/services/default)/[Name of the Service as given in the attribute 'name' of the element 'definition' in the WSDL]/[Name of the Porttype element as given in the WSDL]?WSDL

Example 15–2 URL of the Concrete WSDL for BPEL-based Service

```
<binding.ws
port="http://xmlns.oracle.com/SamplePortalApp#wsdl.endpoint(SamplesCreateCustomer
PartyPortalProvider/SamplesCreateCustomerPartyPortalProvider)" location="http://
{host}:{port}/soa-infra/services/default/SamplesCreateCustomerPartyPortalProvider
/SamplesCreateCustomerPartyPortalProvider?WSDL"/>
```

b. When the referenced service is mediator-based, then the binding.ws element should be populated as shown in Example 15–3.

Example 15–3 Binding.ws element for Mediator-based Service

```
<binding.ws port="[Namespace of the Service as defined in the
wsdl]#wsdl.endpoint([Name of the Porttype element  as given in the WSDL]_ep/[
Name of the Porttype element as given in the WSDL]_pt)" location="[ URL of the
concrete WSDL]"/>
```

The URL of the concrete WSDL uses this format as shown in Example 15–4:

http://{host}:{port}/soa-infra/services/default)/[Name of the Porttype element as given in the WSDL]/[Name of the Porttype element as given in the WSDL]_ep?WSDL

Example 15–4 URL of the Concrete WSDL for Mediator-Based Service

```
<binding.ws
port="http://xmlns.oracle.com/EnterpriseServices/Core/CustomerParty/V2#wsdl.endpo
int(SamplesCustomerPartyEBS_ep/CustomerPartyEBS_pt)" location="http://
{host}:{port}/soa-infra/services/default/SamplesCustomerPartyEBS/SamplesCustomerP
artyEBS_ep?WSDL"/>
```

TroubleshootingTip: You may see error messages indicating invalid SOA composites after a server restart. This is caused by referring to concrete WSDLs where abstract WSDLs should have been used.

You will not see the problem at the first deployment of a new composite X when you reference the concrete WSDL of composite Y. Why? Composite Y is up and running at the time of the deployment of composite X. The problem starts when the server is restarted because there is no guarantee that the composites will be activated in the right order to resolve any dependencies. If service X is activated before composite Y, the reference cannot be resolved because Y is still down, so X remains in status *Invalid*.

6. Alter policies in the fault policy.

The AIA Service Constructor generates fault policies with default values. Modify these default values based on the service's requirements.

For details about setting up fault policies for AIA services, refer to Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging."

7. Alter properties in the service configuration properties file.

The AIA Service Constructor generates a configuration file snippet for service-specific properties. In case of the service configuration file of a provider ABCS, you must enter the value for the property Routing. <PartnerLinkName>.< Target System ID>>.EndpointURI as shown in Example 15–5.

Example 15–5 Service Configuration Properties File

```
<Property name="Routing. PartnerLinkName>.< Target System ID>.EndpointURI
">http://[hostname]:[portnum]/soa-infra/services/default/[name of the
service]/[name of the exposed endpoint of the service]?WSDL</Property>
```

Example 15–6 is a sample of a completed service configuration properties file.

Example 15–6 Example of Configuration File for Service-specific Properties

```
<Property
name="Routing.SamplesCreateCustomerPartyPortalProvider.SamplePortal.EndpointURI">
http://
{host}:{port}/soa-infra/services/default/SamplesCreateCustomerPartyPortalProvider
/SamplesCreateCustomerPartyPortalProvider?WSDL</Property>
```

> **Note:** You must provide the preceding property in the service configuration file if the value for the property **Routing.<PartnerLink>.RouteToCAVS** is given as *false.*

8. Change the **componenttype file** and **composite.xml** as detailed in the following sections.

   You must make changes to these files to make them point to the abstract WSDLs in the MDS.

Changes Needed in the componenttype File

The attribute ui:wsdlLocation in Service element should point to abstract WSDLs in the MDS as shown in Example 15–7.

Example 15–7 Componenttype File Pointing to Abstract WSDLs in the MDS

```
<service
ui:wsdlLocation="oramds:/apps/AIAMetaData/AIAComponents/ApplicationConnectorServ
iceLibrary/SampleSEBL/RequesterABCS/SamplesCreateCustomerSiebelReqABCSImpl.wsdl"
 name="SamplesCreateCustomerSiebelReqABCSImpl">
```

If the composite has a reference, then the attribute ui:wsdlLocation in *reference* element should point to abstract WSDLs in the MDS as shown in Example 15–8:

Example 15–8 Componenttype File for a Composite with a Reference

```
<reference name="SamplesCreateCustomerPartyPortalProvider"
ui:wsdlLocation="oramds:/apps/AIAMetaData/AIAComponents/ApplicationConnectorServ
iceLibrary/SamplePortal/V1/SamplesCreateCustomerPartyPortalProvider.wsdl">
<interface.wsdl
interface="http://xmlns.oracle.com/SamplePortalApp#wsdl.interface(SamplesCreateC
ustomerPartyPortalProvider? )"/>
</reference>
```

Changes Needed in the composite.xml File

In the composite.xml, the attribute location of the element import should point to abstract WSDLs in the MDS as shown in Example 15–9.

Example 15–9 Composite.xml Pointing to Abstract WSDLs in the MDS

```
<import
location="oramds:/apps/AIAMetaData/AIAComponents/ApplicationConnectorServiceLibr
ary/SampleSEBL/RequesterABCS/SamplesCreateCustomerSiebelReqABCSImpl.wsdl"
namespace=
"http://xmlns.oracle.com/ABCSImpl/Siebel/Samples/CreateCustomerSiebelReqABCSImpl
/V1"/>
```

The attribute ui:wsdlLocation in Service and Reference elements should point to abstract WSDLs in the MDS as shown in Example 15–10.

Example 15–10 Example of composite.xml Pointing to Abstract WSDLs in the MDS

```
<service
ui:wsdlLocation="oramds:/apps/AIAMetaData/AIAComponents/AIAServiceLibrary/Sample
SEBL/RequesterABCS/SamplesCreateCustomerSiebelReqABCSImpl.wsdl"
name="SamplesCreateCustomerSiebelReqABCSImpl">
```

For those referenced services in which the WSDLs do not have partnerLinkType elements in them, their corresponding reference WSDLs should also be changed to refer the abstract WSDLs from the MDS.

For instance, the composite of a requester ABCS has a mediator as a referenced service. Hence, change the location attribute of the element 'import' in the <EBS Reference WSDL> file as shown in Example 15–11.

Example 15–11 Referenced Service WSDLs with no partnerLinkType Elements Pointing to Abstract WSDLs in the MDS

```
<wsdl:import
namespace="http://xmlns.oracle.com/EnterpriseServices/Core/CustomerParty/V2"
location="oramds:/apps/AIAMetaData/AIAComponents/EnterpriseBusinessServiceLibr
ary/Core/EBO/CustomerParty/V2/CustomerPartyEBS.wsdl"/>
```

15.3 Constructing an ABCS Composite Using JDeveloper

An ABCS can be constructed using either AIA Service Constructor or JDeveloper.

While the previous section dealt with constructing an ABCS using Service Constructor, this section provides a high-level overview of constructing the ABCS composite in design mode, using JDeveloper.

The high-level tasks to be performed are:

- Configure JDeveloper to access MDS.

  See Section 2.1.3.4, "Using MDS in AIA."

- Develop abstract WSDL for the ABCS.

  See Chapter 14, "Designing Application Business Connector Services".

- Construct the ABCS composite using JDeveloper.

  See Section 15.3.1, "How to Construct the ABCS Composite Using JDeveloper".

- Develop the BPEL process.

  See Section 15.3.2, "Developing the BPEL Process".

15.3.1 How to Construct the ABCS Composite Using JDeveloper

To create an ABCS using JDeveloper:

1. In JDeveloper, create an SOA composite project.
2. Open the composite.xml in design mode.
3. Add a BPEL component in the Components swim lane.
4. Add a web service as external reference service in the References swim lane.

This reference service could represent an EBS, an Adapter Service, or a participating application. An abstract WSDL of the EBS, Adapter Service, or participating application should be accessible from the MDS Repository.

Note: In the composite.xml, populate the element 'binding.ws' as shown in step 5 of Section 15.2.2, "How to Complete ABCS Development for the AIA Service Constructor".

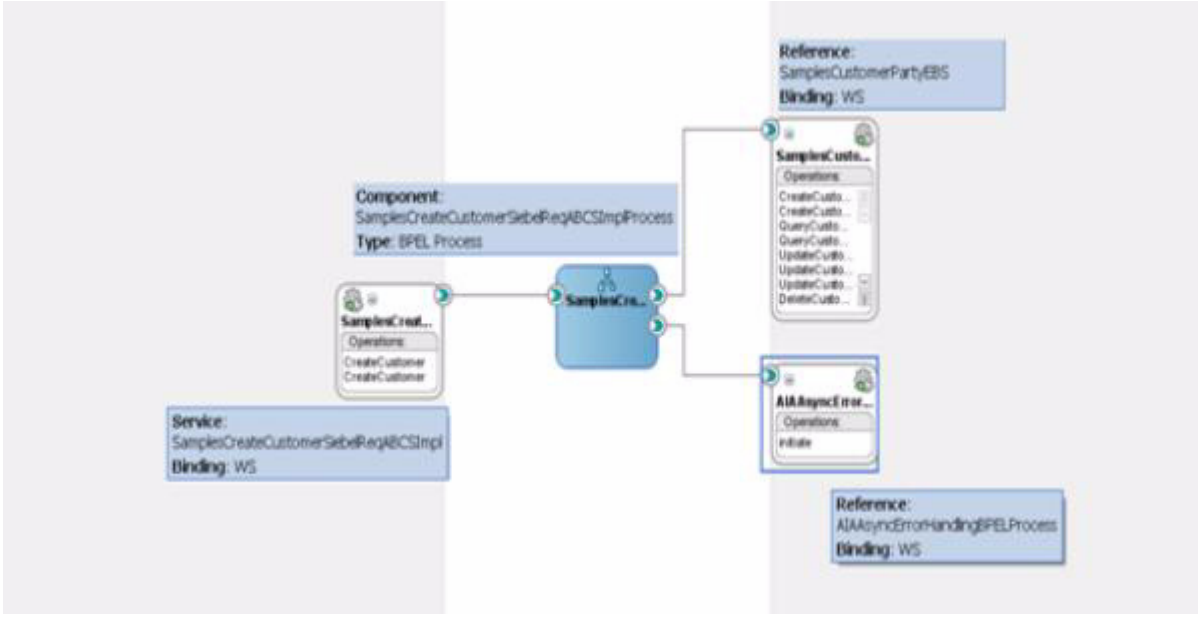
5. Wire the BPEL component to the external reference component created in step 4.
6. Complete the following for handling the faults.
   - Add a web service as external reference service in the References swim lane. Provide the reference to the abstract WSDL of the AIAAsyncErrorHandlingBPELProcess in the MDS repository.

     Wire the BPEL component to the AIAAsyncErrorHandlingBPELProcess service.

     For more information about fault handling in AIA services, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging."

Figure 15–3 illustrates a requester ABCS composite.

Figure 15–3 Example of a Requester ABCS Composite



15.3.2 Developing the BPEL Process

After you have developed the composite in design mode using JDeveloper, the ABCS that is implemented as a BPEL process component should be constructed to its completion.

In the ABCS, the following tasks are accomplished:

- Message enrichment
- Message header population
- Message content transformation
- Error handling and logging

- Extensibility
- CAVS enablement
- Security context implementation

For more information about completing ABCS development, see Chapter 16, "Completing ABCS Development".

15.3.3 How to Create References, Services, and Components

- For each target, add a reference.
  - The target can be a web service or adapter.
  - If the target service requires transformation, create a mediator component in between.
- For each interface, add a service.
  - The service can be a web service or adapter.
  - If the adapter interface requires transformation, create a mediator component in between.

15.3.4 Moving Abstract Service WSDLs in MDS

SOA Suite 11*g* has introduced a central repository, **Metadata Service (MDS)**, that backs up your application (and hence your composites) at design time and run time. MDS is like a version management system, where you can store and use the shared common artifacts at design and run time.

AIA leverages the MDS repository to store common design time artifacts that are shared with all developers. There is a common directory of abstract WSDLs in the MDS, which is a centrally accessible endpoint.

The artifacts that are stored in the MDS include:

- Schemas: EBO schemas and ABM schemas
- WSDLs: Abstract WSDLs of EBS, ABCS, Adapter Services, CBPs and EBFs

All the abstract WSDLs of all AIA services are put in MDS. You build your consuming AIA service composite application using the abstract WSDL of a referenced service. **No dependency exists on the order of deployment for composites.**

Since there is no dependency on the referenced services during the deployment of the composites, there is no dependency on the order of deployment for the composites. A referenced service does not have to be deployed to be referenced by another service. This also resolves the cases where there are cyclic references involved among the services.

> **Tip:** Always use the abstract WSDL to refer to the services that must be invoked. The reference binding component should always use the abstract WSDL.
>
> Do not use the concrete WSDLs directly in your consuming artifacts.
>
> Do not reference the deployed concrete WSDL of a Service Provider
>
> The following paragraphs describe how this can be accomplished.

To get a better understanding of why this is recommended, see Section 29.1.6, "Separation of Concerns."

For details of how MDS is used in AIA, refer to Section 2.1.3.4, "Using MDS in AIA."

Tip: Abstract WSDLs must be modified to access the required schemas from the MDS and then moved to MDS.

Troubleshooting Tip: You may see error messages indicating invalid SOA composites after a server restart. This is caused by referring to concrete WSDLs where abstract WSDLs should have been used.

You will not see the problem at the first deployment of a new composite X when you reference the concrete WSDL of composite Y. Why? Composite Y is up and running at the time of the deployment of composite X. The problem starts when the server is restarted because there is no guarantee that the composites will be activated in the right order to resolve any dependencies. If service X is activated before composite Y, the reference cannot be resolved because Y is still down, so X remains in status *Invalid*.

For ABCSs and Adapter services, the abstract WSDLs are to be located in MDS at:

AIAMetaData/AIAComponents/ApplicationConnectorServiceLibrary/<PRODUCT_CODE>/<Version Number>/<Service Type>

Possible values for <Service Type> are *RequesterABCS, ProviderABCS, AdapterServices*.

Possible values for <Version Number> are *V1, V2*.

The utility for moving artifacts to MDS is described in Section 2.1.3.4, "Using MDS in AIA".

Examples:

```
AIAMetaData/AIAComponents/ApplicationConnectorServiceLibrary/Siebel/V1/RequesterABCS
AIAMetaData/AIAComponents/ApplicationConnectorServiceLibrary/Siebel/V1/ProviderABCS
```

15.4 Implementing the Fire-and-Forget MEP

If you are implementing the fire-and-forget MEP, this section provides the necessary guidelines.

For more information about the scenarios for which this MEP is recommended, see Section 14.3.2.2, "When to Use the Asynchronous Request Only (Fire-and Forget) MEP".

In the fire-and-forget pattern, no response is sent back to the requester. In situations in which the provider ABCS is not able to successfully complete the task, transactions should be rolled back and error notifications should be sent.

For more information on how to model transactions, see Chapter 25, "Working with Message Transformations"

For more information about how to handle faults, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging."

In scenarios where the RequesterABCS is invoked by a JMS-based transport adapter, the JMS destination can be thought of as a milestone. In these circumstances, it is possible to configure the JMS consumer adapter and the requester ABCS for the automatic message resubmission of the failed messages.

For more information about message resubmission, see "Using the Message Resubmission Utility" in the *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack*.

This section includes the following topics:

- Section 15.4.1, "When to Use Compensating Services"
- Section 15.4.2, "How to Invoke the Compensating Service"
- Section 15.4.3, "Additional Tasks Required in Provider ABCS to Implement This MEP"
- Section 15.4.4, "How to Ensure Transactions"
- Section 15.4.5, "How to Handle Errors"

15.4.1 When to Use Compensating Services

Sometimes an automatic correction of data or a reversal of what has been done in requester service is needed. The typical scenario is one in which an error occurs in the transaction and the transactions cannot be rolled back.

In these situations, the requester application can implement the compensation service operation and pass the name of the service operation to the provider ABCS. The provider ABCS invokes this compensation service operation if there are errors. There may be a requirement to implement a compensating service for the requesting service.

15.4.2 How to Invoke the Compensating Service

To invoke the correct compensating service from the providing service:

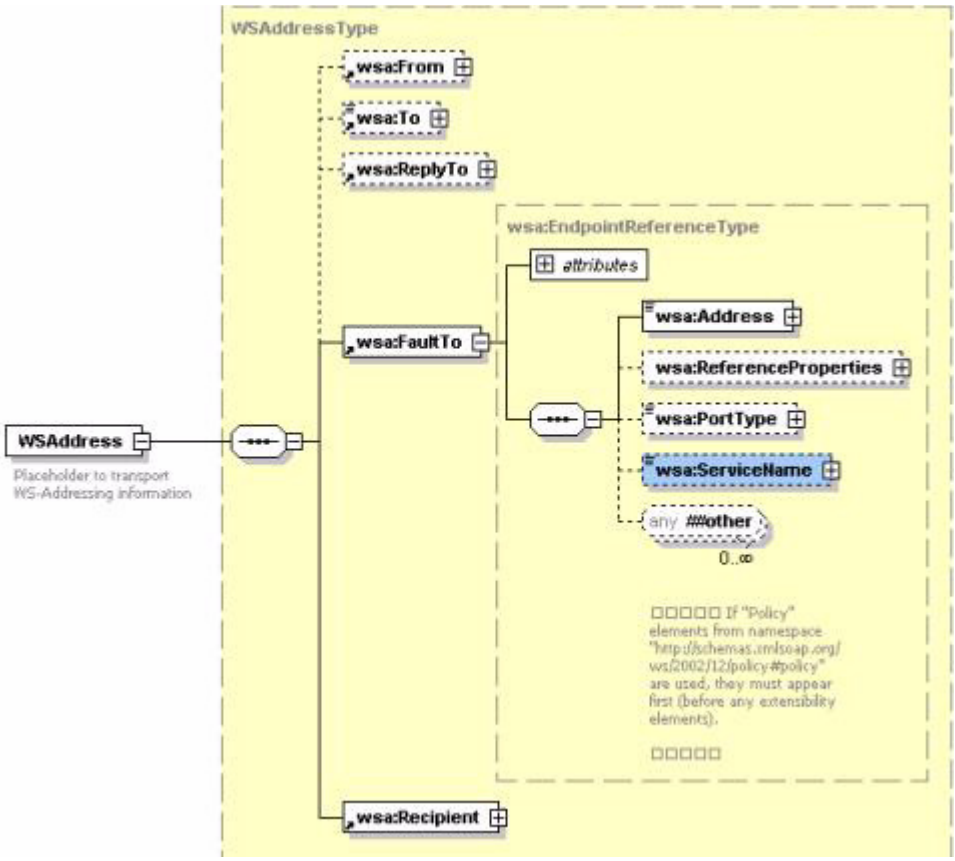
1. Populate the EBMHeader/Sender/WSAddress/wsa:FaultTo/wsa:ServiceName with the name of the compensating service in the transformation used for constructing the request EBM.

   Example of the name of the compensation service: *CompensateCreateOrderSiebelReqABCSImpl*

   For more information about naming, see Chapter 31, "Oracle AIA Naming Standards for AIA Development."

   Figure 15–4 illustrates the structure of the **WSAddressType**.

Figure 15–4 Structure of the WSAddressType



This information is used in the compensate operation of the EBS to route the request for compensation to the correct compensating service.

15.4.3 Additional Tasks Required in Provider ABCS to Implement This MEP

For information about implementing this MEP, see Section 15.6, "Implementing Provider ABCS in an Asynchronous Message Exchange Scenario".

15.4.4 How to Ensure Transactions

For information about ensuring transactions, see Section 16.6.1, "How to Ensure Transactions in AIA Services".

15.4.5 How to Handle Errors

For information about error handling, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging.".

15.5 Implementing the Asynchronous Request Delayed Response MEP

If you are implementing an asynchronous request-delayed response MEP, this section provides the necessary guidelines.

For more information about the scenarios for which this MEP is recommended, see Section 14.3.2.3, "When to Use the Asynchronous Request Delayed Response MEP".

This section discusses how to implement the asynchronous request-delayed response MEP.

Implementing this MEP requires that:

- You have populated the EBM header for asynchronous delayed response.
- You have defined correlation sets and correlation IDs.
- You have used the appropriate programming model for Requester ABCS, EBS, ProviderABCS and ResponseEBS for handling the scenario where error response has to be sent back to either Requester ABCS or to a error handler service.

This section includes the following topics:

15.5.1 How to Populate the EBM Header for Asynchronous-Delayed Response

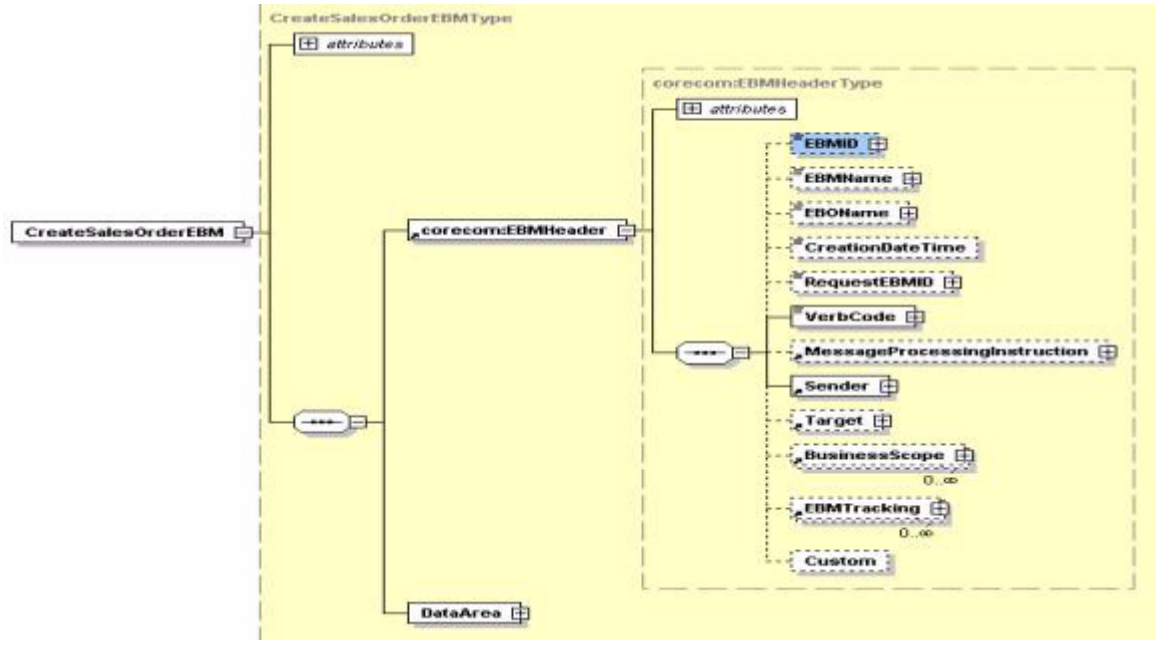
To populate the EBM header for asynchronous delayed response:

1. Populate the **EBMID**.

   This is used for correlation of the response to the correct requesting service instance.

   Figure 15–5 illustrates the structure of the **CreateSalesOrderEBMType**.

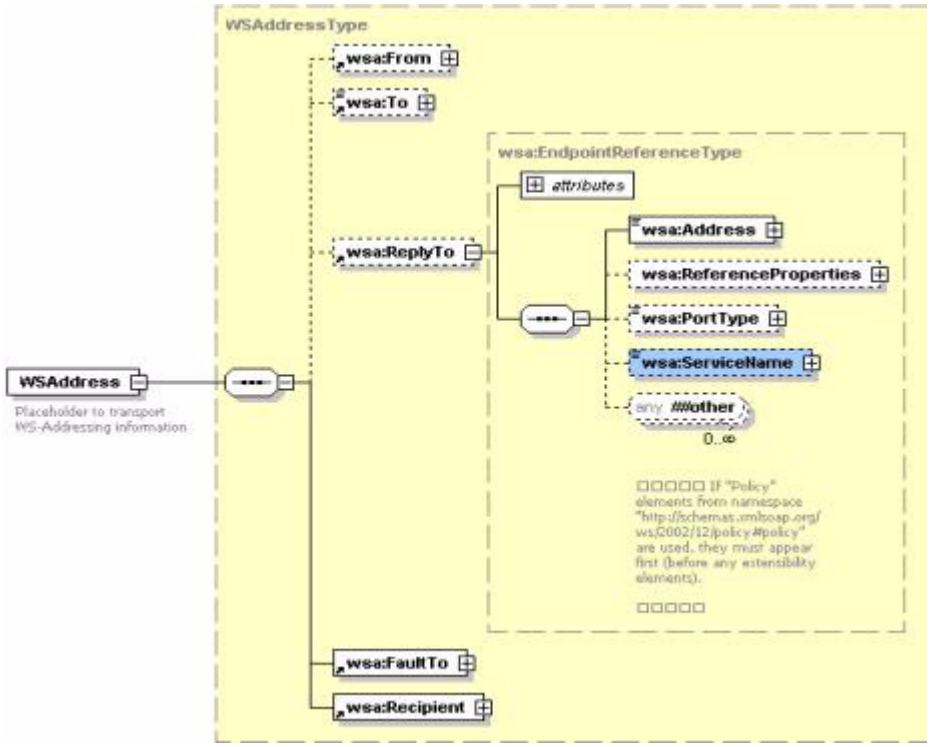
Figure 15–5 Structure of the CreateSalesOrderEBMType



2. Set the **EBMHeader/Sender/WSAddress/wsa:ReplyTo/wsa:ServiceName** to the name of the requesting service name in ABM to EBM transformation as shown in Figure 15–6.

   This is the name of the service that must be invoked by the EBS for processing the response message. In most of the situations, it is the same requester ABCS that is also be responsible for processing the response message coming from provider ABCS.

Figure 15–6 Structure of the SWSAddressType



   This information is used in the response operation of the EBS to route the response from the providing service to the correct requesting service.

3. Set the **responseCode** attribute of the EBM verb.

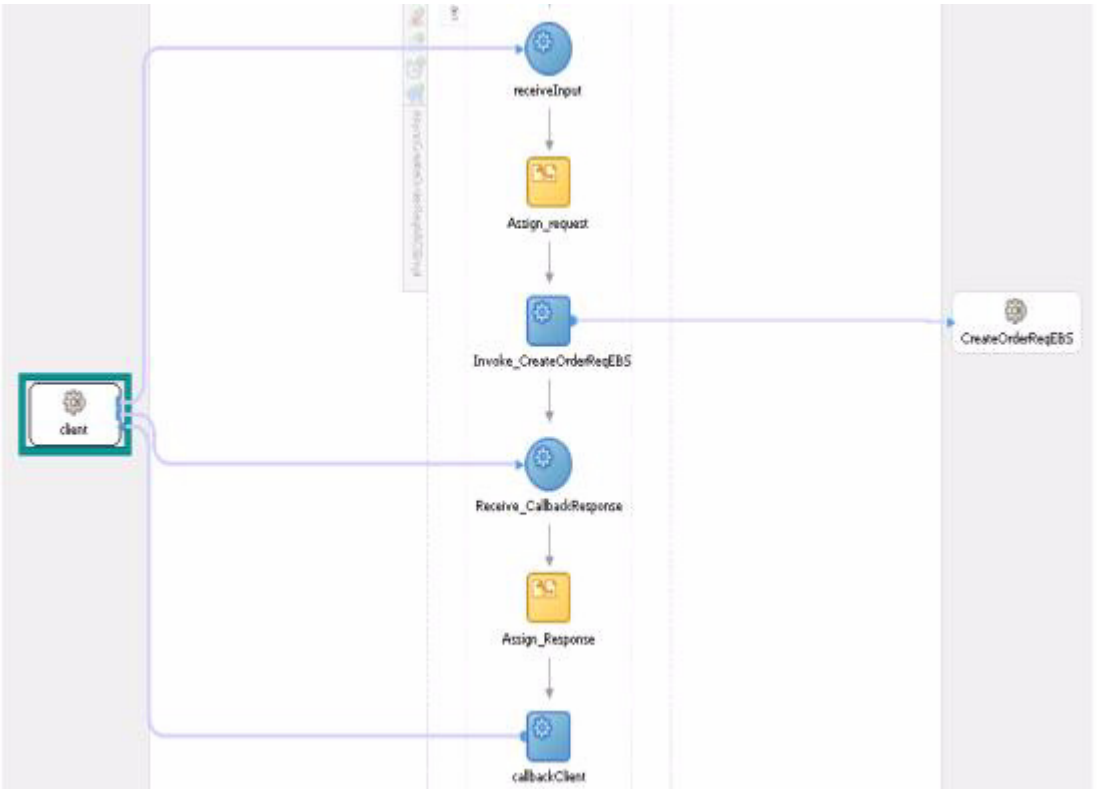
   The **responseCode** attribute of the EBM verb is set to indicate to the providing service that the requesting service is expecting a response. This is evaluated in the providing service to send back a response.

4. Set correlation information in the requesting service partner link.

   The delayed response from the providing service routed by a response EBS would be received by a **Receive** activity. After the **Invoke** step is performed in the requesting service and the process moves to the Receive step, the BPEL process instance is suspended and dehydrated. This restarts after the response is received. To facilitate this behavior, a correlation set has to be specified on the partnerLink for the Receive.

   The requester ABCS process should look like the example in Figure 15–7. This is the process for which you must set the correlation IDs.

Figure 15–7 Example of Requester ABCS Process



15.5.2 Setting Correlation for the Asynchronous Request-Delayed Response MEP

In the process described in Section 15.5.1 you must set the correlation in two places:

- **Correlation Set**

  Add a correlation ID and create a correlation set for the *Invoke* activity where the process would go into the dehydration store

  Add a correlation ID and create a correlation set for the *Receive* activity where the process would be recalled from the dehydration store after getting a delayed response from the provider/edge application.

- **Correlation Property**

  Add a standard name-value pair for each partnerLink that is linked to the Invoke or Receive activities where the correlation sets are defined as mentioned previously. The property should always be defined as *correlation = correlationSet*.

15.5.3 Programming Models for Handling Error Response in the Asynchronous Request-Delayed Response MEP

This section discusses programming models for:

- Using a separate service for error handling
- Using JMS queue as milestone between RequesterABCS and the EBS
- Using a parallel routing rule in the EBS

15.5.3.1 Programming Model 1: Using a Separate Service for Error Handling

In this model, when a message is successfully processed, the response is sent back to the same requester that initiated the process. However, when the message errors out during the processing in the ProviderABCS, it is routed to a separate error handler service using ResponseEBS.

An inbound message reaching the Requester ABCS is passed to the EBS which invokes the Provider ABCS, for the message processing.

After the message is processed, the Provider ABCS pushes the message into a JMS Queue.

If an error occurring during the processing of the message, an error response message should be constructed and the EBM response header should indicate that the response is indeed an error message.

In a successful scenario, the response EBS routes the response message to the initiating requesterABS.

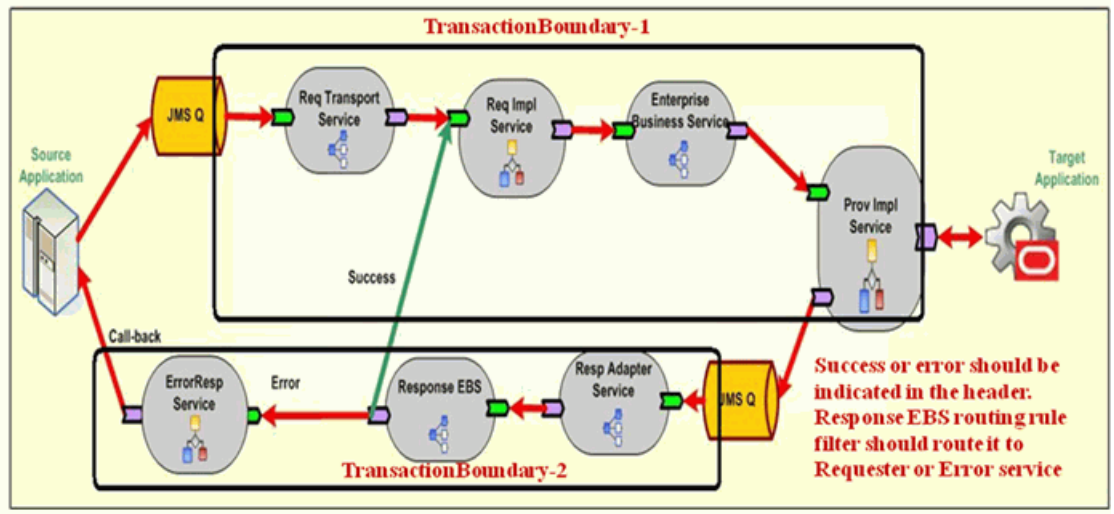
In a failure scenario, the response EBS should route the message to an error handler service.

> **Note:** Publish the messages to JMS Queue from Provider ABCS in both **Success** and **Error** scenarios, if the Provider ABCS is required to send the response or the error message.

This model has two transactions as shown in Figure 15–8.

Transaction 1 starts with de-queuing message by the requesterABCS or the external application directly calling the RequesterABCS. This transaction ends when the provider ABCS publishes either the reply or error message, as the case may be, to the JMS Queue.

Figure 15–8 Programming Model 1: Using a Separate Service for Error Handling



15.5.3.2 Programming Model 2: Using JMS Queue as a Milestone Between Requester ABCS and the EBS

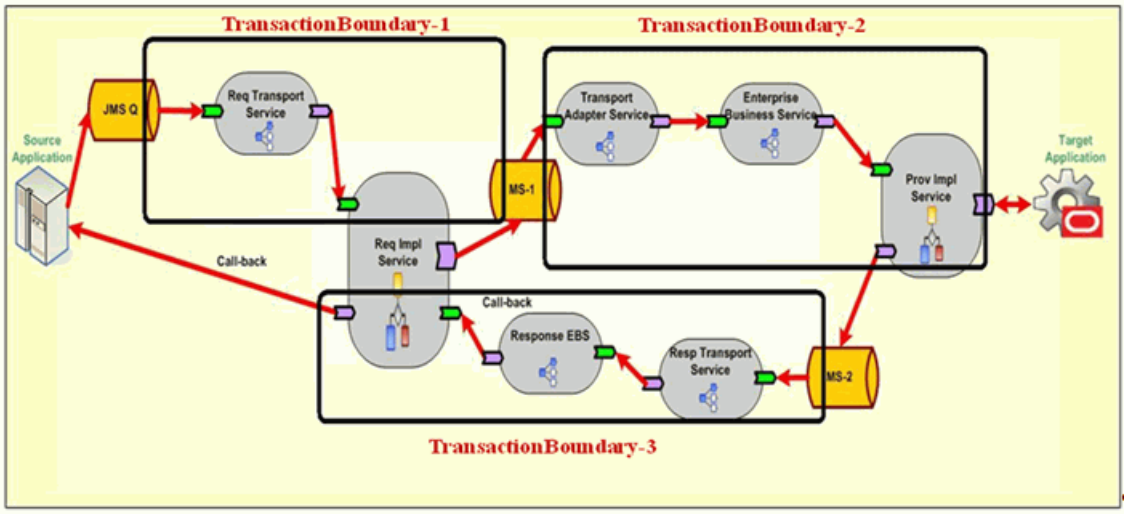
In this model, shown in Figure 15–9, the requester ABCS publishes the inbound message to a JMS milestone. The transaction starts with de-queueing of the message by the requester ABCS or the external application directly calling the requester ABCS. The transaction ends with requester ABCS publishing the message to the JMS queue.

A second transaction starts with de-queueing the message from the JMS queue and invoking EBS. The EBS routes the inbound message to the ProviderABCS for the processing.

In the case of a successful message processing, the provider ABCS invokes the response EBS, using a native binding call, in the existing transaction.

If an error occurs during the message processing, the provider ABCS publishes the errored-out message into another JMS queue. The response EBS picks up the message and invokes the fault handler service

Figure 15–9 Programming Model 2: Using JMS Queue as a Milestone Between Requester ABCS and the EBS



Tip: Use this Queue in the Provider ABCS Reference component ONLY for an Error scenario.

15.5.3.3 Programming Model 3: Using a Parallel Routing Rule in the EBS

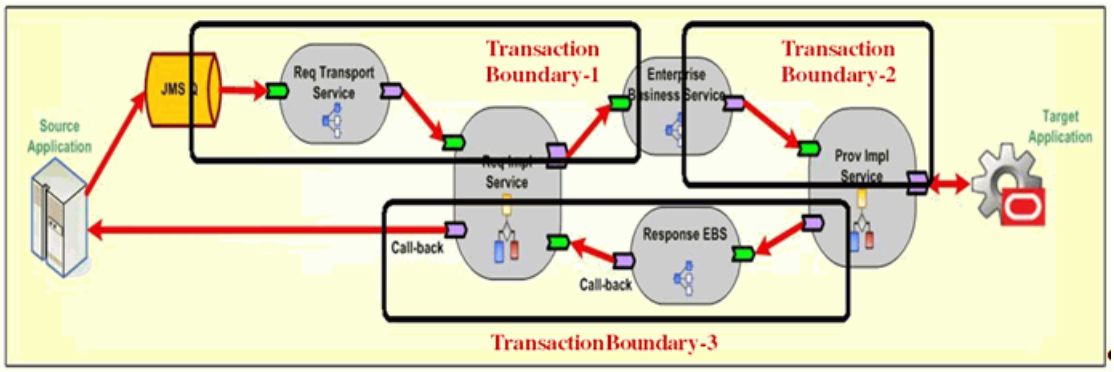
An inbound message reaching the Requester ABCS is passed to the EBS. The EBS routes the message to the Provider ABCS for the message processing, using a parallel routing rule. The transaction starts with de-queuing of the message by the requester ABCS or the external application directly calling the requester ABCS. The transaction ends when the EBS persists the inbound message for a queued execution.

Figure 15–10 illustrates this programming model.

A second transaction starts with de-queuing message from EBS. In case of a successful message processing, provider ABCS making a native binding call to the response EBS in the existing transaction. The response EBS routes the response to the requester ABCS by invoking another receive activity. In case of errors, the Provider ABCS makes a web service call to invoke the Response EBS thereby causing a new transaction to

start. In this transaction, the Response EBS is responsible for sending the error message back to the application using either Requester ABCS or directly.

Figure 15–10 Programming Model 3: Using a Parallel Routing Rule in the EBS



15.5.4 What Tasks Are Required in Provider ABCS to Implement This MEP

For details about the tasks see Section 15.6, "Implementing Provider ABCS in an Asynchronous Message Exchange Scenario".

15.6 Implementing Provider ABCS in an Asynchronous Message Exchange Scenario

If you are implementing asynchronous MEP in the provider ABCS, this section provides the necessary guidelines.

For more information about the scenarios for which this MEP is recommended, see Section 14.3.2.3, "When to Use the Asynchronous Request Delayed Response MEP".

The provider ABCS is implemented to either behave as a one-way service call pattern or request-delayed response pattern.

In the request-delayed response pattern, the provider ABCS receives the request message from the EBS request routing service, processes the message, and finally responds to the requesting service (requester ABCS or Enterprise Business Flow [EBF]) by invoking the response operation of the EBS response routing service. In some scenarios, the provider ABCS can also publish the response and/or the error message to a JMS queue. The EBS request routing service is not waiting for the response.

See Section 15.5.3.1, "Programming Model 1: Using a Separate Service for Error Handling"

All the provider ABCSs (and EBFs) should have the capability to invoke the callback operation, but should have a switch case to do it only if the requester wants a callback. Evaluate the responseCode attribute on the verb element of the EBM to determine whether the requesting service is expecting a response.

15.6.1 How to Implement the Asynchronous MEP

To implement the asynchronous MEP in the provider ABCS:

1. Populate the EBM header of the request message in the providing service with fault information.

If an error occurs in the provider service before evaluation of the requirement of a response and an exception is issued, enter the fault information in the request message EBM header. This facilitates passing the entire request EBM along with the fault message to the catch block for processing.

2. Implement Error Handling in the providing service.

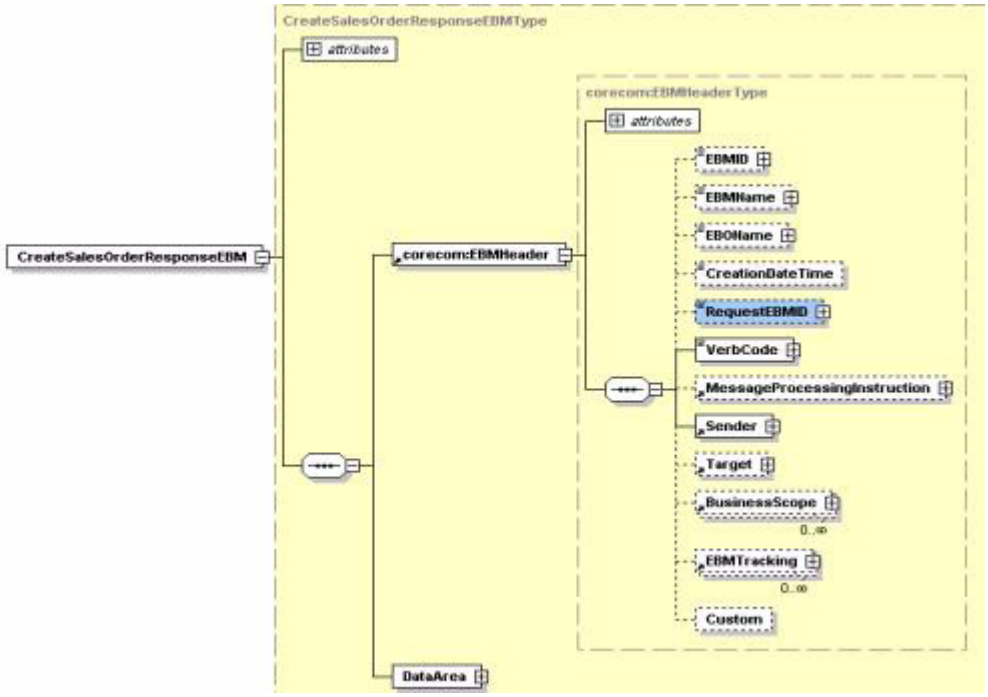
   If a particular error needs compensation service to be invoked, then the compensate operation of the EBS is invoked for routing the request to the compensation service.

   For more information, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging."

3. Enter the correlation information in the EBM header in the providing service.

   Use the EBMID in the EBM header for correlation of the response message to the correct requesting service instance. To facilitate correlation, the EBMID in the EBM header of the request message must be copied to the RequestEBMID in the EBM header of the response message in the ABM to EBM transformation of the providing service as shown in Figure 15–11.

Figure 15–11 Structure of CreateSalesOrderResponseEBMType Element



   For information about the technique of passing the EBMHeader of the request message in a variable into the ABM to EBM XSLT, see Chapter 25, "Working with Message Transformations."

   This is required to copy the relevant information from the EBM header of the Request EBM to the EBM header of the Response EBM.

4. Populate the EBM header of response message in the providing service with fault information.

   If an error occurs in the provider service after evaluation of the requirement of a response, you must populate the fault information in the response message EBM

header fault component. This facilitates passing the response EBM along with fault message to the catch block for processing.

The requesting service receives a response with fault elements populated.

15.6.2 Using the Programming Models for the Request-Delayed Response Pattern

If you use the programming models found in Section 15.5.3, "Programming Models for Handling Error Response in the Asynchronous Request-Delayed Response MEP" for the request-delayed response pattern, follow these guidelines.

Programming Model 1: Using a Separate Service for Error Handling

The provider ABCS should publish the messages to JMS Queue in both successful message processing and error scenarios, if the provider ABCS is required to send the response or the error message.

Note: A new transaction is started with de-queuing of the message from JMS.

Programming Model 2: Using JMS Queue as a Milestone Between the Requester ABCS and the EBS

In the case of successful message processing, the provider ABCS invokes the response EBS, using a native binding call, in the existing transaction.

If an error occurs during the message processing, the provider ABCS publishes the errored-out message into another JMS queue. The response EBS picks up the message and invokes the fault handler service

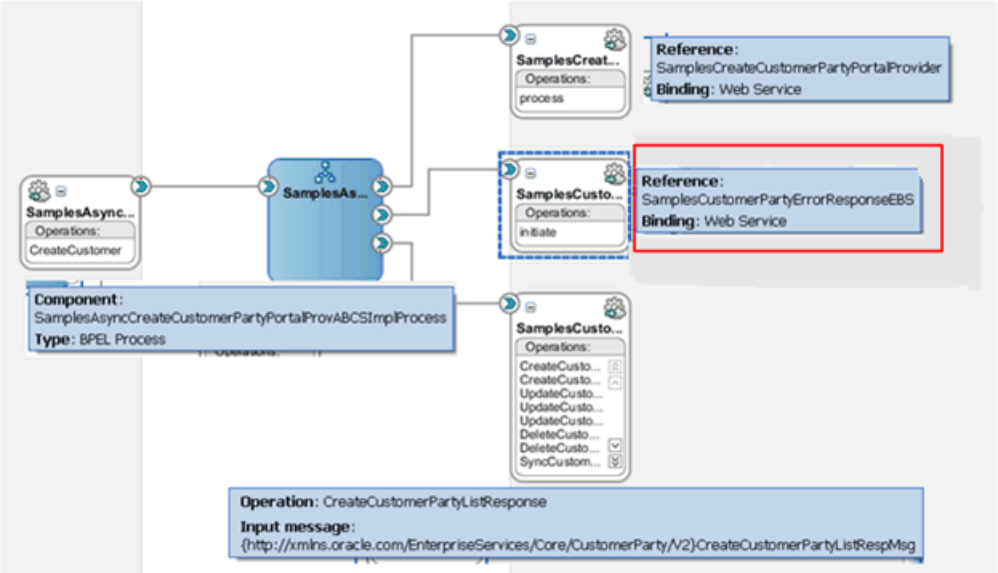
Programming Model 3: Using a Parallel Routing Rule in the EBS

The provider ABCS has two references to the response EBS composite. Follow the naming conventions to name the provider ABCS 's second reference to the EBS.

For details on naming conventions, see Chapter 31, "Oracle AIA Naming Standards for AIA Development"

Figure 15–12 illustrates the asynchronous provider ABCS composite with a second reference to the ResponseEBS. The label for the second reference is *<EBSName>ErrorResponseEBS*.

Figure 15–12 Async Provider ABCS Composite with a Second Reference to ResponseEBS



When the message is processed successfully, the provider ABCS invokes the callback operation on the response EBS. The response EBS routes the response to the requester ABCS by invoking another receive activity.

When en error occurs during the processing of the message, the provider ABCS invokes the response EBS through its second reference to the response EBS. In this case, the responseEBS is invoked using a SOAP call, not a native call. This is achieved by having the following property set on the second reference to the response EBS, in the composite of the provider ABCS.

```
<binding.ws port="<port>" location="<url>"/>
      <property name="oracle.webservices.local.optimization
type="xs:boolean">false</property>
  </binding.ws>
```

Note: Use this property in the provider ABCS reference component ONLY for the error scenario.

15.6.3 How to Ensure Transactions in Services

For more information about ensuring transactions, see Section 16.6.1, "How to Ensure Transactions in AIA Services".

15.6.4 How to Handle Errors in the Asynchronous Request-Delayed Response MEP

See Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging."

15.7 Implementing Synchronous Request-Response Message Exchange Scenario

If you are implementing synchronous request-response MEP, this section provides the necessary guidelines.

For more information about the scenarios for which this MEP is recommended, see Section 14.3.2.1, "When to Use the Synchronous Request-Response MEP".

In this scenario, requester ABCS synchronously invokes the EBS. The EBS invokes the Provider ABCS and waits for a response. Synchronicity should manifest in the WSDLs of requester ABCS, EBS, and provider ABCS. All of the WSDLs should have the operation defined with input and output message.

15.7.1 How to Ensure Transactions in Services

This MEP need not support transactions. No break points such as midprocess Receive, wait, Pick, onMessage activities should be in any of the BPEL processes participating in implementation of this MEP.

For more information about ensuring transactions, see Section 16.6.1, "How to Ensure Transactions in AIA Services".

15.7.2 How to Handle Errors in the Synchronous Request-Response MEP

See Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging."

15.7.3 How to Optimize the Services to Improve Response Time

Because this MEP involves implementation of transient services (no break point activities and hence, no dehydration in the middle), Oracle AIA highly recommends that the audit details for BPEL services are persisted only for faulted instances. The service instances associated with the successfully completed integration flows are not visible using Oracle Enterprise Manager. This approach significantly improves the response time.

auditLevel property sets the audit trail logging level. This configuration property is applicable to both durable and transient processes. This property controls the amount of audit events that are logged by a process. Audit events result in more database inserts into the audit_level and audit_details tables, which may impact performance. Audit information is used only for viewing the state of the process from Oracle Enterprise Manager Console. Use the Off value if you do not want to store any audit information. Always choose the audit level according to your business requirements and use cases.

For synchronous BPEL processes, AIA recommends nonpersistence of instance details for successfully completed instances. For this, set the auditLevel property to off at the service component level. This general guideline could be overridden for individual services based on use cases.

15.8 Invoking Enterprise Business Services

This section discusses the guidelines for invoking an EBS from an ABCS and for working with operations of an EBS. This content is provided from the perspective of invoking an EBS operation and also helps in understanding EBS operations from an implementation perspective.

This section includes the following topics:

- Section 15.8.1, "Create"
- Section 15.8.2, "Update"
- Section 15.8.3, "Delete"
- Section 15.8.4, "Sync"
- Section 15.8.5, "Validate"
- Section 15.8.6, "Process"
- Section 15.8.7, "Query"

These sections present a detailed view of each of the verbs in the context of:

- When the verb should be used
- What should be the content payload when the verb is used
- What are the attributes of the verb, if any
- What is the corresponding response verb, if any
- What is the content payload for the response verb

15.8.1 Create

The **Create** verb indicates a request to create a business object using the information provided in the payload of the Create message. It is used in operations that are intended to create an instance of a business object.

When to Use the Create Verb

If both the service requester and service provider have the same object, then Sync would be a more appropriate verb to use. However, the usage of Sync is conditional on the service provider having an implementation of a Sync operation.

A typical use of a Create operation is a front-end customer management system that could take service requests from customers and based on the information provided, request a work order system to create a work order for servicing the customer.

The Create verb is not used for composite operations; it always involves the creation of one (or more for List) instance of a business object.

Content Payload

The payload of an operation that uses a Create verb is typically a complete business object, and in general every business object has only two messages with a Create operation (Single and List).

Verb Attributes

The Create verb has an optional ResponseCode attribute that communicates the payload that is expected in the response message to the Create request. The possible values for the ResponseCode are restricted to either *ID* (response payload is expected to be the Identifier of the object that was created) or *OBJECT* (response payload is expected to be the entire object that was created).

Corresponding Response Verb

The Create verb has a paired **CreateResponse** verb that is used in the response message for each Create EBM. The response is expected to be provided by the target

application only if the original request message explicitly requests a response by setting the ResponseCode attribute in the Create message.

Response Content Payload

The payload type of the response verb is always based on the payload type of the Create Request operation. It is implemented as a different type to support user customization if needed.

15.8.2 Update

The **Update** verb indicates a request to a service provider to update an object using the payload provided in the Update message. It is used in operations that are intended for updating one or more properties of a business object or array of business objects.

Operations that use the Update verb *must* create or update content and *should not* be an orchestration of other operations.

When to Use the Update Verb

Similar to Create, a business process invokes an Update operation mainly in cases in which the source event that triggers the invocation is *not* the updating of the object in the requesting system. If both the service requester and service provider have the same object, then Sync would be a more appropriate verb to use. However, use of Sync would be conditional to the service provider having an implementation of a Sync operation.

An example of an Update operation is a receiving system updating a purchase order line with the quantity of items received, or an order capture system updating the customer record with customer address and other information.

The content included in the business payload of an EBM using the Update verb is assumed to be *only* the fields that must be updated in the target application. The Update verb uses the *ActionCode* property of the business components and common components to communicate processing instructions to the target system. This is necessary for hierarchical, multilevel objects when the update happens at a child level.

The ActionCode property exists in an EBO for all components that have a multiple cardinality relationship with its parent. The possible values for the ActionCode are *Create*, *Update*, and *Delete*. It is intended for use only with the Update verb to communicate the processing action to be applied to that business component. Ensure that the ActionCode applies only if the object has child business components-if not, an ActionCode is not needed and the verb alone is sufficient to convey this information.

A use case for ActionCode is a case in which a requisition is updated in a self-service requisitioning system, and this updated information must be communicated to the Purchasing system, which also maintains a copy of the requisition.

Assume that a user updates a requisition and does the following actions (in a single transaction):

- Updates the description in the requisition header
- Adds a new requisition line (line number 4)
- Modifies the item on a requisition line (line number 3)
- Deletes a requisition line (line 2)
- Modifies the accounting distribution of a requisition line, and adds a new accounting distribution (line 1)

The content of the DataArea business payload in the instance XML document that communicates the preceding changes is shown in Example 15–12.

Example 15–12 DataArea Business Payload in the instance XML Document

```
<UpdateRequisition actionCode="Update">   Root Action Code not processed
    <Description>New Description</Description>
   <RequisitionLine actionCode="Update"> Indicates that some property or
association of this line is being updated. In this example, the accounting
distribution Percentage has been updated for Line 1, and a new
AccountingDistribution line has been added
         <Identification>
               <ID>1</ID>
         </Identification>
         < RequisitionAccountingDistribution actionCode="UPDATE">
               <Identification>
                     <ID>1</ID>
               </Identification>
               <AccountingDistribution>
                     <Percentage>15</Percentage>
               </AccountingDistribution>
         < /RequisitionAccountingDistribution>
         < RequisitionAccountingDistribution actionCode="ADD">
               <Identification>
                     <ID>3</ID>
               </Identification>
               <AccountingDistribution>
                     <Percentage>15</Percentage>
               </AccountingDistribution>
         < /RequisitionAccountingDistribution>
   </RequisitionLine>
   <RequisitionLine actionCode="DELETE"> Indicates this line has been deleted
         <Identification>
               <ID>2</ID>
         </Identification>
   </RequisitionLine>
   <RequisitionLine actionCode="UPDATE">
         <Identification>
               <ID>3</ID>
         </Identification>
<ItemReference>
<Identification>
                     <ID>1001</ID>
               </Identification>
</ItemReference>
   </RequisitionLine>
   </RequisitionLine>
   <RequisitionLine actionCode="ADD"> Indicates this line has been added
         <Identification>
               <ID>4</ID>
         </Identification>
<ItemReference>
<Identification>
                     <ID>1005</ID>
               </Identification>
</ItemReference>
   </RequisitionLine>
</UpdateRequisition>
```

Content Payload

The payload of an operation that uses an Update verb is typically the entire EBO and in general every business object has only two messages with an Update operation (single and list).

Situations may occur in which subsets of an EBO must be updated, in which case multiple update messages may possibly exist, each with a distinct payload. An example is a possible UpdateSalesOrderLineEBM message with a payload that contains only SalesOrderLine.

Verb Attributes

The Update verb has an optional ResponseCode attribute that is intended to communicate the payload that is expected in the response message to the Update request. The possible values for the ResponseCode are restricted to either *ID* (response payload is expected to be the Identifier of the object that was created) or *OBJECT* (response payload is expected to be the entire object that was created).

Corresponding Response Verb

The Update verb has a paired **UpdateResponse** verb that is used in the response message for each Update EBM. The response is expected to be provided by the target application only if the original request message explicitly requests a response by setting the ResponseCode attribute in the Update message. The payload of the response is either the ID or the entire object that was updated, depending on the ResponseCode specified in the Update request.

Response Content Payload

The payload type of the response verb is always based on the payload type of the Update Request operation. It is implemented as a different type to support user customization if needed.

15.8.3 Delete

The **Delete** verb is a request to a service provider to delete the business object identified using the object Identification provided in the payload of the Delete message.

When to Use the Delete Verb

The Delete verb is used for operations that are intended to delete a business object.

Operations that use the Delete verb *must* delete content and *should not* be an orchestration of other operations.

> **Note:** Currently, AIA does not support using **Delete** for components of a business object, that is, Delete Purchase Order Line is allowed

Content Payload

The payload of the Delete verb must be only an identification element that uniquely identifies the business object to be deleted.

Verb Attributes

The Delete verb has an optional ResponseCode attribute that is intended to communicate the payload that is expected in the response message to the Update

request. The only possible value for the ResponseCode for Delete is *ID* (response payload is expected to be the Identifier of the object that was created).

Corresponding Response Verb

The Delete verb has a paired **DeleteResponse** verb that is used in the response message for each Delete EBM. The Response is expected to be provided by the target application only if the original request message explicitly requests a response by setting the ResponseCode attribute in the Delete message. The only allowed value for the ResponseCode is *ID*.

15.8.4 Sync

The **Sync** verb indicates a request to a service provider to synchronize information about an object using the payload provided in the Sync message.

When to Use the Sync Verb

The Sync verb is used in situations in which applications provide operations that can accept the payload of the operation and create or update business objects as necessary.

Operations that use the Sync verb *must* create or update content and *should not* be an orchestration of other operations.

The primary usage scenario for Sync is generally batch transactions in which the current state of an object is known, but what has changed between the previous sync and the current sync is not known. Sync can also be used to synchronize individual instances of objects. The initiator of Sync is generally the system that owns the data to be synchronized.

Using the Sync operation implies that the object exists in both the source and the target system, and the result of Sync is that both the source and target have the same content. Sync is different from the other verbs in that it assumes a dual processing instruction-Sync can both create and update existing content.

The content of the Sync reflects the current state of the object in the system generating the message. In this mode, a single Sync message can contain multiple instances of nouns, with none of them having any specific change indicator. The source system generates the message, and all systems that subscribe to the message are expected to synchronize their data to reflect the message content.

Sync is probably the most practical approach for master data management scenarios in which change is not frequent, and it may not be practical or necessary from an operational point of view to synchronize data on a real-time basis.

The Sync verb has an optional *syncActionCode* attribute that can be used to further instruct the recipient of a Sync message about the expected processing behavior. The possible values for the syncActionCode are:

- *CREATE_REPLACE*: This is the default behavior of Sync when no syncActionCode is specified. The target system that receives a Sync message with no syncActionCode attribute, or with a syncActionCode attribute value of NULL or CREATE_REPLACE, is expected to create the object if it does not exist in the target system, or if it does exist, the entire object is to be replaced with the definition that has been provided in the Sync message.
- *CREATE_UPDATE*: A Sync message with the value of syncActionCode as CREATE_UPDATE is expected to be processed as follows: create the object if it does not exist in the target system, or if it does exist, update the object with the content that has been provided in the Sync message.

Content Payload

Generally speaking, there is only one Sync message per EBO (with a single and list implementation) and the payload of the message is the entire EBO.

Sync should always be used to synchronize the entire business object. Multiple Sync messages may exist in cases in which different named views of the business object exist, but never for synchronizing a specific component of a business object.

> **Tip:** Unlike the OAGIS implementation of Sync, Oracle AIA has opted not to have specific attributes in Sync to indicate Add, Change, Replace, and Delete. The Enterprise Object Library (EOL) implementation of Sync is a verb that is intended to change the target object to exactly what is specified in the message payload. Sync cannot be used for deleting content-an explicit delete operation must be used in this case.

Verb Attributes

The Sync verb has an optional ResponseCode attribute that is intended to communicate the payload that is expected in the response message to the Sync request. The possible values for the ResponseCode for Sync is restricted to *OBJECT* (response payload is expected to be the entire object that was created or updated) and *ID* (response is only the ID of the object that was created or updated)

Corresponding Response Verb

The Sync verb has a paired **SyncResponse** verb that is used in the response message for each Sync EBM. The response is expected to be provided by the target application only if the original request message explicitly requests a response by setting the ResponseCode attribute in the Sync message.

> **Note:** The design intent is to avoid having two verbs with the same objective. The Sync verb also supports the creation of an object, but is intended for use primarily in the scenario in which the object exists in both the source and the target systems. That is, the semantics of usage of Sync as a verb communicates to the recipient application the fact that the object being synchronized exists in both the source and target, whereas the usage of Create or Update is intended to communicate the fact that the object being created or updated does not exist in the source system.

Response Verb Content Payload

The payload type of the response verb is always based on the payload type of the Sync Request operation. It is implemented as a different type to support user customization if needed.

15.8.5 Validate

The **Validate** verb is a request to a service provider to validate the content provided in the payload of the message. It is used for operations that are intended to verify the validity of data.

When to Use the Validate Verb

Operations that use the Validate verb *do not* create or update business objects, and can internally be implemented as an orchestration of other operations. For example,

validating a purchase order for approval may involve validating whether a budget is available, if the supplier is still active, if the requisitions that need the items on the line are still valid, and so on.

Content Payload

The payload of any operation that falls in the Validate category can be a business object, a business component of a business object, or any other user-defined payload.

Verb Attributes

Not applicable.

Corresponding Response Verb

The Validate verb has a paired **ValidateResponse** verb that is used in the response message for each Validate EBM. The Validate verb is always implemented synchronously.

Response Content Payload

The response payload of a Validate operation is user-definable.

15.8.6 Process

The **Process** verb is a request to a service provider to perform the corresponding operation on the business object associated with the service. It is generally used for operations that orchestrate multiple other operations, are intended to achieve a specific business result, or both.

When to Use the Process Verb

Process is used as a single verb to categorize all business operations that result in content updates but do not fall in the Create/Update/Delete category to avoid a proliferation of verbs for specific business object actions.

Operations that use the Process verb *must* always result in creation or updating of one or more business objects and may represent an orchestration of other operations.

The Process verb can also be used for operations that act on a single business object, but have specific business semantics that cannot be communicated using an Update operation. In general, such actions are implemented in applications with distinct access control, and specific business rules that are applicable when the action is performed. Examples of such actions are state changes, updating meter readings on equipment, and so on.

Because multiple operations can be performed by the Process verb, potentially multiple Process verbs can be used in any given EBS.

> **Tip:** Operations that implement the Process verb can be implemented as synchronous or asynchronous. This is a deviation from the other verbs for which a consistent implementation pattern applies across all operations that use them.

Content Payload

The nature of the operation performed by a Process verb may require properties and values that are not part of the business object on which the operation is being performed, but are required by the business rules that are implemented by the operation. For example, approval of a sales order can record a comment as part of the approval, but the comment in itself may not be a property of the sales order.

In general, the request and response payload of operations that use the Process verb need the ability to reflect the method signature of the application, without a restriction that the content that forms the payload *must* come from the business object to which the service operation is associated.

To support the preceding, the payload of each Process operation is not restricted to content from the EBO definition. This is unlike all the other EBOs for which the EBM business content must be the same as or a subset of the EBO content.

> **Note:** Currently, AIA does not support assembling Process EBM payloads using content from other business components. So a Process operation for credit verification that is defined for a customer party cannot include content from Sales Order EBO to build its message payload.

The List pattern used in the other generic verbs is also applicable here, but does not apply generically; that is, in an EBS, both a single and List implementation of a Process operation may exist, or just one or the other. Unlike the other generic verbs for which single and list are both consistently created for all EBOs, Process is driven by the requirements of the corresponding operation.

Verb Attributes

Not applicable.

Corresponding Response Verb

The Process verb has a paired **ProcessResponse** verb that is used in the response message for each Process EBM.

Response Verb Payload

The payload of the response verb is specific to each process operation and is determined by the objective of the operation. Similar to the Process verb payload, there is no restriction that the content of the response is restricted to the content of the EBO.

15.8.7 Query

The **Query** verb is a request to a service provider to perform a query on a business object using the content provided in the payload of the Query message, and return the query result using the corresponding response message associated with the query. The requester can optionally request a specific subset of the response payload.

When to Use the Query Verb

Similar to the other verbs, the Single and the List pattern apply to queries also. The use of Query for each of these patterns has been listed separately in the following sections.

> **Tip:** The same verb applies to both the patterns, but the implementation and attributes applicable are completely different.

Single Object Query Intended to Return One and Only One Instance

The Single Object Query operation is a simple get by ID operation that enables callers to look up an EBO by its identifier. It is intended to request a single instance of a business object by specifying the ID of the object and optionally a QueryCode and ResponseCode with a set of parameters and their values. The identifier of the object is specified in the DataArea of the Query EBM.

The single object query does not support any other query criteria to minimize the possibility of the query returning multiple objects, and the response payload for the simple query is restricted to a single instance of the object being queried.

The Single Object Query contains the following elements:

- QueryCode within the Query element (optional)

  The QueryCode in a single object Query is used mainly as a supplement to the Identification element provided in the DataArea of the Query. The Query Code can be used for a single object query for cases in which there is a need to communicate more than the ID to the query service provider to successfully run the query. The code could be used to either refine the query, or to select the object to be queried based on the Query Code.

  > **Tip:** For this to happen, the service provider should implement the processing of such a code. Currently, Oracle AIA does not predefine any generic Query Codes as part of the EOL.

- ResponseCode within the Query Element (optional)

  The ResponseCode is a predefined code that instructs a query service provider to filter the response content of the EBO to a specific subset of the response object.

  The return payload for a generic Query is always the entire EBO; that is, by default, the response payload for a QuerySalesOrder operation is always the entire SalesOrder with lines, shipment, and so on. If the requester wants the service provider to provide only the SalesOrder header with none of the child components, then the ResponseCode can be used as an instruction to the service provider to build the response payload accordingly.

  > **Tip:** For this to happen, the service provider should implement the processing of such a code. Currently, Oracle AIA does not predefine any generic Request or Response Codes as part of the EOL.

Content Payload

The payload of a single object query is always the ID of the object to be queried. This is specified within the Identification element of the DataArea as shown in Example 15–13.

Example 15–13 Content Payload of a Single Object Query

```
<QueryAccountBalanceAdjustmentEBM>
<corecom:EBMHeader>

   </corecom:EBMHeader>
   <DataArea>
         <Query>
         </Query>
         <QueryAccountBalanceAdjustment>
               <corecom:Identification>
                     <corecom:ID>1005</corecom:ID>

               </corecom:Identification>
               <Custom/>
         </QueryAccountBalanceAdjustment>
   </DataArea>
</QueryAccountBalanceAdjustmentEBM>
```

Verb Attributes

The simple Query can have an optional getAllTranslationsIndicator to indicate whether the service provider is expected to provide all translations to be populated in the response for all translatable elements. The default is to bring back data in the language of the request only.

Based on the preceding, there are ways in which in which a simple query can be constructed:

- **Simple Query with just ID**: An example of querying for a single object would be querying Purchase Order with ID="3006". No other code or parameters are needed in this example.
- **Simple Query with QueryCode:** As an example, consider an application that maintains a distinction between a person as a customer versus an organization as a customer. A single Customer Query service exists, but to successfully run the query, the service provider must be told whether the ID to be queried belongs to an organization or to a person. In this case, the QueryCode can be used to communicate the Person/Organization information.

List Query That Can Return Multiple Instances

The single object query does not support any search criteria beyond a simple search by identification, and can return only one instance of an object. All other queries are treated as List queries.

A List Query may return multiple records in response to the query and supports the ability to build complex queries.

The List Query is implemented using the following elements:

QueryCode within the Query element (optional)

The QueryCode in a List Query serves as a means for a service provider to limit the possible queries that can be built using a List Query. In the absence of a QueryCode, a service provider should be able to generically support all possible queries that can be communicated using the QueryCriteria element explained subsequently.

For this to happen, the service provider should implement the processing of such a code. Currently, Oracle AIA does not predefine any generic Query Codes as part of the EOL.

ResponseCode within the Query Element (optional)

The ResponseCode is a predefined code that instructs a query service provider to filter the response content of the EBO to a specific subset of the response object. For list queries, this can serve as an alternate mechanism instead of specifying the ResponseFilter element of a Query.

The return payload for a generic Query is always the entire EBO. For example, by default, the response payload for a QuerySalesOrder operation is always the entire SalesOrder with lines, shipment, and so on. If the requester wants the service provider to provide only the SalesOrder header with none of the child components, then the ResponseCode can be used as an instruction to the service provider to build the response payload accordingly.

For this to happen, the service provider should implement the processing of such a code. Currently, Oracle AIA does not predefine any generic Request or Response Codes as part of the EOL.

QueryCriteria (1 to n instances)

The QueryCriteria element enables a user to build a complex query statement *on a specific node* of the object being queried. At least one QueryCriteria element must be specified for a List Query.

Multiple QueryCriteria elements can be present in a Query *if the query spans multiple nodes* of the object being queried. Multiple Query criteria is similar to a subselect in that the result set of one Query Criteria is filtered by the second to arrive at a smaller subset of records.

Each QueryCriteria element consists of:

- **QualifiedElementPath (0 or 1 instance):** This enables the user to specify the node on which the QueryCriteria applies. If the element is not included in the Query, or if it is included with no value or a NULL value, or if it has a value of /, the query criteria applies to the root element of the object.

  QueryExpression (exactly 1 instance, with optional nesting of other QueryExpressions within it): This element is the container for the actual query. The QueryExpression is a nested construct that enables you to define complex queries. Each QueryExpression consists:

  - A **logicalOperatorCode attribute** that can have a value of either AND or OR. These attributes can be specified to indicate the logical operation to be performed on the content (nested multiple QueryExpressions *or* list of ValueExpressions) within the QueryExpression.
  - A choice of one or more QueryExpressions or one or more ValueExpressions.
    * A QueryExpression may contain other QueryExpressions when you must combine multiple AND or OR operations in a query.
    * If the Query can be expressed with a single AND or OR operator, then it can be built using multiple ValueExpressions within an outer QueryExpression.
- **ValueExpression (1 or more instances):** Each ValueExpression represents an assignment of a value to a specific node within the node represented by the QualifiedElementPath (or the root node if the QualifiedElementPath is not present). The ValueExpression is specified using:
  - An **ElementPath** element that represents either a node (expressed as a simpleXPath expression) to which the query value is being assigned, or a Code in case the element cannot be found in the document. For example, /SalesOrderLine/Status/Code or just StatusCode. No explicit way is available to indicate whether the ElementPath contains a code or an Xpath expression.
  - **Value** element that contains the value assigned to the ElementPath, for example, Approved.
  - A **queryOperatorCode** attribute that specifies the operator applicable to the Value assigned to the ElementPath. The possible operators are EQUALS, NOT_EQUALS, GREATER_THAN, GREATER_THAN_EQUALS, LESS_THAN, LESS_THAN_EQUALS, CONTAINS, DOES_NOT_CONTAIN, LIKE, NOT_LIKE, LIKE_IGNORE_CASE, NOT_LIKE_IGNORE_CASE, IS_BLANK, IS_NOT_BLANK, BETWEEN, NOT_BETWEEN, IN, NOT_IN
- **SortElement (0 or more instances):** Each QueryCriteria can have one or more SortElements defined. A SortElement is used to request the Query result set to be sorted using the criteria specified in the SortElement. Each SortElement has an optional sortDirectionCode attribute that identifies the order of sorting-ASC (ascending) or DESC (descending).

Example 15–14 Defining SortElements for Single QueryCriteria

```
<QueryCriteria>
<QueryExpression>
         <ValueExpression>

         </ValueExpression>
   </QueryExpression>
   <SortElement sortDirection="DESC">/OrderDateTime</SortElement>
   <SortElement sortDirection="ASC">/Description</SortElement>
</QueryCriteria>
```

In Example 15–14, the result set of the QueryExpression is to be sorted in descending order of OrderDateTime and ascending order of Description.

If multiple QueryCriteria exist, then each can have its own SortElement. The result set of one QueryCriteria is sorted, and then after the next QueryCriteria filter is applied, the resultant subset is sorted using the SortElement specified for that QueryCriteria.

Example 15–15 Defining SortElements for Multiple QueryCriteria

```
<QueryCriteria>
<QueryExpression>
         <ValueExpression>

         </ValueExpression>
   </QueryExpression>
   <SortElement sortDirection="DESC">/OrderDateTime</SortElement>
   <SortElement sortDirection="ASC">/Description</SortElement>
</QueryCriteria>
<QueryCriteria>
   <QualifiedElementPath>/SalesOrderLine/SalesOrderLineBase</QualifiedElementPath>
   <QueryExpression>
   ...
   </QueryExpression>
   <SortElement>/SalesOrderLine/SalesOrderLineBase/ListPrice</SortElement>
</QueryCriteria>
```

In Example 15–15, the SalesOrder root query is sorted by OrderDateTime and Description, while the child node SalesOrderLine is sorted by price.

QueryCriteria Examples

Example 1 Query with a Single QueryCriteria Element and Single Query Expression

Example 15–16 is an example of a query with a single QueryCriteria element and a single QueryExpression element that contains only ValueExpression elements. The query to be run is Query SalesOrder, in which "SalesOrder/CurrencyCode = "USD" **AND** "SalesOrder/OrderDateTime = "2003-12-04". The query expression is defined for the root node; hence, the QualifiedElementPath is not present (optional).

This is modeled as two ValueExpressions nested within a QueryExpression. The logicalOperatorCode on the QueryExpression indicates the operation **(AND)** to be performed across the two ValueExpressions.

Example 15–16 Example of Query with Single QueryCriteria Element and Single QueryExpression

```
<Query>
<QueryCriteria>
         <QueryExpression logicalOperatorCode="AND">
               <ValueExpression queryOperatorCode="EQUALS">
                     <ElementPath>/SalesOrderBase/CurrencyCode</ElementPath>
                     <Value>USD</Value>
               </ValueExpression>
               <ValueExpression queryOperatorCode="GREATER_THAN_EQUALS">
                     <ElementPath>/SalesOrderBase/OrderDateTime</ElementPath>
                     <Value>2003-12-04</Value>
               </ValueExpression>
         </QueryExpression>
   </QueryCriteria>
</Query>
```

Example 2 Query with a Single QueryCriteria and Nested QueryExpressions

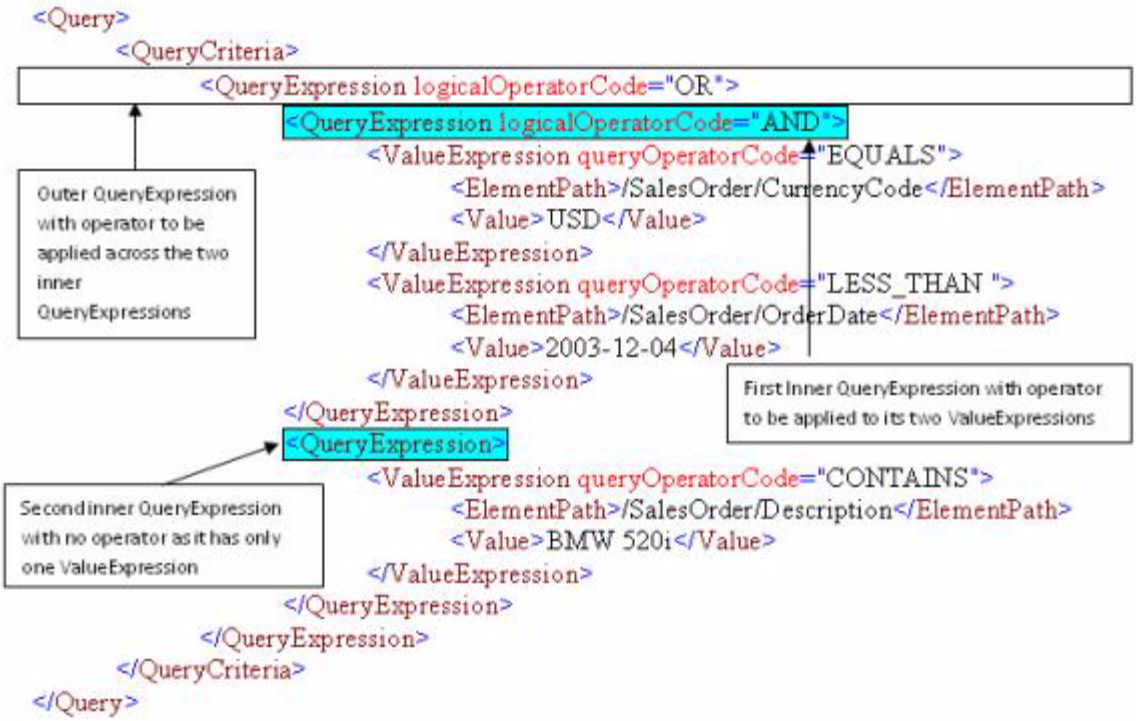
Figure 15–13 is an example of a Query with a single QueryCriteria but with nested QueryExpressions. The query to be run is Query SalesOrders, in which SalesOrder/CurrencyCode=USD **AND** SalesOrder/OrderDateTime<2003-12-04 **OR** SalesOrder/Description **CONTAINS** "BMW 520i". Both the query expressions are defined for the root node; hence, the QualifiedElementPath is not present (optional).

Note: When nested QueryExpressions exist, they are all on the same QualifiedElementPath.

This is modeled as two QueryExpressions nested within an outer QueryExpression.

The outer QueryExpression identifies the operand to be applied to the two inner QueryExpressions (OR). The ValueExpressions contain the actual query data with the query operator to be applied to the data element as shown in Figure 15–13.

Figure 15–13 Example of Query with Nested Query Expressions



Example 3 Query with Multiple QueryCriteria

Example 15–17 is an example of a Query with multiple QueryCriteria. When multiple QueryCriteria exist, no logical operation is present that is applicable across them-they are the equivalent of running the query specified in the first QueryCriteria element, then applying the second QueryCriteria to the result of the first.

The query to be executed is: Query CustomerParty where Type = GOLD and filter the result set to only accounts whose status is "ACTIVE".

This is implemented as two QueryCriteria. The first is to query the CustomerParty and retrieve all Customers of TYPE-"GOLD". This query is run on the CustomerParty root node.

The result set of this query is then filtered by applying the second Query Criteria. This queries the CustomerParty/Account node to get all CustomerParty Accounts that have STATUS = "ACTIVE".

Example 15–17 Example of Query with Multiple QueryCriteria

```
<Query>
<QueryCriteria>
        <QueryExpression>
              <ValueExpression queryOperatorCode="EQUALS">
                    <ElementPath>/Type</ElementPath>
                    <Value>GOLD</Value>
              </ValueExpression>
        </QueryExpression>
        <SortElement>/LastName</SortElement>
   </QueryCriteria>
   <QueryCriteria>
        <QualifiedElementPath>/CustomerAccount</QualifiedElementPath>
```

```
        <QueryExpression>
              <ValueExpression queryOperatorCode="EQUALS">
                    <ElementPath>/CustomerAccount/Status/Code</ElementPath>
                    <Value>ACTIVE</Value>
              </ValueExpression>
        </QueryExpression>
   </QueryCriteria>
</Query>
```

ResponseFilter (0 to 1 instance)

The ResponseFilter enables a requester to indicate which child nodes he or she is interested or not interested in. The excluded child nodes are not populated by the application when the response to the query is being built. If supported by participating applications, this feature improves performance by not querying the nodes that are excluded from the query.

Example 1 Request for Single Return to QueryInvoice Message

Example 15–18 illustrates requesting only the InvoiceLine to be returned in response to a QueryInvoice message.

Example 15–18 Requesting a Single Return to QueryInvoice Message

```
<Query>
<QueryCriteria>

   </QueryCriteria>
        <ResponseFilter>
              <ChildComponentPath>InvoiceLine</ChildComponentPath>
        </ResponseFilter>
</Query>
```

Example 2 Request for Specific Message Return to QueryInvoice Message

Example 15–19 is an example of returning all QueryInvoice message content except for Charge and PaymentTerm.

Example 15–19 Requesting Specific Message Return to QueryInvoice Message

```
<Query>
<QueryCriteria>

   </QueryCriteria>
   <ResponseFilter>
        <ExclusionIndicator>true</ExclusionIndicator>
        <ElementPath>Charge</ElementPath>
        <ElementPath>PaymentTerm</ElementPath>
   </ResponseFilter>
</Query>
```

Tip: In the absence of a very comprehensive framework for processing queries, this option is difficult to implement practically, because in theory infinite ways are available by which the query can be built, and the service provider would not be able to process all possible ways in which the QueryCriteria is specified.

Content Payload

No content payload for a List Query is available-the query is defined entirely within the Verb element of the DataArea.

Verb Attributes

- **getAllTranslationsIndicator(optional):** A Query can have an optional getAllTranslationsIndicator to indicate whether the service provider is expected to provide all translations to be populated in the response for all translatable elements. The default is to bring back data in the language of the request only.
- **recordSetStart(optional):** This is an instruction to the query service provider to return a subset of records from a query result set, with the index of the first record being the number specified in this attribute.

  As an example, consider a query that returns 200 records in the result set. The requester can invoke the query with a recordSetStart parameter set to "101", in which case the service provider is expected to process this value and build a result set that contains the 101st to the 200th record of the result set.
- **recordSetCount(optional):** The presence of this attribute is an instruction to the service provider to return the same attribute with the count of the number of records in the response to the Query. Absence of this attribute indicates that a record set count is not expected in the response to the query.
- **maxItems(optional):** This is an instruction to the query service provider that the maximum number of records returned in the query response should not exceed the value specified for this attribute.

  The result set of a query may result in multiple records that meet the query criteria, but the query service Requester may be able to process only a specific number of records at a time. For example, a query might result in a result set of a 1000 records, but the query Requester can process only 100 records at a time. The service Requester can use the maxItems attribute to instruct the service provider to return only 100 records in the response.

Corresponding Response Verb

The Query Verb has a paired **QueryResponse** verb that is used in the response message for each Query EBM.

Response Verb Payload

The payload of the response verb is typically the entire business object.

15.9 Invoking the ABCS

This section assumes that you have completed the ABCS construction. The different ways in which your ABCS can be invoked are discussed here.

An ABCS can be invoked by an application or by another service. The service, if it happens to be an AIA artifact, could be either a transport adapter or an EBS. This section describes what must be done in each of the three scenarios.

This section includes the following topics:

- Section 15.9.1, "How to Invoke an ABCS Directly from an Application"
- Section 15.9.2, "How to Invoke an ABCS Using Transport Adapters"
- Section 15.9.3, "When Does an Enterprise Business Service Invoke an ABCS"

15.9.1 How to Invoke an ABCS Directly from an Application

For the inbound interactions of applications with the ABCS:

1. Start with identifying the participating applications.
2. Analyze the integration capabilities of each participating application.
3. Work with the application providers (developers) to decide the best way to interact with the application to achieve the needed functionality.

   The interaction mechanism can be one of the following:

   - SOAP WebService
   - Events/Queues
   - JCA
4. Obtain relevant details from the applications in situations in which the interactions between the participating applications and the ABCS happen through a message queue adapter or a JCA Adapter.

   These details are used in configuring the adapters. The configuration results in the WSDL creation. Care must be taken to ensure that the environment-specific details are configured in relevant resource files on the server and not directly in the WSDLs.

15.9.2 How to Invoke an ABCS Using Transport Adapters

When the Requester ABCS is invoked by the transport adapters, the interaction between ABCS and the transport adapters is using either SOAP Web Service or native binding.

More details are specified in the section Section 16.3.1, "Interfacing with Transport Adapters".

The details on establishing the inbound connectivity between adapters and the ABCS are given in Chapter 23, "Establishing Resource Connectivity."

15.9.3 When Does an Enterprise Business Service Invoke an ABCS

An EBS invokes:

- A provider ABCS, which implements an EBS operation. This is true because an EBS provides the mediation between the requesting services and providing services.
- A requester ABCS to send the response to it. In scenarios in which a requesting service is waiting for a delayed response, the delayed response from the providing service can be routed by a Response EBS to the waiting Requester ABCS.

16 Completing ABCS Development

This chapter describes how to develop Application Business Connector Services (ABCS), handle errors and faults, work with adapters, develop ABCS for CAVS enablement, secure ABCS, enable transactions, enable message delivery and version ABCS, enable resequencing in Oracle Mediator and Layered Customizations.

This chapter includes the following sections:

16.1 Developing Extensible ABCS

An ABCS, regardless of whether it is requester or provider specific, can invoke custom code a minimum of either two or four times during its execution. These serve as extensibility points.

The ABCS supporting request-response pattern in either synchronous or asynchronous mode has four extensibility points. An ABCS supporting fire-and-forget patterns has two extensibility points. You can develop "add-ins" and have them hooked to these extensibility points. These "add-ins" - customer-developed services- behave as an extension to the delivered ABCS. Each extension point allows one hook so only a single customer extension can be plugged in.

This section describes the mechanism for creating ABCSs with extensible services, which enables you to have service implementations that require no modifications to the delivered ABCS.

16.1.1 Introduction to Enabling Requester ABCS for Extension

For request/response Requester ABCS, you can hook your custom code to four extensibility points:

1. Just before the execution of transformation of application business message (ABM) to Enterprise Business Message (EBM). Use this configuration property name: ABCSExtension.PreXformABMtoEBM.
2. Just before the invocation of the enterprise business service (EBS). Use this configuration property name: ABCSExtension.PreInvokeEBS.
3. Just before the execution of transformation of EBM to ABM and after invoking the EBS. Use this configuration property name: ABCSExtension.PostInvokeEBS.
4. Just before the invocation of callback service or response return and transforming EBM to ABM. Use this configuration property name: ABCSExtension.PostXformEBMtoABM.

The third and fourth extension points are available only in ABCS implementing request-response pattern.

For Fire and Forget Requester ABCS, you can hook your custom code to two extensibility points:

1. Just before the execution of transformation of application business message (ABM) to EBM. Use this configuration property name: ABCSExtension.PreXformABMtoEBM.
2. Just before the invocation of the enterprise business service (EBS). Use this configuration property name: ABCSExtension.PreInvokeEBS.

For more information about configuration parameters, see section 15.1.3.1.

Figure 16–1 depicts the high-level flow of activities in a requester-specific ABCS. The diagram assumes that the EBS with which it is interacting employs a request-response interaction style. The steps for executing the customer extension to do additional tasks are optional.

Figure 16–1 Extending the Request-Response Interaction Style

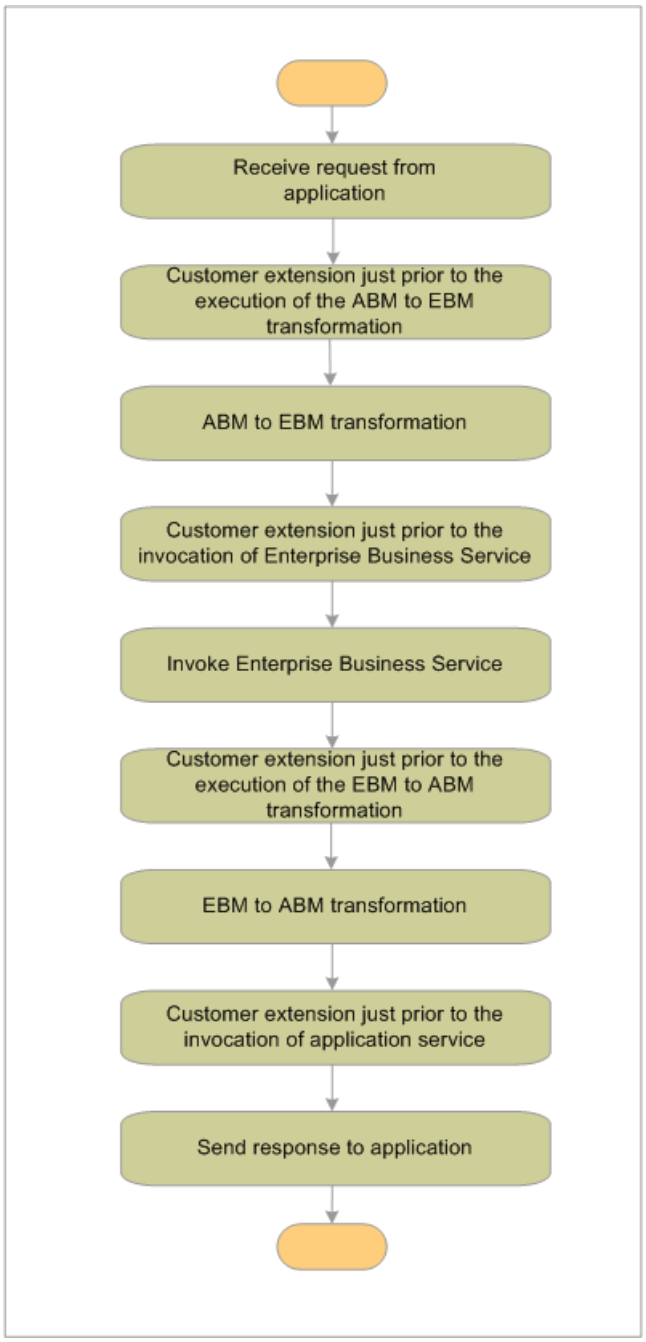
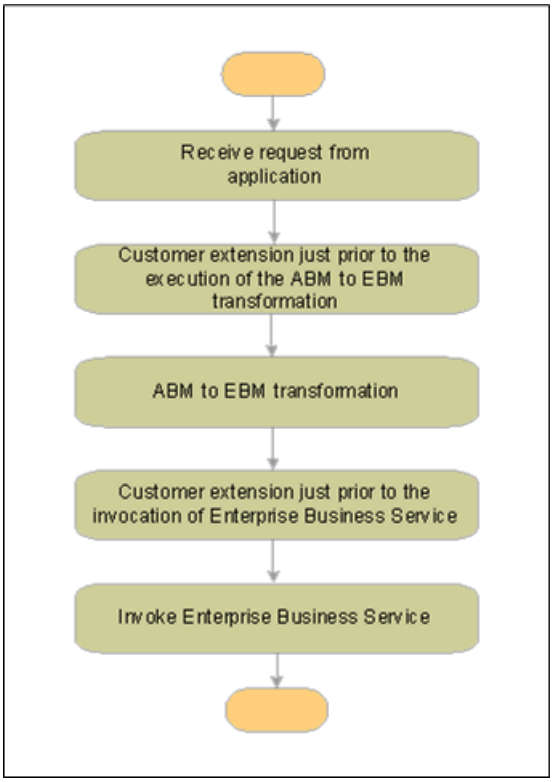


Figure 16–2 shows the high-level flow of activities in a requester-specific ABCS. The diagram assumes that the EBS with which it is interacting employs a fire-and-forget interaction style. The steps for executing the customer extension to do additional tasks are optional.

Figure 16–2 Requester-Specific ABCS Using Fire-and-Forget Interaction Style



The first extensibility point made available to the implementers of the requester ABCS can be used to perform custom message inspection. This extensibility point can be used to inject code to perform tasks such as custom validation, message alteration, message filtering and so on. The custom code has access to the ABM that is about to be transformed and can return either an enhanced ABM or raise a fault.

The second extensibility point can be used to perform custom message augmentation. The extensibility point can be used to inject code to perform tasks such as EBM enhancement, custom validation and so on. The custom code has access to EBM that is about to be used for invocation of EBS and can return either an enhanced EBM or raise a fault.

The third extensibility point can be used to inject code to perform tasks such as custom validation, message alteration, message filtering and so on. The custom code has access to EBM that is about to be transformed to ABM. The custom code can return either an altered EBM or raise a fault.

The fourth extensibility point can be used to inject code to perform tasks such as ABM enhancement, custom validation and so on. The custom code has access to ABM that is about to be used for invocation of callback services. The custom code can return either an altered ABM or raise a fault

16.1.2 Introduction to Enabling Provider ABCS for Extension

For request/response Requester ABCS, you can hook your custom code to four extensibility points:

1. Customer extension just before the execution of the EBM to ABM transformation. Use this configuration property name: ABCSExtension.PreXformEBMtoABM.

2. Customer extension just before the invocation of application service. Use this configuration property name: ABCSExtension.PreInvokeABS.
3. Customer extension just before the execution of the ABM to EBM transformation and after invoking Application Service. Use this configuration property name: ABCSExtension.PostInvokeABS.
4. Customer extension just before the invocation of Enterprise Business Service and after transforming Application Message to Enterprise Business Message. Use this configuration property name: ABCSExtension.PostXformABMtoEBM.

The third and fourth extension points are available only in the ABCS implementing request-response pattern.

For Fire and Forget Requester ABCS, you can hook your custom code to two extensibility points:

1. Customer extension just before the execution of the EBM to ABM transformation. Use this configuration property name: ABCSExtension.PreXformEBMtoABM.
2. Customer extension just before the invocation of application service. Use this configuration property name: ABCSExtension.PreInvokeABS.

For more information about configuration parameters, see section 15.1.3.1.

Figure 16–3 depicts the high-level flow of activities in a provider-specific ABCS. The diagram assumes that the EBS with which it is interacting employs a request-response interaction style.

Figure 16–3 Provider-Specific ABCS Using Request-Response Interaction Style

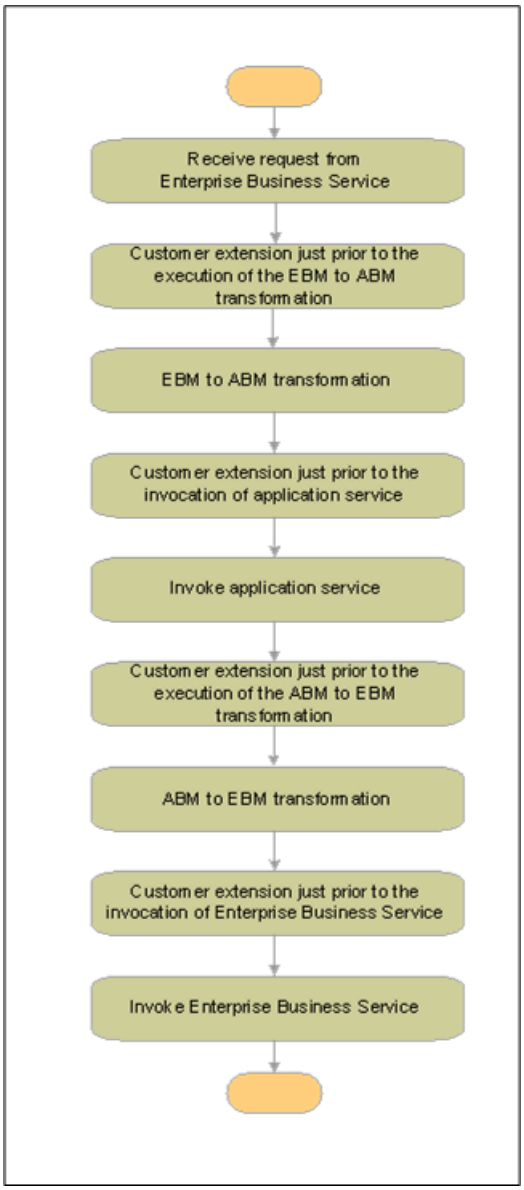
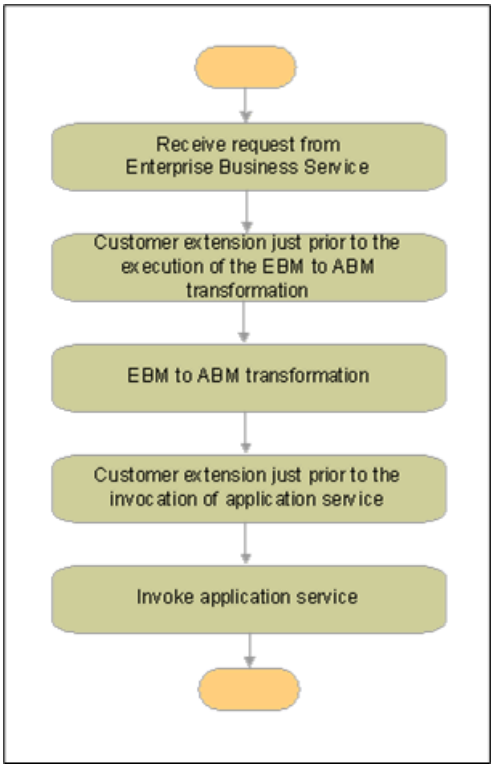


Figure 16–4 depicts the high-level flow of activities in a provider-specific ABCS. The diagram assumes that the EBS with which it is interacting employs a fire-and-forget interaction style.

Figure 16–4 Provider-Specific ABCS Using Fire-and-Forget Interaction Style



The first extensibility point made available to the implementers of the Provider ABCS can be used to inject code to perform tasks such as custom validation, message alteration, message filtering and so on. The custom code has access to EBM that is about to be transformed. The custom code can return either an enhanced EBM or raise a fault.

The second extensibility point can be used to perform custom message augmentation. The extensibility point can be used to inject code to perform tasks such as ABM enhancement, custom validation and so on. The custom code has access to ABM that is about to be used for invocation of application service. The custom code can return either an enhanced ABM or raise a fault.

The third extensibility point can be used to inject code to perform tasks such as custom validation, message alteration, message filtering and so on. The custom code has access to ABM that is about to be transformed to EBM. The custom code can return either an altered ABM or raise a fault.

The fourth extensibility point can be used to perform custom message augmentation. The extensibility point can be used to inject code to perform tasks such as EBM enhancement, custom validation and so on. The custom code has access to EBM that is about to be used for invocation of callback services. The custom code can return either an altered EBM or raise a fault.

16.1.3 How to Design Extensions-Aware ABCS

Each of the extensibility points is modeled as a service operation having a well-defined interface. ABCS authors define these interfaces. The extensibility interfaces consist of service operations that the ABCS invokes to execute the custom message enrichment or transformation or validations specific code implemented by the customer.

Each ABCS is accompanied by a corresponding customer extension service. A request-response ABCS has four extensibility points, therefore, the ABCS extension service has four service operations. For a fire-and-forget ABCS, the corresponding ABCS extension service has two service operations.

As delivered, the implementation of these service operations for all of the ABCS extension services invokes the same piece of code that always returns the same message. This piece of code has been implemented as a *Servlet*.

The ABCS is developed to invoke the appropriate service operation at each of the extensibility points. To minimize overhead, a check is made to ensure that the service for the relevant extensibility interface has been implemented. Oracle AIA Configuration properties have one property for each extensibility point. Setting the property to 'Yes' indicates that there is a custom implementation for the extensibility point. The default value for these properties is *No*, therefore, the ABCS never invokes the implementations of these extensions as they were delivered.

Table 16–1 lists the service operations for the requester ABCS-specific extensibility points:

Table 16–1 Service Operations for Requester ABCS-Specific Extensibility Points

Extensibility Point	Service Operation Name
Just before the execution of transformation of ABM to EBM	Pre-ProcessABM
Just before the invocation of the EBS	Pre-ProcessEBM
Just before the execution of transformation of EBM to ABM	Post-ProcessEBM
Just before the invocation of callback service or response return	Post-ProcessEBM

Table 16–2 lists the service operations for the provider ABCS-specific extensibility points:

Table 16–2 Service Operations for Provider ABCS-Specific Extensibility Points

Extensibility Point	Service Operation Name
Just before the execution of transformation of EBM to ABM	Pre-ProcessEBM
Just before the invocation of Application Service	Pre-ProcessABM
Just before the execution of transformation of ABM to EBM	Post-ProcessABM
Just before the invocation of callback EBS or response return	Post-ProcessEBM

AIA recommends that the ABCS and that the customer extension services be co-located. In SOA 11g, when the services are deployed on the same server, the SOA 11g run time uses native invocation.

AIA recommends that the extension service should be enlisted as part of the ABCS transaction. When the extension service is implemented using SOA 11g technology, then set the transaction property bpel.config.transaction to 'required' on the extension service to enlist itself in the ABCS transaction.

For more information, see Section 16.6.1, "How to Ensure Transactions in AIA Services".

16.1.3.1 Configuration Parameters

Operations at the extension points are invoked based on the values of the specific parameters in the file AIAConfigurationProperties.xml. Each service configuration section specific to an ABCS requires that these parameters be specified. The parameters for each of the four extension points are:

- ABCSExtension.PreXformABMtoEBM
- ABCSExtension.PreInvokeEBS
- ABCSExtension.PostInvokeEBS
- ABCSExtension.PostXformEBMtoABM

- ABCSExtension.PreXformEBMtoABM
- ABCSExtension.PreInvokeABS
- ABCSExtension.PostInvokeABS
- ABCSExtension.PostXformABMtoEBM

These parameters must be configured with value '*true*' for a service invocation. When these parameters are not specified in the configuration section for an ABCS, the value considered by default is '*false*'.

The intent of this section is to provide developers with a set of guidelines on how to create and leverage these configuration properties to optimize extensibility. The service operation names and extension configuration properties provided in this section are recommendations only

Also, it is not within the scope of this discussion to provide guidelines for extension points for every one of the invocations. The number of parameters for any particular ABCS could be different. Product management and engineering teams are best equipped to identify the additional extension points where additional extensibility is required.

IT organizations developing services that could potentially be extended by the departmental teams or by their partners will find a need for ABCS to made extension aware. Given this use case, if a customer has to develop ABCS with additional extension points than the recommended, they can come up with appropriate names by following the appropriate naming conventions for service operation names and configuration properties similar to the currently proposed properties.

16.1.4 Designing an ABCS Composite with Extension

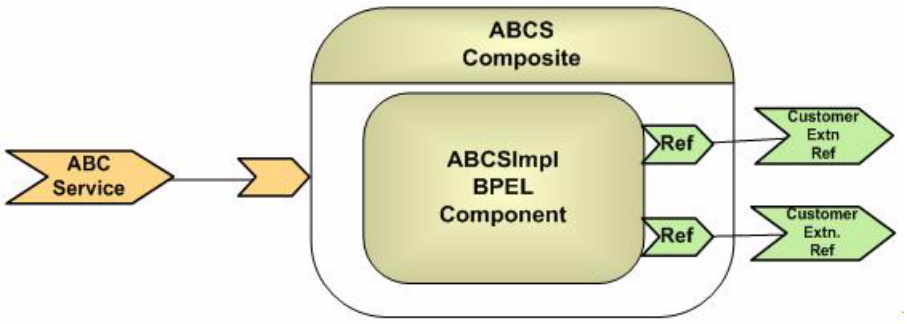
When an ABCS composite is developed as an extension-enabled service:

- The ABCS is implemented as a component (using BPEL technology) in the composite.
- The extension service is referred to by the ABCS composite as an external Web service.

The service at the extension point is developed in a separate composite if it is developed over an Oracle Fusion Middleware platform.

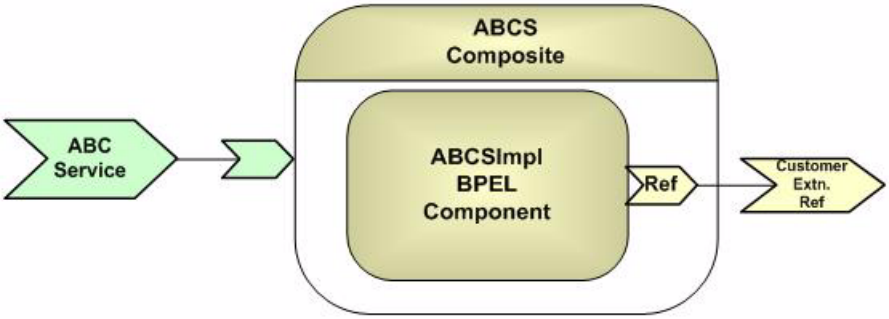
When different external extension services are referred to by the ABCS composite, multiple hooks exist from the ABCS composite to the External Service. That is, when an ABCS invokes two different services at two of its extension points, the composite must be designed to have two distinct external references, as shown in Figure 16–5:

Figure 16–5 Example of Composite with Two Distinct External References



However, when an ABCS composite invokes two distinct operations that are exposed on the **same** external service, then the ABCS composite has only one external reference, as shown in Figure 16–6:

Figure 16–6 Example of Composite with One External Reference



16.1.5 Defining Service at Extension Points

At the extension points, partner links are defined to set up the conversational relationship between two services by specifying the roles played by each service in the conversation and specifying the port type provided by each service to receive messages within the context of the conversation.

This is accomplished with the help of a Web Services Description Language (WSDL) file that describes the services the partner link offers. The WSDL is an XML document that describes all the Service Provider contracts to the service consumers.

The WSDL separates the service contract into two distinct parts, the Abstract WSDL and the Concrete WSDL:

- The **Abstract WSDL** includes elements such as types, messages that define the structure of the parameters (input and output types), fault types, and the port type, which is known as the interface.
- The **Concrete WSDL** includes bindings to protocols, concrete address locations, and service elements.

16.1.6 Defining a Service Using an Abstract WSDL

An abstract WSDL defines the external reference service at design time. This WSDL provides interfaces to the operations to be invoked at the extension points. It should also include or import the schemas for the application business object (ABO) and Enterprise Business Object (EBO). It is devoid of communication transport protocols, network address locations, and so on.

The abstract WSDL used to define the extension point service should be placed in the project folder. Do not push this into MDS, unlike other abstract WSDLs of the AIA services.

Note: An abstract WSDL may be used temporarily for design-time modeling only. At the time the composite is deployed, the binding should be specified.

When AIA Service Constructor is used to construct ABCS, it creates the extension service references in the composite.xml file.

When you use JDeveloper, follow these high-level steps to design the composite:

To design the composite to extension-enable ABCS:

1. In JDeveloper, open a SOA composite project.
2. Open the composite.xml in design mode.
3. To reference an extension point service, add a Web service as an external reference service in the References swim lane. Use the Abstract WSDL for design time modeling.

   Tip: This abstract WDSL will not be in the MDS. It is associated with the project and is in the project folder.

4. Wire the BPEL component to the external reference component created in the previous step.

   When the composite is opened in the source mode, you see code similar to the Example 16–1.

   This code example was reproduced from the composite.xml, where the extension service is defined using an abstract WSDL:

Example 16–1 Wiring the BPEL component to the external reference component

```
<reference name="SamplesCreateCustomerSiebelReqABCSImplExtExtension"
ui:wsdlLocation="SamplesCreateCustomerSiebelReqABCSImplExtExtensionAbstract.wsdl">
        <interface.wsdl
interface="http://xmlns.oracle.com/ABCSImpl/Siebel/Samples/CreateCustomerSiebelR
eqABCSImplExtExtension/V1#wsdl.interface(SamplesCreateCustomerSiebelReqABCSImpl
ExtExtensionService)"/>
        <binding.ws port="" location=""/>
    </reference>
```

16.1.7 How to Specify a Concrete WSDL at Deployment Time

A concrete WSDL is a copy of the abstract WSDL used for defining the extension service. The concrete WSDL also defines the binding element that provides information about the transport protocol and the service element that combines all ports and provides an endpoint for the consumer to interact with the service provider.

Initially, at the time of development, the concrete WSDL points to the sample extension service, which is a Servlet. The Servlet is named MirrorServlet and is shipped with the Foundation Pack.

You should push the concrete WSDL into the MDS repository to the folder 'ExtensionServiceLibrary'.

16.1.7.1 Populating the binding.ws Element in the composite.xml

To invoke the external reference service, a run-time WSDL with concrete bindings must be specified in the ABCS composite.xml. The Port type in the WSDL should have a concrete binding. That is, in the composite, the attributes of the element, *binding.ws*, cannot be empty.

> **Note:** Populating the 'location' attribute of the element binding.ws with the URL of a run-timeWSDL, does not amount to violating the principle of 'designing the service using only abstract WSDLs'. The reason is that this concrete WSDL is not accessed either at composite's design time or at composite's deploy time. This WSDL is only accessed at composite's run time.

To deploy the composite that references an external Web service, which is defined using an abstract WSDL, the attributes of the element *binding.ws* must be populated, as shown below:

```
<binding.ws port="[namespace of the ABCS Extension Service as
defined in the WSDL]/V1#wsdl.endpoint(<Name of the ABCS Service
as given in the WSDL>/<Name of the Porttype as given in the
WSDL>" location="[location of the concrete WSDL in the MDS]"
xmlns:ns="http://xmlns.oracle.com/sca/1.0"/>
```

The name of the ABCS Service is the value of the attribute definitions/name in the abstract WSDL.

This follows from naming conventions for the Service name in the ABCS composite. According to naming conventions, the name of the service is <name of the composite>, which in turn is the value of the attribute 'name' of the 'definitions' element in the WSDL.

At the time when the service at the extension point is developed and deployed by the customer,

- Customer can replace the sample concrete WSDL in the MDS with the concrete WSDL that is developed for the extension service and redeploy the ABCS.
- Customer can change binding.ws.location to point to the concrete WSDL of the deployed service and redeploy the ABCS.

For more information, see Section 2.1.3.4, "Using MDS in AIA".

16.1.8 Designing Extension Points in the ABCS BPEL Process

The following section details the steps to be completed to provide extension points in a Requester ABCS with a one-way invocation call.

In a Requester ABCS with a request-only call to a service, two possible extension points exist. The first extension point is just before the transformation from ABM to the EBM, and second extension point is just before the invocation of the service. The service operations at these extension points are defined as Pre-ProcessABM and

Pre-ProcessEBM, respectively. The service operation at the extension points is an invocation to Servlet that only returns the payload.

16.1.9 How to Set Up the Extension Point Pre-ProcessABM

When AIA Service Constructor is used to construct ABCS, the extension point is set up by the tool.

The Requester ABCS requires the following process activities in the order specified:

1. Switch

   The Switch activity determines if the service invocation at the extension point is to be made or not. This is achieved by checking the value of the property variable in the file AIAConfigurationProperties.xml. The name of the configuration parameter is 'ABCSExtension.PreProcessABM '. Check this parameter for a value of 'true'. This is achieved by using an appropriate 'AIAXPathFunction' in the switch expression.

2. Assign-Invoke-Assign

   The Assign-Invoke-Assign subsequence of process activities is embedded in the Switch activity.

   - Assign

     The Assign activity assigns the value of the input variable of the 'receive' step to the input variable of the 'invoke' step that follows.

   - Invoke

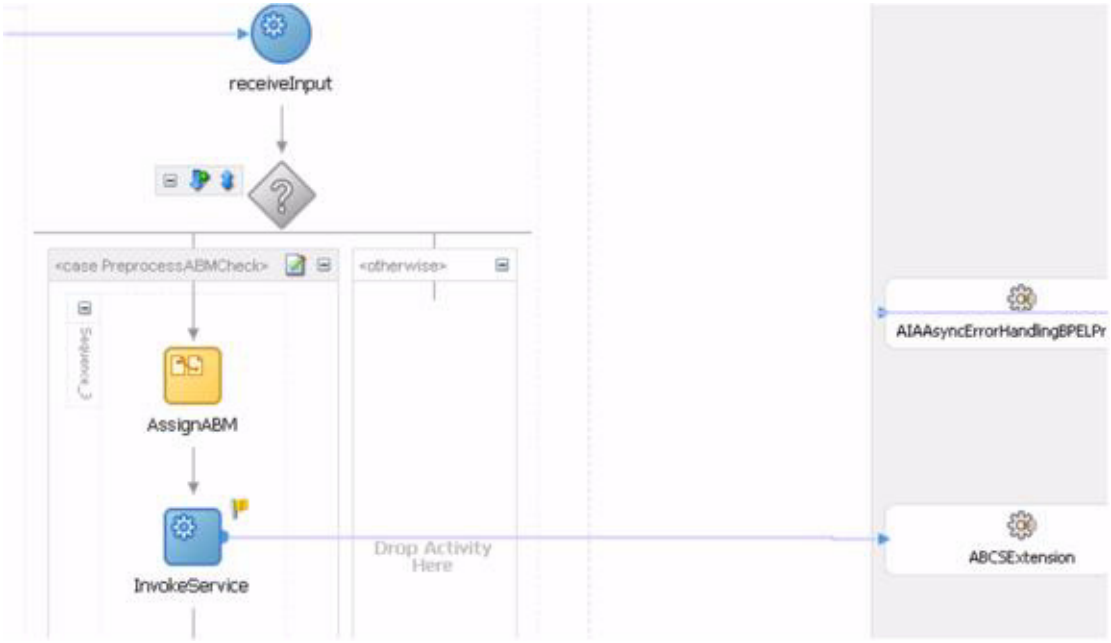
     The Invoke activity makes the service-call on the partner-link. The programming model for the ABCS extension points sets up a contract on the service to return the same payload that it receives.

   - Assign

     The Assign activity assigns the value of the response payload from the service invocation to the variable of the process activity that follows.

   Figure 16–7 illustrates the sequence:

Figure 16–7 Setting up the Extension Point Pre-ProcessABM



16.1.10 How to Set Up the Extension Point Pre-ProcessEBM

When AIA Service Constructor is used to construct ABCS, the extension point is set up by the tool.

The Requester ABCS requires the following process activities in the order specified:

1. Switch

   The Switch activity determines if the service invocation at the extension point is to be made or not. This is achieved by checking the value of the property variable in the file AIAConfigurationProperties.xml. The name of the configuration parameter is 'ABCSExtension.PreProcessEBM'. Check the parameter for a value of 'true'. This is achieved by using an appropriate 'AIAXPathFunction' in the switch expression.

2. Assign-Invoke-Assign

   The Assign-Invoke-Assign subsequence of process activities is embedded in the Switch activity.

   - Assign

     The Assign activity assigns the value of EBM to the input variable of the 'invoke' step that follows.

   - Invoke

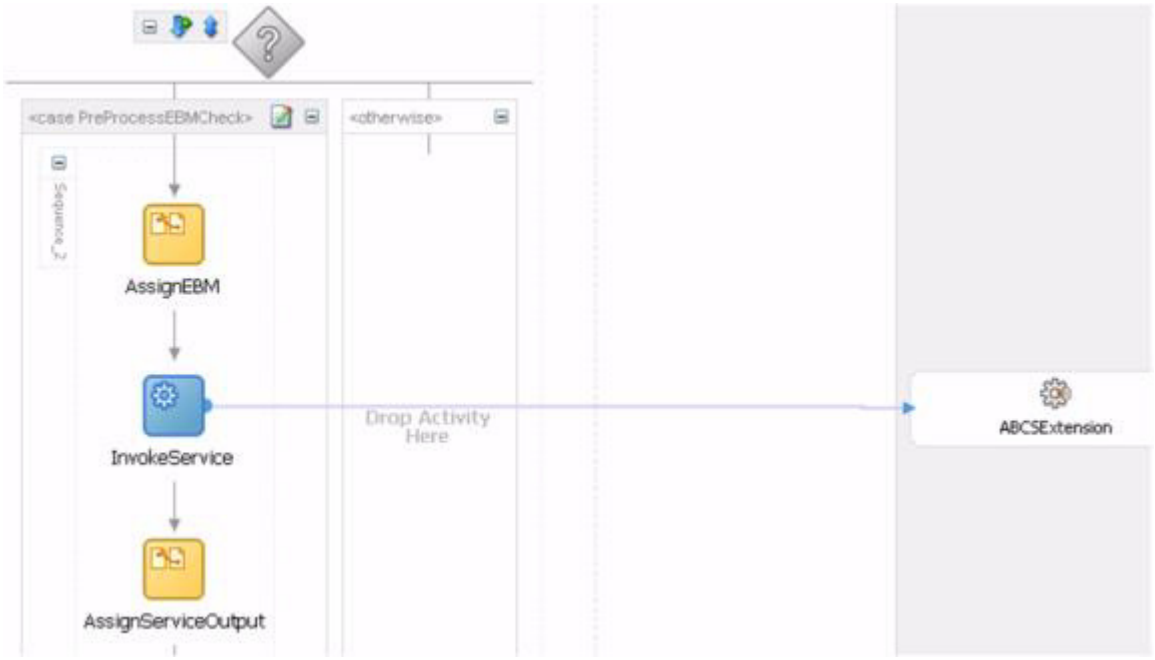
     The Invoke activity makes the service-call on the partner-link. The programming model for the ABCS extension points sets up a contract on the service to return the same payload that it receives.

   - Assign

     The Assign activity assigns the value of the response payload from the service invocation to variable of the process activity that follows.

   Figure 16–8 illustrates the sequence:

Figure 16–8 Setting up the Extension Point Pre-process EBM



16.1.11 How to Test the Extensibility with Servlet as Sample Extension Service

The sample extension service used in the document is a java servlet that mirrors the input payload back. The Servlet (shown in Example 16–2) as the endpoint has the advantage that it can be used as a partner-link service to test any extension. This is useful when you must test any extension-aware ABCS that have PartnerLink services defined using Abstract WSDL.

Example 16–2 Servlet to test extension

```
<service name="CreateCustomerSiebelReqABCSImplExt">
<port name="CreateCustomerSiebelReqABCSImplExt"
binding="tns:CreateCustomerSiebelReqABCSImplExt_Binding">
               <soap:address
location="http://{host}:{port}/MirrorServlet/mirror"/>
          </port>
    </service>
```

The implementations for the methods PreProcessABM are not required; the SOAP request is treated as the Servlet payload, and the Servlet outputs back the entire payload. The SOAPAction element is also rendered a dummy since it is not used at the Servlet-side.

The test Servlet is a java file with name Mirror.java. When deployed, it is accessible at: http://[hostname].com:[portno]/Mirror/mirror

16.2 Handling Errors and Faults

To determine how faults are handled and passed by a participating application, you must make sure the application error handling capabilities are in line with the integration platform's error handling capabilities.

16.2.1 How to Handle Errors and Faults

- In synchronous request-response message exchange patterns (MEPs), the requesting services are waiting for response.

  Whenever an error occurs in the provider services, an exception is raised and the fault message is propagated back to the requesting service.

- In asynchronous fire-and-forget MEPs, the requesting service does not expect a response.

  If an error occurs in the providing service, compensation may be needed. In such situations, the compensatory operations in EBS must be used for triggering compensations.

- In asynchronous request-delayed response MEPs, the requesting service is in a suspended mode, waiting for a response.

  If an error occurs in the providing service, the response to the requesting service includes details about the error.

  For details on implementing the ABCS and EBS services in this scenario, see Section 15.5, "Implementing the Asynchronous Request Delayed Response MEP."

  For more information about implementing error handling in BPEL and Mediator processes in each of the MEPs, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging".

- In an asynchronous MEP, the message initiated from a sender is persisted until it is successfully delivered to and acknowledged by the receiver, if an acknowledgment is expected.

  The sender and receiver are not necessarily the participating applications. Rather, they can be logical milestones in an Oracle AIA integration scenario. Each persistence store represents a milestone and may be a database, file system, or JMS persistence. Multiple milestones may be configured in an integration scenario to ensure the movement of messages from one persistence point to another.

  For more information about configuring milestones for Guaranteed Message Delivery in an Integration flow, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging".

16.3 Working with Adapters

This section discusses working with transport and version adapters.

This section includes the following topics:

16.3.1 Interfacing with Transport Adapters

Use transport adapters to interface with the ABCS in these scenarios:

- For packaged applications, such as Siebel, PeopleSoft, J.D. Edwards, and SAP, the preferred route is to use the respective packaged-application adapters. These

adapters can be deployed as JCA resource adapters. This solution is better than using the conventional SOAP interface.

- In situations for which the participating applications do not expose their business logic as Web services, interactions with these applications must occur using technology adapters such as database adapters, JMS adapters, and so forth.

Transport adapters allow connectivity to the systems/applications that were not originally developed using Web services technologies. Some examples of applications that can use adapters are:

- Systems that use non-xml for communication
- Packaged software
- Database systems
- Data sources or persistent stores such as JMS, and so on

Investigate whether the services exposed by the participating applications provide support for proprietary message formats, technologies, and standards. If the applications that implement the functionality do not have inherent support for standards and technologies such as XML, SOAP, and JMS, then the transformations must happen in the ABCS.

For example, the application might be able to receive and send messages only using files, and EDI is the only format it recognizes. In this case, the ABCS is responsible for integrating with the application using a file adapter, translating the EDI-based message into XML format, exposing the message as a SOAP message.

When to Use Adapters for Message Aggregation

In some situations, you must combine responses to a request that originated from multiple sources. For example, for convergent billing in the telecommunication solution, the Application Business Connector Service for the getBillDetails EBS might have to retrieve details from multiple participating applications.

For more information about the Message Aggregation design pattern, see Chapter 27, "Working with AIA Design Patterns".

When to Use Adapters for Event Aggregation

The Event Aggregation model provides a comprehensive methodology for the business use case in which events, entity, or message aggregation is needed. In such scenarios, multiple events are raised before the completion of a business message, and all such fine-grained message events are consolidated into a single coarse-grained event.

In such use cases, a requester ABCS is invoked by an event consumer adapter service, which feeds the requester ABCS with an aggregated event message.

For more information about the event aggregation design pattern, see Chapter 27, "Working with AIA Design Patterns".

16.3.2 How to Develop Transport Adapters

Here are the high-level steps from the ABCS perspective.

To develop JMS Consumer Adapter:

1. Add JMS adapter in the Exposed Services swim lane.
2. Configure the JMS adapter service with the help of Adapter Configuration wizard.

3. Add a BPEL component in the Components swim lane.
4. Wire the BPEL component to the JMS adapter service.
5. Wire the BPEL component to the Referenced Service.
6. Open the BPEL component in design mode and add 'invoke activity' to invoke the JMS adapter partner link.
7. Complete the coding for the BPEL process.

For more information about JMS Adapters, see *Oracle Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server.*

To develop Portal DB Adapter:

1. Add a BPEL component in the Components swim lane.
2. Add DB adapter as external reference service in the References swim lane.
3. Configure the adapter service with the help of Adapter Configuration wizard.
4. Wire the BPEL component to the db adapter service.
5. Open the BPEL component in design mode and add 'invoke activity' to invoke the db adapter partner link.
6. Complete the coding for the BPEL process.

16.3.3 When to Put Adapters in a Single Composite

In principle, an ABCS composite is a component implemented along with other components and wires between those components. For example, you can implement a Requester ABCS composite using:

- A BPEL process component representing the ABCS process flow.
- BPEL or Mediator-based adapter components representing the adapters used or required by the process component.
- Both the process component and the adapter components promoted as *Services* of the composite in which they are defined.

An ABCS and a TransportAdapter service can be in the same composite and when they are, the composite name is the same as that of the ABCS. Alternatively, you can develop an ABCS service and a TransportAdapter service as separate composites.

AIA recommends that you put adapters that are interfaced with ABCS in a different composite from that of ABCS when the same transport adapter service could be used with multiple ABCSs.

16.3.4 Planning Version Adapters

When service providers release a new version of an application service, you must have multiple versions of it running concurrently when the consumer code is migrated.

The version adapter allows the client request and response to consume a different release of a service and routes requests to the appropriate service endpoint based on the content.

Examples:

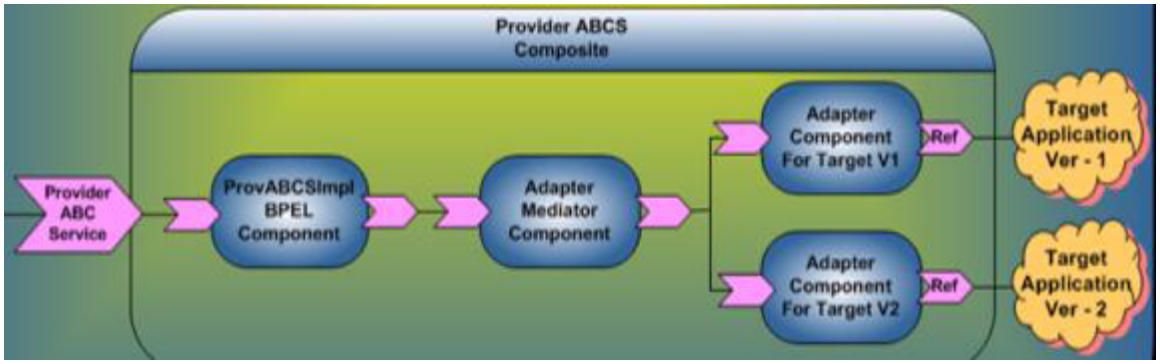
- For assets, the view definition being used in integration is different between Oracle 11.5.10 and R12.

- For payment authorization, the API name is different in Oracle 11.5.10 and R12.

Changes such as these between different versions of applications require a new adapter service to be created for each application version.

When the changes between different versions of applications are minor, introduce a version adapter, based on a Mediator component, between the existing connector service and the provider application, as shown in Figure 16–9:

Figure 16–9 Introducing a Version Adapter



This approach ensures that the connector service remains agnostic of the version of the applications. This option should be considered when no transformations are needed or when input and output transformations in the version adapter are simple and do not involve extensive logic affecting the performance.

16.3.5 How to Configure a Version Adapter

To configure a version adapter:

The version adapter service is a mediator-based component that sits between the provider ABCS implementation service and the actual participating applications.

The mediator-based version adapter service should have references to all the application adapters:

1. Configure the routing rules in the version adapter to route to corresponding adapter services of the applications.
2. For each of the connecting application's adapters, two routing rules should exist to enable both content-based routing and property-based (from AIAConfiguration file) routing.
3. When the changes between different application versions are minor with regard to the content to be passed and the content is available in the transformed ABM, use a transformation map to transform the input of the version adapter to the corresponding adapter service of the provider application.
4. Map the response from the different adapters to the schema that the connector service is expecting.

16.4 Developing ABCS for CAVS Enablement

This section provides instructions on how to develop Oracle AIA services that are ready to work with the Composite Application Validation System (CAVS). This section includes the following topics:

- Section 16.4.1, "How to CAVS Enable Provider ABCS"
- Section 16.4.2, "How to CAVS Enable the Requester ABCS"
- Section 16.4.3, "Introduction to the CAVSEndpointURL Value Designation"
- Section 16.4.4, "Purging CAVS-Related Cross-Reference Entries to Enable Rerunning of Test Scenarios"

> **Tip:** CAVS configurations are only required when simulators are a part of the testing scenario. These configurations lay the foundation for allowing services to route messages to simulators.

For more information about using the CAVS, see *Oracle Application Integration Architecture Foundation Pack: Infrastructure Components and Utilities Guide*, "Working with the CAVS."

16.4.1 How to CAVS Enable Provider ABCS

Provider ABCSs that are asynchronous and invoke a callback to a ResponseEBS service must also CAVS-enable that invocation.

Developers must populate a value for the property: Routing.<PartnerLinkName>.<TargetSystemID>.EndpointURI as shown below.

You must provide the above property in the service configuration file if the value for the property **Routing.<PartnerLink>.RouteToCAVS** is given as *'false'*.

To code a provider ABCS for dynamic CAVS-enabled PartnerLinks for invoking target participating application Web services:

1. Define the following service-level configuration properties for the provider ABCS as shown in Example 16–3:

Example 16–3 Defining service-level configuration properties for the provider ABCS

```
<Property name="Default.SystemID">[DefaultTargetSystemID]</Property>
<Property name="Routing.[PartnerlinkName].RouteToCAVS">[true|false]</Property>
<Property
name="Routing.[PartnerlinkName].[TargetSystemID].EndpointURI">[AppEndpointURL]<
/Property>
<Property
name="Routing.[PartnerlinkName].CAVS.EndpointURI">[CAVSEndpointURL]</Property>
```

   The CAVSEndPointURL value is set at design time.

   For more information about the design-time designation of the CAVSEndPointURL, see Section 16.4.3, "Introduction to the CAVSEndpointURL Value Designation".

2. Ensure that you have the following namespace prefixes defined in your BPEL process as shown in Example 16–4:

Example 16–4 Defining namespace prefixes in the BPEL process

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsa=http://schemas.xmlsoap.org/ws/2003/03/addressing
xmlns:corecom=http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2
```

3. Add this variable to your global variables section:

```
<variable name="SystemID" type="xsd:string"/>
```

4. Add an attachment named *AddTargetSystemID.xsl* to your BPEL project in the 'xsl' directory. Be sure to replace the **[ABCServiceNamespace]** and **[ABCServiceName]** tokens in the file appropriately.
5. Add the assignment shown in Example 16–5 as the first step in the BPEL process. Be sure to replace the tokens appropriately:

Example 16–5 Adding an assignment

```
<assign name="GetTargetSystemID">
    <copy>
        <from expression="ora:processXSLT('AddTargetSystemID.xsl',
bpws:getVariableData('[RequestEBMVariableName]', '[RequestEBMPartName]'))"/>
        <to variable="[RequestEBMVariableName]"
            part="[RequestEBMPartName]"/>
    </copy>
    <copy>
        <from variable="[RequestEBMVariableName]"
              part="[RequestEBMPartName]"
              query="/[NamespacePrefixedEBMName]/corecom:EBMHeader/corecom:
               Target/corecom:ID"/>
        <to variable="SystemID"/>
    </copy>
</assign>
```

6. Add the following <scope> as shown in Example 16–6 **once** for each PartnerLink before its invoke activity:
    a. Replace the tokens in the AssignDynamicPartnerlinkVariables assignment activity to set the variables appropriately.
    b. Update the PartnerLink name token in the AssignPartnerlinkEndpointReference assignment activity.
    c. Updating the embedded Java code should not be necessary.

Example 16–6 Adding <scope> for each PartnerLink

```
<scope name="SetDynamicPartnerlinkScope">
    <variables>
        <variable name="TargetEndpointLocation" type="xsd:string"/>
        <variable name="EndpointReference" element="wsa:EndpointReference"/>
        <variable name="ServiceName" type="xsd:string"/>
        <variable name="PartnerLinkName" type="xsd:string"/>
        <variable name="EBMMsgBpelVariableName" type="xsd:string"/>
        <variable name="EBMMsgPartName" type="xsd:string"/>
    </variables>
    <sequence name="SetDynamicPartnerlinkSequence">
        <assign name="AssignDynamicPartnerlinkVariables">
            <copy>
```

```
                <from expression="'{[ABCServiceNamespace]}[ABCServiceName]
                 '"/>
                <to variable="ServiceName"/>
            </copy>
            <copy>
                <from expression="'[PartnerlinkName]'"/>
                <to variable="PartnerLinkName"/>
            </copy>
            <copy>
                <from expression="'[RequestEBMVariableName]'"/>
                <to variable="EBMMsgBpelVariableName"/>
            </copy>
            <copy>
                <from expression="'[RequestEBMPartName]'"/>
                <to variable="EBMMsgPartName"/>
            </copy>
        </assign>
        <bpelx:exec name="GetTargetEndpointLocation" language="java"
                    version="1.5">
            <![CDATA[/*------------------------------------------------------
This code snippet will derive the dynamic endpoint URI for a partnerlink.
------------------------------------------------------------------*/

java.lang.String serviceName = (java.lang.String)getVariableData("ServiceName");
java.lang.String partnerLinkName =
(java.lang.String)getVariableData("PartnerLinkName");
java.lang.String cavsEndpointPropertyName =
"Routing."+partnerLinkName+".CAVS.EndpointURI";
java.lang.String ebmMsgBpelVariableName =
(java.lang.String)getVariableData("EBMMsgBpelVariableName");
java.lang.String ebmMsgPartName =
(java.lang.String)getVariableData("EBMMsgPartName");
java.lang.String systemIdBpelVariableName = "SystemID";
java.lang.String targetEndpointLocationBpelVariableName =
"TargetEndpointLocation";
java.lang.String routeToCavsPropertyName =
"Routing."+partnerLinkName+".RouteToCAVS";
java.lang.String defaultSystemIdPropertyName = "Default.SystemID";
java.lang.String targetEndpointLocation = null;
java.lang.String targetID = null;
boolean addAudits = false;

if (addAudits) addAuditTrailEntry("Partnerlink = " + partnerLinkName);

// check configuration for CAVS routing flag
try {
    boolean routeToCAVS =
java.lang.Boolean.parseBoolean(oracle.apps.aia.core.config.Configuration.
getServiceProperty(serviceName, routeToCavsPropertyName));
    if (addAudits) addAuditTrailEntry("RouteToCAVS = " + routeToCAVS);
    if (routeToCAVS) {
        targetEndpointLocation = oracle.apps.aia.core.config.Configuration.
         getServiceProperty(serviceName, cavsEndpointPropertyName);
        if (addAudits) addAuditTrailEntry("Endpoint = '" + targetEndpoint
         Location + "' selected from configuration property " +
cavsEndpointPropertyName);
    }
}
catch (oracle.apps.aia.core.config.PropertyNotFoundException e) {
    if (addAudits) addAuditTrailEntry("Configuration property " + cavs
```

```
EndpointPropertyName + " not found!");
}

if (targetEndpointLocation==null || targetEndpointLocation=="") {

    // check bpel variable for already retrieved target system Id
    try {
        targetID = (java.lang.String)getVariableData(systemIdBpel
         VariableName);
        if (addAudits && targetID!=null) addAuditTrailEntry("Using previously
stored Target System ID = '" + targetID + "'");
    }
    catch (com.oracle.bpel.client.BPELFault e) {
    }

    if (targetID==null || targetID=="") {
        // try to get Target system ID from EBM Header
        try {
oracle.xml.parser.v2.XMLElement targetIdElement =
 (oracle.xml.parser.v2.XMLElement)
getVariableData(ebmMsgBpelVariableName,
ebmMsgPartName, "/*/corecom:EBMHeader[1]/corecom:Target/corecom:ID
[text()!='']");
            targetID = targetIdElement.getText();
            if (addAudits) addAuditTrailEntry("Target System ID = '" + targetID
+ "', selected from EBM header");
        }
        catch (com.oracle.bpel.client.BPELFault e) {
            if (addAudits) addAuditTrailEntry("Unable to retrieve Target System
ID from message header");
        }
        try {
            if (targetID!=null && targetID!="")
setVariableData(systemIdBpelVariableName, targetID);
        }
        catch (com.oracle.bpel.client.BPELFault e) {
        }
    }

    if (targetID==null || targetID=="") {
        // try to get Target system ID from configuration
        try {
            targetID = oracle.apps.aia.core.config.Configuration.getService
             Property(serviceName, defaultSystemIdPropertyName);
            if (addAudits) addAuditTrailEntry("Target System ID = '" + targetID
+ "', selected from configuration property " + defaultSystemIdPropertyName);
        }
        catch (oracle.apps.aia.core.config.PropertyNotFoundException e) {
            if (addAudits) addAuditTrailEntry("Configuration property "
+ defaultSystemIdPropertyName + " not found!");
        }
        try {
            if (targetID!=null && targetID!="") setVariableData(systemIdBpel
             VariableName, targetID);
        }
        catch (com.oracle.bpel.client.BPELFault e) {
        }
    }

    if (targetID!=null || targetID!="") {
```

```
        // try to get EndpointLocation from Configuration
        java.lang.String endpointPropertyName = "Routing."+partnerLinkName+
         "."+targetID+".EndpointURI";
        try {
            targetEndpointLocation =
oracle.apps.aia.core.config.Configuration.getServiceProperty(serviceName,
endpointPropertyName);
            if (addAudits) addAuditTrailEntry("Endpoint = '" +
targetEndpointLocation + "' selected from configuration property " +
endpointPropertyName);
        }
        catch (oracle.apps.aia.core.config.PropertyNotFoundException e) {
            if (addAudits) addAuditTrailEntry("Configuration property " +
endpointPropertyName + " not found!");
        }
    }
}

try {
    setVariableData(targetEndpointLocationBpelVariableName, targetEndpoint
     Location);
}
catch (com.oracle.bpel.client.BPELFault e) {
}]]>
        </bpelx:exec>
        <switch name="Switch_SetEndpoint">
            <case condition="string-length(bpws:getVariableData('Target
             EndpointLocation'))>0">
                <assign name="AssignPartnerlinkEndpointReference">
                    <copy>
                        <from>
                            <wsa:EndpointReference xmlns:wsa="http://schemas.
                             xmlsoap.org/ws/2003/03/addressing">
                                <wsa:Address/>
                            </wsa:EndpointReference>
                        </from>
                        <to variable="EndpointReference"/>
                    </copy>
                    <copy>
                        <from variable="TargetEndpointLocation"/>
                        <to variable="EndpointReference"
                            query="/wsa:EndpointReference/wsa:Address"/>
                    </copy>
                    <copy>
                        <from variable="EndpointReference"/>
                        <to partnerLink="[PartnerlinkName]"/>
                    </copy>
                </assign>
            </case>
            <otherwise>
                <empty name="Empty_NoSetEndpoint"/>
            </otherwise>
        </switch>
    </sequence>
</scope>
```

16.4.2 How to CAVS Enable the Requester ABCS

The general programming model concept is similar to how the endpoint URI is dynamically computed in the provider ABCS to achieve dynamic target invocation.

To code a requester ABCS for CAVS-enabled invocation of an Enterprise Business Service that is implemented as mediator service:

1. In the BPEL artifact, add a switch activity.
2. The condition expression in the 'case' tests for the nonzero value in the element EBMHeader/MessageProcessingInstruction/DefinitionID.
3. Populate the wsa:EndpointReference element with the value of the element EBMHeader/MessageProcessingInstruction/DefinitionID as shown in Example 16–7.

Example 16–7 Populating the wsa:EndpointReference element

```
<switch name="Switch_CAVSEnablement">
<case condition="string-length(bpws:getVariableData('CreateCustomerPartyList_
InputVariable','CreateCustomerPartyListEBM','/custpartyebo:CreateCustomerPartyLis
tEBM/corecom:EBMHeader/corecom:MessageProcessingInstruction/corecom:DefinitionID'
))>0">
                <bpelx:annotation>
                    <bpelx:pattern>Is_CAVS_DefinitionID</bpelx:pattern>
                </bpelx:annotation>
                <sequence>
                    <assign name="Assign_TargetEndpointLocation">
                        <copy>
                            <from variable="CreateCustomerPartyList_
InputVariable"
                                  part="CreateCustomerPartyListEBM"
                                  query="/
corepartyebo:CreateCustomerPartyListEBM/corecom:EBMHeader/corecom:Message
ProcessingInstruction/corecom:DefinitionID"/>
                            <to variable="TargetEndpointLocation"/>
                        </copy>
                    </assign>
                    <assign name="AssignPartnerlinkEndpointReference">
                        <copy>
                            <from>
                                <wsa:EndpointReference>
                                    <wsa:Address/>
                                </wsa:EndpointReference>
                            </from>
                            <to variable="EndpointReference"/>
                        </copy>
                        <copy>
                            <from variable="TargetEndpointLocation"/>
                            <to variable="EndpointReference"
                                query="/wsa:EndpointReference/wsa:Address"/>
                        </copy>
                        <copy>
                            <from variable="EndpointReference"/>
                            <to partnerLink="SamplesCustomerPartyEBS"/>
                        </copy>
                    </assign>
                </sequence>
            </case>
            <otherwise> <empty name="Empty_NoSetEndpoint"/> </otherwise>
```

```
</switch>
```

Tip: Some requester ABCSs must communicate back directly with the calling participating application. For this type of partnerlink, the requester ABCS acts similarly to a provider ABCS because it is invoking a participating application Web service.

16.4.3 Introduction to the CAVSEndpointURL Value Designation

The CAVSEndPointURL value is set at design time as follows:

- If the ABCS or Enterprise Business Flow (EBF) is invoking a synchronous service, then the CAVSEndPointURL value in the AIAConfigurationProperties.xml file is set to a default value of:
  http://<host>:<port>/AIAValidationSystemServlet/syncresponsesimulator
- If the ABCS or EBF is invoking an asynchronous one-way service, then the CAVSEndPointURL value in the AIAConfigurationProperties.xml file is set to a default value of:
  http://<host>:<port>/AIAValidationSystemServlet/asyncrequestrecipient
- If the ABCS or EBF is invoking an asynchronous request-delayed response service, then the CAVSEndPointURL value in the AIAConfigurationProperties.xml file is set to a default value of:
  http://<host>:<port>/AIAValidationSystemServlet/asyncresponsesimulator
- If the ABCS or EBF is invoking a Response EBS, then the CAVSEndPointURL value in the AIAConfigurationProperties.xml file is set to a default value of:
  http://<host>:<port>/AIAValidationSystemServlet/asyncresponserecipient

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

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.

This purge is accomplished as follows:

1. Pre-Built Integrations that are delivered to work with Oracle AIA Foundation Pack 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 Pre-Built Integration is delivered with cross-reference systems *CAVS_EBIZ* and *CAVS_SEBL*.
2. For every system type defined on the System - Application Registry 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. To access the System - Application Registry page, access the AIA Home Page, click Go in the Setup area,

and select the System tab. Figure 16–10 illustrates the system and CAVS-related entries.

Figure 16–10 System and CAVS-related System Entries on the System-Application Registry Page

3. When testing a provider ABCS in isolation, the EBM is 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 now contains the CAVS-specific Internal ID value set for the system on the System - Application Registry page.
5. When testing an entire flow (requester ABCS-to-EBS-to-provider ABCS), you must set the Default.SystemID property of the provider ABCS to *CAVS_<XYZ>*, where *<XYZ>* is the system.

   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.

   Access the Configuration page and click Reload to load the edit.

   You can now begin testing the entire flow.

> **Note:** If the test scenario is an entire flow that includes multiple instances of the same system, then the approach described previously does not work. In this case, data created in the cross-reference remains, making the same test case incapable of being rerun.

16.5 Securing the ABCS

The participating applications are developed during different times with different concepts and implementation of authentication and authorization. When applications are integrated, you must pass authentication and authorization information between applications.

The information related to security is exchanged between participating applications and ABCS. The AIA application security context standardizes the exchange of participating applications' authentication and authorization information between various applications so that any application can be integrated with any other application.

16.5.1 How to Secure the ABCS

You should answer the following questions:

- Does the application need the security credentials to authenticate?

  If yes, can you use a generic account or should you use the requester's credentials, as specified in the message?

- How are these credentials transmitted? Are they adopting a WS-Security scheme or a custom mechanism?

  If a custom mechanism, is it through the header or as part of the payload?

Refer to Chapter 28, "Working with Security" for details about how to:

- Transform application security context into standard format in ABCS.
- Propagate the standard security context from ABCS to ABCS through EBS and EBF.
- Transform standard security context to application security context.

16.6 Enabling Transactions

When SOA is used to encapsulate and integrate cross-enterprise workflows, multiple services may take part in transactions spanning system boundaries. End-to-end business transactions that cross application boundaries involve multiple applications. Creating robust integration solutions requires more than simply exchanging messages.

An important consideration is that the Web services participating in an end-end message flow are often asynchronous, stateless, distributed, and opaque. You must understand the mechanics of transaction management.

16.6.1 How to Ensure Transactions in AIA Services

Business requirements drive transactional requirements. Transaction considerations must start during the design phase. You cannot postpone them until construction time. You must:

- Identify all the ABCS services and the EBFs that comprise a business activity.
- Organize these activities into meaningful groups to have an overall unifying purpose.
- Indicate transaction demarcation points.

The grouping of activities provides the starting point for transaction demarcation so that the relevant business operations are performed in the context of a transaction.

In the asynchronous fire-and-forget pattern, where no milestones are configured, the requester, EBS, and provider ABCS must participate in the same global transaction, including cross-references in Oracle XA.

In an asynchronous MEP, where JMS queues are set up and the participating applications interact with the queues, the JMS consumer adapters must also participate in the global transaction, as do the requester ABCS, EBS, and provider ABCS.

In such a case, when an undergoing transaction is aborted before the message is persisted at a milestone, the message is rolled back into the JMS queue.

In other words, the transaction demarcation begins with the JMS consumer adapter picking up the message from the queue and ends when the message is delivered to the consumer application or when the message is persisted at a configured milestone.

In a SOA-based integration, long-running business transactions often involve incompatible trust domains, asynchrony, and periods of inactivity, which present challenges to traditional ACID-style transaction processing. You must predefine the transaction boundaries based on technical or business criteria. Pre-Built Integration designers/architects should divide the whole transaction into parts by setting transaction boundaries using JMS interface at logical points.

Non-retryable application services must have corresponding compensatory services.

ABCS invoking non-retryable application services should invoke compensatory services in case of roll-back.

Application services implemented using JCA adapters can leverage session management to facilitate transaction coordination.

16.6.2 Transactions in Oracle Mediator

If a transaction is present, Oracle Mediator participates in that existing transaction. If a transaction is not present, Oracle Mediator starts a transaction. In the case of sequential routing rules, all rules are executed in the same transaction and, if an error occurs, a rollback is issued.

16.6.3 Transactions in BPEL

The BPEL component, by default, creates a new transaction on the basis of a request. That is, when a BPEL process is invoked, a new transaction begins by default. If a transaction exists, then it is suspended and a new one created. To chain an execution into a single global transaction, you must set the transaction flag on the provider process (callee BPEL component).

16.6.3.1 Impact of BPEL Activities on Transaction Scope

BPEL engine ends its local transaction when it reaches the end of flow or hits a breakpoint activity, for example, receive, onMessage, onWait, and Alarm.

Transaction design considerations are also governed by:

- Invocation patterns employed across the AIA integration

  ABCS -- EBS - ABCS

- Technology pattern: BPEL -- MEDIATOR-BPEL

  ABCS - EBS - EBF - EBS -- ABCS

- Technology Pattern: BPEL - MEDIATOR- BPEL - MEDIATOR- BPEL

  Based on the transaction semantics in FMW, you must design and configure transactions across ABCS, EBS and Enterprise Business Flows.

- Participating applications

  Participating applications should be able to support transactions using their application services. However, the reality is that many application services are able to do this, so compensatory services are needed in case of rollback (orchestrated by AIA layer).

Design integration logic and transaction boundaries in the AIA layer to account specifically for the peculiarities of the participating application services, SCA Composite transaction semantics, and business requirements.

16.6.4 Developing ABCS to Participate in a Global Transaction

In SOA 11*g*, you need not set up any configuration properties on the calling (consuming) process to chain an execution into a single global transaction.

You only must set the appropriate transaction flag on the **Callee** BPEL component. You can do this in the composite editor's source by adding **bpel.config.transaction** into a BPEL component, with value as '**required**' as shown in Example 16–8:

Example 16–8 Setting the transaction flag on the callee BPEL component

```
<component name="SamplesCreateCustomerSiebelReqABCSImplProcess">
    <implementation.bpel
src="SamplesCreateCustomerSiebelReqABCSImplProcess.bpel"/>
    <property name="bpel.config.transaction">required</property>


</component>
```

With the transaction flag set as shown, BPEL inherits the transaction that is there, started by the parent process. If the parent process has not started a transaction, BPEL creates a new one. For a global transaction, the transaction is committed when the initiator commits.

If BPEL has an inbound interface with an adapter, you must set the appropriate transaction flag in the composite of the adapter.

Example 16–9 shows how transaction properties should be set on the BPEL adapter interface:

Example 16–9 Setting transaction properties on the BPEL adapter interface

```
<component name="SamplesCreateCustomerSiebelJMSConsumerProcess">
     <implementation.bpel
src="SamplesCreateCustomerSiebelJMSConsumerProcess.bpel"/>
        <property name="bpel.config.transaction">required</property>


</component>
```

In the case of BPEL invoking Mediator, the Mediator participates in the existing transaction if a transaction is present. If a transaction is not present, then Mediator starts a transaction.

16.6.5 How to Transaction-Enable AIA Services

The following sections provide details on how to transaction enable AIA services in different scenarios:

- Section 16.6.5.1, "Synchronous Request-Response Scenarios"
- Section 16.6.5.2, "AIA Services in Asynchronous MEP"
- Section 16.6.5.3, "Asynchronous Operation from an ABCS in the Same Thread but in a Different Transaction"

16.6.5.1 Synchronous Request-Response Scenarios

Invoking a synchronous BPEL process results in creation of BPEL instance within its local transaction by default. This transaction can be enlisted to a global transaction given proper configurations.

Hence, the transaction settings in the ABCS services should be as shown in Example 16–10:

Example 16–10 Transaction Settings in the composite.xml of the RequesterABCS

```
<component name="SamplesCreateCustomerSiebelJMSConsumerProcess">
     <implementation.bpel
src="SamplesCreateCustomerSiebelReqABCSImplProcess.bpel"/>
        <property name="bpel.config.transaction">required</property>

</component>
```

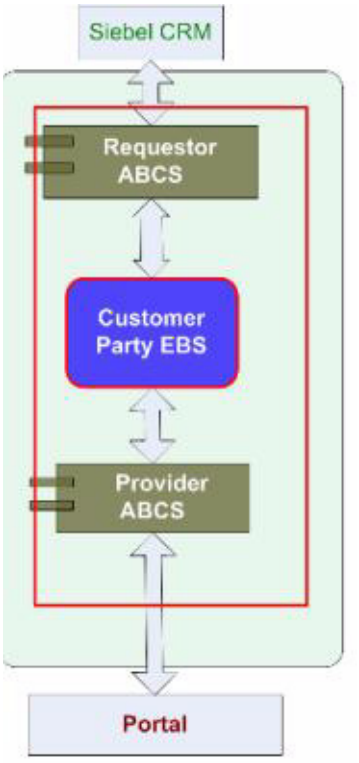
This setting ensures that the caller's transaction is joined, if there is one, or a new transaction is created when there is not one.

The EBS inherits the transaction initiated by the Requester ABCS.

A Mediator always enlists itself into the global transaction propagated through the thread that is processing the incoming message.

Figure 16–11 illustrates the transaction boundary.

Figure 16–11 Transaction Boundary



16.6.5.2 AIA Services in Asynchronous MEP

Invoking an asynchronous BPEL process results in persistence of invocation message in invoke message table within part of the caller's transaction. The Caller's transaction

ends. The asynchronous BPEL executes in its own local transaction. For the Callee to be executed within the Caller's transaction, the transaction flag must be set to *required*, as shown below, for the BPEL component, in the Callee's composite.

bpel.config.transaction=required

To ensure a single threaded execution of the Callee, a case of a one-way operation, you must have a sync-type call. Set the property for the BPEL component in the Callee's composite as shown below:

bpel.config.oneWayDeliveryPolicy=sync

Therefore, the transaction settings in the ABCS services should be as shown in Example 16–11:

Example 16–11 Transaction settings for asynchronous MEP

```
<component name="SamplesCreateCustomerSiebelReqABCSImplProcess">
     <implementation.bpel
src="SamplesCreateCustomerSiebelReqABCSImplProcess.bpel"/>
        <property name="bpel.config.transaction">required</property>
   <property name=bpel.config.oneWayDeliveryPolicy">sync</property>
```

If the Requester ABCS has an inbound interface with an adapter, then in the adapter's composite, set the property at the BPEL component level, that is, **bpel.config.oneWayDeliveryPolicy=sync.**

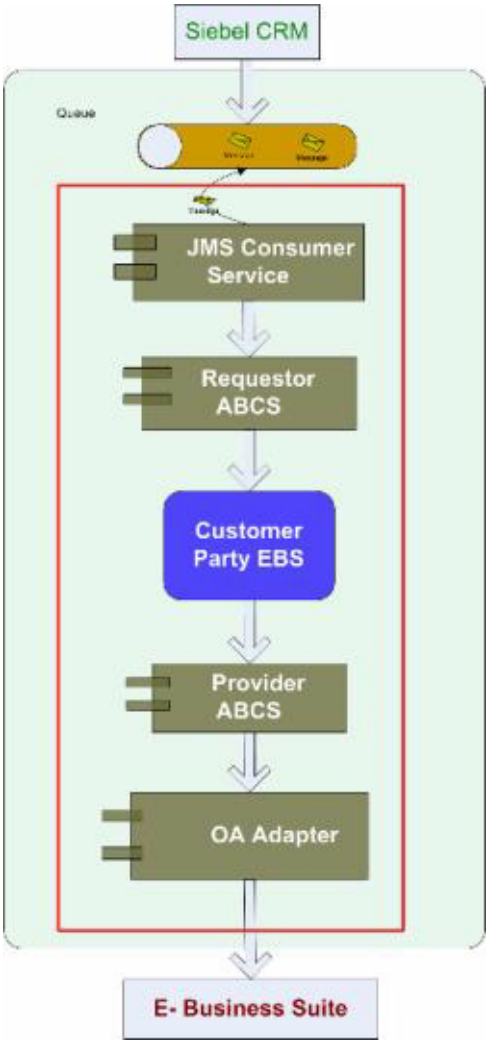
Example 16–12 shows how transaction properties are set on the adapter component:

Example 16–12 Transaction properties set on adapter component

```
<component name="SamplesCreateCustomerSiebelJMSConsumerProcess">
     <implementation.bpel
src="SamplesCreateCustomerSiebelJMSConsumerProcess.bpel"/>
        <property name="bpel.config.transaction">required</property>
   <property name=bpel.config.oneWayDeliveryPolicy">sync</property>
</component>
```

Figure 16–12 illustrates the transaction boundary.

Figure 16–12 Transaction Boundary in Asynchronous MEP



16.6.5.3 Asynchronous Operation from an ABCS in the Same Thread but in a Different Transaction

When an ABCS has to invoke an asynchronous operation on a process, which must be executed in the same thread as that of ABCS but in a different transaction, set the following configuration in the called BPEL composite:

```
<property name="bpel.config.oneWayDeliveryPolicy">sync</property>
```

In this case, the transaction configuration, **bpel.config.transaction,** need *not* be set in the called BPEL composite because the called BPEL should be executed in a transaction different from that of the Caller.

16.7 Guaranteed Message Delivery

Guaranteed message delivery means that a message being sent from a producer to a consumer is not lost in the event of an error.

AIA uses queues for asynchronous and reliable delivery of messages. For example,

- PeopleSoft CRM, upon occurrence of a business event, can either push the message directly into a queue or send a SOAP message over HTTP to a JMS

message producer (a JMS transport adapter) that is responsible for entering the message in the queue.

PeopleSoft CRM can consider the message as sent as soon as the message is dropped into the queue. With this mechanism, the PeopleSoft CRM application can keep sending new messages regardless of whether the CRM on Premises is available or not.

- A JMS consumer (another JMS transport adapter) is responsible for dequeuing the messages and invoking PeopleSoft CRM ABCSs.

  Having the PeopleSoft CRM ABCSs, the EBS, the CRM on Premises ABCS and the Web services as part of the transaction initiated by JMS message producer ensures that the message is removed from the queue only after the successful completion of the task in the CRM on-premise application.

16.8 Versioning ABCS

Versioning of a service means:

- A new service with Version suffix in the name of the Composite and name of the Service Component.
- The target namespace of the WSDL of the Service Component indicates a new version.

For more information about AIA naming standards, see Chapter 31, "Oracle AIA Naming Standards for AIA Development".

16.8.1 Guidelines for Versioning

During the preparatory phase, be aware of the following guidelines:

- An ABCS is used across multiple Pre-Built Integrations. It is not Pre-Built Integration specific.

  For example, *EBizRequesterCreateItemABCS* is used by ISCM Pre-Built Integration and Agile PLM - Ebiz Integration Pre-Built Integration.

- Use the Oracle Enterprise Repository to discover the existence of a service.

  Service definition annotations are persisted in Oracle Enterprise Repository. Oracle Enterprise Repository documents the various conceptual and physical assets depicting the Pre-Built Integration and constituting services.

- Inline changes made to ABCS without changing the version suffix should be backward compatible.

- Multiple versions for a service with semantically and technically incompatible contracts among versions are not acceptable.

  It is not permissible to version an existing ABCS when the contract is totally different, even though the operation to be performed or the verb is same.

- ABCS is versioned independently of an application version.

  ABCS is not bound to a specific application version. So, when an application undergoes a version change, it is not binding on ABCS to undergo a similar version change. Thus, ABCS is designed to be agnostic of version changes of the application.

- Avoid concurrent multiple versions of an ABCS.

For example, Team A owns Version 1; Team B does not like the contract and creates Version 2 with a different MEP/Contract. Now Team A is creating Version 3 with a contract that contradicts the previous one.

- The system does not allow multiple services for an application to perform a single business activity using a specific role (Requester/Provider).
- Do not have multiple services with same logic but different names, either a different name for composite, for service, or having a different portType in the WSDL.
- Do not have multiple services with different logic but the same names.

The operation defined on the EBS determines the implementing ABCS.

The operation defined on an EBS is dictated by the business activity/task.

Variations of business activity, task, or both are possible. These are passed in the form of context and implemented by different ABCSs.

Multiple teams involved in producing, consuming, or both producing and consuming an ABCS for a specific application and a business activity must reach consensus regarding the contracts.

16.9 Resequencing in Oracle Mediator

This section provides an overview of the Requencing feature in Oracle Mediator and discusses how to set up and use it in the Oracle AIA Asynchronous message exchange pattern.

In the Oracle AIA asynchronous message exchange pattern, participating applications push messages to the Oracle AIA layer where they are processed in a store-and-forward fashion.

In this message exchange pattern, there is a need to process messages pertaining to the same object instance in the same sequence in which they are generated. The Resequencer in Oracle Mediator helps to maintain correct processing sequence.

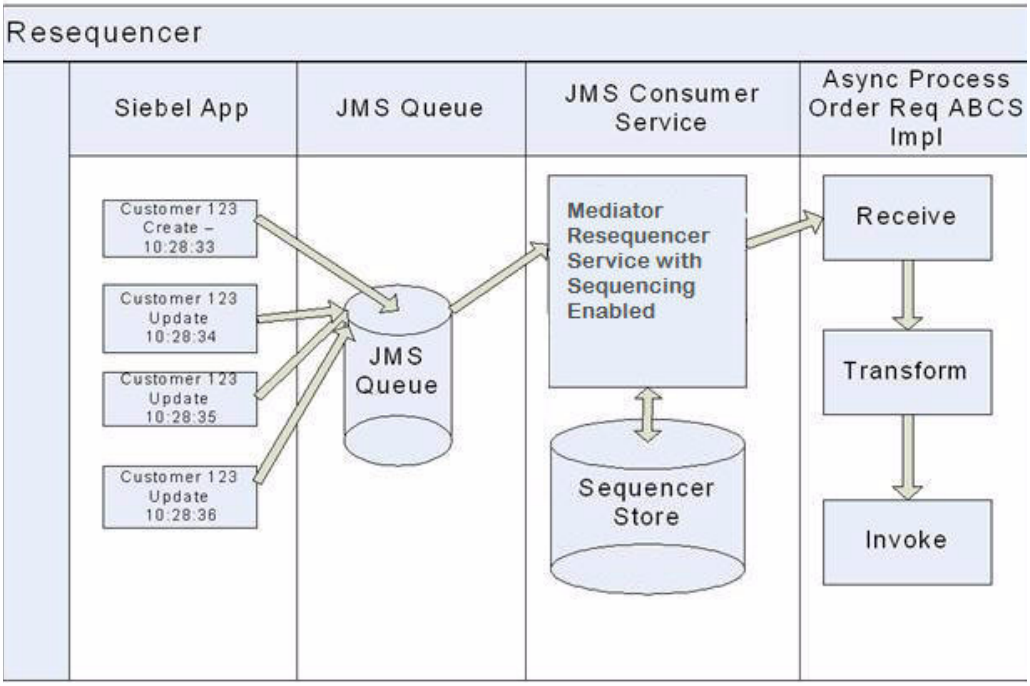
A Resequencer is used to rearrange a stream of related but out-of-sequence messages back into order. It sequences the incoming messages that arrive in a random order and then send them to the target services in an orderly manner.

The sequence of messages is maintained until the next persistence point. If the next persistence point is not an application, then the messages must be resequenced again.

Consider an integration scenario where in the Siebel application is sending CreateOrder/UpdateOrder messages which are enqueued in JMS queue. These messages are de-queued by a JMS consumer adapter service and sent down stream for processing. When incoming messages arrive at JMS queue, they may be in a random order owing to different reasons such as network latency, downtime of server and so on. The resequencer orders the messages based on sequential or chronological information, and then sends the message to the target services in an orderly manner.

The Figure 16–13 depicts an usecase where an incoming stream of messages from Siebel application which are persisted in a JMS destination are processed in a correct sequence.

Figure 16–13 Message Sequencing Using the Oracle Mediator's Resequencing Feature



The sequencing is done based on the sequencing strategy selected. The mediator provides us with three types of resequencers:

- Standard Resequencer
- FIFO Resequencer
- BestEffort Resequencer

Note: For more information about the Oracle Mediator Resequencer feature, see "Resequencing in Oracle Mediator" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

16.9.1 Configuring the Oracle Mediator Service to Use its Resequencer Feature

As this is a feature of the Oracle Mediator, a Mediator service must be in place for it to work as designed.

This section discusses how to configure the Oracle Mediator service to use its re-sequencing feature.

- Open the Mplan file of the Mediator component in the design mode.
- Select the 'Resequence Level' as 'operations'

> **Note:** Mediator allows us to select the Resequence Level as either 'operations' or 'component'. For Mediator components which only have a single operation, it does not matter whether you select Resequence Level as either 'operations' or 'component'. For a Mediator that has multiple operations defined on it, selecting the 'component' option means resequencing is applied to all operations in it.

In AIA, for the mediator, it is recommended to select the 'Resequence Level' as 'operations'. Figure 16–14 is the screen shot of a composite mplan where the Resequence level is selected at operation level.

Figure 16–14 Resequence Level is Selected at Operation Level

For the resequencer to work, have one or both of following values in the source payload depending on the resequencing strategy employed.

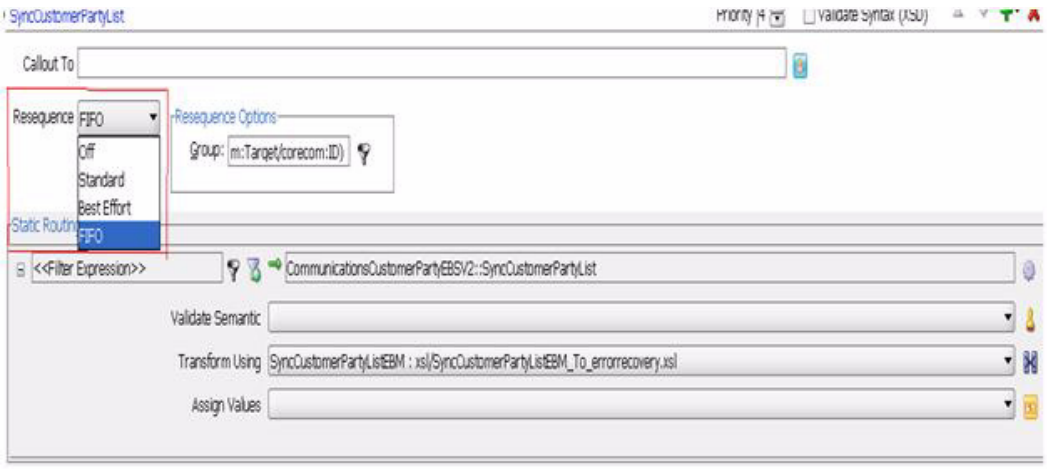
The first one is a sequenceID, and the next one is a groupdID. The sequenceID is used as identifier for the message itself. The groupID is used to group the messages in which the resequencer rearranges the messages based on the sequenceID.

16.9.2 How to Configure the Resequencing Strategy

1. Determine and set the Resequence type from the following three options:
   - **Standard:** Use when the incoming message has an integer as a sequence ID with a known increment and the starting sequence number.
   - **BestEffort:** Use when the incoming message has a timestamp as a sequence ID.
   - **First in First out (FIFO):** Use in all other cases.

   Figure 16–15 is the screenshot that shows Resequence types.

Figure 16–15 Resequence Types



2. Set resequence options.

   a. Determine and set the Group

      The group ID is a parameter (field or attribute) on the incoming message that uniquely identifies the group of messages to be sequenced.

      For example, CustomerId, OrderId, ProductId, Account ID and so forth.

      The group ID is configured by setting the 'Resequence Options' called 'Group' with an XPath expression that points to the field in the payload which the resequencer uses to group incoming messages. This ensures that the messages belonging to a specific group are processed in a sequential order.

      For example, *SyncCustomerPartyListEBM/ns0:SyncCustomerPartyListEBM/ns0:DataArea/ns0:SyncCustomerPartyList/ns0:CustomerPartyAccount/corecom:Identification/corecom:ApplicationObjectKey/corecom:ID[@schemeID='AccountId']*.

      Figure 16–16 is the screenshot that shows XPath expression in the **Group** field.

Figure 16–16 Resequence Options

   b. Determine and set the ID

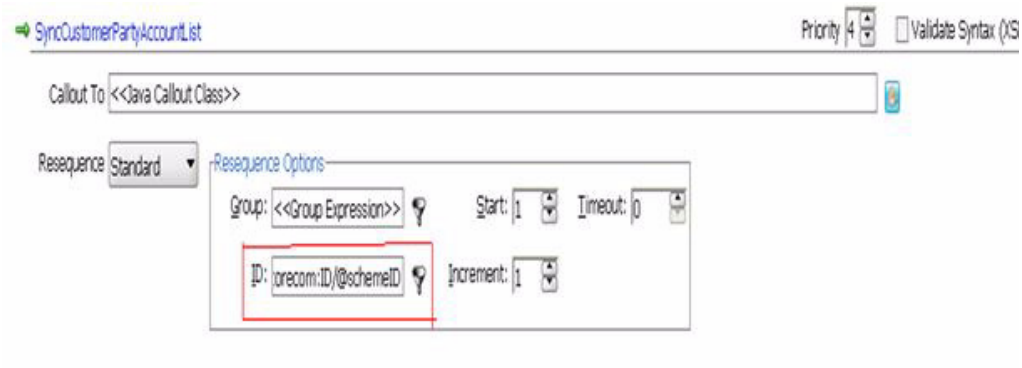
      The ID is a parameter (field or attribute) on the incoming message that indicates the position of the message in the sequence.

The ID is declared by an XPath expression that points to the field in the incoming message on which the resequencing is done.

For example,
$in.SyncCustomerPartyAccountListEBM/ns0:SyncCustomerPartyAccountListEBM/ns0:DataArea/ns0:SyncCustomerPartyAccountList/ns0:CustomerPartyAccount/corecom:Identification/corecom:ApplicationObjectKey/corecom:ID /@schemeID

Figure 16–17 is the screenshot that shows XPath expression in the **ID** field.

Figure 16–17 Setting the ID Parameter



c. Set the Resequence Options for Best Effort Resequence Mode. The incoming messages cannot provide information to the resequencer about the identifier to use for sequencing. Typically, the identifier used for sequencing in such scenarios is of a dateTime type or numeric type. Using the dateTime field as the sequence ID XPath enables you to control the sequencing.

Figure 16–18 is the screenshot that shows the option dateTime selected.

Figure 16–18 Resequence Options for Best Effort Resequence Mode

When a resequencer mediator service is placed between the JMS Consumer adapter service and the Requester ABCS, the message is removed from the JMS queue when it goes into resequencer so that transaction boundary is shifted to resequencer store.

16.9.3 Processing Multiple Groups Parallelly

The messages that arrive for resequencing are split into groups and the messages within a group are sequenced according to the Id selected resequencing strategy. Sequencing within a group is independent of the sequencing of messages in any other group. Groups in themselves are not dependent on each other and can be processed independently of each other.

For example, when the 'accountID' in Update Order message is used as group ID, the incoming Update Order messages that have the specific account ID are processed as a group independent of the Update Order messages that have a different account ID.

Resequencer helps maintain correct processing sequence up to the point that directly receives message from resequencer. The sequence is not guaranteed if a-synchronicity is introduced in the process that receives the re-sequenced messages. In other words, the sequence of messages is maintained till the next persistence point. After that the messages must be resequenced again.

16.9.4 Describing Oracle Mediator Resequencer Error Management

When a message for a group fails, message processing for that group is stopped and pending messages are held back. Error messages are placed in the Error Hospital for fixing and resubmission for all the errors that can be retried.

As all messages of same group id are processed by a single thread in different transactions. If a message fails to get processed, the rest of the messages for that group are suspended. So resubmit the message that has error. Processing resumes from next sequence only after you either retry or abort the message.

The processing of messages for other groups continues without interruption.

If a new message arrives for the group that has error, the messages get stored in the Resequencer store so that they not lost. When failed messages are resubmitted from the Oracle Enterprise Manager (OEM) Console or another tool, the associated group that has error is unlocked and normal processing resumes for that group.

16.9.4.1 Resubmitting Messages that have Errors

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 topic or Resequencer. It is here that the message is persisted until it can be resubmitted for delivery to the service or component.

For Message Resubmission Utility to resubmit messages that have errors, the incoming messages must be first populated using Message resubmission parameters when they are de-queued from the source milestone using the JMS consumer.

For more details on how to populate these values in the incoming message, see Section 26.5.3.1, "Populating Message Resubmission Values".

16.9.4.2 Using the Message Resubmission Utility API

The Message Resubmission Utility API enables external programs to enable a message that is in error state to be consumed for a transaction. This utility is typically run to fix the problem that caused the message to end in error.

For more information about the Resubmission Utility, see section "Resequencer Based Resubmission" of the chapter "Using the Message Resubmission Utility" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack*.

The message that generates an error is moved to the error hospital. The group is moved to an error state and is not picked up by the worker thread. You can resubmit the error-generating message using the error hospital. After you resubmit the message, the group is moved to a normal state, so that it can be picked up by the worker thread.

16.9.5 Tuning the Resequencer

The following parameters help you with turning the resequencer.

ResequencerWorkerThreadCount: The number of threads used by resequencers.

ResequencerMaxGroupsLocked: The maximum number of groups that can be locked by a thread.

The sleep interval: If your message frequency is high, set the sleep interval to low. It is desirable to set the number of groups locked to a value greater than the worker thread count so that worker threads are not idle. By default, the value of ResequencerMaxGroupsLocked and ResequencerWorkerThreadCount are set to same.

Note: Note: You cannot turn off the resequencer at runtime after you turn it on for a composite.

16.10 Developing Layered Customizations

Oracle application composites are created for specific industry needs. Oracle also provides options to further customize these components for business specific needs of individual organizations. Customization refers to modification of delivered artifacts to produce new behavior. You can use this option to customize the out-of-box composites delivered by Oracle and also and deploy them in place of original composites. When you customize a few composites, you create a specific solution for your organization.

However, these customizations can be overwritten when you apply a patch or upgrade when there is a new version. To ensure customization at a higher layer is not lost during an upgrade the customized layer and the base process are kept in their own metadata files. This externalization of customizations away from the base composite is possible only when the base version has been enabled for customization. Only when you want to deploy a solution the layers are merged with the base process for creating a single executable process.

For more information about layered customizations, see "Customizing SOA Composite Applications" in *Oracle® Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Can all the composites be customized?

For the services that are delivered in the certain set of Out of the Box Prebuilt Integration, customizable scopes have been added in the BPEL flow. Refer to your pre-built integration implementation guide to check whether your pre-built integration allows customization of services. You can add the custom logic in these scopes when you open the BPEL in JDeveloper using customization developer role. These customizations are kept separate from the base definition. The advantage of adding the custom logic in these scopes is that, the customization is safe when you upgrade. When you uptake the subsequent patches or new releases, they can merge these customizations on top of the new base definition and deploy the BPEL.

Pre-built integrations that do not have services containing customizable scopes cannot be customized.

Adhere to following guidelines while doing the customizations:

- Customizations are allowed only for BPEL based AIA artifacts that have customizable scopes. Inline customizations done to any other artifacts are overwritten when you apply a patch or upgrade to a newer.
- Do not customize a composite which is not marked as customizable because they encounter merge conflicts when the composite is patched at a later point in time.
- Do not modify the out of the box defined variables or messages as these variables may have been used in the section following customized code resulting in adverse impact on the behavior of the BPEL flow.

16.10.1 Deploying services after customizations

To deploy a service after customization:

1. Right click the customized composite in JDeveloper tree.
2. Select **Make<customized composite name>.jar** to compile the changes.
3. Open the **CustomizeApp** folder from the Jdeveloper work location.

   You can see the folder named **merged** that contains Out of the box code and Customized Code.
4. Copy this merged folder to your server and deploy the composite using Application Deployment Driver.

   For instructions on how to deploy the changed services see,Chapter 8, "Generating Deployment Plans and Deploying Artifacts"
5.

16.10.2 Customizing the Customer Version

If you want to customize customer version of the SOA composite application, see "Customizing the Customer Version" in *Oracle® Fusion Middleware Developer's Guide for Oracle SOA Suite*.

16.10.3 Applying patches after customization

To apply patches after you have customized, see "Upgrading the Composite" in *Oracle® Fusion Middleware Developer's Guide for Oracle SOA Suite*. Instructions here help you to apply layers to the new base composite.

17

Designing and Constructing Composite Business Processes

This chapter provides an overview of Composite Business Processes (CBP) and discusses how to define the contract for CBP, create a contract for a CBP, implement the CBP as a BPEL service.

This chapter includes the following sections:

17.1 Introduction to Composite Business Processes

Composite Business Processes (CBPs) are the implementation of process services. Process services orchestrate a series of human and automated steps, including enterprise-wide policies captured in business rules. These services run the implementations of the business processes in the Oracle Application Integration Architecture (AIA) Reference Process Models.

AIA recommends using BPEL for implementing CBPs. CBPs are long-running processes that may run from few seconds to days. A CBP has an interface and message structure that is detailed enough to capture all of the information about the source of the triggering event. In most cases, the event is triggered by customer-facing applications.

For more information about detailed definition and high-level design and development guidelines, see "The AIA Shared Service Inventory" in the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

17.2 How to Define the Contract for a CBP

The AIA methodology for designing and implementing a CBP is *contract first* methodology. Therefore, the contract must be defined and created before the CBP is implemented.

To define the contract for a CBP:

1. Identify the CBP.
2. Identify the pattern for the CBP.

17.2.1 How to Identify the CBP

The first task involved in designing a new service is to verify whether it is necessary. After the need for the CBP is established, review each of the services and the description of the operation and any metadata in the Oracle Enterprise Repository before deciding to create a service or an operation.

A CBP may be needed when an enterprise-wide business process is needed for a business event. This process is described in the Reference Process Model and spans multiple operational units for enterprises.

To identify a CBP:

1. From the BPA Reference Process Model, identify the business process to implement.
2. Click the link to Oracle Enterprise Repository (OER), and review the details in the OER.
3. If no link is available, create a definition for the CBP as part of the relevant AIA Project in the AIA Project Lifecycle Workbench application.

17.2.2 How to Identify the Message Pattern for a CBP

The CBP is always modeled to implement a single operation with one-way patterns. No responses to the client are made. Any error situation or response is modeled as an update operation back to the client.

17.3 How to Create the Contract for a CBP

To create the contract for a CBP:

1. Identify the message structure.
2. Construct the WSDL for the CBP.
3. Annotate the service interface.

   See Chapter 12, "Annotating Composites".
4. Ensure WS-1 basic profile conformance.

17.3.1 How to Construct the WSDL for the CBP

The CBP development starts with constructing a WSDL, and the result of the technical design process is a CBP WSDL.

17.4 How to Implement the CBP as a BPEL Service

To implement the CBP:

1. Create a WSDL.

   Create a WSDL for the CBP following the CBP naming standards and the WSDL templates provided.
2. Implement the CBP as a one-way BPEL process.

   For more information about the details of creating BPEL projects in Oracle JDeveloper, see the *Oracle BPEL Process Manager Developer's Guide*.

Refer to Chapter 15, "Constructing the ABCS" for details about implementing a one-way pattern

3. Enable error handling and logging.

   Enterprise Business Services (EBSs) should handle errors to allow clients or administrators to resubmit or re-trigger processes. This is accomplished using a central error handler.

   For more information, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging".

4. Enable extensibility points in CBP.

   For more information, see Section 16.1, "Developing Extensible ABCS".

18

Designing and Constructing Enterprise Business Flows

This chapter provides an overview of Enterprise Business Flows (EBF) describes how to define and implement EBFs.

This chapter includes the following sections:

- Section 18.1, "Introduction to Enterprise Business Flows"
- Section 18.2, "How to Define the Contract for an EBF"
- Section 18.3, "How to Create the Contract for an EBF"
- Section 18.4, "How to Implement the EBF as a BPEL Service"

18.1 Introduction to Enterprise Business Flows

The EBF is used for implementing a business activity or a task that involves leveraging capabilities available in multiple applications. The EBF is about stringing a set of capabilities available in applications to implement a coarse-grained business activity or task and composing a new service leveraging existing capabilities. The EBF involves only system-to-system or service-to-service interaction. The EBF has no activity that needs human intervention.

In a canonical integration, the EBF is an implementation of an Enterprise Business Service (EBS) operation and calls other EBSs. It never calls an Application Business Connector Service (ABCS) or the applications directly. In other integration styles, the caller invoking the EBF can be either an application or any other service.

Figure 18–1 and Figure 18–2 illustrate how some EBFs in the Order to Cash Process Integration Pack Pre-Built Integration are implemented to leverage existing capabilities.

Figure 18–1 shows that the EBF is orchestrating a flow for syncing customers between the source application and the target application. When the sync operation is invoked from the source application, the EBF first checks whether the customer exists in the target application. If so, it updates the customer in the target application; otherwise, it creates the customer in the target application.

Figure 18–1 Example of EBF Orchestrating a Flow from Source to Target

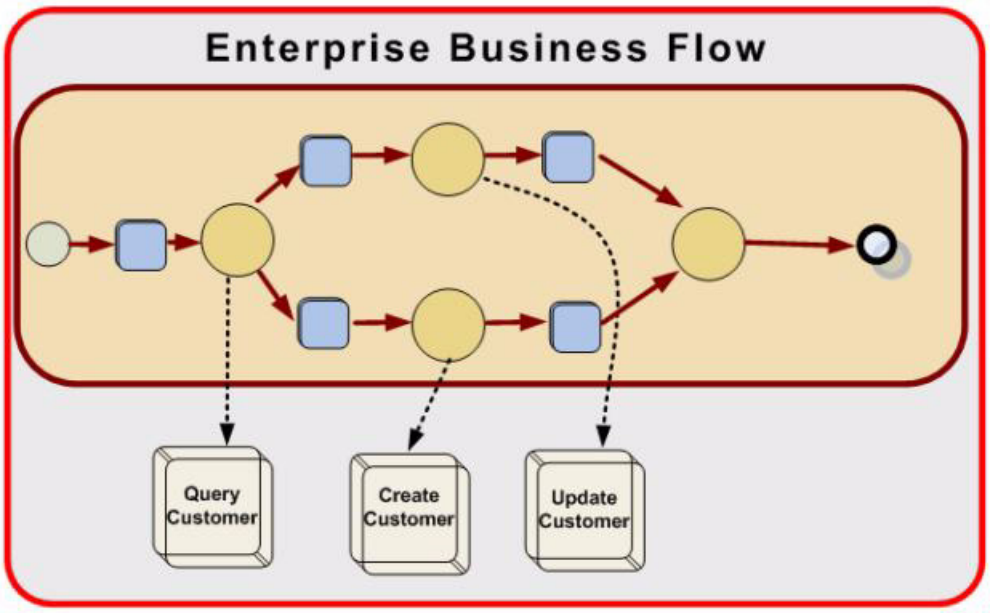
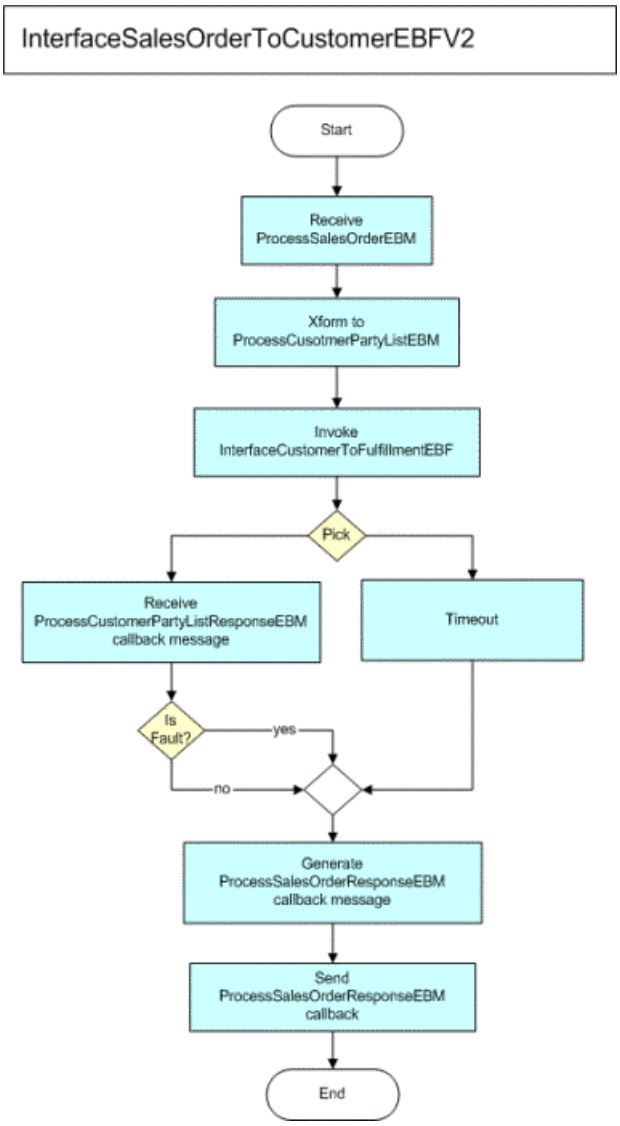


Figure 18–2 demonstrates a flow that is receiving the sales order and orchestrating across customer creation, fulfillment, and update back in the source application with response.

Figure 18–2 Example of Sales Order Flow



18.2 How to Define the Contract for an EBF

The Oracle Application Integration Architecture (AIA) methodology for designing and implementing an EBF is *contract first* methodology. Hence, the contract must be defined and created before the implementation of the EBF.

To define the contract:

1. Identify the EBF.
2. Identify the pattern for the EBF.
3. Identify the Enterprise Business Message (EBM) to be used for the requests and responses (if any).

18.2.1 How to Identify the Need for an EBF

The first task involved in designing a new service is to verify whether it is necessary.

- Situations may occur in which other services exist that are providing some or all of the features identified. Each of the services and the operation's description and any metadata should be reviewed before you decide to create either a new service or an operation.
- An EBF is needed when an EBS operation must be implemented with a set of tasks and involves invoking of multiple services.
- In a canonical based integration, an EBF can invoke only another EBS.

18.2.2 How to Identify the Message Pattern for an EBF

The Enterprise Business Flow is modeled to implement a single operation.

To identify the message exchange pattern (MEP) for an EBF:

1. Based on the understanding of the business process requirement from the Functional Design Document (FDD), identify the triggering event for the EBF.

Example 18–1 Example of Message Exchange Pattern identification for EBF

```
<operation name="InterfaceSalesOrderToCustomer">
<documentation>
               <svcdoc:Operation>
                   <svcdoc:Description>This operation is used to interface
                                       sales order to
customer</svcdoc:Description>
                   <svcdoc:MEP>ASYNC_REQ_RESPONSE</svcdoc:MEP>

<svcdoc:DisplayName>InterfaceSalesOrderToCustomer</svcdoc:DisplayName>
                   <svcdoc:LifecycleStatus>Active</svcdoc:LifecycleStatus>
                   <svcdoc:Scope>Public</svcdoc:Scope>
               </svcdoc:Operation>
           </documentation>
           <input message="client:InterfaceSalesOrderToCustomerReqMsg"/>
   </operation>
```

   a. If the control is to be blocked until a response is returned to the point of invocation, choose EBF Request-Reply pattern. This would be a **synchronous** call.

   b. If after the EBF is invoked the triggering point does not wait for the response and continues on, this invocation of the EBF would be an **asynchronous** call.

      Check whether the processing of the EBF results in a response.

      Is there a need to correlate the request and the response?

      If the answer is yes, this is a case of delayed response. Use the EBF request-delayed response pattern. If the answer is no, then choose the EBF fire-and-forget pattern.

      Any EBF operation invoked because of a subscription to a publish event should use the EBS subscribe pattern.

18.2.3 How to Identify the Message Structure

To identify the message structure:

1. A CBP is implemented by orchestrating calls to external services. The event triggering the CBP indicates the business object with which the CBP will be dealing.
2. Depending on the integration style, the message structure is decided as described below:
   a. If the business object is available in the existing Enterprise Business Message (EBM), use it.
   b. If the business object is not available in the existing EBM but can be assembled from the existing Enterprise Business Object (EBO) Library, construct it.
   c. If the business object is not available in the EBO Library, identify a payload suitable for capturing all relevant information and processing in the CBP.

18.3 How to Create the Contract for an EBF

To create the contract for an EBF:

1. Identify the EBM.
2. Construct the WSDL for the EBF.
3. Annotate the service interface.
4. Ensure WS-1 basic profile conformance.

18.3.1 Constructing the WSDL for the EBF

Because the EBF development starts with constructing a WSDL, the result of the technical design process is an EBF WSDL.

18.4 How to Implement the EBF as a BPEL Service

To implement the EBF:

1. Create a WSDL.

   Create a WSDL for the EBF following the EBF naming standards and the WSDL templates provided.
2. Implement the EBF as a synchronous or asynchronous BPEL process.

   For more information about the details of creating BPEL projects in Oracle JDeveloper, see the *Oracle BPEL Process Manager Developer's Guide*.

   Follow Section 15.5, "Implementing the Asynchronous Request Delayed Response MEP" for details on how to use BPEL to implement asynchronous request-delayed response pattern. The difference is that the EBF deals with EBMs.
3. Enable error handling and logging.

   EBSs should handle errors to enable clients or administrators to resubmit or retrigger processes. This is done through a central error handler.

   For more information, see Chapter 26, "Configuring Oracle AIA Processes for Error Handling and Trace Logging"

4. Enable extensibility points in the EBF.

19

Introduction to B2B Integration Using AIA

Business-to-business (B2B) integration requires the ability to exchange business information with trading partners using a choice of B2B document protocols. This chapter provides an overview of B2B integration using AIA, B2B document flows, B2B component of Oracle Fusion Middleware and Foundation Pack infrastructure for B2B.

This chapter includes the following sections:

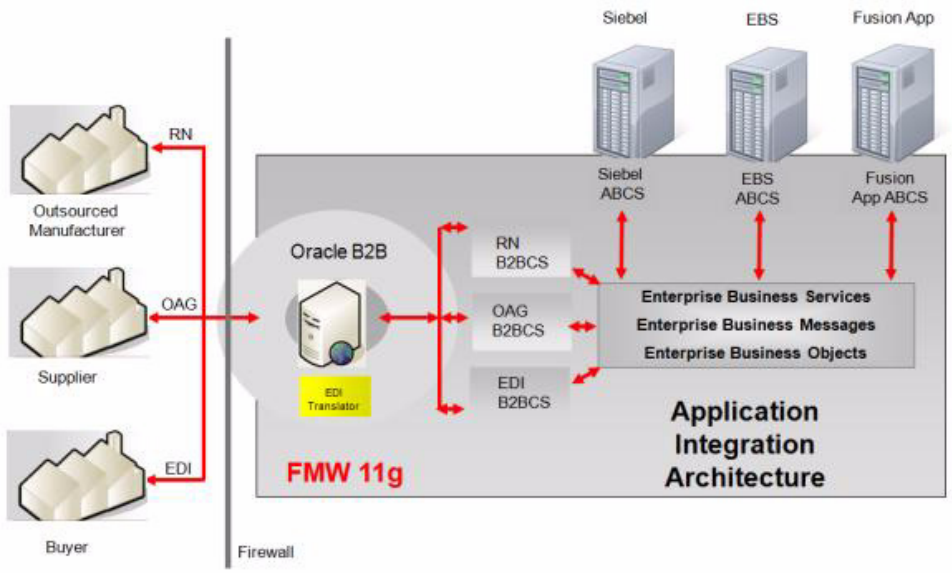
19.1 Overview of B2B Integration Using AIA

The Oracle Application Integration Architecture (AIA) B2B solution consists of the following key features:

- A flexible integration architecture that can be used to rapidly build support for new B2B document protocols.
- Infrastructure components that can be used to build end-to-end B2B flows using AIA and Oracle Fusion Middleware.
- Prebuilt support for popular B2B document protocols.
- Support for customization of shipped B2B integration artifacts.

Figure 19–1 illustrates how the canonical-based integration architecture of AIA can be applied to meet the B2B integration needs of large enterprises.

Figure 19–1 Schematic Overview of the B2B Architecture of AIA



Following are some key benefits of using AIA for B2B integrations:

- AIA decouples participating applications from the B2B integration layer.

  Participating applications no longer required to track the varying B2B integration needs of trading partners. In addition, customers can add support for new B2B protocols and trading partner requirements without any impact to participating application code.

- B2B document flows are often a subset of tasks that comprise complex business processes, such as Order Capture.

  By building B2B functionality as a layer on top of Foundation Pack, the functionality is made available for reuse by multiple applications and business processes.

- Existing AIA Enterprise Business Objects (EBOs) such as Customer, Supplier, Item, Purchase Order, Shipment, Invoice, and Catalog Address map to most of the commonly used B2B documents.

  Usage of these readily available EBOs, along with prebuilt B2B connector services (B2BCSs) for these EBOs, result in significant time and cost savings for B2B integrations.

19.2 Understanding B2B Document Flows

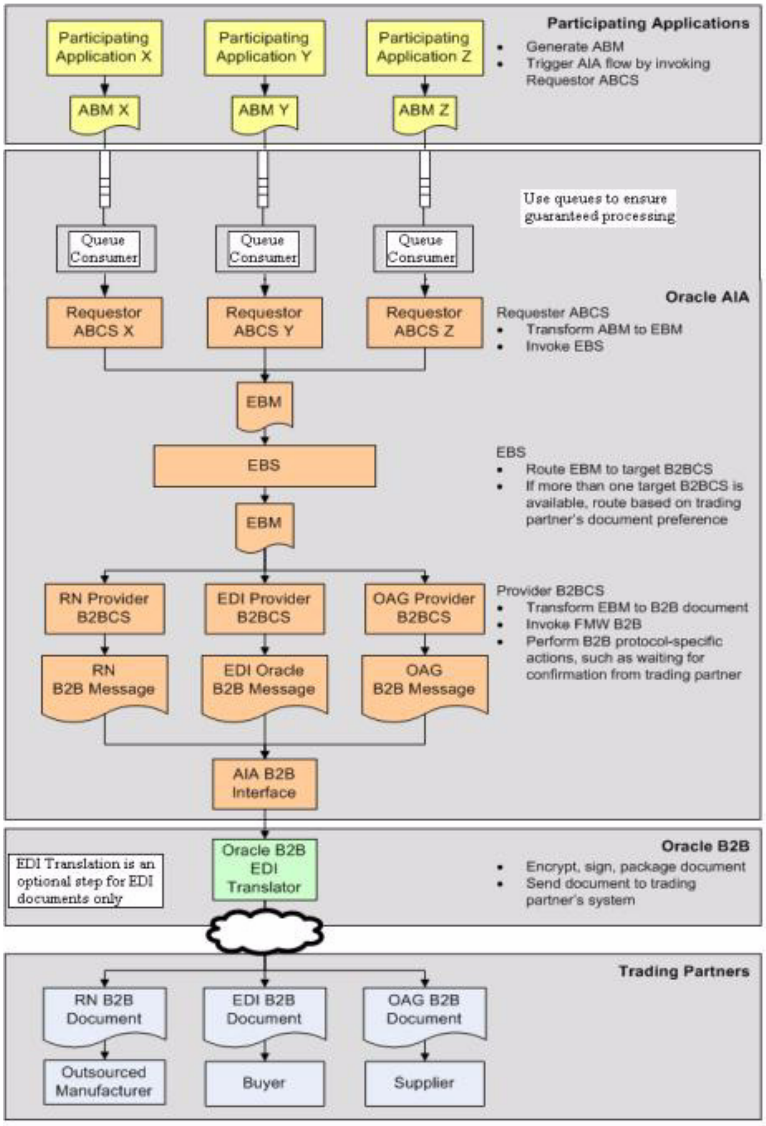
This section includes the following topics:

- Section 19.2.1, "Describing Outbound B2B Document Flows Built Using AIA"
- Section 19.2.2, "Describing Inbound B2B Document Flows Built Using AIA"

19.2.1 Describing Outbound B2B Document Flows Built Using AIA

Figure 19–2 illustrates a sample outbound B2B flow implemented using AIA:

Figure 19–2 Outbound B2B Document Flow



Following are the key steps in an outbound B2B document flow:

1. Participating applications trigger requester Application Business Connector Services (ABCS) with Application Business Messages (ABMs) as their payloads.

   A best-practice recommendation is to use queue-based communication between the application and the requester ABCS while modeling fire-and-forget flows. This ensures that request messages are persisted, thereby guaranteeing their processing.

2. Requester ABCSs transform ABMs into Enterprise Business Messages (EBMs) and then invoke the Enterprise Business Service (EBS) implementation.

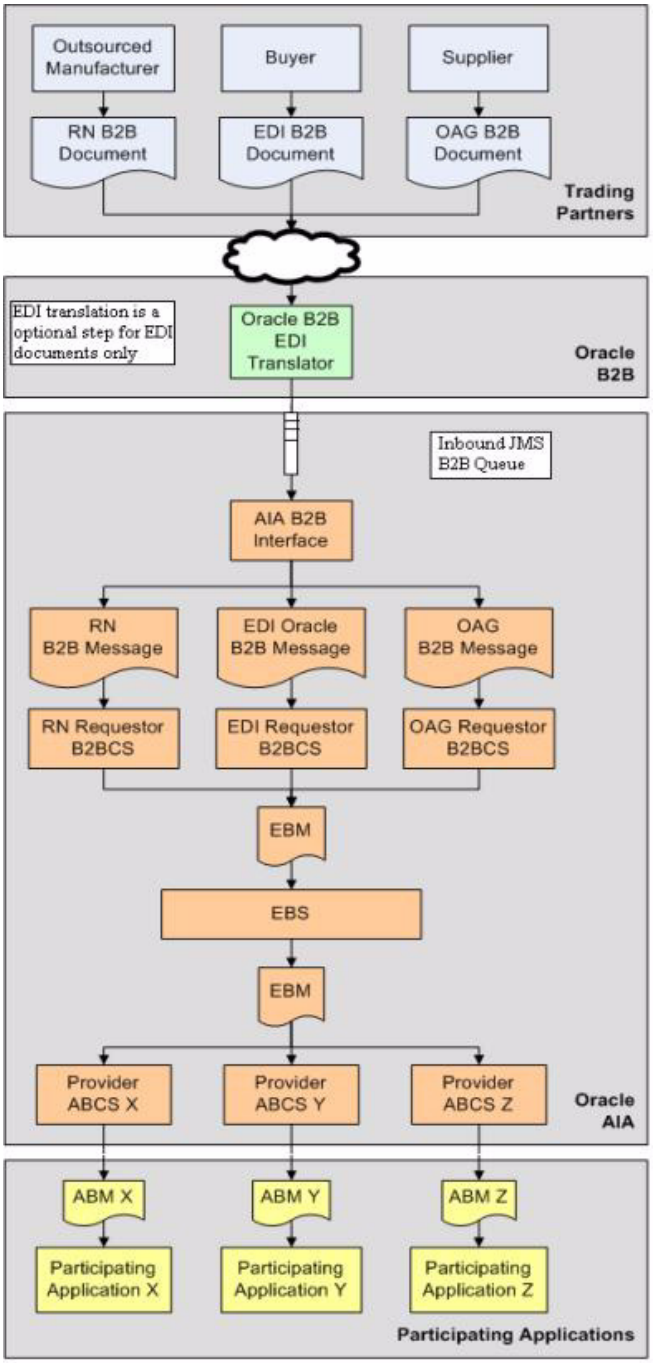
3. The EBS introspects the EBM and routes it to the correct provider B2BCS based on the trading partner's B2B document preference.

4. Provider B2BCSs transform the EBM into an industry-standard B2B format and invoke the AIA B2B interface service.
5. The AIA B2B interface service populates the required header parameters and invokes the Oracle Fusion Middleware B2B component, passing the B2B payload as input.
6. Oracle B2B looks up the trading partner agreement corresponding to the trading partners and document type and securely transports the B2B document to the remote trading partner
7. In the case of non-XML format-based protocols such as Electronic Data Interchange (EDI), the translation feature in Oracle B2B can be used to transform the intermediate XML generated by AIA layer to the non-XML B2B document format.

19.2.2 Describing Inbound B2B Document Flows Built Using AIA

Figure 19–3 illustrates a sample inbound B2B flow implemented using AIA:

Figure 19–3 Inbound B2B Document Flow



Following are the key steps in an inbound B2B document flow:

1. Oracle B2B receives B2B documents from trading partners and identifies the sending trading partner and B2B document type.
2. In case of EDI other non-XML formats, Oracle B2B translates the inbound payload into an intermediate XML format.
3. The message is queued into a JMS B2B inbound queue. Along with the B2B document payload, the following key metadata about the B2B document are also passed along to the AIA and SOA composite layer:

- From Trading Partner
- To Trading Partner
- Document Type
- Document Type Revision

4. The AIA B2B interface receives the B2B document and header information from Oracle B2B and routes it to the correct requester B2BCS based on the Document Type parameter.
5. The requester B2BCS transforms the B2B document to the EBM and invokes the EBS.
6. The EBS introspects the EBM and routes it to the correct provider ABCSs.
7. The provider ABCSs transform the EBM to ABM formats and invoke participating application APIs or web services.

19.3 Understanding the Oracle B2B Component of Oracle Fusion Middleware

Oracle B2B is an e-commerce gateway that enables the secure and reliable exchange of business documents between an enterprise and its trading partners. Oracle B2B supports B2B document standards, security, transports, messaging services, and trading partner management. The Oracle SOA Suite platform, of which Oracle B2B is a binding component, enables the implementation of e-commerce business processes.

This section includes the following topic: Section 19.3.1, "How AIA Complements Oracle Fusion Middleware Oracle B2B."

19.3.1 How AIA Complements Oracle Fusion Middleware Oracle B2B

As described in the previous section, AIA leverages the capabilities of the Oracle B2B component to build end-to-end B2B integration solutions. Note especially the key distinctions between the capabilities of Oracle B2B and AIA in the area of B2B integration.

At a high level, Oracle B2B provides B2B support by contributing the following functionality:

- Transport of B2B document payloads to remote trading partner systems.
- Support of secure integrations through encryption, digital signatures, and nonrepudiation.
- Support of industry-standard message-packaging protocols, such as Applicability Statement 2 (AS2) and RosettaNet Implementation Framework (RNIF).
- Transparent, configuration-based translation between non-XML formats, such as EDI and positional or delimiter-separated flat files, and XML.
- Support of a single, enterprise B2B information gateway, including auditing and reporting.

At a high level, AIA provides B2B support by contributing the following functionality:

- Prebuilt transformation between AIA canonical objects and B2B standard document formats.

- Canonical enterprise object library and services, reference architecture, infrastructure, and developer tools that can be used to build custom B2B integrations.
- Business process orchestration of B2B flows using SOA integration best practices.

For more information about Oracle B2B, see *Oracle Fusion Middleware User's Guide for Oracle B2B.*

19.4 Understanding the Foundation Pack Infrastructure for B2B

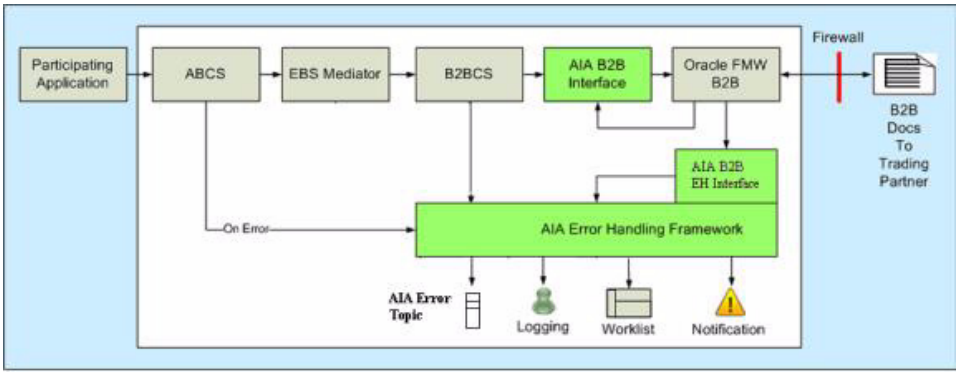
This section includes the following topics:

- Section 19.4.1, "B2B Support in AIA Error Handling Framework"
- Section 19.4.2, "AIA B2B Interface"

Along with the architecture and programming model, additional infrastructure components are delivered with Foundation Pack in support of B2B integrations.

This section provides a high-level description of the B2B infrastructure delivered with Foundation Pack. Chapter 20, "Developing and Implementing Outbound B2B Integration Flows" and Chapter 21, "Developing and Implementing Inbound B2B Integration Flows" provide more details on how to use the infrastructure components shown in Figure 19–4 when you develop and implement B2B flows using AIA.

Figure 19–4 Foundation Pack B2B Infrastructure Components



19.4.1 B2B Support in AIA Error Handling Framework

The AIA Error Handling Framework includes support for error scenarios in B2B integration flows.

1. The AIA B2B Error Handling Interface composite listens for errors that occur in the Oracle Fusion Middleware B2B component.
2. Errors that occur in Oracle B2B are dequeued from the B2B queues and transformed into the canonical AIA fault structure.
3. The AIA Error Handling Framework is then triggered with this fault message as input.
4. After the AIA Error Handling Framework has been triggered, its delivered error-handling capabilities can be leveraged for error resolution.

Some key capabilities of the AIA Error Handling Framework include:

- Transformation of all errors to a canonical fault message and queuing of the error to the AIA Error Topic.
- Logging of error information.
- Option to use the Oracle BPM Worklist application to track error resolution tasks.
- Option to issue error notifications to preconfigured roles.
- Ability to launch custom error handling and compensation processes.

These ready-to-use error-handling capabilities reduce duplication of error-handling code across integration code.

For more information about the functionality and uptake of the Error Handling Framework, see "Setting Up Error Handling" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

19.4.2 AIA B2B Interface

The AIA B2B interface is a reusable mediator-based SOA composite that acts as an abstraction layer between B2BCSs and the Oracle B2B component in Oracle Fusion Middleware. The AIA B2B interface shields B2BCSs from the choice of internal delivery channel employed to connect to Oracle B2B.

Upon installation, the Oracle AIA Installer configures the AIA B2B interface to connect to Oracle B2B using JMS-based queues. However, based on specific integration needs, an implementation may require the use of any of the other available internal delivery channels, such as Oracle Advanced Queuing, File Delivery Channel, and B2B adapter to integrate AIA with Oracle B2B. The AIA B2B interface composite can be modified to replace the use of JMS with any of these internal delivery channels without having to modify every B2BCS.

For more information about the B2B adaptor, see "Getting Started with Oracle B2B" in *Oracle Fusion Middleware User's Guide for Oracle B2B.*

20

Developing and Implementing Outbound B2B Integration Flows

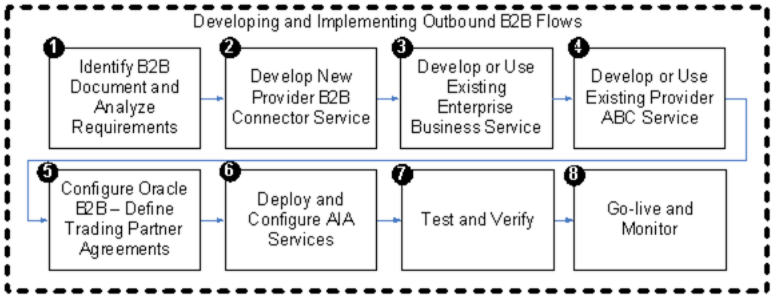
This chapter provides an overview of developing and implementing outbound B2B integration flows and describes how to identify the B2B document and analyze requirements, develop new provider B2B connector service, develop or extend an existing Enterprise Business Service, develop or extend an existing requester ABCS, configure Oracle B2B and define trading partner agreements, deploy and configure AIA services and finally how to test and go live.

This chapter includes the following sections:

20.1 Introduction to Developing and Implementing Outbound B2B Integration Flows

Figure 20–1 shows the high-level steps involved in developing a simple outbound business-to-business (B2B) flow from an application to trading partners using Oracle Application Integration Architecture (AIA).

Figure 20–1 High-Level Steps to Develop and Implement a Simple Outbound B2B Flow



Each of the steps listed in this diagram are described in detail in the following sections.

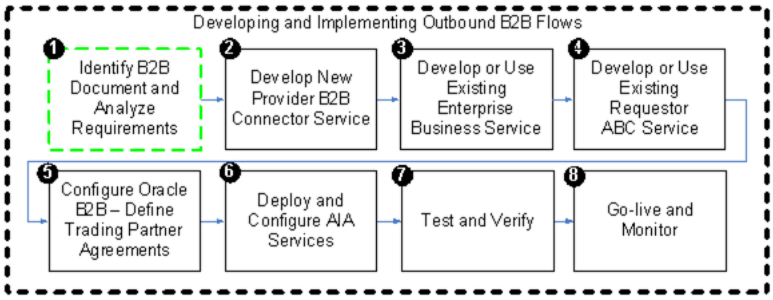
20.2 Step 1: Identifying the B2B Document and Analyzing Requirements

This section includes the following topics:

- Section 20.2.1, "How to Identify the B2B Document Protocol"
- Section 20.2.2, "How to Identify the B2B Document Type and Definition"
- Section 20.2.3, "How to Define the Document in Oracle B2B"
- Section 20.2.4, "How to Define the Document in AIA"
- Section 20.2.5, "How to Identify the EBO, EBS, and EBM to Be Used"
- Section 20.2.6, "How to Design Mappings for the B2B Document"
- Section 20.2.7, "How to Publish Mapping to Trading Partners"

The first step in building an outbound B2B flow is to identify the B2B document that must be generated by the flow. As a part of this step, as shown in Figure 20–2, you must also analyze the requirements for the AIA services that must be built or extended to support the flow.

Figure 20–2 Step 1: Identifying B2B Document and Analyzing Service Requirements



As a part of this step, the integration development team performs the following tasks:

- Identify the B2B document protocol to be used.
- Identify the B2B document type to be used.
- Define the B2B document in Oracle B2B.
- Define the B2B document in AIA.
- Identify the AIA Enterprise Business Object (EBO), Enterprise Business Message (EBM), and Enterprise Business Service (EBS) to be used.

- Identify the message exchange pattern.
- Map the EBM, B2B document, and Application Business Message (ABM).
- Document and publish the mapping.

20.2.1 How to Identify the B2B Document Protocol

To develop a B2B flow, you first identify the B2B document to be used. Often, the choice of B2B document is influenced by the capabilities of the trading partners or e-commerce hub that must be integrated with your applications.

However, if you have the choice to choose a B2B document definition to be used for a particular integration use case, several industry-standard e-commerce standards exist today that vary in popularity in different geographies and industries.

Examples of B2B document protocols are Open Applications Group Integration Specification (OAGIS), Electronic Data Interchange (EDI) X12, Electronic Data Interchange For Administration, Commerce and Transport (EDIFACT), Health Insurance Portability and Accountability Act (HIPAA), EANCOM, RosettaNet, Universal Business Language (UBL), Health Level 7 (HL7), and 1SYNC.

Consider the following questions when choosing a B2B document protocol to integrate with your trading partners:

- Does the document protocol have the list of documents to support your integration needs?
- Does the document protocol have adequate support for your industry?
- Is the document protocol supported by your key trading partners?
- Is the document protocol popular in your geography?
- Is the document protocol based on open standards such as XML and XML Schema?
- Is the protocol support by your vendor, for example, Oracle?

Along with the B2B document protocol, the version of the B2B document must also be determined. Similar considerations as listed previously can be used to determine the B2B document protocol version.

20.2.2 How to Identify the B2B Document Type and Definition

After you have selected the B2B document protocol, the next task is to identify the B2B document that meets the specific B2B integration need. Each document protocol supports a list of B2B documents or document types. Refer to the protocol documentation and document usage guidelines to identify the specific document that is best-suited to support your specific business flow. Document usage can vary by industry and geography.

For example, you must implement a B2B flow to send purchase orders electronically to your supplier as part of your procurement business flow. The EDI X12 is chosen as the B2B document protocol based on your integration needs. The EDI X12 document protocol recommends that the 850 (Order) document be used for this integration requirement.

After the B2B protocol, document, and version are identified, you must obtain official versions of the following artifacts related to the document:

- Documentation and usage guidelines, which describe the B2B document in detail, including individual elements.

- Sample XML instances, which illustrate the usage of the B2B document.
- XML Schema definition of the B2B document.

For XML-based B2B document protocols, internet resources pertaining to the standard can be used as the source for downloads of the official user documentation, XML schemas, and sample document instances. For example, the OAGIS website (www.oagi.org) provides all of this information for every major release of the OAG document standards.

You can also obtain the official artifacts listed previously as a part of the B2B Document Editor software, which is part of Oracle Fusion Middleware.

For more information about creating guideline files, see "Creating Guideline Files" in *Oracle Fusion Middleware User's Guide for Oracle B2B.*

For non-XML protocols such as EDI and flat-file, Oracle B2B provides translation capabilities to convert the documents from non-XML formats to XML. However, this XML is an intermediate XML definition of the B2B document, which still must be transformed to the AIA EBM. You can export the B2B document in Oracle B2B Integration format using the B2B Document Editor to obtain the XML schema representation of this intermediate XML for use with the AIA B2B Connector Service (B2BCS).

Supporting Document Variations

As an integration best practice, AIA recommends that the standard B2B document definition be used as-is for B2B integration. Frequently, however, the B2B document definition must be modified to accommodate additional elements to meet custom integration requirements.

If such variations to the published standard are required, then the B2B document guideline and schemas obtained in the previous step must be modified to include these additional details.

20.2.3 How to Define the Document in Oracle B2B

After you have identified the B2B document type and definition, you must define the document in Oracle B2B. This is a prerequisite to using these documents in trading partner agreements.

For more information about creating document definitions, see "Creating Document Definitions" in *Oracle Fusion Middleware User's Guide for Oracle B2B.*

20.2.4 How to Define the Document in AIA

Per AIA integration best-practice recommendations, all reusable SOA artifacts, such as XML Schema and WSDL files, are hosted centrally in the SOA metadata store. Centrally hosting reusable artifacts and referencing them from the AIA services helps improve the performance and ease-of-maintenance of AIA services. The SOA metadata store provides caching and clustering capabilities that are also leveraged when you use it as the central store.

The XML schema definitions of B2B documents that are used by the B2B connector services are also hosted in the metadata store.

To deploy the B2B documents to the metadata store:

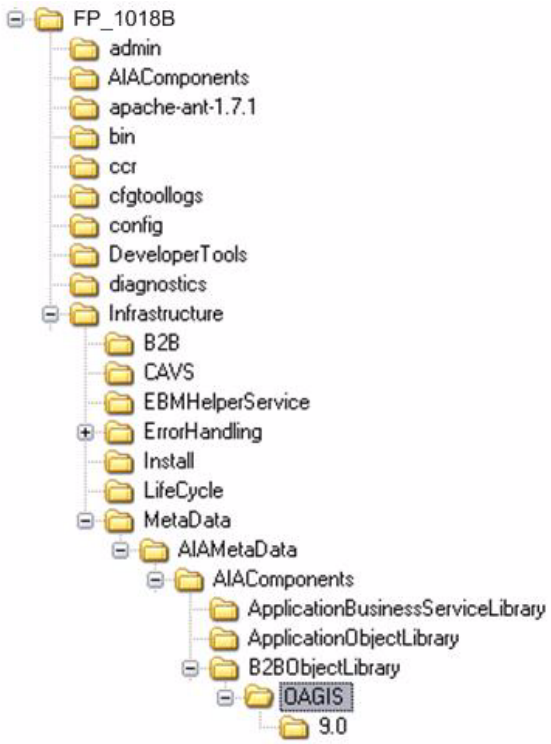
1. Create a directory that corresponds to the B2B standard in the B2BObjectLibrary.

2. Copy the XML schema files and directories that define the B2B document types to this location.

   The B2BObjectLibrary directory is a child directory of $AIA_HOME/MetaData/AIAMetaData/AIAComponents.

   For example, as shown in Figure 20–3, create directory *OAGIS* to which to copy OAG business object definitions.

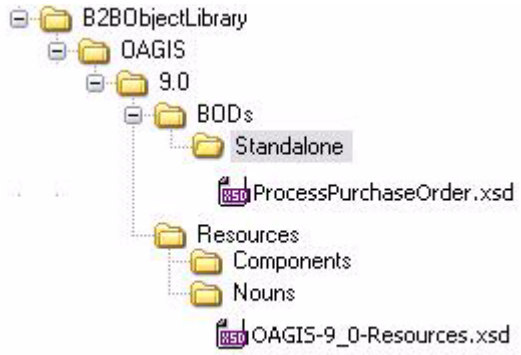
Figure 20–3 OAGIS Directory Created to Store OAG Business Object Definitions



3. Copy the XML schema files that correspond to the B2B document type under this folder.

   For example, as shown in Figure 20–4, copy the OAGIS ProcessPurchaseOrder.xsd file and all of its supporting schemas to this folder location, while also preserving the relative directory structure.

Figure 20–4 Corresponding XML Schema Files Stored in the OAGIS Directory



4. Deploy the new contents of the B2B Object Library to the SOA metadata store. To do this:

   a. Edit the $AIA_HOME/config/UpdateMetaDataDeploymentPlan.xml file, as shown in Figure 20–5, to specify that the directory B2BObjectLibrary has to be updated in Oracle Metadata Services (MDS).

Figure 20–5 UpdateMetaDataDeploymentPlan.xml Edited to Update the B2BObjectLibrary in Oracle Metadata Services

```
<?xml version="1.0" standalone="yes"?>
<DeploymentPlan component="MetaData" version="3.0">
 <Configurations>
       <UpdateMetadata wlserver="fp" >
             <fileset dir="${AIA_HOME}/Infrastructure/MetaData/AIAMetaData">
                   <include name="AIAComponents/B2BObjectLibrary" />
             </fileset>
       </UpdateMetadata>
 </Configurations>
</DeploymentPlan>
```

   b. Source the file aiaenv.sh located under $AIA_HOME/bin. For example: `$source /slot/ems5813/oracle/AIA3_HOME/bin/aiaenv.sh`

   c. Change the directory to $AIA_HOME/Infrastructure/Install/config. For example: `$cd $AIA_HOME/Infrastructure/Install/config`.

   d. Issue the AIA command to deploy the metadata. For example: `$ant -f UpdateMetadata.xml`.

   e. Verify that the command was successful and that no errors occurred.

20.2.5 How to Identify the EBO, EBS, and EBM to Be Used

To be able to integrate the B2B documents using AIA, the next step is to identify the AIA EBO, EBM, and EBS to be used. Select the AIA EBO and EBM for which the description most closely matches the B2B document to which you must integrate.

For example, to develop an outbound B2B integration flow in which EDI 850 (Order) documents must be communicated to a supplier by a procurement application, you can use the following EBO, EBM, and EBS:

- EBO: PurchaseOrderEBO
- EBM: CreatePurchaseOrderEBM
- EBS operation: CreatePurchaseOrder

20.2.6 How to Design Mappings for the B2B Document

The next step is to analyze your business requirements and map the ABM to the EBM and the EBM to the B2B document.

Consider the following guidelines while developing mappings for B2B documents:

- Identify and clearly list the subset of target B2B document elements to which you must map based on your integration needs.
- Document and publish the list of B2B elements being mapped. For each element being mapped, document the following information:
  - Data type
  - Cardinality
  - Example value
  - Description
  - DVM name and columns used
  - Mandatory indicator

Consider the following guidelines while mapping identification and reference components in the B2B documents:

- Map universal identifiers if available and supported by your application. For example, map UPC codes for identifying Items and UDEX codes for identifying Item Categories.
- Map all available external identifiers used by your trading partners. For example, map the Vendor Part Number in outbound B2B documents that must be sent to your vendors.
- Map free-form attributes such as Description, Notes, Attachments, and so forth to enable users to communicate and provide more information to trading partners.
- While mapping reference components, map content from the referenced entity and not just the identification. For example, if a Ship To Location has to be mapped in an outbound Purchase Order document being sent to a supplier, in addition to the location identifiers, map the actual postal address and contact details of the Ship To Location.
- While mapping Identification type components, you can directly map the actual application IDs to the B2B document. This is unlike the application-to-application (A2A) use case in which an XREF is used to store the mapping between the participating application identifiers and the AIA identifiers.

  For example, assume that you are working with a SyncItem from an Inventory system to a trading partner. Assume that ITEM_001 is the item identifier in the source ABM. This value can be directly mapped as-is to the Identification element. Using an XREF has little additional value because the ID generated in the trading partner application because of the sync flow is not captured in the XREF. A direct mapping is preferable instead.
- Map ahead of current requirements. Even if your current integration use case may require that only a few elements be mapped, a best-practice recommendation is to map all the available fields that can be processed by your application. This allows the connectors to be reused for multiple use cases.
- Extend EBOs, if required. If the target B2B document contains attributes that must be mapped, but that are not available in the EBO, extend the EBO to include the target elements and map them.

- Capture the mapping in an easy-to-understand spreadsheet and keep it current when any changes must be made.

20.2.7 How to Publish Mapping to Trading Partners

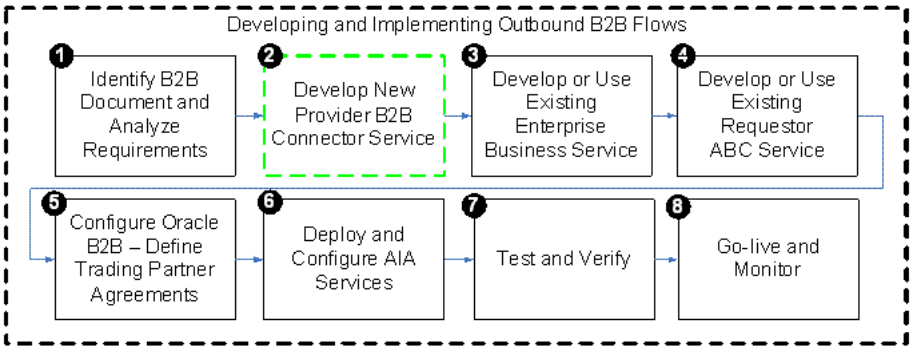
The next step is to share the mapping document with trading partners. This enables your trading partners to make any changes needed in their internal system to process outbound B2B documents received from you. Also, any feedback received from your trading partners can be used to revise the mapping design.

20.3 Step 2: Developing a New Provider B2B Connector Service

This section includes the following topics:

The next step, as shown in Figure 20–6, is to develop the provider B2BCS to support the outbound B2B integration flow.

Figure 20–6 Step 2: Developing a New Provider B2B Connector Service



The provider B2BCS is very similar to a provider Application Business Connector Service (ABCS), with the only difference being that it integrates with trading partners through Oracle B2B instead of integrating with an application. Hence, AIA recommends that you familiarize yourself with the design and development of ABCS (requester and provider) as well.

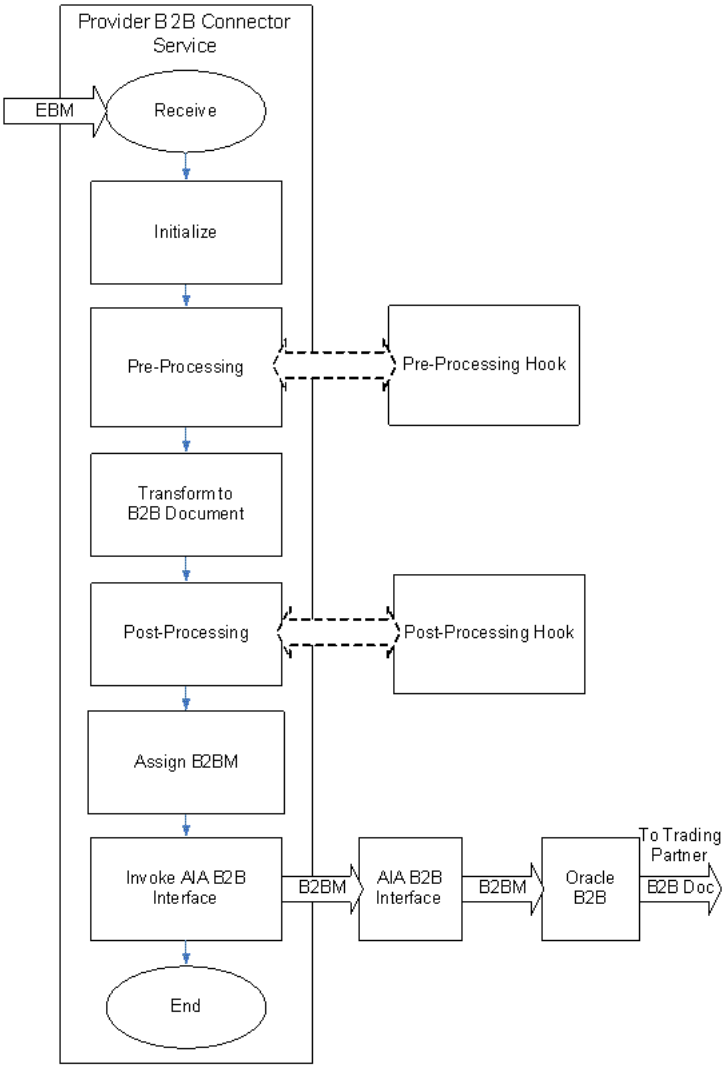
20.3.1 Introduction to a Provider B2B Connector Service

The key functionality provided by a provider B2BCS is to enable outbound B2B document integration by performing the following tasks:

- Transforming AIA EBMs into B2B documents.
- Invoking Oracle B2B to send these B2B documents to trading partners.

Figure 20–7 illustrates the processing that takes place in a simple fire-and-forget message exchange pattern-based provider B2BCS.

Figure 20–7 Processing Flow for a Fire-and-Forget Provider B2BCS



Step-by-step instructions for developing B2BCSs are provided in the following sections.

20.3.2 How to Identify the Message Exchange Pattern

Most B2B document flows can be modeled as asynchronous one-way calls.

For example, consider a use case in which a purchase order created in a Procurement application must be sent to a supplier using an outbound B2B document flow. Upon receipt of this Purchase Order document, the supplier processes it and sends back a Purchase Order Acknowledgement B2B document.

In this use case, even if a response is expected from the supplier in the form of the Purchase Order Acknowledgement document, the Procurement application can continue with its processing, which involves the purchase order being communicated to and processed by the vendor.

The Procurement application also generally can correlate the Purchase Order Acknowledgement document to the original purchase order because the Vendor is expected to provide the same purchase order identification used in the request in the Acknowledgement message. In this way, the request and response can be modeled as two asynchronous one-way flows.

However, a similar analysis must be performed for each B2B document flow being designed.

For more information about identifying message exchange patterns, see Section 14.3, "Identifying the MEP."

20.3.3 How to Develop a B2BCS Service Contract

First, define the service contract (WSDL) of the provider B2BCS. The service contract of the provider B2BCS specifies how it is invoked by an AIA flow. The service contract specifies the EBS operation being implemented, the input request type, and the message exchange pattern supported by the B2BCS.

For more information about creating WSDLs for ABCSs, see Section 14.2, "Defining the ABCS Contract."

Here are the recommended naming conventions for use in B2BCS WSDL definitions.

- WSDL File Name:
  - Naming guideline: <B2BStandard><Verb><EBO>ProvB2BCSImpl.wsdl
  - Example: X12UpdateSalesOrderProvB2BCSImpl.wsdl
- Service Namespace:
  - Naming guideline: http://xmlns.oracle.com/B2BCSImpl/{Core|Industry/<IndustryName>}/<B2BStandard><Verb><EBO>ProvB2BCSImpl/<ABCSVersion>
  - Example: http://xmlns.oracle.com/B2BCSImpl/Core/X12UpdateSalesOrderProvB2BCSImpl/V1

    Ensure that the ABCS Service version is independent of the B2B document/standard version.

    For more information about recommendations on versioning AIA services, see Section 16.8, "Versioning ABCS."
- Service Name:
  - Naming guideline: <B2BStandard><Verb><EBO>ProvB2BCSImpl
  - Example: X12UpdateSalesOrderProvB2BCSImpl
- Port Type Name:
  - Naming guidelines: <B2BStandard><Verb><EBO>ProvB2BCSImplService
  - Example: X12UpdateSalesOrderProvB2BCSImplService
- Operation Name:
  - Naming guideline: <Verb><EBO>

    - Example: UpdateSalesOrder
- Request Message Name:
    - Naming guideline: <Verb><EBO>ReqMsg
    - Example: UpdateSalesOrderReqMsg
- Request Message WSDL part element:
    - Naming guideline: <EBM>
    - Example: <wsdl:part name="UpdateSalesOrderEBMelement=" salesordebo: UpdateSalesOrderEBM"/>
- Imported Schemas:
    - Naming guideline:

      oramds:/apps/AIAMetaData/AIAComponents/EnterpriseObjectLibrary/Core/Common/V2/Meta.xsd

      oramds:/apps/AIAMetaData/AIAComponents /B2BObjectLibrary/<B2BMessageSchema>

      oramds:/apps/AIAMetaData/AIAComponents /EnterpriseObjectLibrary/<EBM Schema>
    - Example:

      oramds:/apps/AIAMetaData/AIAComponents/EnterpriseObjectLibrary/Core/Common/V2/Meta.xsd

      oramds:/apps/AIAMetaData/AIAComponents/EnterpriseObjectLibrary/Core/EBO/SalesOrder/V2/SalesOrderEBM.xsd

      oramds:/apps/AIAMetaData /AIAComponents/B2BObjectLibrary/X12/4010/Oracle/855_4010.xsd
- Imported WSDLs:
    - Naming guideline: oramds:/apps/AIAMetaData/AIAComponents/UtilityArtifacts/RuntimeFault.wsdl
    - Example: oramds:/apps/AIAMetaData/AIAComponents/UtilityArtifacts/RuntimeFault.wsdl

20.3.4 How to Store a WSDL in the Oracle Metadata Repository

As a SOA best practice, AIA recommends that all shared service artifacts, such as WSDL and XSD files, be stored in a central location that can be accessed by multiple services.

All of the AIA-shared artifacts are stored in Oracle Metadata Services (MDS). Storage of shared artifacts in MDS not only makes them globally accessible, but also enables AIA to leverage features in MDS that support caching and clustering.

Provider B2BCS WSDLs are stored in the following location in MDS: /apps/AIAMetaData/AIAComponents/B2BServiceLibrary/Connectors/wsdls/<B2B_STANDARD> /ProviderB2BCS/ <VERSION> / <B2B_STANADRD> <VERB> <OBJECT> ReqB2BCSImpl.wsdl

To store a WSDL in MDS:

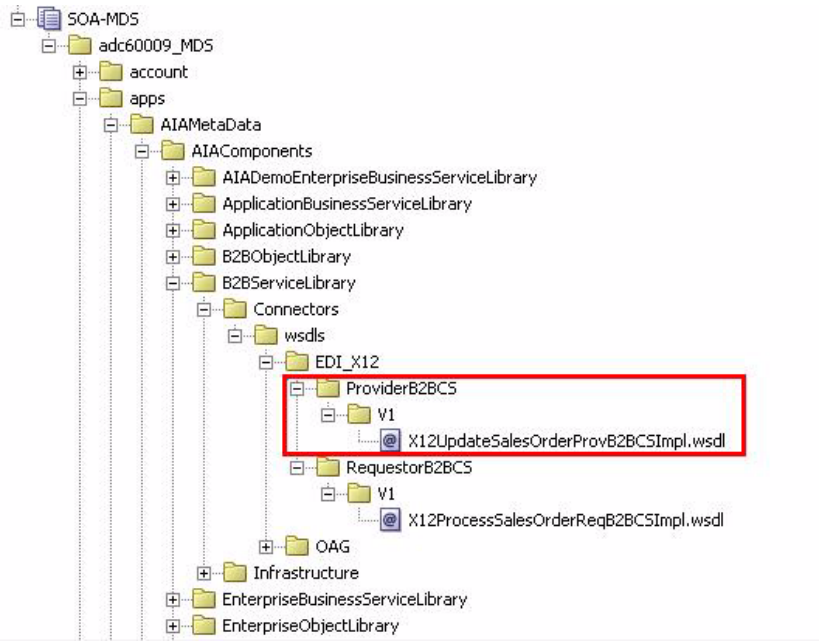
1. Copy the handcrafted WSDL to the following location under AIA_HOME: $AIA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/AIACompone

nts/B2BServiceLibrary/Connectors/wsdls/ <B2B_STANDARD> /ProviderB2BCS/ <VERSION> / <wsdl file> .wsdl

2. Run the UpdateMetaDataDP.xml present at $AIA_HOME/aia_instances/$INSTANCE_NAME/config/UpdateMetaDataDP.xml.

3. Using a SOA-MDS server connection to the MDS, verify that the AIAMetadata has been populated, as shown in Figure 20–8.

Figure 20–8 AIA Metadata in MDS



4. You can now access the WSDL using a URL similar to the following: oramds:/apps/AIAMetaData/AIAComponents/B2BServiceLibrary/Connectors/wsdls/EDI_X12/ProviderB2BCS/V1/ X12UpdateSalesOrderProvB2BCSImpl.wsdl

20.3.5 How to Develop a B2B Connector Service

The next step in the process is to develop the B2BCS. The provider B2BCS WSDL created in the previous step is used as the interface while you are developing the concrete B2BCS.

Because the Service Constructor does not support the autogeneration of B2B services, use Oracle JDeveloper to develop the B2BCS. Develop a composite with a BPEL process based on the abstract WSDL created in the previous step.

Following are the key activities that must be developed in the B2BCS implementation BPEL.

To develop a B2BCS:

1. Define variable <EBM>_Var.

   This is the input variable to the BPEL process and is used in the receive activity.

2. Define variable EBM_HEADER of type corecom:EBMHeader.

   This variable is used to store the AIA process context information and is used by the fault handling mechanism.

3. Define variable B2BM_HEADER of type corecom:B2BMHeader.

   This variable is used to store the B2B-specific AIA process context information and is used by the fault handling mechanism.

4. Define variable <B2BDoc>B2BM_Var of type corecom:B2BM.

   This variable is used as input to the AIAB2BInterface to send the outbound B2B document to Oracle B2B.

5. Define variable <B2BDoc>_Var using the external B2BDocument definition.

   This is used as the target of the transformation from EBM. This variable is then assigned to the <B2BDoc>B2BM_Var/Payload.

6. Define the partner link to the AIAB2BInterface.

   This service is invoked to send the B2B document to trading partners through Oracle B2B. The abstract WSDL to the AIAB2BInterface can be referenced from oramds:/apps/AIAMetaData/AIAComponents/B2BServiceLibrary/Infrastructure/wsdls/V1/B2BInterfaceV1.wsdl.

7. Assign the EBM_HEADER variable.

   Assign values of the EBM_HEADER local variable by copying all the values from the <input>EBM/EBMHeader element.

8. Assign the B2BM_HEADER variable.

   Assign the values as shown in Table 20–1.

Table 20–1 B2BM_HEADER Variable Assignment Values

Element	Value	Example
B2BMHeader/B2BMID	Map value from the EBMHeader/EBMID to the field.	111A1CD2121
B2BMHeader/B2BDocumentType/TypeCode	Populate with the B2B document type as defined in Oracle B2B.	X12_850
B2BMHeader/B2BDocumentType/Version	Populate with the B2B document revision configured in Oracle B2B.	4010
B2BMHeader/SenderTradingPartner/TradingPartnerID	Populate with value from the input EBM. / <XXX> EBM/EBMHeader/B2BProfile/SenderTradingPartner/TradingPartnerID	Acme
B2BMHeader/ReceiverTradingPartner/TradingPartnerID	Populate with value from input EBM. / <XXX> EBM/EBMHeader/B2BProfile/ReceiverTradingPartner/TradingPartnerID	GlobalChips

9. Transform the EBM into the B2B document.

   Use a transform activity to transform the EBM into the B2B Document format. Invoke an XSLT developed per the mapping created in the previous step.

10. Assign <B2BDoc>B2BM_Var.

    Assign values to the <B2BDoc>B2BM_Var as shown in Table 20–2.

Table 20–2 <B2BDoc>B2BM_Var Assignment Values

Element	Value
<B2BDoc>B2BM_Var//B2BMHeader	Contents of B2BM_HEADER variable
<B2BDoc>B2BM_Var//Payload	Contents of B2B Document variable: <B2BDoc>_Var

11. Invoke AIAB2BInterface.

    Invoke the AIAB2BInterface to send the document to trading partners. Use the <B2BDoc>B2BM_Var assigned in the previous step as input to the AIAB2BInterface.

    Invoke activity parameters as shown in Table 20–3.

Table 20–3 AIAB2BInterface Activity Invocation Parameters

Activity Name	InvokeAIAB2BInterface
Input	<B2BDoc>_B2BMVar
Partner Link	AIAB2BInterface
Port Type	B2BInterface
Operation	SendB2BMessage

12. Compile the BPEL process and ensure that no errors occur. You can use JDeveloper to deploy the BPEL process to a development server that is installed with AIA Foundation Pack.
13. Test the service by using sample EBM payloads as input.

20.3.6 How to Customize the AIA B2B Interface

As described previously in this document, the AIA B2B Interface is a reusable composite that can be used to integrate the B2B Connector Services with Oracle B2B. The AIA B2B Interface uses JMS-based internal delivery channels to integrate with Oracle B2B.

The AIA B2B Interface is a unified interface between the AIA and Oracle B2B layers for all B2BCSs. It supports operations to send B2B documents to Oracle B2B and receive B2B documents from Oracle B2B. The benefits of using the AIA B2B Interface to integrate with Oracle B2B are as follows.

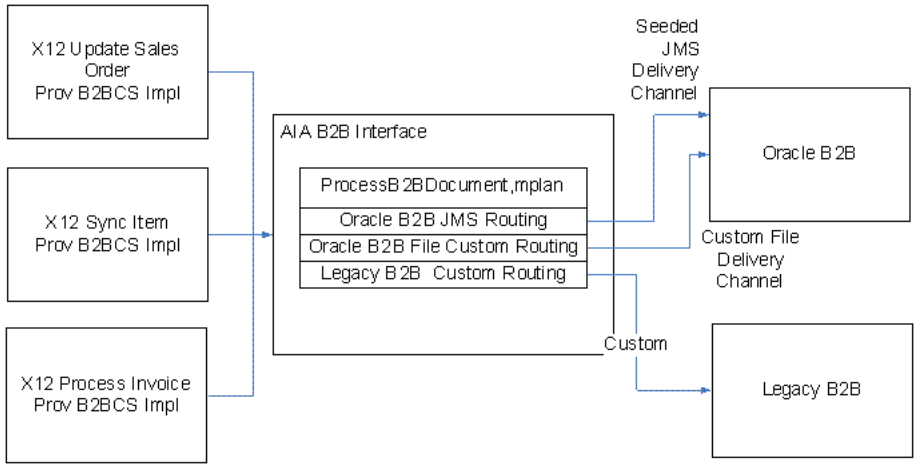
If the choice of internal delivery channel to be used for communication between B2B and AIA/SOA has to be changed at a later stage in the implementation, the B2BCS code need not be modified.

If you are using legacy or third-party B2B software, the AIA B2B Interface can be customized to add routing rules to send B2B documents to this legacy software. The B2BM_Var//B2BMHeader/GatewayID can be used as a filter expression to choose between Oracle B2B and third-party B2B software as the target for outbound documents. This allows the B2BCS to be shielded from change irrespective of the choice of B2B software.

Because this service is invoked by all outbound B2B document flows, you can customize this composite to support any common outbound functionality, such as audit logging, Oracle Business Activity Monitoring (BAM) instrumentation, and so forth.

Figure 20–9 illustrates the use of the AIA B2B Interface to support custom internal delivery channels (file) and legacy B2B connectivity.

Figure 20–9 AIA B2B Interface Support for Custom Internal Delivery Channels and Legacy B2B Connectivity



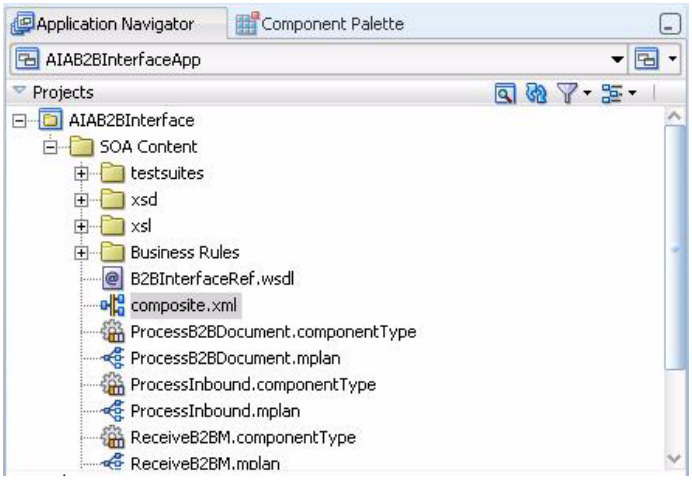
For example, you might have outbound Order documents being routed to a Fusion Middleware Oracle B2B component, and outbound Invoice documents being routed to an older B2B instance. To support this use case, define two routing rules: one for JMS invocation of Oracle B2B and another for Oracle Advanced Queuing invocation of the older B2B instance.

A given B2B document instance is likely to be routed to B2B using one of the multiple routing rules in the ProcessB2BDocument.mplan. In the highly unlikely event that a B2B document generated by the AIA layer must be routed to multiple B2B software instances, the routing rules in the ProcessB2BDocument.mplan mediator should be configured for sequential invocation.

To make customizations to the AIA B2B Interface:

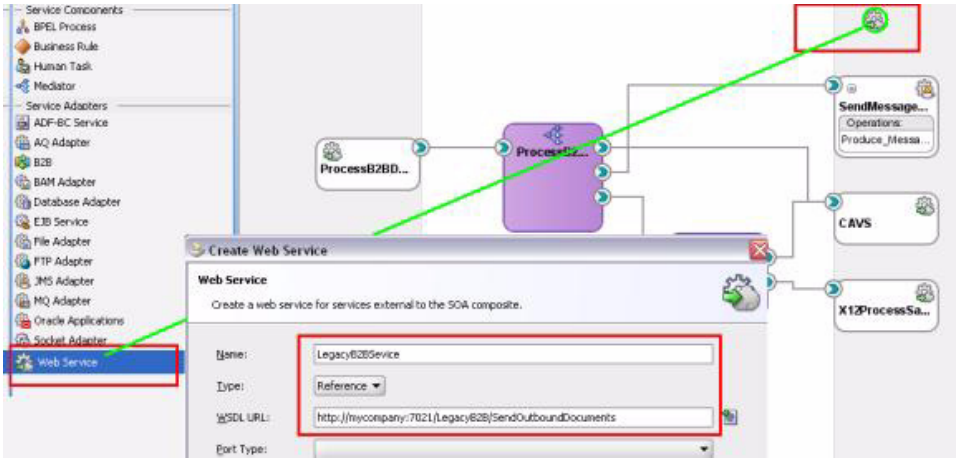
1. Open the SOA application, $AIA_HOME/Infrastructure/B2B/src/AIAB2BInterfaceApp, using Oracle JDeveloper.
2. Edit the AIAB2BInterface/composite.xml file using the Composite Editor, as shown in Figure 20–10.

Figure 20–10 AIAB2BInterface/composite.xml in the Composite Editor



3. Add a new adapter or web-service call required to interface to the third-party B2B software, as shown in Figure 20–11.

Figure 20–11 Addition of New Web Service Call to Interface with Third-Party B2B Software



4. Modify the AIAB2BInterface/ProcessB2BDocument.mplan file to add custom routing rules to route outbound B2B documents to third-party B2B software, as shown in Figure 20–12 and Figure 20–13.

Figure 20–12 AIAB2BInterface/ProcessB2BDocument.mplan in the Composite Editor

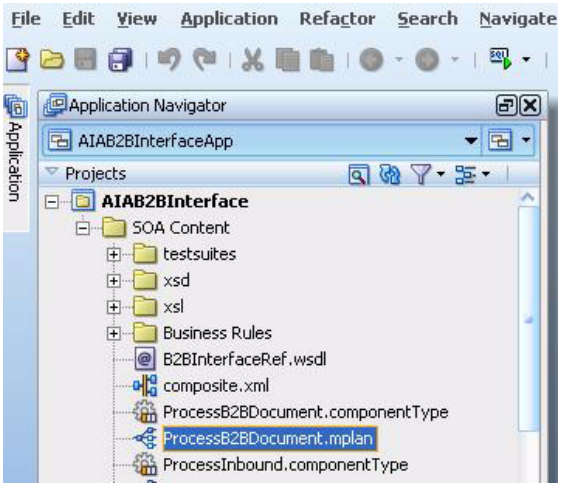
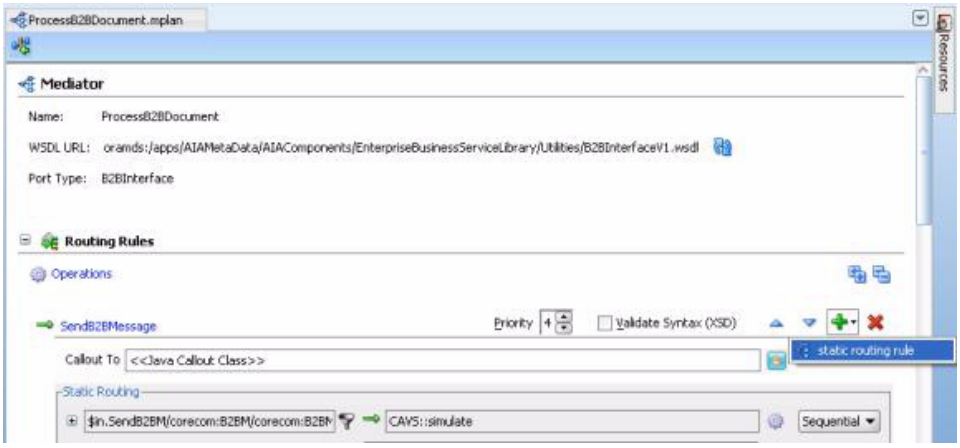


Figure 20–13 Addition of Custom Routing Rules to AIAB2BInterface/ProcessB2BDocument.mplan



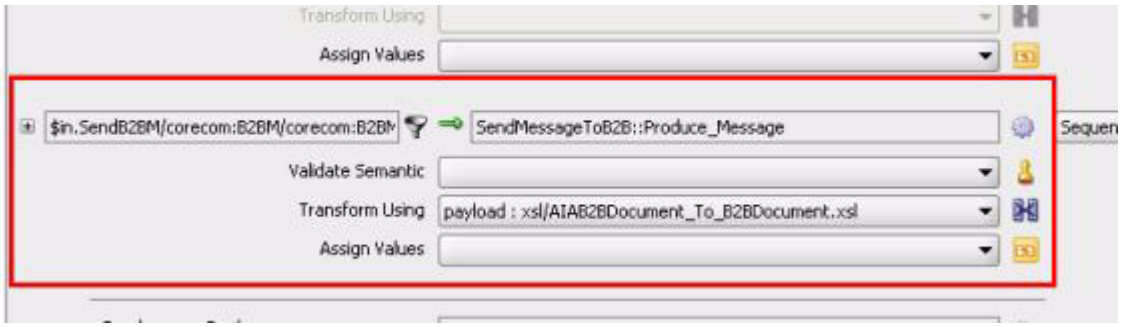
5. Use a condition expression, as shown in Example 20–1, on the GatewayID element to identify which B2B documents must be routed to the third-party B2B software.

Example 20–1 Condition Expression on the GatewayID Element

```
<condition language="xpath">
   <expression>$in.SendB2BM/corecom:B2BM/corecom:B2BMHeader/corecom:GatewayID =
    'MyB2BSoftware'</expression>
</condition>
```

6. Use a transformation step, assign step, or both, as required to provide input to the B2B software based on the input B2BM variable, ($in), as shown in Figure 20–14. Support for Oracle B2B is prebuilt and does not require any modifications.

Figure 20–14 Transformation Step to Provide Input to B2B Software



7. The input B2BM variable is received from the provider B2BCSs. Table 20–4 provides the information from the B2BM variable that can be used to provide input to the B2B software.

Table 20–4 B2BM Variable Information That Can Be Used to Provide Input to B2B Software

Input Element	Description	Usage While Integrating with Oracle B2B
$in.B2BM/B2BMHeader/SenderTradingPartner/TradingPartnerID	Sending trading partner in outbound flow.	Map to: JMSProperty.FROM_PARTY
$in.B2BM/B2BMHeader/ReceiverTradingPartner/TradingPartnerID	Receiving trading partner in outbound flow.	Map to: JMSProperty.TO_PARTY
$in.B2BM/B2BMHeader/B2BDocumentType/TypeCode	B2B Document Type used in the outbound flow. For example, 850.	Map to: JMSProperty. DOCTYPE_NAME
$in.B2BM/B2BMHeader/B2BDocumentType/Version	B2B Document Type version used in the outbound flow. For example, 4010.	Map to: JMSProperty. DOCTYPE_REVISION
$in.B2BM/B2BMHeader/B2BDocumentType/TypeCode/listAgencyID	B2B standard used in the outbound flow. For example, X12.	Not used.
$in.B2BM/B2BMHeader/B2BMID	Unique B2B document identifier. Unique across all trading partners.	Map to: JMSProperty.MSG_ID
$in.B2BM/B2BMHeader/Payload	Actual payload being sent to trading partners.	JMS text payload

8. The B2BCS is responsible for supplying the value of the GatewayID element to ensure that the desired value of the GatewayID is supplied before the AIAB2B Interface is invoked.
9. Save changes. Compile, test, and redeploy the AIA B2B Interface.

20.3.7 How to Annotate B2B Connector Services

To make key metadata about the B2BCS available to the Project Lifecycle Workbench and Oracle Enterprise Repository, the composite.xml file of the B2BCS Implementation SOA composite must be annotated in a specific manner.

For more information about the Project Lifecycle Workbench, see Chapter 3, "Working with Project Lifecycle Workbench."

For more information about annotating B2B services, see Chapter 12, "Annotating Composites."

To annotate B2B Connector Services:

1. Table 20–5 lists the annotation elements that must be added to the service element composite.xml, as described subsequently.

Table 20–5 Service Annotation Elements in composite.xml

Annotation Element	Description	Example
AIA/Service/InterfaceDetails/ServiceName	Name of EBS implemented by this provider B2BCS Impl	SalesOrderEBS
AIA/Service/InterfaceDetails/Namespace	Namespace of EBS implemented by this provider B2BCS Impl	http://xmlns.oracle.com/EnterpriseServices/Core/ SalesOrder/V2
AIA/Service/InterfaceDetails/ArtifactType	EnterpriseBusinessService	EnterpriseBusinessService
AIA/Service/InterfaceDetails/Service Operation	<Verb><EBOName>	UpdateSalesOrder
AIA/Service/ImplementationDetails/ArtifactType	ProviderB2BCSImplementation	ProviderB2BCSImplementation
AIA/Service/ImplementationDetails/ServiceOperation/Name	<Verb><EBOName>	UpdateSalesOrder
AIA/Service/ImplementationDetails/B2BDocument	Target B2B Document Type of this B2BCS	855
AIA/Service/ImplementationDetails/B2BDocumentVersion	Target B2B Document Version of this B2BCS	4010
AIA/Service/ImplementationDetails/B2BStandard	Target B2B Standard of this B2BCS	X12
AIA/Service/ImplementationDetails/B2BStandardVersion	Target B2B Standard Version of this B2BCS	4010

The following XML snippet, as shown in Figure 20–15, is an example of an annotated B2BCS called X12ProcessSalesOrderReqB2BCSImpl composite.xml.

Figure 20–15 Example Snippet of an Annotated B2BCS Called X12ProcessSalesOrderReqB2BCSImpl composite.xml

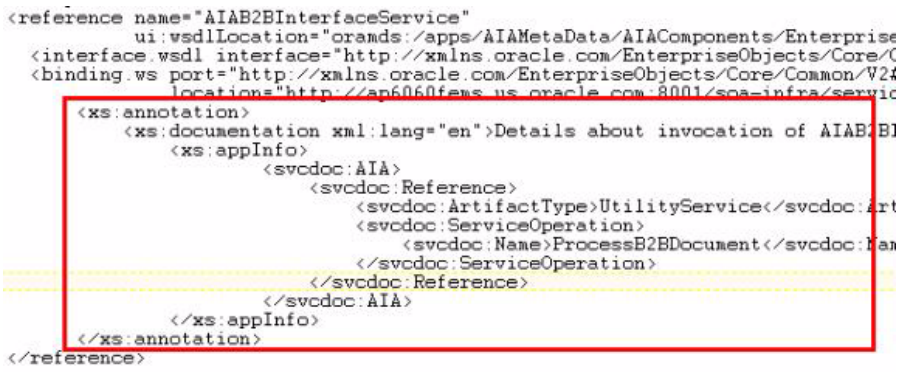
2. Table 20–6 lists AIA B2B interface utility service annotation elements in composite.xml.

Table 20–6 AIA B2B Interface Utility Service Annotation Elements in composite.xml

Annotation Element	Description
`AIA/Reference/ArtifactType`	Enter value `UtilityService`
`AIA/Reference/ServiceOperation/Name`	Enter value `ProcessB2BDocument`

The reference to the AIA B2B Interface utility service should be annotated in the composite.xml, as shown in Figure 20–16.

Figure 20–16 AIA B2B Interface Utility Service Annotation Elements in composite.xml snippet



3. The $AIA_HOME/Infrastructure/B2B/src/B2BConnectors/EDIX12B2BCSApp/X12Update SalesOrder ProvB2BCSImpl/composite.xml file has sample provider B2BCS Impl annotations for your reference.

20.3.8 How to Support Trading Partner-Specific Variants

This section includes the following topics:

- Section 20.3.8.1, "Supporting Trading Partner-Specific Custom Extensions"
- Section 20.3.8.2, "Supporting Trading Partner-Specific XSLTs"
- Section 20.3.8.3, "Supporting Trading Partner-Specific Document Types and Versions"

Frequently, in B2B implementations different trading partners must support different versions or mapping guidelines for the same B2B document. If a given B2B document must be sent to multiple trading partners, based on the version and mapping guideline determined for that trading partner, the B2BCS of the document can be built to support this trading partner-specific transformation logic. Multiple ways are available by which this can be achieved, as described in the following section.

20.3.8.1 Supporting Trading Partner-Specific Custom Extensions

If the trading partner-specific mappings are an addition to a common core mapping that remains unchanged, explore the possibility of using the custom XSLT template calls to have partner-specific mappings.

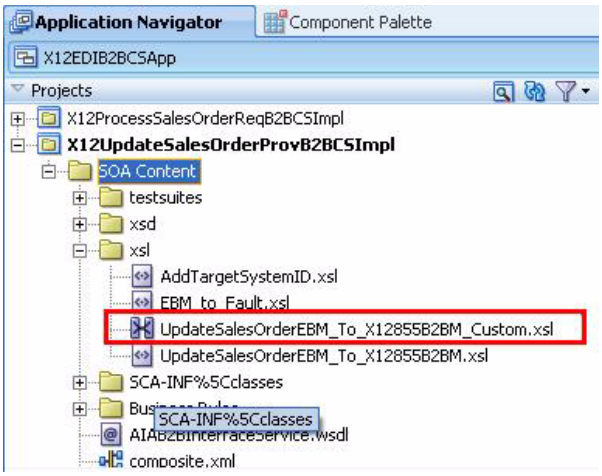
For more information about support for XSLT extension using callouts to custom XSLT templates, see Section 16.1, "Developing Extensible ABCS."

In short, at the end of every business component mapped in the XSLT file, an invocation to a custom XSLT template is made from the shipped B2BCSs, as shown in Figure 20–17.

Figure 20–17 Business Component Mapped in the XSLT, including an invocation from the Shipped B2BCS to a Custom XSLT template

The custom XSLT templates are defined in a custom XSLT file, as shown in Figure 20–18, which is included in the main XSLT.

Figure 20–18 Custom XSLT Templates Defined in a Custom XSLT File



During implementation, you can add additional mappings to this custom XSLT file. The trading partner ID can be passed as an input to the custom XSLT template call and can be used to conditionally map to target B2B elements, as shown in Figure 20–19.

Figure 20–19 Trading Partner ID Passed as an Input to the Custom XSLT Template Call

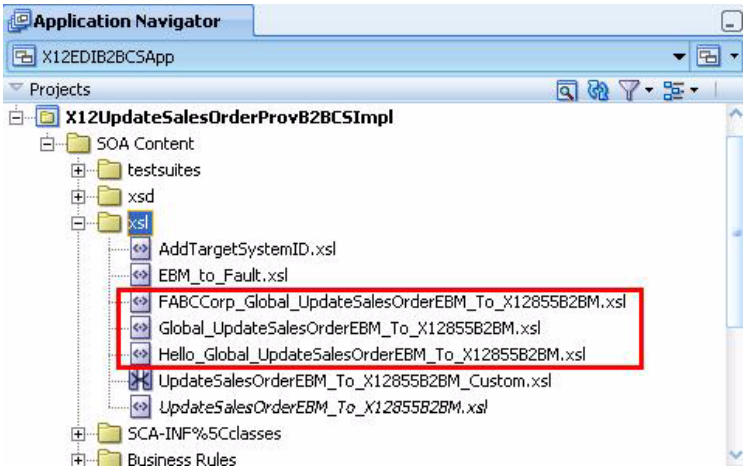
```
<xsl:template name="Segment-ST_ext">
    <xsl:param name="currentNode">
    <xsl:param name="$TradingPartnerID">
    <xsl:if test='$TradingPartnerID= "ABCCorp"'>
       <!-- Add custom mappings to Segement-ST for trading partner "ABC Corp" here.
    </xsl:if>
    <xsl:if test='$TradingPartnerID= "Hello Inc"'>
       <!-- Add custom mappings to Segement-ST for trading partner "Hello Inc" here.
    </xsl:if>
    <xsl:if test='$TradingPartnerID= "Global"'>
       <!-- Add custom mappings to Segement-ST for trading partner "Global" here.
    </xsl:if>
</xsl:template>
```

20.3.8.2 Supporting Trading Partner-Specific XSLTs

If the trading partner-specific mappings conflict with each other or if the partner-specific mappings must be added at arbitrary locations in the XML document and not just at the end of each mapping template, the previous approach of using the custom XSLT templates to define partner-specific mappings does not meet requirements.

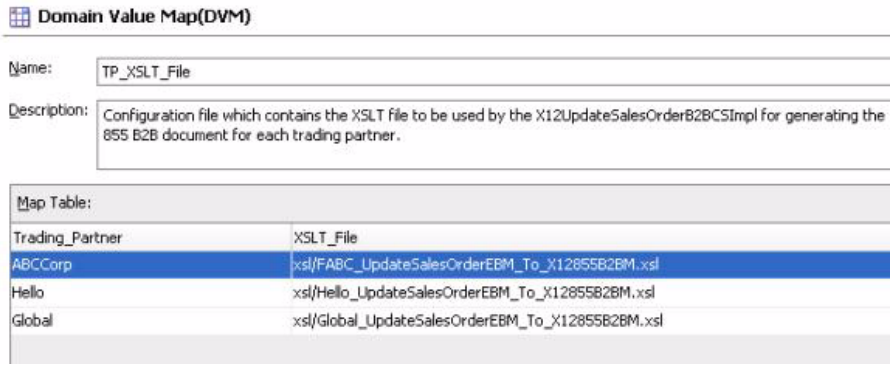
To meet such requirements, you can decide to develop one XSLT file for each trading partner that needs custom mappings. You can make a copy of a shipped XSLT file, edit it to include partner-specific mapping logic, and save it by using a partner prefix in the filename, as shown in Figure 20–20.

Figure 20–20 Copies of Shipped XSLT Files Edited to Include Partner-Specific Mapping Logic and Saved Using a Partner Prefix in the Filename



Next, you can define a configuration file that contains information about which XSLT file has to be used for each trading partner. For example, you can use a domain value mapping (DVM) file to store this configuration information, as shown in Figure 20–21.

Figure 20–21 DVM File Used to Map Trading Partners to XSLT Files



In the provider B2BCS Impl BPEL process, as shown in Figure 20–22, you can do a lookup on this DVM file, as shown in Figure 20–23, to obtain the XSLT filename, as shown in Figure 20–24.

Figure 20–22 Provider B2BCS Implementation BPEL Process Requiring Trading Partner-Specific XSLT

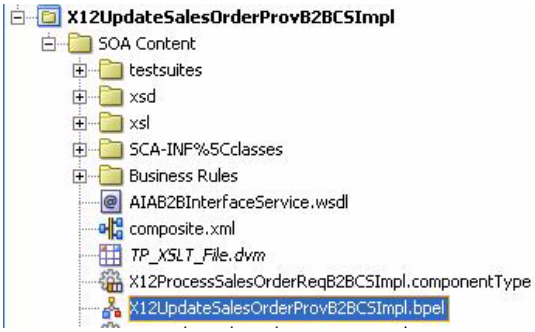


Figure 20–23 DVM Lookup to Obtain XSLT Filename

```
<assign name="GetXSLTFileName">
  <copy>
    <from expression='dvm:lookupValue("TP_XSLT_File.dvm","TradingPartner",
    <to variable="xslt_file_name"/>
  </copy>
</assign>
```

Figure 20–24 Retrieval of XSLT Filename

```
<assign name="XformUpdateSalesOrderEBMToX12Transaction855">
  <bpelx:annotation>
    <bpelx:pattern>transformation</bpelx:pattern>
  </bpelx:annotation>
  <copy>
    <from expression="ora:processXSLT(bpws:getVariableData('xslt_file_name',bp
    <to variable="EDIX12Transaction_855Variable"/>
  </copy>
</assign>
```

20.3.8.3 Supporting Trading Partner-Specific Document Types and Versions

Along with the need for trading partner-specific XSLTs, a different Document Type may possibly be used in Oracle B2B for defining the trading partner agreements for the same external B2B document.

For example, trading partner Global might need the 5010 version of the X12 855 B2B document and trading partner ABC Corp might need the 4010 version of the same document.

Using the approach described in the previous section, you can use the same provider B2BCS Implementation to generate B2B documents for both the trading partners.

However, while the AIA B2B Interface is being invoked based on the trading partner involved for the specific instance of the service, the corresponding B2B document version must be specified, 4010 for Global and 5010 for ABC Corp, for example.

To support these requirements, use a DVM file similar to the one used in the previous approach to store the B2B Document Type and Document Revision information for each trading partner, as shown in Figure 20–25. This information can be looked up by the BPEL process to populate the /B2BMHeader/B2BDocumentType/TypeCode and /B2BMHeader/B2BDocumentType/Version attributes.

Figure 20–25 DVM Used to Store Trading Partner B2B Document Type and Document Revision Information

Domain Value Map(DVM)

Name: TP_B2B_PARAMS_CONFIG

Description: Capture trading parnter specific B2B configuration

Map Table:

TRADING_PARTNER	B2B_DOCTYPE_NAME	B2B_DOCTYPE_REVISION
ABC Corp	850	5010
Global	850	4010

While the configuration files described previously can be created and stored locally within the Oracle JDeveloper project, a best-practice recommendation is to externalize these files, store them in MDS, and refer to them from your B2BCS Implementation BPEL project using oramds lookup.

Though this approach of using DVM files to store the B2B document preference and trading partner-specific transformation information works, the Trading Partner information between the application and these DVM files must be kept synchronized.

Each time a new trading partner is defined in the application, a corresponding record should be created in this DVM to store the B2B document preference of the trading partner. A well-established administrative process to manage these changes should be available.

20.3.9 How to Enable Error Handling

For more information about how to enable AIA services for error handling and recovery, see "Setting Up Error Handling" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

In short, the following steps must be taken:

- Associate a fault-policy.xml with the B2BCS composite.
- Invoke the AIAAsyncErrorHandlingBPELProcess for business faults, as shown in Figure 20–26 and Figure 20–27.

Figure 20–26 B2BCS Composite Defined to Invoke AIAAsyncErrorHandlingBPELProcess

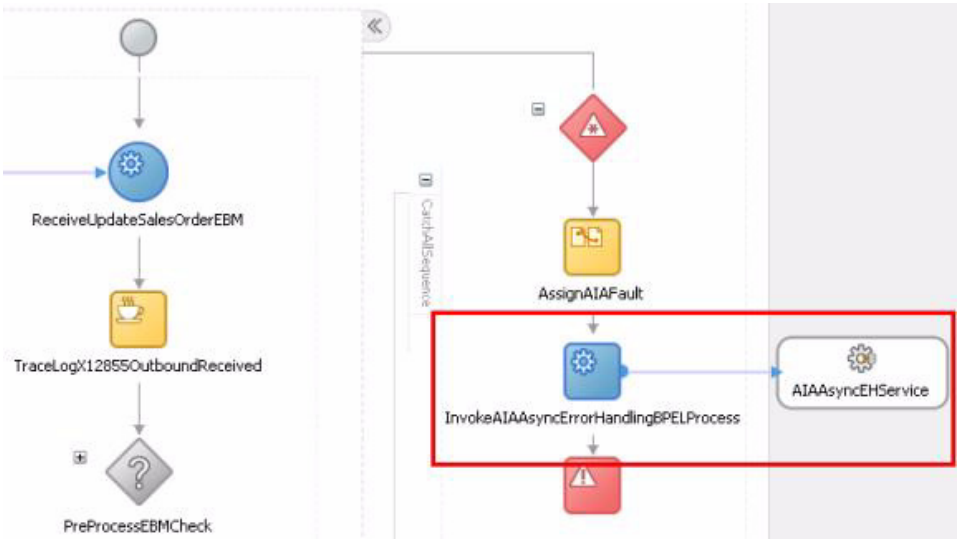


Figure 20–27 Invocation of the AIAAsyncEHService

```
    </assign>
          <invoke name="InvokeAIAAsyncErrorHandlingBPELProcess"
                  portType="aiaasynceh:AIAAsyncErrorHandlingBPELProcess" inputVariable="AIAA
                  partnerLink="AIAAsyncEHService"
                  operation="initiate"/>
          <throw name="ThrowAIAFault" faultName="corecom:Fault" faultVariable="AIAFaultMessa
       </sequence>
    </catchAll>
</faultHandlers>
```

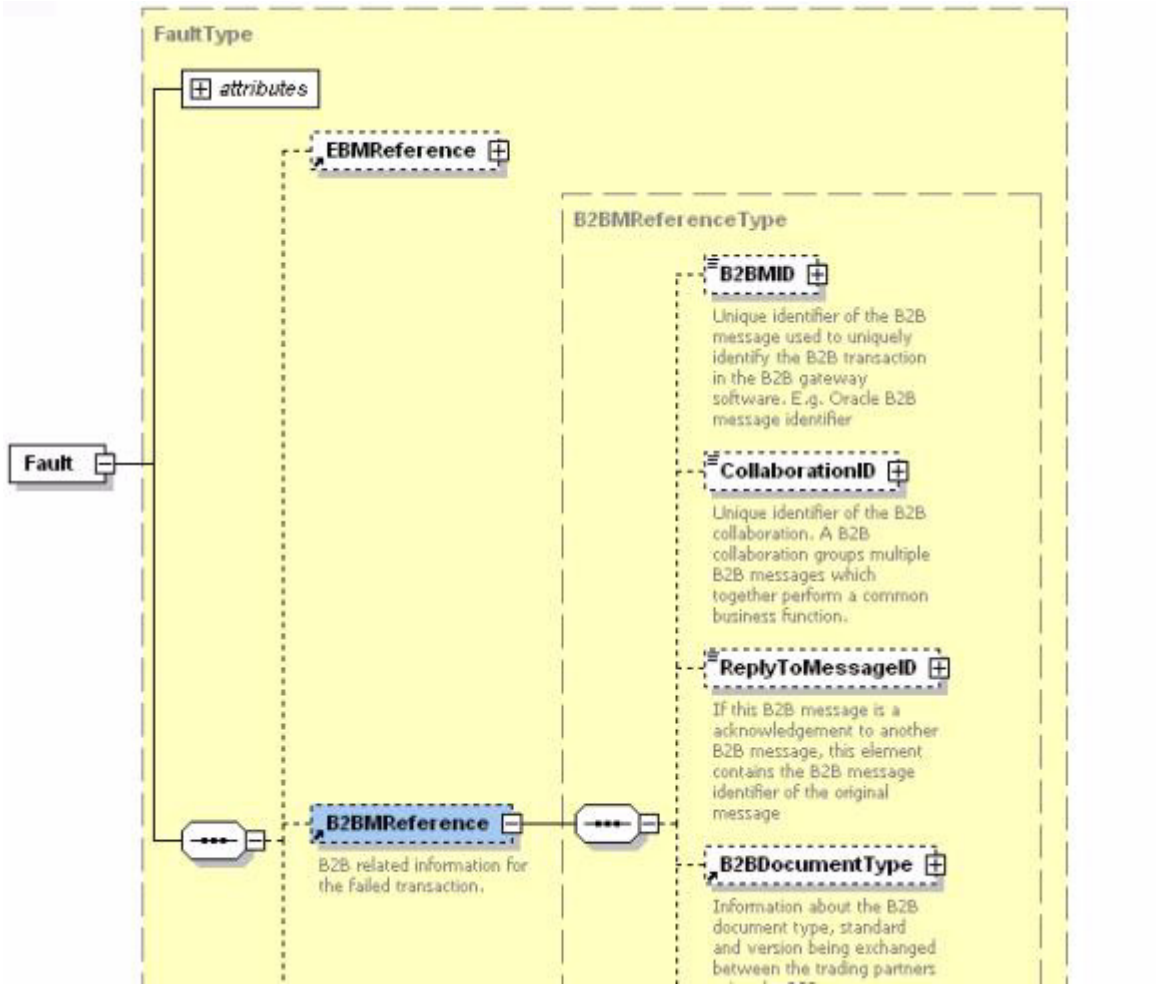
While invoking the AIAAsyncErrorHandlingBPELProcess, the following B2B-specific elements in the fault schema can be populated as described in Table 20–7.

Table 20–7 B2B-Specific Elements in the Fault Schema That Can Be Populated by AIAAsyncErrorHandlingBPELProcess

Fault Element Schema	Description	Example
Fault/B2BMReference/B2BMID	Unique identifier of the B2B document	13232325
Fault/B2BMReference/B2BDocumentType/TypeCode	Document type of the B2B document being generated by the provider B2BCS	855
Fault/B2BMReference/B2BDocumentType/Version	Document version of the B2B document being generated by the provider B2BCS	4010
Fault/B2BMReference/B2BDocumentType/TypeCode/@listAgencyID	Standard of the B2B document being generated by the provider B2BCS	X12
Fault/B2BMReference/GatewayID	Name of the B2B software being used	Oracle B2B
Fault/B2BMReference/SenderTradingPartner/TradingPartnerID	Sender trading partner, mapped from the EBMHeader	MyCompany
Fault/B2BMReference/ReceiverTradingPartner/TradingPartnerID	Receiver trading partner, mapped from the EBMHeader	Global

Figure 20–28 provides the B2B-specific elements in the corecom:Fault.

Figure 20–28 B2B-Specific Elements in the corecom:Fault

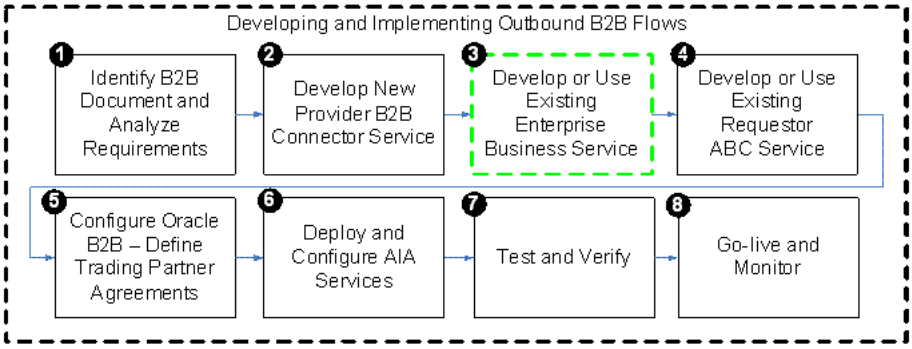


The B2B details of the failed AIA service that are available in the fault instance are logged and available for debugging the failed flow.

20.4 Step 3: Developing or Extending an Existing Enterprise Business Service

The next step, as shown in Figure 20–29, is to develop a new EBS or extend an existing EBS to invoke the provider B2BCS developed in the previous step. For example, the UpdateSalesOrderEBS has to be developed or modified to invoke the X12UpdateSalesOrderProvB2BCSImpl process.

Figure 20–29 Step 3: Developing or Extending an Existing Enterprise Business Service



For more information about creating a new EBS, see Chapter 13, "Designing and Developing Enterprise Business Services."

20.4.1 How to Route Based on Trading Partner B2B Preferences

A B2B implementation may possibly exist in which different trading partners may require that the same EBM information be sent using different B2B document protocols.

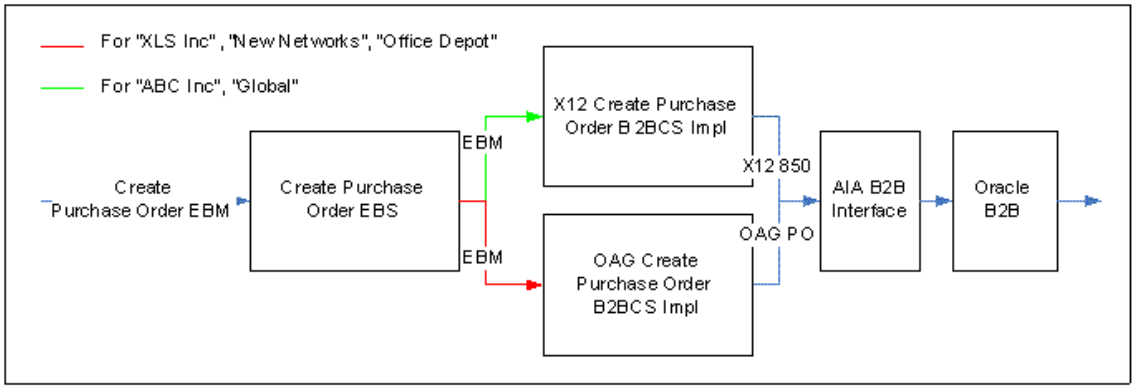
For example, suppliers XLS Inc., New Networks, and Office Systems may require that new purchase orders be sent using the OAG Process Purchase Order document format, while suppliers ABC Inc. and Global may require that new purchase orders be sent using the X12 850 document format.

To support the needs of this implementation, two B2BCSs must be developed, as shown in Figure 20–30:

- OAGCreatePurchaseOrderB2BCSImpl, which transforms the CreatePurchaseOrderEBM into an OAG Process Purchase Order B2B document.
- X12CreatePurchaseOrderB2BCSImpl, which transforms the CreatePurchaseOrderEBM into an X12 850 B2B document.

The CreatePurchaseOrderEBM implementation must include routing rules to invoke both of these B2BCSs.

Figure 20–30 B2B Implementation in which Trading Partners Need the Same EBM Data Sent Using Different B2B Document Protocols



If a small number of trading partners are involved, the filter expression in the EBS routing rules to each of the B2BCSs can be used to route the EBM based on the trading partner involved, as shown in Figure 20–31.

Figure 20–31 Filter Expression in the EBS Routing Rule to Each B2BCS Used to Route the EBM Based on the Trading Partner

```
<operation name="CreatePurchaseOrder" deliveryPolicy="AllOrNothing" priority="4" validateSchema="false">
   <switch>
      <case executionType="direct"
         name="OAGCreatePurchaseOrderProvB2BCS.CreatePurchaseOrder">
         <condition language="xpath">
            <expression>(($in.CreatePurchaseOrderEBM/EBMHeader/B2BProfile/ReceiverTradingPartnerID='XLS Inc') or
                        ($in.CreatePurchaseOrderEBM/EBMHeader/B2BProfile/ReceiverTradingPartnerID='New Networks') or
                        ($in.CreatePurchaseOrderEBM/EBMHeader/B2BProfile/ReceiverTradingPartnerID='Office Deport'))
            </expression>
         </condition>
         <action>
            <transform/>
            <invoke reference="OAGCreateSalesOrderProvB2BCSImpl"
                    operation="CreatePurchaseOrder"/>
         </action>
      </case>
      <case executionType="direct"
         name="X12CreatePurchaseOrderProvB2BCS.CreatePurchaseOrder">
         <condition language="xpath">
            <expression>(($in.CreatePurchaseOrderEBM/EBMHeader/B2BProfile/ReceiverTradingPartnerID='ABC Inc') or
                        ($in.CreatePurchaseOrderEBM/EBMHeader/B2BProfile/ReceiverTradingPartnerID='Global'))
            </expression>
         </condition>
         <action>
            <invoke reference="X12CreateSalesOrderProvB2BCSImpl"
                    operation="CreatePurchaseOrder"/>
         </action>
      </case>
   </switch>
</operation>
```

However, if a large number of trading partners are involved, the trading partner's B2B preferences can be stored in an external configuration file, for example, in a DVM file, as shown in Figure 20–32.

Figure 20–32 DVM Used to Store Trading Partner B2B Preferences

Domain Value Map(DVM)

Name: CreatePurchaseOrder_TP_B2BConfig_DVM

Description: This DVM file contains the B2B document format supported by different trading partners to receive new Purchase Orders.

Map Table:

TRADING_PARTNER	B2B_DOC_TYPE
Global	X12_850
XLS Inc	OAG_PROCESS_PO
New Networks	OAG_PROCESS_PO
Office Depot	OAG_PROCESS_PO
ABC Inc	X12_850

This configuration can be looked up during run time by the EBS implementation to determine the target provider B2BCS to be invoked, as shown in Figure 20–33.

Figure 20–33 EBS Implementation Lookup to Determine the Target Provider B2BCS to be Invoked

Though this approach works, a need exists to keep the trading partner information between the application and these DVM files synchronized.

Each time a new trading partner is defined in the application, a corresponding record should be created in this DVM to store the B2B document preference of the trading partner. A well-established administrative process to manage these changes should exist.

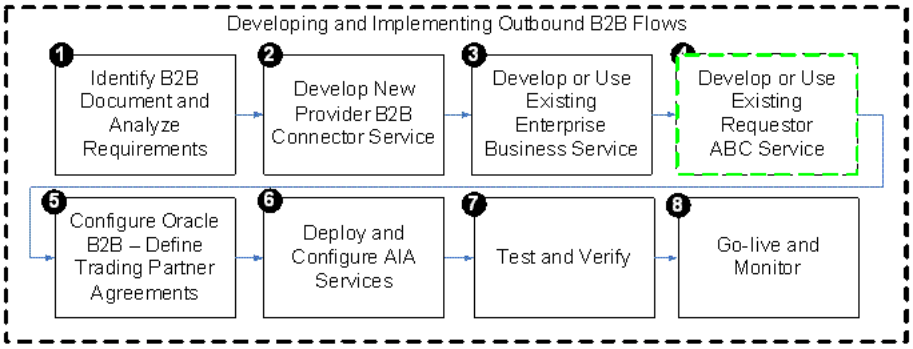
20.5 Step 4: Developing or Extending an Existing Requester ABCS

This section includes the following topics:

- Section 20.5.1, "What You Must Know About Message Exchange Patterns"
- Section 20.5.2, "What You Must Know About Transformations"

The next step, as shown in Figure 20–34, is to develop a new requester ABCS or extend an existing requester ABCS.

Figure 20–34 Step 4: Developing or Extending an Existing Requester ABCS



The requester ABCS is the AIA service that is triggered from an application UI action or business event and is the first step in the outbound B2B flow. The requester ABCS acts on behalf of the triggering application in the AIA layer and invokes the AIA EBS.

For more information about how to design and construct a requester ABCS, see Chapter 14, "Designing Application Business Connector Services" and Chapter 15, "Constructing the ABCS."

The requester ABCS always invokes an EBS implementation and hence can be used in both A2A and B2B integration scenarios. It is a best-practice recommendation to anticipate that the requester ABCSs can be potentially used both in A2A and B2B integration scenarios.

Consider the following information when developing requester ABCSs to be used in B2B integration flows.

20.5.1 What You Must Know About Message Exchange Patterns

B2B integration flows are primarily processed asynchronously. Thus, while evaluating the message exchange pattern for developing new requester ABCSs, you should consider the possibility of their being used in B2B flows.

20.5.2 What You Must Know About Transformations

One of the key tasks of the requester ABCS is to transform the ABM into the canonical EBM and use the EBM as input to invoke the AIA EBS.

While developing this transformation from ABM to EBM, remember that the transformation should support the generation of an EBM payload that can be routed to and used by a B2BCS. The B2B mapping guidelines described in Step 1: Identifying the B2B Document and Analyzing Requirements should be considered to identify the fields in the EBM that are required by the B2BCS to create the B2B documents.

A few other considerations include the following points:

- Follow the AIA recommendation to always map all available fields, instead of just a subset of fields required for a specific integration scenario.
- External global identifiers such as UPC Product Codes and DUNS should be mapped along with application internal identifiers.
- Reference components in the EBM should be fully mapped. For example, you should map not just a Location Identifier, but also the actual address details, because a remote trading partner may not be able to resolve the location identifier to an actual location.

- Table 20–8 provides the fields in the EBM header must be mapped so that they can be used to perform trading partner-specific routing in the EBS layer, as described in the previous section.

Table 20–8 EBM Header Elements that Must be Mapped to Enable Trading Partner-Specific Routing in the EBS Layer

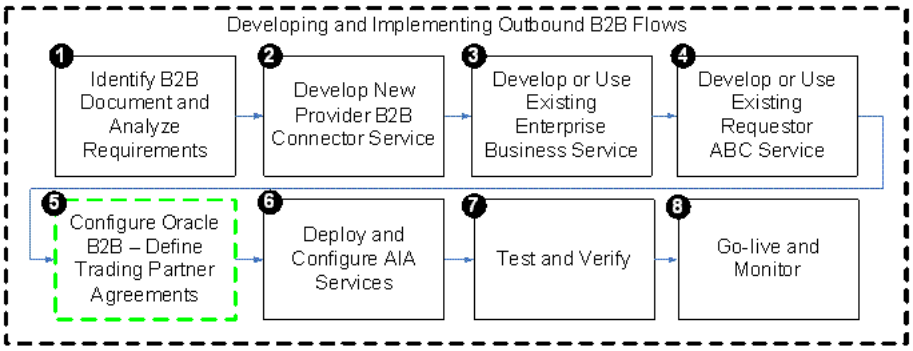
EBM Header Element	Description	Example Value
/EBMHeader/B2BProfile/SenderTradingPartner/TradingPartnerID	ID of the sending trading partner as defined in Oracle B2B. For outbound flows, this is the host" trading partner.	*MyCompany*
/EBMHeader/B2BProfile/ReceiverTradingPartner/TradingPartnerID	ID of the receiving trading partner as defined in Oracle B2B. For outbound flows, this is the remote trading partner.	*Globe Inc*

At the end of this step, all of the required AIA services for developing an outbound B2B integration flow are ready.

20.6 Step 5: Configuring Oracle B2B and Defining Trading Partner Agreements

The next step, as shown in Figure 20–35, is to create trading partner agreements in Oracle B2B.

Figure 20–35 Step 5: Configuring Oracle B2B and Defining Trading Partner Agreements



For more information about how to define trading partners and associate B2B capabilities with them, see "Configuring Trading Partners" in *Oracle Fusion Middleware User's Guide for Oracle B2B*.

In addition, for EDI-based outbound B2B flows, Oracle B2B can be configured for batching outbound documents. Parameters such as the batch size and time-out can be configured in Oracle B2B without any changes to the AIA layer.

The values used for naming trading partners in Oracle B2B should match the values that are sent from the requester ABCS using the following fields:

- /EBMHeader/B2BProfile/SenderTradingPartner/TradingPartnerID (for Host Trading Partner Name)
- /EBMHeader/B2BProfile/ReceiverTradingPartner/TradingPartnerID (for Remote Trading Partner Name)

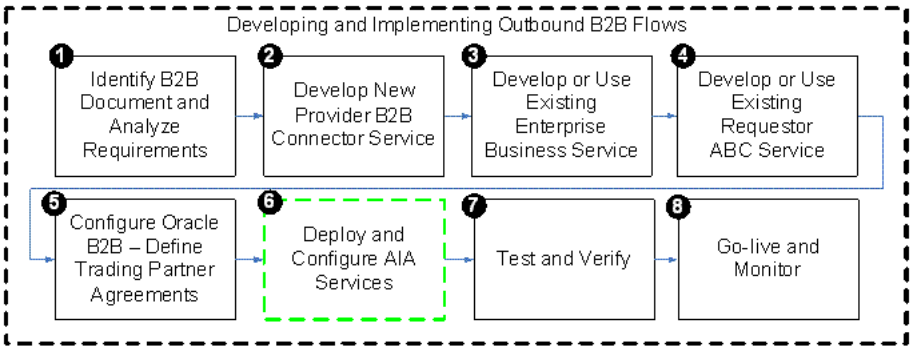
If the values provided by the source application for these fields and the trading partner names in Oracle B2B do not match, you can maintain a DVM file that contains the mapping between the source application's trading partner IDs and trading partner names of Oracle B2B.

This DVM can be looked up during the transform from the ABM to the EBM in the requester ABCS to populate the SenderTradingPartner/TradingPartnerID and ReceiverTradingPartner/TradingPartnerID fields.

20.7 Step 6: Deploying and Configuring AIA Services

The next step, as shown in Figure 20–36, is to deploy the AIA services. You can deploy the services to a target Oracle SOA server using Oracle JDeveloper.

Figure 20–36 Step 6: Deploying and Configuring AIA Services



If any DVM and configuration files are used by the AIA services required for the outbound integration, you must enter data corresponding to your B2B configuration.

You can also use the Project Lifecycle Workbench application to create a bill of material XML file for the AIA project, which can be used to autogenerate a deployment plan. This deployment plan can be used to deploy the AIA services and resources that form the integration project in multiple development, test, and production environments.

For more information about generating bills of material, see Chapter 3, "Working with Project Lifecycle Workbench."

For more information about generating deployment plans, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

In addition, configure the AIA Error Handling framework and set up appropriate roles to be notified of errors in AIA flows.

For more information about error handling, see "Setting Up Error Handling" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

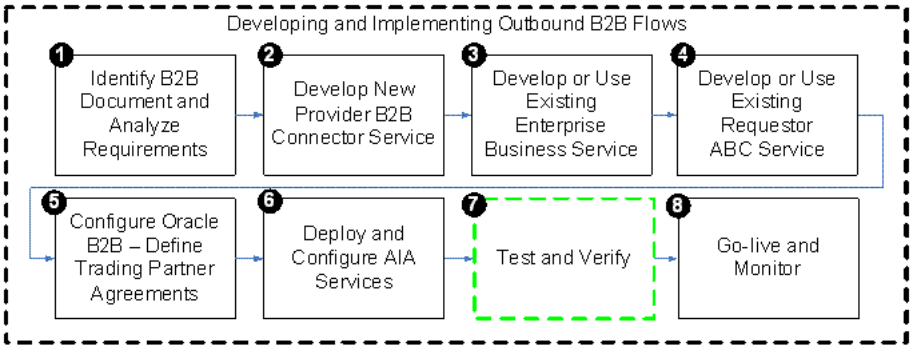
20.8 Step 7: Testing and Verifying

This section includes the following topics:

- Section 20.8.1, "How to Test Using CAVS"
- Section 20.8.2, "How to Test Using Dummy Trading Partner Endpoints"

The next step, as shown in Figure 20–37, is to test your newly developed or deployed AIA services that constitute the B2B integration flow.

Figure 20–37 Step 7: Testing and Verifying



Before you go live with your B2B integration flows with your trading partners, AIA recommends that you complete the following sequence of tests.

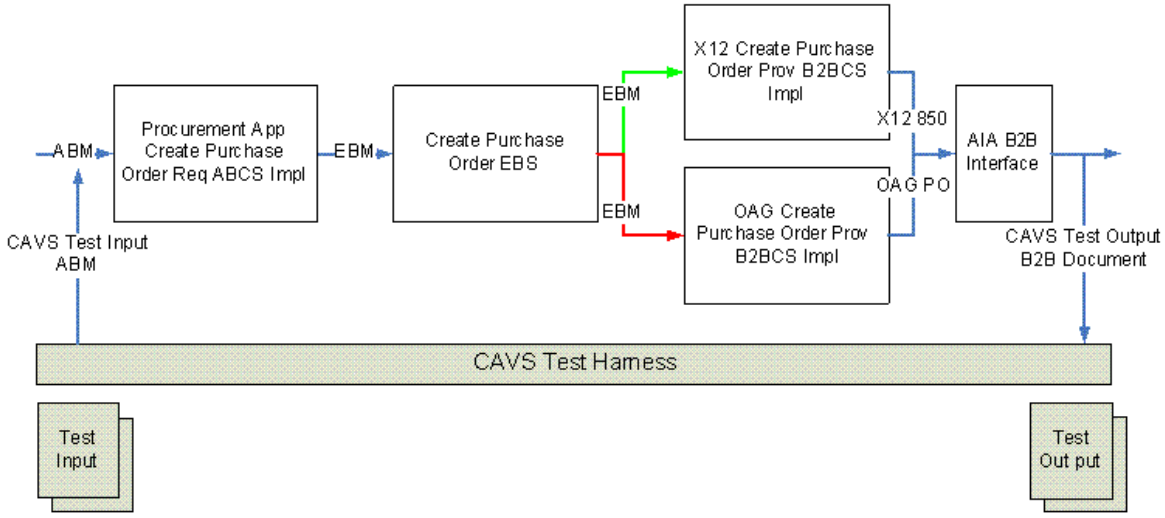
20.8.1 How to Test Using CAVS

If your AIA services are Composite Application Validation System (CAVS)-enabled, you can test your AIA services using a test simulator.

Using a CAVS simulator, you can simulate the behavior of your trading partners by sending messages to and receiving messages from a test harness, instead of your actual trading partners. This testing can take place without any knowledge of your trading partners.

The objective of this testing, as shown in Figure 20–38, is to verify that the AIA services are properly developed, deployed, and configured.

Figure 20–38 B2B Integration Flow Test Using CAVS



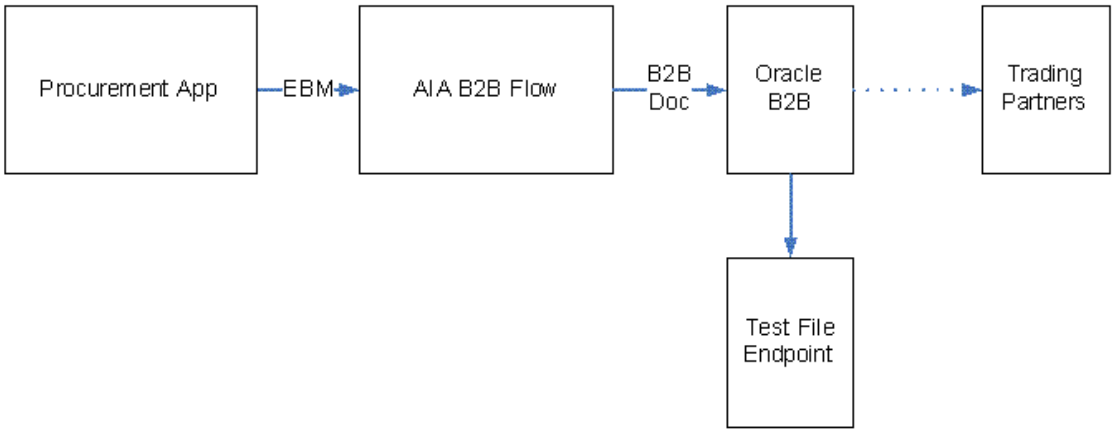
20.8.2 How to Test Using Dummy Trading Partner Endpoints

This section includes the following topics:

- Section 20.8.2.1, "How to Test Using the Production Code Value Set to "Test""
- Section 20.8.2.2, "How to Test Using Dummy Business Data"

Another common approach to testing is to create dummy delivery channels in your trading partner setup, as shown in Figure 20–39. For example, instead of configuring your trading partner agreement to send outbound documents to the remote trading partner's server, you can configure the agreement to create the outbound B2B files in a temporary B2B file location on the server.

Figure 20–39 B2B Integration Flow Test Using Dummy Trading Partner Endpoints



In this case, if you test an outbound B2B flow, all of the components involved in the flow are invoked, but the generated B2B document is copied to a temporary file directory. You can review the B2B document for completeness and correctness and also share it for review with your trading partners.

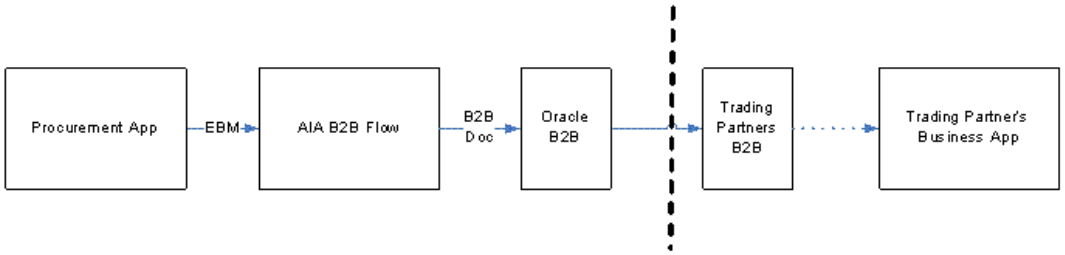
- The objective of this testing is to verify the integration between AIA services, the application, and B2B and to verify that the configuration across these layers is consistent.
- This testing can also take place without any knowledge of your trading partners.

20.8.2.1 How to Test Using the Production Code Value Set to "Test"

Certain B2B document protocols support a protocol envelope-level flag that can be used to specify whether the B2B document is being sent in test or production mode. The trading partner agreements in Oracle B2B can be configured to set this indicator to `Test` mode.

Now when you trigger an outbound B2B flow, the document is sent across to your trading partners, but the trading partner system recognizes the inbound document as a test instance and does not pass it on to the backend application, as shown in Figure 20–40.

Figure 20–40 B2B Integration Flow Test Using the Product code Value Set to "Test"



The objective of this testing is to verify the handshake between your B2B server and the trading partner's B2B server. Setup of transport and messaging features such as acknowledgement, encryption, packaging, partner and document identification, and so forth are verified.

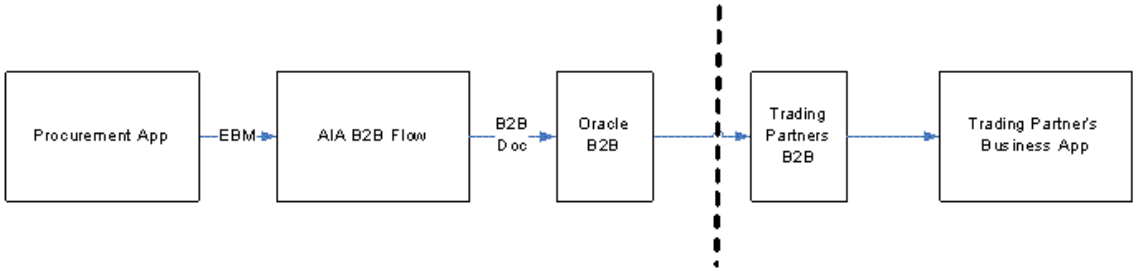
This approach requires that this test mechanism be discussed and agreed upon with each of your trading partners.

20.8.2.2 How to Test Using Dummy Business Data

Finally, you can test integration from your production systems to your remote trading partner's production systems by using dummy business data in the B2B documents being exchanged. For example, to test end-to-end outbound Purchase Order integration from your (buyer) system to your trading partner's (vendor) system, which are integrated by an outbound EDI X12 850 B2B flow, place orders either using an EDI test item created for the trading partner or by using a price of one cent for the items being ordered.

The order request is received and processed successfully by the trading partner's business application, as shown in Figure 20–41, however, the trading partner's business users or rules discard the order request.

Figure 20–41 B2B Integration Flow Test Using Dummy Business Data



This approach also requires that the test mechanism be discussed and agreed upon with each of your trading partners.

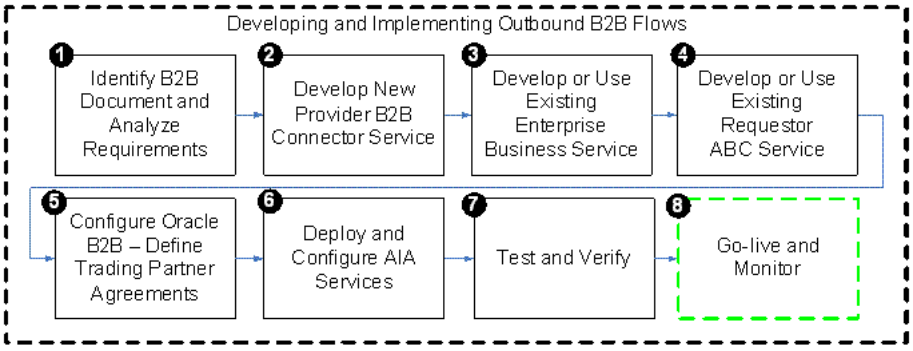
20.9 Step 8: Going Live and Monitoring

This section includes the following topics:

- Section 20.9.1, "Monitoring Using Oracle B2B Reports"
- Section 20.9.2, "Monitoring Using Oracle Enterprise Manager Console"
- Section 20.9.3, "Monitoring Using Error Notifications"

The final step, as shown in Figure 20–42, is to go live with your trading partner for the outbound B2B document flow. The AIA and Oracle B2B deployments are duplicated in the product servers and rolled out to the end users.

Figure 20–42 Step 8: Going Live and Monitoring



20.9.1 Monitoring Using Oracle B2B Reports

Oracle B2B's built-in reporting allows for the creation of the reports listed in Table 20–9. These reports query the run-time transactions in Oracle B2B and can be used to monitor individual transactions with trading partners.

Oracle B2B allows reports to be created and saved as report definitions, which can be used to generate reports for specific criteria, such as View Purchase Orders exchanged with trading partner Global, for example. You can also create ad hoc reports.

Table 20–9 Oracle B2B Report Types

Report Type	Description	Example Report
Business Message Report	These reports provide business messages based on specified search criteria.	View all Purchase Order documents received from trading partner "Global"
Wire Message Status Report	These reports enable you to query for wire messages in the native data formats sent to and received from trading partners. Wire message reports contain transport and protocol-related information in addition to the business data.	View all messages sent to trading partner ABC Corp including the HTTP headers.
Collaboration Status Reports	These reports help you group related individual transactions that together perform a task. This report is supported only for the RosettaNet document protocol.	View all RosettaNet 3A4 collaborations.
Error Reports	These reports help you monitor all failed transactions in Oracle B2B	View all messages received from trading partner Global that failed in Oracle B2B due to any error.

For more information about how to use the reporting functionality in B2B, see "Creating Reports" in *Oracle Fusion Middleware User's Guide for Oracle B2B.*

20.9.2 Monitoring Using Oracle Enterprise Manager Console

Oracle Enterprise Manager Console can be used to monitor the AIA flows. To monitor the outbound B2B integration flows, you can log in to the Oracle Enterprise Manager Console and look for instances of the requester ABCS that triggered the B2B flows.

The status of the ABCS instance, payload, and child processes can all be monitored from the Oracle Enterprise Manager Console.

By looking up the instances of the composite AIAB2BInterface[1.0], for example, you can view the JMS payload being passed to Oracle B2B, including the various B2B header parameters.

20.9.3 Monitoring Using Error Notifications

In the case of failures in either the AIA layer or Oracle B2B, the AIA Error Handler gets triggered. The AIA Fault Message contains the error text and description. In addition, B2B information pertaining to the failed transaction, such as Sender, Receiver Trading Partner, Document Type, and so forth, are also supplied in the AIA Fault.

For more information about error notifications, see "Using Error Notifications" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

21

Developing and Implementing Inbound B2B Integration Flows

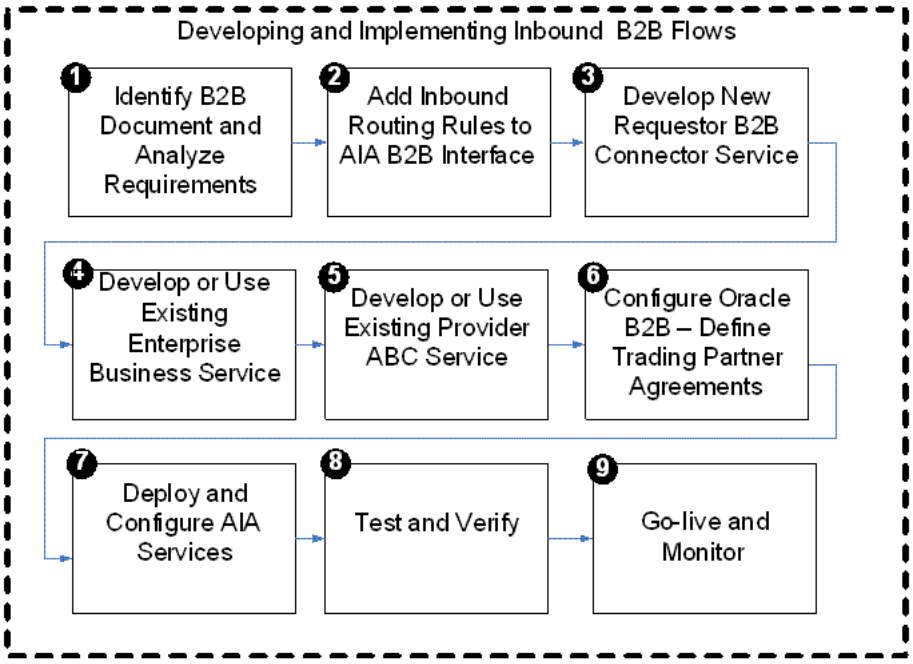
This chapter provides an overview of developing and implementing inbound B2B integration flows and describes how to identify the B2B document and analyze requirements, add inbound routing rules to an AIA B2B interface, develop new requester B2B connector service, develop or extend an existing Enterprise Business Service, develop or extend an existing requester ABCS, configure Oracle B2B and define trading partner agreements, deploy and configure AIA services and finally how to test and go live.

This chapter discusses the following topics:

21.1 Introduction to Developing and Implementing Inbound B2B Integration Flows

Figure 21–1 shows the high-level steps involved in developing a simple inbound business-to-business (B2B) flow from an application to trading partners using Oracle Application Integration Architecture (AIA).

Figure 21–1 High-Level Steps to Develop and Implement a Simple Inbound B2B Flow

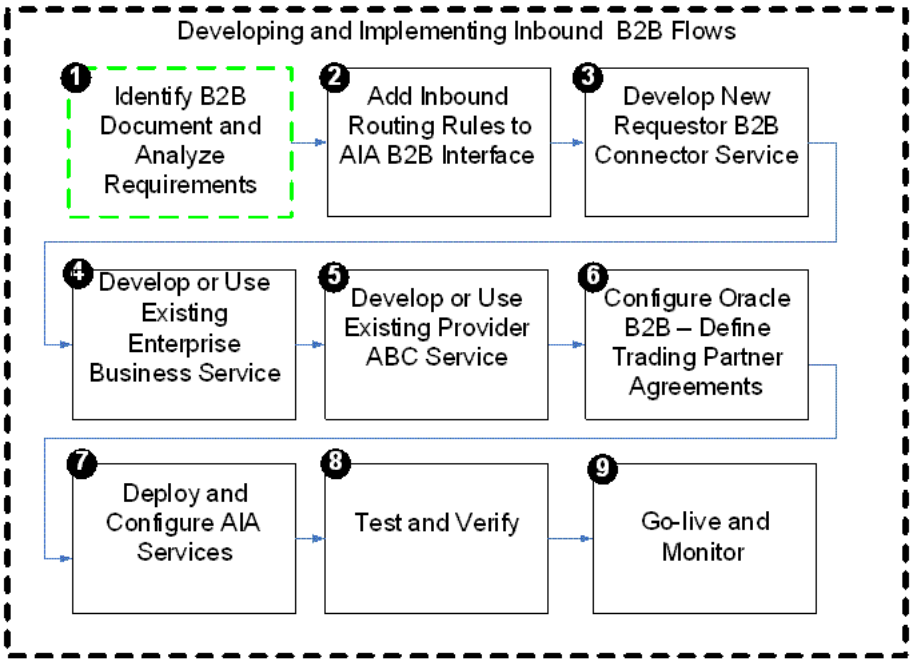


Most of the preceding steps are described inChapter 20, "Developing and Implementing Outbound B2B Integration Flows." This chapter is referred to whenever applicable.

21.2 Step 1: Identifying the B2B Document and Analyzing Requirements

The first step in building an inbound B2B flow, as shown in Figure 21–2, is to identify the B2B document that must be generated by the flow. As a part of this step, you must also analyze the requirements for the AIA services that must be built or extended to support the flow.

Figure 21–2 Step 1: Identifying B2B Document and Analyzing Service Requirements



The only differences in this step for the outbound B2B document flows and the steps for the inbound B2B document flows are as follows:

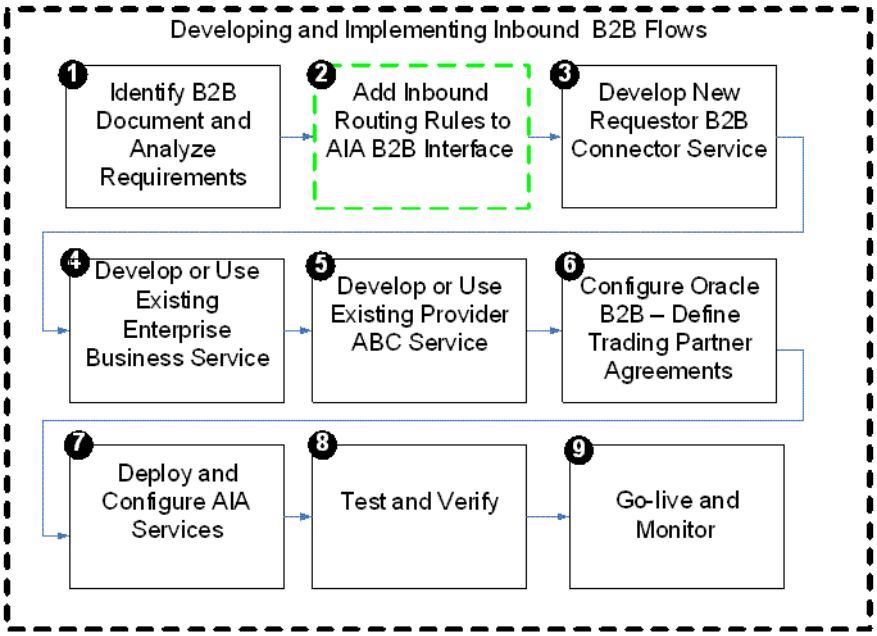
1. The source and targets in the mapping are reversed. The B2B document received from trading partners is the source of the mapping. The AIA Enterprise Business Message (EBM) is the target of the mapping.
2. At the end of this step:
   a. The B2B document is defined in Oracle B2B.
   b. The XML schema of the B2B document is uploaded in the AIA Metadata Repository.
   c. The Enterprise Business Object (EBO) and the EBM to be used in the integration are identified.
   d. Functional mapping between the B2B document and the AIA EBM is complete.

For more information about identifying the B2B document and analyzing requirements, see Section 20.2, "Step 1: Identifying the B2B Document and Analyzing Requirements."

21.3 Step 2: Adding Inbound Routing Rules to an AIA B2B Interface

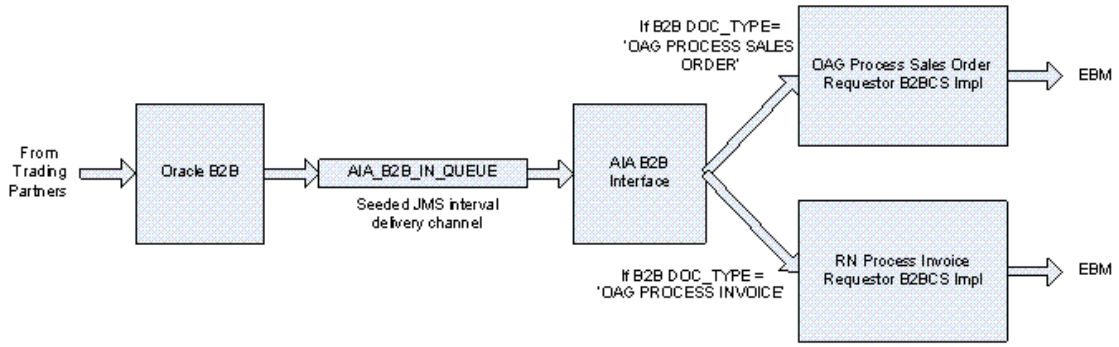
The next step in building an inbound B2B flow, as shown in Figure 21–3, is to add routing rules to the AIAB2BInterface Infrastructure service.

Figure 21–3 Step 2: Adding Inbound Routing Rules to an AIA B2B Interface



In inbound B2B document flows, the AIAB2BInterface service listens for new B2B documents received by Oracle B2B and routes them to the requester B2B services, as shown in Figure 21–4.

Figure 21–4 Oracle B2B Routing New B2B Documents to Requester B2B Services



The AIA B2B Interface composite has a JMS adapter to listen for new inbound B2B documents processed by Oracle B2B. The trading partner agreement in Oracle B2B is configured to route inbound B2B documents to the AIA_B2B_IN_QUEUE JMS queue.

The AIA B2B Interface identifies the B2B document type of the inbound B2B document and routes the document to the requester B2B Connector Service (B2BCS) Implementation that is responsible for processing the B2B document.

For example, if the B2B document type is OAG_Process Invoice, then the AIA B2B Interface routes the document to the OAGProcessInvoiceRequesterB2BCSImpl service.

The input to the requester B2BCS Implementation is the B2BM element. The B2BM element is defined in the AIAComponents/EnterpriseObjectLibrary/Infrastructure/V1/Meta.xsd file.

The AIA B2B Interface constructs the B2BM element from the B2B document received from Oracle B2B as described here:

- /corecom:B2BM /corecom:B2BMHeader /corecom:B2BMID
  - Mapped from: B2B document JMS header property MSG_ID
  - Description: The unique identifier generated by Oracle B2B for the inbound B2B document.

    This field is to correlate the AIA flow with the corresponding transaction in Oracle B2B.
  - Example: 5602359000000
- /corecom:B2BM /corecom:B2BMHeader /corecom:B2BDocumentType /corecom:TypeCode
  - Mapped from: DOCTYPE_NAME
  - Description: The document type of the inbound document as identified by Oracle B2B.
  - Example: OAG_PROCESS_INVOICE
- /corecom:B2BM /corecom:B2BMHeader /corecom:B2BDocumentType /corecom:Version
  - Mapped from: DOCTYPE_REVISION
  - Description: The document revision of the inbound document as identified by Oracle B2B.
  - Example: 9.0
- /corecom:B2BM /corecom:B2BMHeader /corecom:SenderTradingPartner /corecom:TradingPartnerID
  - Mapped from: FROM_PARTY
  - Description: The source (from) trading partner of the B2B document.

    The value is the trading partner name as defined in Oracle B2B.
  - Example: GlobalChips
- /corecom:B2BM /corecom:B2BMHeader /corecom:ReceiverTradingPartner /corecom:TradingPartnerID
  - Mapped from: TO_PARTY
  - Description: The destination (to) trading partner to whom the inbound B2B document was sent.

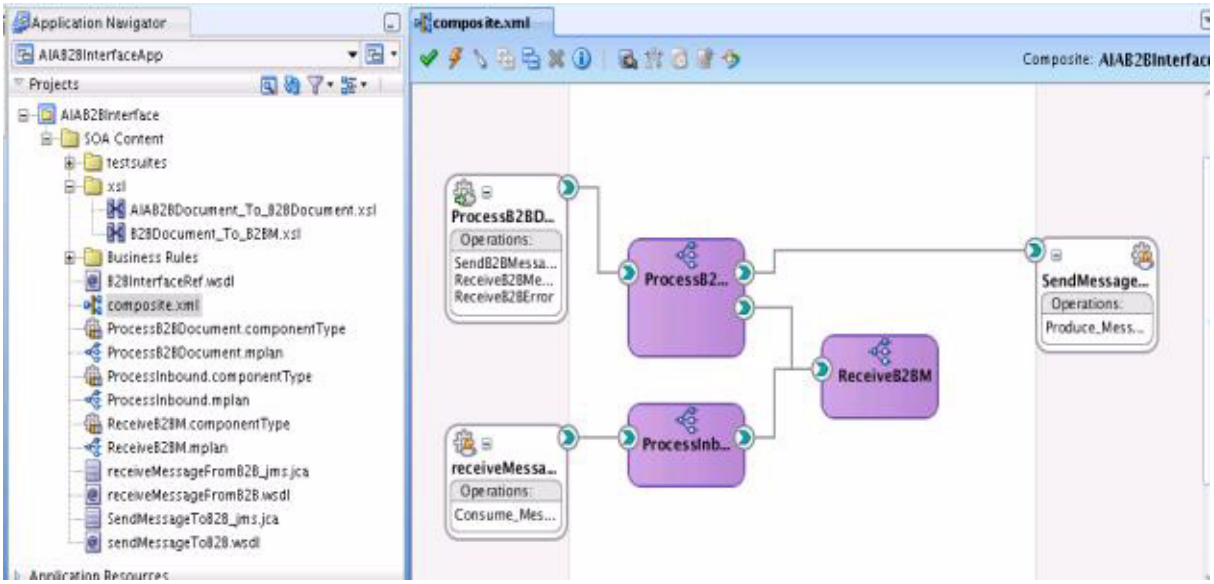
    The value is the trading partner name as defined in Oracle B2B.
  - Example: Acme
- /corecom:B2BM /corecom:Payload
  - Mapped from: JMS payload
  - Description: The actual JMS text payload that contains the B2B document sent by the trading partner.
  - Example: <?xml version="1.0"?> <PROCESS_INVOICE_002> <CONTROLAREA>…

21.3.1 How to Add a New Routing Rule to the AIA B2B Interface

To add a new routing rule to the AIA B2B Interface:

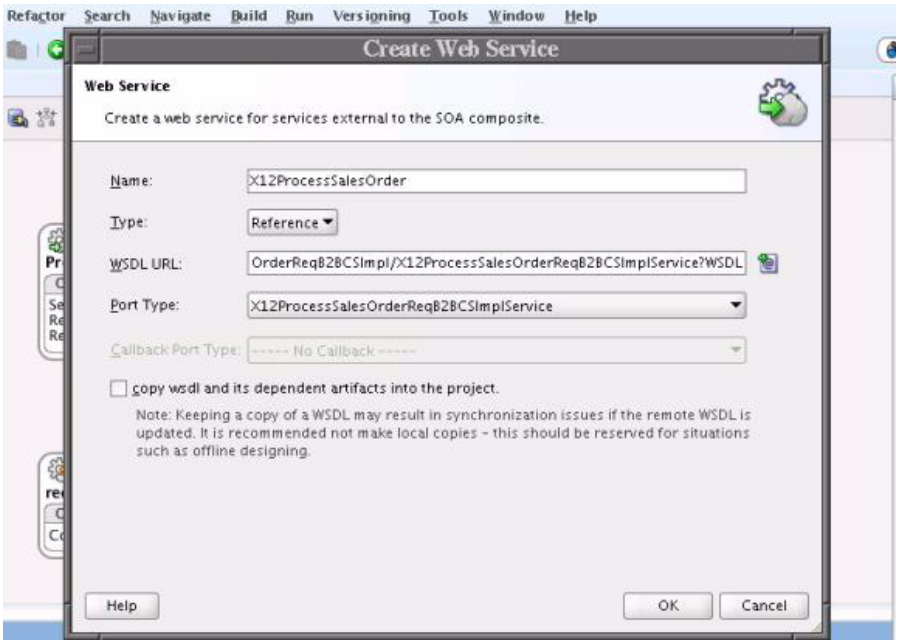
1. Open the $AIA_HOME/Infrastructure/B2B/AIAB2BInterfaceApp/AIAB2BInterfaceApp.jws application using Oracle JDeveloper.
2. Open the composite.xml file using Oracle JDeveloper, as shown in Figure 21–5.

Figure 21–5 composite.xml in Oracle JDeveloper



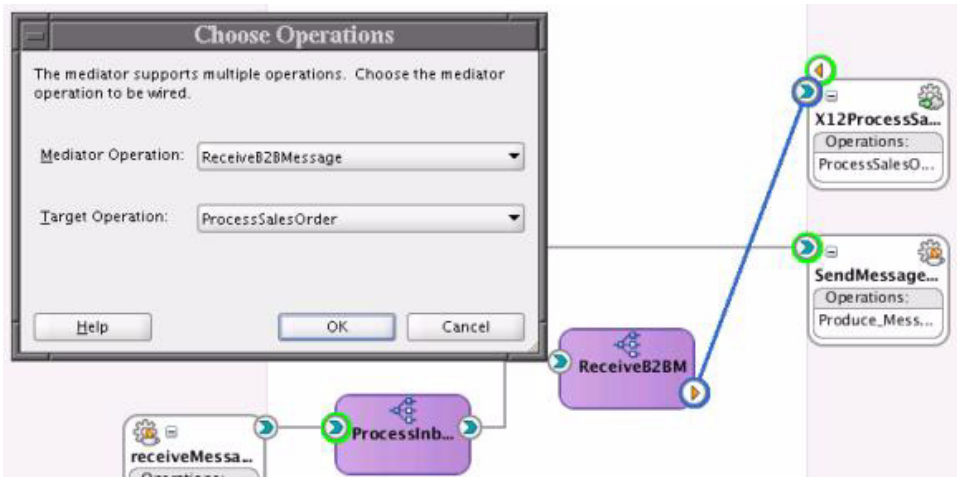
3. Add a new service reference to the requester B2BCS that processes the new inbound B2B document type as shown in Figure 21–6. The next section of this document explains how to develop requester B2BCSs.

Figure 21–6 New Service Reference Added to the Requester B2BCS that Processes the New Inbound B2B Document Type



4. Add a wiring to the newly added service from the Receive B2BM Mediator component in the composite as shown in Figure 21–7.

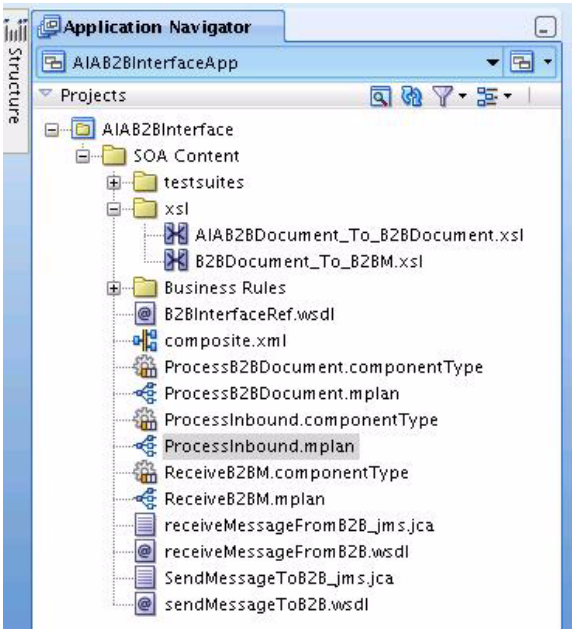
Figure 21–7 Wiring Added to the Newly Added Service from the Receive B2BM Mediator Component in the Composite



5. Choose the Mediator operation ReceiveB2BMessage as the source of the wiring and the target as the operation exposed by the target requester B2BCS. Save your changes.

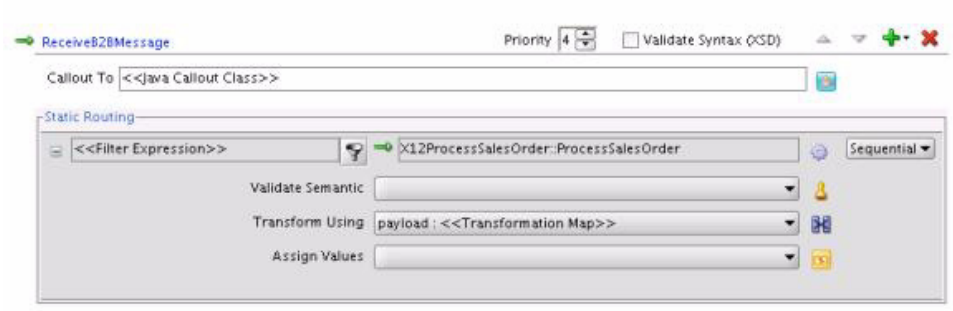
6. Open the file ProcessInbound.mplan as shown in Figure 21–8.

Figure 21–8 ProcessInbound.mplan



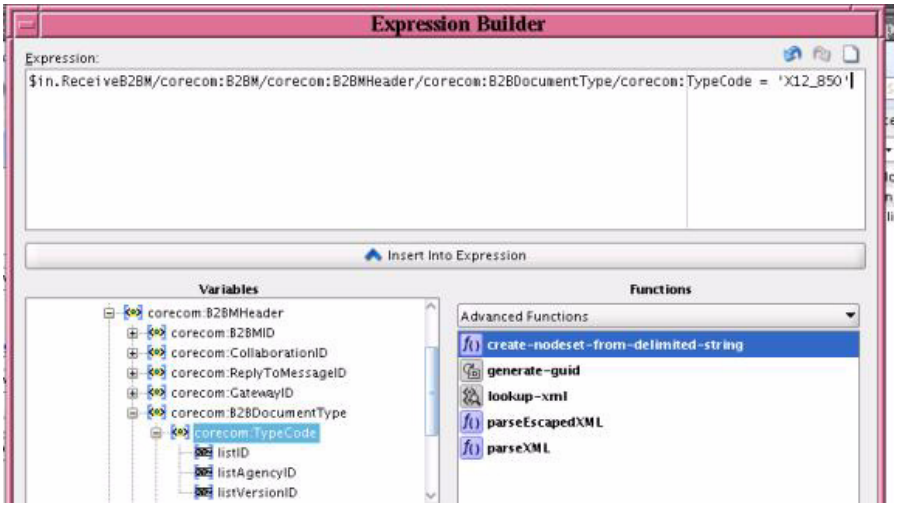
7. Add a new routing rule to invoke the requester B2BCS using the service reference added to the file composite.xml, as shown in Figure 21–9.

Figure 21–9 New Routing Rule to Invoke the Requester B2BCS



8. Add a new routing rule to the ReceiveB2BMessage operation.
9. Edit the filter operation for the newly added routing rule. Define the filter based on the B2B document type. For example, the routing filter in Figure 21–10 ensures that the EDI X12 Process Sales Order documents are routed to the X12 ProcessSalesOrder B2BCS Implementation.

Figure 21–10 Routing Filter Ensuring that EDI X12 Process Sales Order Documents are Routed to ProcessSalesOrder B2BCS Implementation



10. Save your changes.
11. Deploy the modified SOA composite application.

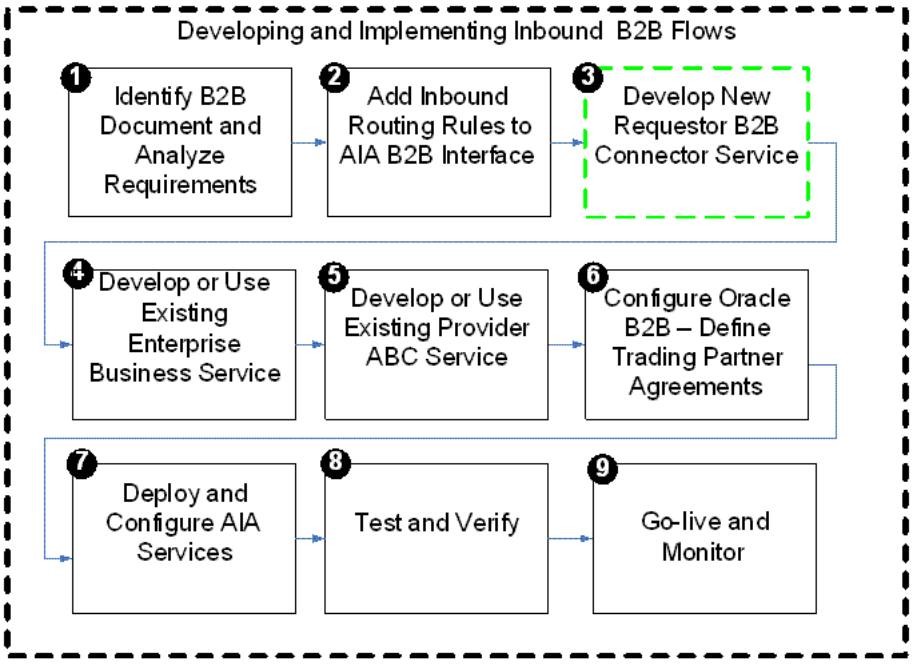
21.4 Step 3: Developing a New Requester B2B Connector Service

This section includes the following topics:

- Section 21.4.1, "Introduction to Requester B2B Connector Services"
- Section 21.4.2, "How to Identify the Message Exchange Pattern"
- Section 21.4.3, "How to Develop a B2BCS Service Contract"
- Section 21.4.4, "How to Store a WSDL in Oracle Metadata Services Repository"
- Section 21.4.5, "How to Develop a B2B Connector Service"
- Section 21.4.6, "How to Annotate B2B Connector Services"
- Section 21.4.7, "How to Support Trading Partner-Specific Variants"
- Section 21.4.8, "How to Enable Error Handling"

The next step, as shown in Figure 21–11, is to develop the requester B2BCS to support the outbound B2B integration flow.

Figure 21–11 Step 3: Developing a New Requester B2B Connector Service



The requester B2BCS is very similar to a requester Application Business Connector Service (ABCS), with the only difference being that it integrates with trading partners through Oracle B2B instead of integrating with an application. Hence, AIA recommends that you familiarize yourself with the design and development of ABCSs (requester and provider).

For more information about developing ABCSs, see Chapter 15, "Constructing the ABCS."

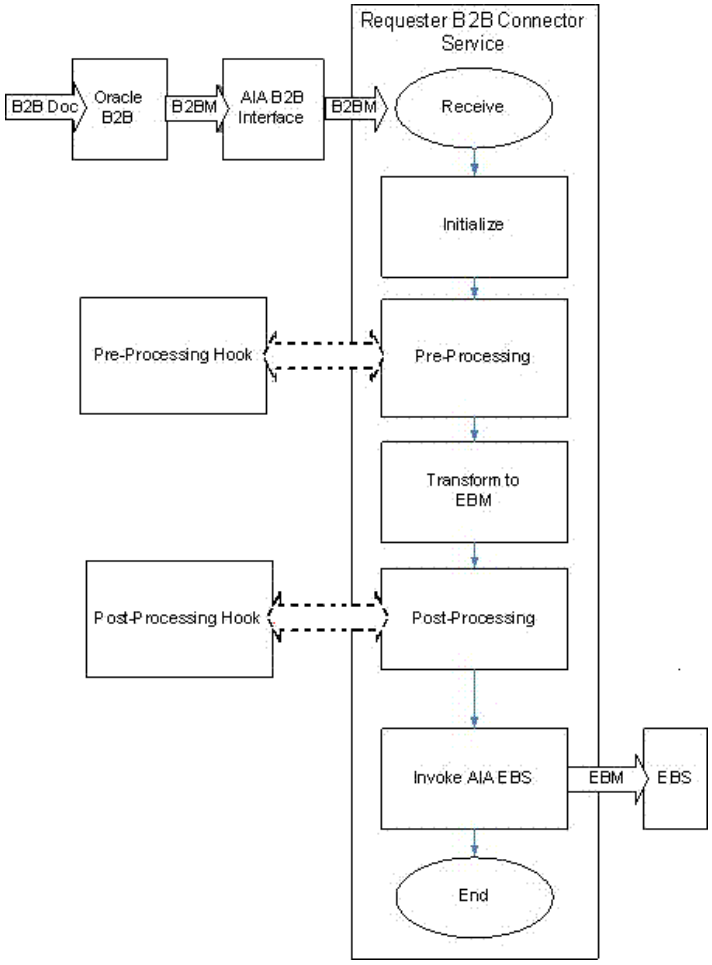
21.4.1 Introduction to Requester B2B Connector Services

The key function provided by a requester B2BCS is to enable inbound B2B document integration by performing the following tasks:

- Receive B2B documents sent by trading partners from Oracle B2B.
- Transform B2B documents into AIA EBMs.
- Use EBMs as request payloads to invoke AIA Enterprise Business Services (EBSs).

Figure 21–12 illustrates the processing that takes place in a simple fire-and-forget message exchange pattern-based provider B2BCS.

Figure 21–12 Process Flow of a Simple Fire-and-Forget Message Exchange Pattern-Based Provider B2BCS



Step-by-step instructions for developing B2BCSs are provided in the following sections.

21.4.2 How to Identify the Message Exchange Pattern

Similar to outbound B2B flows, most inbound B2B flows can be modeled using the fire-and-forget message exchange pattern.

Responses to be sent to trading partners can be modeled using separate outbound fire-and-forget flows.

Also, depending on the protocol involved, Oracle B2B can be configured to automatically send a confirmation or acknowledgement message to trading partners.

For more information, see Section 20.3.2, "How to Identify the Message Exchange Pattern."

For more information about identifying message exchange patterns, see Section 14.3, "Identifying the MEP."

21.4.3 How to Develop a B2BCS Service Contract

First, define the service contract (WSDL) of the requester B2BCS. The service contract of the provider B2BCS specifies how it is invoked by an AIA flow. The service contract

specifies the B2B operation being implemented and the B2B document type that it is capable of processing as the input request message.

For more information about creating WSDLs for ABCSs, see Section 14.2, "Defining the ABCS Contract."

Following are the naming conventions recommended for use in the B2BCS WSDL definitions:

- WSDL File Name:
  - Naming guideline: <B2BStandard><Verb><EBO>ReqB2BCSImpl.wsdl
  - Example: X12ProcessSalesOrderReqB2BCSImpl.wsdl
- Service Namespace:
  - http://xmlns.oracle.com/B2BCSImpl/{Core|Industry/<IndustryName>}/<B2BStandard><Verb><EBO>ReqB2BCSImpl/<ABCSVersion>
  - Example: http://xmlns.oracle.com/B2BCSImpl/Core/X12ProcessSalesOrderProvB2BCSImpl/V1

    The ABCS Service version is independent of the B2B document/standard version.

    For more information about recommendations on versioning AIA services, see Section 16.8, "Versioning ABCS."
- Service Name:
  - Naming guideline: <B2BStandard><Verb><EBO>ReqB2BCSImpl
  - Example: X12ProcessSalesOrderProvB2BCSImpl
- Port Type Name:
  - Naming guideline: <B2BStandard><Verb><EBO> ReqB2BCSImplService
  - Example: X12ProcessSalesOrderProvB2BCSImplService
- Operation Name:
  - <Verb><EBO>
  - ProcessSalesOrder
- Request Message Name:
  - <Verb><EBO>ReqMsg
  - ProcessSalesOrderReqMsg

21.4.4 How to Store a WSDL in Oracle Metadata Services Repository

As a SOA best practice, AIA recommends that all shared service artifacts, such as WSDL and XSD files, be stored in a central location that can be accessed by multiple services.

All of the AIA-shared artifacts are stored in the Oracle Metadata Services (MDS) repository. Storage of shared artifacts in the MDS not only makes them globally accessible, but also enables AIA to leverage features in MDS that support caching and clustering.

Provider B2BCS WSDLs are stored at the following location in the MDS: /apps/AIAMetaData/AIAComponents/B2BServiceLibrary/Connectors/wsdls/

<B2B_STANDARD> /ProviderB2BCS/ <VERSION> / <B2B_STANDARD> <VERB> <OBJECT> ReqB2BCSImpl.wsdl

To store a WSDL in MDS:

1. Copy the handcrafted WSDL to the following location: $AIA_HOME/aia_instances/$INSTANCE_

   NAME/AIAMetaData/AIAComponents/B2BServiceLibrary

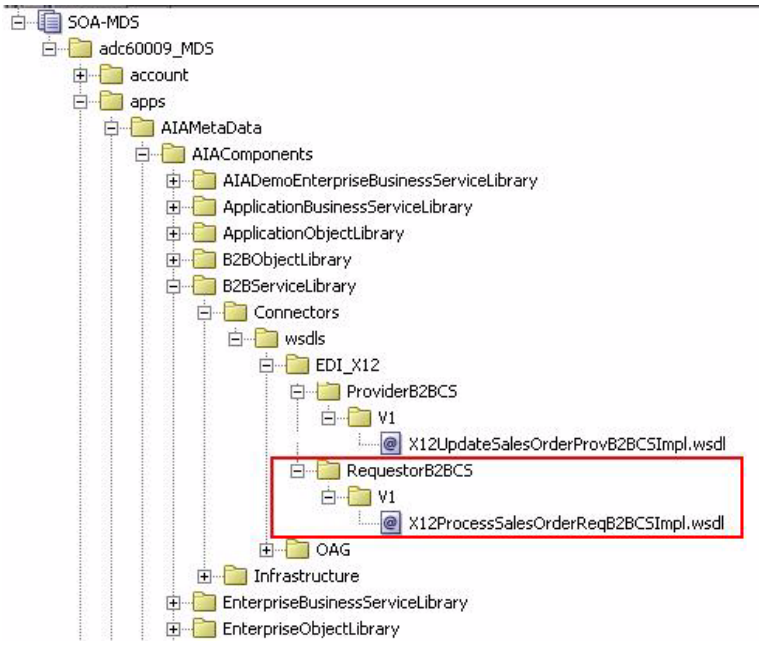
   /Connectors/wsdls/<B2B_STANDARD>/RequesterB2BCS

   /<VERSION>/<wsdl file>.wsdl

2. Run the UpdateMetaDataDP.xml file present in $AIA_HOME/aia_instances/$INSTANCE_NAME/config/.

3. Using a SOA-MDS server connection to MDS, verify that AIA Metadata has been supplied, as shown in Figure 21–13.

Figure 21–13 AIA Metadata in the MDS



4. The WSDL can now be accessed using a URL similar to the following: oramds:/apps/AIAMetaData/AIAComponents/B2BServiceLibrary/Connectors/wsdls/EDI_X12/RequesterB2BCS/V1/X12ProcessSalesOrderProvB2BCSImpl.wsdl

21.4.5 How to Develop a B2B Connector Service

The next step in the process is to develop the B2BCS. The requester B2BCS WSDL created in the previous step is used as the interface while you are developing the concrete B2BCS.

Because Service Constructor does not support the autogeneration of B2B services, use Oracle JDeveloper to develop the B2BCS. Develop a composite with a BPEL process based on the abstract WSDL created in the previous step.

Following are the key activities that must be developed in the B2BCS implementation BPEL.

To develop a B2B connector service:

1. Create a SOA Composite application containing a SOA project with a BPEL component.
2. Choose the WSDL created in the previous step as the interface for the SOA composite.
3. The values to be used for creating the SOA application and project are listed in Table 21–1.

Table 21–1 Values Used to Create the SOA Application and Project

Activity	Value
Application Name	`<B2BStandard> <Verb> <EBO> ReqB2BCSApp`
Project Name	`<B2BStandard> <Verb> <EBO> ReqB2BCSImpl`
Project Namespace	`http://xmlns.oracle.com/B2BCSImpl/Core/ <Standard> <Verb> <Object> ReqB2BCSImpl/V1` (Same as B2BCS WSDL namespace)
Composite Name	`<B2BStandard> <Verb> <EBO> ReqB2BCSImpl`
Composite Template	Composite with BPEL
BPEL Process Name	`<B2BStandard> <Verb> <EBO> ReqB2BCSImpl`
Namespace	`http://xmlns.oracle.com/B2BCSImpl/Core/ <Standard> <Verb> <Object> ReqB2BCSImpl/V1` (Same as B2BCS WSDL namespace)
Template	Base on WSDL
WSDL URL	URL of B2BCS service WSDL referred from MDS

4. Define variable <B2BM>_Var.

   This is the input variable to the BPEL process and is used in the receive activity.

   Define the variable based on the corecom:B2BM global element defined in the Meta.xsd.

5. Define variable EBM_HEADER of type corecom:EBMHeader.

   This variable is used to store the AIA process context information and is used by the fault-handling mechanism.

   Define the variable based on the corecom:B2BM global element defined in the Meta.xsd.

6. Define variable B2BM_HEADER of type corecom:B2BMHeader.

   This variable is used to store the B2B-specific AIA process context information and is used by the fault-handling mechanism.

   Define the variable based on the corecom:B2BMHeader global element defined in the Meta.xsd.

7. Define variable <B2BDoc>_Var using the external B2BDocument definition.

   This is used as the source of the transformation from EBM. This variable is then assigned to the <B2BDoc>B2BM_Var/Payload.

8. Define a partner link to the EBS.

   This is the AIA EBS that is invoked by the requester B2BCS.

The abstract EBS WSDL can be referenced from: oramds:/apps/AIAMetaData/AIAComponents/EnterpriseBusinessServiceLibrary/Core/<EBO>/V<x>/<EBOName>.wsdl.

9. Transform the EBM to the B2B Document.

   Use a transform activity to transform the EBM to the B2B Document format. Invoke an XSLT developed per the mapping created in the previous step.

10. Assign B2B-specific EBMHeader variables as shown here:

    - Element: corecom:EBMHeader/corecom:B2BProfile/corecom:SenderTradingPartner/corecom:TradingPartnerID

      Source B2BM XPATH: corecom:B2BM/corecom:B2BMHeader/corecom:SenderTradingPartner/corecom:TradingPartnerID

    - Element: corecom:EBMHeader/corecom:B2BProfile/corecom:ReceiverTradingPartner/corecom:TradingPartnerID

      Source B2BM XPATH: corecom:B2BM/corecom:B2BMHeader/corecom:ReceiverTradingPartner/corecom:TradingPartnerID

    This way, the sender and receiver trading-partner information identified by Oracle B2B is passed on to the EBM.

11. Invoke the EBS.

    Invoke the AIA EBS to process the EBM.

12. Compile the BPEL process and ensure that no errors occurred. You can use Oracle JDeveloper to deploy the BPEL process to a development server that has AIA Foundation Pack installed.

21.4.6 How to Annotate B2B Connector Services

To make key metadata about the B2BCS available to the Project Lifecycle Workbench and Oracle Enterprise Repository, the composite.xml file of the B2BCS Implementation SOA composite must be annotated in a specific manner.

For more information about the Project Lifecycle Workbench, see Chapter 3, "Working with Project Lifecycle Workbench."

For more information about annotating B2B services, see Chapter 12, "Annotating Composites."

To annotate requester B2B Connector Services:

1. Table 21–2 lists the annotation elements that must be added to the service element composite.xml, as described subsequently.

Table 21–2 Service Annotation Elements in composite.xml

Annotation Element	Description	Example
AIA/Service/InterfaceDetails/Service Operation	<Verb><EBOName>	ProcessSalesOrder
AIA/Service/ImplementationDetails/ArtifactType	RequesterB2BCSImplementation	RequesterB2BCSImplementation

Table 21–2 (Cont.) Service Annotation Elements in composite.xml

Annotation Element	Description	Example
AIA/Service/Implementation Details/ServiceOperation/Name	<Verb><EBOName>	ProcessSalesOrder
AIA/Service/Implementation Details/B2BDocument	B2B Document Type processed by this B2BCS	850
AIA/Service/Implementation Details/B2BDocumentVersion	B2B Document Version processed by this B2BCS	4010
AIA/Service/Implementation Details/B2BStandard	B2B Standard processed by this B2BCS	X12
AIA/Service/Implementation Details/B2BStandardVersion	B2B Standard Version processed by this B2BCS	4010

The following XML snippet, as shown in Figure 21–14, is an example of an annotated B2BCS called X12ProcessSalesOrderReqB2BCSImpl composite.xml.

Figure 21–14 Example Snippet of an Annotated B2BCS Called X12ProcessSalesOrderReqB2BCSImpl composite.xml

```
<service name="X12ProcessSalesOrderReqB2BCSImplService"
         ui:wsdlLocation="oramds:/apps/AIAMetaData/AIAComponents/B2BServiceLibrary/wsdls/EDI_X12/Requ
  <interface.wsdl interface="http://xmlns.oracle.com/B2BCSImpl/Core/X12ProcessSalesOrderReqB2BCSImpl/
  <binding.ws port="http://xmlns.oracle.com/B2BCSImpl/Core/X12ProcessSalesOrderReqB2BCSImpl/V1#wsdl.e
    <!--<svcdoc:AIA>
                <svcdoc:Service>
                     <svcdoc:ImplementationDetails>
                         <svcdoc:ApplicationName></svcdoc:ApplicationName>
                         <svcdoc:BaseVersion></svcdoc:BaseVersion>
                         <svcdoc:DevelopedBy>Oracle</svcdoc:DevelopedBy>
                         <svcdoc:OracleCertified>Yes</svcdoc:OracleCertified>
                         <svcdoc:ArtifactType>RequesterB2BCSImplementation</svcdoc:ArtifactType>
                         <svcdoc:ServiceOperation>
                             <svcdoc:Name>ProcessSalesOrder</svcdoc:Name>
                         </svcdoc:ServiceOperation>
                        <svcdoc:B2BDocument>850</svcdoc:B2BDocument>
                        <svcdoc:B2BDocumentVersion>4010</svcdoc:B2BDocumentVersion>
                        <svcdoc:B2BStandard>EDI_X12</svcdoc:B2BStandard>
                        <svcdoc:B2BStandardVersion>4010</svcdoc:B2BStandardVersion>
                       </svcdoc:ImplementationDetails>
                 </svcdoc:Service>
          </svcdoc:AIA>-->
</service>
```

2. The reference to the AIA EBS invoked by this B2BCS should also be annotated in the composite.xml. Annotation elements are listed in Table 21–3.

Table 21–3 composite.xml Annotations Used to Reference the AIA EBS Invoked by the B2BCS

Annotation Element	Description	Example
AIA/Reference/ArtifactType	Enter value "EnterpriseBusinessService"	EnterpriseBusinessService
AIA/Reference/ServiceOperation/Name	EBS operation name	ProcessSalesOrder

Annotations as they appear in composite.xml are shown in Figure 21–15.

Figure 21–15 Reference to the AIA EBS Invoked by the B2BCS Annotated in the composite.xml

For your reference, the $AIA_HOME/Infrastructure/B2B/src/B2BConnectors/EDIX12B2BCSApp/X12Process SalesOrder ReqB2BCSImpl/composite.xml file contains sample Requester B2BCS Impl annotations.

21.4.7 How to Support Trading Partner-Specific Variants

Support for Trading Partner-specific processing can be built into the requester B2BCS in a manner similar to the way they are built into the provider B2BCS.

For more information about how to build trading partner-specific support in B2BCSs, see Section 20.3.8, "How to Support Trading Partner-Specific Variants."

21.4.8 How to Enable Error Handling

For more information about how to enable AIA services for error handling and recovery, see "Setting Up Error Handling" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

In short, the following steps must be taken:

- Associate a fault-policy.xml with the B2BCS composite.
- Invoke the AIAAsyncErrorHandlingBPELProcess for business faults, as shown in Figure 21–16 and Figure 21–17.

Figure 21–16 B2BCS Composite Defined to Invoke AIAAsyncErrorHandlingBPELProcess

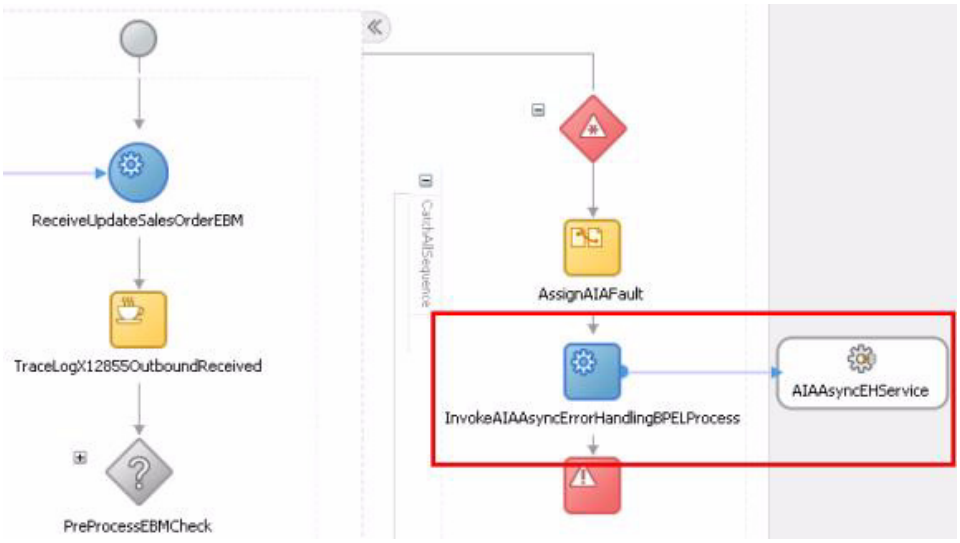


Figure 21–17 Invocation of the AIAAsyncEHService

```
    </assign>
          <invoke name="InvokeAIAAsyncErrorHandlingBPELProcess"
                  portType="aiaasynceh:AIAAsyncErrorHandlingBPELProcess" inputVariable="AIAA
                  partnerLink="AIAAsyncEHService"
                  operation="initiate"/>
          <throw name="ThrowAIAFault" faultName="corecom:Fault" faultVariable="AIAFaultMessa
        </sequence>
    </catchAll>
</faultHandlers>
```

While invoking the AIAAsyncErrorHandlingBPELProcess, the B2B-specific elements in the fault schema can be populated as shown in Table 21–4.

Table 21–4 B2B-Specific Elements in the Fault Schema That Can Be Populated by AIAAsyncErrorHandlingBPELProcess

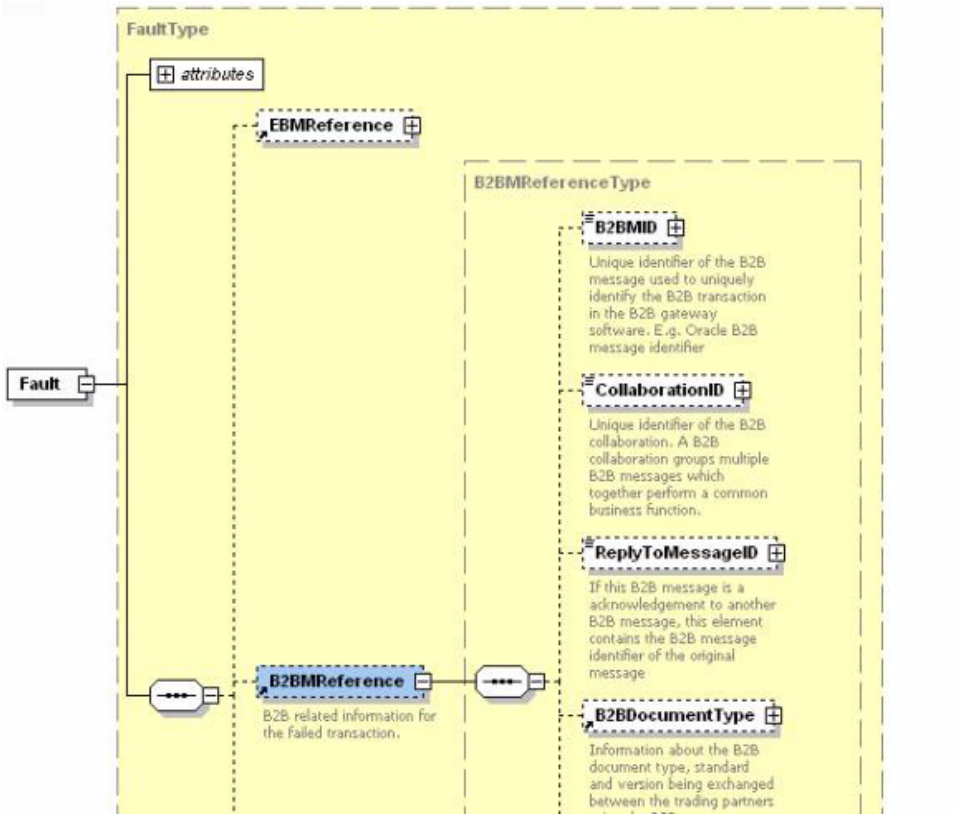
Fault Element Schema	Description	Example
Fault/B2BMReference/B2BMID	Unique identifier of the B2B document	13232325
Fault/B2BMReference/B2BDocumentType/TypeCode	Document type of the B2B document being processed by the requester B2BCS	855
Fault/B2BMReference/B2BDocumentType/Version	Document version of the B2B document being processed by the Requester B2BCS	4010
Fault/B2BMReference/B2BDocumentType/TypeCode/@listAgencyID	Standard of the B2B document being processed by the Requester B2BCS	X12
Fault/B2BMReference/GatewayID	Name of the B2B software being used.	Oracle B2B
Fault/B2BMReference/SenderTradingPartner/TradingPartnerID	Sender trading partner, mapped from the B2BM	MyCompany

Table 21–4 (Cont.) B2B-Specific Elements in the Fault Schema That Can Be Populated by AIAAsyncErrorHandlingBPELProcess

Fault Element Schema	Description	Example
Fault/B2BMReference/ReceiverTradingPartner/TradingPartnerID	Receiver trading partner, mapped from the B2BM	Global

Figure 21–18 provides the B2B-specific elements in the corecom:Fault.

Figure 21–18 B2B-Specific Elements in corecom:Fault

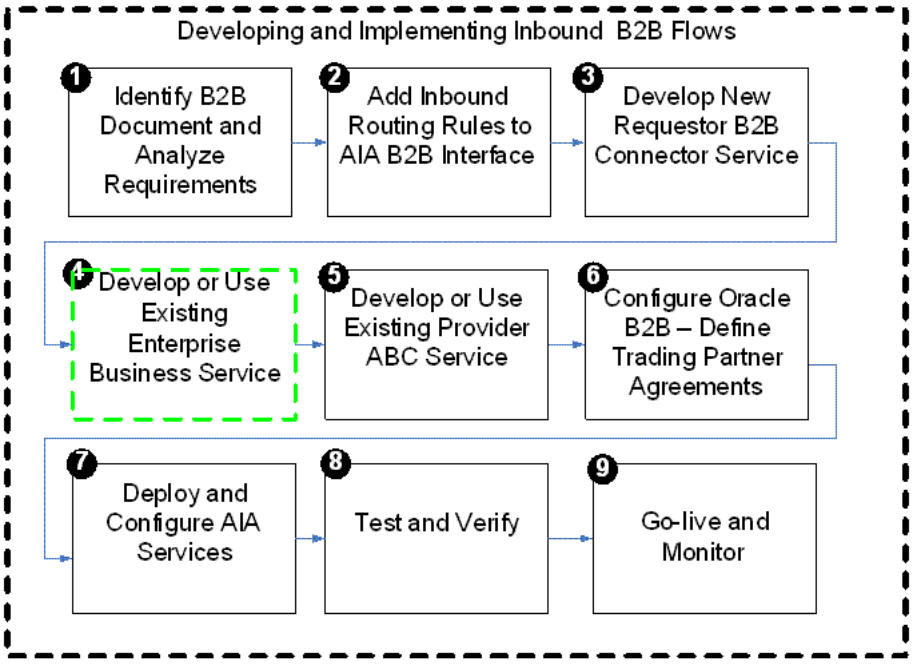


The B2B details of the failed AIA service that are available in the fault instance are logged and available for debugging the failed flow.

21.5 Step 4: Developing or Extending an Existing Enterprise Business Service

The next step, as shown in Figure 21–19, is to develop a new EBS or use an existing EBS that is invoked by the requester B2BCS.

Figure 21–19 Step 4: Developing or Extending an Existing Enterprise Business Service

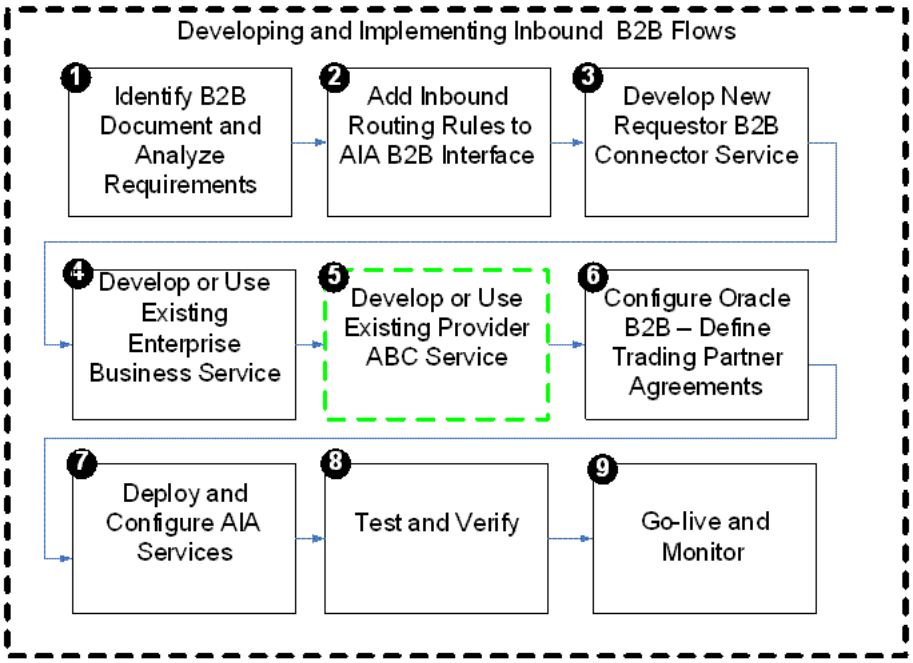


For more information about creating a new EBS, see Chapter 13, "Designing and Developing Enterprise Business Services."

21.6 Step 5: Developing or Extending an Existing Provider ABCS

The next step, as shown in Figure 21–20, is to develop a new or extend an existing provider ABCS. The provider ABCS processes the AIA EBM by invoking application APIs or web-services.

Figure 21–20 Step 5: Developing or Extending an Existing Provider ABCS



For more information about how to design and construct a provider ABCS, see Chapter 14, "Designing Application Business Connector Services" and Chapter 15, "Constructing the ABCS."

21.6.1 What You Must Know About Transformations

While you are developing this transformation from EBM to ABM, the Sender and Receiver Trading Partner information can be mapped to appropriate fields in the ABM to capture the source and target of the B2B message, as shown in Table 21–5.

Table 21–5 Sender and Receiver Trading Partner Fields in the EBM

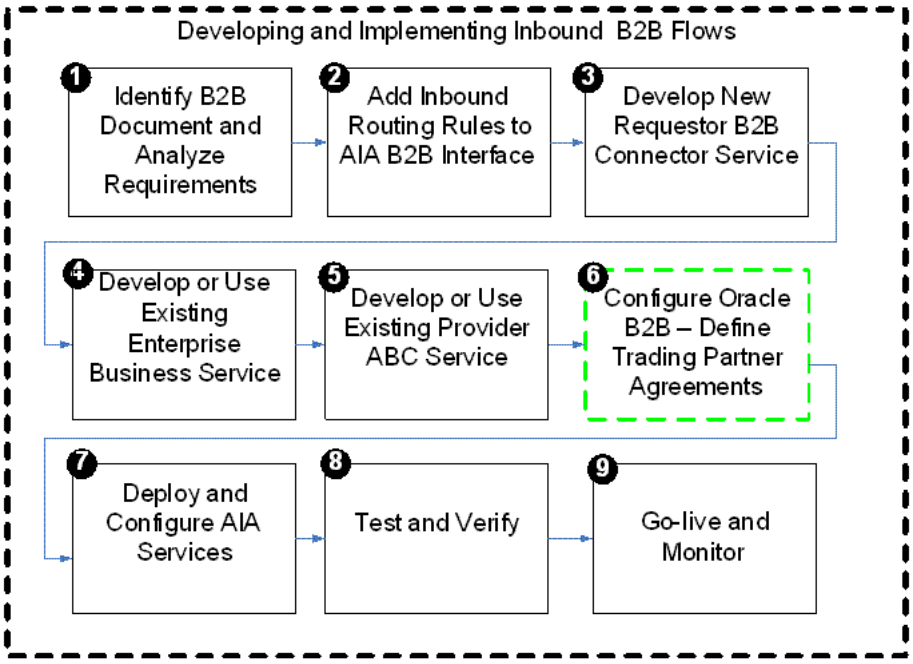
EBM Header Element	Description	Example Value
/EBMHeader/B2BProfile/SenderTradingPartner/TradingPartnerID	ID of the sending trading partner as defined in Oracle B2B. For inbound flows, this is the remote trading partner.	GlobalChips
/EBMHeader/B2BProfile/ReceiverTradingPartner/TradingPartnerID	ID of the receiving trading partner as defined in Oracle B2B. For inbound flows, this is the host trading partner.	Acme

At the end of this step, all of the required AIA services for developing an outbound B2B integration flow are ready.

21.7 Step 6: Configuring Oracle B2B and Defining Trading Partner Agreements

The next step, as shown in Figure 21–21, is to create trading partner agreements in Oracle B2B.

Figure 21–21 Step 6: Configuring Oracle B2B and Defining Trading Partner Agreements



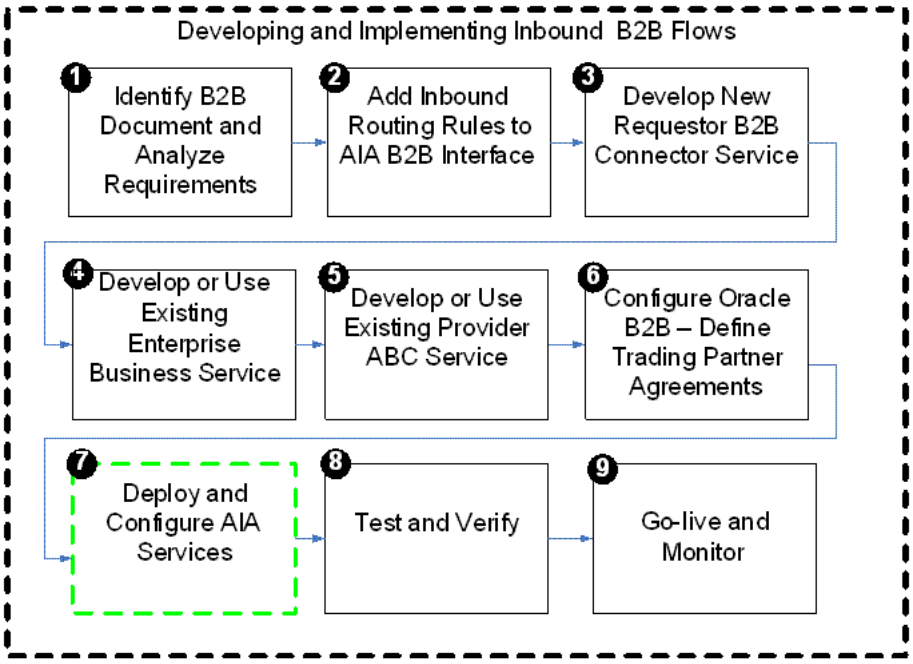
For more information about how to define trading partners and associate B2B capabilities with them, see "Configuring Trading Partners" in *Oracle Fusion Middleware User's Guide for Oracle B2B*.

In addition, for EDI-based outbound B2B flows, Oracle B2B can be configured for batch processing of outbound documents. Parameters such as the batch size and time-out can be configured in Oracle B2B without having to make any changes to the AIA layer.

21.8 Step 7: Deploying and Configuring AIA Services

The next step, as shown in Figure 21–22, is to deploy the AIA services. You can deploy the services to a target Oracle SOA server using Oracle JDeveloper.

Figure 21–22 Step 7: Deploying and Configuring AIA Services



If any domain value mapping (DVM) and configuration files are used by the AIA services required for the outbound integration, you must enter data corresponding to your B2B configuration.

You can also use the Project Lifecycle Workbench application to create a bill of material XML file for the AIA project, which can be used to autogenerate a deployment plan. This deployment plan can be used to deploy the AIA services and resources that constitute the integration project in multiple development, test, and production environments.

For more information about generating bills of material, see Chapter 3, "Working with Project Lifecycle Workbench."

For more information about generating deployment plans, see Chapter 8, "Generating Deployment Plans and Deploying Artifacts."

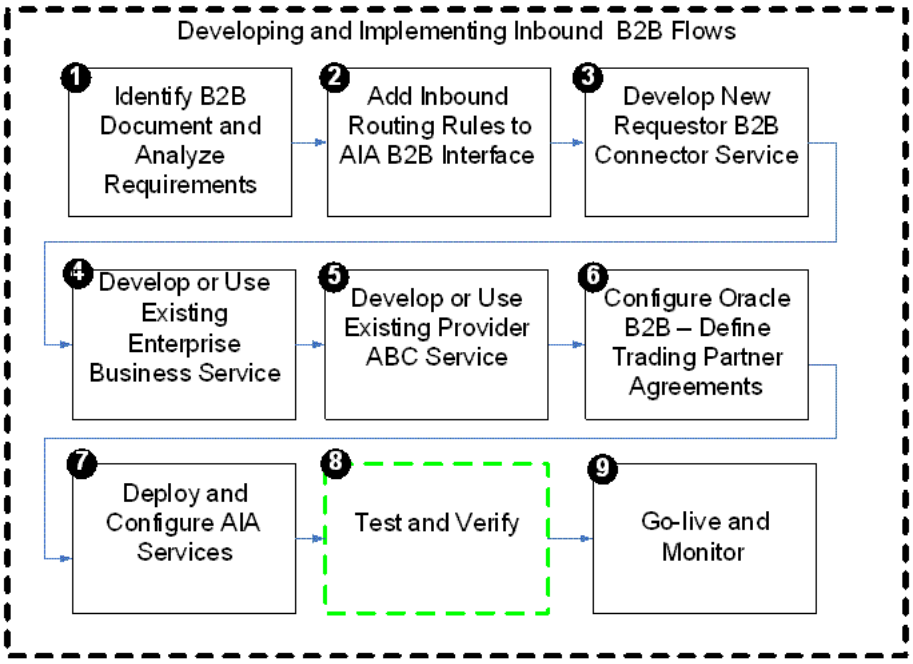
In addition, configure the AIA Error Handling framework and set up appropriate roles to be notified of errors in AIA flows.

For more information about error handling, see "Setting Up Error Handling" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

21.9 Step 8: Testing and Verifying

The next step, as shown in Figure 21–23, is to test and verify. Before you go live with your B2B integration flows with your trading partners, Oracle AIA recommends that you complete a sequence of tests for your newly developed or deployed AIA services, which constitute the B2B integration flow.

Figure 21–23 Step 8: Testing and Verifying

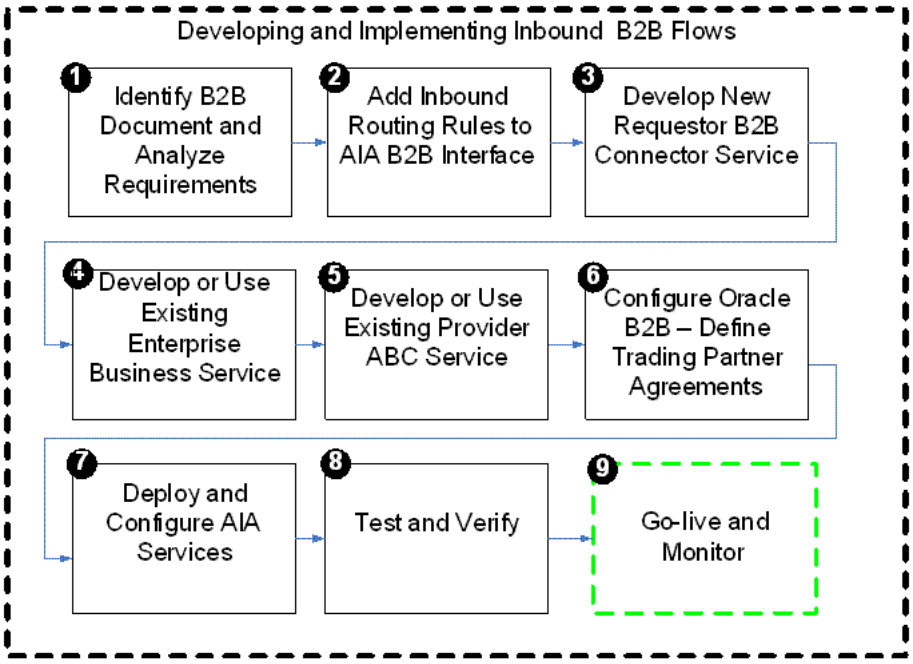


For more information, see Section 20.8, "Step 7: Testing and Verifying."

21.10 Step 9: Going Live and Monitoring

The final step, as shown in Figure 21–24, is to go live with your trading partner for the outbound B2B document flow. The AIA and Oracle B2B deployments are duplicated in the product servers and rolled out to the end users.

Figure 21–24 Step 9: Going Live and Monitoring



For more information, see Section 20.9, "Step 8: Going Live and Monitoring."

22

Describing the Event Aggregation Programming Model

This chapter provides an overview of the Event Aggregation programming model and discusses how to implement it.

This chapter includes the following sections:

- Section 22.1, "Overview"
- Section 22.2, "Implementing the Event Aggregation Programming Model"

22.1 Overview

The Event Aggregation programming model provides a comprehensive methodology for business use cases in which event, entity, and message aggregation is necessary.

For example, Event Aggregation may be needed in a case in which multiple events are being raised before the completion of a business message, and each incomplete message triggers an event, which causes a business event in the integration layer.

The Event Aggregation programming model helps to create a coarse-grained message (event) from fine-grained messages (events) generated at discrete intervals. The messages, which are generated at certain intervals, may be incomplete or duplicates.

The Event Aggregation programming model can be used for relationship- and time-based aggregation.

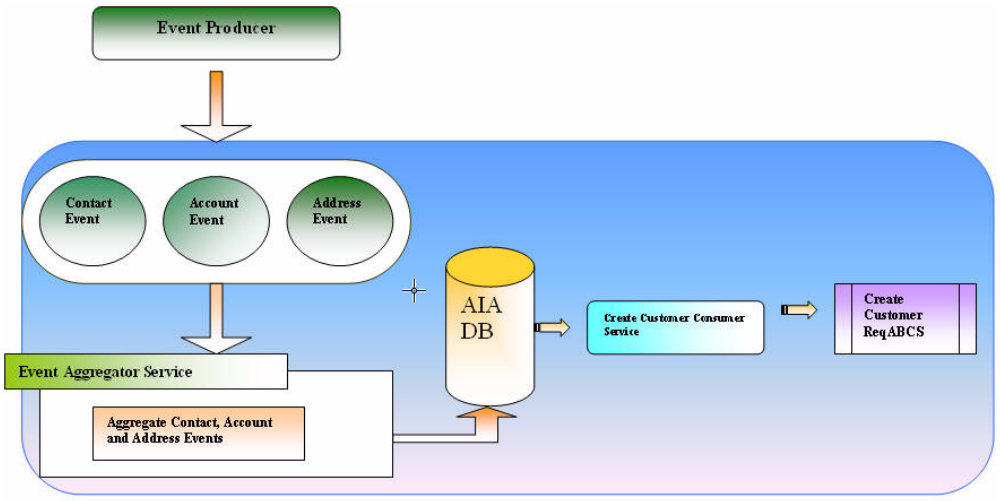
The Event Aggregation programming model provides:

- Synchronization of an entity, providing a single, holistic view of the entity.
- Consolidation of several fine-grained events into a single coarse-grained event.
- Merging of duplicates of the same event.
- Increased performance.

Parallel simulations of fine-grained applications usually generate a large number of messages. The overhead for sending these messages over a network can dramatically limit the speed of a parallel simulation. In this case, event aggregation can increase the granularity of the application and reduce the communication overhead.

The Figure 22–1 illustrates how Event Aggregation can be achieved in a business integration scenario. Create Customer is a coarse-grained event and Create Contact, Create Account, and Create Address are the fine-grained events that are produced by the Event Producer. The Event Aggregator service can be used to consolidate all of them and raise a single coarse-grained event.

Figure 22–1 Event Aggregation Service Raising a Single Coarse-Grained Event from Multiple Fine-Grained Events



22.1.1 Event Producer

The Event Producer produces the messages that are aggregated. The messages produced could be incomplete and related to or dependent on other messages.

For example, the Event Producer could be a Siebel CRM system in which a new Account object is created, triggering an associated event. This Account entity may have a Contact entity as a child object, which may raise another fine-grained event when it is created along with the Account entity.

22.1.2 Event Aggregator Service

The Event Aggregator Service consolidates fine-grained events and then raises a single coarse-grained event. Implement the relationship between the Contact, Account, and Address events using Java or PL/SQL.

There are two parts to the Event Aggregator Service.

The first part implements the actual programming logic to maintain the aggregation and relationship between the entities.

The second part is the service wrapper that invokes this programming logic from the external client. If the programming logic is developed using PL/SQL, then these database objects can be exposed using a database adapter interface. If the programming logic is developed in Java, then the Java object can be exposed using the Web Services Invocation Framework (WSIF) interface.

Oracle recommends the use of BPEL to serve as the front end of the Event Aggregator Service. When the Java or PL/SQL object is invoked, it may fail for various reasons, in which case they must be handled by notifying the Event Producer. BPEL provides fault and exception handling functionality that makes it a good choice for this scenario.

Oracle recommendeds the use of Java to implement the programming logic that maintains the relationships between entities. This is because extensibility, modularity, and exception handling are comfortable with Java.

22.1.3 Consumer Service

The real event aggregation happens in the database. This is a time-based aggregation, which means that the Consumer Service spawns a thread to poll the table object with the help of the database adapter and looks for the messages pushed into the table. The polling occurs based on a configurable time interval. The Consumer Service fetches all messages that fall into that particular time interval and passes them to the requestor application business connector (ABC) service.

After the messages are fetched from the database, the Consumer Service deletes them.

22.2 Implementing the Event Aggregation Programming Model

Implementing the Event Aggregation programming model involves creating an Event Aggregation Service and a Consumer Service, and implementing error handling.

Using this approach, the aggregation occurs in the database layer with the help of a single self- referencing table and stored procedures. The self-referencing aggregation table structure supports multilevel relationships between entities.

The stored procedures help populate the aggregation table upon appropriate event generation. This also helps to eliminate duplicate records for first-level objects. The stored procedure obtains an optimistic lock before updating records in the aggregation table.

Use Case: Customer Master Data Management, with Siebel CRM

This implementation discussion is based on this use case.

In the Oracle Customer Master Data Management (MDM) Customer process integration flows, the Siebel CRM application has create and update triggers defined at the business layer level. Any update or create action can potentially lead to multiple events being raised for integration. Therefore, aggregate these events and process them in batches, instead of processing each fine-grained event individually.

These events may be raised on these business entities: Account, Contact, and Address.

These relationships between the Account, Contact, and Address entities must be maintained throughout the aggregation:

- An Account can have one or more Contacts and Addresses attached to it.
- A Contact can have one or more Addresses attached to it.
- A Contact and Address can be shared across multiple Accounts.

22.2.1 Creating the Event Aggregation Service

This section discusses the creation of the Event Aggregation service, including how to:

- Create the PL/SQL objects
- Create the database service and aggregate service

22.2.1.1 Creating the PL/SQL objects

To create PL/SQL objects:

1. Create a table object "AIA_AGGREGATED_ENTITIES" in the database as illustrated in Figure 22–2.

Figure 22–2 Example for Creating Table Object

```
CREATE TABLE "AIA_AGGREGATED_ENTITIES"
  (    "AGGREGATED_ENTITY_ID" NUMBER,
       "PARENT_AGGREGATED_ENTITY_ID" NUMBER,
       "ENTITY_ID" VARCHAR2(150 BYTE) NOT NULL ENABLE,
       "ENTITY_TYPE" VARCHAR2(32 BYTE) NOT NULL ENABLE,
       "CREATION_DATE" DATE,
       "LAST_UPDATE_DATE" DATE,
       "IS_FIRST_CLASS_ENTITY" VARCHAR2(1 BYTE),
       "LANGUAGE" VARCHAR2(50 BYTE),
       "LOCALE" VARCHAR2(50 BYTE),
       "MESSAGE_ID" VARCHAR2(150 BYTE),
       "ENTERPRISE_SERVER_ID" VARCHAR2(50 BYTE)
  );

  ALTER TABLE "AIA_AGGREGATED_ENTITIES" ADD CONSTRAINT "AIA_AGGREGATED_ENTITIES_PK" PRIMARY KEY ("AGGREGATED_ENTITY_ID");

  create or replace TYPE AIA_AGGREGATOR_LIST_OF_IDS_TBL AS TABLE OF VARCHAR2(150);

  CREATE SEQUENCE  "AIA_AGGREGATED_ENTITIES_SEQ"  MINVALUE 1000 INCREMENT BY 10 START WITH 1000 CACHE 20 NOORDER  NOCYCLE ;
```

2. Create a stored procedure object "AIA_AGGREGATOR_PUB," which contains the programming logic to maintain the relationships between the Contact, Account, and Address events in the table object created in step 1 as illustrated in Figure 22–3.

Figure 22–3 Example for Creating a Stored Procedure Object

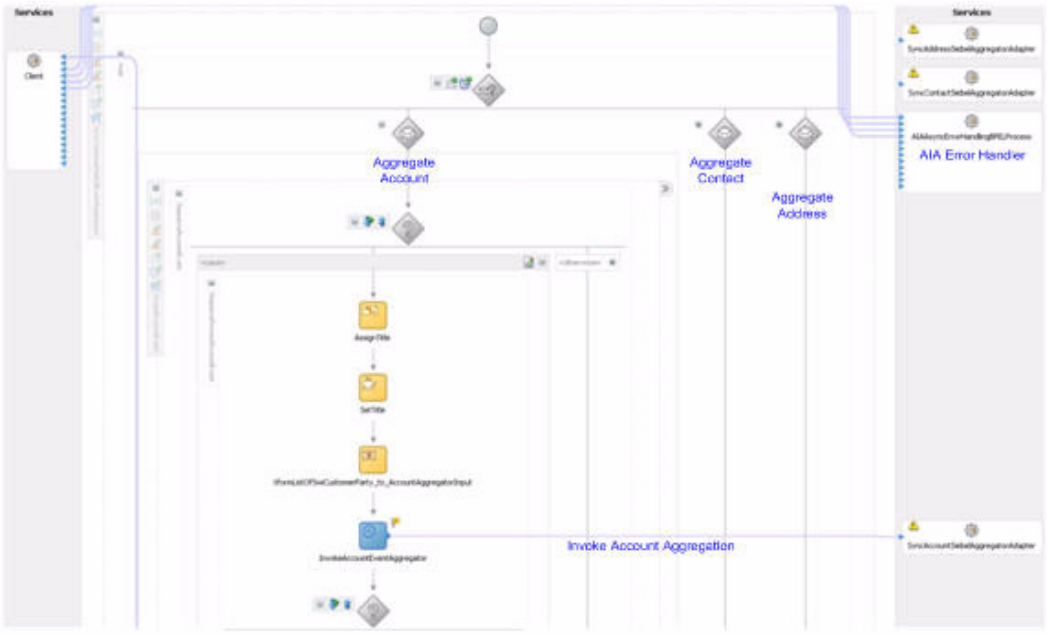
```
PROCEDURE SIEBEL_AGGREGATE_ACCOUNT(ACCOUNT_ID  IN VARCHAR2
                                 , CONTACT_IDS IN AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                 , ADDRESS_IDS IN AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                 , EP_SERVER_ID IN VARCHAR2 DEFAULT NULL
                                 , MESSAGE_ID IN VARCHAR2 DEFAULT NULL
                                 , LANGUAGE IN VARCHAR2 DEFAULT NULL
                                 , LOCALE IN VARCHAR2 DEFAULT NULL);


PROCEDURE SIEBEL_AGGREGATE_ADDRESS(ADDRESS_ID IN VARCHAR2
                                 , ACCOUNT_IDS IN AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                 , CONTACT_IDS IN AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                 , EP_SERVER_ID IN VARCHAR2 DEFAULT NULL
                                 , MESSAGE_ID IN VARCHAR2 DEFAULT NULL
                                 , LANGUAGE IN VARCHAR2 DEFAULT NULL
                                 , LOCALE IN VARCHAR2 DEFAULT NULL);

PROCEDURE SIEBEL_AGGREGATE_CONTACT(CONTACT_ID IN VARCHAR2
                                 , ACCOUNT_IDS IN AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                 , ADDRESS_IDS AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                 , EP_SERVER_ID IN VARCHAR2 DEFAULT NULL
                                 , MESSAGE_ID IN VARCHAR2 DEFAULT NULL
                                 , LANGUAGE IN VARCHAR2 DEFAULT NULL
                                 , LOCALE IN VARCHAR2 DEFAULT NULL);
```

22.2.1.2 Create the Database Service and Aggregate Service

To create the database services and aggregate service:

1. Create a BPEL project to invoke the database services created in the previous procedure as illustrated in Figure 22–4.

Figure 22–4 BPEL Project to Invoke the Database Services



In case of an unavailable database, failure of a stored procedure, or any other error at the service level, this service should implement error handling to gracefully notify the client service.

> **Note:** For more information, see Section 22.2.3, "Implementing Error Handling for the Event Aggregation Programming Model".

The external client invokes this BPEL service for Event Aggregation.

Oracle recommends that external clients (Siebel CRM, for example) post messages to a persistent queue from which the Event Aggregator Service can pick up messages for event aggregation. If implementation of this recommendation is not possible, the Event Aggregator Service can be invoked directly from the external client.

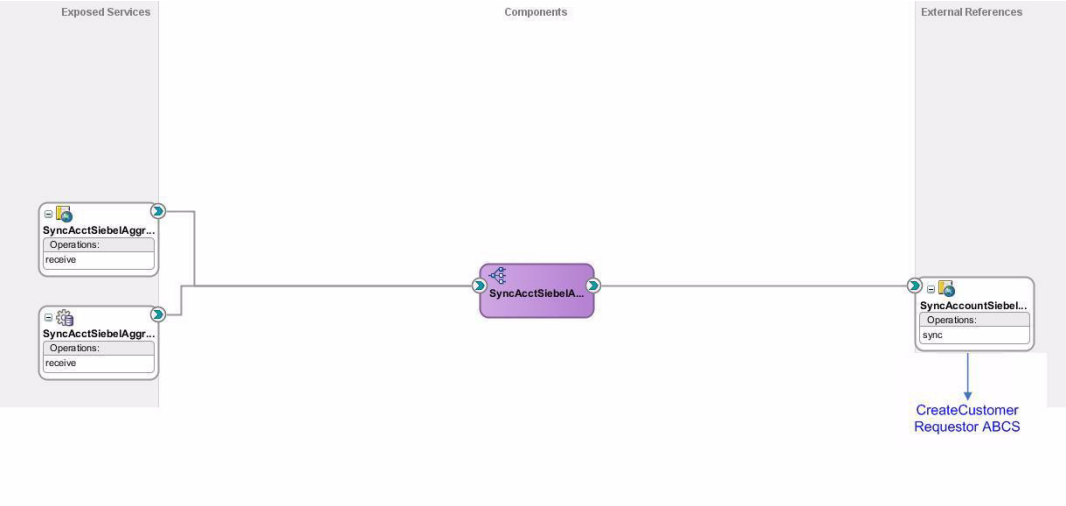
22.2.2 Creating Consumer Service

To create the Consumer Service:

1. Create a consumer service with mediator composite.

   Oracle recommends that you implement the Consumer Service using mediator, unless you must perform data enrichment. Use database adapter functionality to purge records from the database upon successful processing. Figure 22–5 illustrates how to implement Consumer Service using Mediator Composite.

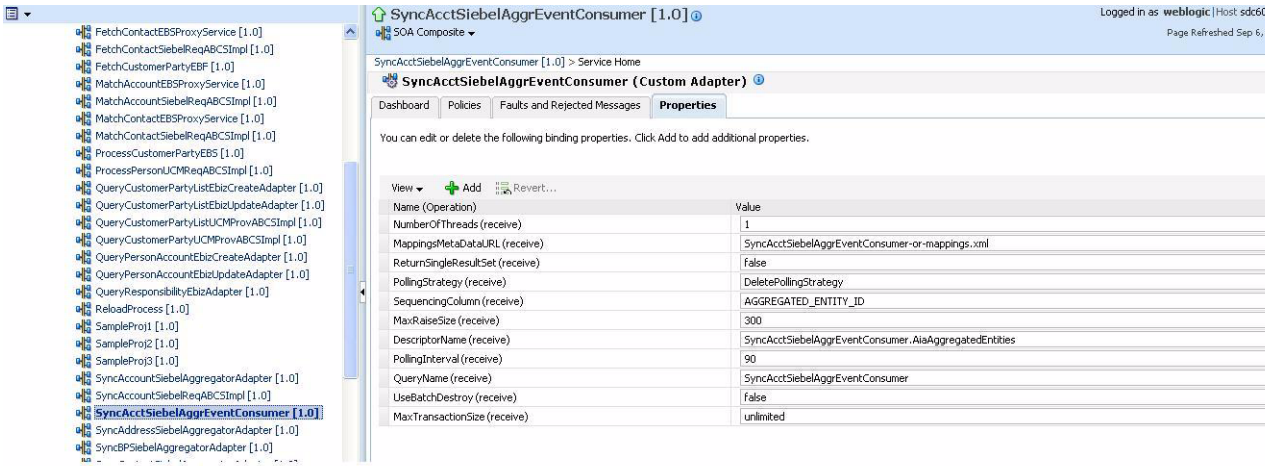
Figure 22–5 Consumer Service with Mediator Composite



2. Configure the time interval for polling on the consumer service.

   The real aggregation occurs based on this time interval set on the Consumer Service. The Consumer Service fetches messages that fall into a particular time interval and all records in the interval are processed as a batch. Figure 22–6 illustrates how to configure time interval for polling on the consumer service.

Figure 22–6 Configure the Time Interval for Polling on the Consumer Service



Note: For information on how to configure polling interval and other configurable properties, see section "Configuring PollingInterval, MaxTransactionSize, and Activation Instances" in *Oracle Fusion Middleware User's Guide for Technology Adapters*.

22.2.3 Implementing Error Handling for the Event Aggregation Programming Model

Error handling for the Event Aggregation programming model should follow Oracle Application Integration Architecture (AIA) error handling recommendations.

Note: For more information about the Error Handling framework, see "Setting Up Error Handling" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack*.

When the Event Aggregator Service errors out, whether it is due to an unavailable resource or an application error, the error should be handled in the Event Aggregator Service layer and propagated to the Producer Service. If the event generated by the Event Producer is unable to participate in the aggregation, the Event Producer should be notified.

Oracle recommends using BPEL for the implementation of the Event Aggregator Service layer. BPEL helps that the user has greater control over error handling. Regardless of the programming language in which this layer is implemented, it must be able to handle application exceptions.

If the Event Aggregator Service is implemented in PL/SQL, it should provision to propagate the OUT parameter to the Event Producer. Defining proper OUT variables for exception handling can be as simple as providing a reply consisting of either a SUCCESS or FAILURE message to the Procedure Service.

If the Event Aggregator Service is implemented in Java, it should provision to propagate the exception using the WSIF interface. Define proper exception objects to be used in the WSIF interface.

23

Establishing Resource Connectivity

This chapter describes how Oracle Application Integration Architecture (AIA) services interact with external resources and discusses inbound and outbound connectivity. The final sections describe specific guidelines for establishing connectivity with Siebel applications and Oracle E-Business Suite.

This chapter includes the following sections:

23.1 Introduction to Resource Connectivity

Participating applications drive the business processes. They are either initiators of the business process or play roles in one or more steps in the business process. The interactions of the participating applications with the business process can be either **outbound** or **inbound** from the AIA layer perspective.

A business process is a combination of outbound and inbound interactions with various participating applications.

23.1.1 Inbound Connectivity

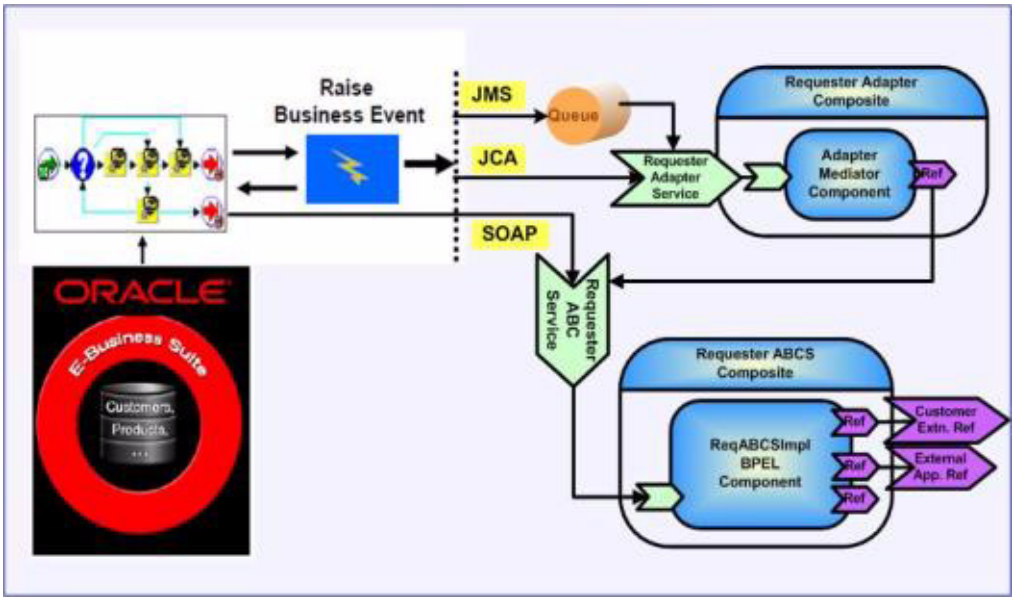
Inbound means inbound into the AIA layer, as shown in Figure 23–1.

Figure 23–1 illustrates how an inbound interaction to the business process is made by a participating application. Clicking the submit button in the order capture module within a CRM application results in the initiation of the Order Fulfillment business process.

- Services exposed by AIA services are invoked by proxies created by applications. Inbound interactions with AIA might occur due to the notification or broadcast of an event by the participating application or for retrieval of information from external sources. For example - clicking of the 'Get Account Balance' button in a CRM application module could result in invocation of an AIA service by sending a SOAP message over HTTP.
- Applications push messages to JMS queues and topics. JMS servers trigger registered JMS adapter agents and JMS adapter agents trigger AIA services. For

example, creation of a customer or submission of an order within CRM modules could trigger the broadcast of these events to AIA in the form of messages to JMS queues and topics.

Figure 23–1 Example of Inbound Connectivity



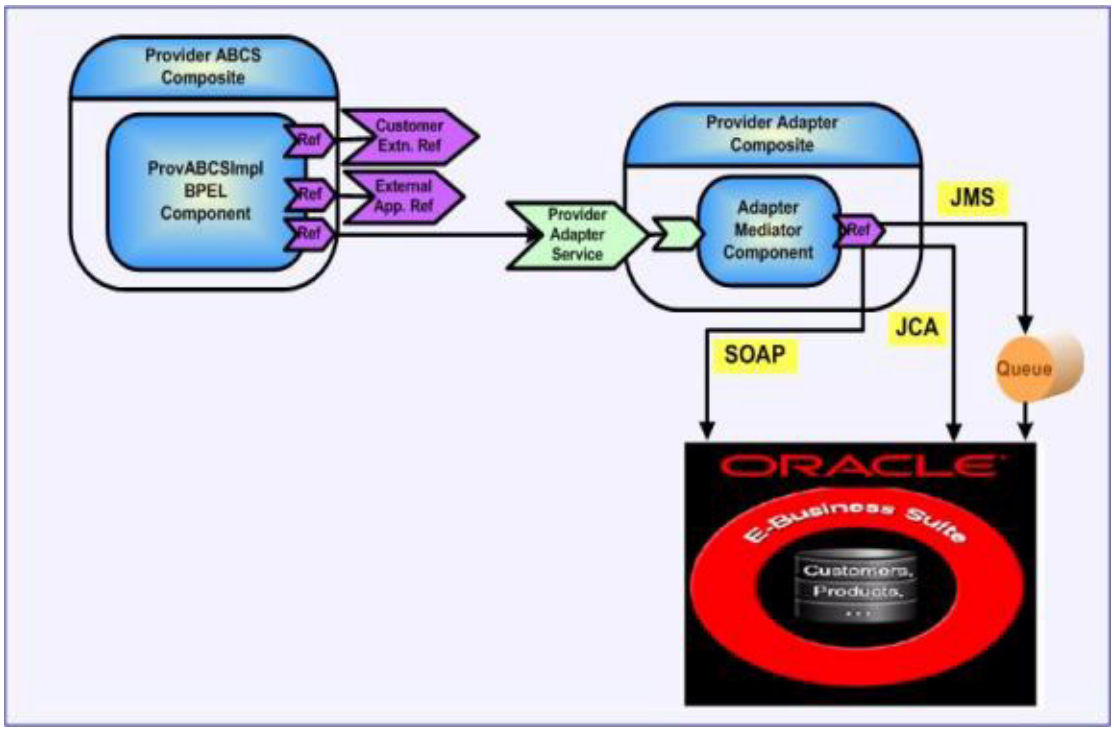
23.1.2 Outbound Connectivity

Outbound means outbound from the AIA layer, as shown in Figure 23–2.

- AIA services invoke services exposed by applications. AIA services could invoke the application services or the external services to retrieve additional information, to send business events, or to request a task be done. An example is the creation of a customer in billing application when an order is created in the CRM module or an order is submitted for an existing customer in CRM module, but that customer profile is not available in the billing application. Most manufacturing systems publish an event when an item is added however these events contain minimal information about the item. The AIA service consuming the event must invoke an application service to get complete information about the item.
- AIA services interact with applications using JCA adapters and push messages to JMS queues/topics. JMS servers trigger JMS adapter agents registered by applications.

Figure 23–2 illustrates how an outbound interaction is made by the business process. A step within that business process could result in invocation of CRM Order Capture application to send the order status updates.

Figure 23–2 Example of Outbound Connectivity



23.2 Modes of Connectivity

Participating applications use different types of mechanisms for inbound and outbound interactions.

The following sections discuss guidelines and recommendations for participating applications to interact with the AIA architecture.

This section includes the following topics:

- Section 23.2.1, "Web Services with SOAP/HTTP"
- Section 23.2.2, "When to Use Web Services with SOAP/HTTP"
- Section 23.2.3, "Session Management for Web Services with SOAP/HTTP"
- Section 23.2.4, "Error Handling for Web Services with SOAP/HTTP"
- Section 23.2.5, "Security for Web Services with SOAP/HTTP"
- Section 23.2.6, "Message Propagation Using Queues or Topics"
- Section 23.2.7, "Ensuring Guaranteed Message Delivery"
- Section 23.2.8, "When to Use JCA Adapters"

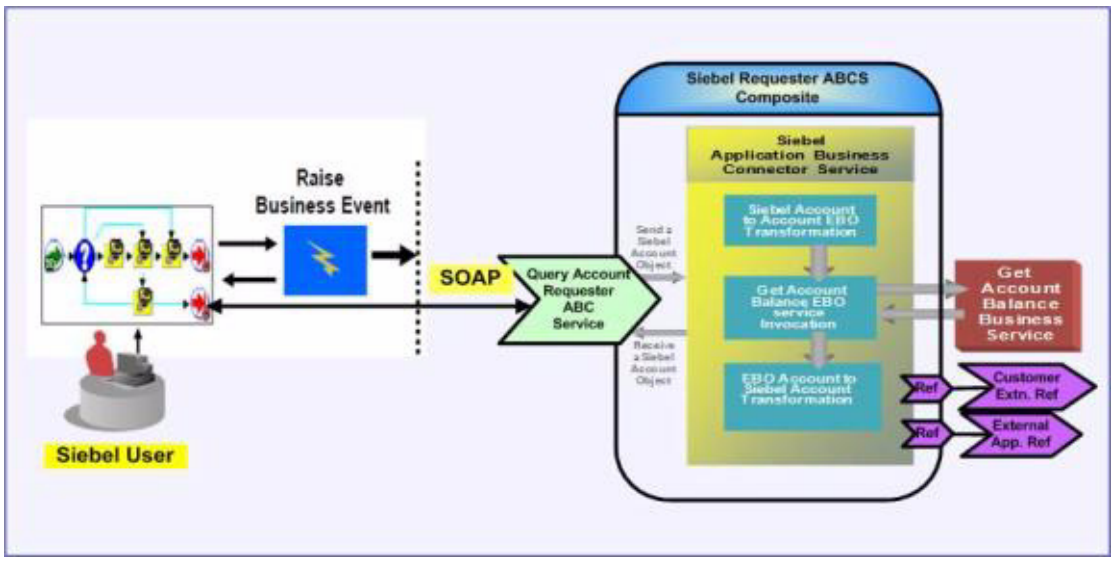
Note: The contents of this section are general guidelines and recommendations only. Programming methodologies for AIA services can vary based on a participating application's capabilities and currently, some participating applications may not have the capabilities discussed in this section. Refer to the relevant application-specific guides for the facilities provided.

23.2.1 Web Services with SOAP/HTTP

For inbound interaction, the participating application invokes the Application Business Connector Services (ABCS) provided by AIA as Web services. The participating application uses the ABCS WSDL for invoking the ABCS Web services. The ABCS WSDLs are consumed by the participating application to create stubs or proxies. These are internally invoked in the participating application leading to the invocation of the ABCS.

AIA recommends that participating applications leverage Web services transport, as shown in Figure 23–3, for invoking AIA services only to fetch information residing in external systems. For example, querying the account balance details of a customer from billing application using CRM application by a CSR or a self-service application screen.

Figure 23–3 Illustration of Using Web Services with SOAP/ HTTP



In other types of interactions, like create customer, submit order, and so on, AIA recommends that you use either JCA or JMS transports to ensure the interaction is reliable and transactional. If the participating application does not have the capability to use either JMS or JCA transports to publish events to the AIA layer, then you can use the Web services transport, which can help in publishing the events to a queue in the AIA layer for further processing.

Since the ABCS is developed using the schemas from the participating applications, the outbound message format and the ABCS message format are the same.

Inbound Interaction

Inbound messages should have the following attributes:

- Message ID

  The Message ID is the unique identifier for a message generated by the application and stamped on the message. For example, a customer ID or an order ID would be unique in the payload.

- Message payload name

  The name of the business object should be passed.

- Participating application instance

A unique identifier for the application instance is needed to set the cross reference for the entity IDs, so this is also expected in the payload.

- Event session locale

  The locale information from the client session provides context to the request being made by the application. This helps in ensuring the response is in the same locale context. The locale specific headers should be negotiated with the application teams to decide upon the ABM header attributes. Setting and retrieving the locale specific information is a factor in the ABCS implementation during the request and response transformation steps done on ABM.

Outbound Interaction

For outbound interactions, the ABCS invokes the participating application APIs exposed as Web services. The WSDLs of the participating application Web services are consumed by the ABCS.

23.2.2 When to Use Web Services with SOAP/HTTP

You can leverage Web services with SOAP/HTTP to fetch information from the applications and provide it to the requesters. They are useful when the requirement for interaction reliability is low and operations are limited to querying the information stores.

The message format is XML (Extensible Markup Language).

This mode of connectivity is used in AIA for:

- Request-response
- Request only

23.2.2.1 Request-Response

The SOAP client uses an HTTP **GET** method to request a representation of a specified resource. The SOAP server triggers the executable and responds back with the output. This is an idempotent action, meaning that multiple identical requests should have the same effect as a single request.

For example, a Customer Sales Representative queries the Customer Master Data Management system for existence of a customer record before creating a new customer in the system.

23.2.2.2 Request Only

The SOAP client uses an HTTP `POST` to submit data to be processed (for example, from an HTML form) to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both. This is a non-idempotent action; therefore sending an identical POST request multiple times may further affect state or cause further side effects.

For example, a Customer submits an order for processing from a Composite Application UI.

23.2.2.3 Advantages of Using Web Services with SOAP/HTTP

Web services with SOAP/HTTP are:

- Platform independent
- Language independent

23.2.2.4 Disadvantages of Using Web Services with SOAP/HTTP

XML format of message processing is slower; therefore, messages must be smaller or must leverage compression techniques.

Inherently, HTTP is stateless (retains no data between invocations); therefore, it needs an explicit login for every call, leading to performance overhead.

The notion of atomic transactions with state management is not supported; workarounds are session state management, where a connection can be reused for multiple requests as discussed in Section 23.2.3, "Session Management for Web Services with SOAP/HTTP".

23.2.2.5 Important Considerations for Using Web Services with SOAP/HTTP

AIA requires that the developed and deployed Web services conform to published WS-I profiles to foster interoperability.

- Leverage WS-Security to secure message exchanges using XML Encryption and XML Signature in SOAP; alternative is using secure HTTP (HTTPS).
- Leverage WS-Addressing to insert address in the SOAP header.
- Leverage WS-ReliableMessaging to reliably deliver messages between distributed applications in the presence of software component, system, or network failures.

23.2.3 Session Management for Web Services with SOAP/HTTP

To overcome the disadvantage of HTTP being stateless and unable to support atomic transactions, session management is needed. This section includes the following topics:

- Section 23.2.3.1, "Session Types"
- Section 23.2.3.2, "SessionToken"
- Section 23.2.3.3, "Session Pool Manager"

23.2.3.1 Session Types

The three types of sessions are:

- **None**

  A new session is opened for each request and then closed after a response is sent out.

- **Stateless**

  A new session is opened for an initial request, and the session remains open for subsequent requests. Re-login occurs automatically (transparent to the user) if the session is closed. **UsernameToken** and **PasswordText** must be included as SOAP headers in the initial request to open a stateless session.

- **Stateful**

  A new session is opened for an initial request, and the session remains open for subsequent requests. Re-login does not occur automatically if the session is closed. **UsernameToken** and **PasswordText** must be included as SOAP headers in the initial request to open a stateful session.

For Stateless or Stateful modes, Web services are expected to return a SessionToken in the SOAP: HEADER. This depends on the application implementation.

23.2.3.2 SessionToken

A SessionToken is the encryption of the Session ID, UserToken, plus PasswordText.

For each Stateless or Stateful call, an updated SessionToken is returned. This is as a safety measure against replay attacks.

The process of updating the SessionToken does not close the session. Therefore, for the next call with the updated session token there is no re-login. The session remains open. The Session is closed when a call is posted with a Session Type set to `None` or a timeout occurs. After a second call you have two Session Tokens: the one returned on first call and the updated one from the second call. At this point either of these SessionTokens can be sent for a third call (which returns a third SessionToken). Posting a call with Session Type set to `None` terminates the Session ID, so all these Session Tokens become invalid.

URL and SOAP Header for the Siebel Application Examples

Example 23–1, Example 23–2, and Example 23–3 provide the URL and SOAP header for the Siebel application examples.

Example 23–1 URL for Calling the Siebel Application and Passing Login Information in the SOAP Header

```
http://sdcp1952i028.corp.siebel.com/eai_enu/start.swe?SWEExtSource=
SecureWebService&SWEExtCmd=Execute&WSSOAP=1
```

The SOAP header looks like this:

Example 23–2 SOAP Header

```
<soapenv:Header>
<UsernameToken xmlns="http://siebel.com/webservices">rreddy</UsernameToken>
<PasswordText xmlns="http://siebel.com/webservices">rreddy</PasswordText>
<SessionType
xmlns="http://siebel.com/webservices">Stateless</SessionType></soapenv:Header>
```

The response looks like this:

Example 23–3 Response to SOAP Header

```
<SOAP-ENV:Header>
<siebel-header:SessionToken
xmlns:siebel-header="http://siebel.com/webservices">0f2cnvf0Ii5qsp-zk-
SEyjl2p0JD-QdYLt1LYvARXQMZfAL9YL.THekJHI1cVjZbBGQckQN.
cIfOGPKWKwUd6E0D4LD.VS.CKWsXw...</siebel-header:SessionToken>
</SOAP-ENV:Header>
```

23.2.3.3 Session Pool Manager

For business integration flows that require a high number of concurrent request-response calls to applications, AIA recommends that you send and receive session token information rather than sending user credentials on each call.

- The Session Pool Manager is a service that manages a pool of session tokens to be reused for subsequent requests.
- The sessions are persisted either in memory or in a database.
- The login credentials and URLs are configured by an administrator in the Session Pool Manager.

The implementation of the Session Pool Manager is application-specific, taking into consideration the Web service framework implementations.

23.2.4 Error Handling for Web Services with SOAP/HTTP

This section discusses error handling for Web services with SOAP/HTTP:

23.2.4.1 For Inbound Connectivity

Participating applications should have the capability to consume WSDLs with fault messages defined.

The WSDLs provided to participating applications are generated for the coarse-grained request-response services created using BPEL (ABCS). The input and output payloads for these services are schemas provided by the participating applications. Fault schemas provided by applications aid in incorporating them in the definitions of the services (WSDLs).

23.2.4.2 For Outbound Connectivity

The WSDLs of the participating application services are consumed by the AIA services. The error handling depends on the message exchange pattern.

23.2.4.3 Request-Response and Request-Only System Errors

- Call from the participating application not successful

  The AIA layer returns HTTP **4xx Client Error** and the participating applications must have the mechanism to resubmit manually or automatically.

  Example: FMW is down and AIA service is not accessible.

- Call from the participating application successful but system resource not accessible

  The AIA layer returns **5xx Server Error** and the participating applications must have the mechanism to resubmit manually or automatically.

  Example: From the AIA service execution, the cross-reference database is not accessible.

23.2.4.4 Request-Response Business Errors

- WSDL of the participation application service has named fault

  In this case, the WSDL has a named fault with a specified fault message format. The AIA service, on encountering a business error, puts the error information in the fault message and reply back to the calling service. The participating application takes action accordingly.

  Example: An order with invalid options is pushed by CRM application to ERP application and fails validation in the ERP application. Upon receiving the fault, the AIA service constructs an appropriate error message, transforms it to the fault schema of the CRM application, and sends it back as a named fault.

- WSDL of the participation application service has no named fault

This case has two possibilities:

– WSDL response message has component and elements for specifying error

  In this case, the AIA service, on encountering a business error, would put the error information in the response message fault component and elements and reply back to the calling service. The participating application takes action accordingly.

– WSDL specifies no valid way for receiving fault information

  Two options:

  * Send back the fault as a SOAP fault if the FMW component used for the AIA service supports it.

  * Model the interaction as Request-Only and make provisions for a separate participating application Web service to receive the result.

23.2.5 Security for Web Services with SOAP/HTTP

The participating applications should receive authentication information in WS Security username token profile format. They should be able to design and decrypt the request and sign and encrypt the response. They should preferably receive authorization information in xacml format inside SOAP headers.

23.2.6 Message Propagation Using Queues or Topics

Participating applications can enqueue messages to queues (JMS, AQ, MQ, MSMQ, and so on) for inbound interactions. The participating applications construct messages for various events raised and enqueue the messages into named queues. The AIA services subscribing to these queues are triggered. These are asynchronous in nature. Publishing of a message into a queue should be part of the same transaction that occurred within the application.

Participating applications also can dequeue messages from queues (JMS, AQ, MQ, MSMQ, and so on) for outbound interactions. AIA services construct messages for various events raised and enqueue the messages into named queues. The participating applications subscribing to these queues de-queue and process the messages. These are asynchronous in nature. Publishing of a message into a queue should be part of same transaction that occurred within the AIA service.

The queuing mechanisms are asynchronous in nature. They consist of one-way calls.

23.2.6.1 Event Notification Without Payloads

For event notifications without payloads, the ABCS adapter that is subscribing to or polling the events pulls out the message from the participating application. A series of events are triggered, but there is no guarantee that the snapshot of the updated entity pulled from the participating application corresponds to the event. AIA suggests that events be user-triggered so that the data integrity is not compromised. Message sequencing is not possible in this case.

To ensure the guaranteed delivery of the event notifications, these notifications should be able to receive an acknowledgment from AIA layer and the participating applications should be able to resend or do proper error handling of these event notifications. All these events should be preserved and aggregated based on the ID or timestamp in the AIA layer and further processing should be done at regular intervals to ensure the data integrity.

23.2.6.2 Events Leading to Message Queuing

When messages are queued because of events raised, the messages capture the snapshot of the entity state. If there are a series of operations on an entity in close succession, a series of messages for the same entity with different states arrive at the queue. Due to network latencies or errors in messages being processed, the ordering of messages in such a situation cannot be guaranteed.

AIA services can process the messages in the correct sequence if some requirements are fulfilled.

- Requirements for sequencing of messages

  For the messages to be sequenced, there should be a defined set of messages that must be sequenced. In addition, every message should be identifiable as a part of a sequence.

- Identifier for the set of messages to be sequenced

  A named parameter like a **Group ID** is defined. All the messages having a common value are part of the same group or set of messages to be sequenced.

- Identifier for the message to be sequenced

  A named parameter like a **Sequence ID** is defined. The value for this parameter can be a number or date-time. This value determines the position of each message in a sequence.

23.2.6.3 When to Use Message Propagation Using Queues or Topics

Meeting either of these two criteria leads to using message propagation using queues/topics:

- Must break processes into atomic transactions
- Event triggering system cannot wait till the message is processed

23.2.6.4 Types of Queues

The two types are:

- Queues
- Topics

Queues

A Queue is a persistent storing mechanism designed for holding elements before processing.

Points to note:

- Queues are point-to-point.
- Only one consumer gets the message.
- The producer does not have to be running at the time the consumer consumes the message, nor does the consumer have to be running at the time the message is sent.
- Every message successfully processed is acknowledged by the consumer.

Considerations for Using Queues

When using queues, consider the following:

- AIA recommends that you use WLSJMS Queues for inbound and outbound interactions with participating applications.
- If the application is not compatible or does not have the ability to send or receive messages to and from AIA using WLSJMS, then any other supporting messaging should be used. For example, AQJMS or TIBCOJMS, and the required foreign JMS server setup should be done on native WLSJMS to interact with the applications.
- AIA recommends that you configure a file-based persistent store for WLSJMS queues. If you must use the database persistence for bulk messages, or for policy or business requirements, configure database persistence. Configuring the persistent store as file-based or database-based can be decided during deployment time.
- The Foundation Pack AIA installation driver automatically creates the error queues for each queue.
- To generate the error queue name, the Foundation Pack deployment plan generator scripts use the resource name from annotations to get the queue or topic name and then append "_ErrorQ" as suffix. For example, for the "AIASalesOrderQueue", the generated error queue would be "AIASalesOrderQueue_ErrorQ".
- The Foundation Pack AIA installation driver uses the deployment plans to ensure creating, configuring, or assigning the generated error queue name for each JMS queue that it creates on WLS.
- Foundation Pack creates one generic "non-XA" connection factory for connecting to all the error queues from the error resubmission utility. The name of the generic connection factory is "AIAErrorQueueConnectionFactory".
- The generic error connection factory is created during the Foundation Pack install. The error queues generation happens during the Pre-Built Integration installation based on the annotations.
- For the integration flow milestones, AIA recommends that you use the WLSJMS Queues with file-based persistent store.

Topics

A topic is a persistent storing mechanism designed for holding elements before processing. For processing, messages are delivered to multiple subscribers.

Points to note:

- Multiple consumers can get the message.
- A timing dependency exists between publishers and subscribers. The publisher has to create a subscription in order for clients to be able to subscribe. The subscriber has to remain continuously active to receive messages, unless it has established a durable subscription. In that case, messages published while the subscriber is not connected are redistributed whenever it reconnects.
- For the publish-subscribe model, AIA recommends that you use WLSJMS topics.

23.2.7 Ensuring Guaranteed Message Delivery

Guaranteed message delivery means a message initiated from the sender system is persisted until it is successfully delivered to and acknowledged by the receiver, if acknowledgment is expected. This delivery method ensures that messages are not lost under any circumstance.

The sender and receiver are not necessarily the participating applications. Rather, they are logical milestones in a business process. Multiple milestones may exist, as illustrated in Figure 23–4 and Figure 23–5.

Figure 23–4 Example of Multiple Milestones in a Business Process (1 of 2)

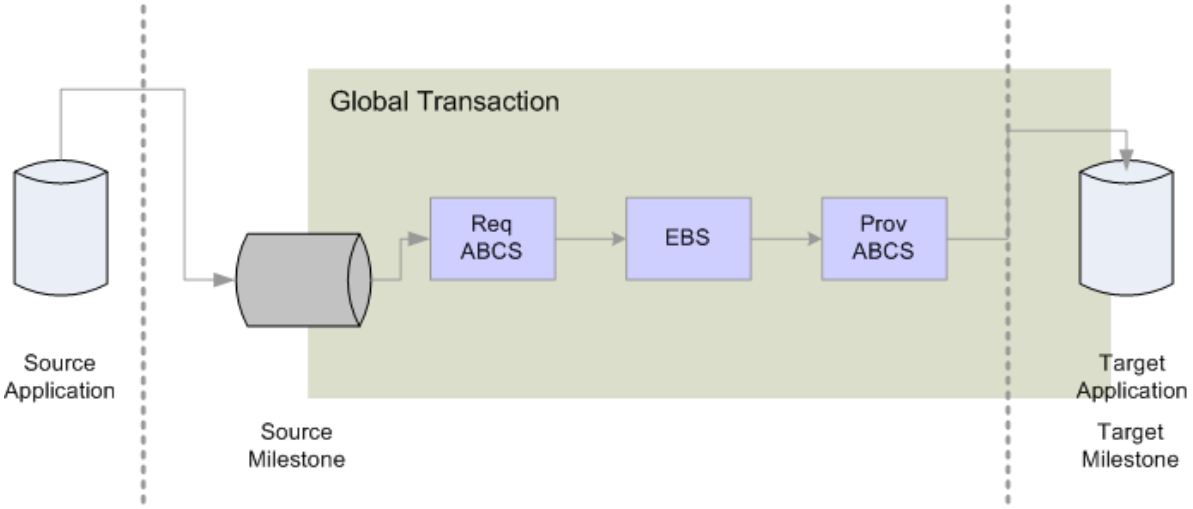
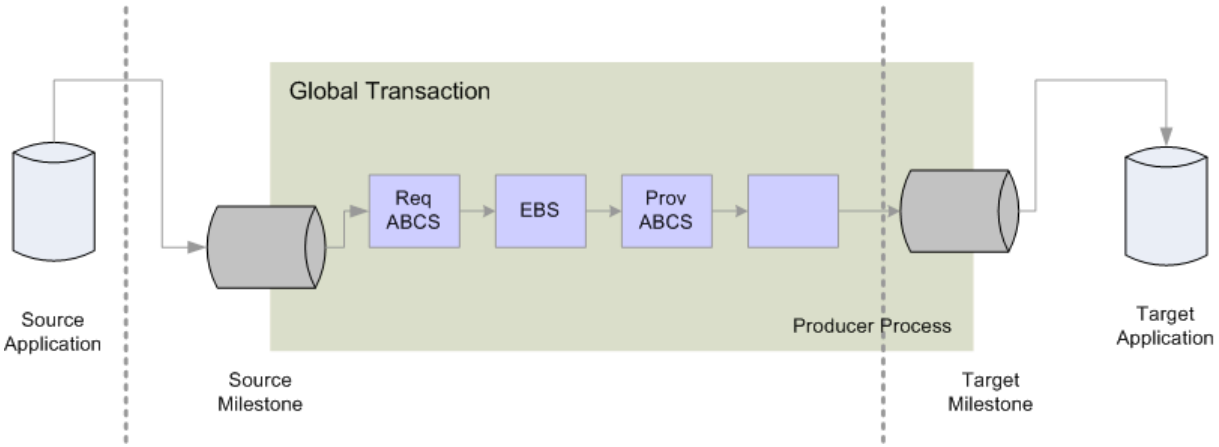


Figure 23–5 Example of Multiple Milestones in a Business Process (2 of 2)



- Temporary unavailability of any hardware or software service does not result in a lost message or a delivery failure, ensuring that the message is delivered.
- The Error Handling framework provides a way for the message to be persisted until the hardware or software service becomes available for retry, and to reprocess the message after correction or initiate a new message after discarding the existing message.
- Message delivery is ensured by treating the message processing between any two integration milestones as a unit of work and binding it in a single global transaction.

23.2.7.1 When to Use Transaction Boundaries

Use transaction boundaries when:

- Different components are involved in the processing of messages. They can be JMS consumer adapter services, BPEL processes, Mediator services, cross reference calls, and JMS Producer adapter services.
- Global transactions are enabled across all the components, and the transaction boundary is established between the integration milestones, which ensures that the messages are persisted in the source milestone until delivered to the target milestone.

23.2.8 When to Use JCA Adapters

Use JCA adapters when the application has the implementation of an adapter based on the Oracle FMW-supported JCA specifications. These adapters can be purchased from Oracle certified third-party vendors if they support the required JCA specifications.

JCA adapters should be transactions enabled. To ensure guaranteed delivery and get the participating application to enlist in the XA transactions, the JCA adapter and the application should build the capabilities which are required for building the AIA composite business processes.

The JCA adapter can be a queue or topic adapter for AQ or JMS. Also, the JCA adapter can expose the business object APIs of a particular application. The granularity of the API demands chatty conversations with the participating application. AIA recommends that an application should expose the coarse-grained API (though it might call multiple fine-grained APIs in the application's implementation). This exposure avoids chatty conversations and improves the overall performance for a business transaction.

For more information about JMS Adapters, see *Oracle Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server.*

23.3 Siebel Application-Specific Connectivity Guidelines

The following sections discuss how to establish inbound and outbound connectivity with Siebel applications:

- Section 23.3.1, "Inbound: Siebel Application Interaction with AIA Services"
- Section 23.3.2, "Web Services with SOAP/HTTP"
- Section 23.3.3, "Creating JMS Consumers to Consume Siebel Messages from JMS Queues/Topics"
- Section 23.3.4, "Outbound - Siebel Application Interaction with AIA Services"
- Section 23.3.5, "Web Services with SOAP/HTTP"

23.3.1 Inbound: Siebel Application Interaction with AIA Services

Siebel applications present requests to AIA services when the Siebel application is the driving application initiating business processes, business activities, and tasks. The Siebel application can either invoke AIA services exposed as Web services or push messages directly to JMS queues triggering AIA JMS consumers.

The format of the requesting messages can either be native to Siebel or conform to the AIA Enterprise Business Objects (EBO).

If the format is native, Siebel Tools generate schemas for the Siebel Integration Objects and provide for creating AIA services.

For more information, see *Integration Platform Technologies: Siebel Enterprise Application Integration*.

23.3.2 Web Services with SOAP/HTTP

Siebel Tools (Siebel IDE) needs AIA service WSDLs. Siebel Tools introspects the WSDLs and generates proxies to invoke AIA services at run time. Siebel Tools generates schemas for the Siebel Integration Objects, and these are used to develop AIA requester ABCS.

Perform the following tasks as part of the AIA Project Management Lifecycle for the Service Conception and Definition phase and the Service Design and Construction phase.

For more information about these AIA lifecycle phases, see Section 2.2.3, "Introduction to the Business Process Decomposition and Service Conception Phase" and Section 2.2.4, "Introduction to the Service Design and Construction Phase"

Tasks for Solution Architects in the Service Conception and Definition phase:

1. Identify the requester ABCS for the Siebel application and add them to the AIA project definition.
2. For new services, work with business analysts to capture the requirements in detail.
3. For existing services, work with business analysts to capture details of changes to be carried out.
4. Work with developers and drive the design of the services.
5. Finalize the format of the message.
6. Finalize the WSDL of the AIA requester ABCS.
7. Ensure the metadata of the service is captured in the Oracle Enterprise Repository.
8. Add the service to the deployment plan of the AIA project definition.

Tasks for Developers in the Service Design and Construction phase:

1. Analyze the Siebel requester Application Business Service definition provided by the Solution Architect.
2. Work with Siebel Application development and discuss the possible design.
3. Finalize the content of the message from Siebel.
4. Get the schema of the message from Siebel and ensure the following:
   a. TargetNameSpace - If higher than version 0, must have suffix `V<N>` where V is abbreviation for version and N is the version number. If there is no version number, it is considered to be version 0. Example 23–4 provides an example of version 1.

Example 23–4 Example of a Version 1 TargetNameSpace

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema targetNamespace="
http://siebel.com/asi/V1" xmlns:xsd=http://www.siebel.com/xml/SWICustomerPartyIO
```

   b. Custom Attributes - Attributes in Example 23–5 are required:

Example 23–5 Required Custom Attributes

```
<xsd:attribute name="Language" type="xsd:string"/>
<xsd:attribute name="Locale" type="xsd:string"/>
<xsd:attribute name="MessageId" type="xsd:string"/>
<xsd:attribute name="EnterpriseServerName" type="xsd:string"/>
```

A sample of custom attributes is provided in Example 23–6.

Example 23–6 Sample Custom Attributes

```
<xsd:complexType name="ListOfSwicustomerpartyio">
<xsd:sequence> <xsd:element name="Contact" type="xsdLocal:Contact" minOccurs=
"0" maxOccurs="unbounded"/></xsd:sequence>
   <xsd:attribute name="Language" type="xsd:string"/>
  <xsd:attribute name="Locale" type="xsd:string"/>
  <xsd:attribute name="MessageId" type="xsd:string"/>
         <xsd:attribute name="EnterpriseServerName" type="xsd:string"/
</xsd:complexType>
```

5. Construct a requester ABCS using the AIA Service Constructor.
6. Provide the WSDL from this service to the Siebel Application development team.

23.3.3 Creating JMS Consumers to Consume Siebel Messages from JMS Queues/Topics

Siebel Tools generates schemas for the Siebel Integration Objects, and these are used to develop AIA requester ABCS.

Perform the following tasks as part of the AIA Project Management Lifecycle, during the Service Conception and Definition phase and the Service Design and Construction phase.

For more information about these AIA lifecycle phases, see Section 2.2.3, "Introduction to the Business Process Decomposition and Service Conception Phase" and Section 2.2.4, "Introduction to the Service Design and Construction Phase".

Tasks for Solution Architects in the Service Conception and Definition phase:

1. Analyze the message to be pushed by the Siebel application.

   Since the Siebel application pushes the message using the Siebel Web Service framework, it is wrapped in the <SiebelMessage/> envelope. This must be stripped off in the JMS consumer.
2. Create the definition of JMS consumer solution component in the AIA Lifecycle Workbench and mark it as of suitable asset type.

Tasks for Developers in Service Design and Outline Construction phase:

1. Create a JMS Consumer Service Composite to be triggered by the message in the JMS queue and invoke the above ABCS.
2. Identify the name of the Queue.
3. Use the SOA Mediator component to create the adapter composite.
4. Annotate the `composite.xml`.

   For more information see Section 12.6, "How to Annotate the Transport Adapter Composite".
5. Harvest to Oracle Enterprise Repository.

For more information see Chapter 5, "Harvesting Oracle AIA Content".

23.3.4 Outbound - Siebel Application Interaction with AIA Services

Siebel applications accept requests from AIA services when an action or event in the Siebel application is part of business processes, business activities, or tasks. AIA services can either invoke the Siebel application exposed Web services or push messages directly to JMS queues triggering Siebel JMS consumers.

The format of the accepting messages can either be native to Siebel or conform to the AIA Enterprise Business Objects (EBO). If the format is native, Siebel Tools generates WSDLs for the Siebel Web services and provides for consumption by AIA services.

23.3.5 Web Services with SOAP/HTTP

Siebel Tools (Siebel IDE) generates Web Service WSDLs that are used to develop AIA Provider ABCS.

Perform the following tasks as part of the AIA Project Management Lifecycle for the Service Design and Construction phase:

Tasks for Developers in Service Design and Construction phase:

1. Analyze the Siebel Provider ABCS definition provided by Solution Architect.
2. Work with Siebel Application development and discuss the possible design of the needed Web services.
3. Finalize the content of the message to Siebel.
4. Get the WSDL of the Web Service and the accompanying schema from Siebel application team.
5. Put them in relevant folders in the MDS under AIAMetaData/ApplicationObjectLibrary.
6. Complete development of the Siebel provider ABCS.

23.3.5.1 Session Management

Siebel Web Service Framework supports Non and Stateless type sessions. To have a different session type, you must add SessionType in the SOAP Header.

Siebel Authentication and Session Management SOAP headers can be used to send and receive user credentials and session information. Username and password can be sent for login that invokes one of the following sessions:

- One that closes after the outbound response is sent.
- One that remains open after the response is sent.

Go to these links for details:

- http://supportweb.siebel.com/support/private/content/Bookshelf/80Siebel/books/EAI2/EAI2_WebServices25.html
- http://supportweb.siebel.com/support/private/content/Bookshelf/80Siebel/books/PDF/EAI2.pdf, page 99
- http://supportweb.siebel.com/support/private/content/Bookshelf/80Siebel/books/EAI2/EAI2_WebServices24.html
- http://globaldc.oracle.com/perl/twiki/view/AppIntegrationArchPIPDev/SessionPoolManager

Use the `Stateless` type for session management. `Stateless` keeps the Siebel session persistent.

Siebel Web Service Framework Stateless Session is independent of the Web server.

Every response has a new SessionToken that must be used in the next request.

23.4 Oracle E-Business Suite Application-Specific Connectivity Guidelines

The following sections discuss how you can establish inbound and outbound connectivity with Oracle E-Business Suite (E-Business Suite) applications:

- Section 23.4.1, "Inbound: E-Business Suite Application Interaction with AIA Services"
- Section 23.4.2, "Concurrent Program Executable"
- Section 23.4.3, "Business Event Subscription (JCA Connectivity Using OAPPS Adapter)"
- Section 23.4.4, "Outbound: Oracle E-Business Suite Application Interaction with AIA Services"

23.4.1 Inbound: E-Business Suite Application Interaction with AIA Services

The E-Business Suite application sends requests to AIA services when E-Business Suite is a driving application initiating business processes, business activities, or tasks. E-Business Suite can either invoke AIA services exposed as Web services with the help of concurrent program executable or raise business events to the AIA layer through JCA Adapter (Oracle Apps Adapter). JCA Adapter should be configured to subscribe for a particular business event.

The format of the requesting messages can either be native to E-Business Suite Application Business Message (ABM) or conform to the AIA Enterprise Business Message (EBM).

23.4.2 Concurrent Program Executable

E-Business Suite needs AIA service WSDLs. E-Business Suite should generate proxies and use them in the concurrent program executable to invoke AIA Services at run time. E-Business Suite should generate the schemas (ABMs) which are used to define the contracts (WSDLs) between AIA and E-Business Suite and to develop AIA requester ABCS.

Perform the following tasks as part of the AIA Project Management Lifecycle for the Service Conception and Definition phase and the Service Design and Construction phase:

Tasks for Solution Architects in Service Conception and Definition Phase:

1. Identify the requester ABCS for E-Business Suite and add them to the AIA project definition.
2. For new ABCS, work with business analysts to capture the requirements in detail.
3. For existing ABCS, work with business analysts to capture details of extensions to be carried out.
4. Work with developers and drive the design of the ABCS.

5. Choose the right connectivity based on the E-Business Suite capability to communicate the required ABM to AIA based on the business requirements: Select Concurrent Program Executable or Business Event Subscription.
6. Finalize the format of the message (E-Business Suite ABM).
7. Finalize the contract between E-Business Suite and AIA, that is, define the WSDL of the AIA requester ABCS.
8. Define and finalize the error or fault handling message format between E-Business Suite and AIA.
9. Define the error handling mechanism and the style of AIA errors to be displayed and logged for the E-Business Suite application user.
10. Finalize on the MEP between E-Business Suite and AIA.

Follow Best Practices and Design Asynchronous Patterns to Avoid Long Running Transactions

1. Finalize on the type of acknowledgment for asynchronous operations.
2. Ensure the metadata of the service is captured in the Oracle Enterprise Repository.
3. Add the service to the deployment plan of the AIA project definition.

Tasks for Developers in Service Design and Construction phase:

1. Analyze the E-Business Suite requester ABS definition provided by solution architect.
2. Work with E-Business Suite development and discuss the possible design.
3. Finalize the content of the message from E-Business Suite.
4. Get the schema of the message from E-Business Suite and ensure the following:

   TargetNameSpace - If higher than version 0, must have suffix V<N> where V is abbreviation for version and N is the version number. If there is no version number, it is considered to be version 0. Example 23–7 provides an example of version 0.

Example 23–7 Example of a Version 0 TargetNameSpace

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://xmlns.oracle.com/ebiz/CurrencyExchange"
xmlns:db="http://xmlns.oracle.com/ebiz/CurrencyExchange"
elementFormDefault="qualified">
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/db/top/CreatePayable
InvoiceListEbizDBAdapterV1"
xmlns=http://xmlns.oracle.com/pcbpel/adapter/db/top/CreatePayableInvoiceList
EbizDBAdapterV1
```

5. Construct a requester ABCS using AIA Service Constructor.

   Provide the WSDL from this service to the E-Business Suite development team.

23.4.3 Business Event Subscription (JCA Connectivity Using OAPPS Adapter)

Subscription for Enterprise Business Service (EBS) Workflow business events can be achieved by having the Business Event Adapter (OAPPS adapter) available in Oracle JDeveloper as a plug-in, which can be used to invoke any service to raise a business event in E-Business Suite. A WF_BPEL_Q queue gets created as a subscription to the

event and the adapter reads the events from the WF_BPEL_Q as and when a message arrives in the queue it triggers the adapter service.

E-Business Suite should generate the schemas (ABMs), which are used to define the contracts (WSDLs) between AIA and E-Business Suite and to develop AIA requester ABCS.

For some business objects, the ABMs generated have only event information. In such situations, the business object ID is extracted from the event information published and E-Business Suite is queried to get the whole object.

Perform the following tasks as part of the AIA Project Management Lifecycle for the Service Conception and Definition phase and the Service Design and Construction phase:

Tasks for Solution Architects in Service Conception and Definition Phase:

1. Identify the requester ABCS for E-Business Suite and add them to the AIA project definition.
2. For new ABCS, work with business analysts to capture the requirements in detail.
3. For existing ABCS, work with business analysts to capture details of extensions to be carried out.
4. Work with developers and drive the design of the ABCS.
    a. Choose the best connectivity based on the E-Business Suite capability to communicate the required ABM to AIA based on the business requirements: Select Concurrent Program Executable or Business Event Subscription.
    b. Finalize the format of the message (ABM).
    c. Finalize the contract between E-Business Suite and AIA. Define the WSDL of the AIA requester ABCS.
    d. Define and finalize the error or fault handling message format between E-Business Suite and AIA.
    e. Define the error handling mechanism and the style of AIA errors to be displayed/logged/alert mechanism to E-Business Suite user.
    f. Finalize on the MEP between E-Business Suite and AIA.
    g. Follow the best practices and design asynchronous patterns to avoid long running transactions.
    h. Finalize on the type of acknowledgment for asynchronous operations.
5. Ensure the metadata of the service is captured in the Oracle Enterprise Repository.
6. Add the service to the deployment plan of the AIA project definition.

Tasks for Developers in Service Design and Construction Phase:

1. Analyze the E-Business Suite requester ABCS definition provided by the solution architect.
2. Work with E-Business Suite development and discuss the possible design.
3. Finalize the content of the message from E-Business Suite.
4. Get the schema of the message from E-Business Suite and ensure the following:

    TargetNameSpace - If higher than version 0, must have suffix V<N> where V is abbreviation for version and N is the version number. If there is no version

number, it is considered to be version 0. Example 23–8 provides an example of version 1:

Example 23–8 Example of a Version 1 TargetNameSpace

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://xmlns.oracle.com/ebiz/CurrencyExchange"
xmlns:db="http://xmlns.oracle.com/ebiz/CurrencyExchange"
elementFormDefault="qualified">
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/db/top/CreatePayable
InvoiceListEbizDBAdapterV1"
xmlns=http://xmlns.oracle.com/pcbpel/adapter/db/top/CreatePayableInvoiceList
EbizDBAdapterV1
```

5. Construct a requester ABCS using AIA Service Constructor.
6. Provide the WSDL from this service to the E-Business Suite development team.
7. Create an E-Business Suite Adapter Service Composite to be triggered by the business event in E-Business Suite and invoke the JMS producer service.
8. Identify the connection factory name and the JNDI reference.
9. Identify the business event to be subscribed.
10. Identify the schema to be confirmed for the incoming message or business event.
11. Use the SOA Mediator component to create the adapter composite.
12. Identify the target queue name, connection factory name, and the JNDI reference.
13. Annotate the composite.xml.

    For more information see Section 12.6, "How to Annotate the Transport Adapter Composite".
14. Harvest to Oracle Enterprise Repository.

    For more information see Chapter 5, "Harvesting Oracle AIA Content".

23.4.4 Outbound: Oracle E-Business Suite Application Interaction with AIA Services

The E-Business Suite application accepts requests from AIA services when AIA service, composite business process, or Enterprise Business Flow is initiating a request to E-Business Suite. AIA can invoke E-Business Suite in one of the following ways:

- A PL/SQL procedure or function can be invoked using database or Oracle Apps Adapter. These adapters are available as plug-ins in Oracle JDeveloper. Oracle Apps Adapter is a wrapper over DB Adapter where it sets the FND Apps context before invoking the PL/SQL procedure/function.
- Invoke a JAVA based Web Service exposed using Business Service Objects (BSO).In Oracle- E-Business Suite, there is no direct way to expose a JAVA API as a service hosted on Oracle- E-Business Suite application server. However, it is possible to expose Oracle Application Development Framework Application Modules as services using the BSO feature provided by Oracle Applications tech-stack. These services come up in Integration Repository (IRep) and can be invoked as service from remote applications.

  For more information about BSO, see the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

- Loading data into Interface tables and calling a concurrent program to process the data. AIA can populate data into EBS interface tables using DB/Apps adapter and then call a concurrent program or post-processing APIs using DB/Apps adapter to process this data.
- A DB adapter can be made to poll from Oracle- E-Business Suite tables to initiate integration services thereof. A DB adapter must be configured with the help of the JDeveloper wizard by selecting the appropriate BusinessEvents or API. JDeveloper creates .sql files which must be executed on the Ebiz database to create subscriptions for listening to the events.

The format of the outgoing messages should be in the native format to E-Business Application Business Message (ABM).

The development and design tasks are similar to the inbound connectivity as discussed above. The architect or designer should identify how the business functionality is exposed in E-Business and identify the available path from the four ways listed above to connect to E-Business. Based on the general guidelines for each transport, the architect or designer should decide which transport would be suitable for the given business requirement.

23.5 Design Guidelines

AIA services are leveraged to implement tasks, business activities, and business processes.

These common design and development guidelines are applicable to all interactions using different resources and are applicable to both inbound and outbound interactions.

The following points should be considered depending on the type of pattern:

- Push event notifications without message
  - Event notification with guaranteed delivery requirement - leverage JCA Adapter, if available, otherwise use queues
  - Event notification for non-critical situations - use Web services
- Push events with message
  - Request event soliciting information and waiting for information - leverage JCA Adapter, if available, or use Web services
  - Request event propagating state change - leverage JCA Adapter, if available, otherwise use queues

  For more information see Chapter 27, "Working with AIA Design Patterns".

24

Using Oracle Data Integrator for Bulk Processing

Bulk data processing is the processing of a batch of discrete records between participating applications. Oracle AIA uses Oracle Data Integrator (ODI) , a component of Oracle Fusion Middleware SOA Suite to perform bulk data integrations. This chapter provides an overview of design patterns for AIA ODI architecture and describes how to handle high volume transactions with Xref table, build ODI projects, use XREF knowledge module, work with ODI, work with Domain Value Maps, use Error Handling, use ODI Ref functions and how to publish the Package and Data Model as Web Service.

This chapter includes the following sections:

For information about using Oracle Data Integrator, see *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator.*

24.1 Introduction to Design Patterns for AIA-Oracle Data Integrator Architecture

Since Oracle Data Integrator data transfer is always point-to-point, the source and target systems must be capable of processing batch loads of data. An integration project should not adopt Oracle Data Integrator as a solution if there is a limitation in the number of rows that can be processed either on the source side or on the target-side application.

This section describes AIA-approved design patterns for using Oracle Data Integrator with AIA architecture. Design patterns approved by AIA are:

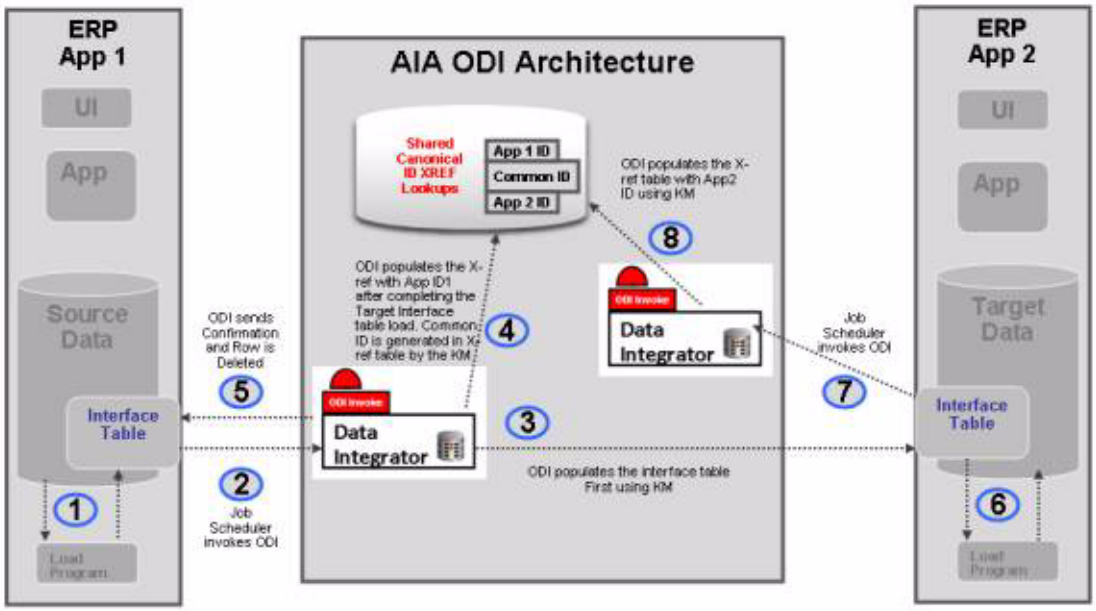
- Section 24.1.3, "High Volume Transactions with Xref Table"
- Section 24.1.4, "Intermittent High Volume Transactions"

24.1.1 Initial Data Loads

In this design pattern, shown in Figure 24–1, the initial set of data of a particular object is loaded from the source database to the target database; for example, loading Customer Account information or loading Invoice information into a new application database from an existing source application database. In the process, Xref data may or may not get established depending on the integration requirement.

The Oracle Data Integrator package that is developed for a specific integration cannot be reused for loading data into another participating application.

Figure 24–1 Initial Data Loads



24.1.2 How to Perform the Oracle Data Integrator Data Load

To perform the Oracle Data Integrator Data load:

> **Note:** The following sample steps describe how you perform the initial data load from ERP Application 1 to ERP Application 2 as shown in Figure 24–1.

1. The source application ERP APP1 populates the interface table using its native technology.

   Some applications can choose other strategies such as views or base tables as opposed to interface tables.

2. A job scheduler invokes the source side Oracle Data Integrator package.

3. Oracle Data Integrator extracts the data from Source Interface table and populates the Target Interface table.

4. After populating the Target interface table, you can choose to have Oracle Data Integrator populate the Xref table with *App 1 ID* and generate *common ID*.

   This step is optional.

5. Oracle Data Integrator either deletes or updates the rows that were processed from the Source interface table to indicate that the rows are processed.

6. In the target application ERP APP2, the native application extracts data from the target interface table and populates the target database, thereby generating ERP *Application 2 ID*.

7. A job scheduler on the target application invokes the Oracle Data Integrator package to populate the *Application 2 ID* onto the Xref table matching on the *Common ID*.

For more information about Oracle Data Integrator, see *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

24.1.3 High Volume Transactions with Xref Table

Whenever a need exists for a high-volume data transfer, AIA recommends using the Oracle Data Integrator solution for data transfer between applications. Using this approach, the Oracle Data Integrator package transfers data from source to target system on a regular basis.

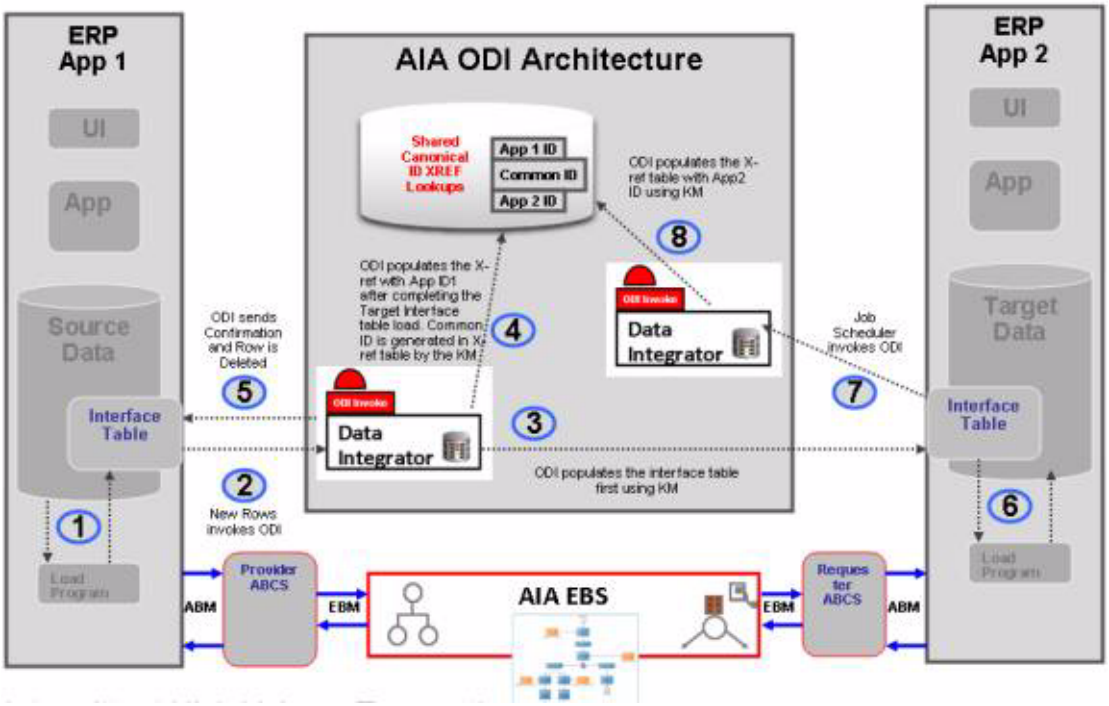
For details about how to load data, see Section 24.1.2, "How to Perform the Oracle Data Integrator Data Load".

AIA recommends that the interface tables on the source side have a mechanism to indicate processed rows.

24.1.4 Intermittent High Volume Transactions

If you have a requirement that batch loading co-exists with regular online transactions, AIA recommends the approach illustrated in Figure 24–2.

Figure 24–2 Intermittent High-Volume Transactions



In this scenario, two different flows send data from the source to the target application, one using the AIA Oracle Data Integrator approach, and the other using the standard AIA approach. The responsibility for ensuring data integrity lies with the participating applications. AIA recommends that only *new records* should be loaded using the AIA Oracle Data Integrator architecture approach.

For details about how to send data from source to target using AIA-Oracle Data Integrator architecture, see Section 24.1.2, "How to Perform the Oracle Data Integrator Data Load".

Create operations should be performed using the AIA Oracle Data Integrator approach while all other operations should be performed using AIA.

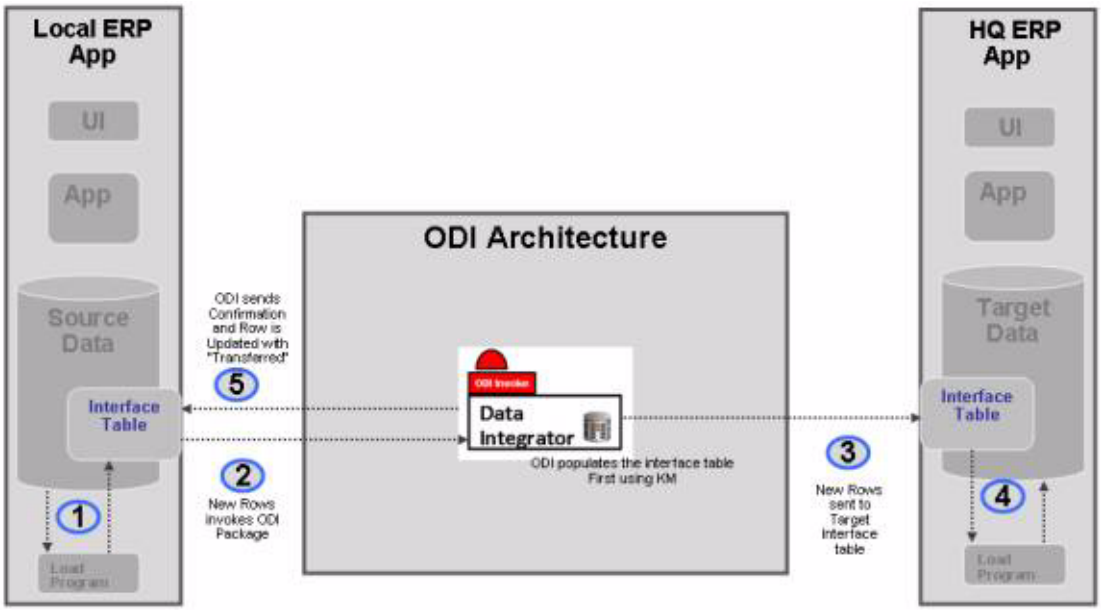
In this design pattern, do not use the AIA Oracle Data Integrator approach as an alternate route to send data when Oracle Fusion Middleware is unavailable. Instead, messages should be queued using an appropriate message queuing technology such as JMSQ, and handled using the guaranteed message technology recommended by AIA.

For more information about guaranteed messages, see Section 16.7, "Guaranteed Message Delivery."

24.2 High-Volume Transactions with Xref Table

For situations in which storing Xref data for high-volume transactions does not make sense, AIA recommends using point-to-point integration using Oracle Data Integrator, as shown in Figure 24–3.

Figure 24–3 High-Volume Transactions



For example, the headquarters of a retail store chain receives data from individual retail stores every day at the close of business. In this scenario, you need not store Xref data between each individual local store with HQ because there are not any DML operations on those data sets.

For details about how to load data, see Section 24.1.2, "How to Perform the Oracle Data Integrator Data Load".

There is no AIA component in this architecture. Local ERP applications load their interface table and invoke an Oracle Data Integrator package to send data to the HQ Interface table. After the data is processed, Oracle Data Integrator updates the local ERP application's Interface table with a *Transferred* or *Processed* status for each row.

24.3 Building Oracle Data Integrator Projects

The Bulk Data processing strategy for AIA using Oracle Data Integrator is about building point-to-point solutions, taking into account the need to set up data in the Xref, using DVM, and ensuring that the processed data can participate in AIA services at run time.

24.3.1 How to Build Oracle Data Integrator Projects

To build Oracle Data Integrator projects:

1. Define data servers.

   The source and target data server that is defined is a logical entity referring to the physical database schema chosen for bulk data processing.

   Link each data server you define to the physical data base schemas.

   For more information, see *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

2. Reverse engineer data servers.

Reverse engineer the data server to generate the various models along with the data stores.

3. Define interfaces.

   Create interfaces for each of the data stores, as required.

   In the process of creating interfaces, you specify the mapping between the source and target fields.

4. Define packages.

   The packages are the steps in which the interfaces created are run along with some intermediate steps involving the Xref and also usage of DVM. If the package chooses to implement Xref, it must use a special integration knowledge module (IKM).

   The package also has steps to clean up the source tables if you choose to do so.

24.4 Using the XREF Knowledge Module

In Oracle Data Integrator, the creation of XREF data is a two-step process. Each step is an interface.

- In the first interface, the user's source table is the source in Oracle Data Integrator and the user's target table is the target in Oracle Data Integrator.

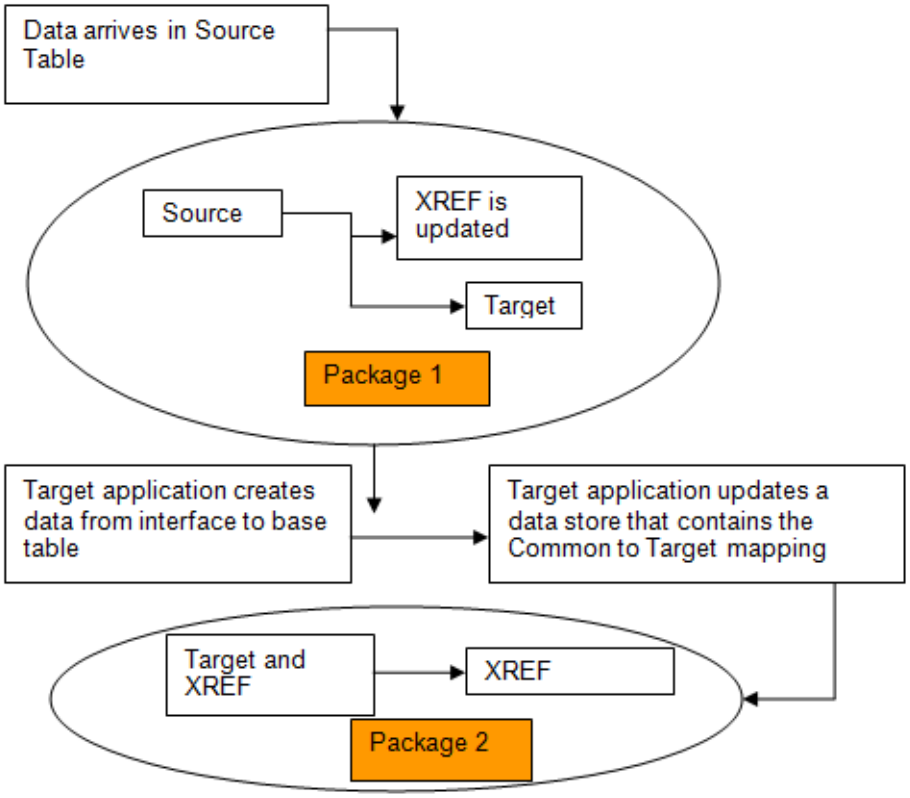
  While transporting data from source to target table, create XREF data for the source and common rows. In this process, if you want to populate any column of the target with the COMMON identifier, the Oracle Data Integrator knowledge module takes care of that too.

  Note: IIf the target interface table does not contain the placeholder for common data, you may have to either populate the source identifier or ask the application to identify a placeholder for the common value for each source row.

- In the final step, after data is posted from the interface table to the base table, the target application must identify a data store where the mapping between target identifier and common (or source) identifier that you have sent during previous interface processing is available.

  A second interface must be run in which this data store is the source in Oracle Data Integrator and the XREF table is the target. This creates the appropriate target columns in the XREF table.

Figure 24–4 Using the XREF Knowledge Module



24.4.1 What You Must Know About Cross-Referencing

Cross-referencing is an Oracle Fusion Middleware function, available through the Mediator component, and leveraged typically by any loosely coupled integration that is built on the Service Oriented Architecture (SOA) principles. It is used to manage the run-time correlation between the various participating applications of the integration.

While loading data from source tables to a target table, you must establish the cross-reference data in the SOA database just as it is done in the trickle feed architecture. While standard APIs are available in the SOA suite to populate the cross-reference tables, those APIs cannot be used in Oracle Data Integrator because that may lead to row-by-row processing as opposed to set-based processing.

The following sections describe how to load source table data to the target table and, at the same time, populate the cross-reference.

For more information about cross-referencing, see Section 25.4, "Working with DVMs and Cross-References."

24.5 Working with Oracle Data Integrator

Before working with Oracle Data Integrator, complete these prerequisites:

1. Define the master and work repository.
2. Define all topology parameters.
   a. Physical architecture (for source, target, and XREF_DATA)
   b. Logical architecture

c. Contexts

3. Define your data models.
4. Create a project.
5. Import the following Knowledge Modules into your project.
   a. KM_LKM SQL to SQL (Mediator XREF) or KM_LKM SQL to Oracle (Mediator XREF)
   b. CKM Oracle (delivered)
   c. KM_IKM SQL Control Append (Mediator XREF)

For complete details on how to set up ODI, see *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

24.5.1 How to Define Variables (Source and Target Column Names)

Because XREF column names cannot be hardcoded, two variables must be defined to hold the source and target column names. Normally, these column names are derived from the AIAConfiguration file. This section does not describe how to get that from the XML but rather it describes how to refresh this from a SQL select statement.

For complete details on variables, see "Working with Variables" in the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

To define the source and target column names:

1. Create a variable `GetSourceColumnName`.

   This variable is used to determine the column name of the `XREF_DATA` table. The **Refreshing** tab, as shown in Figure 24–5, has the appropriate SQL to get the value from the source, depending on the implementation.

Figure 24–5 Variable: GetSourceColumnName Page

2. Create a variable `GetTargetColumnName`.

   This variable is used to determine the column name of the `XREF_DATA` table. The **Refreshing** tab has the appropriate SQL to get the value from the source depending on the implementation.

24.5.2 How to Create the First Interface (Source to Target)

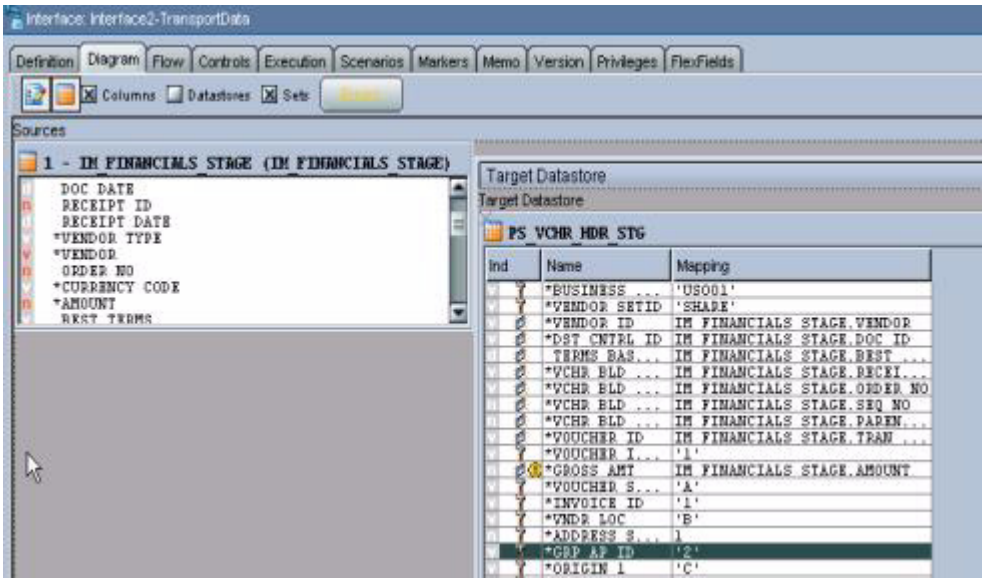
To create the first interface:

1. Create an interface.
   a. The source table in your data model should be dropped in Sources (in this example, IM_FINANCIALS_STAGE)

b. The target table (in this example, PS_VCHR_HDR_STG) appears in the target data store.

2. Provide mapping for each field, as shown in Figure 24–6.

Figure 24–6 Interface Diagram Page



3. Go to the **Flow** tab.

4. Select `LKM` as the customized KM_LKM SQL to SQL (ESB XREF).

   This has an option called `SOURCE_PK_EXPRESSION`. Pass the expression that represents the source key value in the XREF table in this option. If the source table has just one column defined as key, simply mention that column name (in this example `SEQ_NO`) for this option. If the source key has multiple columns, use the expression to derive the key value.

   For example, if two key columns are in the table and you want to store the concatenated value of those columns as your source value in XREF table, put this expression, `SEQ_NO|DOC_DATE`, in the options value. This option is mandatory.

   If you are using update/delete along with XREF, then update the other options in the LKM. If you are not using update/delete, set the option `SRC_UPDATE_DELETE_ACTION` as *None*.

5. In the IKM, choose the customized knowledge module IKM SQL to SQL (ESB Xref).

   In this module, you must define the options listed in Table 24–1:

Table 24–1 Required Options in Customized Knowledge Module IKM SQL to SQL

Term	Description
XREF_TABLE_NAME	Name of your XREF table.
XREF_COLUMN_NAME	Name of the source column in the XREF table. Enter the variable name that you defined earlier (#GetSourceColumnName) in this option.

Table 24–1 (Cont.) Required Options in Customized Knowledge Module IKM SQL to SQL

Term	Description
XREF_SYS_GUID_EXPRESSION	Select whether you want to use GUID or a sequence for the common identifier. For GUID, use SYS_GUID. For sequence, use the sequence name for this value.
XREF_ROWNUMBER_EXPRESSION	The value that goes into the ROWNUMBER column of the XREF_DATA table. Use the default value of GUID unless you require a sequence.

6. Choose CKM Oracle on the Controls tab when you select Knowledge Module.

   If you need not send the common value to the target table, ignore this step.

   If the target table does not have any placeholder for the common identifier and you are planning to supply the *source* identifier in one of the target table columns, you must use the standard mapping rules of Oracle Data Integrator to indicate what source identifier to populate in which column. This integration knowledge module does not do any work for you in that case.

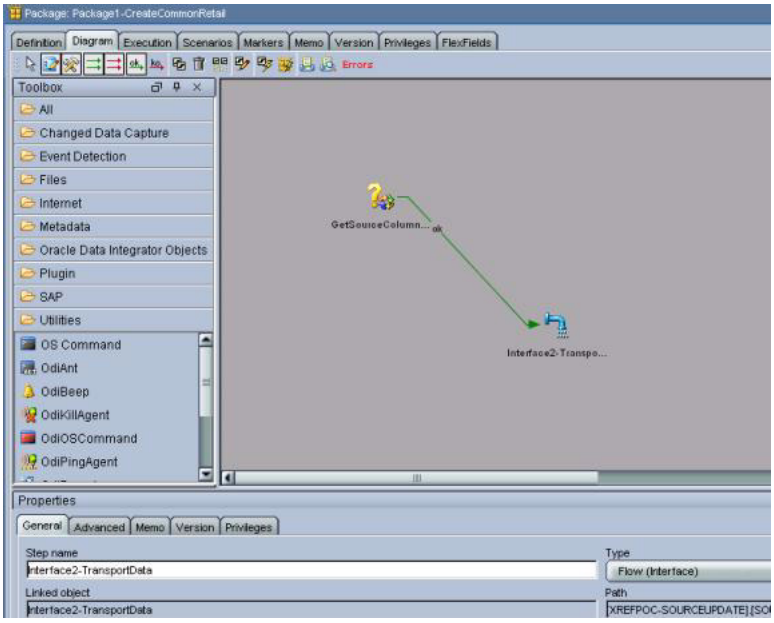
   If the target column that you want to hold the common identifier is a unique key of the target table, then put a dummy mapping on that column. This is due to an Oracle Data Integrator limitation; otherwise, the key is not shown next time you open the interface. At run time, this dummy mapping is overwritten with the generated common identifier by the integration knowledge module. Mark the UD1 column to indicate which target column the column value goes in.

7. Validate and save the interface.

To create a package for the first interface:

1. Create a package to run the interfaces, as shown in Figure 24–7.

Figure 24–7 Package to Run the Interfaces



This should contain at least two steps.

a. Refresh the variable that holds the source column name.

b. Run the interface.

Note: Every implementation adds its own error-handling steps in the package.

2. Validate and save the package.

3. Run the package.

   Most likely, this package runs as soon as the data arrives in the source table. You can achieve this by using the Oracle Data Integrator changed data capture.

   How to run a package when the data arrives in a source table is described in the following sections.

24.5.3 How to Define the XREF View in SOA

To define the XREF view in SOA:

Create an XREF View in the XREF Database as shown in Example 24–1.

Example 24–1 Creation of an XREF View in the XREF Database

```
CREATE OR REPLACE FORCE VIEW "ORAESB"."INVOICE_XREF_VW" ("ROW_NUMBER",
"XREF_TABLE_NAME", "RETL_01", "COMMON", "PSFT_01") AS
  select row_number, XREF_TABLE_NAME,
  max(decode(XREF_COLUMN_NAME, 'RETL_01', VALUE,null)) RETL_01,
  max(decode(XREF_COLUMN_NAME, 'COMMON', VALUE,null)) COMMON,
  max(decode(XREF_COLUMN_NAME, 'PSFT_01', VALUE,null)) PSFT_01
  from XREF_DATA
GROUP BY row_number, XREF_TABLE_NAME;
```

Note: Construct this view for each implementation.

24.5.4 How to Create the Second Interface (Update Target Identifier in XREF)

After the data is moved to target base tables and the target identifier is created, the data must get back to the XREF database corresponding to the source identifier to complete the loop. In the previous step, the common (or source) identifier was passed to the target system. Now the target system must provide a map between that common (or source) identifier and the target base identifier. This may come in the same interface table or it may come in a separate table. This mapping data store is used in this interface in the source. This interface is packaged and finally a separate process from target system runs that package.

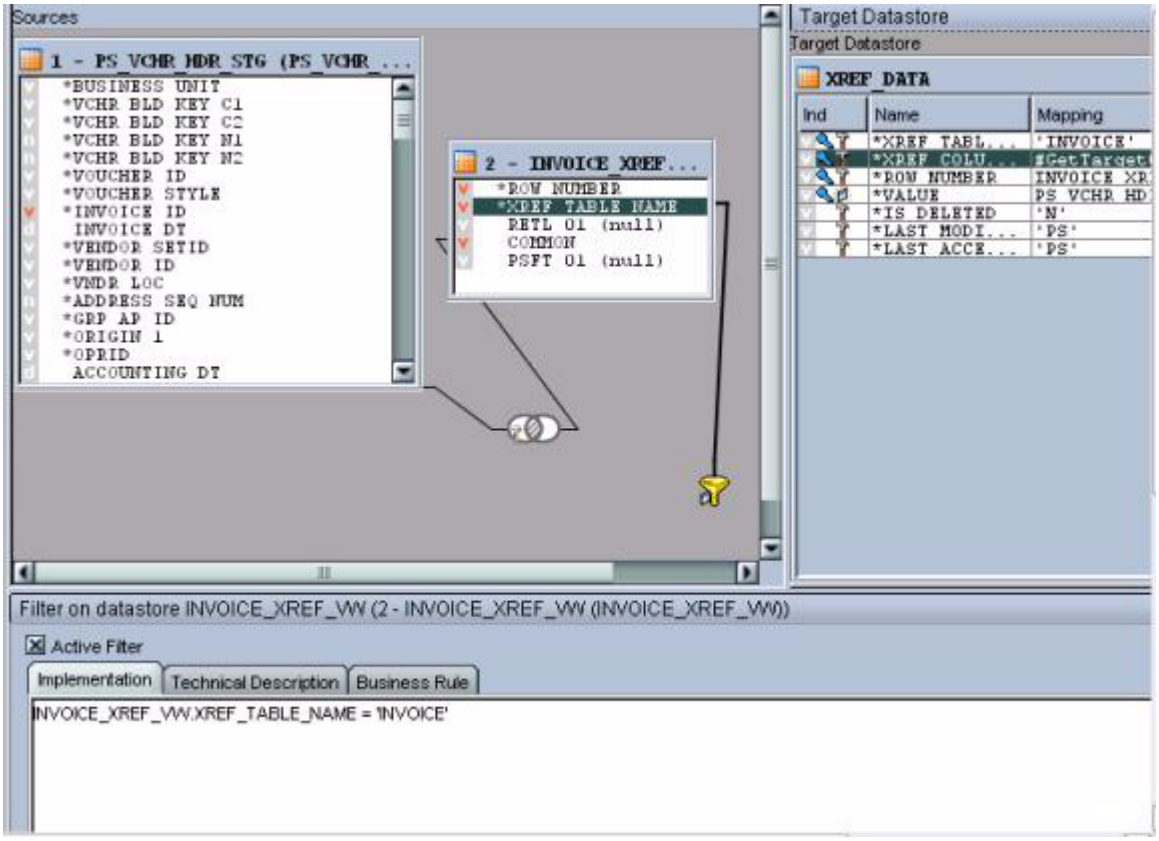
To create the second interface:

1. Create an interface for data transport.

   In the sources section, drop the XREF_VW view and the mapping data store (in this example, the same interface table in PeopleSoft PS_VCHR_HDR_STG). In the target data section, select the XREF_DATA table.

2. Apply a filter for XREF_VW with a WHERE clause to filter data from your table name only, as shown in Figure 24–8. For example, `XREF_VW.XREF_TABLE_NAME='INVOICE'`,if you are using this for INVOICE.

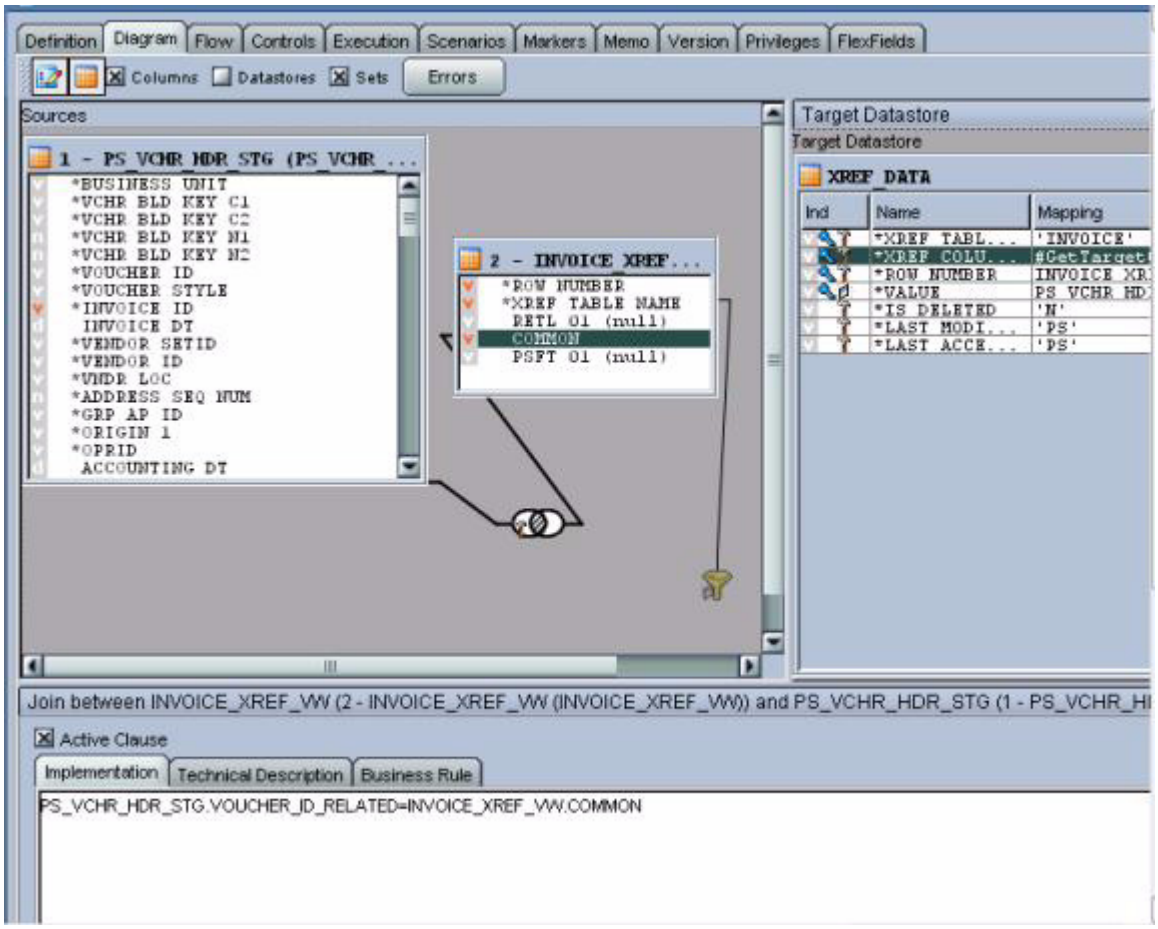
Figure 24–8 Filter for XREF_VW with a WHERE Clause to Filter Data from Your Table Name Only



3. Join the mapping data store and XREF_VW with the columns that store the common (or source) identifier, as shown in Figure 24–9.

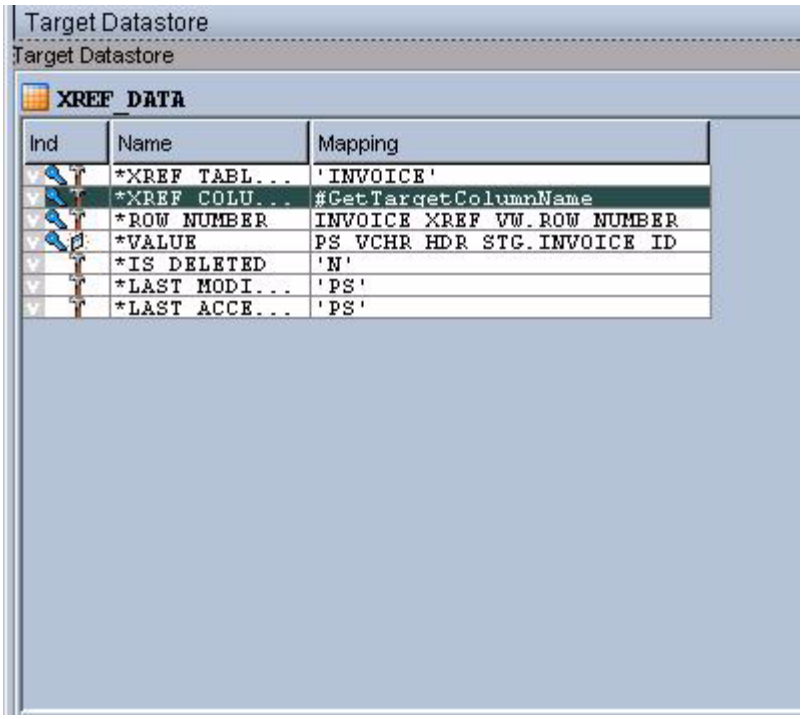
   In this example, the column of the PeopleSoft interface table that stores common data is `VOUCHER_ID_RELATED`.

Figure 24–9 Mapping Data Store and XREF_VW Joined with Columns that Store the Common ID



4. The XREF_TABLE_NAME map should be the XREF_TABLE name of the implementation.
5. XREF_COLUMN_NAME map should be `#GetTargetColumnName` (pointing to the variable that was created earlier).
6. Map ROW_NUMBER to the ROW_NUMBER of the XREF_VW.
7. The map for the VALUE field is the column that stores the target identifier in the mapping data store (in this example, the INVOICE_ID column of the PS_VCHR_HDR_STG).
8. The map for IS_DELETED is set to `N`
9. The map for LAST_MODIFIED and LAST_ACCESSED is different for each implementation.
10. Mark XREF_TABLE_NAME, XREF_COLUMN_NAME and VALUE as `Key`, as shown in Figure 24–10.

Figure 24–10 XREF_TABLE_NAME, XREF_COLUMN_NAME and VALUE Marked as Keys



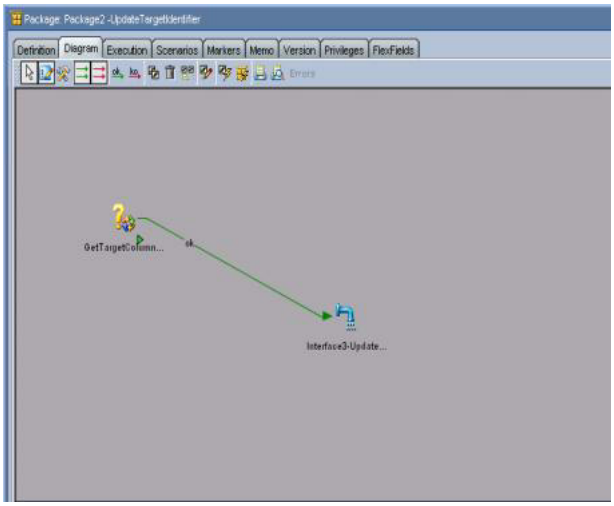
11. On the Flow tab, use the load knowledge module, `LKM SQL to Oracle`.
12. Use the integration knowledge module, `IKM Oracle incremental update`.
13. On the `Controls` tab, use the check knowledge module, `CKM Oracle`.
14. Validate and save the interface.

24.5.4.1 How to Create a Package for the Second Interface

To create a package for the second interface:

1. Create a package to run the interface, as shown in Figure 24–11.

Figure 24–11 Package Created to Run the Interface



This should contain at least two steps:

a. Refresh the variable that holds the target column name.

b. Run the first interface.

> **Note:** Every implementation adds its own error-handling steps in the package.

2. Validate and save the package.

3. Run the package.

   Most likely, this package runs as soon as the data arrives in the target mapping data store. This can be achieved by using the Oracle Data Integrator changed data capture.

24.6 Working with Domain Value Maps

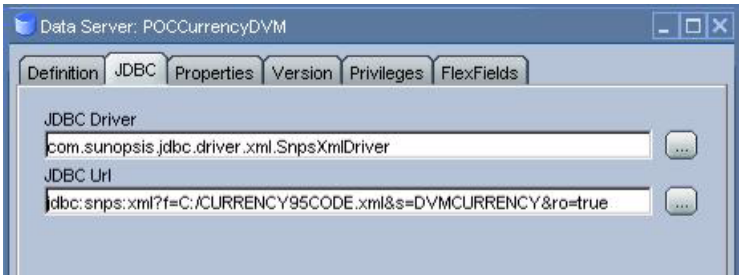
The Domain Value Maps (DVM) are available as XML files and can be used as delivered.

To use the DVM:

1. Reverse-engineer the DVM and it results in multiple relational tables.

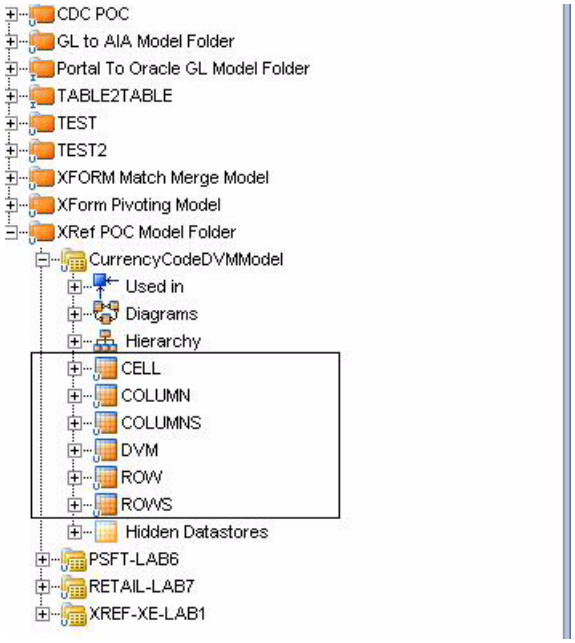
   Here is how the DVM XML is converted after reverse-engineering. We have used the XML itself in the JDBC description for the data server and not any schema for reverse engineering, as shown in Figure 24–12.

Figure 24–12 XML Used in the JDBC Description for the Data Server



   The DVM XML turns into six tables after the reverse-engineering, as shown in Figure 24–13.

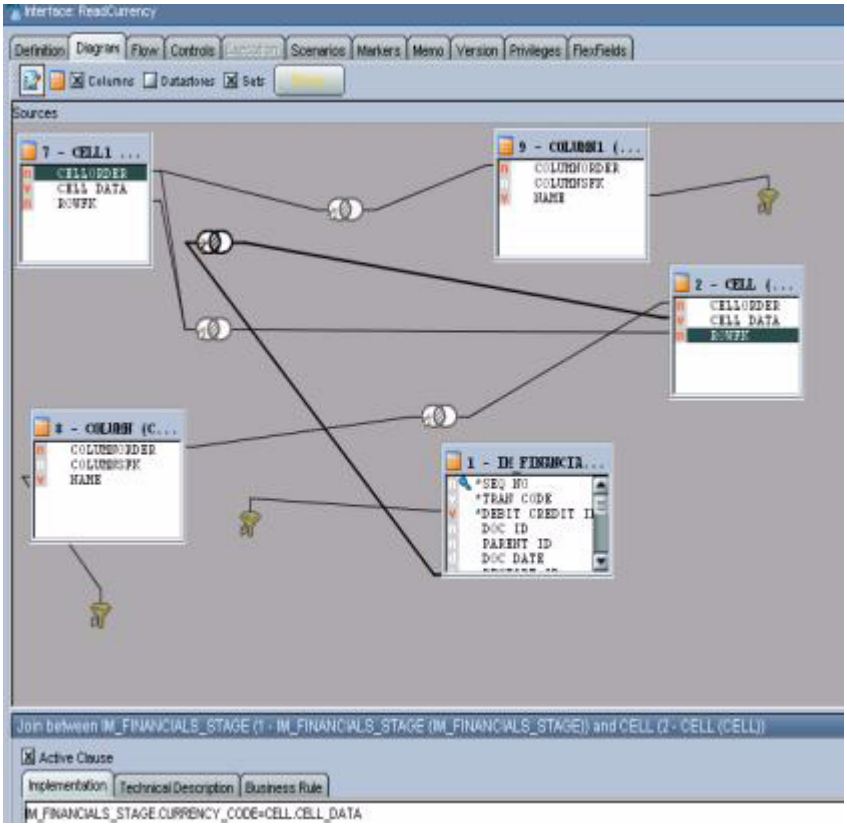
Figure 24–13 DVM XML Results in Six Tables Following Reverse-Engineering



2. Join those multiple tables to derive the corresponding mapping in the interface.
3. Join IM_FINANCIALS_STAGE with the CELL table, as shown in Figure 24–14.

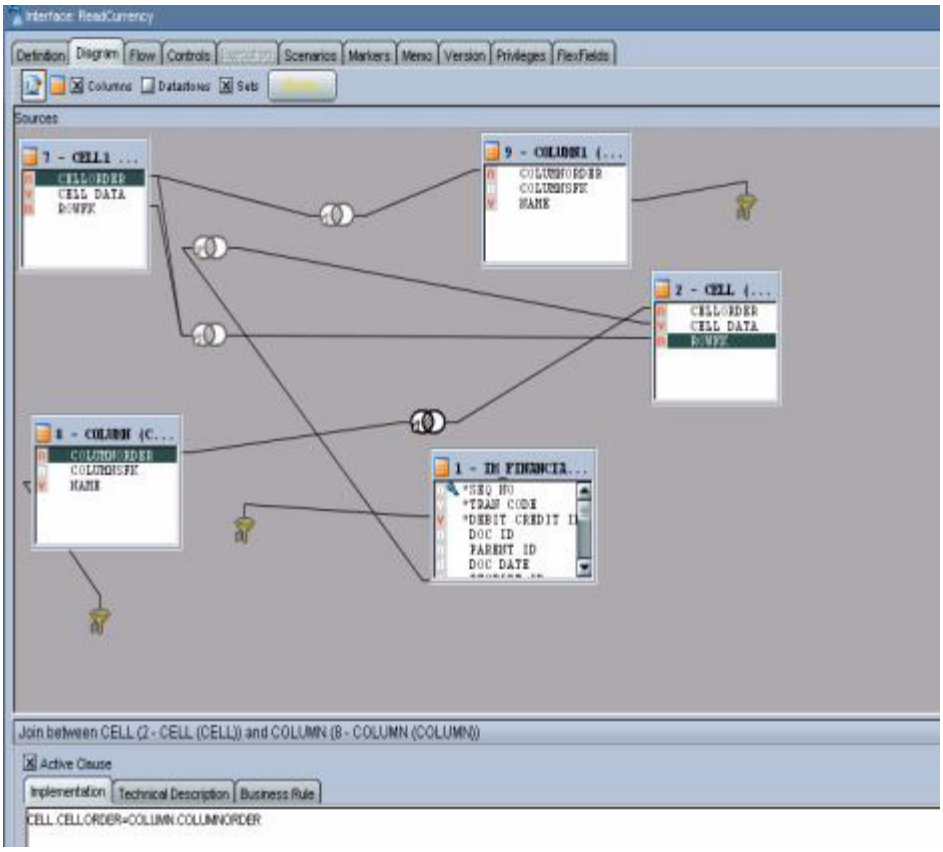
   Use the column in your main source table for DVM in the join.

Figure 24–14 Join of IM_FINANCIALS_STAGE with the CELL Table



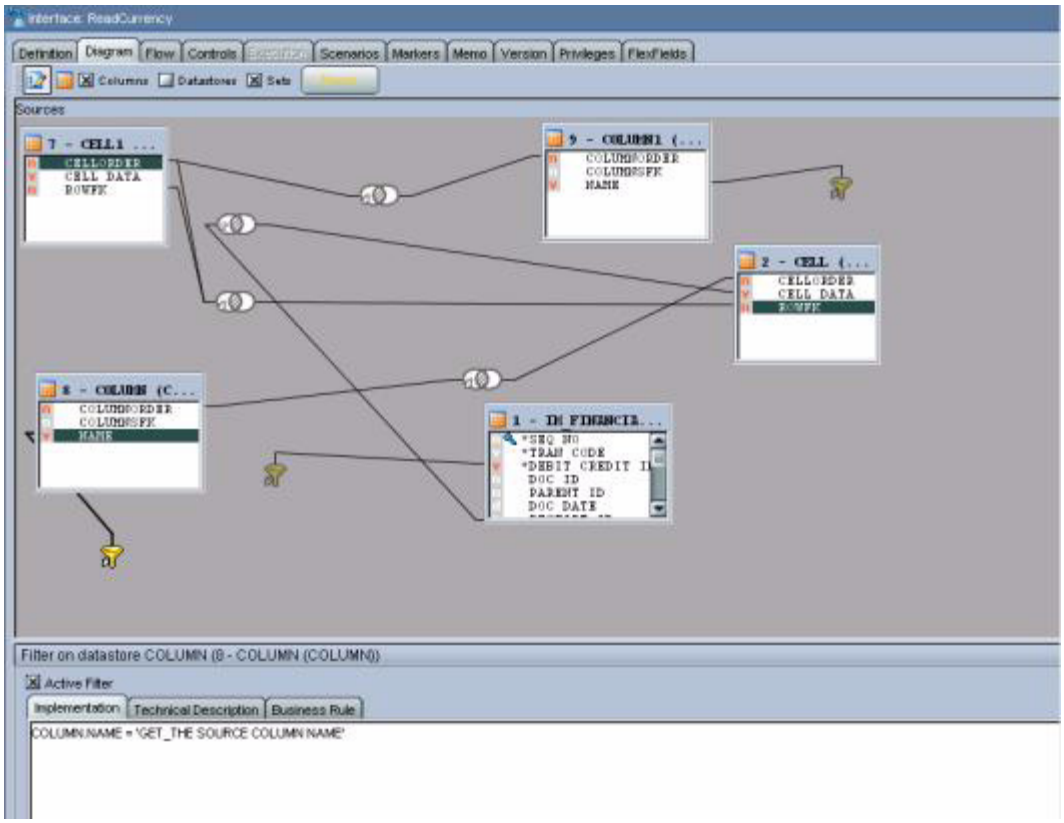
4. Join the CELL table with COLUMN table, as shown in Figure 24–15.

Figure 24–15 Join of the CELL Table with the COLUMN Table



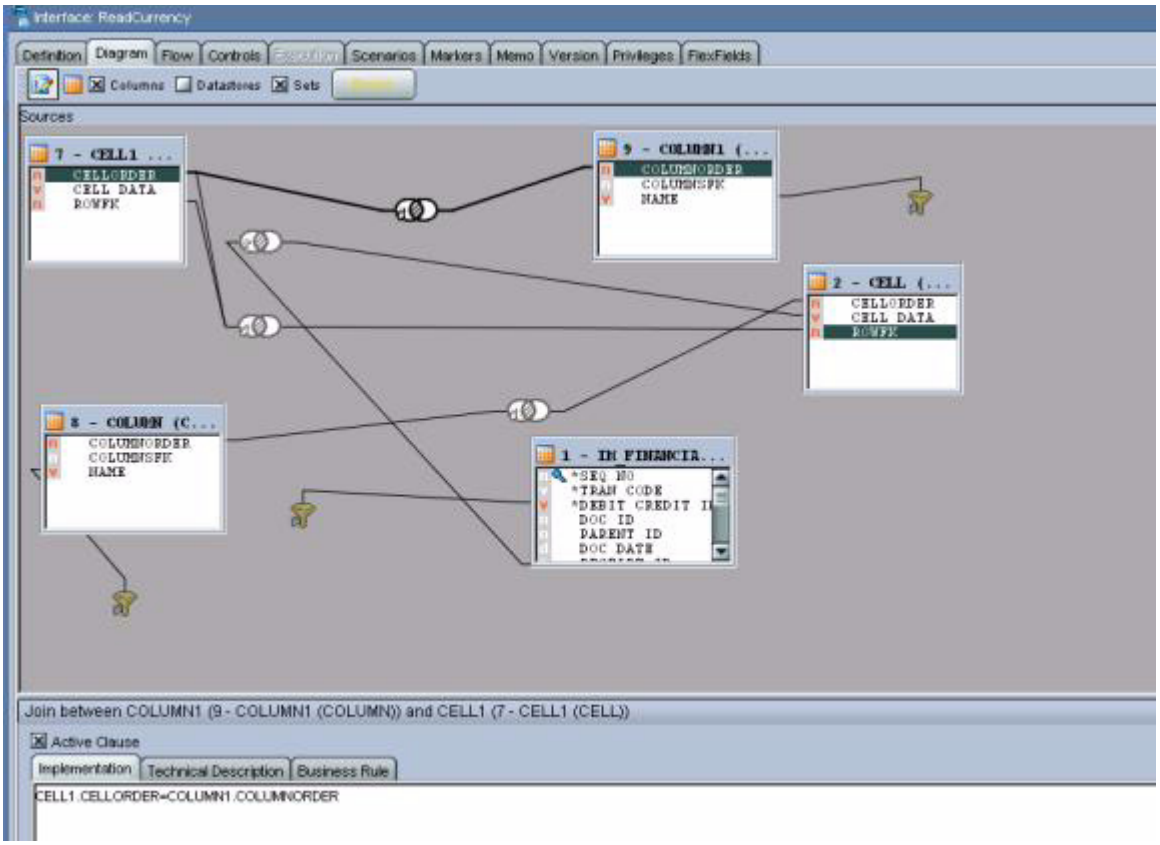
5. Add a filter for the COLUMN table (to determine the source column name), as shown in Figure 24–16. This filter can use a variable that holds the source column name.

Figure 24–16 Filter Added to the COLUMN Table



6. Drop the CELL table and COLUMN table once more in the interface. They can be renamed, but for now settle with CELL1 and COLUMN1. These duplicate sets fetch the target values.
7. Join CELL1 with COLUMN1 table, as shown in Figure 24–17.

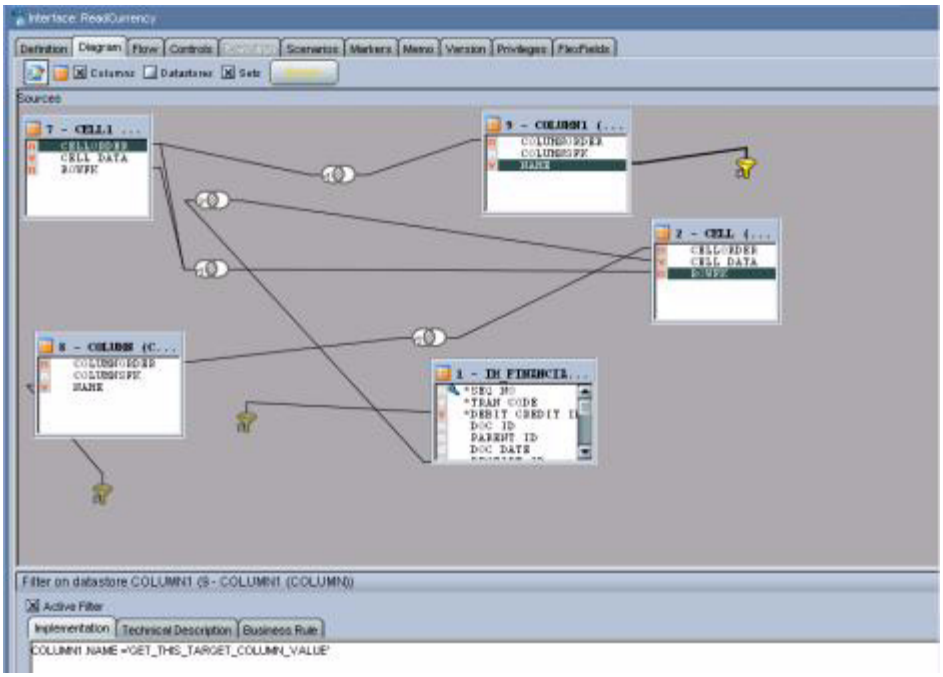
Figure 24–17 Join of CELL1 with the COLUMN1 Table



8. Add a filter on the COLUMN1 table (to determine the target column name), as shown in Figure 24–18.

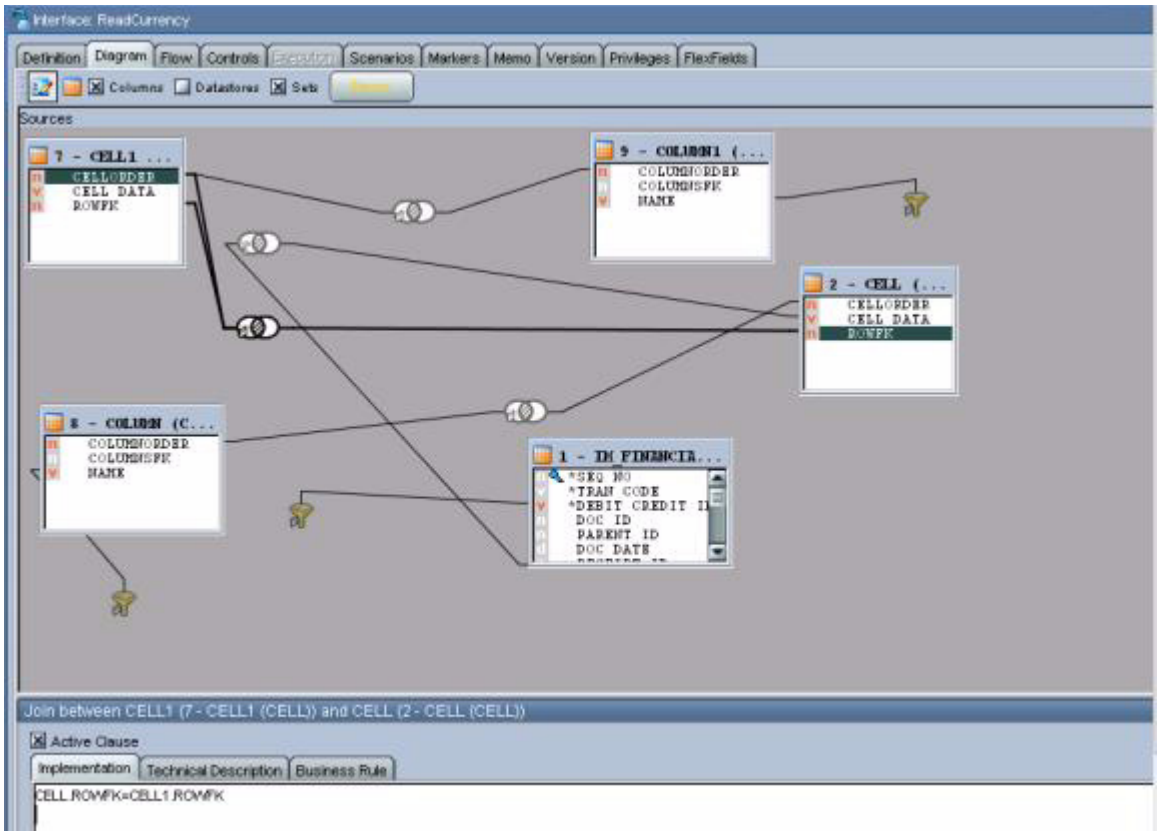
   This filter can use a variable that holds the source column name.

Figure 24–18 Filter Added on the COLUMN1 Table



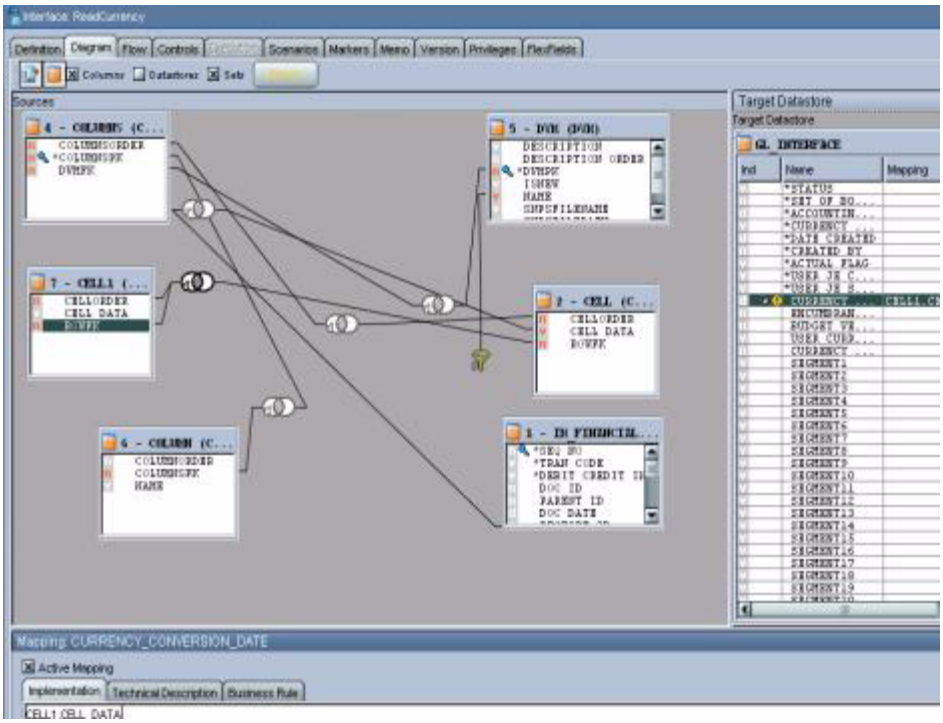
9. Finally, self-join the CELL table with another CELL (CELL1) table, as shown in Figure 24–19.

Figure 24–19 Self-Join of the CELL Table with Another CELL (CELL1) Table



10. Use this second CELL data (CELL1 table's data) in the target interface mapping, as shown in Figure 24–20.

Figure 24–20 Second CELL Data in the Target Interface Mapping



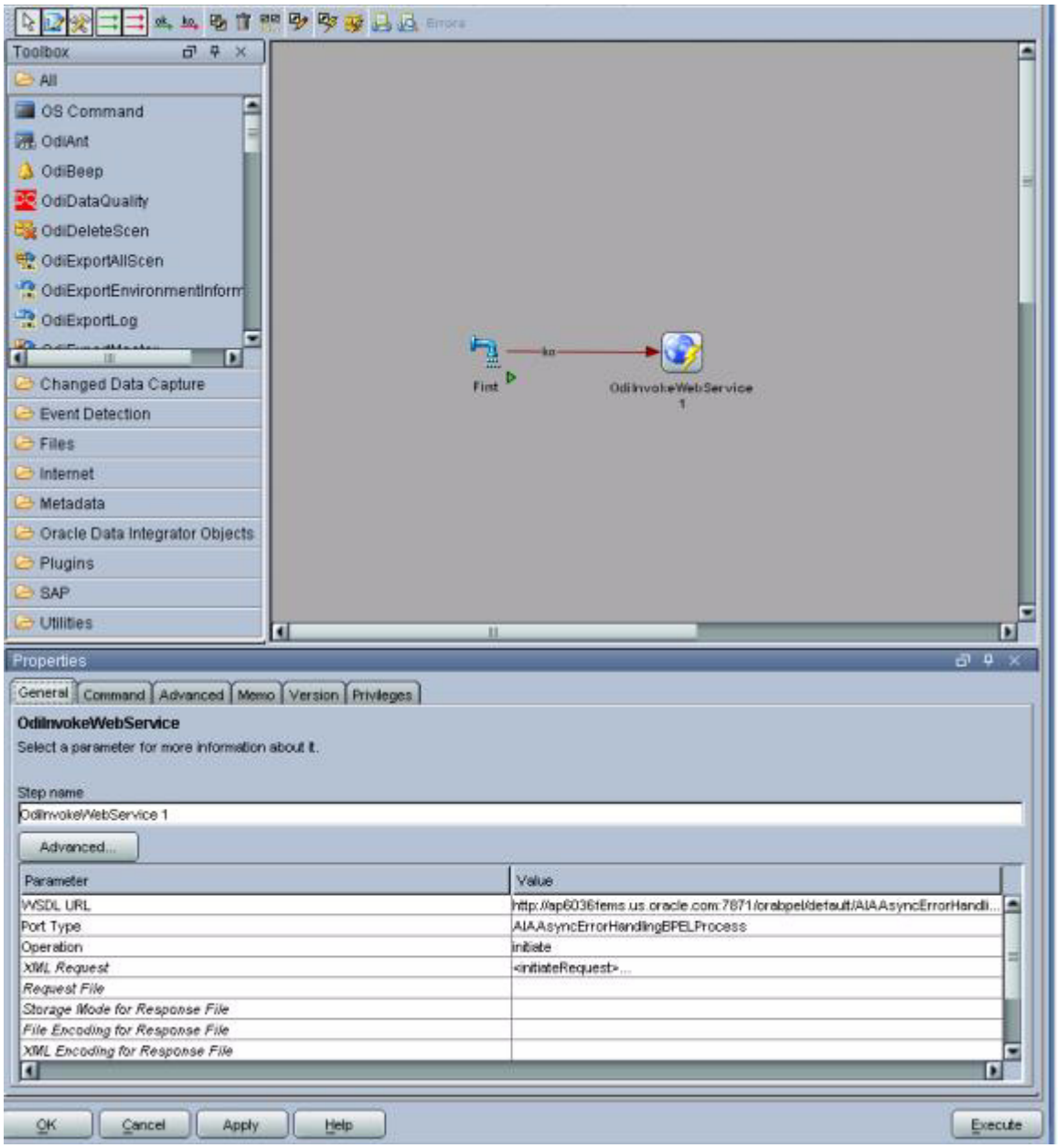
Note: You must repeat all joins for each DVM-centric column.

24.7 Using Error Handling

To use error handling for the Oracle Data Integrator flows:

The package shown in Figure 24–21 is a sample interface that invokes the **AIAAsyncErrorHandlingBPELProcess** when it ends in error.

Figure 24–21 Sample Interface that Invokes AIAAsyncErrorHandlingBPELProcess When it Ends in Error



The XML Request Input for this process is provided in Example 24–2.

Example 24–2 XML Request Input for AIAAsyncErrorHandlingBPELProcess

```
<initiateRequest>
<Fault>
<EBMReference>
<EBMID/>
<EBMName/>
<EBOName/>
<VerbCode/>
<BusinessScopeReference>
<ID/>
<InstanceID>[End to End Process Name, Hard code, every developer must know
```

```
this process name]</InstanceID>
<EnterpriseServiceName/>
<EnterpriseServiceOperationName/>
</BusinessScopeReference>
<SenderReference>
<ID>[Sender Reference ID - Hard code the system id of the source system]</ID>
<SenderMessageID/>
<TransactionCode/>
<ObjectCrossReference>
<SenderObjectIdentification>
<BusinessComponentID/>
<ID/>
<ContextID/>
<ApplicationObjectKey>
<ID/>
<ContextID/>
</ApplicationObjectKey>
<AlternateObjectKey>
<ID/>
<ContextID/>
</AlternateObjectKey>
</SenderObjectIdentification>
<EBOID/>
</ObjectCrossReference>
<Application>
<ID/>
<Version/>
</Application>
</SenderReference>
</EBMReference>
<FaultNotification>
<ReportingDateTime><%=odiRef.getPrevStepLog("BEGIN")%></ReportingDateTime>
<CorrectiveAction>[ODI does not have it, leave this element null]</
CorrectiveAction >
<FaultMessage>
<Code>[ODI Error return code to be passed here. Must find the ODI system
variable to get the return code]</Code>
<Text><%=odiRef.getPrevStepLog("CONTEXT_NAME" )%>/<%=odiRef.getSession( "SESS_
NAME" )%>/<%=odiRef.getPrevStepLog("STEP_
NAME")%>:<![CDATA[<%=odiRef.getPrevStepLog("MESSAGE")%>]]></Text>
<Severity>[ODI does not have it, hard code it to 1]</Severity>
<Stack/>
</FaultMessage>
<FaultingService>
<ID>[Hard code a meaningful name of the Process, For example, CustomerInitialLoad
(this
could be same as package name)] </ID>
<ImplementationCode>ODI</ImplementationCode>
<InstanceID><%=odiRef.getPrevStepLog("SESS_NO")%></InstanceID>
</FaultingService>
</FaultNotification>
</Fault>
</initiateRequest>
```

24.8 Oracle Data Integrator Ref Functions

Table 24–2 lists the Oracle Data Integrator Ref functions with a description of each.

Table 24–2 Oracle Data Integrator Ref Functions

Ref Function	Description
`<%=odiRef.getPrevStepLog("BEGIN")%>`	The date and time when the step that ended in error began
`<%=odiRef.getPrevStepLog("MESSAGE")%>`	The error message returned by the previous step, if any. It is a blank string if no error occurred.
`<%=odiRef.getPrevStepLog("SESS_NO")%>`	The number of the session
`<%=odiRef.getPrevStepLog( "CONTEXT_NAME" )%>`	The name of the context in which the step was performed.
`<%=odiRef.getSession( "SESS_NAME")%>`	The name of the session.
`<%=odiRef.getPrevStepLog( "STEP_NAME" )%>`	The name of the step.

24.9 How to Publish the Package and Data Model as Web Service

To publish the package and data model as a Web service:

1. Install Axis2 as a standalone server.
   a. Set an environment variable JAVA_HOME to the path name of the directory in which you installed the JDK release.
   b. Download Apache Axis2 Version 1.2 (http://axis.apache.org/axis2/java/core/download.cgi) and unpack the Axis2 Standard Binary Distribution into a convenient location so that the distribution resides in its own directory. Set an environment variable AXIS2_HOME to the path name of the extracted directory of Axis2 (for example, /opt/axis2-1.2).
   c. Start the Standalone Axis2 server by running the following command: `$AXIS2_HOME\bin\axis2server.bat` (Windows)

      After startup, the default web services that are included with Axis2 are available by visiting http://localhost:8080/axis2/services/
   d. Download the `axis2.war` from http://axis.apache.org/axis2/java/core/download.cgi.
2. Deploy `axis2.war` on the OC4J Application Server.
   a. Deploy the `axis2.war` on the OC4J Application Server (www.oracle.com/technology/software/products/ias/htdocs/utilsoft.html) or OC4J Server of the SOA Suite can also be used. Go to `http://<Host:port>` and launch the application server that takes you to OC4J home. Click the **Application** tab and then click the **Deploy** tab. It asks you the location of the file. Browse and point to the `Axis2.war` file and then deploy.
   b. After the WAR is successfully deployed, test it by pointing the web browser to `http://<host :port>/axis2`. This opens the **Axis2 Web Application Home Page**.
   c. Click **Validate** to ensure that the procedure ended successfully.
3. Install the Oracle Data Integrator Public Web Services on Axis2.

In Axis2, go to the *Administration* page.

a. Select the **Upload Service** link.

b. Browse for the Oracle Data Integrator Web Services .aar file.

It is located in the `/tools/web_services/` subdirectory in the Oracle Data Integrator installation directory.

c. Click the **Upload** button.

Axis2 uploads the Oracle Data Integrator Web Services. You can now see Data Integrator Public Web Services in the Axis2 services list.

d. The services and the operations of successfully installed services appear on the available services page (`http://<Host>:<HTTP port>/axis2/services/listServices`), where you can see OdiInvoke

4. Environment setup for Data Services

a. The database drivers (http://www.oracle.com/technology/software/tech/java/sqlj-jdbc/htdocs/jdbc9201.html) must be installed in the appropriate directory. ORACLE_HOME/j2ee/home/applib for OC4J.

b. Create the JDBC Datasource pointing to the data server:

- Connect to OC4J administration interface.
- On the Administration tab, in Services | JDBC Resources, click Go to task.
- Click the Create button in the Connection Pools section.
- Select the Axis2 application, select New Connection Pool, then Continue.
- Fill in the fields for the JDBC data source and click Finish.
- Click the Create button in the Data Sources section.
- Select the Axis2 application, select Managed Datasource, then Continue.

c. META-INF/context.xml and WEB-INF/web.xml is updated in the iAxis directories.

Update application.xml in the given folder ORACLE_HOME\j2ee\home\applications\axis2\META-INF as shown in Example 24–3.

Example 24–3 Update application.xml in ORACLE_HOME\j2ee\home\applications\axis2\META-INF

```
<Context >
<Resource
name="jdbc/TestDS"
type="javax.sql.DataSource"
driverClassName="oracle.jdbc.OracleDriver"
url="jdbc:oracle:thin:@abijayku-idc:1522:orcl1"
username="master"
password="master"
maxIdle="2"
maxWait="-1"
maxActive="4"/>
</Context>
(Resource name will be reused in the web.xml file and in the Model in Designer.)
(driverClassName, url, username and password will explicitly point to the
data source.)
```

```
Update the web.xml file with the resource name of the context.xml file
(here res-ref-name) in the given folder ORACLE_
HOME\j2ee\home\applications\axis2\axis2\WEB-INF as follows:
<resource-ref>
<description>Data Integrator Data Services on
Oracle_SRV1</description>
<res-ref-name>jdbc/TestDS</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

d. Set an environment variable ODI_JAVA_HOME to the path name of the directory into which you installed the JDK release.

5. Configure the topology, as shown in Figure 24–22.

a. In Topology Manager's Physical Architecture view, select the Axis2 technology. Right-click and select **Insert Dataserver**. If you are using a different Web Services container, then choose the appropriate technology instead.

b. Complete the following fields on the **Definition** tab:

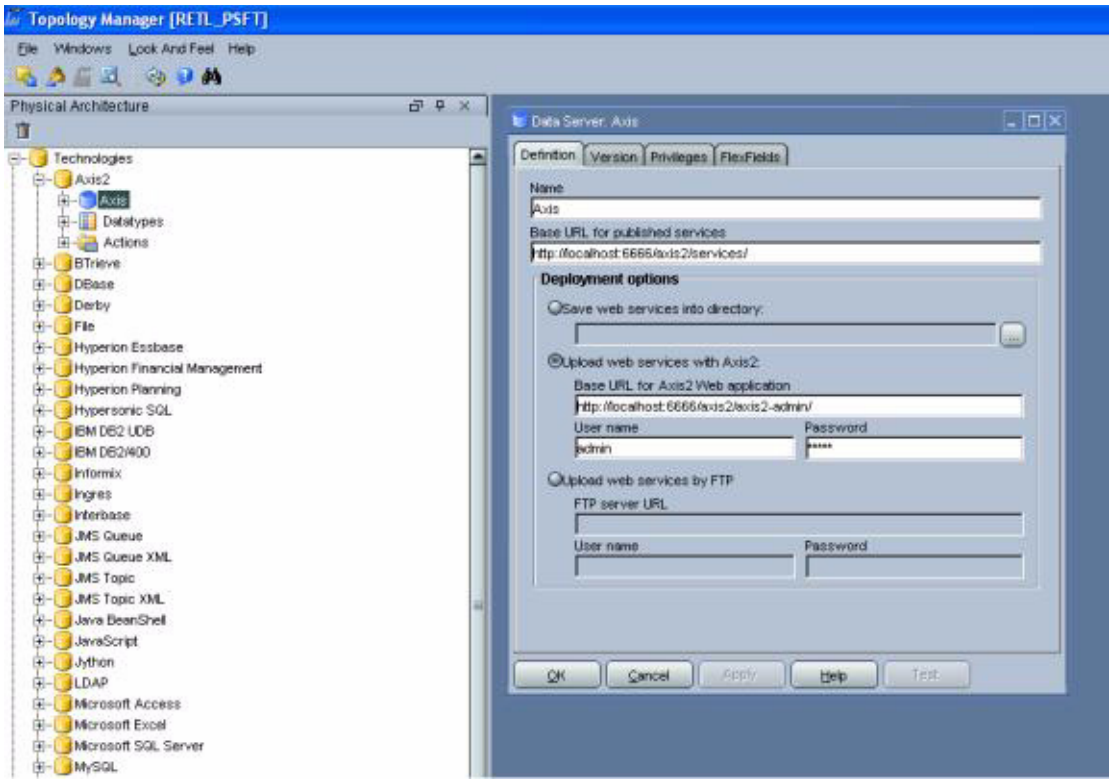
- **Name**: Name of the dataserver as it appears in Oracle Data Integrator.
- **Base URL for published services**: `http://<Host>:<HTTP port>/axis2/services`
- Select the option corresponding to the chosen method of deployment.
- **Web Service Upload**: Specify the root URL for the Axis2 application, typically `http://<Host>:<HTTP port>/axis2/axis2-admin/`, and the user name (`admin`) and password (`axis2`) of the Axis2 administrator.

c. Click **OK**. A window opens to enable you to create a physical schema.

d. Go to the **Context** tab, and define one logical schema for each context in which to deploy the Data Services.

e. Click **OK**.

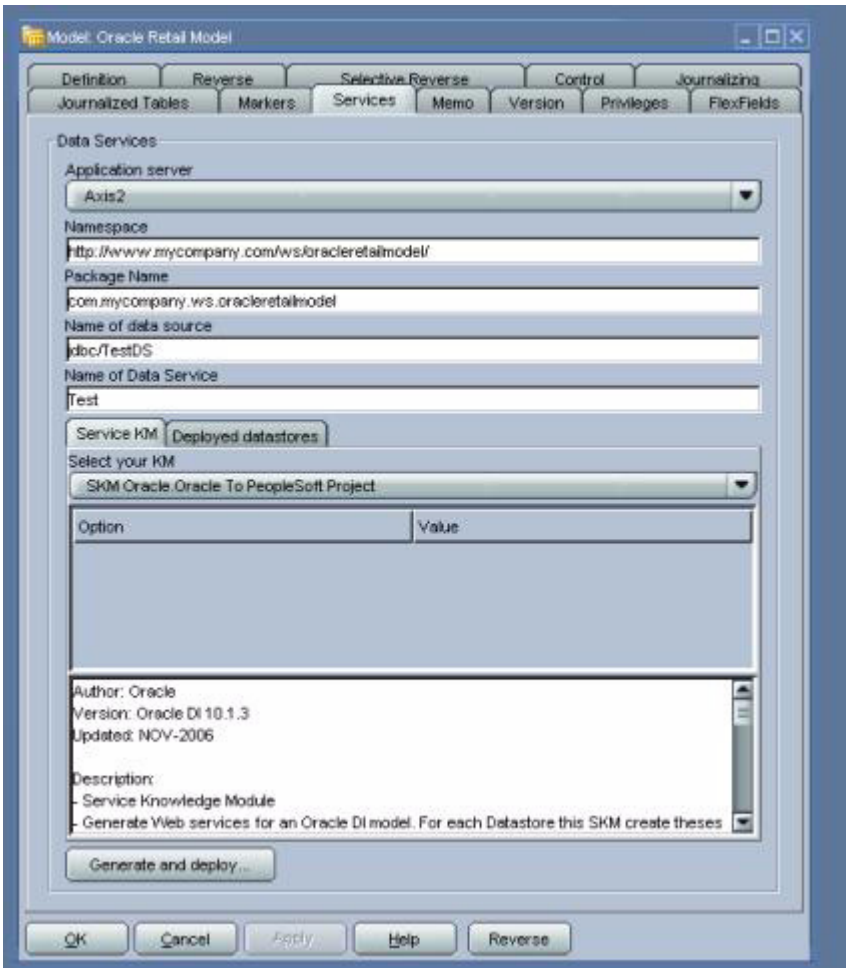
Figure 24–22 Topology Configuration



6. Set up the Data Model to access the data of the table using a web service.
   a. Open the model and go to the **Services** tab as shown in Figure 24–22.
   b. From the **Application Server** list, select the Web Services container that you set up earlier.
   c. Enter the Namespace to be used in the generated WSDL.
   d. Specify the Package name used to name the generated Java package that contains your Web Service. Generally, this is of the form `com.<company name>.<project name>`.
   e. In the **Name of the data source** field, copy and paste the name of the data source that you defined for the server when setting up the data sources. (Provide the JNDI location of the data source.)
   f. Define the Data Service Name.
   g. Select a service knowledge module (SKM Oracle) from the list, and set its options.
   h. Go to the **Deployed Datastores** tab.
   i. Select every data store needed to expose as a Web Service. For each one, specify a Data Service Name and the name of the Published Entity.
   j. Click **OK** to save your changes.
   k. Click the **Generate and Deploy** tab to deploy it on the OC4J server.
   l. The deployed data store can be viewed at `http://<Host>:<HTTP port>/axis2/services/listServices` along with all the available operations.

m. The generated data service can be tested using OdiInvokeWebService tool provided in the package.

Figure 24–23 Services - Data Services Page



7. Run a scenario using a Web Service.

OdiInvoke web service is used to run a scenario using a web service. The WSDL is `http://<Host>:<HTTP port>/axis2/services/OdiInvoke?wsdl`. The port to use is called `invokeScenario`.

This web service commands an agent to connect a given work repository and to start a specific scenario. Parameters are similar to the ones used when running a scenario from an OS command. Example 24–4 is a sample SOAP request for this web service.

Example 24–4 Sample SOAP Request for the OdiInvoke Web Service

```
<invokeScenarioRequest>
<invokeScenarioRequest>
<RepositoryConnection>
<JdbcDriver>oracle.jdbc.driver.OracleDriver</JdbcDriver>
<JdbcUrl>jdbc:oracle:thin:@sbijayku-idc:1522:orcl1</JdbcUrl>
<JdbcUser>MASTER_FINPROJ</JdbcUser>
<JdbcPassword>master</JdbcPassword>
```

```
<OdiUser>SUPERVISOR</OdiUser>
<OdiPassword>SUNOPSIS</OdiPassword>
<WorkRepository>WORK</WorkRepository>
</RepositoryConnection>
<Command>
<ScenName>ABC</ScenName>
<ScenVersion>001</ScenVersion>
<Context>RETL_TO_PSFT</Context>
<LogLevel>5</LogLevel>
<SyncMode>1</SyncMode>
</Command>
<Agent>
<Host>sbijayku-idc</Host>
<Port>20910</Port>
</Agent>
</invokeScenarioRequest>
</invokeScenarioRequest>
```

The scenario execution returns a SOAP response as shown in Example 24–5.

Example 24–5 SOAP Response Returned by the Scenario Execution

```
<odi:invokeScenarioResponse xmlns:odi="xmlns.oracle.com/odi/OdiInvoke">
<odi:CommandResultType>
<odi:Ok>true</odi:Ok>
<odi:SessionNumber>1148001</odi:SessionNumber>
</odi:CommandResultType>
</odi:invokeScenarioResponse>
```

25

Working with Message Transformations

This chapter provides an overview of Transformation Maps and describes how to create transformation maps, work with domain value maps (DVMs) and cross-references, and populate Enterprise Business Message (EBM) headers.

This chapter includes the following sections:

25.1 Introduction to Transformation Maps

Use transformation maps when the document expected by a source or target application varies from the document generated by a source or target application in terms of data shape and semantics. Transformation maps resolve these structural and semantic differences.

25.1.1 Connecting Applications to Implement Business Processes

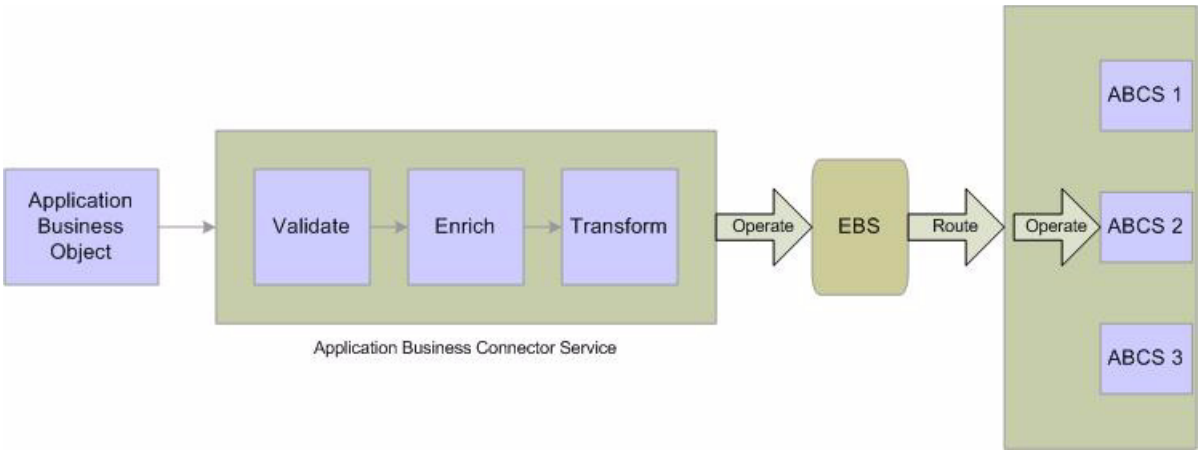
Oracle Application Integration Architecture (AIA) leverages canonical patterns and direct patterns to connect applications for implementing business processes.

25.1.1.1 Canonical Patterns

AIA introduces a set of generic data structures called Enterprise Business Objects (EBOs). An EBO represents a common object definition for business concepts such as *Account*, *Sales Order*, *Item* and so on. The business integration processes work only on messages that are either a complete EBO or a subset of an EBO. This approach allows the cross-industry application processes to be independent of participating applications. EBOs contain components that satisfy the requirements of business objects from the target application data models. Transformations map the application business-specific message to the EBM which is AIA canonical data model.

Figure 25–1 illustrates how a canonical pattern is implemented in AIA.

Figure 25–1 Canonical Pattern Implemented in AIA



25.1.1.2 When to Use Direct Integrations

When data integration involves either a large batch of records, or very large data, you can choose to implement a direct integration specializing on the movement of data with high performance with a trade-off of reusability. Direct integrations can encompass bulk processing and trickle feeds which focus only on implementation decoupling and not data decoupling.

For bulk processing, the ETL tool is used and transformations are done at the data layer using the tool. Although application agnostic objects are not used in trickle feed implementations, the requester application still must transform the content that is produced into the data shape expected by the provider application.

For more information, see Chapter 24, "Using Oracle Data Integrator for Bulk Processing."

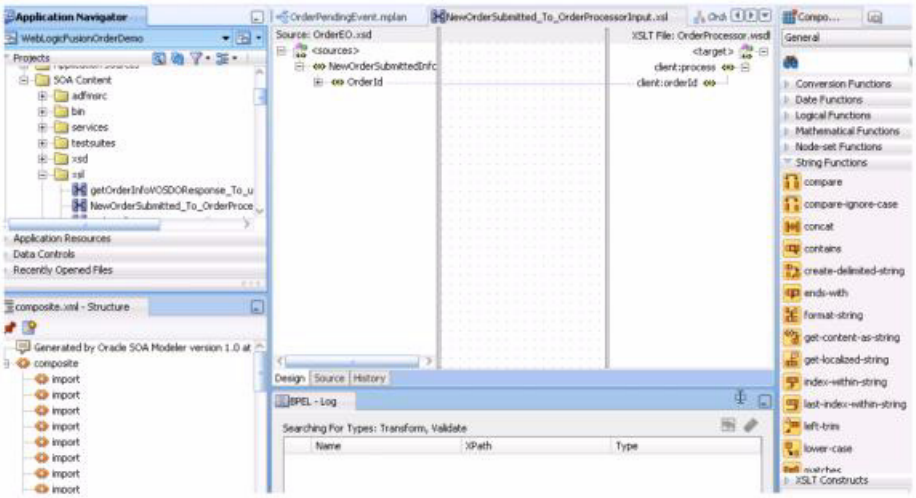
25.1.2 Using Tools and Technologies to Perform Message Transformations

AIA recommends the use of XSLT vocabulary for constructing the transformation maps.

The XSLT Mapper in JDeveloper enables you to create data transformations between source schema elements and target schema elements within the context of services developed using either Oracle BPEL Process Manager or Oracle Mediator.

You use the XSLT Mapper transformation tool to create the contents of a map file. Figure 25–2 shows the layout of the XSLT Mapper.

Figure 25–2 Layout of the XSLT Mapper



For more information about the XSLT Mapper, see "Creating Transformations with the XSLT Mapper" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

25.2 Creating Transformation Maps

One of the steps in configuring the integration objects for your integration process is to map the components and fields in your internal integration object to the message elements in the external integration object. You use these maps to move the data from one integration object to another.

25.2.1 Considerations for Creating Transformation Maps

When creating transformation maps, consider the following guidelines:

- Do not make unnecessary or redundant AIAConfigurationProperties.xml lookup calls.

  Retrieve static configuration properties once in the very beginning of transformation.

  To externalize configuration for deployment time using AIAConfigurationProperties.xml, refer to Section 2.1.3.9, "How to Work with AIAConfigurationProperties.xml in $AIA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/config."

- Ensure that your design does not include infinite loops.
- Develop transformation maps for each component.

  You should build reusable component-specific transformation maps that can be used across applications.

- Create transformation templates for every custom element.
- Keep the transformation maps clutter free.
- Use domain value maps only for static lookups, not for storing configuration or setup data.

Example 25–1 illustrates the domain value map lookup call to fetch *Currency Code*. The domain value maps should be used only for static lookup calls such as *Currency Codes*, *Location Codes*, and so on.

Example 25–1 Domain Value Map Lookup Call to Fetch Currency Code

```
<corecom:PreferredFunctionalCurrencyCode>

<xsl:value-of select="dvm:lookupValue('oramds:/apps/AIAMetaData/dvm/CURRENCY_
CODE.dvm',$SenderSystemId,/xsdLocal:ListOfCmuAccsyncAccountIo/xsdLocal:Account/
xsdLocal:CurrencyCode,$TargetSystemId,'')"/>

</corecom:PreferredFunctionalCurrencyCode>
```

25.2.2 Handling Missing or Empty Elements

Transformation logic and xpath should be designed with the possibility of missing or empty elements, as shown in Example 25–2.

Example 25–2 Sample Code Illustrating Logic Designed to Handle Missing or Empty Elements

```
<xsl:iftest="normalize-
space="/xsdLocal:ListOfCmuAccsyncAccountIo/xsdLocal:Account/xsdLocal:AccountId/
text() )">
<corecom:ApplicationObjectKey>
     <corecom:ID>
           <xsl:value-of
select="/xsdLocal:ListOfCmuAccsyncAccountIo/xsdLocal:Account/xsdLocal:
AccountId"/>
     </corecom:ID>
</corecom:ApplicationObjectKey>
  </xsl:if>
```

In this example, an `if` condition exists to check the empty elements. You should adopt such transformation logic to avoid transporting empty elements across the wire.

25.2.3 How to Map an Optional Source Node to an Optional Target Node

When mapping an optional source node to an optional target node, you should surround the mapping with an `xsl:if` statement that tests for the existence of the source node. If you do not do this, as shown in Example 25–3, and the source node does not exist in the input document, then an empty node is created in the target document.

Example 25–3 Statement Without xsl:If

```
<portal:PHONE>
            <xsl:value-of
select="corecom:Contact/corecom:ContactPhoneCommunication[1]/corecom:
PhoneCommunication/corecom: WorkContactNumber"/>
          </portal:PHONE>
```

If the *PHONE* field is optional in both the source and target and the *WorkContactNumber* does not exist in the source document, then an empty *PHONE* element is created in the target document. To avoid this situation, add an `if` statement to test for the existence of the source node before the target node is created, as shown in Example 25–4.

Example 25–4 Statement With xsl:If

```
<xsl:if
test="corecom:Contact/corecom:ContactPhoneCommunication[1]/corecom:
PhoneCommunication/corecom:WorkContactNumber">
        <portal:PHONE>
            <xsl:value-of
select="corecom:Contact/corecom:ContactPhoneCommunication[1]/corecom:
PhoneCommunication/corecom:WorkContactNumber"/>
         </portal:PHONE>
  </xsl:if>
```

25.2.4 How to Load System IDs Dynamically

ABCS and XSL scripts should not be hard-coded to work against one specific logical application instance, as shown in Example 25–5. No hard-coded system IDs should exist in XSL Scripts and ABCS.

Example 25–5 Sample Code Showing Hard-Coded System IDs - Not Recommended

```
<corecom:PreferredFunctionalCurrencyCode>
             <xsl:value-of
select="dvm:lookupValue('oramds:/apps/AIAMetaData/dvm/CURRENCY_CODES.
dvm','SEBL_01',
/xsdLocal:ListOfCmuAccsyncAccountIo/xsdLocal:Account/xsdLocal:CurrencyCode,
'COMMON','')"/>
</corecom:PreferredFunctionalCurrencyCode>
```

The recommended approach is to load the system IDs dynamically, as shown in Example 25–6.

Example 25–6 Sample Code Showing Use of Variables Instead of Hard-coded System IDs - Recommended

```
<corecom:PreferredFunctionalCurrencyCode>
<xsl:value-of select="dvm:lookupValue('oramds:/apps/AIAMetaData/dvm/CURRENCY_
CODES.dvm',$SenderSystemId,/xsdLocal:ListOfCmuAccsyncAccountIo/xsdLocal:
Account/xsdLocal:CurrencyCode,$TargetSystemId'')"/>
</corecom:PreferredFunctionalCurrencyCode>
```

25.2.5 Using XSLT Transformations on Large Payloads

For BPEL and Oracle Mediator, Oracle recommends that you do not apply the XSLT transformation on large payloads because it results in out-of-memory errors when XSLT must traverse the entire document.

25.2.6 When to Populate the LanguageCode Attribute

Every EBM should populate the `languagecode` element, which specifies the locale in which the request has to be sent. Language sensitive data elements must have the `languageCode` attribute populated if it is different from the one set at EBM root element. The `languageCode` specified in the `DataArea` takes precedence.

25.2.7 How to Name Transformations

Place each transformation type in a separate core XSL file.

The file name should follow this convention:

`<Source Schema Type>_to_<Destination Schema Type>.xsl`

Example: `CreateOrderEBM_to_CreateOrderSiebelABO.xsl`

25.3 Making Transformation Maps Extension Aware

Every component in the EBM, including the EBM header, contains a custom area that can be configured to add the necessary elements. You should develop the core transformation XSL file to provide extension capabilities using the guidelines in the following section.

For more information about the EBM header, see Section 25.6, "Introducing EBM Header Concepts".

25.3.1 How to Make Transformation Maps Extension Aware

To make transformation maps extension aware:

1. Create an extension XSL file for every transformation.

   An example of a core transformation XSL file: `PurchaseOrder_to_PurchaseOrder.xsl`

   Every core XSL file name has one extension XSL file, for example: `PurchaseOrder_to_PurchaseOrder_Custom.xsl`.

2. Define empty named templates in the extension file for every component in the message.
3. Include the extension file in the core transformation file.
4. Add a call-template for each component in core transformation.
5. Enable the transformation to accommodate transformations for custom element as shown in Example 25–7.

Example 25–7 Transformation Enabled to Accommodate Transformations for Custom Element

```
<!-XSL template to transform Address-->
<xsl: template name="Core_AddressTemplate">
<xsl:param name = "context" select ="."/>
<xsl:param name = "contextType" select ="'CORE'"/>
<address1><xsl:value-of select="$context/address1"><address1>
<address2><xsl:value-of select="$context/address1"></address2>
<city><xsl:value-of select="$context/city"></city>
<state><xsl:value-of select="$context/state"></state>
<zip><xsl:value-of select="$context/zip"></zip>

   <xsl:if test="$contextType!='CORE'">
<xsl:call-template name="Vertical_AddressTemplate">
<xsl:with-param name="context" select="$context"/>
<xsl:with-param name="contextType" select="$contextType">
      </xsl:call-template>
</xsl:if>
</xsl: template>
```

25.3.2 How to Make the Transformation Template Industry Extensible

Mapping rules pertaining to customer-specific schema extensions are defined in these xsl extension templates.

To make the transformation template industry extensible:

1. Put the extension templates into a separate transformation file.
2. Import from the main transformation file. Example 25–8 shows the industry extensible transformation template.

Example 25–8 Industry Extensible Transformation Template

```
<xsl: template name="Core_AddressTemplate">
<xsl:param name = "context" select =".">/>
<xsl:param name = "contextType" select ="'CORE'"/>
<address1><xsl:value-of select="$context/address1"><address1>
<address2><xsl:value-of select="$context/address1"></address2>
<city><xsl:value-of select="$context/city"></city>
<state><xsl:value-of select="$context/state"></state>
<zip><xsl:value-of select="$context/zip"></zip>
   <xsl:if test="$contextType!='CORE'">
         <xsl:call-template name="Vertical_AddressTemplate">
<xsl:with-param name="context" select="$context"/>
<xsl:with-param name="contextType" select="$contextType">
</xsl:call-template>
</xsl:if>
</xsl: template>
```

25.4 Working with DVMs and Cross-References

This section discusses DVMs, cross-references, and naming standards, and when to use them.

25.4.1 Introduction to DVMs

Use domain values only for static lookups. Do not use them to store configuration or setup data. A separate facility is provided to store and retrieve parameterized configuration information.

To access the DVM at run time, use the lookup-dvm XSL function found in the JDeveloper XSL Mapper.

DVMs are stored in the MDS repository, which uses the database persistence, and are managed using tools provided by JDeveloper or Foundation Pack.

For more information about naming standards, see Chapter 31, "Oracle AIA Naming Standards for AIA Development."

25.4.2 When to Use DVMs

DVMs can be categorized into three groups:

- 100 percent of the possible values are seeded, and you are not permitted to define any more.

  Since logic is added to these values and they are defined by the participating applications, you cannot add values.

- All or a few samples are seeded based on the seeded values found in the participating applications.

  You can add more to the participating applications and can add more to the DVMs in the middle to accommodate.

  Typically, logic is not added to these values; a match gets the value and passes it on to the receiving application. You must enhance the list to include all possible values that you have defined in your participating applications.

- Customized.

  A naming convention is provided for you to define your own DVM types and values in case you have added a column that depends on it. These types or values are not affected during an upgrade.

When creating cross-reference virtual tables in the cross-reference tables, follow the naming standards described in the following section.

25.4.3 Using Cross-Referencing

One of the core design tasks of ABCS is maintaining cross references for unique identifiers from the loosely coupled applications.

AIA does not support hierarchical cross-references. Child object common keys are unique on their own without being under the parent's context. The child objects have their own entry within the cross-references table with a GUID used for the common ID.

The Cross References API does not support multi-part keys. In cases where the application does not have a single unique key, AIA recommends key concatenation:

```
{Key1}::{Key2}::Key3
```

Most requester ABCSs use the following logic in the transformations:

1. Look up in Xref for an existing common ID corresponding to the application ID.
2. If common ID is not found, then generate a new ID and populate the Xref table with the new ID.
3. Use the found or new common ID in the transformation.

Cross referencing is achieved using EBM Object identification elements and the cross referencing APIs (Populate Xref, Lookup Xref, Delete Xref) provided by Oracle Mediator.

Table 25–1 lists a cross referencing configuration implemented in the Product synchronization scenario between the Portal BRM and Siebel CRM systems

Table 25–1 Cross Referencing Configuration

Entity	Siebel CRM ID	Common ID	Oracle BRM ID
Item ID	Product ID	Auto Generated GUID	Product ID
Price Line ID	Price Line ID	Auto Generated GUID	Product ID

`ITEM_ID` cross-references the BRM (Portal) ProductID and the Siebel ProductID.

`PRICELINE _ID` cross-references the BRM (Portal) Product ID to Siebel PriceLineID.

25.4.4 How to Set Up Cross References

Oracle SOA Suite provides a cross-reference infrastructure that consists of these components:

- A single Xref database table that contains virtual tables for the different cross-reference object types and command line utilities to import/export data.
- XPath functions to query, add, and delete cross-references.

For more information about naming standards, see Chapter 31, "Oracle AIA Naming Standards for AIA Development."

For more information about cross-references, see "Working with Cross References" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

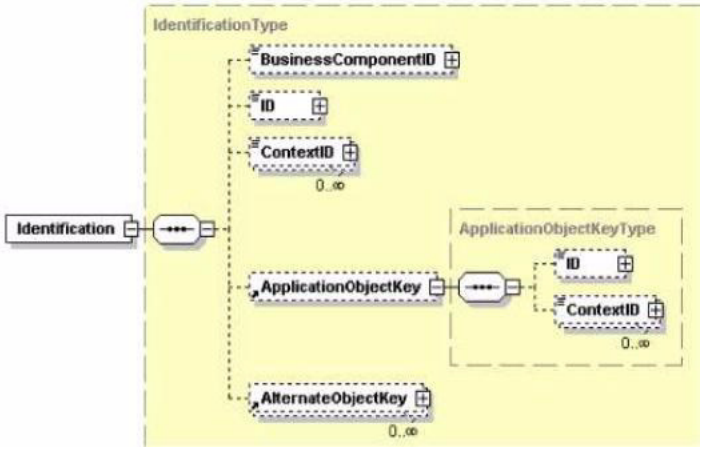
25.5 Mapping and Populating the Identification Type

Each of the EBMs has at least one element to hold an object's instance identifying information. An EBM containing details about multiple objects might have elements dedicated to holding object instance identifying details for each one of them. In the EBM, there is a scheme to uniquely identify the top-level object's instance and the child and the grandchild instances of the business components embedded within the overall object. The identification scheme is the same regardless of whether the identification is for the top-level instance (for example - Order Instance Identification) or for a grandchild instance (for example - Schedule Line Item Identification).

The commonly practice is to adopt the identification theme as defined in the data type **Identification Type** that is made available in the Common Components schema module. Each of the objects can have representations in various applications and at the integration layer. Each of the representations is uniquely identified using identifying keys. For example, Order Number can uniquely identify either an Order ID or an Order instance in a Siebel application; whereas in a PeopleSoft application, the order instance is uniquely identified with the combination of Business Unit and Order ID.

As you can see in the example, the identifying keys of these representations found across various applications and systems are quite different. The Identification Type enables you to capture the identifying keys of these representations for a single object instance. Figure 25–3 illustrates the schema definition for Identification Type.

Figure 25–3 Structure of the Identification Type Element



The Identification Type structure enables you to capture three specific identifiers plus one more general identifier for a single object. Table 25–2 describes each of the identifiers:

Table 25–2 Identifiers in the Identification Type Structure

Name	Purpose	Details
BusinessComponentID	Unique Key for the application-agnostic representation of the object instance	Business documents generated by Oracle AIA applications have the BusinessComponentID populated. The BusinessComponentID is generated using the API provided by Oracle AIA infrastructure.
ID	Business friendly identifier found in the participating application for this object instance	Business documents generated by Oracle AIA applications have these populated wherever they are applicable. *PO Number* and *Order Number* are examples.
ContextID	Optional element to identify additional qualifiers for the ID. Used for multi-part keys - for example if an Item is unique within a set, then the Item Number would be the ID and the SetID value would be a ContextID value	Business documents generated by Oracle AIA applications have these populated wherever they are applicable. *SetID* within a set is an example
ApplicationObjectKey	Participating application specific internally generated unique identifier for this object instance	Business documents generated by Oracle AIA applications populate this information. This represents the primary key of the object at the participating application.
AlternateObjectKey	One or more ways of additionally identifying the object's instance.	**Optional** Use this element to capture additional identifying details if necessary.

To define the context in which an identifier is valid, a set of attributes that describes the context of the key is supported in addition to the actual key value.

Table 25–3 describes the purpose of each of the attributes.

Table 25–3 Key Context Attributes

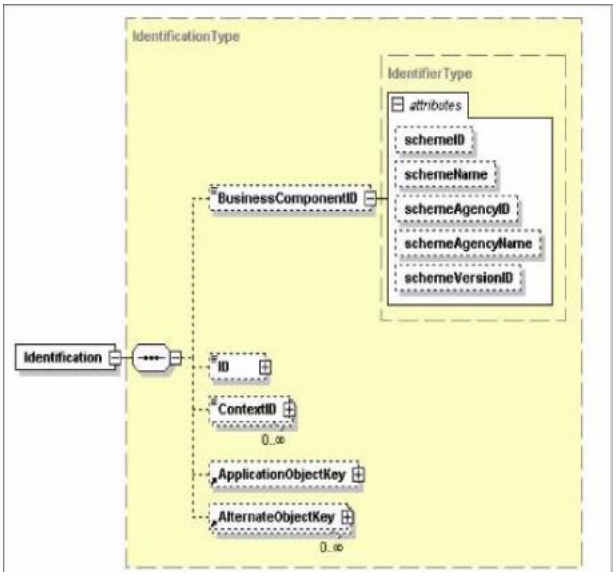
Name	Type	Description
BusinessComponentID/ID ApplicationObjectKey/ID AlternateObjectKey/ID	Element	The element is used to store the actual object ID.

Table 25–3 (Cont.) Key Context Attributes

Name	Type	Description
schemeID	Optional attribute	Identification scheme of the identifier. Attribute schemeID provides information about the object type identified by the ID, for example ItemGUID for the GUID of an item and PartyGUID for the GUID of a party. For the BusinessComponentID, the schemeID is set to the name of the object followed by ' GUID. For the ID, the schemeID is set to the name of the element as known in the participating application.
scheme VersionID	Optional attribute	Version of the identification scheme.
schemeAgencyID	Optional attribute	ID of the agency that manages the identification scheme of the identifier. The GUIDs generated by Oracle AIA have AIA 01 populated in this attribute. For identifiers generated by participating applications, the short code for that of the participating application is stored in this attribute.

Figure 25–4 illustrates the Business Component ID:

Figure 25–4 Structure of the Business Component ID



Example 25–9 provides an example of how the BusinessComponentID is populated in the EBM.

Example 25–9 BusinessComponentID Populated by the EBM

```
<telcoitem:Identification>
<corecom:BusinessComponentID
mlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V1"
schemeID="ITEM_ID_COMMON" schemeAgencyID="AIA_
20">313038383735393433303139373935 31
</corecom:BusinessComponentID>
<corecom:ID
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V1"
schemeID="ItemNumber" schemeAgencyID="PORTAL">0.0.0.1 /product
349330 1</corecom:ID>
<corecom:ApplicationObjectKey xmlns:corecom="http://xmlns.oracle.com/Enterpriser
Objects/Core/Common/V1">
<corecom:ID schemeID="PortalPoid" schemeAgencyID="PORTAL">0.0.0.1 /product
349330 1</corecom:ID>
 </corecom:ApplicationObjectKey>
</telcoitem:Identification>
```

25.5.1 How to Populate Values for corecom:Identification

Follow these guidelines for corecom:Identification or any of its derivations.

- SchemeAgencyID for Common is `COMMON`.
- SchemeAgencyID for ID/ContextID/ApplicationObjectKey is the same as exsl:node-set ($senderNodeVariable)/ID, which is also used as Column Name for lookupXref and lookupDVM.
- Scheme ID for BusinessComponentID is the same as XrefTableName in lookupXref/populateXref.

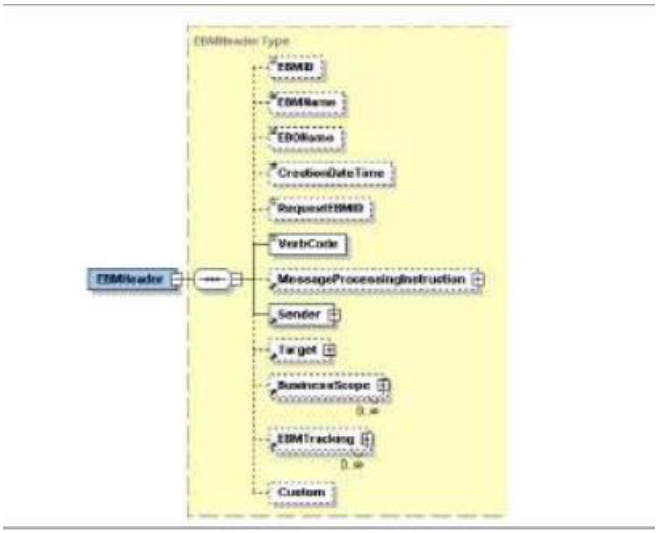
25.6 Introducing EBM Header Concepts

Every EBM that is processed by ABCS and Enterprise Business Service (EBS) contains an EBM header in addition to object-specific information. The objective of the EBM header is to carry information that can be used to:

- Track important information
- Audit
- Indicate source and target systems
- Handle errors and traces

Figure 25–5 illustrates the components of an EBM header:

Figure 25–5 EBM Header Components



The following sections provide details about each of the components.

25.6.1 Standard Elements

This section discusses the standard elements in a message header.

25.6.1.1 EBMID

This element contains a GUID to uniquely identify the EBM, as shown in Example 25–10. The GUID is generated using a provided standard XPath function.

XPath function to generate GUID: `orcl:generate-guid()`

Example 25–10 EBMID

```
<EBMHeader>
<EBMID>33303331343032313534393138393830</EBMID>
. . .
</EBMHeader>
```

25.6.1.2 EBOName

This element contains the fully qualified name of the EBO in the notation: `{namespaceURI}localpart`, as shown in Example 25–11.

Example 25–11 EBOName

```
<EBMHeader>
<EBOName>{http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V1}Query
Invoice</EBOName>
. . .
</EBMHeader>
```

25.6.1.3 RequestEBMID

This element contains the originating request GUID that uniquely identifies the originating request ID, as shown in Example 25–12. The EBS populates this field in the response message by extracting it from the request message.

Example 25–12 RequestEBMID

```
<EBMHeader>
<RequestEBMID>33303331343032313534393138393830</RequestEBMID>
. . .
</EBMHeader>
```

25.6.1.4 CreationDateTime

This element contains a timestamp that reflects when the EBM was created, as shown in Example 25–13. The timestamp should be presented in UTC, which can be presented in current datetime and GMT offset as:

```
yyyy '-' mm '-' dd 'T' hh ':' mm ':' ss ('.' s +)? (zzzzzz)?
XPath function to generate CreationDateTime:
xp20:current-dateTime()
```

Example 25–13 CreationDateTime

```
<EBMHeader>
<CreationDateTime>200 7-04-2 7T12:22:17-08:00</CreationDateTime>
. . .
</EBMHeader>
```

25.6.1.5 VerbCode

This element contains the verb represented by the EBM, as shown in Example 25–14.

Example 25–14 VerbCode

```
<EBMHeader>
<VerbCode >Query</VerbCode>
. . .
</EBMHeader>
```

25.6.1.6 MessageProcessingInstruction

This element contains instructions on how the message should be processed. This section indicates if the EBM is a production system-level message that should go through its standard path, or if it is a testing-level message that should go through the testing or simulation path.

MessageProcessingInstruction contains two fields:

- EnvironmentCode:

  Can be set to either `CAVS` or `PRODUCTION`. The default is `PRODUCTION`. Setting the value to `PRODUCTION` indicates that the request must be sent to a concrete endpoint. Setting the value to `CAVS` indicates that the request must be sent to CAVS Simulator.

- DefinitionID:

  Specifies the test definition ID.

  It should be populated with the value of the property "Routing.[PartnerlinkName].CAVS.EndpointURI" from `AIAConfigurationProperties.xml` when the

"Routing.[PartnerlinkName].RouteToCAVS" property is set to `true` in `AIAConfigurationProperties.xml`.

25.6.1.7 When to Populate Standard Header Fields

The standard header elements should be populated whenever a message is created. This occurs when:

- Transforming an application request ABM into an EBM in a requester ABC implementation service.
- Transforming an application response ABM into an EBM in a provider ABC implementation service.

25.6.1.8 How to Populate the Message Processing Instruction

To populate the message processing instruction

1. Add instructions to indicate how a message is to be processed.

   This instruction is normally used to tell the system to run the message in production mode or in simulation/testing mode, as shown in Example 25–15.

2. These fields are currently populated by the CAVS.

   Usually, the fields are populated in the SOAP Header before initiating a testing request.

Example 25–15 Message Processing Instruction Population

```
<corecom:MessageProcessingInstruction>
    <corecom:EnvironmentCode>
 <xsl:text disable-output-escaping="no">Production</xsl:text>
    </corecom:EnvironmentCode >
</corecom:MessageProcessingInstruction>
```

25.6.2 Sender

The Sender contains information about the originating application system, as shown in Figure 25–6.

Figure 25–6 Structure of the Sender Element

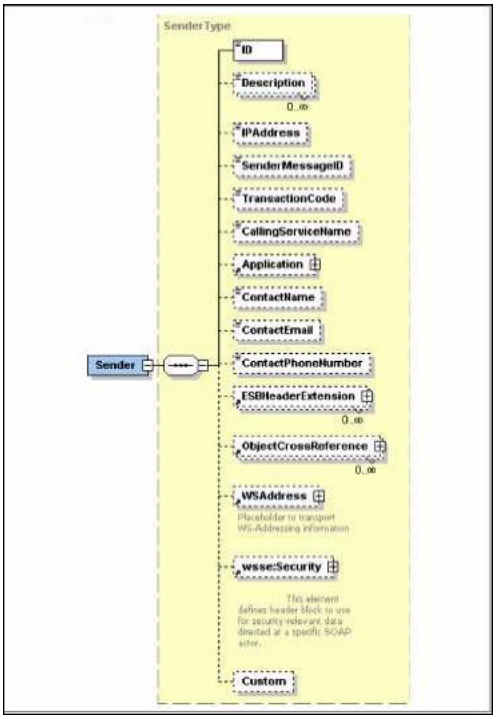


Table 25–4 provides details about each element in the Sender element.

Table 25–4 Elements in the Sender Element

Element	Description
ID	Contains the sender system identifier code. This is a mandatory element. This element represents the unique identifier of each source system. ReqABCSImpl should call the API detailed in Example 25–16 to populate this value.
Description	This is the long description of the Sender System. ReqABCSImpl should call the API detailed in Example 25–16 to populate this value.
IPAddress	Represents the IP address of the sender system. ReqABCSImpl can call the API detailed below to populate this value.
SenderMessageID	This is the ID that can uniquely identify the message sent by the sender system. ReqABCSImpl may have this information and that information can be used to populate this element.
Transaction Code	This is the task code in the sender system that is used to generate this message. ReqABCSImpl has this information and should use that while preparing the EBM.
CallingServiceName	Name for the calling ABCS. Populated by source ABCS.
Application	Holds information about the originating application.

Table 25–4 (Cont.) Elements in the Sender Element

Element	Description
Application/ID	The sender system can designate the originating application code in this element. ReqABCSImpl should call the API detailed in Example 25–16 to populate this value.
Application/Version	The sender system can designate the version of the originating application in this element. ReqABCSImpl should call the API detailed in Example 25–16 to populate this value.

Example 25–16 Sender Element Code Sample

```
<corecom:Sender>
<corecom:ID>SEBL_01</corecom:ID>
<corecom:Description/>
<corecom:IPAddress/>
<corecom:SenderMessageID>33303331343032313534393138393830</corecom:
SenderMessageID>
<corecom:CallingServiceName>{http://xmlns.oracle.com/SampleSiebelApp}
CreateCustomerSiebelInputFileReader</corecom:CallingServiceName>
<corecom:Application>
<corecom:ID/>
<corecom:Version>1.0</corecom:Version>
</corecom:Application>
<corecom:ContactName/>
<corecom:ContactEmail/>
<corecom:ContactPhoneNumber/>
</corecom:Sender>
```

25.6.2.1 SenderMessageID

This element uniquely identifies the message that originated in the source application. Inbound request message is one of the potential sources for this element. The Application Business Message (ABM) (payload sent by the source application) can either have SenderMessageID populated in the payload or stored in ABM Header. Populated during the inbound message transformation from ABM into EBM in the ReqABCSImpl.

This information could subsequently be used by downstream services that are planning to send a message to the same sender application to establish the context.

25.6.2.2 When to Populate Sender System Information

The Sender System information should be populated whenever an EBM is created. This occurs when:

- Transforming a request ABM into an EBM in a requester ABC implementation service.
- Transforming a response ABM into an EBM in a provider ABC implementation service.

25.6.2.3 How to Populate Sender System Information

To populate sender system information:

1. Use the following XPath function to retrieve the Sender System information from AIA System Registration Repository based on the system ID/Code and then return the data as an XML Node-Set:

```
ebh:getEBMHeaderSenderSystemNode(senderSystemCode as string,
senderSystemID as string,
) return senderSystem as node-set
```

2. Pass either the senderSystemCode or the senderSystemID.

   The function locates the system information in AIA System Registration Repository based on either the code or the ID.

25.6.2.4 TransactionCode

The TransactionCode is used to identify the operation in the sender system that generated the sender message.

The inbound request message is one of the potential sources for this element.

To preserve the context, the value of this element is used when constructing the 'AIAFaultMessage' in the *catch* of a fault handler.

This information could subsequently be used by downstream services that are planning to send a message to the same sender application to establish the context.

25.6.2.5 ContactName

This element is the name of the contact of the sender system. The value is retrieved from AIA System Registration Repository by doing a lookup. ReqABCSImpl should call the API detailed in Example 25–16 to populate this value. This information can be made available in the fault message or in the request message to be sent to external businesses by the downstream services.

25.6.2.6 ContactEmail

This element is the email of the contact of the sender system. The value is retrieved from AIA System Registration Repository by doing a lookup. ReqABCSImpl should call the API to populate this value. This information can be made available in the fault message or in the request message to be sent to external businesses by the downstream services.

25.6.2.7 ContactPhoneNumber

This element is the phone number of the contact of the sender system. The value is retrieved from AIA System Registration Repository by doing a lookup. ReqABCSImpl should call the API to populate this value. This information can be made available in the fault message or in the request message to be sent to external businesses by the downstream services.

25.6.2.8 ESBHeaderExtension

This element accommodates the transmission of additional information from the source application. Some data passed from the sender system must be transported through the EBM message life cycle and is needed for identifying and processing the target system. The target Application Business Message (ABM) may not have a placeholder for those kinds of data. Since they cannot be forced to provide a

placeholder for every such element, this Enterprise Service Bus (ESB) header is used to hold that information. ESB has name and value pair and this component can have as many of those elements as required.

Figure 25–7 Structure of the ESBHeaderExtension Element

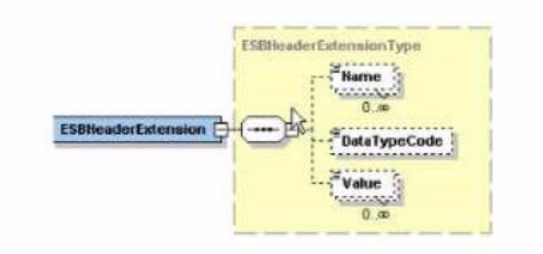


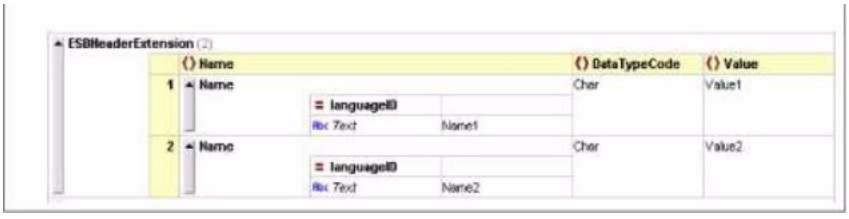
Table 25–5 describes the elements in the ESBHeaderExtension element.

Table 25–5 Elements in the ESBHeaderExtension Element

Element	Description
ESBHeaderExtension/Name	ESB Header element name is the same as the ID. Even though it allows multiple names for EBM header scenarios there is only one value that is same as the ID. Any transformation in the life cycle of the EBM header can populate this field.
ESBHeaderExtension/DataTypeCode	ESB Header element data type is populated by source ABCS.
ESBHeaderExtension/Value	ESB Header element value is populated by source ABCS. Even though it allows different placeholders for different data types, for simplicity only this element is populated. Any transformation in the life cycle of the EBM header can populate this field.

Figure 25–8 illustrates the structure of the ESBHeaderExtension element.

Figure 25–8 Structure of the ESBHeaderExtension Element



25.6.2.9 ObjectCrossReference

This component stores the identifier information of the sender objects and the corresponding cross-reference identifier, as shown in Figure 25–9. Since the EBM can be used for bulk processing this element can be repeated as well. This data may be repeated in the data area but to maintain uniform XPath location information about them, they are maintained here as well. ReqABCSImpl populates this value.

Figure 25–9 Structure of the ObjectCrossReference Element

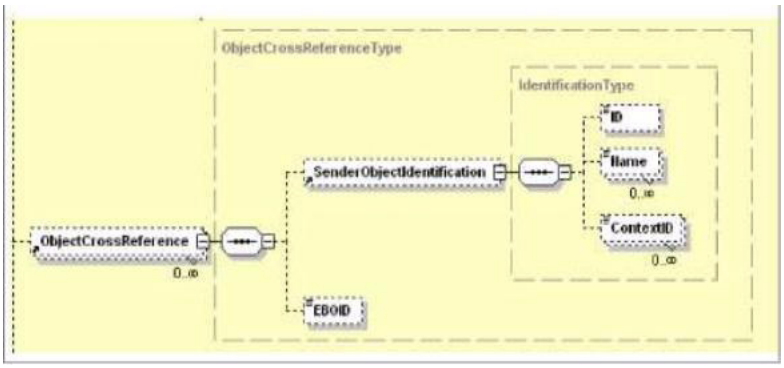


Table 25–6 describes the elements in the ObjectCrossReference element.

Table 25–6 Elements in the ObjectCrossReference Element

Element	Description
ObjectCrossReference/SenderObjectIdentification	Contains all the key field names and key field values for the object. The data is provided by the sender system. ReqABCSImpl has this information and populates this value.
ObjectCrossReference/SenderObjectIdentification/ID	Identifies the object type of the sender system for example ORDER ReqABCSImpl has this information and populates this value
ObjectCrossReference/SenderObjectIdentification/Name	This identifies the description of the sender system, for example, *Purchase Order*. ReqABCSImpl has this information and populates this value.
ObjectCrossReference/SenderObjectIdentification /ContextID	This identifies the sender system's object identifiers. If the sender's object has multiple values then repeat this as many times. Use the attribute schemeID to set the Key Name and the Key value should be stored in the element itself. ReqABCSImpl has this information and populates this value.
ObjectCrossReference/EBOID	This is the corresponding cross-reference identifier stored in the integration layer. ReqABCSImpl has this information and populates this value.

25.6.2.10 How to Add Object Cross Reference information in the Sender System Information

To add a cross-reference entry in the sender system's ObjectCrossReference:

You must add the identifier information of the sender objects and the corresponding cross-reference identifier.

- EBOID - Provide the corresponding cross-reference identifier in the integration layer. `ReqABCSImpl` has this information and populates this value.
- SenderObjectIdentification - Populate the key field name and key field values for the object. The data is provided by the Sender System.

- ID - Identifies the object type of the sender system, for example, Order. `ReqABCSImpl` has this information and populates this value
- NAME - Identifies the description of the sender system, for example, Purchase Order. `ReqABCSImpl` has this information and populates this value.
- ContextID - Identifies the sender system's object identifiers. If the sender's object has multiple values then repeat this as many times. Use the attribute `SchemeID` to set the key name and store the key value in the element itself. `ReqABCSImpl` has this value and populates the value.

25.6.2.11 WS Address

This element holds all WS-Addressing elements and transports WS-Addressing elements and exchanges WS-Addressing elements between the requester and target system. A local version of WS-Address schema is stored in a specified directory and ReqABCSImpl populates this. This element must be populated in the request EBM only when the provider application sends the response in an asynchronous mode. The provider application sending an asynchronous response leverages the value of this element to identify the callback address.

25.6.2.12 Custom

This element is the complex type that you can extend to add your own elements.

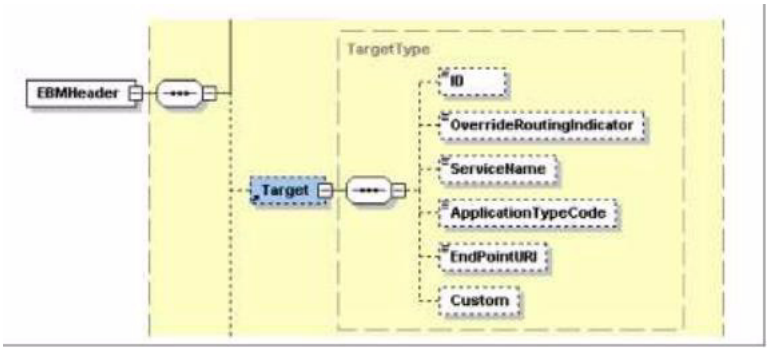
For more information about specific usage of this element, see Section 13.6, "Implementing the Fire-and-Forget Message Exchange Pattern."

25.6.3 Target

The Target section is populated only when there is a need to override the routing rules for the target system defined in Oracle Mediator. For bulk processing it is assumed that all objects reach the same target system defined in a routing rule. In that scenario define the appropriate target system information in this element to override the routing rule. The overriding target system information is applicable to all the objects in the data area. The requester ABCS should never populate the target system. The Enterprise Business Flow (EBF) or an EBS alone can populate the details.

The Target element contains information regarding the target or receiving system and has the elements shown in Figure 25–10.

Figure 25–10 Structure of the Target Element



25.6.3.1 ID

Use this element, shown in Example 25–17, to identify target systems to route to when the routing rules are overridden. It is populated by the EBS when the target system must be overridden with the value of the participating application instance code.

25.6.3.2 ApplicationTypeCode

This element identifies the type of application. An identifier for the application type where multiple instances of the same application type may be registered in the integration platform. The application type may contain the version number of the application if multiple versions are supported on the system.

This field, as seen in Example 25–17, should be populated at the same time as the Target/ID field is populated in the EBS (or in an EBF), usually in the ABCS. The value of this field should come from the function `aia:getSystemType(<ID>)`, where ID is the system ID value that is populated in Target/ID. EBS routing rules almost always check this field for a value or lack of value when determining the routing target.

25.6.3.3 Custom

This is the complex type that you can extend to add your own elements when needed.

Example 25–17 shows an example of target element code.

Example 25–17 Target Element Code Sample

```
<corecom:Target>
<corecom:ID>PORTAL_01</corecom:ID>
<corecom:ApplicationTypeCode>PORTAL_01</corecom:ApplicationTypeCode>
</corecom:Target>
```

25.6.4 BusinessScope

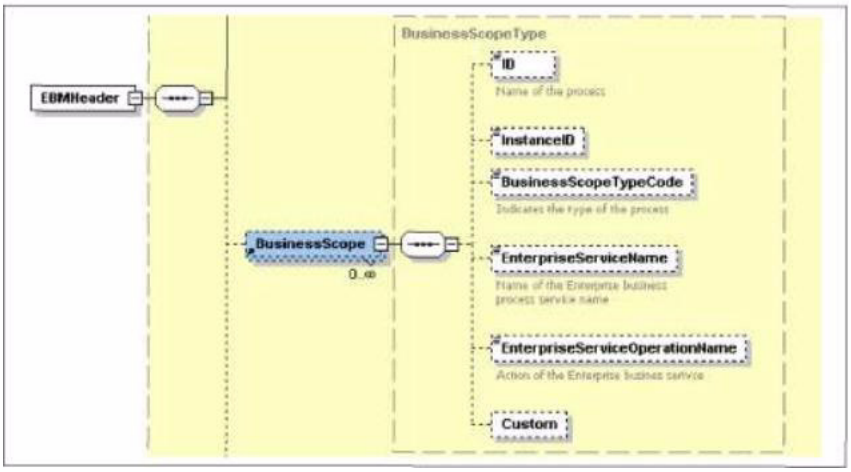
This section captures business scope specification related information defined in UN/CEFACT Standard business definition header.

Every EBM must contain at least two rows for these elements.

- One row with type **Business Scope** describes the end-to-end business process that the message is part of.
- The second row describes the main message associated in the flow (for example, order message in ProcessOrder flow).

For most of the cases, each end-to-end process has one message only. However in complex business scenarios there could be multiple messages spawned or forked from the process. In that case each spawned message must be a row in this section. Figure 25–11 describes how this works.

Figure 25–11 Structure of the BusinessScope Element



25.6.4.1 ID

An optional identifier that identifies the contract this instance relates to. ReqABCSImpl populates this value. This is the name of the process or message given by the applications.

25.6.4.2 InstanceID

A unique identifier that references the instance of the scope (for example, process execution instance of document instance). This is an alpha numeric code assigned by the application team concatenated with a GUID. For message type business scope section use the same EBMID as used in the top section.

25.6.4.3 BusinessScopeTypeCode

This element indicates the kind of business scope. Values are:

- `BusinessScope` (UMM)
- `BusinessService` (for ebXML)
- `Message`

ReqABCSImpl populates this value.

25.6.4.4 EnterpriseServiceName

Name of the EBS where this message belongs. Known to the message creator, be it ABCS or EBS. ReqABCSImpl populates this value.

25.6.4.5 EnterpriseServiceOperationName

Name of the action of the EBS this message belongs. Known to the message creator, be it ABCS or EBS. ReqABCSImpl populates this value.

25.6.4.6 Custom

A complex type for you to extend to add extra elements.

25.6.4.7 How to Add Business Process Information

Every EBM must contain at least two rows for these elements:

- One row with type **Business Process** describes the end-to-end business process the message is part of.
- The second row describes the main message associated in the flow, for example, the order message in ProcessOrder flow.

In most cases, each end-to-end process has only one message. However in complex business scenarios there could be multiple messages spawned or forked from the process. In that case each spawned message must be a row in this section.

The use case example below describes how this works. To keep things simple the example limits the messages to the immediate child of the process and the subsequent chaining of messages is not taken into account.

25.6.5 Use Case: Request-Response

GetAccountBalance: Sends a request message with account number as input and receives account balance as response, as shown in Figure 25–12.

Figure 25–12 Use Case: Request-Response

25.6.5.1 Request EBM

Example 25–18 shows two rows: one for the Process and one for the Request EBM Message involved in the process.

Example 25–18 Request EBM for Request-Response Use Case

```
<BusinessScope>
<ID>Siebel- Portal-Get -Account-Balance</ID>
<InstanceID>GETACCTBAL/1001</InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>GetAccountBalance</
EnterpriseServiceOperationName>
  <BusinessScope>
  <BusinessScope>
<ID> AccountBalanceReqMessage </ID>
<InstanceID> ACCTBALMSG/9001[the EBMID in the top element to be used here]
</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>GetAccountBalance
</EnterpriseServiceOperationName>
</BusinessScope>
```

25.6.5.2 Response EBM

Example 25–19 shows two rows: one for the process and one for the Response EBM Message involved in the process.

Example 25–19 Response EBM for Request-Response Use Case

```
<BusinessScope>
<ID>Siebel- Portal-Get -Account-Balance</ID>
```

```
<InstanceID>GETACCTBAL/10 01</InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>GetAccountBalance
</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID>] AccountBalanceResMessage </ID>
<InstanceID> ACCTBALMSG/9002[the EBMID in the top element to be used here
</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>GetAccountBalance</
EnterpriseServiceOperationName>
</BusinessScope>
```

25.6.6 Use Case: Asynchronous Process

SyncProduct: Portal sends an async message to Siebel to sync the product details, as shown in Figure 25–13.

Figure 25–13 Use Case: Asynchronous Process

25.6.6.1 Request EBM

Example 25–20 shows two rows: one for the process and one for the Request EBM Message involved in the process.

Example 25–20 Request EBM for Asynchronous Use Case

```
<BusinessScope>
<ID>Portal-Siebel-Product-Sync</ID>
<InstanceID>PRODSYNC/1003</InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>ProductEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>SyncProduct</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID>  ProductSyncReqMessage </ID>
<InstanceID> PRODSYNCREQ/9003 </InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>ProductEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>SyncProduct</EnterpriseServiceOperationName>
</BusinessScope
```

25.6.7 Use Case: Synchronous Process with Spawning Child Processes

ProcessOrder: Siebel sends an order. It first spawns CreateAccount in the portal, and then it processes the order in the portal, as shown in Figure 25–14.

Figure 25–14 Synchronous Process with Spawning Child Processes

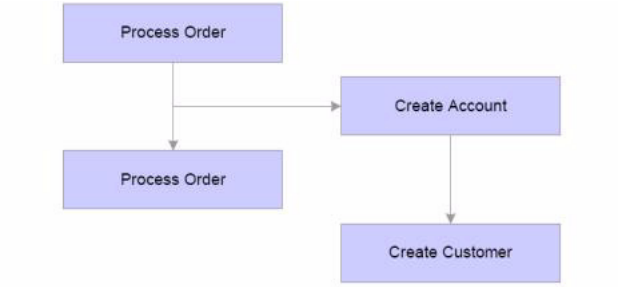
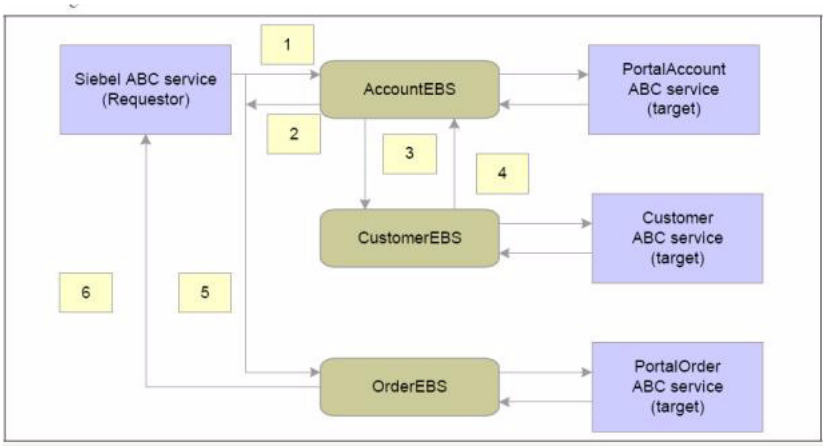


Figure 25–15 illustrates the ProcessOrder flow.

Figure 25–15 ProcessOrder flow



Message 1: Create Account Request EBM

Example 25–21 shows four rows:

- One for the process
- One for the Create Account Request Message in the process
- One for the create customer request message in the process (spawned immediate child)
- One for the create customer response message in the process (spawned immediate child)

Example 25–21 Create Account Request EBM

```
<BusinessScope>
<ID> Portal-Account-Creation </ID>
<InstanceID> CREATEACCT/1008 </InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID> Create-Account-Request-Message </ID>
<InstanceID> CREATEACCTREQMSG/9008 </InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOperationName>
```

```
</BusinessScope>
<BusinessScope>
<ID> Create-Customer-Request-Message </ID>
<InstanceID> CREATECUSTREQMSG/9009 </InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID> Create-Customer-Response-Message </ID>
<InstanceID> CREATECUSTRESPMSG/9021 </InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOperationName>
</BusinessInstruction>
```

Message 2: Create Account Response EBM

Example 25–22 shows two rows:

- One for the process
- One for the Create Account Response Message in the process

Example 25–22 Create Account Response EBM

```
<BusinessScope>
<ID> Portal-Account-Creation-Response </ID>
<InstanceID> CREATEACCTRESP/1008 </InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID> Create-Account-Response-Message </ID>
<InstanceID> CREATEACCTRESPMSG/9020 </InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOperationName>
</BusinessScope>
```

Message 3: Create Customer Request EBM

Example 25–23 shows two rows:

- One for the process
- One for the Create Customer request message in the process

Example 25–23 Create Customer Request EBM

```
<BusinessScope>
<ID> Oracle-Customer-Create </ID>
<InstanceID> CREATECUST/1009 </InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID> Create-Customer-Request-Message </ID>
<InstanceID> CREATECUSTREQMSG/9009 </InstanceID>
```

```
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOperationName>
</BusinessScope>
```

Message 4: Create Customer Response EBM

Example 25–24 shows two rows:

- One for the process
- One for the Create Customer response message in the process

Example 25–24 Create Customer Response EBM

```
<BusinessScope>
<ID> Oracle-Customer-Create-Response </ID>
<InstanceID> CREATECUSTRESP/10 09 </InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID> Create-Customer-Response-Message </ID>
<InstanceID> CREATECUSTREQMSG/9021 </InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOperationName>
</BusinessScope>
```

Message 5: ProcessOrder Request EBM

Example 25–25 shows four rows:

- One for the process
- One for the Process Order Request Message in the process
- One for the create account request message in the process (spawned immediate child)
- One for the create account response message in the process (spawned immediate child)

Example 25–25 ProcessOrder Request EBM

```
<BusinessScope>
<ID>End-to-End-Order-Processing </ID>
<InstanceID> ORDPROCESSING/1004 </InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>OrderEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>ProcessOrder</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID> Process-Order-Request-Message </ID>
<InstanceID> PROCESSORDERREQMSG /9004 </InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>OrderEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>ProcessOrder</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
```

```
<ID>Create-Account-Request-Message </ID>
<InstanceID> CREATEACCTREQMSG /9005 </InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID> Create-Account-Response-Message </ID>
<InstanceID> CREATEACCTRESPMSG /9020</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOperationName>
</BusinessScope>
```

Message 6: Process Order Response EBM

Example 25–26 shows two rows:

- One for the process
- One for the Process Order Response in the process

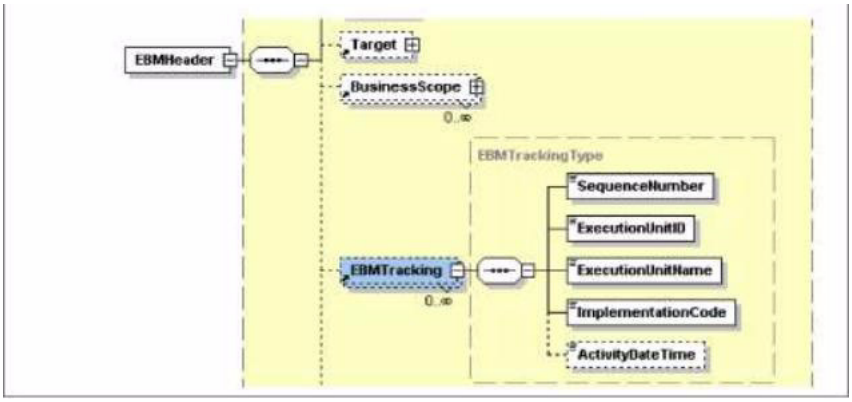
Example 25–26 Process Order Response EBM

```
<BusinessScope>
<ID>End-to-End-Order-Processing </ID>
<InstanceID> ORDPROCESSING/1004 </InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>OrderEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>ProcessOrder</EnterpriseServiceOperationName>
</BusinessScope>
<BusinessScope>
<ID> Process-Order-Response-Message</ID>
<InstanceID> PROCESSORDERRESPMSG /9019</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>OrderEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>ProcessOrder</EnterpriseServiceOperationName>
</BusinessScope>
```

25.6.8 EBMTracking

EBMTracking contains tracking information about each node that the EBM has been through, a shown in Figure 25–16. EBMTracking may appear multiple times, once for each execution unit it passes through.

Figure 25–16 Structure of the EBMTracking element



25.6.8.1 SequenceNumber

This element contains the sequence number of the node the EBM has been through.

25.6.8.2 ExecutionUnitID

This element contains the ID of the execution unit, node, or process ID.

25.6.8.3 ExecutionUnitName

This element contains the fully qualified name of the execution unit, node, or process ID.

25.6.8.4 ImplementationCode

This element contains the category of the execution unit, which indicates the type of the execution unit such as BPEL, Mediator, or Java Service.

25.6.8.5 ActivityDateTime

This element contains the timestamp indicating when the EBM was processed by the execution unit. The timestamp should be presented in UTC, which can be presented in current datetime, and GMT offset as: `yyyy '-' mm '-' dd 'T' hh ':' mm ':' ss ('.' s+)? (zzzzzz)?`

25.6.8.6 When to Populate EBM Tracking Information

Add an EBMTracking entry, as shown in Example 25–27, to the EBM header whenever a message is processed by a service.

This section should be populated when:

- Transforming a request ABM into an EBM in a requester ABC implementation service.
- Transforming a response ABM into an EBM is a provider ABC implementation service.
- The message passes through a BPEL process.

Example 25–27 Populating EBM Tracking Information

```
<EBMTracking>
<SequenceNumber>1</SequenceNumber>
<ExecutionUnitID>6 8 56 8</ExecutionUnitID>
<ExecutionUnitName>{http://xmlns.oracle.com/ABCSImpl/Siebel/Invoice/v0}
```

```
QueryInvoiceSiebelReqABCSImpl</ExecutionUnitName>
<CategoryCode>BPEL</CategoryCode>
<ActivityDateTime>2 001-12-17T09:30:47-05:00</ActivityDateTime>
</EBMTracking>
<EBMTracking>
<SequenceNumber>2</SequenceNumber>
<ExecutionUnitID>4435</ExecutionUnitID>
<ExecutionUnitName>OrderEBS</ExecutionUnitName>
<ExecutionUnitName>{http://xmlns.oracle.com/EnterpriseServices/Core/Invoice/v0}
QueryInvoiceEBS</ExecutionUnitName>
<CategoryCode>ESB</CategoryCode>
<ActivityDateTime>2 001-12-17T09:30:47-05:00</ActivityDateTime>
</EBMTracking>
<EBMTracking>
<SequenceNumber>3</SequenceNumber>
<ExecutionUnitID>6 8 594</ExecutionUnitID>
<ExecutionUnitName>{http://xmlns.oracle.com/ABCSImpl/Portal/Invoice/v0}Query
InvoicePortalProvABCSImpl</ExecutionUnitName>
<CategoryCode>BPEL</CategoryCode>
<Activity DateTime>2 001-12-17T09:30:47-05:00</ActivityDateTime>
</EBMTracking>
```

25.6.9 Custom

You can extend the custom types to add any extra elements or attributes you may need. You can also take advantage of the XSLT file extensibility features to add the necessary code to either populate or read values or both from the custom section.

26

Configuring Oracle AIA Processes for Error Handling and Trace Logging

This chapter provides an overview of Oracle BPEL and Mediator Process Error Handling and AIA Error Handler Framework and describes how to enable AIA Processes for Fault Handling, implement Error Handling for the Synchronous Message Exchange Pattern, implement Error Handling and Recovery for the Asynchronous Message Exchange Pattern to ensure guaranteed message delivery, configure AIA Services for notification, describe the FaultNotification Element, extend fault messages, extend error handling and how to configure Oracle AIA Processes for Trace Logging.

This chapter includes the following sections:

For more information about Oracle AIA B2B error handling, see Chapter 19, "Introduction to B2B Integration Using AIA."

26.1 Overview of Oracle BPEL and Mediator Process Error Handling

This section includes the following topics:

26.1.1 Understanding Oracle BPEL Error Handling

The Oracle Application Integration Architecture (AIA) Error Handling Framework groups BPEL process errors into two categories:

- Run-time faults
- Business faults

Run-time Faults

A run-time fault can occur in one of two scenarios:

- The partner link invocation fails.
- The partner link receives a named fault indicating that it is a run-time fault.

These faults are not user-defined and are issued by the system. Some situations in which a BPEL process can encounter a run-time fault include the following:

- The process tries to use a value incorrectly.
- A logic error occurs.
- A SOAP fault occurs in a SOAP call.
- An exception is issued by the server, and so forth.

AIA Services built as BPEL processes should be enabled and configured to catch and handle the run-time faults.

Business Faults

A business fault can occur in one of two scenarios:

- A BPEL process runs a throw activity after evaluating a condition as a fault.
- An Invoke activity receives a named fault indicating that it is a business fault.

Business faults are the application-specific faults that are generated when erroneous conditions take place when the message is being processed. These faults are specified by the BPEL process component and are defined in the WSDL.

AIA Services built as BPEL processes should be enabled and configured to catch and handle the run-time, and business faults.

26.1.2 Understanding Oracle Mediator Error Handling

A Mediator component can handle both run-time faults and business faults.

Run-time Faults

These are faults that occur because of some problem in the underlying system, such as the network not being available.

Business Faults

These are exceptions that are returned by called Web services. These are application-specific and are explicitly defined in the service's WSDL file.

AIA Services built as Mediator components should be configured to catch and handle the business faults.

However, fault policies are applicable to parallel routing rules only. For sequential routing rules, the fault goes back to the caller and it is the responsibility of the caller to handle the fault.

AIA recommends the usage of sequential routing rules only.

For more information about configuring the Mediator to handle business faults arising from synchronous invocations using sequential routing rules, see Section 26.4.3, "Guidelines for Configuring Mediator for Handling Business Faults."

26.2 Overview of AIA Error Handler Framework

For more information about the Error Handling Framework and its features, see "Setting Up Error Handling" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

26.3 Enabling AIA Processes for Fault Handling

This section includes the following topics:

- Section 26.3.1, "What You Must Know About Fault Policy Files"
- Section 26.3.2, "How to Implement Fault Handling in BPEL Processes"

26.3.1 What You Must Know About Fault Policy Files

A fault policy bindings file associates the policies defined in a fault policy file with the SOA composite application (or the service component or reference binding component). The fault policy bindings file must be named fault-bindings.xml. This conforms to the fault-bindings.xsd schema file.

Fault policy file names are not restricted to one specific name. However, AIA recommends a naming convention to be followed for the fault policy files. All fault policy files should be named using the convention <ServiceName>FaultPolicy.xml. They must conform to the fault-policy.xsd schema file.

For more information about naming conventions, see Chapter 31, "Oracle AIA Naming Standards for AIA Development."

AIA recommends that the fault policy bindings file should be defined to associate the policies defined in a fault policy file with the SOA composite application.

AIA Foundation Pack comes with a default fault policy, which is stored in Oracle Metadata Services (MDS), in the AIAMetaData/faultPolicies/V1 folder. When default fault policies are to be used, then the composite.xml file should have the elements shown in Example 26–1 added to it

Example 26–1 Elements to be Added to composite.xml

```
<property name="oracle.composite.faultPolicyFile">[pointer to the fault policy
 xml file in the MDS]</property>
<property name="oracle.composite.faultBindingFile">[pointer to the fault policy
 bindings file fault-bindings.xml in the MDS]</property>
```

When Service Constructor is used to construct the AIA Services, and if the developer opts for using a default fault policy file, then Service Constructor automatically inserts the preceding elements in the `composite.xml` file.

For more information about Service Constructor, see Chapter 4, "Working with Service Constructor."

If a developer chooses to have a customized, service-specific fault policy file for her AIA Service, then, AIA recommends that the fault policy file and fault policy bindings

file (fault-bindings.xml) be placed in the same directory as the composite.xml file of the SOA composite application.

When a developer is using Service Constructor to construct the AIA Services and opts for using a service-specific fault policy file and fault policy bindings file, then the tool creates a template file in the same directory as the composite.xml file of the SOA composite application. The developers must define the fault policies in those template files. In this case, the tool dos not create the XML elements <property> in the composite.xml.

For more information, see "Schema Definition File for Fault-policies.xml" and "Schema Definition File for Fault-bindings.xml" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

26.3.1.1 Associating a Fault Policy File with Fault Policy Bindings File

The following example shows how to associate a fault policy defined in a sample fault-policy file with a fault-policy binding.xml file.

Consider a sample fault policy file, SamplesQueryCustomerPartyPortalProvABCSImplFaultPolicy.xml, with the fault policies defined as shown in Example 26–2.

Example 26–2 Sample Fault Policy File with Fault Policies Defined

```
<faultPolicies xmlns="http://schemas.oracle.com/bpel/faultpolicy">
   <faultPolicy version="2.0.1"
   id="SamplesQueryCustomerPartyPortalProvABCSImplFaultPolicy" . . . . . .>
   </faultPolicy>
</faultPolicies>
```

Associate the policies defined in the preceding fault policy file with the level of fault policy binding that you are using-either a SOA composite application or a component (BPEL process or Oracle Mediator service component).

To do this, modify the template fault-bindings.xml file (created by the AIA Service Constructor when the developer chooses to have a service-specific fault policy instead of using a default fault policy).

In the fault-bindings.xml file, the association is done as shown in Example 26–3.

Example 26–3 Association in fault-bindings.xml

```
<faultPolicyBindings version="2.0.1"xmlns="http://schemas.oracle.com/bpel/
 faultpolicy" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <compositefaultPolicy="SamplesQueryCustomerPartyPortalProvABCSImpl
    FaultPolicy"/>
</faultPolicyBindings>
```

> **Note:** In this example, the association is made at the level of the SOA composite application.

AIA recommends that the fault policy binding level be a SOA composite application by default because this conforms with our recommendation that a composite must be built with a single component in it.

26.3.2 How to Implement Fault Handling in BPEL Processes

To implement fault handling in a BPEL process:

1. Define an EBM HEADER variable in the BPEL process and populate it as the first step.

   The Error Handling Framework uses this variable to populate the fault message with contextual details from the Enterprise Business Message (EBM) header. If the BPEL process is an Application Business Connector Service (ABCS), then input is an Application Business Message (ABM), and the EBM HEADER variable should be populated as soon as the ABM is converted to an EBM. This way, if the error occurs on a partner link after this transformation step, the Error Handling Framework accesses and uses the EBM header details.

2. Define a fault policy for the BPEL process and bind the process with this policy in fault-bindings.xml.

   When you are using a service-specific fault policy file, always use the CompositeJavaAction, oracle.apps.aia.core.eh.CompositeJavaAction, as specified in the default policy. Including this CompositeJavaAction accounts for error notifications and error logging.

   For more information about how to define a fault policy XML file to handle business faults, see Section 26.4.2.1, "Handling Business Faults."

   For more information about how to define a fault policy XML file to handle run-time faults in the synchronous message exchange pattern (MEP), see Section 26.4, "Implementing Error Handling for the Synchronous Message Exchange Pattern."

   For more information about how to define a fault policy XML file to handle run-time faults in the asynchronous MEP, see Section 26.5, "Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery."

3. Define the catch blocks.

   The default behavior of the fault policy after the CompositeJavaAction is to do a rethrow. This returns the execution control to the catch or catch-all block specified in the BPEL process.

   In a way, interception of faults using a fault policy is transparent to you because the CompositeJavaAction rethrows the same fault that has been intercepted by it. So in BPEL, you must catch the fault, such as a binding or remote fault, which is expected out of the invoke activity.

   Hence, define a catch block for each business fault and run-time fault that can be expected at design time.

4. Define a catch-all block.

   This assists in catching any unexpected errors that may occur while you are running the process.

For more information about defining BPEL catch and catch-all blocks for the synchronous request-response MEP, see Section 26.4.2, "Guidelines for BPEL Catch and Catch-All Blocks in Synchronous Request-Response."

For more information about defining BPEL catch and catch-all blocks for the asynchronous MEP, see Section 26.5.4, "Guidelines for BPEL Catch and Catch-All Blocks."

26.4 Implementing Error Handling for the Synchronous Message Exchange Pattern

This section includes the following topics:

- Section 26.4.1, "Guidelines for Defining Fault Policies"
- Section 26.4.2, "Guidelines for BPEL Catch and Catch-All Blocks in Synchronous Request-Response"
- Section 26.4.3, "Guidelines for Configuring Mediator for Handling Business Faults"

26.4.1 Guidelines for Defining Fault Policies

This section includes the following topics:

- Section 26.4.1.1, "Defining a Fault Policy XML File for Handling Run-time Faults"
- Section 26.4.1.2, "Defining a Fault Policy XML File for Handling Business Faults"

26.4.1.1 Defining a Fault Policy XML File for Handling Run-time Faults

Define faults in the fault policy XML file per guidelines illustrated in the following code snippet. In the fault policy, define a section under Conditions as shown in Example 26–4.

Example 26–4 Fault Definition in the Fault Policy XML File

```
<faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"name="bpelx:
 remoteFault">
   <condition>
      <action ref="aia-ora-java"/>
   </condition>
</faultName>
<faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension" name="bpelx:
 bindingFault">
   <condition>
      <action ref="aia-ora-java"/>
   </condition>
</faultName>
```

Though AIA recommends that by default, remote and binding faults should be defined, as shown previously, other run-time faults can be handled in the same way if required per the functionality of the service.

For example, if you are required to handle the subLanguageExecutionFault fault, then define the section as shown in Example 26–5.

Example 26–5 subLanguageExecutionFault Fault Handling

```
<faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"name="bpelx:
 subLanguageExecutionFault">
   <condition>
      <action ref="aia-ora-java"/>
   </condition>
</faultName>
```

However, all the run-time faults that are defined in the fault policy file must be caught in the BPEL process in a catch block, which is specific to the fault.

For more information about defining BPEL catch and catch-all blocks for the synchronous request-response MEP, see Section 26.4.2, "Guidelines for BPEL Catch and Catch-All Blocks in Synchronous Request-Response."

26.4.1.2 Defining a Fault Policy XML File for Handling Business Faults

The Fault Management Framework is used to handle the business faults that are for an invoke activity. In other words, only business faults thrown by external services and applications when invoked using the invoke activity are intercepted by the Oracle Fusion Middleware Fault Management Framework, according to the definitions specified in the fault policy file.

The business faults that are internal to the BPEL, business faults thrown by a throw activity, for example, are not intercepted by the Fault Management Framework.

To define a fault policy to intercept Oracle AIA faults:

1. Examine your partner link WSDL and check to determine whether it is throwing any Oracle AIA faults.
2. If it is throwing Oracle AIA faults, look for the partner link namespace and name of the fault in the partner link WSDL.
3. In the fault policy, define a section under Conditions as shown in Example 26–6.

Example 26–6 Conditions Element in the Fault Policy

```
<Conditions>
   <faultName xmlns:corecustomerpartyebs="http://xmlns.oracle.com/Enterprise
    Services/Core/CustomerParty/V2" name="corecustomerpartyebs:fault">
      <condition>
         <test>[ XPath expression to be evaluated for the fault variable
          available in the fault]</test>
         <action ref="aia-ora-java"/>
      </condition>
   </faultName>
</Conditions>
```

> **Note:** To avoid unnecessary processing, ensure that you specify retry options only when explicitly required.

4. Configure the default condition to call the aia-ora-java action, as shown in Example 26–7.

Example 26–7 Default Condition Configuration Used to Call the aia-ora-java action

```
<Conditions>
   <faultName xmlns:corecustomerpartyebs="http://xmlns.oracle.com/Enterprise
    Services/Core/CustomerParty/V2" name="corecustomerpartyebs:fault">
      <condition>
         <test>[XPath expression to be evaluated for the fault variable
          available in the fault]</test>
         <action ref="aia-ora-java"/>
      </condition>
   </faultName>
   <faultName>
      <condition>
         <action ref="aia-ora-java"/>
```

```
      </condition>
   </faultName>
</Conditions>
```

5. All business faults defined in the fault policy file must be caught in the BPEL process in a catch-block that is specific to the business fault.

   For more information, see Section 26.4.2, "Guidelines for BPEL Catch and Catch-All Blocks in Synchronous Request-Response."

26.4.2 Guidelines for BPEL Catch and Catch-All Blocks in Synchronous Request-Response

Each BPEL process should have explicit catch blocks for remote faults, binding faults, business faults (Oracle AIA faults), and any other fault expected on a partner link at design time. Use these guidelines for defining these catch blocks.

26.4.2.1 Handling Business Faults

To handle an internal business fault:

1. In the case of a BPEL process carrying out a throw activity, construct a business fault message (Oracle AIA fault message) and populate the AIA Fault message with the ECID, as shown in Example 26–8.

Example 26–8 Business Fault Message

```
<sequence name="SequenceBusinessFault">
   <assign name="AssignBusinessFault">
      <copy>
         <from expression="ora:processXSLT('xsl/EBM_to_Fault.xsl',bpws:
          getVariableData('EBM_HEADER'))"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault"/>
      </copy>
      <copy>
         <from expression="'invalid account id'"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultMessage/corecom:Text"/>
      </copy>
      <copy>
         <from expression="'invalid account id'"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultMessage/corecom:Stack"/>
      </copy>
      <copy>
         <from expression="ora:getCompositeInstanceId()"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:
          Fault/corecom:FaultNotification/corecom:FaultingService/corecom:
          InstanceID"/>
      </copy>
      <copy>
         <from expression="'BPEL'"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultingService/corecom:
          ImplementationCode"/>
      </copy>
      <copy>
         <from expression="ora:getCompositeName()"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:
```

```
          Fault/corecom:FaultNotification/corecom:FaultingService/corecom:ID"/>
      </copy>
      <copy>
         <from expression="ora:getECID()"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultingService/corecom:
          ExecutionContextID"/>
   </assign>
   <throw name="Throw_custom_business_fault" faultName="client:fault"
    faultVariable="AIAFaultMessage"/>
</sequence>
```

> **Note:** Ensure that in EBM_to_Fault.xsl, the `<corecom:ExecutionContextID/>` element is injected under the `<corecom:FaultingService>` element.

2. Catch the preceding fault message in the catch block. In the catch block:
   - Send the AIA Fault Message as a reply.

     Invoke the AIAAsyncErrorHandlingBPELProcess with this Oracle AIA fault message as input.
   - Throw the AIA Fault Message that has been caught. This rethrow enables the process to appear as faulted in the Enterprise Manager Console.

To handle an external business fault:

In the case of an Invoke activity in the BPEL receiving an AIA fault message as a response, catch the AIA fault message in the catch block. In the catch block:

- Send the AIA Fault Message as reply.
- Throw the AIA Fault Message that has been caught. This rethrow enables the process to appear as faulted in the Oracle Enterprise Manager Console.

> **Note:** In this case, AIA does not invoke the AIAAsyncErrorHandlingBPELProcess because the business fault is handled by the Oracle Fusion Middleware Fault Management Framework, according to the fault polices defined in the associated fault policy file.

26.4.2.2 Handling Run-time Faults Defined in the Fault Policy File

For each of the run-time faults that has been defined in the fault policy xml file, have a catch block in the BPEL.

To handle run-time faults defined in the fault policy file:

1. In the catch block, construct an Oracle AIA fault message.
2. Send this Oracle AIA fault message as the reply.
3. Rethrow the fault that has been caught. This enables the process to appear as faulted in the Oracle BPEL Console.

26.4.2.3 Handling Run-time Faults Not Defined in the Fault Policy File

Each BPEL process should also have a catch-all block to process run-time faults that are not caught in catch-blocks and not defined in the fault policy file.

To define the catch-all block:

1. Construct an Oracle AIA fault message. Populate the AIA Fault message with ECID as shown in Example 26–9.

Example 26–9 Catch-All Block Construction

```
<sequence name="SequenceCatchAll">
   <assign name="AssignFault">
      <copy>
         <from expression="ora:processXSLT('xsl/EBM_to_Fault.xsl',bpws:
          getVariableData('EBM_HEADER'))"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault"/>
      </copy>
      <copy>
         <from expression="ora:getFaultAsString()"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultMessage/corecom:Text"/>
      </copy>
      <copy>
         <from expression="ora:getFaultAsString()"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultMessage/corecom:Stack"/>
      </copy>
      <copy>
         <from expression="ora:getCompositeInstanceId()"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultingService/corecom:
          InstanceID"/>
      </copy>
      <copy>
         <from expression="'BPEL'"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultingService/corecom:
          ImplementationCode"/>
      </copy>
      <copy>
         <from expression="ora:getCompositeName()"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultingService/corecom:ID"/>
      </copy>
      <copy>
         <from expression="ora:getECID()"/>
         <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/
          corecom:FaultNotification/corecom:FaultingService/corecom:
          ExecutionContextID"/>
   </assign>
</sequence>
```

Note: Ensure that in EBM_to_Fault.xsl, the `<corecom:ExecutionContextID/>` element is injected under the `<corecom:FaultingService>` element.

2. Invoke the AIAAsyncErrorHandlingBPELProcess with this Oracle AIA fault message as input.
3. Send this Oracle AIA fault message as the reply.
4. Throw AIA fault message. This enables the process to appear as faulted in the Oracle Enterprise Manager Console.
5. Unless otherwise required, these catch and catch-all blocks can be defined at the top-level scope and are not required to be defined at the scope for each partner link.

For more information about the Fault Management Framework, see "Using the Fault Management Framework" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

26.4.3 Guidelines for Configuring Mediator for Handling Business Faults

Oracle Mediator provides fault policy-based error handling for business faults. However, fault policies are applicable to parallel routing rules only. For sequential routing rules, the fault goes back to the caller (that has invoked the mediator) and it is the responsibility of the caller to handle the fault.

AIA recommends using only sequential routing rules.

When a service invoked by the mediator throws a business fault, this fault must be propagated up to the service that has invoked the mediator.

This section discusses how to configure the mediator to handle business exceptions returned by the services invoked by the mediator.

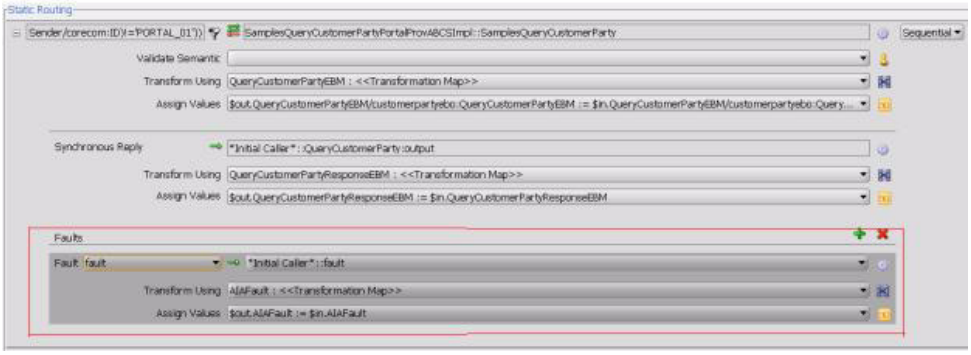
Here are some points to consider:

- When the mediator invokes a service, the invoked service can throw any of the business faults that are defined in its WSDL.
- The fault that is thrown by the invoked service is propagated back to the mediator.

To configure the mediator to handle business exceptions returned by the services invoked by the mediator:

1. Open the mediator in design-mode using Oracle JDeveloper.
2. Using the Static Routing panel, assign the inbound fault reaching the mediator to the outbound fault, which is now propagated by the mediator to the service that invoked it, as shown in Figure 26–1.

Figure 26–1 Assignment of the Faults in the Mediator



3. As shown in the preceding diagram, assign the inbound fault from the target service's WSDL operation to the outbound fault that is propagated by the mediator to its invoker service.

   For more information about how to assign faults, see "How to Handle Faults" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

26.5 Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery

This section includes the following topics:

26.5.1 Overview

In the context of AIA, guaranteed message delivery for the asynchronous MEP means that the message initiated from a sender is persisted until it is successfully delivered to and acknowledged by the receiver, if acknowledgement is expected.

The sender and receiver are not necessarily the participating applications. Rather, they are logical milestones in an Oracle AIA integration flow. Multiple milestones could be in an Oracle AIA integration scenario.

Temporary unavailability of any hardware or software service in an asynchronous message flow does not result in a lost message or a delivery failure. The Error Handling framework provides a way for the message to be persisted until the hardware or software service becomes available.

After an integration administrator has been notified of the unavailable resource by the Error Console, she can address the resource issue. The integration administrator can then use the Message Resubmission Utility to resubmit the persisted message into the integration scenario from the appropriate transaction milestone point, enabling its delivery to the next component or milestone.

For more information about running the Message Resubmission Utility, see "Using the Message Resubmission Utility" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

These points summarize primary aspects of an implementation of the guaranteed message delivery programming model for the asynchronous MEP.

- Message persistence milestones

  Messages are picked from a persistence store (source), processed, and pushed to the next persistence store (target). The message is not removed from the source until it has been successfully processed and delivered to the target. The source and target may be applications. Each persistence store represents a milestone and may be a database, file system, JMS queue, or JMS topic.

For more information about configuring milestones, see Section 26.5.2, "Configuring Milestones."

- Global transaction

  These tasks must be accomplished as a part of the global transaction:

  - Picking up the message from the source.
  - Processing the message by one or more services.
  - Delivering the message to the target.
  - The initiation of a service from the source with an input message initiates a transaction. All the services invoked downstream participate in this global transaction. This global transaction ends or is committed when the message is successfully delivered to the target and removed from the source.

    In the case of an error, an exception is raised and the transaction initiated is rolled back with the message safe in the source. The message is either in the source or target and is not lost.

    For more information about configuring the global transaction, see Section 26.5.3, "Configuring Services Between Milestones."

- Error handling and recovery

  Any exceptions due to system or business errors must generate a rollback of all preceding services and trigger a single error notification to the Integration Administrator. This requires marking the message in the source as faulted, preventing it from being processed until the error condition is removed.

  For more information about configuring error rollback, see Section 26.5.6, "Configuring Fault Policies to Not Issue Rollback Messages."

In case of system errors, after the exception condition has been removed, the faulted messages in the source must be reset. This enables them to be resubmitted. The Message Resubmission Utility can be used to resubmit the messages for reprocessing by the correct source.

In case of business errors, the faulted messages in the source must be removed and sent to fallout management for further action. Fallout management is a custom implementation in which the messages encountering business errors are segregated and processed separately.

For example, suppose that orders submitted for processing encounter a business error. As a part of an Order Fallout Management implementation, the Order message and error message are routed to an application that introspects the error messages and raises a trouble ticket that provides an explanation of the error and the suggested remedial action. After the remedial action is taken, the order is reprocessed.

Error handling and recovery for the asynchronous MEP are implemented as follows to ensure guaranteed message delivery:

1. Ensure that each message has a unique message identifier.
2. Populate the EBM header with the source milestone identifier.
3. Ensure that the fault notification contains the message identifier and source milestone identifier of the faulted message.
4. Use the Message Resubmission Utility to recover and resubmit a faulted message.

For more information about how to implement these configurations, see Section 26.5.2, "Configuring Milestones" and Section 26.5.3, "Configuring Services Between Milestones."

For more information about using the Message Resubmission Utility, see "Using the Message Resubmission Utility" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

For more information about the Message Resubmission Utility API, see Section 26.5.7, "Using the Message Resubmission Utility API."

26.5.2 Configuring Milestones

As a part of implementing error handling and recovery for the asynchronous MEP to ensure guaranteed message delivery, messages must be persisted at milestones. The movement of messages between milestones must be guaranteed.

A milestone can be a JMS queue or a JMS topic.

Figure 26–2 and Figure 26–3 illustrate a few possible milestone locations across an integration flow.

Figure 26–2 Integration Flow in Which the Receiver Target Milestone is the Target Participating Application

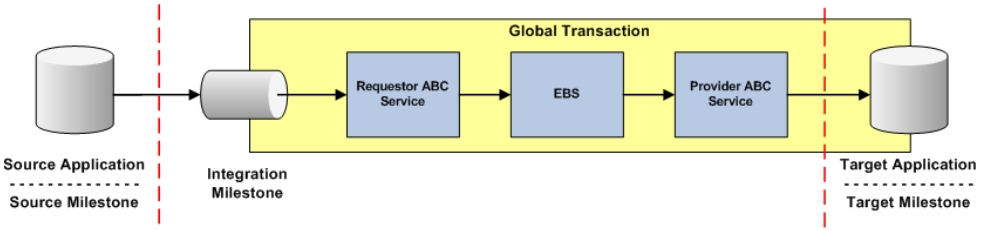
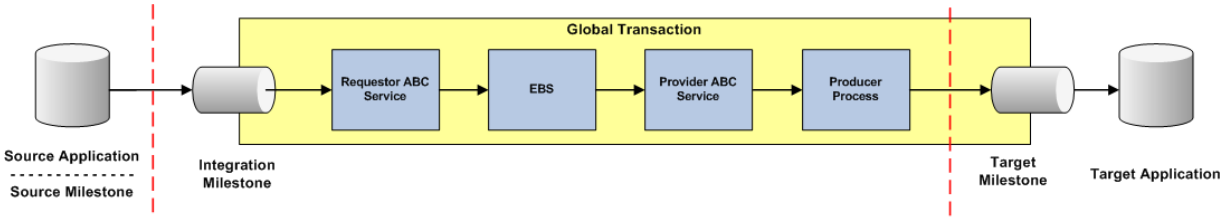


Figure 26–3 Integration Flow in Which the Receiver Target Milestone is Within the Global Transaction Space



26.5.3 Configuring Services Between Milestones

This section includes the following topics:

- Section 26.5.3.1, "Populating Message Resubmission Values"
- Section 26.5.3.2, "Configuring All Services to Participate in a Single Global Transaction"

Completing these activities ensures that the services between milestones are configured to provide error handling and recovery for the asynchronous MEP to ensure guaranteed message delivery.

26.5.3.1 Populating Message Resubmission Values

These parameters in the EBM header must be populated in the JMS consumer service transformation:

- SenderResourceTypeCode

  Indicates the type of resource or system in which the rolled-back message is stored, whether Queue or Topic.

- SenderResourceID

  Identification of the resource/system of type SenderResourceTypeCode.

- SenderMessageID

  Identification of the message persisted in the resource/system associated with type SenderResourceTypeCode.

Scenario 1

When an ABM in the JMS Queue or Topic is triggering the JMS Consumer Service, then the preceding information must be passed to the requester ABCS as a part of the ABM header.

For more information, see Section 26.5.3.1.1, "Populating the ABM with Message Resubmission Values in JMSConsumerAdapter."

In the JMS Consumer Service, this information must be configured to be sent to the ABM header. In the requester ABCS, this information is extracted from the ABM header and sent to the EBM header in the transformation.

For more information, see Section 26.5.3.1.2, "Populating the EBM Header with Resubmission Values in the Requester ABCS."

Scenario 2

When an EBM in the JMS Queue or Topic is triggering the JMS consumer service, use an EBM-to-EBM transformation and populate (overwrite) the resubmission values in the EBM message. The following values in the EBM header fields of the inbound message should be overwritten with the new values for the ResourceType, Resource Name, and JMS Message ID pertaining to the current milestone.

- `<corecom:SenderResourceTypeCode>`
- `<corecom:SenderResourceID>`
- `<corecom:SenderMessageID>`

Example 26–10 provides sample code snippet from the EBM-to-EBM XSL.

Example 26–10 EBM-to-EBM XSL Code Example

```
<corecom:IntermediateMessageHop>
   <corecom:SenderResourceTypeCode>
      <xsl:value-of select="/custebo:CreateCustomerPartyListEBM/corecom:
       EBMHeader/corecom:FaultNotification/corecom:FaultMessage/corecom:
       IntermediateMessageHop/corecom:SenderResourceTypeCode"/>
   </corecom:SenderResourceTypeCode>
   <corecom:SenderResourceID>
      <xsl:value-of select="/custebo:CreateCustomerPartyListEBM/corecom:
       EBMHeader/corecom:FaultNotification/corecom:FaultMessage/corecom:
       IntermediateMessageHop/corecom:SenderResourceID"/>
   </corecom:SenderResourceID>
   <corecom:SenderMessageID>
```

```
      <xsl:value-of select='mhdr:getProperty("in.property.jca.jms.
       JMSMessageID")'/>
   </corecom:SenderMessageID>
</corecom:IntermediateMessageHop>
```

26.5.3.1.1 Populating the ABM with Message Resubmission Values in JMSConsumerAdapter

Ensure that the ABM is enriched with the following content:

- The unique Message ID. The JMSMessageID in the JMS header is used as the value.
- The resource name. This is the JMS Queue/Topic name. This is the same as the `<svcdoc: ResourceName>` value in the JMSConsumerAdapter's composite.xml.
- The type of the resource. The possible values are Queue or Topic.

Use specifically designated fields in the ABM for this purpose. These fields are identified at the time of design.

Within the mediator-based JMSConsumerAdapter:

- Use the transformation step in the mediator routing rule.
- In the XSL used by the transformation, assign the values of the JMS Message ID, Resource Name, and Resource Type to the specifically designated fields of the ABM.

Example 26–11 illustrates how to assign the JMS Message ID to the ABM.

Example 26–11 Example of How to Assign the JMS Message ID to the ABM

```
<xsl:attribute name="MessageId">
   <xsl:value-of select='mhdr:getProperty("in.property.jca.jms.JMSMessageID")'/>
</xsl:attribute>
```

In this example, the assumption is that the ABM has a specific attribute, MessageId, to which the JMS Message ID is assigned.

In some cases, only one designated field may be available in the ABM. In such scenarios, concatenate the values of the JMS Message ID, Resource Name, and Resource Type and assign the value to the specific designated field of the ABM. While concatenating these values, AIA recommends using `::` as the separator.

For example, consider a Siebel Customer ABM, ListOfCmuAccsyncAccountIo. In this ABM, assume that the MessageId attribute of the ListOfCmuAccsyncAccountIo element is designated to hold the information about JMS Message ID, Resource Name, and Resource Type at design time. Example 26–12 illustrates how to concatenate the data and assign it to the ABM.

Example 26–12 Example of How to Concatenate Data and Assign it to the ABM

```
<xsl:attribute name="MessageId">
   <xsl:value-of select='concat(mhdr:getProperty("in.property.jca.jms.
    JMSMessageID"),":: SampleQueue","::Queue")'/>
</xsl:attribute>
```

26.5.3.1.2 Populating the EBM Header with Resubmission Values in the Requester ABCS

When the ABM arrives at the requester ABCS, it contains the JMS Message ID, Resource

Name, and Resource Type values because these values were made available in the designated fields of the ABM.

> **Tip:** Ensure that the `<corecom: FaultNotification/>` element is inserted into the EBM header when transforming the ABM to the EBM.

Extract these values from the ABM and enter the following elements in the EBM header within the `<corecom:IntermediateMessageHop>` element:

- `<corecom:SenderResourceTypeCode>`: Populate it with the Resource Type value
- `<corecom:SenderResourceID>`: Populate it with the Resource Name value
- `<corecom:SenderMessageID>`: Populate it with the JMS Message ID value

The XSL that performs the ABM-to-EBM transformation should accomplish this task. The task is straightforward when three different ABM fields are designated for holding the three resubmission values, as shown in Example 26–13.

Example 26–13 Example Illustrating Three ABM Fields Used to Hold Three Resubmission Values

```
<corecom:IntermediateMessageHop>
   <corecom:SenderResourceTypeCode>
      <xsl:value-of select="[xpath to the ABM field holding the ResourceType]"/>
   </corecom:SenderResourceTypeCode>
   <corecom:SenderResourceID>
      <xsl:value-of select="[xpath to the ABM field holding the
       Resource Name]"/>
   </corecom:SenderResourceID>
   <corecom:SenderMessageID>
      <xsl:value-of select="[xpath to the ABM field holding the Message Id]"/>
   </corecom:SenderMessageID>
</corecom:IntermediateMessageHop>
```

The following content discusses how resubmission values are extracted from the ABM and assigned to the EBM header when all three resubmission values are concatenated and assigned to a single designated field in the ABM.

For example, consider a Siebel Customer ABM, ListOfCmuAccsyncAccountIo. In this ABM, assume that the MessageId attribute of the ListOfCmuAccsyncAccountIo element has been designated to hold the information about the JMS Message ID, Resource Name, and Resource Type, concatenated using `::` as a separator. See the previous section for more information.

Example 26–14 provides a code snippet that extracts the resubmission values and assign them to EBM header elements.

Example 26–14 Code Used to Extract Resubmission Values and Assign Them to EBM Header Element

```
<xsl:variable name="MsgId">
   <xsl:value-of select="substring-after(/seblcustabo:ListOfCmuAccsyncAccountIo/
    @MessageId,'::')"/>
</xsl:variable>
<corecom:IntermediateMessageHop>
   <corecom:SenderResourceTypeCode>
      <xsl:value-of select="substring-after($MsgId,'::')"/>
   </corecom:SenderResourceTypeCode>
```

```
   <corecom:SenderResourceID>
      <xsl:value-of select="substring-before($MsgId,':: ')"/>
   </corecom:SenderResourceID>
   <corecom:SenderMessageID>
      <xsl:value-of select="substring-before(/seblcustabo:ListOfCmuAccsync
       AccountIo/@MessageId,'::')"/>
   </corecom:SenderMessageID>
</corecom:IntermediateMessageHop>
```

The XSL that performs the ABM-to-EBM transformation should accomplish this task.

For more information about resubmitting messages, see "Using the Message Resubmission Utility" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

26.5.3.2 Configuring All Services to Participate in a Single Global Transaction

Configure a single global transaction as follows:

- Ensure that no commit points are between two milestones.
- Ensure that the work done between two milestones is one logical unit of work.

For more information about configuring the global transaction, see Chapter 13, "Designing and Developing Enterprise Business Services", Chapter 14, "Designing Application Business Connector Services", Chapter 15, "Constructing the ABCS", and Chapter 18, "Designing and Constructing Enterprise Business Flows."

26.5.4 Guidelines for BPEL Catch and Catch-All Blocks

This section includes the following topics:

- Section 26.5.4.1, "Handling Run-time Faults Defined in the Fault Policy File"
- Section 26.5.4.2, "Handling Run-time Faults Not Defined in the Fault Policy File"

26.5.4.1 Handling Run-time Faults Defined in the Fault Policy File

For each of the run-time faults that has been defined in the fault policy xml file, have a catch block in the BPEL. In the catch block, rethrow the fault that has been caught. This enables the process to appear as faulted in the Oracle Enterprise Manager Console.

26.5.4.2 Handling Run-time Faults Not Defined in the Fault Policy File

Each BPEL process should also have a catch-all block to process run-time faults that are not caught in catch-blocks and not defined in the fault policy file.

To define the catch-all block:

1. Construct an Oracle AIA fault message. Populate the AIA Fault message with ECID as shown in Example 26–15.

Example 26–15 AIA Fault Message with an ECID Defined

```
<copy>
   <from expression="ora:getECID()"/>
   <to variable="AIAFaultMessage" part="AIAFault" query="/corecom:Fault/corecom:
    FaultNotification/corecom:FaultingService/corecom:ExecutionContextID"/>
</copy>
```

2. Invoke the AIAAsyncErrorHandlingBPELProcess with this Oracle AIA fault message as input.

3. Throw the AIA fault message. This enables the process to appear as faulted in the Oracle Enterprise Manager Console.
4. Unless otherwise required, these catch and catch-all blocks can be defined at the top-level scope and are not required to be defined at the scope for each partner link.

For more information, see *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

26.5.5 Guidelines for Defining Fault Policies

For more information, see Section 26.4.1.1, "Defining a Fault Policy XML File for Handling Run-time Faults."

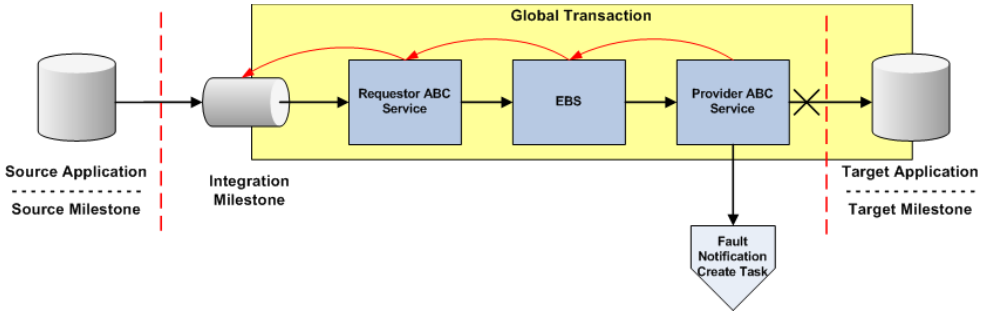
26.5.6 Configuring Fault Policies to Not Issue Rollback Messages

According to the guaranteed message delivery programming model, 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 an Oracle Advanced Queue, JMS Topic, or Mediator Resequencer Store. The message is persisted here until it can be resubmitted for delivery to the service or component.

The BPEL processes along the transaction rollback path issue fault messages and should be configured to not issue rollback messages as well. The configuration deciphers a rollback transaction so that services in the rollback path do not issue unnecessary notifications.

Without this configuration to suppress rollback messages, these processes issue unnecessary notifications. For example, in the transaction rollback flow illustrated in Figure 26–4, redundant rollback notifications would be sent out by the Requester ABCS, in addition to the one sent out by the Provider ABCS, which is the only one that should be issued.

Figure 26–4 Transaction Rollback Flow



To suppress unnecessary notifications for a rollback transaction:

- Use bpelx:rollback instead of throw in the catch blocks: `<throw name="ThrowEBSFault" faultName="bpelx:rollback"/>`
- Use a Java snippet to invoke the Oracle AIA Error Handler, as shown in Example 26–16.

Example 26–16 Java Snippet to Invoke the Oracle AIA Error Handler

```
<bpelx:exec name="Java_Embedding_1" language="java" version="1.5">
```

```
   <![CDATA[ oracle.apps.aia.core.eh.InvokeBusinessErrorHandler.process
    ((oracle.xml.parser.v2.XMLElement)getVariableData("inputVariable",
    "FaultMessage","/ns1:Fault"));]]>
</bpelx:exec>
```

- Add an empty no-op action to the fault policies of Mediator and BPEL processes along the transaction rollback flow. This empty no-op action is `aia-no-action`.

When a Mediator or BPEL process receives a rollback message, the control is directed to the class oracle.apps.aia.core.eh.CompositeJavaNoAction, which is implemented against the `aia-no-action` action.

The oracle.apps.aia.core.eh.CompositeJavaNoAction class is an empty operation, meaning that it does nothing, and thus suppresses further notifications in the rollback flow.

These sample BPEL and Mediator fault policies illustrate the way in which these conditions should be defined in impacted fault policy files.

The `aia-no-action` fault policy contains a filter expression to perform no action in the case of the rollback fault ORABPEL-02180. An example is illustrated in Example 26–17.

Example 26–17 Sample Fault Policy Using the aia-no-action No-op Action

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicy version="2.0.1" id="SamplesCreateCustomerPartyPortal
 ProvABCSImplFaultPolicy" xmlns:env="http://schemas.xmlsoap.org/soap/
 envelope/" xmlns:xs="http://www.w3.org/2001/XMLSchema"xmlns="http://schemas.
 oracle.com/bpel/faultpolicy"xmlns:xsi="http://www.w3.org/2001/XMLSchema-
 instance">
   <Conditions>
      <faultNamexmlns:plmfault="http://xmlns.oracle.com/EnterpriseServices
       /Core/CustomerParty/V2" name="plmfault:fault">
         <condition>
            <test>$fault.summary/summary[contains(., "ORABPEL-02180")]</test>
            <action ref="aia-do-nothing"/>
         </condition>
      <condition>
            <action ref="aia-ora-java"/>
         </condition>
      </faultName>
      <faultName>
         <condition>
            <test>$fault.summary/summary[contains(., "ORABPEL-02180")]</test>
            <action ref="aia-do-nothing"/>
         </condition>
         <condition>
            <action ref="aia-ora-java"/>
         </condition>
      </faultName>
   </Conditions>
   <Actions>
      <!--  This action will attempt 8 retries at increasing intervals of
       2, 4, 8, 16, 32, 64, 128, and 256 seconds. -->
      <Action id="aia-ora-retry">
         <retry>
            <retryCount>1</retryCount>
            <retryInterval>2</retryInterval>
            <exponentialBackoff/>
            <retryFailureAction ref="aia-ora-java"/>
```

```
            <retrySuccessAction ref="aia-ora-java"/>
         </retry>
      </Action>
      <!--  This is an action will cause a replay scope fault -->
      <Action id="ora-replay-scope">
            <replayScope/>
      </Action>
      <!-- This is an action will bubble up the fault  -->
      <Action id="ora-rethrow-fault">
            <rethrowFault/>
      </Action>
      <!-- This is an action will mark the work item to be "pending recovery
       from console" -->
      <Action id="ora-human-intervention">
         <humanIntervention/>
      </Action>
      <!--  This action will cause the instance to terminate  -->
      <Action id="ora-terminate">
         <abort/>
      </Action>
      <Action id="aia-do-nothing">
         <javaAction className="oracle.apps.aia.core.eh.CompositeJavaNo
          Action" defaultAction="ora-rethrow-fault">
            <returnValue value="REPLAY" ref="ora-terminate"/>
            <returnValue value="RETRHOW" ref="ora-rethrow-fault"/>
            <returnValue value="ABORT" ref="ora-terminate"/>
            <returnValue value="RETRY" ref="aia-ora-retry"/>
            <returnValue value="MANUAL" ref="ora-human-intervention"/>
         </javaAction>
      </Action>
      <Action id="aia-ora-java">
         <javaAction className="oracle.apps.aia.core.eh.CompositeJava
          Action" defaultAction="ora-rethrow-fault">
            <returnValue value="REPLAY" ref="ora-terminate"/>
            <returnValue value="RETRHOW" ref="ora-rethrow-fault"/>
            <returnValue value="ABORT" ref="ora-terminate"/>
            <returnValue value="RETRY" ref="aia-ora-retry"/>
            <returnValue value="MANUAL" ref="ora-human-intervention"/>
         </javaAction>
      </Action>
   </Actions>
</faultPolicy>
```

26.5.7 Using the Message Resubmission Utility API

The Message Resubmission Utility API enables external programs to use the functionality of enabling a message that is in error state to be ready again to be consumed for a transaction. This utility would typically be run after the associated problem that caused the message to end in error is fixed.

For more information about the Resubmission Utility, see "Using the Message Resubmission Utility" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

26.6 How to Configure AIA Services for Notification

This section discusses the standard configuration steps that must be performed when you are handling a BPEL fault.

This section includes the following topics:

- Section 26.6.1, "Defining Corrective Action Codes"
- Section 26.6.2, "Defining Error Message Codes"

For more information about how to define notification roles, see "Setting Up Error Handling" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

26.6.1 Defining Corrective Action Codes

For custom or business faults (business faults thrown by a throw activity), define corrective action codes by adding it to AIAMessages.properties file located under <MW_HOME>/soa/modules/oracle.soa.ext_11.1.1/classes folder and restart the server. Ensure that the translated string in the language-appropriate properties file for that language is located in the same directory.

This custom XPath function is available to get details from this resource bundle in a localized format: `Signature: aia:getCorrectiveAction (String key, String locale, String delimiter)`

Parameter details include:

- Key

  The corrective action code.

- Locale

  A concatenated string of language code, country code, and variant. For example, en-US.

- Delimiter

  The delimiter used in Locale parameter, such as -.

26.6.2 Defining Error Message Codes

For custom or business faults (business faults thrown by a throw activity), define corrective action codes by adding it to AIAMessages.properties file located under <MW_HOME>/soa/modules/oracle.soa.ext_11.1.1/classes folder and restart the server. Ensure that the translated string in the language-appropriate properties file for that language is located in the same directory.

This custom XPath function is available to get details from this resource bundle in a localized format: `Signature: aia:getErrorMessage (String key, String locale, String delimiter)`

Parameter details include:

- Key

  The corrective action code.

- Locale

  A concatenated string of language code, country code, and variant, for example, en-US.

- Delimiter

  The delimiter used in Locale parameter, such as -.

26.7 Describing the Oracle AIA Fault Message Schema

This section includes the following topics:

- Section 26.7.1, "Describing the EBMReference Element"
- Section 26.7.2, "Describing the B2BMReference Element"
- Section 26.7.3, "Describing the FaultNotification Element"

The top-level element of this schema is `Fault`. It has three elements: `EBMReference`, `B2BMReference`, and `FaultNotification`, as shown in Figure 26–5 and Figure 26–6. Fault elements are described in Table 26–1.

Figure 26–5 Fault Element and Its Child Elements (1 of 2)

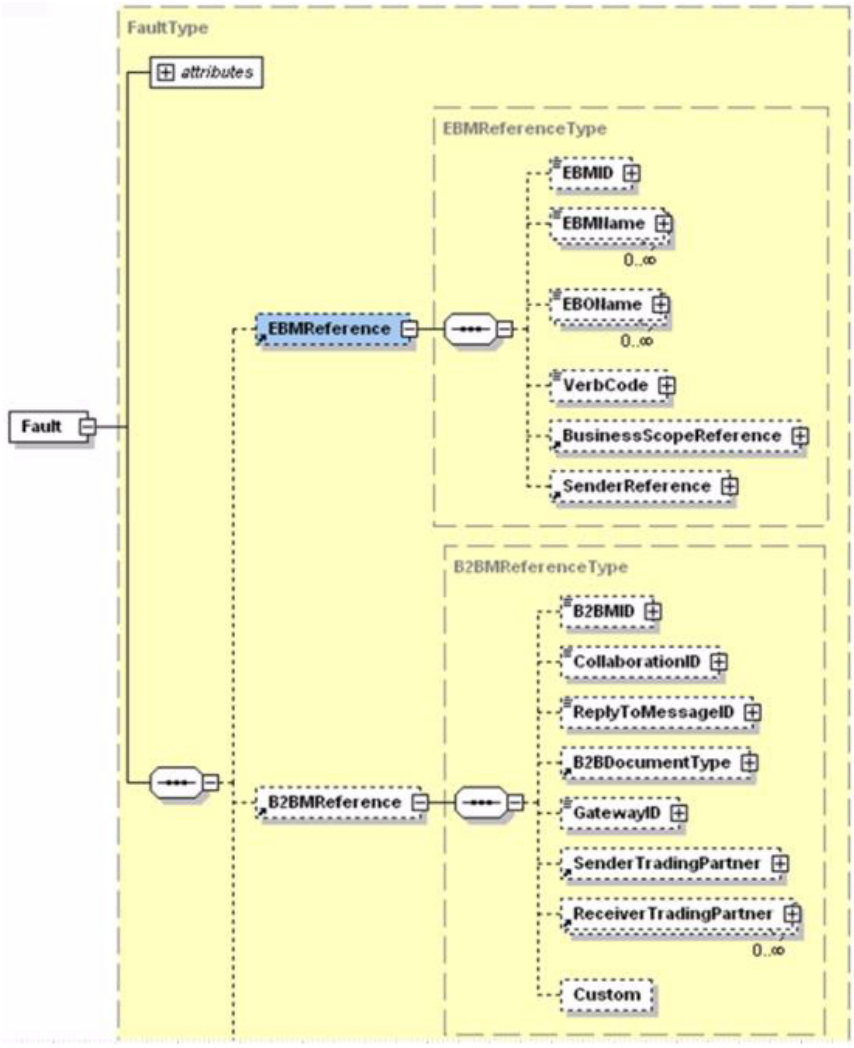


Figure 26–6 Fault Element and Its Child Elements (2 of 2)

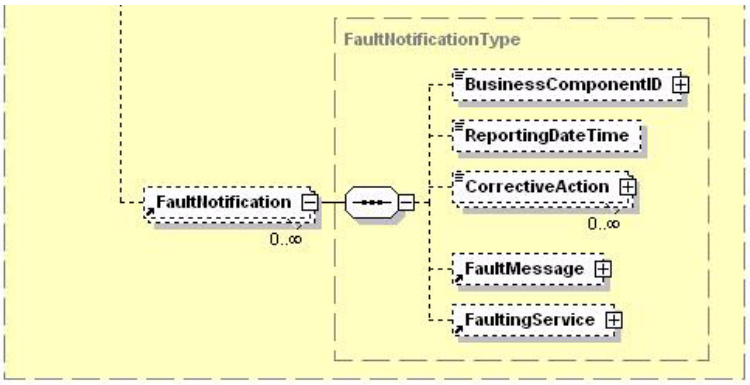


Table 26–1 Fault Elements

Name	Purpose	Details
EBMReference	Provides contextual information about the fault instance. All values are taken from the EBM header of the EBM in a faulted service instance.	For more information, see Section 26.7.1, "Describing the EBMReference Element."
B2BMReference	Provides business-to-business (B2B)-specific details when an error is in a B2B flow from Oracle AIA.	For more information, see Section 26.7.2, "Describing the B2BMReference Element."
FaultNotification	Provides actual details of the fault.	For more information, see Section 26.7.3, "Describing the FaultNotification Element."

26.7.1 Describing the EBMReference Element

This section provides details about the `EBMReference` element in the Oracle AIA fault message schema, as shown in Figure 26–7. EBM Reference elements are discussed in Table 26–2.

Figure 26–7 EBMReference Element and Its Child Elements

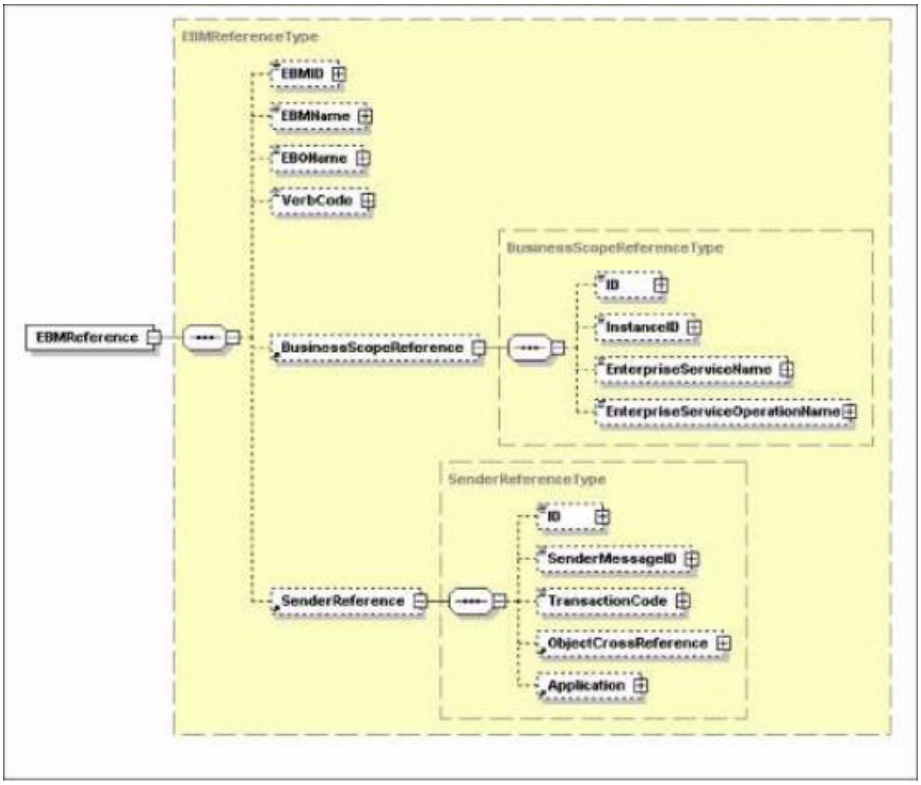


Table 26–2 EBMReference Elements

Name	Purpose
EBMID	Provides the EBMID in the message.
EBMName	Provides the EBMName in the message.
EBOName	Provides the EBOName in the message.
VerbCode	Provides the VerbCode in the message.
BusinessScopeReference	Provides the BusinessScopeReference in the message. Provides details about the end-to-end scenario in which the faulted service instance was participating. This is the instance of the BusinessScopeReference in which BusinessScopeTypeCode equals BusinessProcess.
SenderReference	Provides the SenderReference in the message.

For more information about these elements, see Section 25.6, "Introducing EBM Header Concepts."

26.7.2 Describing the B2BMReference Element

This section provides details about the `B2BMReference` element in the Oracle AIA fault message schema, as shown in Figure 26–8. B2BM reference elements are discussed in Table 26–3.

Figure 26–8 B2BMReference Element and Its Child Elements

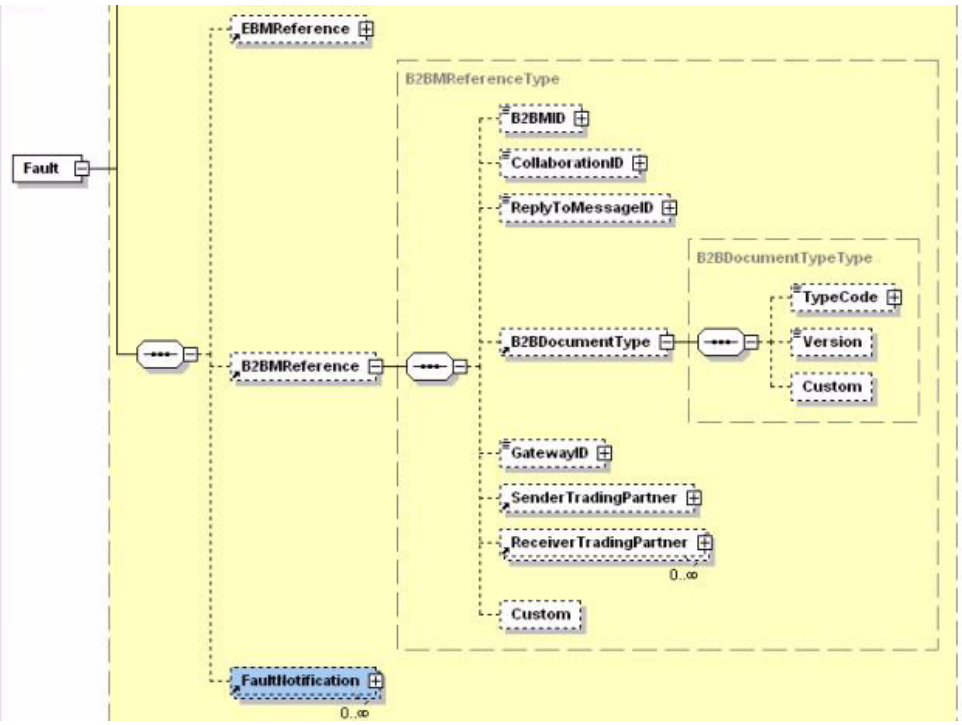


Table 26–3 B2BMReference Elements

Name	Purpose	Details
B2BMID	Provides the message ID used to identify the transaction in Oracle B2B.	A user can use this message ID to query for a failed business message in Oracle B2B and retry the failed transaction. For more information, see Chapter 19, "Introduction to B2B Integration Using AIA."
CollaborationID	Provides the collaboration ID that is common across multiple request-and-response messages related to the same business transaction.	For more information, see Chapter 19, "Introduction to B2B Integration Using AIA."
ReplyToMessageID	Provides the ID of the reply-to message.	For more information, see Chapter 19, "Introduction to B2B Integration Using AIA."
B2BDocumentType/ TypeCode	Provides the document type of the failed transaction in Oracle B2B.	This information from the fault can be used to define document-type-specific error processing. For example, you could assign errors resulting from different document types to different users for resolution. For more information, see Chapter 19, "Introduction to B2B Integration Using AIA."
B2BDocumentType/ Version	Provides the document type version of the failed transaction in Oracle B2B.	For more information, see Chapter 19, "Introduction to B2B Integration Using AIA."

Table 26–3 (Cont.) B2BMReference Elements

Name	Purpose	Details
SenderTradingPartner/TradingPartnerID	Provides the name of the sending trading partner in the B2B flow.	For more information, see Chapter 19, "Introduction to B2B Integration Using AIA."
ReceiverTradingPartner/TradingPartnerID	Provides the name of the receiving trading partner in the B2B flow.	For more information, see Chapter 19, "Introduction to B2B Integration Using AIA."
GatewayID	Provides the name of the B2B software used to initiate the flow, for example, Oracle B2B.	For more information, see Chapter 19, "Introduction to B2B Integration Using AIA."

26.7.3 Describing the FaultNotification Element

This section includes the following topics:

- Section 26.7.3.1, "FaultMessage Element"
- Section 26.7.3.2, "IntermediateMessageHop Elements"
- Section 26.7.3.3, "FaultingService Element"

This section provides details about the `FaultNotification` element in the Oracle AIA fault message schema, as shown in Figure 26–9. Fault notification elements are discussed in Table 26–4.

Figure 26–9 FaultNotification Element and Its Child Elements

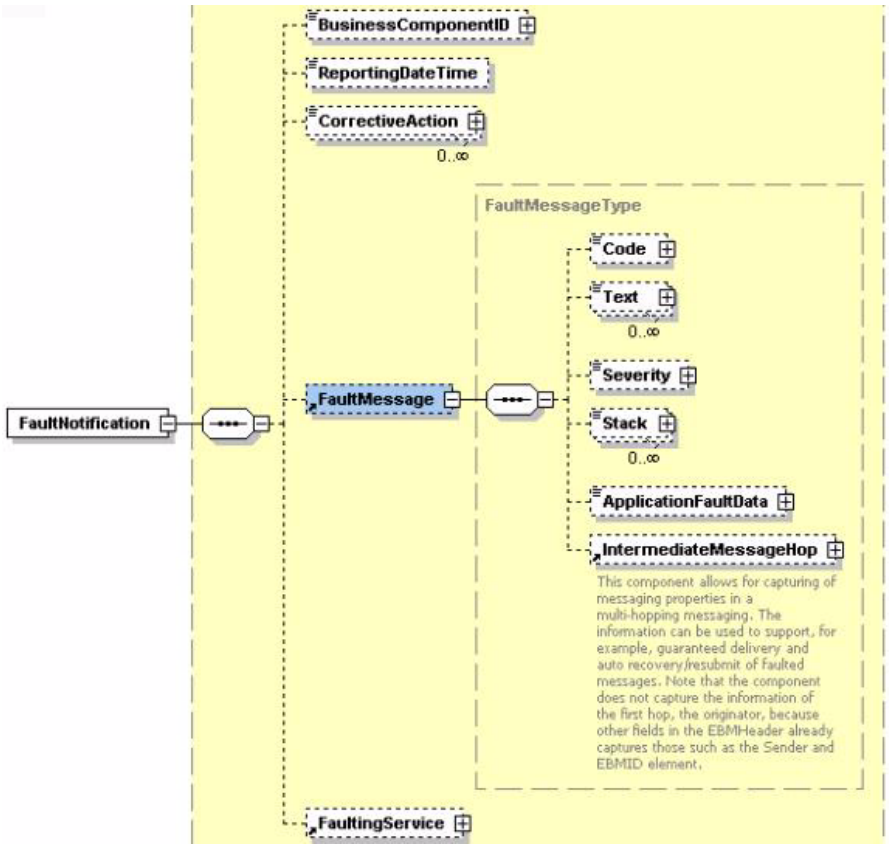


Table 26–4 FaultNotification Elements

Name	Purpose	Details
BusinessComponentID	Unique key for the application.	Provides an agnostic representation of the object instance.
ReportingDateTime	Provides the date and time at which the service faulted.	The date and time at which the service faulted.
CorrectiveAction	Provides the possible corrective action for the fault.	The corrective action for the fault.
FaultMessage	Provides details of the actual fault message.	For more information see Section 26.7.3.1, "FaultMessage Element."
FaultingService	Provides details of the faulting service.	For more information see Section 26.7.3.1, "FaultMessage Element."

26.7.3.1 FaultMessage Element

Table 26–5 discusses fault message elements.

Table 26–5 FaultMessage Elements

Name	Purpose	Details
Code	Provides the error code.	This is the fault code that was received.
Text	Provides error details.	This describes the details of the fault.
Severity	Provides the severity of the error.	This is the severity of the fault expressed as an integer.
Stack	Provides the error stack.	This is the complete fault stack.
ApplicationFaultData	Enables the fault message to be extended to accept any kind of XML input.	Enables a fault message to be extended to include any kind of XML input, as decided by the implementation scenario. For more information about extending fault messages, see Section 26.8, "Extending Fault Messages."
IntermediateMessageHop	Captures properties specific to a message in a multi-hop transaction.	Properties that are captured here can be used to support use cases implementing guaranteed message delivery and message recovery. For more information about implementing guaranteed message delivery and message recovery, see Section 26.5, "Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery."

26.7.3.2 IntermediateMessageHop Elements

Figure 26–10 illustrates `IntermediateMessageHop` elements. Intermediate message hop elements are discussed in Table 26–6.

Figure 26–10 IntermediateMessageHop Elements

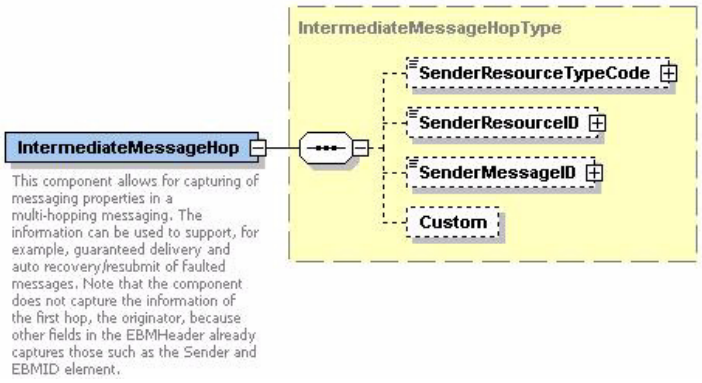


Table 26–6 IntermediateMessageHop Elements

Name	Purpose	Details
SenderResourceTypeCode	Used for storing the type of resource or system that is the sender of this message in the multi-hopping messaging layer.	Example values of this element are Queue, Topic, or Resequence Store.
SenderResourceID	Provides identification of the resource or system associated with the SenderResourceTypeCode.	Identification of the resource or system associated with the SenderResourceTypeCode.
SenderMessageID	Provides message identification in the context of the resource or system associated with the SenderResourceTypeCode.	Message identification in the context of the resource or system associated with the SenderResourceTypeCode.

26.7.3.3 FaultingService Element

Table 26–7 FaultingService Element

Name	Purpose	Details
ID	Provides the date and time at which the service faulted.	This is the name of the faulting service.
ImplementationCode	Provides the possible corrective action for the fault.	This is a string describing the type of service that faulted. Possible values are BPEL, JAVA, and OTHER.
InstanceID	Provides the details of the actual fault message.	This is the instance ID of the faulted service. If the service is a BPEL process, this is the BPEL instance ID.
ExecutionContextID	Provides the value for the ECID.	This is an ID generated for a group of service invocations/executions.

26.8 Extending Fault Messages

This section includes the following topics:

- Section 26.8.1, "Introduction to Extending Fault Messages"
- Section 26.8.2, "Extending a Fault Message"

26.8.1 Introduction to Extending Fault Messages

When an error occurs within an integration flow, within a Mediator service or BPEL process, the Error Handling framework captures the error within a fault message. The fault message is made available in the error details within the Oracle BPM Worklist.

For information about using the Oracle BPM Worklist with Oracle AIA error handling, see "Using the Oracle BPM Worklist" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

Fault message content is defined by the FaultType message schema definition in Meta.xsd, which is located in \EnterpriseObjectLibrary\Infrastructure\V1\Meta.xsd. If your fault message requirements are not met by the default elements of the schema, you can use the ApplicationFaultMessage element included in the schema to extend the scope of the fault details captured in the message.

For more information about the fault message schema, see Section 26.7, "Describing the Oracle AIA Fault Message Schema."

Extending fault details can add functionally rich information to the fault message to help the integration flow consumer better understand the context of the fault, leading to more effective error resolution. These additional fault details can be used to enable extended error handling functionality as well.

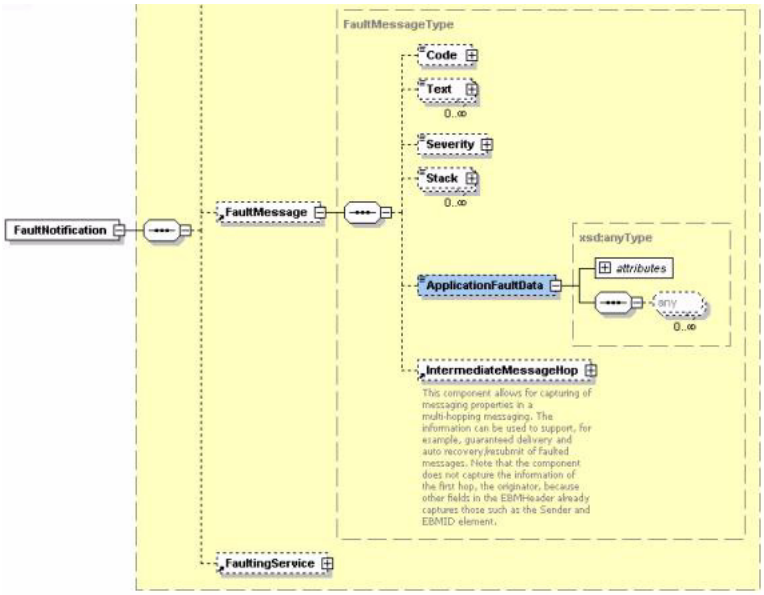
For more information about extending error handling, see Section 26.9, "Extending Error Handling."

For example, you can enrich the fault message with Order Number and Fulfillment System values, which are required to perform extended error handling tasks that update Order tables and create new service requests in an Order Fallout Management application.

26.8.2 Extending a Fault Message

Extending a fault message uses the `ApplicationFaultData` element, highlighted in Figure 26–11, of type xsd:anyType in the FaultType message schema definition in Meta.xsd.

Figure 26–11 ApplicationFaultData Element Highlighted in Meta.xsd



The ApplicationFaultData element is populated by a fault extension handler that you will configure to be invoked by the Error Handling Framework at run time in the case of BPEL faults.

The input to the fault extension handler is the default fault message. The fault extension handler enriches the fault message with additional content defined by the ApplicationFaultData element. Control of the enriched fault message is passed from the fault extension handler to the Error Handling Framework, which then passes the fault message on to the Oracle AIA common error handler for further processing.

To extend a fault message:

1. Create a fault extension handler that will be invoked to enrich the fault message.
2. In the **Error Extension Handler** field on the Error Notifications page, enter the name of the error extension handler that will be invoked to extend the fault message. For example, enter *ORDERFOEH_EXT* for an Order Fallout error extension handler.

   Based on the combination of error code, system code, service name, and process name parameters, the Error Handling framework checks to determine whether the error extension handler has a nondefault parameter defined.

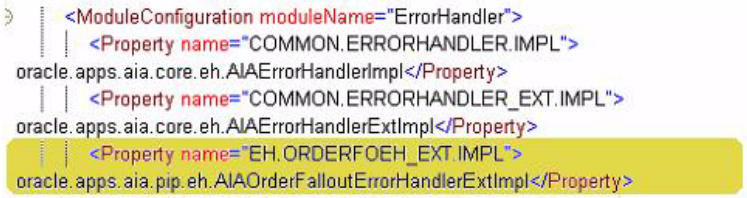
   If so, the framework locates the full classpath for the parameter in the AIAConfigurationProperties.xml file and makes a call out to that handler with the base fault message as input.

   Within this error extension handler, the fault message will be enriched to accommodate custom content. It will then be sent back to the Error Handling framework for further processing.

   For more information about the Error Notifications page, see "How to Set Up AIA Error Handling Configuration Details" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack*.
3. Access the AIAConfigurationProperties.xml file in $AIA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/config. Define

a property name that matches the error extension handler name that you defined in step 2, as shown in Figure 26–12. The value for this property is the fully qualified class path of the handler.

Figure 26–12 Example Error Extension Handler Property and Value in AIAConfigurationProperties.xml



It is through this class that the extension; Order Number and Fulfillment System values, for example; are added to the fault message using `xsd:anyType` in the ApplicationFaultData element in Meta.xsd.

4. Implement the IAIAErrorHandlerExtension interface in your error extension handler class registered in AIAConfigurationProperties.xml. Implement these methods:

   - `handleCompositeSystemError` for BPEL system errors
   - `handleBusinessError` for BPEL custom errors

   Example 26–18 and Example 26–19 illustrate the interface structure.

Example 26–18 IAIAErrorHandler Interface Class

```
public interface IAIAErrorHandler
{
  /**
    *
    * @param ebmHeader
    * @param faultMessage
    */
  public void logErrorMessage(Element ebmHeader, String faultMessage);

  /**
    *
    * @param XMLELfaultMessage
    */
  public void logErrorMessage(XMLElement XMLELfaultMessage);

  /**
    *
    * @param ebmHeader
    * @param faultMessage
    */
  public void logErrorMessage(Node ebmHeader, String faultMessage);

  /**
    * @since FP 2.3
    * @param faultMessage
    * @param jmsCorrelationID
    */
  public void sendNotification(String faultMessage, String jmsCorrelationID,
                                              HashMap compositeDetailsHM);
}
```

Example 26–19 IAIAErrorHandlerExtension Interface Class

```
package oracle.apps.aia.core.eh;
public interface IAIAErrorHandlerExtension
{
  /**
   *
   * @param iFaultRecoveryContext
   * @param faultMessageConstructed
   * @param componentType
   * @return
   */
  public String handleCompositeSystemError(IFaultRecoveryContext
iFaultRecoveryContext,
                                           String faultMessageConstructed,
                                           String componentType);


  /**
   *
   * @param faultMessageConstructed
   * @return
   */
  public String handleBusinessError(String faultMessageConstructed);
}
```

You can view extended field values in the error logs accessed in Oracle Enterprise Manager.

For more information about viewing error logs in Oracle Enterprise Manager, see "Using Trace and Error Logs" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

You should also be able to view them in email notifications if they have been configured appropriately.

For more information about customizing error notifications, see "Customizing Error Notification Emails" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

26.9 Extending Error Handling

This section includes the following topics:

- Section 26.9.1, "Introduction to Extending Error Handling"
- Section 26.9.2, "Implementing an Error Handling Extension"

This section provides an overview of error handling extension and discusses how to implement an error handling extension.

26.9.1 Introduction to Extending Error Handling

The default error handling behavior for BPEL and Mediator errors is to route fault messages to the Oracle AIA common error handler, which logs the error and delivers the fault messages to the Oracle AIA error topic. The default Oracle AIA error listener

subscribes to this Oracle AIA error topic, picks up the fault message, and calls the error notification process, which issues a notification to the Oracle BPM Worklist if configured to do so.

For information about using the Oracle BPM Worklist with Oracle AIA error handling, see "Using the Oracle BPM Worklist" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

You can extend the Error Handling Framework to perform actions beyond these default behaviors.

For example, you may want a particular error to trigger default error-handling behavior, in addition to extended error handling behavior, such as updating a table and creating a new request in an application.

To implement an error-handling extension, extend fault messages to provide additional values.

For more information about extending fault messages, see Section 26.8, "Extending Fault Messages."

26.9.2 Implementing an Error Handling Extension

To implement an error handling extension:

1. On the Error Notifications page, enter an **Error Type** field value for an error code, system code, process name, and service name value combination.

   For more information about the Error Notifications page, see "How to Set Up AIA Error Handling Configuration Details" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

   The Error Handling Framework uses the Error Type value to stamp the JMSCorrelationID JMS header. The JMSCorrelationID is used by the custom error listener to identify fault messages that require its custom error handling.

2. Implement an error extension listener to subscribe to the Oracle AIA error topic.

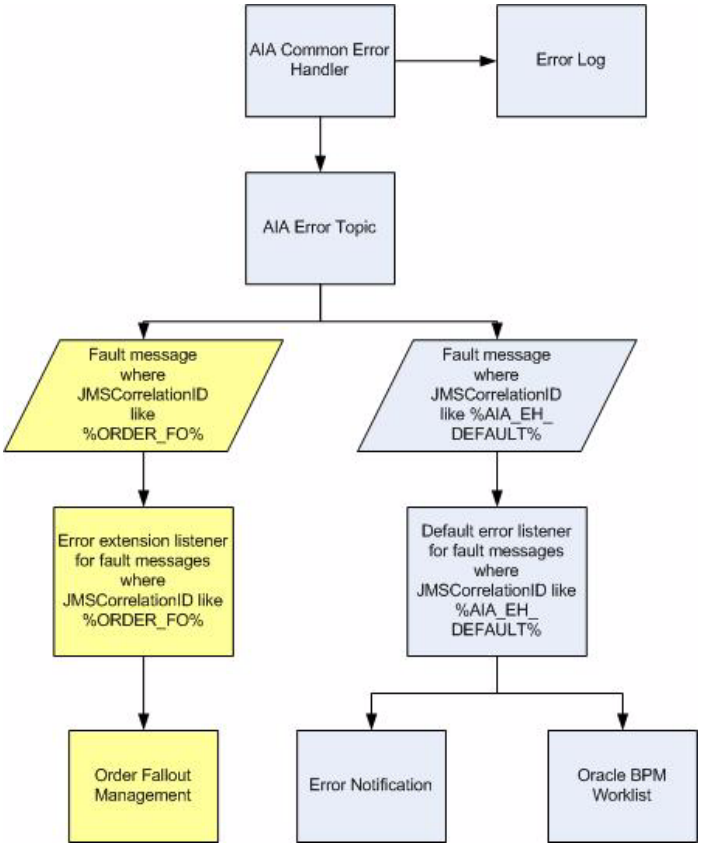
   Configure an error extension listener to filter fault messages based on the JMS Header - JMSCorrelationID value defined by the error type you created in step 1.

For example, you can create an Order Fallout error extension listener that picks up fault messages with the JMSCorrelationID value of **ORDER_FO**. For this example the Error Type field value defined on the Error Notifications page in step 1 should be ORDER_FO.

The Order Fallout error extension listener can then extract values from the fault message, extended to include Order Number and Fulfillment System values, for example. The error extension listener can then pass those values to an Order Fallout Management application, for example, which can use those values to update Order tables and create new service requests.

The extended error handling flow is illustrated in Figure 26–13.

Figure 26–13 Sample Extended Error Handling Flow Alongside a Default Error Handling Flow



26.10 Configuring Oracle AIA Processes for Trace Logging

This section includes the following topics:

- Section 26.10.1, "Describing Details of the isTraceLoggingEnabled Custom XPath Function"
- Section 26.10.2, "Describing Details of the logTraceMessage Custom XPath Function"
- Section 26.10.3, "Describing the Trace Logging Java API"

These custom XPath trace logging functions are available to BPEL and Mediator services operating in an Oracle AIA ecosystem.

- `aia:isTraceLoggingEnabled(String logLevel, String processName)`

  Determines whether trace logging is enabled for the service or at the overall system level.

- `aia:logTraceMessage(String level, Element ebmHeader, String message)`

  Generates the actual trace log.

When developing a BPEL or Mediator process, always call the `aia:isTraceLoggingEnabled()` function first. If it returns a **true** result, then have the process call the `aia:logTraceMessage()` function.

These log files are stored in the <aia.home>/logs/ directory.

In addition to these custom XPath functions, a Java API is also available so that any application developer can use it to log trace messages.

For more information about using trace logs, see "Using Trace and Error Logs" in *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

26.10.1 Describing Details of the isTraceLoggingEnabled Custom XPath Function

The `isLoggingEnabled` custom XPath function is a utility function that returns a Boolean result. The function signature is: `aia:isTraceLoggingEnabled (String logLevel, String processName)`

These are the parameter details:

- `logLevel`

  Possible values include:

  - Severe
  - Warning
  - Info
  - Config
  - Fine
  - Finer
  - Finest

- `processName`

  Name of the Oracle AIA service using this function.

26.10.2 Describing Details of the logTraceMessage Custom XPath Function

The `logTraceMessage` custom XPath function generates a trace message, which contains the details of the message to be included in the trace log.

This function accepts the EBM header and the verbose logging message as parameters. Various elements from the EBM header are used to populate supplemental attributes to the log message. If the EBM header is not passed, these supplemental attributes are set as empty strings.

The function signature is `aia:logTraceMessage (String level, Element ebmHeader, String message)`. These are the parameter details:

- `level`

  Possible values include:

  - Severe
  - Warning
  - Info
  - Config
  - Fine
  - Finer

  - Finest
- `ebmHeader`

  EBM header.
- message

  Verbose text message to be logged.

26.10.3 Describing the Trace Logging Java API

In addition to the `isTraceLoggingEnabled` and logTraceMessage custom XPath functions, a trace Logging Java API is also available so that any application developer can log trace messages. These functions are available through the trace logging Java API.

One of the function signatures is `AIALogger.isTraceLoggingEnabled (String logLevel, String processName)`. This function determines whether trace logging is enabled for the service or at the overall system level. These are the parameter details:

- `logLevel`

  Possible values include:
  - Severe
  - Warning
  - Info
  - Config
  - Fine
  - Finer
  - Finest
- `processName`

  Name of the Oracle AIA service using this function.

Another function signature is `AIALogger.logTraceMessage (String level, Element ebmHeader, String message)`. This function generates the actual trace log. These are the parameter details:

- `level`

  Possible values include:
  - Severe
  - Warning
  - Info
  - Config
  - Fine
  - Finer
  - Finest
- `ebmHeader`

  EBM header.

- `message`

  Verbose text message to be logged.

27

Working with AIA Design Patterns

This chapter provides an overview of AIA message processing patterns, AIA assets centralization patterns, AIA assets extensibility patterns.

This chapter includes the following sections:

- Section 27.1, "AIA Message Processing Patterns"
- Section 27.2, "AIA Assets Centralization Patterns"
- Section 27.3, "AIA Assets Extensibility Patterns"

27.1 AIA Message Processing Patterns

This section discusses AIA message processing patterns and provides solutions to specific problems that you may encounter.

This section includes the following topics:

- Section 27.1.1, "Synchronous Request-Reply Pattern: How to get Synchronous Response in AIA"
- Section 27.1.2, "Asynchronous Fire-and-Forget Pattern"
- Section 27.1.3, "Guaranteed Delivery Pattern: How to Ensure Guaranteed Delivery in AIA"
- Section 27.1.4, "Service Routing Pattern: How to Route the Messages to Appropriate Service Provider in AIA"
- Section 27.1.5, "Competing Consumers Pattern: How are Multiple Consumers used to Improve Parallelism and Efficiency?"
- Section 27.1.6, "Asynchronous Delayed-Response Pattern: How does the Service Provider Communicate with the Requester when Synchronous Communication is not Feasible?"
- Section 27.1.7, "Asynchronous Request Response Pattern: How does the Service Provider Notify the Requester Regarding the Errors?"

27.1.1 Synchronous Request-Reply Pattern: How to get Synchronous Response in AIA

Problem

Many use cases warrant consumers to send requests synchronously to service providers and get immediate responses to each of their requests. These use cases need the consumers to wait until the responses are received before proceeding to their next tasks.

For example, Customer Relationship Management (CRM) applications may provide features such as allowing customer service representatives and systems to send requests to providers for performing tasks such as account balance retrieval, credit check, ATP (advanced time to promise) calculation, and so on. Since CRM applications expect the responses to be used in the subsequent tasks, this precludes the users from performing other tasks until the responses are received.

Solution

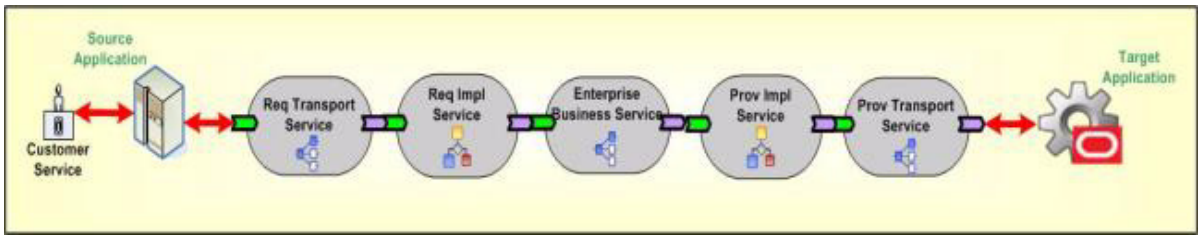
AIA recommends using the synchronous request and response service interface in all the composites involved in processing or connecting the back-end systems. There should not be any singleton service in the services involved in the query processing. The general recommendation is for all of the intermediary services and the service exposed by the provider application to implement a request-response based interface - a two-way operation. Even though it is technically possible to design all services but the initial caller to implement two one-way requests, this implementation technique should be avoided if possible.

Implementation should strive to ensure that no persistence of state information (dehydration) or audit details is done by any of the intermediary services or by the ultimate service provider. These techniques help keep the latency as low as possible.

AIA also recommends that the intermediary services are co-located to eliminate the overheads normally caused by marshalling and un-marshalling of SOAP documents.

Figure 27–1 illustrates how a task implemented as an Enterprise Business Service (EBS) is invoked synchronously by the requester application.

Figure 27–1 Invoking an EBS Synchronously by the Requester Application



Impact

- All the resources are locked in until the response from service provider goes back to the originating system or user.
- Either a transaction timeout or an increased latency may result if any of the services or the participating application takes more time for processing.
- Service providers must be always available and ready to fulfill the service requests.
- Service providers doing real-time retrieval and collation of data from multiple back-end systems to generate a response message could put an enormous toll on the overall resources and increase the latency. The synchronous request-response design pattern should not be used to implement tasks that involve real-time complex processing. Off loading of work must be done; the design must be modified to accommodate the staging of quasi-prepared data so that the real-time processing can be made as light as possible.
- Services involved in implementing synchronous request-response pattern should refrain themselves from doing work for each of the repeatable child nodes present

in the request message. A high number of child nodes in the payload in a production environment can have an adverse impact on the system.

27.1.2 Asynchronous Fire-and-Forget Pattern

Problem

The requester application should be able to continue its processing after submitting a request regardless of whether the service providers needed to perform the tasks are immediately available or not. Besides that, the user initiating the request does not have a functional need to wait until the request is fully processed. The bottom line is, the service requesters and service providers must have the capability to produce and consume messages at their own pace without directly depending upon each other to be present.

Also, the composite business processes should be able to support the infrastructure services like error handling and business activity monitoring services in a decoupled fashion without depending on the participating application or the AIA functional process flows.

For example, order capture applications should be able to keep taking orders regardless of whether the back end systems such as order fulfillment systems are available at that time. You do not want the order capturing capability to be stopped because of non-availability of any of the services or applications participating in the execution of order fulfillment process.

Solution

AIA recommends the fire-and-forget pattern using queuing technology with a database or file as a persistence store to decouple the participating application from the integration layer. The queue acts as a staging area allowing the requester applications to place the request messages. The request messages are subsequently forwarded to service providers on behalf of requester as and when the service providers are ready to process them.

It is highly recommended that the enqueueing of the request message into the queue is within the same transaction initiated by the requester application to perform its work. This ensures that the request message is enqueued into the queue only when the participating application's transaction is successful. The request message is not enqueued in situations where the transaction is not successful. Care must be taken to ensure that the services residing between two consecutive milestones are enlisting themselves into a single global transaction.

Refer to Section 26.5, "Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery" to understand how milestones are used as intermediate reliability artifacts to ensure the guaranteed delivery of messages.

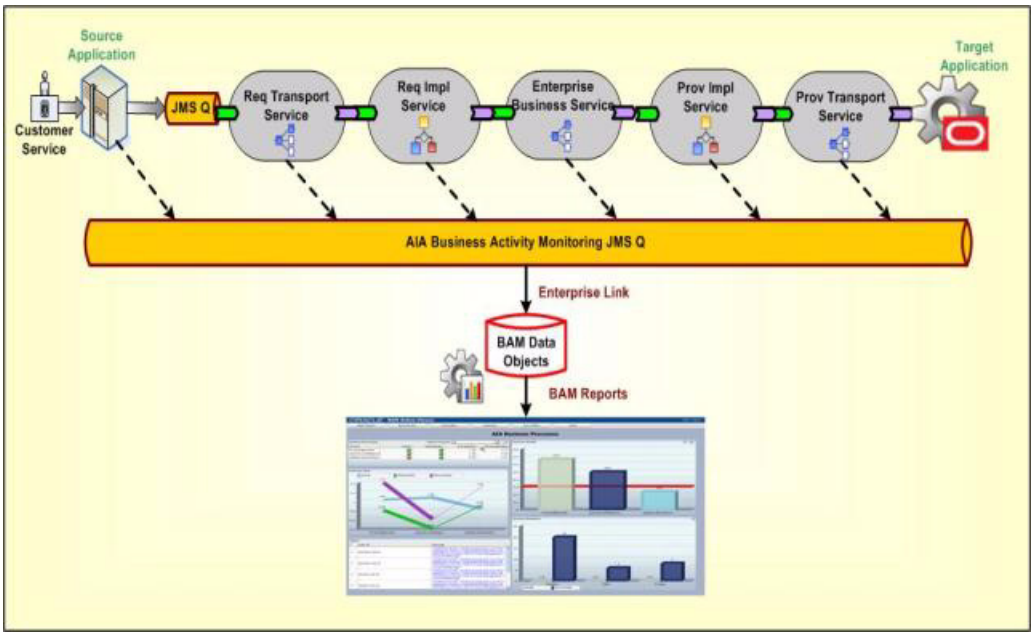
AIA recommends having a queue per business object. All requests emanating from a requester application for a business object use the same queue.

Figure 27–2 shows how JMS queues are used to decouple the source application from the integration layer processing. The source application submits the message to the JMS queue and continues with its other processing without waiting for a response from the integration layer.

The AIA composite business process may not be up and running while the application continues to produce the messages in the JMS queue. When the AIA composite business process comes up, it starts consuming the messages and processes them.

Figure 27–2 also shows how the monitoring services can be decoupled from the main AIA functional flows with the help of the JMS queue. All the AIA services and the participating applications can fire a monitoring message into the queue and forget about how it is processed by the downstream applications. In this pattern the services or participating applications do not expect any response. Similarly, the AIA infrastructure services capture the system or business errors from the composite business processes and publish them to an AIA error topic, which is used by the error handling framework for further processing (resubmission, notification, logging, and so on).

Figure 27–2 Fire-and-Forget Pattern Using Queuing Technology



Impact

The default implementation does not have inherent support for notifying the requester application of success or the failure of messages. Even though the middleware systems provide the ability to monitor and administer the flow of in-flight message transmissions, there will be use cases where requester applications either want to be notified or have a logical reversal of work done programmatically.

For more information about how compensation handlers can be implemented for this message exchange pattern, see Chapter 14, "Designing Application Business Connector Services" and Chapter 15, "Constructing the ABCS".

27.1.3 Guaranteed Delivery Pattern: How to Ensure Guaranteed Delivery in AIA

Problem

Message delivery cannot be guaranteed when the participating systems and the services integrating those systems interact in an unreliable fashion.

For example, the Order Capture system might submit an order fulfillment request that triggers a business process, which in turn invokes the services provided by multiple disparate systems to fulfill the request. Expecting all of the service providers to be always available might be unrealistic.

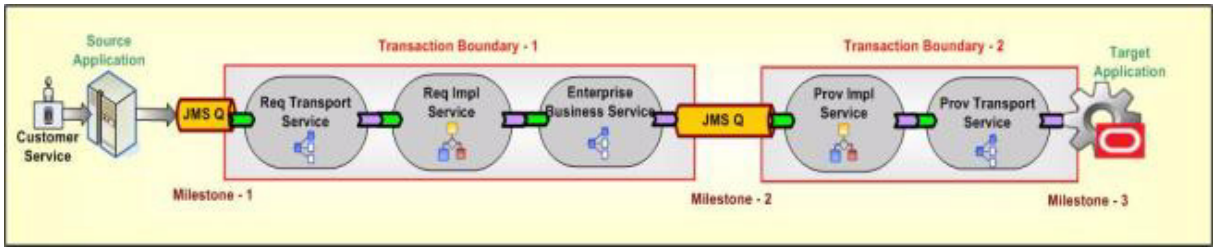
Solution

A **milestone** is a message checkpoint where the enriched message is preserved after completing a certain amount of business functionality at this logical point or is submitted to the participating application for further processing of that message in an end-to-end composite business process. A milestone helps to commit the business transactions done to that logical point and also releases the resources used in the AIA services and the participating application.

AIA recommends introducing milestones at appropriate logical points in an end-to-end integration having the composite business processes, business activity services, or data services designed and implemented for integrating disparate systems to address a business requirement. Queues with database or file as persistence store are used to implement milestones.

Figure 27–3 shows how to ensure guaranteed delivery with the help of milestones for a simple use case where the customer service representative updates the billing profile on the source application and the updated profile is reflected in the back-end billing systems.

Figure 27–3 Ensuring Guaranteed Delivery Using Milestones



AIA recommends that the source application push the billing profile information to Milestone -1 (JMS Queue) in a transactional mode to ensure the guaranteed delivery of the message. This transaction is outside the AIA transaction boundaries and is not shown in the diagram. After the billing profile information is stored in Milestone-1, the source application resources can be released because AIA ensures the guaranteed delivery of that message to the target application.

The implementation service has the message enrichment logic (if needed) and the Enterprise Business Service (EBS) routes the enriched message to appropriate providers or Milestone-2 (optional for very simple and straightforward use cases such as no enrichment or processing requirements for implementation service).

All the enriched messages would either be transferred to Milestone-2 or rolled back to Milestone - 1 for corrective action or resubmission. To achieve this, all the services present between Milestone-1 and Milestone-2 must enlist themselves in a single global transaction. After the messages are transferred to Milestone-2, all the Milestone-1 implementation service instances are released to accommodate new requests.

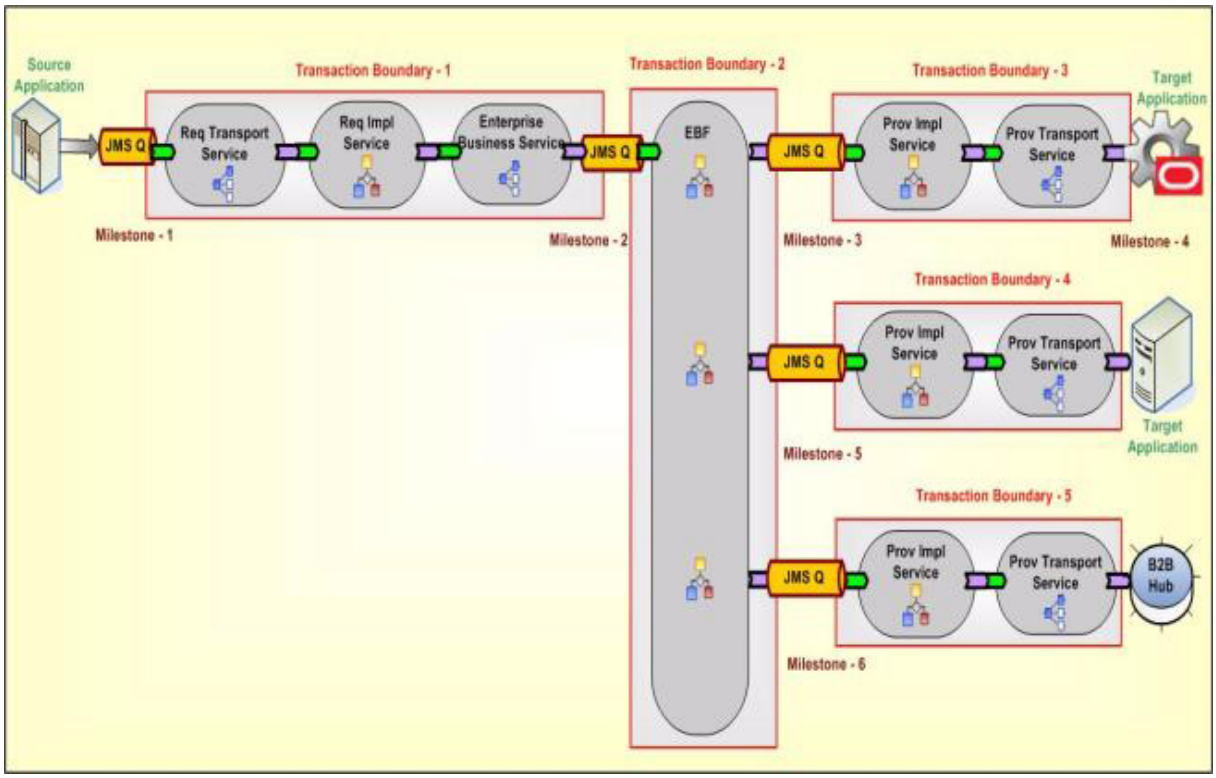
The provider implementation service picks up the billing profile information and ensures the message is delivered to the target application. The target application may or may not have the capability of participating in the transaction between Milestone-2 and Milestone-3 so the implementation service should be designed to take care of the compensatory transactions.

AIA recommends that the target application build the XA capability to ensure guaranteed delivery. If the target application does not have the XA capability, the implementation service should be designed for manual compensations or

semi-automatic or automatic compensations based on the target system's capability to acknowledge the business message delivered from AIA.

Figure 27–4 shows the usage of milestones to ensure guaranteed delivery for a complex use case where the source system submits an order provisioning request to AIA.

Figure 27–4 Using Milestones to Ensure Guaranteed Delivery in a Complex Use Case



Impact

Introducing a milestone adds processing overhead and could impact the performance of composite business processes, business activity services, or data services. So selecting the number of milestones is a challenge for the designers. However, minimizing the milestones, thereby adding more work to a single transaction, could also have a detrimental effect.

Ensuring reliable messaging between the "milestones" and the "application and milestone" to achieve the guaranteed delivery adds transactional overheads. This could impact the performance at run time in some situations (if the amount of work to be done between two milestones is too large) and may also require additional process design requirements for compensation.

27.1.4 Service Routing Pattern: How to Route the Messages to Appropriate Service Provider in AIA

Problem

Requester applications should build tightly coupled services if they must send information to a specific target system. The logic to identify the service provider, and in some cases the service provider instance (when you have multiple application

instances for the same provider) has to be embedded within the logic of caller service. The decision logic in the caller service has to be constantly undergoing changes as and when either a new service provider is added into the mix or an existing service provider is retired from that mix.

For example, a customer might have three billing system implementations-two instances of vendor A's billing system, one dedicated to customers who reside in North America and the second dedicated to the customers residing in Asia-Pacific; one instance of vendor B's billing system dedicated to customers residing in other parts of the world. The requester service must have the decision logic to discern to whom to delegate the request.

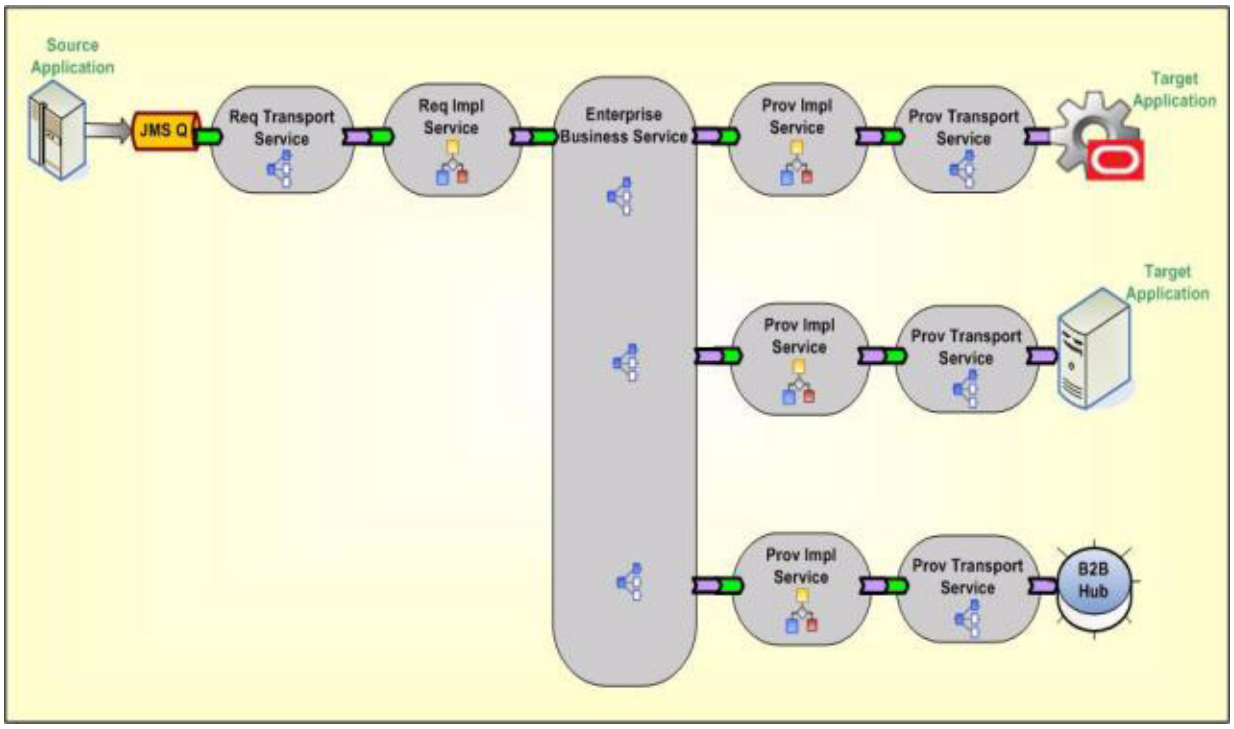
Solution

AIA recommends content-based service routing to appropriate target service implementations to further process this message and send it to the target applications. AIA recommends externalizing the decision logic to determine the right service provider from the actual requester application or service. This decision logic is incorporated in the routing service. This allows for declaratively changing the message path when unforeseen conditions occur. The Enterprise Business Service acts as a routing service. This facilitates one-to-many message routing based on the content-based filtering. A message originating from a requester can go to all the targets, some targets, or just one target based on the business rules or content filters.

The routing service evaluates the rule and deciphers the actual service provider that must be used for processing a specific message. Using this approach, the clients or service requesters are totally unaware of the actual service providers. Similarly, the underlying service providers are oblivious to the client applications that made the request. This enables you to introduce new providers and retire existing providers without making any changes to the actual requester service.

Figure 27–5 illustrates how a request sent by Req Impl Service (requester service) is sent to either one of two target applications or to a trading partner (B2B) using context based routing by routing service.

Figure 27–5 Content-Based Service Instance Routing



Impact

For the requester service to be loosely coupled with the actual service providers, the routing service should act as the abstraction layer; therefore, its service interface has to be carefully designed. Its interface can emulate the actual service provider's interface if only one unique service provider exists or it can have a canonical interface.

Even though Mediator technologies allow for routing rules to be added in a declarative way, validation of different routing paths must be tested before deploying them in production.

Managing the decision logic locally in each of the routing services can lead to duplication and conflict so having them managed in a centralized rule management system such as Oracle Business Rules Engine is an option to be considered.

27.1.5 Competing Consumers Pattern: How are Multiple Consumers used to Improve Parallelism and Efficiency?

Problem

When the requester application produces a huge number of business messages which should be handled and sent to the target application, consuming all the messages to meet the expected throughput is a challenge. This becomes much more profound when the processing of each message involves interacting with a multitude of systems thereby resulting in blocking for a significant period waiting for them to complete their work.

For example, the order capture application submits orders in the order processing queue for the fulfillment of orders. Each of the order fulfillment requests must be picked from the queue and handed over to the order orchestration engine for decomposition. Only upon the acknowledgment can the next message be picked up from the order processing queue. Unplanned outages of the order orchestration

platform can result in a large accumulation of order fulfillment requests and serial processing of these requests upon availability of platform can cause an inordinate amount of delay.

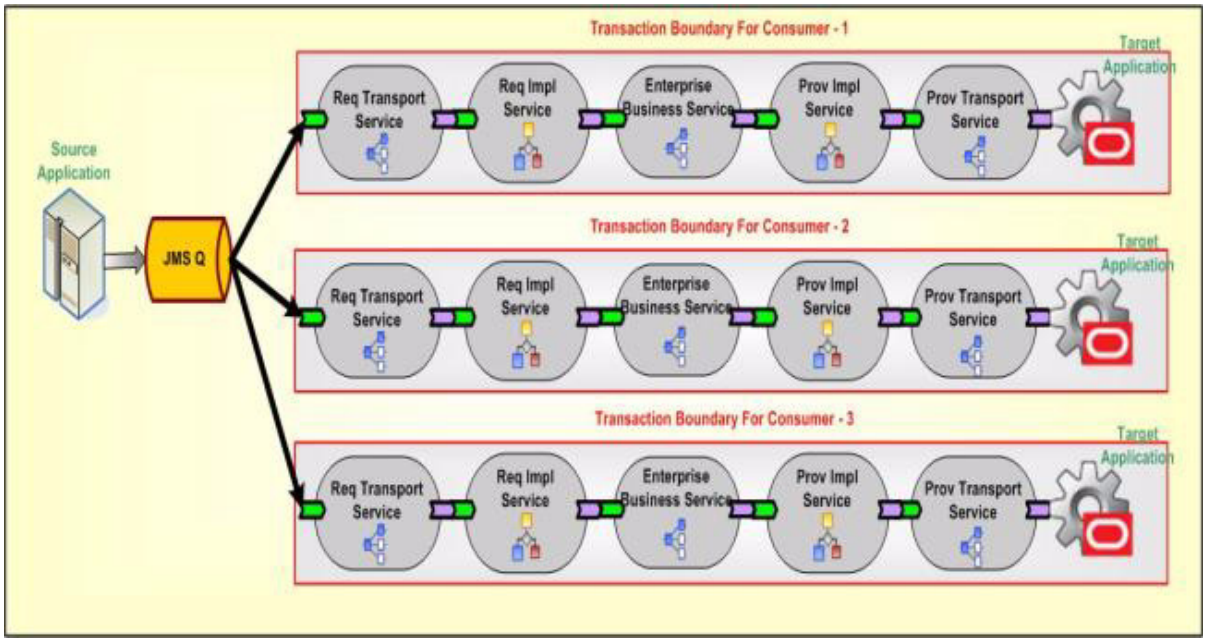
Solution

AIA recommends setting up multiple consumers or listeners connected to the source queue. AIA assumes that the requester application has published messages into the source queue.

Refer to Section 27.1.2, "Asynchronous Fire-and-Forget Pattern" for more details.

Figure 27–6 shows that setting up multiple listeners triggers consumption of multiple messages in parallel. Even though multiple consumers are created to receive the message from a single queue (Point-to-Point Channel), only one of the consumers can receive the message upon the delivery by JMS. Even though all of the consumers compete with each other to be the receiver, only one of them ends up receiving the message. Based on the business requirements, this pattern can be used across the nodes with each consumer set up on each node in an HA environment, or on a cluster to improve the scalability, or within the AIA artifact.

Figure 27–6 Setting Up Multiple Consumers or Listeners Connected to the Source Queue



Impact

This solution only works for the queue and cannot be used with the Topic (Publish-Subscribe channel). In the case of Topic, each of the consumers receives a copy of the message.

Since multiple consumers are processing the messages in parallel, the messages are not processed in a specific order. If they must be processed in sequence without compromising parallelism and efficiency for a functional reason, then you must introduce a staging area to hold these messages until a contiguous sequence is received before delivering them to the ultimate receivers.

27.1.6 Asynchronous Delayed-Response Pattern: How does the Service Provider Communicate with the Requester when Synchronous Communication is not Feasible?

Problem

When a service provider has to take a large amount of time to process a request, how can the requester receive the outcome without keeping the requester in a suspended mode during the entire processing?

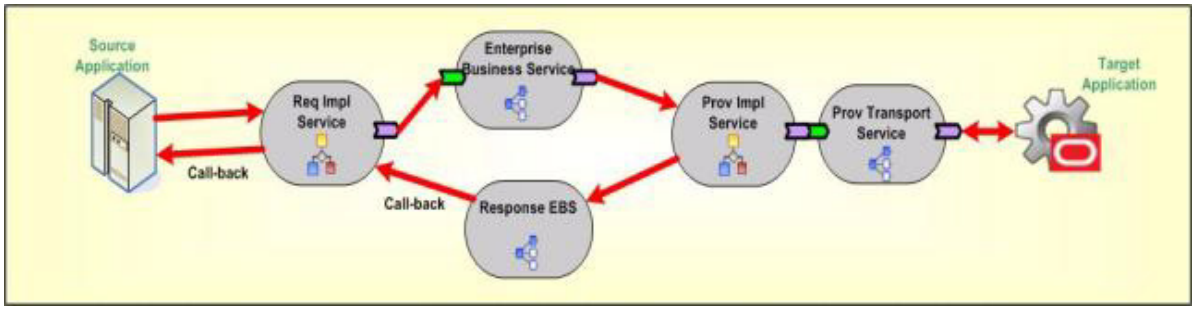
For example, the order capture application submits a request for fulfillment of an order. The order fulfillment process itself could take anywhere from several minutes to several days depending upon the complexity of tasks involved. Even though the order capture application would like to know the outcome of the fulfillment request, it cannot afford to wait idly until the process is completed to know the outcome.

Solution

AIA recommends using the asynchronous delayed response pattern where the requester submits a request to the integration layer and sets up a callback response address by populating the metadata section in the request message (WSA Headers). The callback response address points to the end point location of the original caller's service that is contacted by service provider after the completion. Since multiple intermediary services are involved before sending the message to the ultimate receiver, sending the callback address using the metadata section in the message is needed. In addition to the callback address, the requester is expected to send correlation details that allow the service provider to send this information in future conversations so that the requesters are able to associate the response messages with the original requests.

Since a request-delayed response pattern is asynchronous in nature, the requester does wait for the response after sending the request message. In the flow implementing this pattern as shown in Figure 27–7, it is recommended that all of the services involved have two one-way operations. A separate thread must be invoked to initiate the response message. When the AIA provider service processes the message after getting a reply from the provider participating application, it creates a response thread by invoking the response EBS. The response EBS uses the WSA headers to identify the requesting service and processes the response to invoke the callback address given by the requesting application. The EBS has a pair of operations to support two one-way calls - one for sending the request and another for receiving the response. Both the operations are independent and atomic. A correlation mechanism is used to establish the application or caller's context.

Figure 27–7 Example of the Asynchronous Delayed Response Pattern



Impact

Provider applications must have the capability to persist the correlation details and the callback address sent by the requester application and need these details populated in the response message.

27.1.7 Asynchronous Request Response Pattern: How does the Service Provider Notify the Requester Regarding the Errors?

Problem

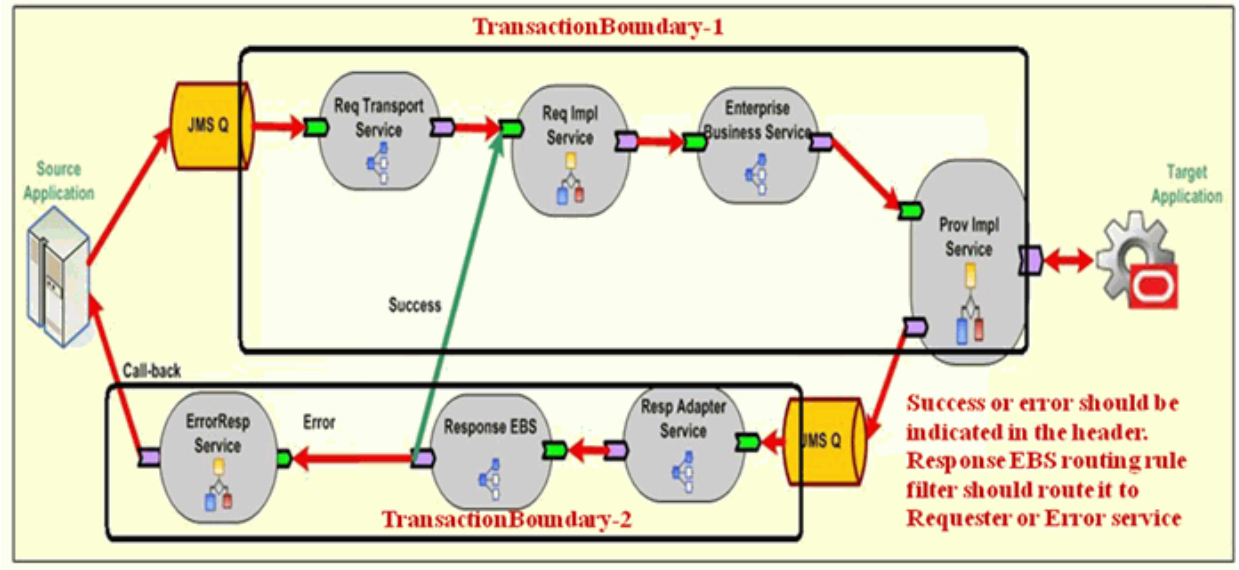
Integration architects should understand the transactional requirements, the participating applications' capability of participating in a global transaction, and the concept of milestones before deciding upon the asynchronous request-response pattern implementation. The following sections present implementation solution guidelines for three different scenarios.

Solution: Success scenario calls the same requester but separate service for the Error Handling

In this solution, the interaction is divided into two transactions; that is, the error handling part is separated from the requester ABCS implementation. A new service should be constructed to accept the error message from the response EBS and the error response service should forward it to the source participating application.

Figure 27–8 shows that if the request is processed successfully, then it is routed back to the requester ABCS to complete the remaining process activities. If there is any error while processing the response, it follows the normal error handling mechanism and a notification is sent back to the source participating application. If the error is sent back to the caller, then a web service call to the error response service should be made in the error handling block of the requester ABCS implementation.

Figure 27–8 Example of Interaction Divided into Two Transactions: Error Handling and Requester ABCS

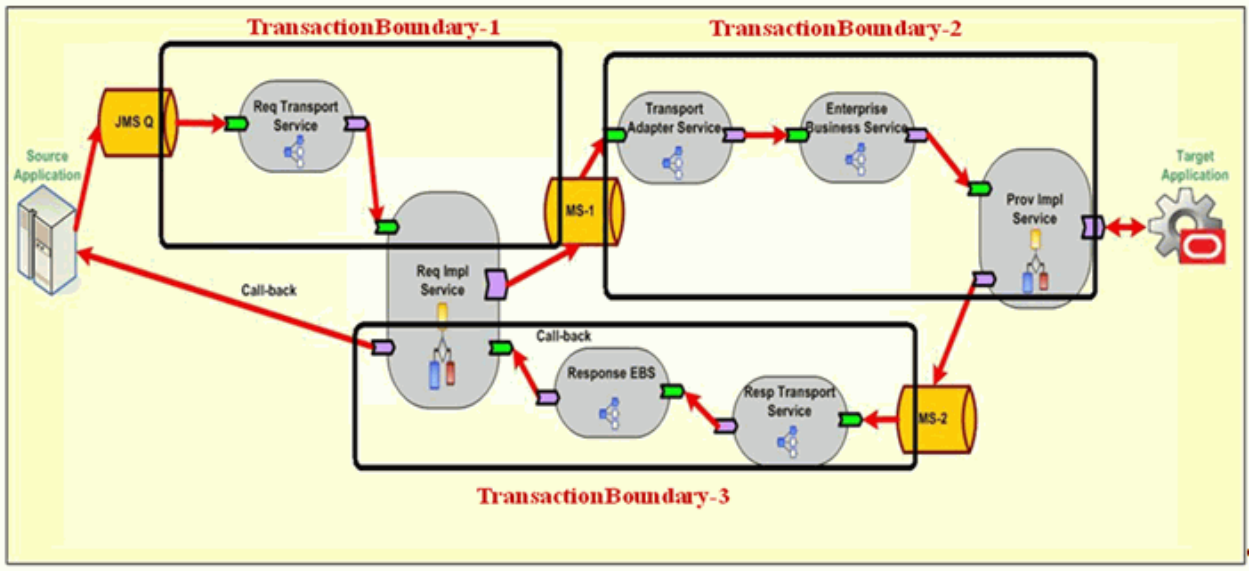


Solution: Using JMS Queue as a milestone

In this solution, the interaction is divided into three transactions. The JMS queue / milestone should be used to enable the transaction boundaries. Figure 27–9 depicts the

error scenario. The success scenario will not have the "JMS-2" queue. Instead the provider ABCS directly invokes the response EBS. So, for the success scenario, the whole interaction is completed in two transactions.

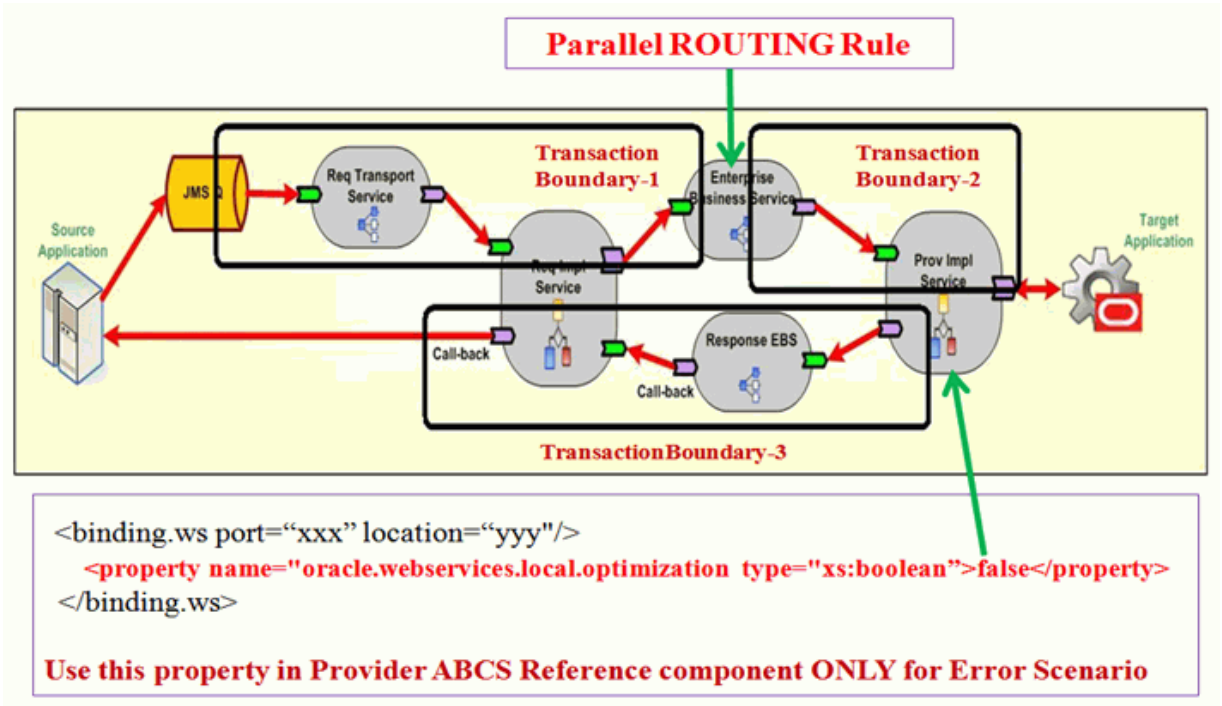
Figure 27–9 Error Scenario when Using JMS Queue as a Milestone



Solution: Using parallel routing rule and Web Service call

In this solution, the interaction is divided into three transactions. Figure 27–10 depicts the error scenario. The parallel routing rule in the mediator (EBS) and a web service call from the provider ABCS reference to response EBS should be used to enable the transaction boundaries. The success scenario directly calls the response EBS using an optimized reference component binding. So, there would be only two transaction boundaries for the success case.

Figure 27–10 Error Scenario Using Parallel Routing Rule and Web Service Call



27.2 AIA Assets Centralization Patterns

This section describes how to avoid redundant data model representation and service contract representation in AIA.

This section includes the following topics:

- Section 27.2.1, "How to Avoid Redundant Data Model Representation in AIA"
- Section 27.2.2, "How to Avoid Redundant Service Contracts Representation in AIA"

27.2.1 How to Avoid Redundant Data Model Representation in AIA

Problem

The service contracts between the applications and AIA interfaces must use similar business documents or data sets which results in the redundant data models representation. A change in the data model is very difficult to apply and it is very difficult to govern.

Solution

AIA recommends separation of the data model or schemas physically from the service contracts or the implementations. These schemas should be centralized at a location which could be referenced easily during design or run time.

All the EBOs (Enterprise Business Objects), ABOs (Application Business Objects), and any other utility schemas should be maintained at a central location in a structured way.

As the EBO schemas are used for various business processes in common, a thorough analysis should be done before coming up with a canonical data model.

Impact

- A change in the data model or schema should be treated very carefully because multiple dependent service contracts could be impacted with a small change.
- Versioning policies should be clearly defined, otherwise redundant service implementations for a minor change may occur.
- Governance is a challenge to manage the shared data models or common assets.

27.2.2 How to Avoid Redundant Service Contracts Representation in AIA

Problem

The granularity of the service contract can be at the operation level or at the entity level. The entity level definition can have various operations defined in the same service contract but the implementation can be at operation level which results in creating multiple implementation artifacts using the same service contract. This causes redundant service contracts representation in various implementations.

Solution

AIA recommends that the service contracts (WSDL) be separated physically from the implementations. These service contracts should be centralized at a location which can be referenced easily during design or run time.

All the EBS (Enterprise Business Service), ABCS (Application Business Connector Service), and any other utility service contracts should be maintained at a central location in a structured way.

Impact

- A change in the service contract should be treated very carefully because multiple dependent service implementations could be impacted with a small change.
- Governance is a challenge to manage the shared service contracts or common assets.

27.3 AIA Assets Extensibility Patterns

This section discusses the extensibility patterns for AIA assets and describes the problems and solutions for extending AIA artifacts.

This section includes the following topics:

27.3.1 Extending Existing Schemas in AIA

Problem

The delivered schemas may not be sufficient for some of your specific business operations, so you may want to add elements to the existing schemas in an upgrade safe manner. This wouldn't be possible by just adding the customer-specific elements in an existing schema, and the schema extension model is needed.

Solution

To allow you to extend the EBO or Enterprise Business Message (EBM) schemas in a nonintrusive manner, the complex type defined for every business component and the common component present in each of the EBO or EBM schemas has an extension placeholder element called *Custom* at the end. The data type for this custom element is defined in the schema modules allocated for holding customer extensions in the custom EBO schema files.

In Example 27–1 the custom element added at the end of the complex type in the "Schema-A" acts as a place holder to bring the customer extensions added in the "Custom Schema-A".

Here the Schema- A is "AIAComponents\EnterpriseObjectLibrary\Industry\Telco\EBO\CustomerParty\V2\CustomerPartyEBO.xsd"

Example 27–1 Custom Element Acting as Placeholder

```
<xsd:complexType name="CustomerPartyAccountContactCreditCardType">
   <xsd:sequence>
   <xsd:element ref="corecom:Identification" minOccurs="0"/>
   <xsd:element ref="corecom:CreditCard" minOccurs="0"/>

<xsd:element name="Custom"
type="corecustomerpartycust:CustomCustomerPartyAccountContactCreditCardType"
minOccurs="0"/>

    </xsd:sequence>
    <xsd:attribute name="actionCode" type="corecom:ActionCodeType"
use="optional"/>
</xsd:complexType>
```

Example 27–2 shows the Custom Schema - A: "AIAComponents\EnterpriseObjectLibrary\Industry\Telco\Custom\EBO\CustomerParty\V2\CustomCustomerPartyEBO.xsd"

Example 27–2 Adding an Extension to EBO or EBM Schemas

```
<!-- ================================================================= -->
   <!-- ==============  CustomerParty Custom Components ================ -->
<!-- ================================================================= -->
   <xsd:complexType name="CustomCustomerPartyAccountAttachmentType"/>
   <xsd:complexType name="CustomCustomerPartyAccountContactType"/>
<xsd:complexType name="CustomCustomerPartyAccountContactCreditCardType">

<!-- CUSTOMERS CAN ADD EXTENSIONS HERE ...>
</xsd:complexType>
<xsd:complexType name="CustomCustomerPartyAccountContactUsageType"/>
```

27.3.2 Extending AIA Services

Problem

The delivered services may not be sufficient to address some of your specific business functionality. You may want to add some validation logic, enrich the existing content in the service, or add some processing logic in between the existing service implementation. Adding this logic in the existing service code would not ensure upgrade safety because the upgrade process would overwrite the existing service code

with new functionality added as part of the upgrade. So, there is a need for a service extension model.

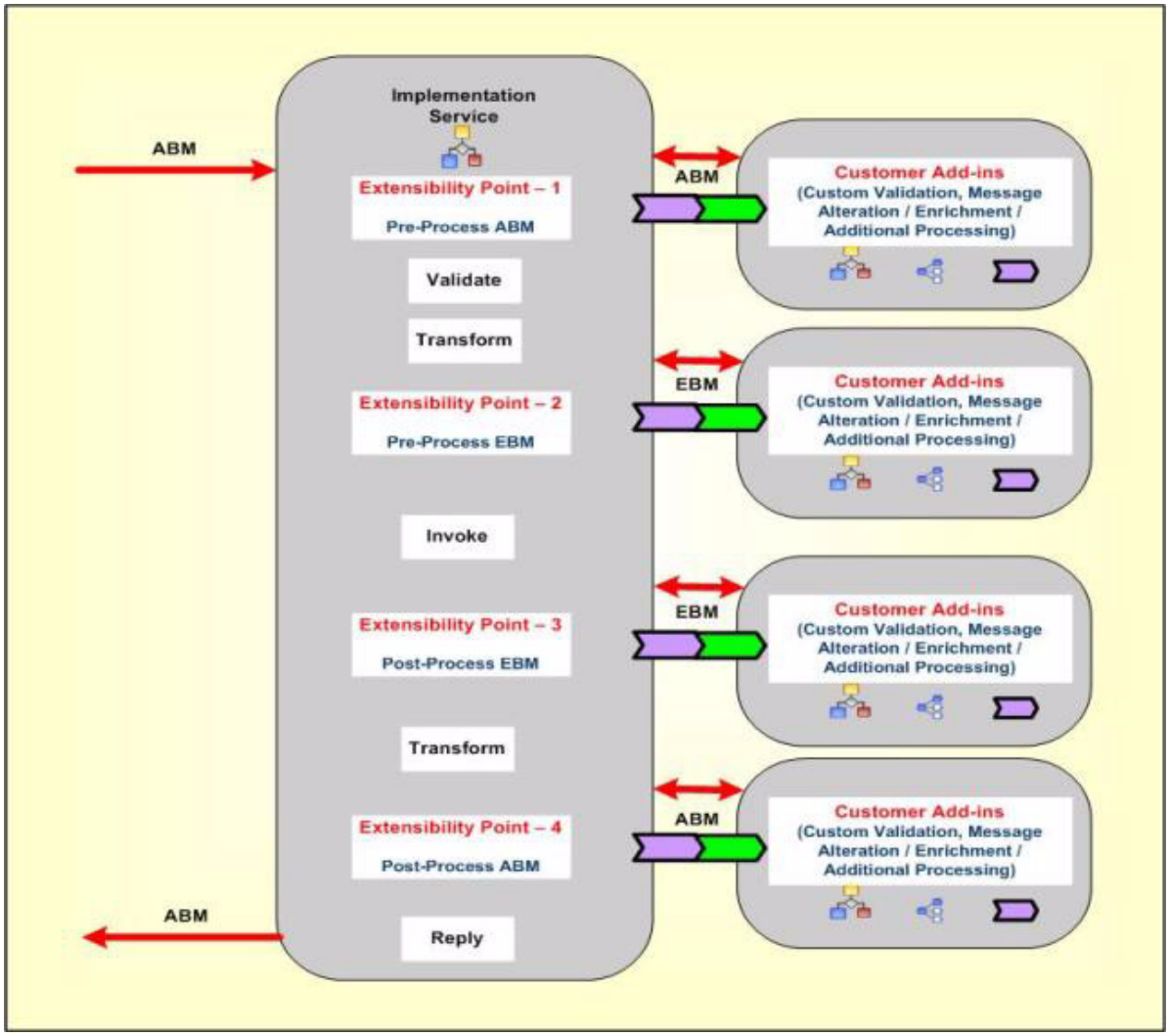
Solution

AIA recommends introducing various extension points at logical points during the service implementation. Figure 27–11 illustrates that there are four logical extension points identified in the ABCS to perform customer-specific validations, content enrichment or message alteration, and any additional logic that you would like to implement with the given payload at that extension point. Typically there are two pre-processing and two post-processing extension points identified: pre-processing ABM, post-processing ABM, pre-processing EBM and post-processing EBM.

The number of extension points may be more or less than four depending on the type of service implementation and the type of message exchange pattern used by that service.

These extension points can be enabled or disabled based on whether you want to implement those custom extension points.

Figure 27–11 Extending AIA Services



27.3.3 Extending Existing Transformations in AIA

Problem

The delivered transformations with existing schemas may not be sufficient for some of your specific business operations. You may want to add elements to the existing schemas and then add transformation maps for the newly added elements to transfer the information from one application to the other in an upgrade safe manner. This would not be possible by just adding the customer-specific transformations in an existing XSL files so the transformations extension model is needed.

Solution

To extend the transformations in a nonintrusive manner, the call template statement is defined for every business component mapping present in each of the XSL transformation files which calls the "Custom Template" defined in the custom XSL transformation.

Example 27–3 is the example to add new transformations for the custom elements added by the customer:

The main XSL mapping "XformListOfCmuAccsyncAccountIoToCreateCustomerPartyListEBM.xsl" would import the custom XSL file and have call template statement to the extension template for each complex type or business component like this:

Example 27–3 Adding New Transformations for Customer-defined Custom Elements

```
<?xml version = '1.0' encoding = 'UTF-8'?>
..……..
xmlns:dvm="http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue">
  <xsl:import href="XformListOfCmuAccsyncAccountIoToCreateCustomerPartyListEBM_
Custom.xsl"/>
..……..
  <xsl:value-of select="aia:getServiceProperty($ConfigServiceName,
'Routing.CustomerPartyEBS.CreateCustomerPartyList.CAVS.EndpointURI', false())"/>
            </corecom:DefinitionID>
          </xsl:if>
<xsl:call-template name="MessageProcessingInstructionType_ext">
            <xsl:with-param name="currentNode" select="."/>
</xsl:call-template>
        </corecom:MessageProcessingInstruction>
```

Example 27–4 shows the custom XSL file that has the template definition where you can add your custom mappings for the newly added custom elements or existing elements which are not mapped in the main transformation.

Example 27–4 Custom XSL Template Definition

```
<!-- HEADER CUSTOMIZATION -->
<xsl:template name="MessageProcessingInstructionType_ext">
  <xsl:template name="EBMHeaderType_ext">
  <xsl:param name="currentNode"/>
    <!-- Customers add transformations here -->
  </xsl:template>


<xsl:param name="currentNode"/>
    <!-- Customers add transformations here -->
  </xsl:template>
```

```
<!-- DATA AREA CUSTOMIZATION -->
 <xsl:template name="ContactType_ext">
   <xsl:param name="currentNode"/>
       <!-- Customers add transformations here -->
 </xsl:template>
```

27.3.4 Extending the Business Processes in AIA

Problem

The delivered composite business processes or Enterprise Business Flows may not be sufficient to address some of your specific business functionality. You may want to add more processing steps, message alteration, or additional processing logic between the existing composite business process implementation. Adding this logic in the existing process orchestration would not ensure upgrade safety so you need a process extension model.

Solution

AIA recommends introducing extension points at logical points during the service implementation. There are no fixed logical extension points identified for the Composite Business Processes (CBP) or Enterprise Business Flows (EBF). The number of extension points depends on the complexity of the business process and the number of message types that the process is handling in the implementation.

It is recommended to have one pre-processing and one post-processing extension point for each message type at various logical points in CBP or EBF implementation. The extension process should always be a synchronous process using the same message payload for request and response for that extension point.

These extension points can be enabled or disabled based on whether you want to implement those custom extension points.

28

Working with Security

This chapter describes how to secure service endpoints for remote communication and application security context which is used for passing authorization context from the source application to the target application through the AIA layer.

This chapter includes the following sections:

28.1 Introduction to Oracle AIA Remote Security

By default all Oracle AIA services are secured. All adapter based services are security enabled using JNDI. All composite service and references using SOAP over http are secured using Oracle Web Services Manager (WSM).

You can override the security if the delivered security is not sufficient for your usecase.

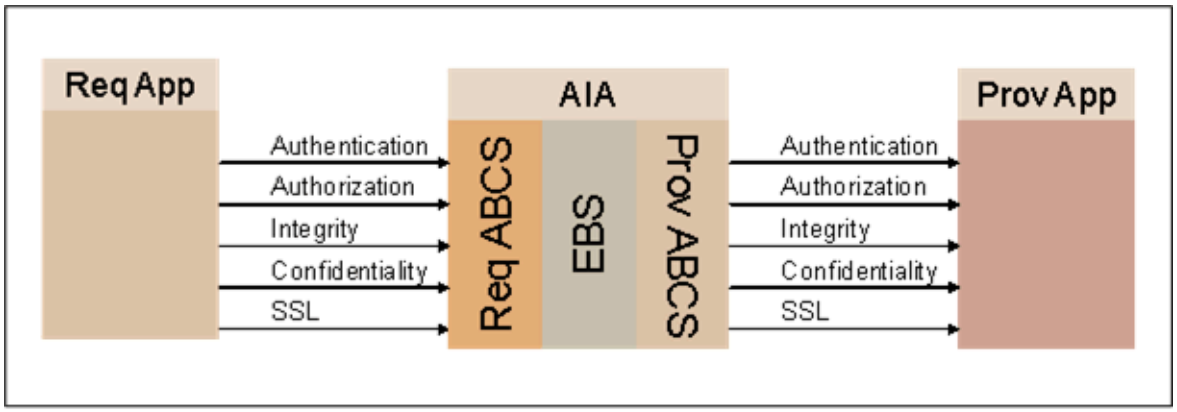
28.1.1 Securing Service to Service Interaction

By securing service to service interaction, you:

- Identify clients through authentication.
- Secure messages through encryption.
- Avoid message tampering with digital signatures.
- Encrypt the channel through SSL.

Figure 28–1 illustrates the high-level security structure for AIA.

Figure 28–1 High-level Security Architecture



AIA recommends using Oracle WSM to configure Web service security in Oracle AIA. To enable security on an AIA service you use Oracle WSM to assign the appropriate service policy. To call a secured service, you assign the appropriate client side policy.

28.1.2 Oracle AIA Recommendations for Securing Services

AIA makes the following recommendations for securing services:

- All Web Services must be secured. This includes:
  - AIA services such as Application Business Connector Services (ABCS), Enterprise Business Services (EBS), and Transport Adapter Services
  - Other application services hosted on Oracle Fusion Middleware
- The standard installation should deploy the services with security policies applied.
- In this release, the minimum protection provided by AIA services is authentication.
- You should further harden the services with message protection in the production environment.

28.1.3 Introduction to Web Service Security Using Oracle Web Services Manager

Oracle Web Services Manager (WSM) security and management is integrated into the Oracle WebLogic Server.

For more information about Oracle Web Services Manager, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services.*

AIA recommends decoupling security logic from service development by configuring Web service security declaratively using Oracle WSM during deployment. You should use Web service security rather than SSL unless you have a compelling reason, such as participating applications that do not support it.

In AIA, Oracle WSM is installed as part of SOA Suite, and there are delivered policies for most commonly used security use cases. Oracle WSM has policies for adding a particular security function as a standalone or in combination with other security functions.

The policies are globally attached to services with varying degree of granularity such as:

- Domain - all services in a domain
- Instance - all services in a WLS server instance
- Based on SOA Composite name - all services in a composite

For a list of the delivered policies, see "Predefined Policies" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services.*

28.2 Implementing Security

This section includes the following topics:

28.2.1 Enabling Security for AIA Services

To enable security in AIA services:

1. Determine what type of security is needed.

   AIA recommends using WS-security for authentication, encryption and integrity.

2. Check if the global security policy is sufficient for the web service.
3. Find the WS-policy with the appropriate combination of features.

   For example, if you need encryption and integrity, then you must find the policy which does both encryption and integrity.

4. Attach policy to *service* to enable security for a service.

   For more information about how to attach policies, see "Attaching Policies to Web Services" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services.*

5. Configure policies.

   You may perform additional configurations for each policy.

   For more information about how to configure each policy, see "Configuring Policies" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services.*

6. Diagnose problems.

   For more information about how to diagnose problems, see "Diagnosing Problems" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

28.2.1.1 Should You Secure All AIA Services?

Security can have a negative impact in terms of performance so it must be carefully used. AIA mandates securing services whose interaction with other services cross organizational boundaries. Any AIA service that is being called from clients outside corporate network must be secured. For A2A integration where all the services are within the same organization and inside the firewall, you may decide it is not necessary to secure all the services.

All AIA services out of the box are secured, so if you think security is not needed for your deployment, you must disable.

28.2.2 Invoking Secured Application Services

To invoke a secured web service:

1. Determine what type of security is needed.

   AIA recommends using WS-security for authentication, encryption and integrity.

2. Check if the global security policy is sufficient for the web service.

3. Find the WS-policy with the appropriate combination of features.

   For example, if you need encryption and integrity, then you must find the policy which does both encryption and integrity.

4. Attach policy to *service* to enable security for a service.

   For more information about how to attach policies, see "Attaching Policies to Web Services" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services.*

5. Configure policies.

   You may perform additional configurations for each policy.

   In the case of WS-Security client side basic authentication policy, you must do the following:

   1. Configure credential store.
   2. Add the UserID and password associating with a key into the store.
   3. Use the key in the policy.

   For more information about how to configure each policy, see "Configuring Policies" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services.*

6. Diagnose problems.

   For more information about how to diagnose problems, see "Diagnosing Problems" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

28.2.3 Overriding Policies Using a Deployment Plan

AIA Foundation Pack provides infrastructure to override the global policies in a declarative way. To override the policy, service developers must:

1. Create service configuration file as described in section Section 28.6.3, "AIA Security Configuration Properties."
2. Place it in the project folder.

28.2.4 Testing Secured Services using CAVS

To test secured services using CAVS, the element <cavs:CAVSRequestInput_1> should have the element shown in Example 28–1 under the <soap:Envelope>.

Example 28–1 <soap:Envelope> Content

```
<soapenv:Header>
      <wsse:Security
xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">
         <wsse:UsernameToken>
            <Username>[user name]</Username>
            <Password>[pwd]</Password>
         </wsse:UsernameToken>
      </wsse:Security>
   </soapenv:Header>
```

If you are using a default user in the Identity store, then [user name] = weblogic and [pwd] = weblogic#1.

28.3 Security for Applications

This section contains the following topics:

- Section 28.3.1, "Enabling Security in Application Services"
- Section 28.3.2, "Invoking Secured AIA Services"

28.3.1 Enabling Security in Application Services

You can use the built-in capabilities of participating applications to enable security for services. To choose a product for enabling security, check if Oracle WSM has agent support for the application, and if it has, use Oracle WSM.

If the applications can enable any kind of security, use Web service security for authentication, encryption, and integrity. Otherwise, you can use SSL to secure the connection.

28.3.2 Invoking Secured AIA Services

When interacting with an AIA service that is enabled for WS-security, you must add a security header in the SOAP header with all the information needed for security functions on AIA service. Based on the security of the AIA service, you must add information for any combination of authentication, encryption and integrity.

Example 28–2 is a sample of a security header for authentication.

Example 28–2 Security Header for Authentication

```
<wsse:Security  env:mustUnderstand="1">
<wsse:UsernameToken  wsu:Id="UsernameToken-dXtD14011QZUTlfIaSrMhw22">
<wsse:Username>weblogic</wsse:Username>
<wsse:Password
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-
profile-1.0#PasswordText">weblogic1</wsse:Password>
</wsse:UsernameToken>
</wsse:Security>
```

If the AIA service requires SSL, then the application should configure SSL for both one way and two-way SSL.

28.4 Deploying Security Policies

AIA recommends applying policies globally rather than constraining policy attachment at the individual service level. The global policies are applied when the Foundation Pack is installed.

In some cases, it is imperative to override the globally attached client policies with directly attached local policies. General guidelines are given below.

- Global Authentication Policies are delivered
  - Eliminates the need to define policies at the composite level.
  - Global Service Policy applied:
    oracle/aia_wss_saml_or_username_token_service_policy_OPT_ON.

    This is a cloned copy of oracle/wss_saml_or_username_token_service_policy with Local Optimization set to ON.
  - Global Service Client Policy applied:
    oracle/aia_wss10_saml_token_client_policy_OPT_ON

    This is a cloned copy of oracle/wss10_saml_token_client_policy with Local Optimization set to ON.
- Assess your individual flow needs and harden the services if necessary.

  Further hardening can be done by associating local policies.
- Applications invoking secured AIA Web Services must send credentials.
- Inter-AIA communication is handled by the Global Service Client Policy.

28.4.1 Oracle AIA Recommendations for Policies

In general, determining which policies to use depends on the basic requirements of your organization's security policy. The following questions help determine which policies can be used.

- Is there a need only to authenticate users?
- Is there a need for message protection?
- Will the token be inserted in the transport layer or in a SOAP header?
- Should you use a particular type of token?

The following policies should be attached globally to the AIA Services:

- oracle/aia_wss_saml_or_username_token_service_policy_OPT_ON
- aia_wss_saml_or_username_or_http_token_service_policy_OPT_ON
- oracle/aia_wss10_saml_token_client_policy_OPT_ON

oracle/aia_wss_saml_or_username_token_service_policy_OPT_ON

This is a cloned copy of oracle/wss_saml_or_username_token_service_policy with Local Optimization set to ON. This is needed for local optimization to work when both client and service composite are co-located.

This policy authenticates users using credentials provided either in SAML tokens in the WS-Security SOAP header or in the UsernameToken WS-Security SOAP header. The credentials in a SAML token are authenticated against a SAML login module, while the credentials in a UsernameToken are authenticated against the configured

identity store. Only plain text mechanism is supported for the UsernameToken. This policy can be applied to any SOAP-based endpoint.

aia_wss_saml_or_username_or_http_token_service_policy_OPT_ON

This is a cloned copy of oracle/wss_saml_or_username_token_service_policy with Local Optimization set to ON and http basic authentication added as an additional option. Clients such as ODI that do not have the infrastructure to use webservices security can call this service using http basic authentication.

This is only attached to AIAAsyncErrorHandlerBPEL service.

oracle/aia_wss10_saml_token_client_policy_OPT_ON

This is a cloned copy of oracle/wss10_saml_token_client_policy with Local Optimization set to ON. This is needed for local optimization to work when both client and service composite are co-located.

This policy includes SAML tokens in outbound SOAP request messages.

28.5 Policy Naming Conventions

This section includes the following topics:

- Section 28.5.1, "Naming Conventions for Global Policy Sets"
- Section 28.5.2, "Naming Conventions for Overriding Config Params"

28.5.1 Naming Conventions for Global Policy Sets

Naming convention for service-specific global policy sets

- AIA_[ServiceType]_WSServicePolicySet
- Possible values for Service Type:
  - ABCS
  - EBS
  - EBF
  - Adapter
  - Producer
  - Consumer
- Example: AIA_ABCS_WSServicePolicySet

Naming convention for client-specific global policy sets

- AIA_[ServiceType]_WSClientPolicySet
- Possible values for Service Type:
  - ABCS
  - EBS
  - EBF
  - Adapter
  - Producer
  - Consumer

- Example: AIA_ABCS_WSClientPolicySet

28.5.2 Naming Conventions for Overriding Config Params

Naming convention for config param - csf key

- AIA_APPSHORTNAME_ServiceName_PortTypeName
  - APPSHORTNAME: Application short name as defined in the service registry
  - ServiceName: Value of the attribute 'name' of the element 'service' in the WSDL of the External Web Service
  - PortTypeName: Value of the attribute 'name' of the element 'portType' in the WSDL of the External Web Service

28.6 How Does AIA Foundation Pack Help in Securing AIA Services?

AIA Foundation Pack automatically:

- Creates the recommended global policy sets
- Attaches the local policies for the composites when required.

This section includes the following topics:

- Section 28.6.1, "What Default Policies are Attached to a Service?"
- Section 28.6.2, "How Can the Global Policy be Overridden for an Individual Service?"
- Section 28.6.3, "AIA Security Configuration Properties"

28.6.1 What Default Policies are Attached to a Service?

The deployment of global policies is handled during the Foundation Pack installation. A set of global services and client policies are attached to the WLS domain at the time of Foundation Pack installation. The result is automatic securing of all services (matching the pattern) deployed on that server.

Foundation Pack installer creates both service-specific and client-specific global policy sets for the composite name patterns:

- ABCS
- EBS
- EBF
- Adapter
- Producer
- Consumer

Foundation Pack Installer creates the following policy sets:

- Global PolicySets attached to Composite Services
  - AIA_ABCS_WSServicePolicySet
  - AIA_EBS_WSServicePolicySet
  - AIA_EBF_WSServicePolicySet
  - AIA_Adapter_WSServicePolicySet

  - AIA_Consumer_WSServicePolicySet
  - AIA_Producer_WSServicePolicySet
- Global PolicySets attached to Composite References
  - AIA_ABCS_WSClientPolicySet
  - AIA_EBS_WSClientPolicySet
  - AIA_EBF_WSClientPolicySet
  - AIA_Adapter_WSClientPolicySet
  - AIA_Consumer_WSClientPolicySet
  - AIA_Producer_WSClientPolicySet

28.6.2 How Can the Global Policy be Overridden for an Individual Service?

Composites that must interact with protected application services needing a different security policy, have local policies attached, overriding the global polices at the time of deployment.

Similarly, the **NoClientAuthenticationPolicy** is attached, overriding the global policy sets for composites that must interact with non-protected application services.

The AIA Deployment Driver provides support, in general, for attaching any overriding local security policy, but supports configuration-overriding only for saml-token and username-token client policies.

When a different client (local) policy is used, the AIA Deployment Driver attaches the policy but its configuration is a manual task.

The structure of the XML file is shown in Example 28–3.

28.6.3 AIA Security Configuration Properties

Composites that require local policies attached to either service endpoints or reference endpoints or both, must furnish the information to the Foundation Pack tool. These composites must have an associated xml-based security configuration file. This file is named **AIASecurityConfigurationProperties.xml**.

The Foundation Pack tool needs the following information:

- Name of the composite
- Name of all service endpoints that require local policies
- Name of all reference endpoints that require local policies
- Name of the policies that must be locally attached

The associated AIASecurityConfigurationProperties.xml of the composite that requires a local policy attachment must furnish listed above. This file is placed along with the project artifacts in the same folder as the composite.xml.

This file should be source-controlled.

When a composite does not require a local policy attachment, then it is not necessary to have this xml file defined for that composite.

Example 28–3 shows a sample AIASecurityConfigurationProperties.xml.

Example 28–3 Sample AIASecurityConfigurationProperties.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Note: the attribute 'compositeName' is the name of the AIA Service
composite preprended by {namespace of the AIA service as defined
in its wsdl} -->
<SecurityConfiguration xmlns="http://xmlns.oracle.com/fp/core/security/V1"
version="1.0"
compositeName="{http://xmlns.oracle.com/ABCSImpl/Siebel/Samples/
SamplesCreateCustomerSiebelReqABCSImpl/V1}SamplesCreateCustomerSiebel
ReqABCSImpl">
<!-- the following element is repeated for each service end point of this
Composite ,which requires a direct (local) policy attachement -->
<Service resourceType='SOA-Service' >
<!-- It is the service endpoint. It should be same as attribute 'name' of
element  'service' in composite.xml -->
<Name>SamplesCreateCustomerSiebelReqABCSImpl</Name>
<!-- This is the port name. For BPEL-based references, its value is Name of the
Porttype as given in the WSDL of this AIA service
For Mediator-based reference, this is [Name of the Porttype element as given in
the WSDL]_pt This example assumes a scenario when services and wsdls are coded
by following the AIA naming conventions.In other scenarios, the value might be
slightly different. Look at the hint below to come up with the correct value
for the element PortName.-->
<PortName>SamplesCreateCustomerSiebelReqABCSImpl</PortName>
<WSPolicies>
<WSPolicyName policyType ="authentication">oracle/wss_username_token_service_
policy</WSPolicyName>
</WSPolicies>
</Service>
<!-- the following element is repeated for each reference end point of this
Composite ,which requires a direct (local) policy attachement -->
<Reference resourceType = 'SOA-Reference'>
<!-- should be same as attribute 'name' of element  'reference' in
composite.xml -->
<Name>SamplesCustomerPartyEBS</Name>
<!-- port name.
For BPEL-based references, its value is name of the Porttype element as given
in the WSDL of this AIA service.
For Mediator-based reference, the value is [Name ofthe Porttype element as
given in the WSDL]_pt
-->
"This example assumes a scenario when services and wsdls are coded by following
the AIA naming conventions. In other scenarios, the value might be slightly
different. Look at the hint below to come up with the correct value for the
element PortName. Hint: In the composite.xml, for the binding.ws of the
'service' element, the attribute 'port' takes value of the following form.
<binding.ws port="[namespace of the service as defined in the
wsdl]/V1#wsdl.endpoint(<Service>/<Port>" Make sure that the PortName provided
by you here, is same as <Port> in the composite.xml"
<PortName>CustomerPartyEBS_pt</PortName>
<WSPolicies>
<WSPolicyName policyType ="authentication">oracle/wss_username_token_client_
policy</WSPolicyName>
<ConfigParams>
 <!-- Param could be a repeating element- Future use only -->

  <!-- APPSHORTNAME should be same as application's short name  -->

  <!-- ServiceName and PortTypeName are as given in the APP's web service WSDL
-->
```

```
 <Param paramName="csf-key">APPSHORTNAME_ServiceName_PortTypeName</Param>

</ConfigParams>
</WSPolicies>
</Reference>
</SecurityConfiguration>
```

> **Tip:** In the composite.xml, for the binding.ws of the 'reference' element, the attribute 'port' takes value of the following form.
>
> <binding.ws port="[namespace of the service as defined in the wsdl]/V1#wsdl.endpoint(<Service>/<Port>"
>
> Ensure that the PortName provided by you here, is the same as <Port> in the composite.xml.

Points to note for a composite:

- If no service endpoint requires a direct policy attachment, but a reference endpoint requires one, then the AIASecurityConfigurationProperties.xml need not have a '**service**' element.
- If no reference endpoint requires a direct policy attachment, but a service endpoint requires one, then the AIASecurityConfigurationProperties.xml need not have a '**reference**' element.
- If there are two or more service endpoints, but only one of them requires a direct policy to be attached, then only one '**service**' element must be present in the AIASecurityConfigurationProperties.xml.
- If there are two or more reference endpoints, but only one of them requires a direct policy to be attached, then only one '**reference**' element must be present in the AIASecurityConfigurationProperties.xml.

28.7 Application Security Context

This section includes the following topics:

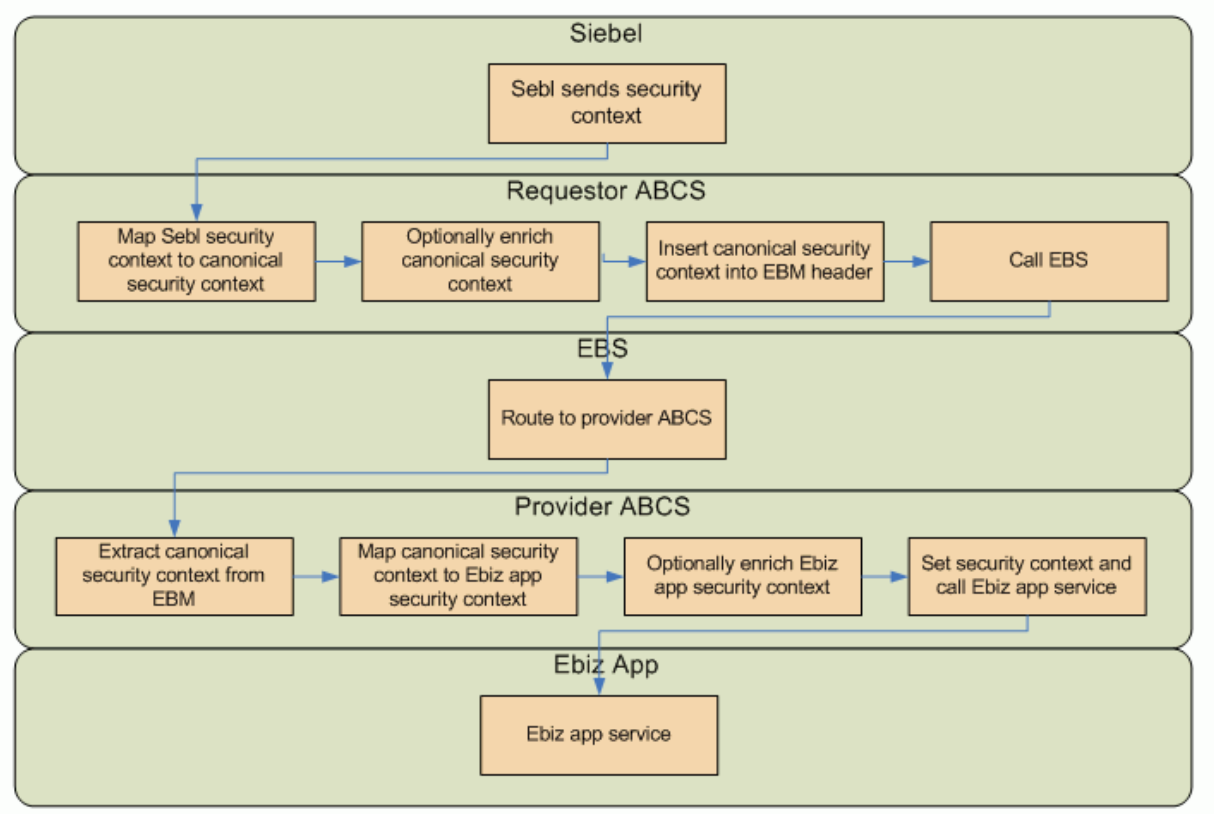
28.7.1 Introduction to Application Security

The Oracle AIA application security model allows AIA to integrate participating applications with different security representations in a standard way by eliminating point-to-point security.

The participating applications are developed at different times with different concepts and implementations of authentication and authorization. When applications are integrated, you must pass authentication and authorization information between applications. AIA application security context standardizes the exchange of participating applications' authentication and authorization information between various applications so that any application can be integrated with any other application.

Figure 28–2 illustrates the high-level security functional flow.

Figure 28–2 Security Functional Flow



28.7.2 How To Exchange Security Context Between Participating Applications and ABCS

App Context is any information that must be sent to the provider application to process the message sent from requester application or vice versa. This includes, but is not limited to, authentication and authorization information. AIA addresses the exchange of authorization information in app context, but the design supports adding other context information.

AIA determined XACML Context Request as the best standard to represent authorization information. XACML is an OASIS standard for managing access control policy. Released in 2003 and based on XML, XACML is designed to become a universal standard for describing who has access to which resources. XACML includes a policy language and a query language that results in a Permit, Deny, Intermediate (error in query), or Not Applicable response. The query language is expressed in XACML context that is recommended by AIA for exchanging authorization information.

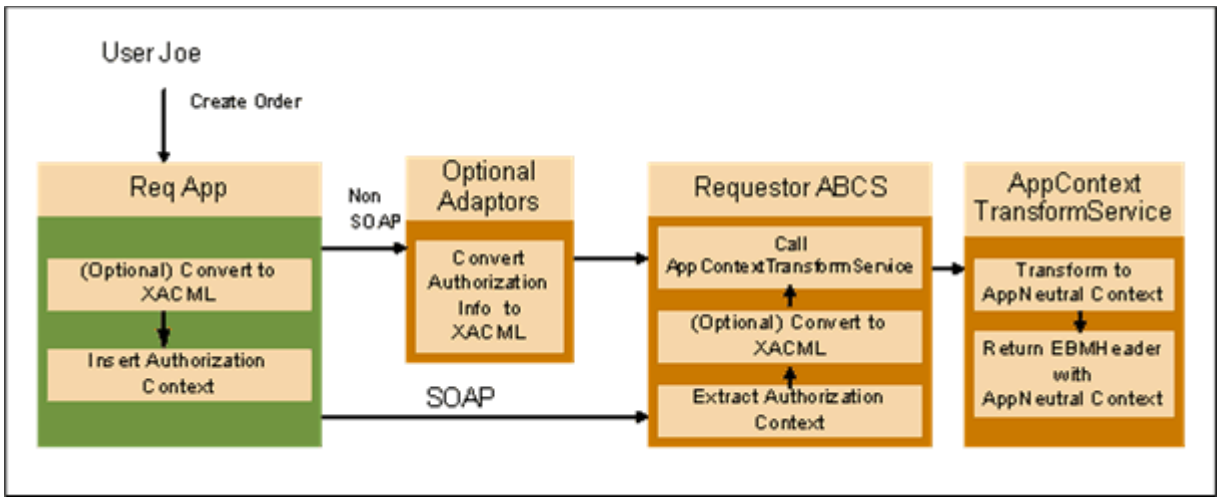
28.7.2.1 Requester Applications

The preferred approach is to let the requester application send application context information as an XACML request to the Requester ABCS. If the applications are not capable of formulating context information in an XACML request, then the participating application send application context information in a SOAP header or as part of business message content.

AIA recommends the use of a protocol specific adapter if the participating application does not use a SOAP interface. In that scenario, the adapter receives the application context in a custom way, prepares the participating application specific XACML request, and sends it to the ABCS.

Figure 28–3 illustrates the requester application flow.

Figure 28–3 Requester Application Flow

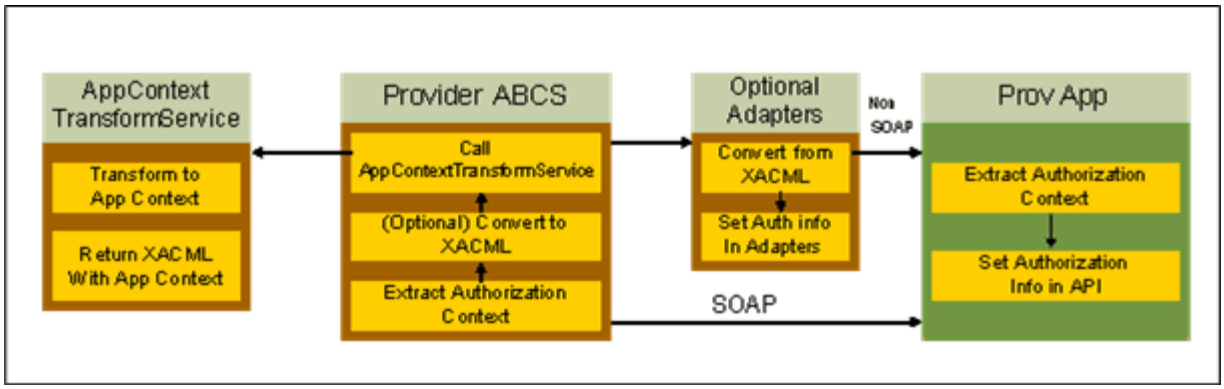


28.7.2.2 Provider Applications

The preferred approach is to let the provider ABCS send the application context as an XACML request to the provider application. If the provider application cannot receive an XACML request, but has a SOAP interface, then the provider ABCS sends the application security context in a custom XML format inside a SOAP header or as part of a business document. If the provider application does not support a SOAP interface, then the provider ABCS sends the application context in an XACML request format to the adapter service that sets the appropriate security context needed for the security mechanism in use.

Figure 28–4 illustrates the provider application flow.

Figure 28–4 Provider Application Flow



28.7.3 Mapping Application Security Context in ABCS To and From Standard Security Context

The requester ABCS either receives the application security context in XACML format or converts it into XACML format. The requester ABCS calls an external service to map application security context to standard security context. The ABCS passes the application security context in XACML format and receives application neutral security context in XACML format.

28.7.4 Using the AppContext Mapping Service

AIA recommends using one external service per application. This service is also responsible for populating additional values needed in the standard or application context that is returned. This service can be implemented as XPath functions or web service with these names:

- Request TransformToAppContext (EBMHeader)
- Request TransformToAppNeutralContext (Request)

Example 28–4 shows a sample of the AppContextMappingService.

Example 28–4 Example of AppContext Mapping Service

```
<definitions
targetNamespace="http://www.oracle.com/AIA/AppContextTransformService"
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
          xmlns:xacml-context="http://docs.oasis-open.org/xacml/access_
control-xacml-2.0-context-schema-cd-04.xsd"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tacs="http://www.oracle.com/AIA/AppContextTransformService"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
   <types>
   <xsd:schema
targetNamespace="http://www.oracle.com/AIA/AppContextTransformService"
elementFormDefault="qualified">
<xsd:import namespace="http://docs.oasis-open.org/xacml/access_
control-xacml-2.0-context-schema-cd-04.xsd"
schemaLocation="http://[HOST:PORT]/AIAComponents/EnterpriseObjectLibrary/Release2
/Core/Common/V2/access_control-xacml-2.0-context-schema-cd-04.xsd" />
<xsd:import namespace="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
```

```
schemaLocation="http://[HOST:PORT]/AIAComponents/EnterpriseObjectLibrary/Release2
/Core/Common/V2/Meta.xsd" />
</xsd:schema>
   </types>
   <message name="Request">
     <part name="Request" element="xacml-context:Request"/>
   </message>
   <message name="EBMHeader">
     <part name="EBMHeader" element="corecom:EBMHeader"/>
   </message>
   <portType name="TransformAppContext">
     <operation name="TransformToAppContext">
       <input message="EBMHeader"              name="EBMHeader"/>
       <output message="Request"/>
     </operation>
     <operation name="TransformToAppNeutralContext">
       <input message="Request"              name="Request"/>
       <output message="Request"/>
     </operation>
   </portType>
 </definitions>
```

This service is implemented for the participating application and meets any integration scenario using that application.

AIA recommends using BPEL with co-location to implement this service. ABCS should call this service using a dynamic partner link so that you can plug in other implementations of this service.

TransformAppContextService is the property used to load the service implementation from AIAConfig property file. By default this property is not configured and the default implementation is used.

The default implementation of this service is based on DVM and cross-reference. Whenever a new application or integration scenario is added, new DVM values must be populated but the service need not be changed.

28.7.5 Understanding the Structure for Security Context

The **XACML Request** element is used as the parameter to the app context structure. This request element carries participating application information and calling service information in addition to authorization information.

Figure 28–5 illustrates the structure of XACML Request.

Figure 28–5 Structure of XACML Request

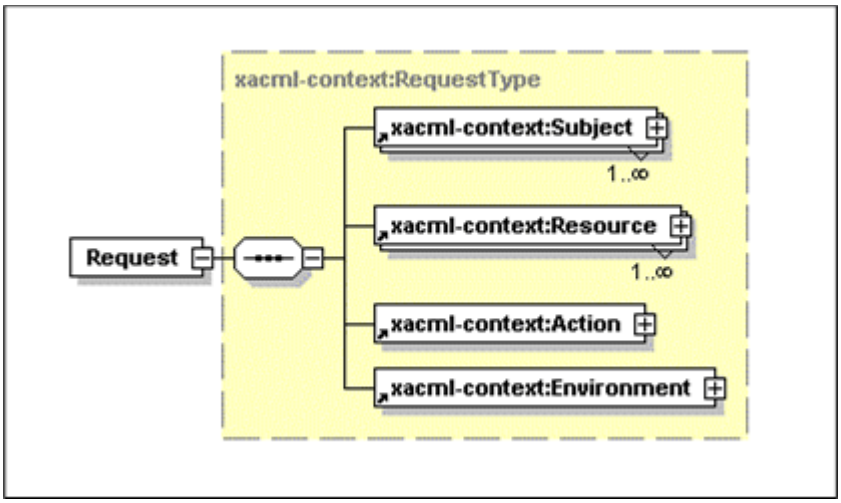


Figure 28–6 illustrates the structure of XACML Subject.

Figure 28–6 Structure of XACML Subject

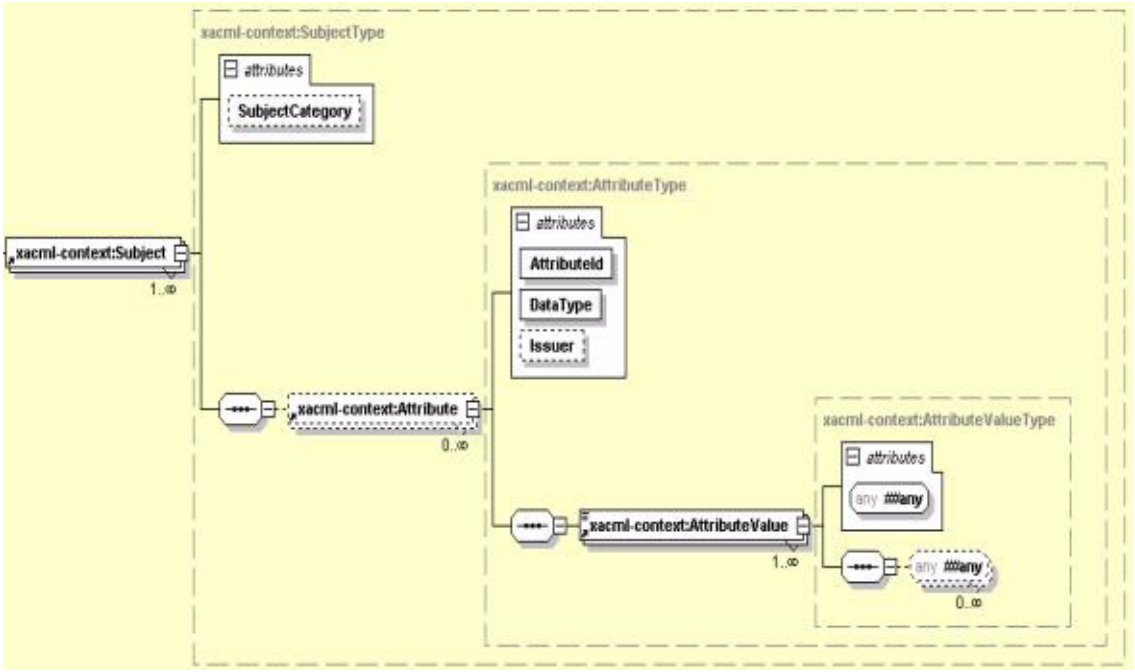


Figure 28–7 illustrates the structure of XACML Resource.

Figure 28–7 Structure of XACML Resource

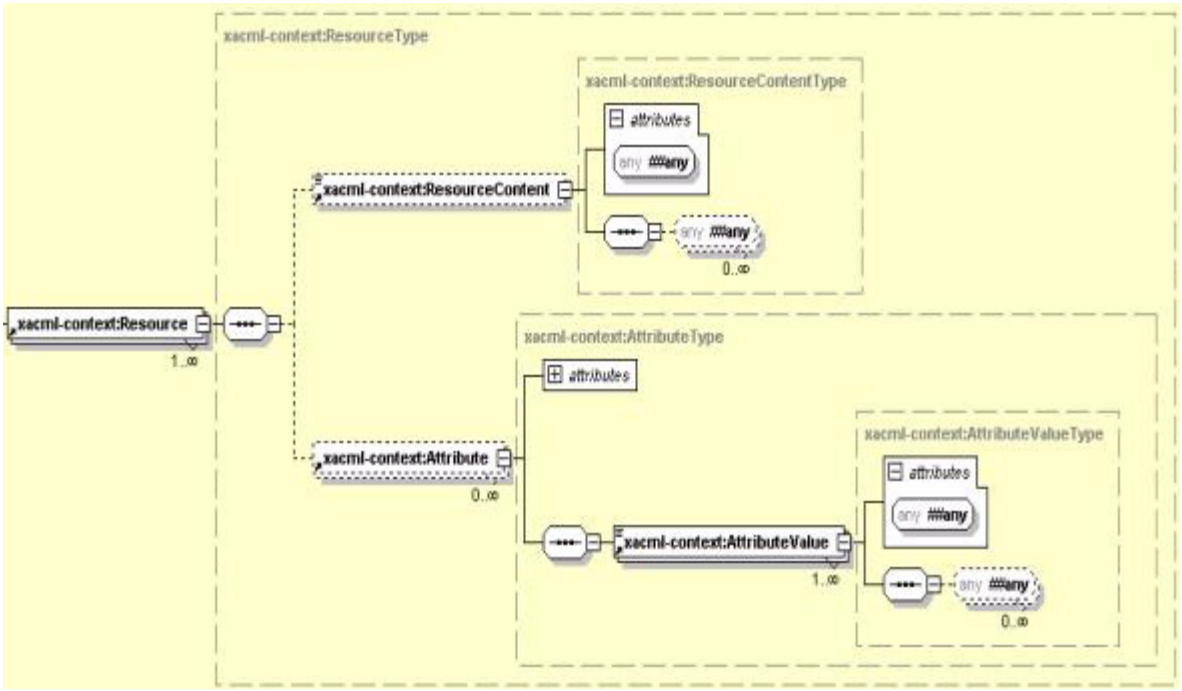


Figure 28–8 illustrates the structure of XACML Action.

Figure 28–8 Structure of XACML Action

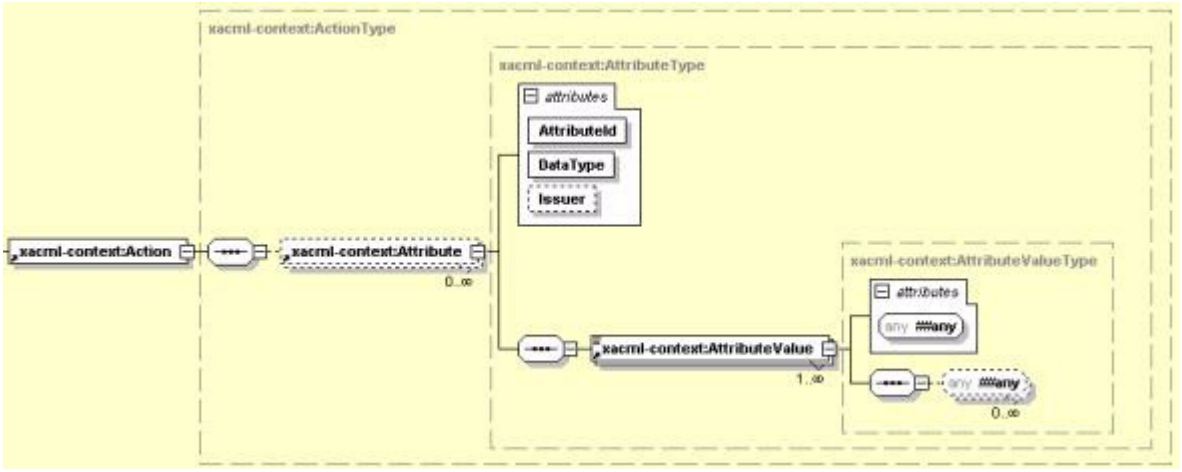
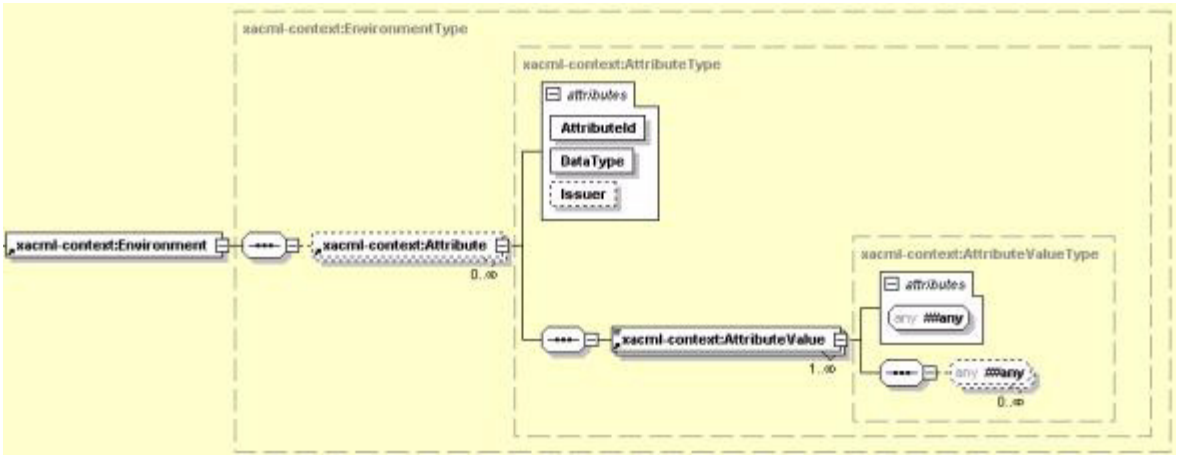


Figure 28–9 illustrates the structure of XACML Environment.

Figure 28–9 Structure of XACML Environment



Example 28–5 shows the SEBL AppContext information that is sent to the security service.

Example 28–5 Example of SEBL AppContext information Sent to the Security Service

```
<AIAAppContext xmlns=http://www.oracle.com/AIA/AppContext>
  <ServiceInfo>
     <ServiceName>O2C2SeibelABCS</ServiceName>
  </ServiceInfo>
  <ParticipatingAppInfo>
     <Name>Siebel</Name>
     <Version>8.0</Version>
  </ParticipatingAppInfo>
  <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:cd:04"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
urn:oasis:names:tc:xacml:2.0:context:schema:cd:04
http://docs.oasis-open.org/xacml/access_
control-xacml-2.0-context-schema-cd-04.xsd">
     <Subject>
      <Attribute AttributeId="siebel:user" DataType="xs:string">
        <AttributeValue>SAdmin</AttributeValue>
      </Attribute>
      <Attribute AttributeId="siebel:org" DataType="xs:string">
        <AttributeValue>siebl1</AttributeValue>
      </Attribute>
     </Subject>
     <Resource>
     </Resource>
     <Action>
     </Action>
     <Environment/>
  </Request>
</AIAAppContext>
```

28.7.6 Using Attribute Names

Use these guidelines for attribute names:

- Service information attributes:

  AIA:Service:Name - Name of the service calling the transform service

- Participating application information attributes:

  AIA:ParticipatingApp:Name - Name of the participating application

  AIA:ParticipatingApp:Version - Version of the participating application

  AIA:ParticipatingApp:SystemID - unique identifier of participating application

- Application attributes:

  AIA recommends using this convention for naming the attributes for all the applications: *Application name: attribute name*.

- Application neutral attributes:

  AIA recommends using AIA as prefix for all the application neutral attributes. These are the application neutral attributes identified so far:

  User: to represent user

  BusinessUnit: to represent organization or operating unit

28.7.7 Propagating Standard Security Context through EBS and EBF

The standard security context is inserted into the Enterprise Business Message (EBM). As an EBM is propagated through various EBSs and EBFs to the destination ABCS, the security context is propagated along with the EBM to the target ABCS where it is used to propagate to the target application

28.7.8 Implementing Application Security Context

The following section provides the high level steps for implementing application security context on both the requester side and the provider side.

28.7.8.1 How to Implement Requester-Side Application Security Context

To implement requester-side application security context:

1. If an adapter is used, convert application security context information into XACML format in the adapter service.
2. If the application is sending information in data directly to the requester ABCS, convert the application's security context information to XACML format.
3. If new standard attributes are needed, work with internal architecture team.
4. Implement application context mapping service.
5. In the Requester ABCS, call the application mapping service to convert application specific app context information to application neutral app context information.
6. Call EBS.

28.7.8.2 How to Implement Provider-Side Application Security Context

To implement provider-side application security context:

1. Implement application context mapping service.
2. In the Provider ABCS, call application context mapping service to convert application neutral app context information to application specific app context information.

3. To send information in data directly to provider application, convert applications security context information from XACML data to required form.
4. If an adapter is used, convert application security context information from XACML format to the required form in the adapter service.

29

Best Practices for Designing and Building End-to-End Integration Flows

This chapter discusses best practices and recommendations for designing and building end-to-end integration flows.

This chapter includes the following sections:

- Section 29.1, "General Guidelines for Design, Development, and Management of AIA Processes"
- Section 29.2, "Building Efficient BPEL Processes"

29.1 General Guidelines for Design, Development, and Management of AIA Processes

This section includes the following topics:

- Section 29.1.1, "Interpreting Empty Element Tags in XML Instance Document"
- Section 29.1.2, "Purging the Completed Composite Instances"
- Section 29.1.3, "Syntactic / Functional Validation of XML Messages"
- Section 29.1.4, "Provide Provision for Throttling Capability"
- Section 29.1.5, "Artifacts Centralization"
- Section 29.1.6, "Separation of Concerns"
- Section 29.1.7, "Adapters Inside ABCS Composite OR as Separate Composite"
- Section 29.1.8, "AIA Governance"
- Section 29.1.9, "Using AIA Service Constructor"

29.1.1 Interpreting Empty Element Tags in XML Instance Document

The XML Instance document should have empty element tags only when the sender intends to have the consumer nullify the values for the elements. Otherwise, these tags should not be present. Having these nonsignificant tags can have a huge impact on memory consumption and hence scalability of the application.

AIA recommends that Enterprise Business Messages (EBMs) have only significant elements.

Some applications can inspect the content and produce an XML document having only the significant tags, whereas the rest do not. Because the ABCS is intimate with the application, it can choose the appropriate style based on the application's behavior. In

situations in which the applications produce Application Business Messages (ABM) containing all of the elements regardless of whether the elements underwent change or not, the requester connector services should assume that the empty elements in ABM are not significant; and should ignore them when producing EBM. The first code example shown in Example 29–1 illustrates how this could be done. In situations in which ABMs contain only elements that underwent a change in value, the connector services should treat the presence of an empty element as the sender's intent to have the consumer nullify the values for that element. Example 29–2 illustrates this use case. With service requesters producing EBM in the preceding manner, the job of the consumer becomes straightforward. It can assume that every element is a significant one-hence, the ABCS consuming the EBM should not skip the empty element. Example 29–3 illustrates how a message containing significant elements can be processed.

Use the construct in Example 29–1 when the source application's ABM contains all the elements defined in its schema regardless of whether all those elements underwent any change in value. In this situation, in the interest of conserving memory, AIA makes the assumption that an empty element is NOT a significant element and ignores it.

Example 29–1 ABM -> EBM Transformation

```
~--> <xsl:if test="ABMSourceElement/text()"> <EBMTargetElement> <xsl:value-of
select="ABMSourceElement"/> </EBMTargetElement> </xsl:if>
```

Use the construct in Example 29–2 when the source application's ABM contains only those elements that underwent a change in value. In this situation, the presence of an empty element can be considered significant and so AIA does not ignore it.

Example 29–2 ABM -> EBM transformation

```
~--> <xsl:if test="ABMSourceElement/text()"> <EBMTargetElement> <xsl:value-of
select="ABMSourceElement"/> </EBMTargetElement> </xsl:if>
```

If the above rules have been followed, then we can assume that any empty element in the EBM is a significant one, and therefore should not be ignored when transforming from EBM to ABM. So use the construct in Example 29–3.

Example 29–3 EBM -> ABM Transformation

```
~--> <xsl:if test="EBMSourceElement"> <ABMTargetElement> <xsl:value-of
select="EBMSourceElement"/> </ABMTargetElement> </xsl:if>
```

29.1.2 Purging the Completed Composite Instances

AIA highly recommends having a process in place to archive and clean up the completed composite instances after a specified period.

- Synchronous request -response-based completed composite instances should be purged from SOA dehydration data store after a month.
- Data synchronization-related composite instances should be purged three to four months after their completion.
- Composite Business Processes should be purged one to two years after their completion.

Even though this duration is subject to change based on your company policies, attempts should be made to do the purging at the earliest time. Purging must be preceded by archiving of the instance data.

For more information, see "" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite.*

29.1.3 Syntactic / Functional Validation of XML Messages

This section discusses:

- Section 29.1.3.1, "Syntactic Validation"
- Section 29.1.3.2, "Data / Functional Validation"

29.1.3.1 Syntactic Validation

With document style, a web service endpoint can use the capabilities of a validating parser and the run time to perform syntactic validation on business documents against their schema definitions.

A schema specifies the data shape of the payload that could be sent to the web service operation. When a SOAP message is received by the web-services handler, the schema pertaining to the operation being called is used to validate the content of the SOAP message.

Because this validation has a performance impact, it should be used judiciously in a production environment. AIA recommends enabling the schema validation only for the service-receiving message from a requester outside its boundary. Even though the requester could be an internal application, the syntactic validation is normally applied at run time only for messages coming from an external source. AIA highly recommends that you turn off schema validation for the intermediary services residing in AIA layer.

In a development environment, AIA highly recommends that you turn on schema validation for all services during certain testing phases to ensure that messages produced by services are syntactically valid.

For more information about how to enable schema validation, see *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.*

29.1.3.2 Data / Functional Validation

AIA highly recommends that the validation be done by the service provider. It could be done either by the provider ABCS or by the application itself.

29.1.4 Provide Provision for Throttling Capability

The implementation of cross-functional business processes often requires integrating endpoint systems having divergent nonfunctional characteristics such as availability and concurrent processing capability. Occasionally, the endpoints become unavailable due to planned or unplanned outages and many systems may not be able to handle large volumes of concurrent messages. Overwhelming these systems with a high-volume of requests may result in the failure of target endpoints, which in turn has a cascading impact on the overall integration. AIA recommends implementing store and forward capability using queues in most of the situations while interacting either with the requester or provider application. This gives opportunities for customers to implement throttling that helps impose a limit on the rate at which message requests are handed over to a specific endpoint.

For more information, see Section 27.1, "AIA Message Processing Patterns".

29.1.5 Artifacts Centralization

To facilitate the governance of shared artifacts such as Enterprise Business Objects, Enterprise Business Service WSDLs, Application Business Message Schemas, ABCS WSDLs, Domain Value Maps, and Cross-Reference Meta data, AIA highly recommends designing and implementing these artifacts independently from the service capabilities that use them. AIA strongly discourages the management and persistence of these artifacts as part of individual services. AIA's programming model recommends the use of MDS to persist these artifacts.

For more information about how MDS is used for centralization of these assets, see Section 2.1.3.4, "Using MDS in AIA".

29.1.6 Separation of Concerns

Separation of concerns is a core principle of SOA. It helps you introduce loose coupling in the integration flow where it is required. However, it must be applied not only at the architectural level, but at the implementation level as well.

Managing dependencies between services at development and run time is a challenging task when developers implement requester and provider ABC services. One of the approaches is to make them de-coupled. Using this approach, the cross-dependencies in the code that handle each concern of the service can be minimized.

You may see error messages indicating invalid SOA composites after a server restart. This is caused by referring to concrete WSDLs where abstract WSDLs should have been used.

You will not see the problem at the first deployment of a new composite X when you reference the concrete WSDL of composite Y. Why? Composite Y is up and running at the time of the deployment of composite X. The problem starts when the server is restarted because there is no guarantee that the composites will be activated in the right order to resolve any dependencies. If service X is activated before composite Y, the reference cannot be resolved because Y is still down, so X remains in status *Invalid*.

You can begin with separating the interface from implementation.

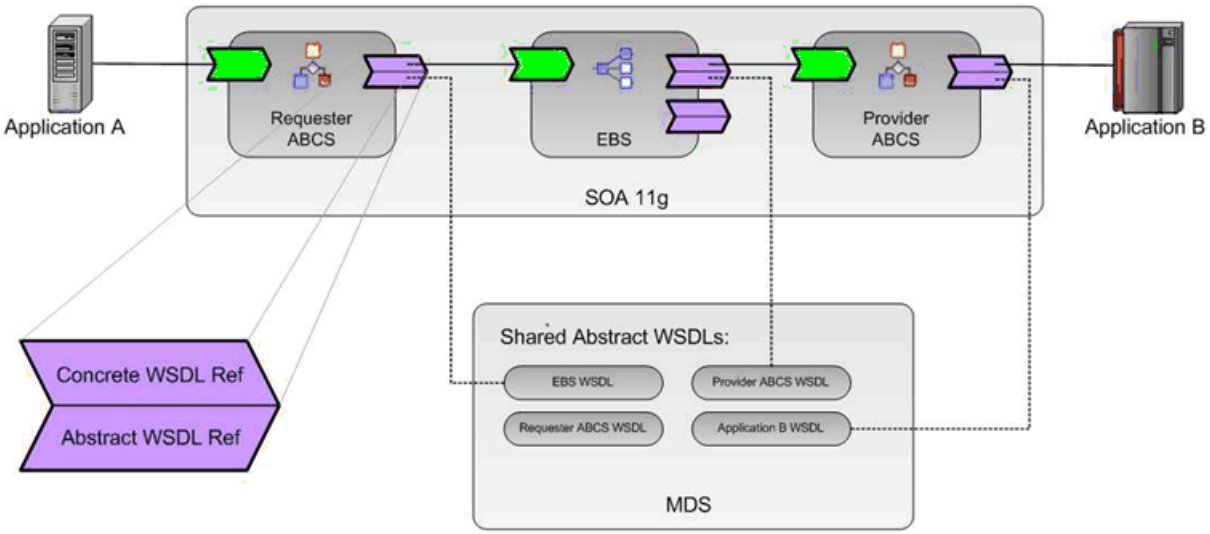
- Do not use the concrete WSDLs directly in your consuming artifacts. If the Service Provider (not as in ProviderABCS) changes, then you must redeploy the consumer.
- Do not reference the deployed concrete WSDL of a Service Provider. If Provider is unavailable, then consumer cannot be compiled or be deployed.

  For example, assume you are building a Provider ABCS that consumes an adapter service, You have directly referenced the concrete WSDL of the adapter service and developed your consuming Provider ABCS. Now, when the adapter service is not available or not deployed as yet, when you are compiling or deploying your Provider ABCS, you hit a road block.
- Introduce a common directory of abstract WSDLs of your services and store these abstract WSDL s on a centrally accessible endpoint.

SOA provides ways to fully de-couple services at design time. AIA advocates always using abstract WSDLs when composites refer to others. Ensure that abstract WSDLs fully describe a service's interface, which is all that is needed at design time. The concrete WSDLs are only needed later at execution time to provide the binding details, that is the information on how the deployed composite can be invoked.

Figure 29–1 illustrates that references between SOA composites actually consist of two parts: an abstract WSDL (design time) and a concrete WSDL (run time).

Figure 29–1 References Between SOA Composites



AIA uses Metadata Services (MDS) for storing each service's abstract WSDL. Thus MDS becomes the source of truth for all service interfaces and the composites should not have any redundant copies.

29.1.6.1 Using MDS as the Central Storage for Abstract WSDLs, and Other Shared Artifacts

SOA Suite 11*g* has introduced a central repository, Metadata Service (MDS), that backs your application (and hence your composites) at design time and run time. MDS is like a version management system, where you can store and use the shared common artifacts at design and run time.

AIA leverages the MDS repository to store common design time artifacts that are shared with all developers. AIA has introduced a common directory of abstract wsdls in the MDS, which is a centrally accessible endpoint.

The artifacts that are stored in the MDS include:

- Schemas - EBO schemas and ABM schemas,
- WSDLs - Abstract WSDLs of EBS, ABCS, Adapter Services, CBPs and EBFs
- DVMs and XREFs.

AIAComponents presents the various schemas and WSDLs referred to by various services. The structure is shown in Table 29–1:

Table 29–1 Structure of AIA Components

Location	Artifacts
ApplicationConnectorServiceLibrary	Abstract WSDLs of various Application Business Connector Services (ABCSs)
ApplicationObjectLibrary	WSDLs of services exposed by applications and schemas of application business objects

Table 29–1 (Cont.) Structure of AIA Components

Location	Artifacts
B2BObjectLibrary	Business-to-business (B2B) schemas
B2BServiceLibrary	Abstract WSDLs of various B2B Connector Services (B2BCSs) and B2B Infrastructure Services
BusinessProcessServiceLibrary	Abstract WSDLs of Composite Business Processes (CBPs) and Enterprise Business Flows (EBFs)
EnterpriseBusinessServiceLibrary	Abstract WSDLs of Enterprise Business Services (EBSs)
EnterpriseObjectLibrary	Schemas of the Oracle AIA Canonical Model
InfrastructureServiceLibrary	Abstract WSDLs of infrastructure services
Transformations	XSLs shared among various services
UtilityArtifacts	Utility schemas and WSDLs
config	AIAConfigurationProperties.xml and AIAEHNotification.xml
dvm	Domain Value Maps
faultPolicies	Default policies applicable to all the services
xref	Metadata for Cross References

For more details on how to upload these artifacts into MDS, and how to update them in the MDS, see Section 2.1.3.4, "Using MDS in AIA"

The abstract WSDLs of all AIA services are stored in the MDS. You build your consumer AIA service composite application using the abstract WSDL of a referenced service.

So where is your MDS located, and how does an application know how to reference it?

You configure your developer environment to point to the MDS. The configuration file is available in $application_home/.adf/META-INF/adf-config.xml and by default it points to your local - JDeveloper file system based MDS. You must configure it to point to the MDS on your managed SOA server.

For details, see Section 2.1.1, "How to Set Up JDeveloper for AIA Development"

In a composite, a referenced service is defined by its abstract WSDL.

As an example, we will use the scenario which is described earlier in this section.

You build a Provider ABCS that consumes an adapter service, by referencing the abstract WSDL of the adapter service.

What does it bring to the table?

First, there is no dependency on the referenced services during the deployment of the composites. Hence there is no dependency on the order of deployment for the composites. A referenced service does not need to be deployed first, in order for it to be referenced by another service. This also resolves the cases where there are cyclic references involved among the services.

After a composite has been designed, you must perform a few tasks. Since you have used an abstract WSDL to reference an external service, the run-time bindings are not generated in the composite. For the referenced service you must open the composite in the source mode and provide this binding information - the values for element binding.ws.

For details about how to populate the binding.ws.port and binding.ws.location elements of your composite, refer to Section 16.1.7.1, "Populating the binding.ws Element in the composite.xml".

29.1.7 Adapters Inside ABCS Composite OR as Separate Composite

The most common and recommended approach to migrate an existing application is to migrate it 1-on-1. This makes every component in the old application an SCA composite. The next step is to use the benefits of SCA; combine multiple SCA composites into a single composite. But do you always gain an advantage by combining composites?

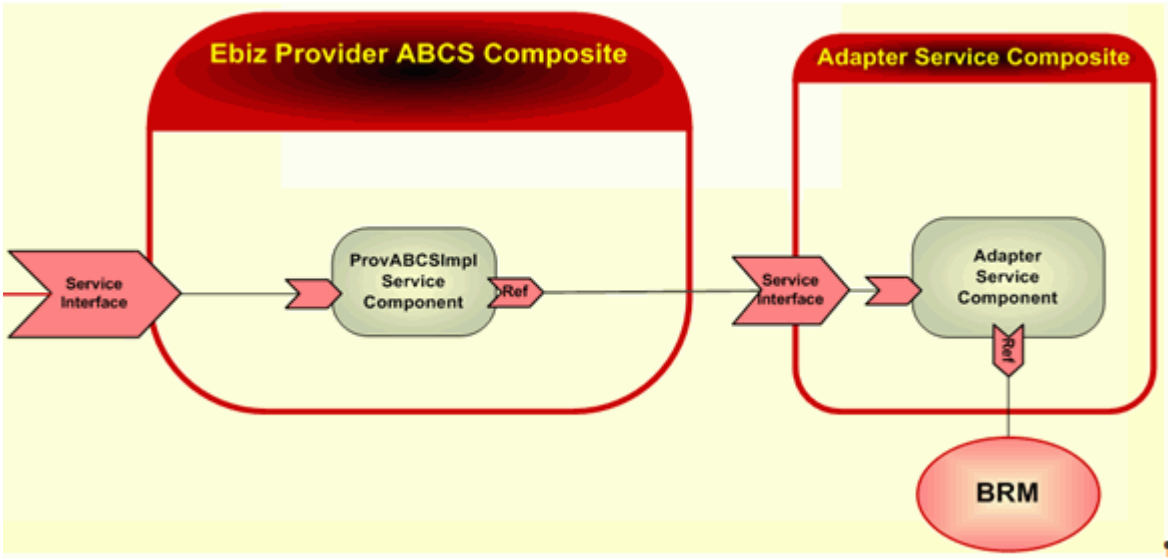
An architectural decision should be made on how you build the SCA. With SOA/SCA it is all about re-usability. Think about this - are you designing your SCA based on the business flow or are you designing our SCA from technical point of view? The answer lies in finding the correct balance.

You might want to design a service once and reuse it many times. From that point of view you would build small SCAs. But you could also choose large SCAs with multiple services that are exposed to the outside world. In that case, since the SCA is relatively large, it increases the maintainability.

What you want is to define and design services and create composites that reference these services.As a result the service is designed once and reused many times.

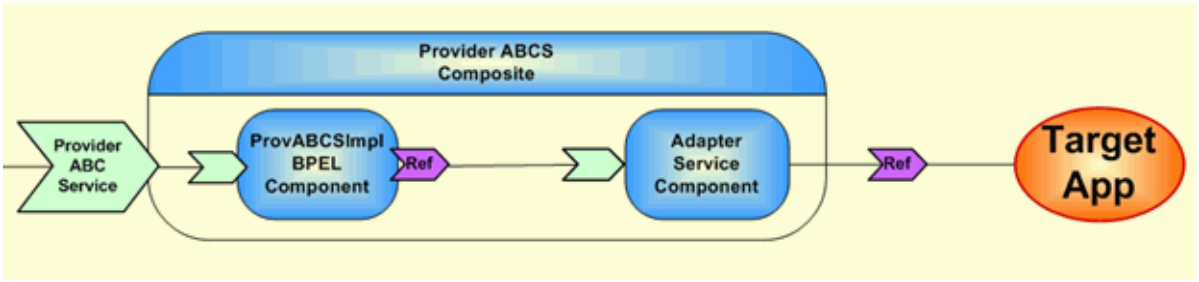
Going back to the question of whether the adapters should be inside the ABCS composite or should be developed as a separate composite, as a best practice, AIA recommends that you put adapters that are interfaced with ABCS in a different composite. as shown in Figure 29–2. This is also the preferred way when the same transport adapter service could be used with multiple ABCS.

Figure 29–2 Example of Adapters Interfaced with ABCS in a Different Composite



However, if no such reusability is foreseen, then there is nothing wrong in using the alternative design where an ABCS and a Transport Adapter service can be in the same composite as shown in Figure 29–3.

Figure 29–3 Example of ABCS and Transport Adapter Service in the Same Composite



29.1.8 AIA Governance

After developed, built and deployed, AIA artifacts must be capable of being governed.

To allow for the governance of shared artifacts such as EBOs, EBS WSDLs, ABM schemas, ABCS WSDLS, DVMs, and XREFs, AIA recommends designing and implementing these artifacts independent of the service's capabilities that use them. They are not to be treated as service artifacts and should be managed and persisted in a central location.

As mentioned in earlier sections, AIA uses MDS to persist these artifacts.

Annotations are injected during the development phase in the composite XML file, apart from annotations in the WSDL file. The annotations in composite files provide detailed information about:

- AIA artifacts and their relationship to other AIA artifacts
- Composite-level descriptor properties, that are used to configure the component at deployment and run time.

AIA Architecture categorizes SOA composites as adapter services, requester services, provider services, and so on. The meta information of these AIA services is used in maintaining OER assets of AIA asset types and linking them to OER assets with native asset types; this is accomplished with the help of the AIA Harvester which harvests the SOA Composites. In line with SOA modeling and development practices, these composites are expected to be harvested multiple times during the development cycle from conception till deployment to production environment.

AIA Harvester is built on top of the Oracle Enterprise Repository Harvester Extension Framework. It introspects SOA artifacts and publishes their ensuing metadata into the Project Lifecycle Workbench back end or Oracle Enterprise Repository (optional), or both, to aid governance and downstream automation.

The best practice for using the AIA Harvester tool is to harvest to both Lifecycle DB and OER. The AIA Harvester then parses the AIA service artifacts and captures metadata into the AIA Project Lifecycle Workbench database and the Oracle Enterprise Repository. The system uses this information to generate deployment plans.

29.1.9 Using AIA Service Constructor

The AIA Service Constructor is an Oracle JDeveloper plug-in used to generate composites conforming to AIA guidelines and naming standards. It also provides guidance for annotating the artifacts to support governance. AIA recommends that you construct the ABCS services using the Service Constructor tool.

For more information about using the AIA Service Constructor, see Chapter 4, "Working with Service Constructor."

29.2 Building Efficient BPEL Processes

This section provides some recommendations on how to keep the BPEL-based processes as lean as possible.

The section includes the following topics:

29.2.1 Using BPEL as "Glue", Not as a Programming Language

Since BPEL is an orchestration language, it should be used primarily as a glue to orchestrate a set of services exposed by the application systems. Even though BPEL does support a wide array of programming constructs, they are not meant for building a complex programming logic within the BPEL process.

29.2.1.1 Keep the Number of BPEL Activities as Minimal as Possible

Use XSL instead of Assign

Avoid the use of multiple assign activities to populate various elements in an XML message. Consider using XSL to do the transformations. Because the invocation of XSL script comes with a cost, usage of XSL for populating the message should be considered only when more than seven to ten assignments must be done.

Use XPath expressions to constrain the data set

Programming scenarios exist in which you must loop through a given array of data (A), and operate on a specific subset of the array (using condition C). So the simple way of doing this is to have a while loop for A, and then a switch condition C inside the while loop. In this approach, you invariably end up looping through all the lines, leading to inefficiency. A better way to approach this situation is to use multi-indexes. So instead of accessing A as A[i] and then checking the condition, you can access A as A[C][i], where you loop through all those elements of A(using i), where condition C is satisfied. This way, you reduce the number of BPEL activities.

29.2.1.2 Avoid Large *While* Loop

A typical usage pattern is the usage of the *while* loop to process:

- Multiple repeating child instances in an XML message
- Large number of discrete object instances in an XML message

In situations in which the *while* loop is being used to process repeating child instances (maxOccurences is unbounded) in an XML message, the BPEL activities in the *while* loop have to be designed keeping the possibility of large number of iterations in mind. For example, having 50 activities to process a single instance in a while loop suddenly

results in creation of 5000 activities at run time when an instance document having 100 repeating instances is processed.

29.2.2 Avoiding Global Variables Wherever Possible

Within the assign activity in BPEL, local variables should be used instead of process variables wherever possible. Local variables are limited to the scope in the BPEL process. These get deleted from memory and from the database after you close the scope. However, the life cycle of a global or process variable is tied to the instance life cycle. These variables stay in memory or disk until the instance finishes. Thus, local variables are preferred to process or global variables. However, if the same variable is being used either in every iteration in the while loop or throughout the entire process, creating one global variable and having that accessed by all iterations would be better.

The BPEL fragment in Example 29–4 illustrates the use of local variables.

Example 29–4 Using Local Variables

```
<scope name="Scope_1">
<variables> <variable name="Invoke_CallprocessBillingMove_InputVariable"
                 messageType=

"sordmovsubabcs:ProcessFulfillmentOrderBillingBRMCommsMoveAddSubProcessRequest
Message"/>
           <variable name="Invoke_CallprocessBillingMove_OutputVariable"
                  messageType=

"sordmovsubabcs:ProcessFulfillmentOrderBillingBRMCommsMoveAddSubProcessResponse
Message"/>
        </variables>
        <sequence name="Sequence_MoveAdd_SubProcess">
                <assign name="Assign_Variable_Invoke_CallMoveAdd">
                      <copy>
                         <from variable="inputSubProcess"

query="/sordsubebo:ProcessFulfillmentOrderBillingBRMCommsSubprocessMessage"/>
<to variable="Invoke_CallprocessBillingMove_InputVariable"
                                    part="payload"
query="/sordsubebo:ProcessFulfillmentOrderBillingBRMCommsSubprocessMessage"/>
</copy>
                 </assign>
               <invoke name="Invoke_CallMove-Add_Subprocess"
partnerLink="ProcessFulfillmentOrderBillingBRMCommsMoveAddSubProcess"
portType="sordmovsubabcs:ProcessFulfillmentOrderBillingBRMCommsMoveAddSubProcess"
operation="processBillingMove"
                           inputVariable="Invoke_CallprocessBillingMove_
InputVariable"
                           outputVariable="Invoke_CallprocessBillingMove_
OutputVariable"/>
          </sequence>
</scope>
```

29.2.3 Avoiding Large FlowN

Careful consideration must be given during design of the BPEL process having FlowN activity. You must have a good understanding of the upper limit of N. Depending on the type of activities performed in a flow, you must strongly consider the option of

running the flows in an asynchronous mode by setting **nonBlockingInvoke property** in composite.xml to *true*.

29.2.4 Controlling the Persistence of Audit Details for a Synchronous BPEL Process

auditLevel property sets the audit trail logging level. This configuration property is applicable to both durable and transient processes. This property controls the number of audit events logged by a process. Audit events result in more database inserts into the audit_level and audit_details tables, which may impact performance. Audit information is used only for viewing the state of the process from Oracle Enterprise Manager Console.

Use the *Off* value if you do not want to store any audit information. Always choose the audit level according to your business requirements and use cases.

For synchronous BPEL processes, AIA recommends non persistence of instance details. For this, set the auditLevel property to *Off* at the service component level. This general guideline can be overridden for individual services based on use cases.

29.2.5 Using Non-Idempotent Services Only When Absolutely Necessary

Idempotent services are retryable. They reproduce the same results regardless of the number of times the service is invoked. They have absolutely no side effects.

The default value for **idempotent property** is *true*. Setting the idempotent property to *false* results in dehydration of the process after running the partnerlink. The decision about the configuration of idempotent property must be done at the design and development time by the developer.

For more information about idempotent property, see *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

29.2.6 Defining the Scope of the Transaction

A transaction tends to grow big when it encompasses a set of activities to process each repeating node in an XML message and when the number of repeating nodes is quite large.

The default value for the **dspMaxThreadDepth** property is set to *600*. If the number of BPEL activities run in a transaction exceeds this value, the BPEL engine issues an automatic implicit commit to end the transaction. This might not be in alignment with the application transaction semantics. This could be eliminated by setting this property to an arbitrarily high value. This approach anticipates that the global transaction as defined by the integration developer ends much before BPEL reaches the new threshold. In most of the situations, setting the property to a very high value helps the transaction to get completed. However, it has the potential to impact the overall scalability of the application. Hence, attempts should be made to keep the scope of the transaction as small as possible.

29.2.7 Disabling the Audit for Synchronous BPEL Process Service Components

AIA recommends turning off the audit completely for synchronous BPEL-based services that have no midprocess breakpoint activities.

auditLevel

This property sets the audit trail logging level and controls the number of audit events that are logged by a process. The value set at the BPEL process service component level overrides the value specified at the SOA Infrastructure, BPEL Process Service

Engine, and Composite Application levels. Override the value only for synchronous BPEL processes that have no midprocess breakpoint activities.

AIA recommends the following value to be set to this property.

Off: The BPEL service engine does not capture the payload. The payload details are not available in the flow audit trails. Payload details for other BPEL activities are collected, except for assign activities. This level is optimal for most normal operations and testing.

Example 29–5 shows how to set the bpel.config.auditLevel property to an appropriate value in the composite.xml file of your SOA project.

Example 29–5 Setting the bpel.config.auditLevel Property in the composite.xml

```
<component name="BPELProcess">
<implementation.bpel src="graphics/BPELProcess.bpel" />
<property name="bpel.config.auditLevel">Off</property>
</component>
```

29.2.8 Including No Break-Point Activity in a Request-Response Flow

AIA highly recommends that a BPEL service implementing the synchronous request-response message exchange pattern has no break-point activity such as *midprocess receive*, *wait*, *onMessage*, *onAlarm*.

Similarly, a mediator service implementing synchronous request-response message exchange pattern should have no parallel routing rules. The ESB routing service implementing the request-response message exchange pattern should have no target services invoked in asynchronous mode.

30 Tuning Integration Flows

To maximize the performance of your integration flows, you must monitor, analyze, and tune all the components. This chapter describes the tools that you can use to monitor performance and the techniques for optimizing the performance of the integration flows and the underlying Oracle Fusion Middleware components.

This chapter includes the following sections:

30.1 Introduction to Tuning

Performance tuning usually involves a series of trade-offs. After you have determined what is causing the bottlenecks, you may have to modify performance in some other areas to achieve the desired results. However, if you have a clearly defined plan for achieving your performance objectives, the decision on what to trade for higher performance is easier because you have identified the most important areas. Tuning is never a one-size-fits-all proposition or a one-off configuration change. Rather, it is an iterative process of monitoring and tuning.

30.1.1 How to Use Baselines

The most effective way to tune is to have an established performance baseline that can be used for comparison if a performance issue arises.

It is important to identify the peak periods at the site and install a monitoring tool that gathers performance data for those high-load times. Optimally, data gathering should be configured when the application is in its initial trial phase during the QA cycle. Otherwise, this should be configured when the system is first in production.

30.1.2 How to Handle Resource Saturation

If any of the hardware resources are saturated (consistently at or near 100% utilization), one or more of the following conditions may exist:

- The hardware resources are insufficient to run the application
- The system is not properly configured.
- The application or database must be tuned.
- There could be some serious problem/bottleneck with in the code/service/process which is consuming more resources or not releasing the resources for a long time.
- There could be a problem with eco systems not responding within the expected time and causing a choking situation.
- There could be situations where unrelated applications / databases belonging to the customer might be running on the same hardware consuming all of the available resources.

Any bottlenecks identified in production should be fixed in the code and a performance improvement patch must be applied.

For a consistently saturated resource, the solution is to reduce load or increase resources. For peak traffic periods when the increased response time is not a solution, consider increasing resources in the form of additional memory or nodes to an existing cluster or determine if there is traffic that can be rescheduled to reduce the peak load, such as throttling at the source or at the middleware or scheduling batch or background operations during slower periods.

30.1.3 How to Use Proactive Monitoring

Proactive monitoring usually occurs on a regularly scheduled interval, where several performance statistics are examined to identify whether the system behavior and resource usage has changed. Proactive monitoring can also be considered as proactive tuning.

Usually, monitoring does not result in configuration changes to the system, unless the monitoring exposes a serious problem that is developing. In some situations, experienced performance engineers can identify potential problems through statistics alone, although accompanying performance degradation is usual.

Experimenting with or tweaking a system when there is no apparent performance degradation as a proactive action can be a dangerous activity, resulting in unnecessary performance drops. Tweaking a system should be considered **reactive tuning**, and the steps for reactive tuning should be followed.

Oracle Fusion Middleware provides a variety of technologies and tools that can be used to monitor Server and Application performance. These tools are described at length in the coming chapters. Monitoring is an important step in performance tuning and enables you to evaluate server activity, watch trends, diagnose system bottlenecks, debug applications with performance problems and gather data that can assist you in tuning the system.

For more information, see *Monitoring Oracle Fusion Middleware*in Oracle Fusion Middleware Administrator's Guide.

Also, refer to *Garbage Collection Configuration* in the Oracle Fusion Middleware Performance and Tuning Guide for monitoring memory by enabling garbage collections.

30.1.4 How to Eliminate Bottlenecks

Tuning usually implies fixing a performance problem. However, tuning should be part of the life cycle of an application-through the analysis, design, coding, production, and maintenance stages. Often the tuning phase is left until the system is in production. At this time, tuning becomes a reactive fire-fighting exercise, where the most important bottleneck is identified and fixed.

System test should be done rigorously against the baseline performance requirements identified during the requirements, analysis, and design phases. System test should be planned with robust test cases considering all the possible scenarios with expected peak loads and peak stress in each use case which helps in eliminating most of the serious bottlenecks. This comes under the preventive tuning category because the problem would be identified before production. Any bottleneck or performance issue identified during production would fall under the reactive tuning category.

The only constraint with the proactive or preventive tuning is that the production-like environment or the deployment topology cannot be predicted during the QA phase though there are some baselines identified during the requirement and analysis phase. Using an appropriate hardware and extrapolation logic helps in addressing this constraint.

During the application's initial trial run, or while entering QA phase, full-fledged efforts should be taken to tune the application and the environment as part of baseline data collection. Rather than deploying all parts of the application and testing the application exactly the way a production environment eventually turns out to be, it would be better to take a bottom-up approach. It is highly recommended to take a single end-to-end flow and optimize it fully before introducing multiple flows into the mix. Even for a single flow, it is better to test it with a single user running the five to ten iterations. You must keep tuning the flow until the average response time and throughput for a specific flow is in adherence to the KPI. Look at the metrics for all of the components participating in the flow; identify the areas where most of the time has been spent and find ways to fix them.

Repeat the above process with multiple concurrent users executing the same flow. Start with small number of concurrent users and gradually increase the number. During this testing phase, in addition to looking at metrics generated by the services, look at how garbage collection is occurring by looking at the garbage collection log. This serves as a key piece of information. This sheds light on how the JVM environment can be configured for better performance and throughput. In addition, the AWR report has to be looked at to check how the database is behaving.

After all the end-to-end flows have been individually optimized to perform according to KPI for concurrent number of users, multiple disparate flows can be tested simultaneously to check whether a specific flow's characteristics has any kind of impact on other flows.

Even though the above section talks about the steps to be carried out in tuning the system before going to production, Oracle AIA emphasizes that monitoring and tuning is a lifelong activity so that the systems are more under control, they are predictable, they are less-fail prone and ofcourse best utilized.

30.1.5 Top Performance Areas

Even though this chapter can be considered a 'Quick Start' guide for fine tuning integration flows, it is not intended to detail a complete list of areas to tune or the techniques on how to identify and fix the issues.

For complete information about fine tuning Fusion Middleware application, see *Oracle Fusion Middleware Performance and Tuning Guide.*

Here is the list of critical Oracle Fusion Middleware performance areas that must be looked at to performance tune your integration flows.

- Oracle database
- SOA dehydration database
  - Configuring database parameters for the most optimal performance
  - Identifying and fixing the bottlenecks
- Java Virtual Machine (JVM)
  - Configuring Garbage Collection to get the most optimal performance
  - Monitoring and Profiling the JVM
- Oracle Fusion Middleware components
  - Configuring FMW components for the most optimal performance
  - Configuring components for concurrency
  - Configuring logging levels
  - Configuring Meta Data Service
- Oracle AIA Process Integration Packs
  - Configuring AIA Services. Pay attention to external contributing factors such as size of payload and latency in edge systems. These factors could certainly play an adverse impact on the overall performance of the PIPs.

The following sections discuss how to configure and tune each of the Oracle Fusion Middleware Performance areas.

30.2 Oracle Database Performance Tuning

This section includes the following topics:

30.2.1 How to Tune the Oracle Database

Oracle Database plays a critical role in the execution of AIA pre-integration packs . Oracle Database is leveraged by SOA Suite and by AIA pre-integration packs for multiple purposes. Apart from the database being used to store the metadata for all of the design-time artifacts, it is being used to persist the state and the audit information

for the in-flight and the completed instances. In addition, the database is used to manage the relationships between the semantically equivalent object instances (Cross Reference data) residing in multiple applications. The Oracle database must be monitored and tuned to get the best performance from AIA applications.

Database administrators must monitor the database (for example, by generating Automatic Workload Repository (AWR) reports for Oracle database) to observe lock contention, I/O usage and take appropriate action to address the issues.

30.2.2 Introducing Automatic Workload Repository

The Automatic Workload Repository (AWR) collects, processes, and maintains performance statistics for problem detection and self-tuning purposes. This data is both in memory and stored in the database. The gathered data can be displayed in both reports and views.

Oracle generates many types of cumulative statistics for the system, sessions, and individual SQL statements. Oracle also tracks cumulative statistics on segments and services. When analyzing a performance problem in any of these scopes, you typically look at the change in statistics (delta value) over the period you are interested in. Specifically, you look at the difference between the cumulative value of a statistic at the start of the period and the cumulative value at the end.

Cumulative values for statistics are generally available through dynamic performance views, such as the V$SESSTAT and V$SYSSTAT views. Ensure that the cumulative values in dynamic views are reset when the database instance is shutdown.

The AWR automatically persists the cumulative and delta values for most of the statistics at all levels except the session level. This process is repeated on a regular time period and the result is called an AWR snapshot. The delta values captured by the snapshot represent the changes for each statistic over the time period.

AWR supports the capture of baseline data by enabling you to specify and preserve a pair or range of AWR snapshots as a baseline. Carefully consider the time period you choose as a baseline; the baseline should be a good representation of the peak load on the system. In the future, you can compare these baselines with snapshots captured during periods of poor performance.

Oracle Enterprise Manager is the recommended tool for viewing both real time data in the dynamic performance views and historical data from the AWR history tables. Oracle Enterprise Manager can also be used to capture operating system and network statistical data that can be correlated with AWR data. For more information, see Oracle Database 2 Day + Performance Tuning Guide.

The statistics collected and processed by AWR include:

- Object statistics that determine both access and usage statistics of database segments
- Time model statistics based on time usage for activities, displayed in the V$SYS_TIME_MODEL and V$SESS_TIME_MODEL views
- Some system and session statistics collected in the V$SYSSTAT and V$SESSTAT views
- SQL statements that are producing the highest load on the system, based on criteria such as elapsed time and CPU time
- Active Session History (ASH) statistics, representing the history of recent sessions activity

30.2.3 Configuring Performance Related Database Initialization Parameters

Below are the minimum basic configurations to be set for the dehydration store.

> **Tip:** This SHOULD be implemented in the staging and production environment.

Table 30–1 provides common **init.ora parameters** and their descriptions. Consider following these guidelines to set the database parameters. One would use either pfile (init[SID].ora) or spfile to manage the parameters. Ultimately, however, the DBA should monitor the database health and tune parameters based on the need.

The values should be considered as starting values. Again these values can vary depending upon the target environment's hardware and software topologies. Additional applications or processes, if any, that might run on the target environment could significantly have an impact on these properties. Further, these values must be adjusted based on the inferences made using the data collected through monitoring tools.

Table 30–1 Common init.ora Parameters

Database Parameter	Recommendstarting values for testing	Description
shared_pool_size	800M	Applicable when SGA Auto Tuning using sga_target and sga_maxsize is not being used. Set the value to as high as possible. Ensure that the sum of the total memory consumed by database and any other applications running on your system along with the operating system do not exceed the amount of available physical RAM. Customer implementations having tens of Giga bytes of shared_pool_size are quite normal. **Note:** If it is a 11g database, consider setting MEMORY_TARGET instead of setting SGA and the PGA separately.

Table 30–1 (Cont.) Common init.ora Parameters

Database Parameter	Recommendstarting values for testing	Description
sga_max_size	1504M	The SGA_MAX_SIZE initialization parameter specifies the maximum size of the System Global Area for the lifetime of the instance. You can dynamically alter the initialization parameters affecting the size of the buffer caches, shared pool, and large pool, but only to the extent that the sum of these sizes and the sizes of the other components of the SGA (fixed SGA, variable SGA, and redo log buffers) does not exceed the value specified by SGA_MAX_SIZE. Set the value to as high as possible. Ensure that the sum of the total memory consumed by database and any other applications running on your system along with the operating system do not exceed the amount of available physical RAM. Customer implementations having tens of Giga bytes of sga_max_size are quite normal. Ensure that you regularly monitor the buffer cache hit ratio and size the SGA so that the buffer cache has an adequate number of frames for the workload. The buffer cache hit ratio may be calculated from data in the view V$SYSSTAT. Also the view V$DB_CACHE_ADVICE provides data that can be used to tune the buffer cache. **Note:** If it is a 11g database, consider setting MEMORY_TARGET instead of setting SGA and the PGA separately.
pga_aggregate_target	800M	Specifies the target aggregate PGA memory available to all server processes attached to the instance. Starting from 11g, set MEMORY_TARGET instead of setting SGA and the PGA separately.
processes	1500	Sets the maximum number of operating system processes that can be connectedto Oracle concurrently. The value of this parameter must account for Oracle background processes. SESSIONS parameter is deduced from this value.
log_buffer (applicable only for 10g database)	30523392	LOG_BUFFER specifies the amount of memory (in bytes) that Oracle uses when buffering redo entries to a redo log file. Redo log entries contain a record of the changes that have been made to the database block buffers. The LGWR process writes redo log entries from the log buffer to a redo log file The goal in sizing *log_buffer* is to set a value that results in the least overall amount of log-related wait events. Common wait events related to a too-small *log_buffer* size include high "*redo log space requests*" and a too-large log_buffer may result in high "*log file sync*" waits.

Table 30–1 (Cont.) Common init.ora Parameters

Database Parameter	Recommendstarting values for testing	Description
db_block_size	8192	**DB_BLOCK_SIZE** specifies (in bytes) the size of Oracle database blocks. The default block size of 8K is optimal for most systems. Set this parameter at the time of database creation.
job_queue_processes	10	**JOB_QUEUE_PROCESSES** specifies the maximum number of processes that can be created for the execution of jobs. It specifies the number of job queue processes per instance
UNDO_MANAGEMENT	AUTO	**UNDO_MANAGEMENT** specifies which undo space management mode the system should use. When set to AUTO, the instance starts in automatic undo management mode. In manual undo management mode, undo space is allocated externally as rollback segments. Starting with Oracle Database 11g Release 1 (11.1), the default value of the `UNDO_MANAGEMENT` parameter is `AUTO` so that automatic undo management is enabled by default. You must set the parameter to `MANUAL` to turn off automatic undo management, if required.
open_cursors	1000	**OPEN_CURSORS** specifies the maximum number of open cursors (handles to private SQL areas) a session can have at once. You can use this parameter to prevent a session from opening an too many cursors. It is important to set the value of OPEN_CURSORS high enough to prevent your application from running out of open cursors. The number varies from one application to another. If a session does not open the number of cursors specified by OPEN_CURSORS, there is no added performance impact to setting this value higher than actually needed. A value of *1000* for open_cursors is a reasonable number to start with.
Sga_target	1504M	Setting this parameter to a nonzero value enables Automatic Shared Memory Management. Consider using automatic memory management, both to simplify configuration and to improve performance. Set the value to as high as possible. Ensure that the sum of the total memory consumed by database and any other applications running on your system along with the operating system do not exceed the amount of available physical RAM. Customer implementations having tens of Giga bytes of SGA_TARGET are quite normal. **Note:** If it is a 11g database, consider setting MEMORY_TARGET instead of setting SGA and the PGA separately.

Table 30–1 (Cont.) Common init.ora Parameters

Database Parameter	Recommendstarting values for testing	Description
MEMORY_TARGET	2500M	**MEMORY_TARGET** specifies the Oracle systemwide usable memory. The database tunes memory to the MEMORY_TARGET value, reducing or enlarging the SGA and PGA as needed.
MEMORY_MAX_TARGET	3000M	**MEMORY_MAX_TARGET** specifies the maximum value to which a DBA can set the MEMORY_TARGET initialization parameter.
Session_cached_cursors	200	**SESSION_CACHED_CURSORS** specifies the number of session cursors to cache. Repeated parse calls of the same SQL statement cause the session cursor for that statement to be moved into the session cursor cache. Subsequent parse calls find the cursor in the cache and do not reopen the cursor. Oracle uses a least recently used algorithm to remove entries in the session cursor cache to make room for new entries when needed. This parameter also constrains the size of the PL/SQL cursor cache which PL/SQL uses to avoid having to re-parse as statements are re-executed by a user. A starting value of 200 is reasonable for a typical AIA system.
TRACE_ENABLED	FALSE	**TRACE_ENABLED** controls tracing of the execution history, or code path, of Oracle. Oracle Support Services uses this information for debugging. Although the performance impact incurred from processing is not excessive, you may improve performance by setting TRACE_ENABLED to *FALSE*.
AUDIT TRIAL	NONE	
NLS SORT	BINARY	

Table 30–1 (Cont.) Common init.ora Parameters

Database Parameter	Recommendstarting values for testing	Description
FAST_START_MTTR_TARGET	3600	Oracle 10g has introduced a new advisory utility that enables you to specify your optimal mean time to recovery (MTTR) recovery interval and uses this to suggest the optimal redo log size. In Oracle 10g the fast_start_mttr_target parameter is used. Oracle recommends using the fast_start_mttr_target initialization parameter to control the duration of startup after instance failure. With 10g, the Oracle database can now self-tune check-pointing to achieve good recovery times with low impact on normal throughput. You no longer have to set any checkpoint-related parameters. This method reduces the time required for cache recovery and makes the recovery bounded and predictable by limiting the number of dirty buffers and the number of redo records generated between the most recent redo record and the last checkpoint. Administrators specify a target (bounded) time to complete the cache recovery phase of recovery with the fast_start_mttr_target initialization parameter, and Oracle automatically varies the incremental checkpoint writes to meet that target.
DISK_ASYNCH_IO	TRUE	DISK_ASYNCH_IO controls whether I/O to datafiles, control files, and logfiles is asynchronous (that is, whether parallel server processes can overlap I/O requests with CPU processing during table scans). If your platform supports asynchronous I/O to disk, it is recommended that you leave this parameter set to its default value. However, if the asynchronous I/O implementation is not stable, you can set this parameter to false to disable asynchronous I/O. If your platform does not support asynchronous I/O to disk, this parameter has no effect. If you set DISK_ASYNCH_IO to false, then you should also set DBWR_IO_SLAVES to a value other than its default of zero to simulate asynchronous I/O

Table 30–1 (Cont.) Common init.ora Parameters

Database Parameter	Recommendstarting values for testing	Description
FILESYSTEMIO_OPTIONS	SETALL	For optimal disk performance, Oracle should always use direct I/O to its data files, bypassing any caching at the OS layer. Direct I/O must be enabled both in Oracle and in the operating system. Oracle controls direct I/O with a parameter named filesystemio_options. filesystemio_options parameter must be set to "setall" (the preferred method, according to the Oracle documentation) or "directio" in order for Oracle to read data blocks directly from disk. Using direct I/O enables you to enhance I/O by bypassing the redundant OS block buffers, reading the data block directly into the Oracle SGA. Using direct I/O also allow you to create multiple blocksized tablespaces to improve I/O performance. Methods for configuring the OS vary depending on the operating system and file system in use

30.2.4 Tuning Redo Logs Location and Sizing

Managing the database I/O load balancing is a non-trivial task. However, tuning the redo log options can provide performance improvement for applications running in an Oracle Fusion Middleware environment, and in some cases, you can significantly improve I/O throughput by moving the redo logs to a separate disk.

The size of the redo log files can also influence performance, because the behavior of the database writer and archiver processes depend on the redo log sizes. Generally, larger redo log files provide better performance by reducing checkpoint activity. It is not possible to provide a specific size recommendation for redo log files, but redo log files in the range of 2g each and atleast 3 redo log groups are considered reasonable. Size your online redo log files according to the amount of redo your system generates. A rough guide is to switch logs at most once every twenty minutes. Set the initialization parameter LOG_CHECKPOINTS_TO_ALERT = TRUE to have\checkpoint times written to the alert file.\The complete set of required redo log files can be created during database creation. After they are created, the size of a redo log size cannot be changed. New, larger files can be added later, however, and the original (smaller) ones can be dropped.

For more information, see *Oracle Fusion Middleware Performance and Tuning Guide*.

30.2.5 Automatic Segment-Space Management (ASSM)

For permanent tablespaces, consider using automatic segment-space management. Such tablespaces, often referred to as bitmap tablespaces, are locally managed tablespaces with bitmap segment space management.

For backward compatibility, the default local tablespace segment-space management mode is *MANUAL*.

For more information, see *Oracle Database Concepts*, "Free Space Management" and *Oracle Database Administrator's Guide* , "Specifying Segment Space Management in Locally Managed Tablespaces."

30.2.6 Tuning Cross Reference Data Table (XREF_DATA)

In situations where cross reference data table has the probability of getting several hundreds of millions of rows over a period, one should seriously consider taking advantage of database schema partitioning feature to distribute the XREF_DATA data. By limiting the amount of data to be examined or operated on, and by providing data distribution for parallel execution, partitioning provides several performance benefits. Partition pruning is the simplest and also the most substantial means to improve performance using partitioning. Partition pruning can often improve query performance by several orders of magnitude. For example, suppose XREF_DATA table containing list of cross references for all logical cross reference tables is partitioned by cross reference table name. A query requesting cross reference information pertaining to an object instance belonging to a particular cross reference table would only access a single partition of the XREF_DATA table.

List partitioning strategy should be employed on XREF_TABLE_NAME. You might want to create named partitions only for cross reference tables that have a large volume. Creating a separate partition for each of these fast growing entities would help in better distribution of data and lesser amount of data access. One could create a partition to hold one or more cross reference tables.

Here is the DDL snippet that shows how partitions can be created. The tablespace names listed are just for reference. Consult your DBAs to identify the right tablespaces.

```
PARTITION BY LIST (XREF_TABLE_NAME) (
PARTITION INSTALLED_PRODUCT_ENTITY VALUES ('INSTALLEDPRODUCT_ID') TABLESPACE TS01,
PARTITION ORDER_ENTITY VALUES ('SALESORDER_LINEID') TABLESPACE TS02,
PARTITION OTHER_DEFAULT VALUES (DEFAULT) TABLESPACE TS03
);
```

Deciding to partition an existing XREF_DATA table in a production environment calls for planned execution. Another table having the same XREF_DATA structure (say XREF_DATA_NEW) must be created with the partitions in place. Data must be copied into the new table from the old table. Completion of data copy must be followed by dropping of existing table and renaming of the newly created table to XREF_DATA. Before performing these tasks, one has to ensure adequate space is available in redo log and in table spaces where the various partitions are created.

30.2.7 Recommendations for managing high-volume BPEL Tables

When the volume of data in the Oracle BPEL Process Manager dehydration store grows very large, maintaining the database can become difficult. Number of BPEL instances being generated in a daily basis and the number of days of retention could play a significant role in impacting the performance and maintenance. For example, generating north of 500 thousand BPEL instances in a day with 5 – 7 days retention could very well result in tables becoming very large very quickly – and hence, be a concern to administrators in terms of performance and maintenance. To address this maintenance challenge, Oracle BPEL Process Manager 10.1.3.5 has been instrumented with partition keys that allow database administrators (DBAs) to take advantage of the Oracle RDBMS partitioning features and capabilities. With the new instrumentation of the BPEL engine, the schema tables can be partitioned using a composite scheme – interval – hash. Please refer to An Oracle White Paper – Oracle BPEL Process Manager 10g Database Schema Partitioning – 133743 to get more information on partitioning strategies.

Partitioning of BPEL tables needs due diligence. The task of partitioning the Oracle SOA Suite tables must be performed by an experienced DBA. Careful analysis and

thought has to put in before implementing the partitioning strategy. Hence, do ensure that sufficient time is given for implementing the task. It is also highly recommended that customer does analysis to decide whether there would be a need to employ partitioning strategy in their environment in a year or two based on projected growth rate; and if yes, it is better to implement it before going to production.

30.2.8 Recommendations for Queue Tables

Reducing contention by spreading queues across table spaces

Great planning must go in the creation of Queue tables. Storage parameters for the Queue tables must be specified when creating a queue table using the storage_clause parameter.

The tablespace of the queue table should have sufficient space to accommodate data from all the objects associated with the queue table. With retention specified, the history table and the queue table can grow to be quite big.

The Data Base Administrator must work with Business Analysts in understanding the behavior of the critical transactions. Understanding the behavior plays a big role in understanding what queues will be accessed concurrently; and what the hotspots will be. Administrators must ensure that not all the queue tables are created in a single tablespace. Efforts should be taken to spread them across multiple tablespaces to mitigate the contention. Refer to Oracle Database SQL Manaual to get a better understanding of storage clauses that must be used to accomplish this.

Limit usual access to a queue from one instance only in the Oracle RAC database

Oracle Real Application Clusters (Oracle RAC) can be used to ensure highly available access to queue data. The entry and exit points of a queue, commonly called its tail and head respectively, can be extreme hot spots. Because Oracle RAC may not scale well in the presence of hot spots, limit usual access to a queue from one instance only. If an instance failure occurs, then messages managed by the failed instance can be processed immediately by one of the surviving instances

You can associate Oracle RAC instance affinities with 8.1-compatible queue tables. You can use ALTER_QUEUE_TABLE or CREATE_QUEUE_TABLE on the queue table and set primary_instance to the appropriate instance_id.

This optional parameter specifies the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. The default value 0 means queue monitor scheduling and propagation is done in any available instance.

Buffered messaging operations in a Real Application Clusters environment will be fastest on the OWNER_INSTANCE of the queue.

The Queue tables have a column called primary_instance – and its default value is 0. The DBA can use either alter table or create table commands to set the primary_instance column in the Queue table to the instance_id of the database instance within an Oracle RAC. With this, enqueueing and dequeueing of messages into / from the queues present in that table happen only in that primary db instance. Failure of that instance leads to processing of messages by one of the surviving database instances within the Oracle RAC.

Here is the PL/SQL Stored procedure that is used to create the queue table.

```
DBMS_AQADM.CREATE_QUEUE_TABLE(
```

```
   queue_table          IN      VARCHAR2,
   queue_payload_type   IN      VARCHAR2,
   [storage_clause      IN      VARCHAR2        DEFAULT NULL,]
  sort_list            IN      VARCHAR2        DEFAULT NULL,
   multiple_consumers   IN      BOOLEAN         DEFAULT FALSE,
   message_grouping     IN      BINARY_INTEGER  DEFAULT NONE,
   comment              IN      VARCHAR2        DEFAULT NULL,
   primary_instance     IN      BINARY_INTEGER  DEFAULT 0,
   secondary_instance   IN      BINARY_INTEGER  DEFAULT 0,
   compatible           IN      VARCHAR2        DEFAULT NULL,
   secure               IN      BOOLEAN         DEFAULT FALSE);
```

primary_instance This optional parameter specifies the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. The default value 0 means queue monitor scheduling and propagation is done in any available instance.

You can specify and modify this parameter only if compatible is 8.1 or higher.

secondary_instance This optional parameter specifies the owner of the queue table if the primary instance is not available. The default value 0 means that the queue table fails over to any available instance.

You can specify and modify this parameter only if primary_instance is also specified and compatible is 8.1 or higher.

The following is the PL/SQL Stored procedure that is used to alter the queue table.

```
DBMS_AQADM.ALTER_QUEUE_TABLE (
   queue_table          IN   VARCHAR2,
   comment              IN   VARCHAR2       DEFAULT NULL,
   primary_instance     IN   BINARY_INTEGER DEFAULT NULL,
   secondary_instance   IN   BINARY_INTEGER DEFAULT NULL);
```

This above procedure alters the existing properties of a queue table.

Table 30–2 Parameters in queue table

Parameter	Description
queue_table	This required parameter specifies the queue table name.
comment	This optional parameter is a user-specified description of the queue table. This user comment is added to the queue catalog.
primary_instance	This optional parameter specifies the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. You can specify and modify this parameter only if compatible is 8.1 or higher.
secondary_instance	This optional parameter specifies the owner of the queue table if the primary instance is not available. You can specify and modify this parameter only if primary_instance is also specified and compatible is 8.1 or higher.

Note: In general, DDL statements are not supported on queue tables and may even render them inoperable. For example, issuing an ALTER TABLE ... SHRINK statement against a queue table results in an internal error, and all subsequent attempts to use the queue table also result in errors. Oracle recommends that you not use DDL statements on queue tables.

Generating table statistics

Process must be in place to gather statistics for Queue Tables. The frequency must be determined based on the data traffic hitting these tables.

Ensure that statistics are being gathered so that the optimal access paths for retrieving messages are being chosen. By default, queue tables are locked out from automatic gathering of statistics. The recommended use is to gather statistics with a representative queue message load and lock them.

The recommendation would be to collect representative statistics on the queue table and lock it so that optimizer uses index for AQ queries. AQ by default does not allow gathering of statistics on the queue table because irrespective of the queue table load it always wants the optimizer to select index. Use dbms_stats package to unlock and collect stats for the queue table.

Refer to Oracle Database Performance Guides for more information.

Establish process for scheduling Queue Maintenance Tasks

The queue table indexes must be coalesced periodically. In 10.2 with automatic space segment management (ASSM), or an online shrink operation may be used for the same purpose. This reduces queue monitor CPU consumption and ensures optimal enqueue & dequeue performance. Please refer to the proper documentation to get information on the following DDL commands

```
Alter table … coalesce
Alter table … shrink space
```

See My Oracle Support Notes 271855.1, 284692.1, 421474.1 for additional details

The above tasks must be run manually before the test runs until the process is established

30.2.9 Recommendations for AIA / BPEL / ESB / AQ tables

Generating database table statistics

A process must be in place to generate the database table statistics at regular intervals. Ensure that statistics are being gathered for the tables that are heavily accessed so that the optimal query plans / access paths are being employed by Cost Optimizer. Statistics must be generated for the Queue tables too.

Statistics must be gathered for all of the tables residing in AIA, orabpel, oraesb, jmsuser and xref schemas. The frequency of statistics gathering can vary anywhere from every 6 hours to once a week. It depends on the volume of inserts, updates & deletes. For example, creation of more than 1 million BPEL instances in a day calls for more frequent statistics gathering.

For queue tables, one has to unlock the statistics before collecting new statistics – and this must be followed by locking the statistics. So one has to ensure that the statistics is being gathered for data population that is representative of a typical day's activity.

No AIA / BPEL / ESB / AQ tables in SYSTEM table space

Ensure that all the queue tables, AIA and BPEL related tables are not in SYSTEM table space. System tablespace is used by Oracle database to data dictionary information. Oracle strongly recommends keeping the user data separate from data dictionary information. This gives you more flexibility in various database administration operations and reduces contention among dictionary objects and schema objects for the same datafiles. AWR & ADDM reports should shed light on the objects that reside in SYSTEM tablespace. You can use multiple tablespaces to:

- control disk space allocation for database data
- assign specific space quotas for database users
- control availability of data by taking individual tablespaces online or offline
- perform partial database backup or recovery operations

Proper space management schemes have to be put in place before moving these artifacts to non SYSTEM tablespace.

Tables / Indices Storage Management

AIA does deliver the SQL script that has the DDL command for creating the table. The SQL script does specify the structural definition for the relational table and the index definitions. However, the DDL script will not have the storage clause (segment and physical attributes) related attributes since they are very closely related to space management – and hence very customer specific.

The table space management design based on characteristics of business transactions, expected growth rate, backup, maintenance and purging strategies, distribution of data leads to identification of the attributes listed below. And the values help you decide whether the default values must be overridden or not.

- number of data tablespaces (single vs many, dictionary managed vs locally managed, read-only tablespace, transactional vs setup, data vs index)
- tablespace in which an object must reside
- Size of the first extent (how much must be allocated for the first extent when Oracle creates this object)
- Size of the second extent
- Rate at which the third and subsequent extents grow over the preceding extent
- Minimum and maximum number of extents for an object
- Percentage of space in each data object reserved for future updates
- Minimum percentage of used space that Oracle maintains for each data block.

Other recommendations

Do not set the db_file_multiblock_read_count parameter (removing it from your spfile or init.ora file) and let the database determine the number of blocks read in multi-block I/O operations.

- In Oracle 10G R2, Oracle defaults the db_file_multiblock_read_count to the maximum number of blocks that can be effectively read. Although this value is also platform-dependent, Oracle documentation states that it is 1 MB for most

platforms. This 1 MB size allows much more data to be read in a single operation in 10GR2 than previous releases.

- The DB_FILE_MULTIBLOCK_READ_COUNT parameter controls the number of blocks pre-fetched into the buffer cache during scan operations, such as full table scan and index fast full scan. Oracle Database 10g Release 2 automatically selects the appropriate value for this parameter depending on the operating system optimal I/O size and the size of the buffer cache.

30.2.10 Recommendations for Securefiles migration (applicable only to 11g R1 / R2)

In 11g R1, Oracle has introduced a new LOB storage mechanism called Oracle SecureFiles. It enables File system-like performance for LOBs. In general, Oracle customers have seen performance improvements of 3 – 6x, with some seeing even larger gains, by migrating to SecureFiles to BasicFiles. To get these gains, data stored in the older format must be migrated to the SecureFile storage format.

There are several ways to migrate applications to SecureFiles. Oracle recommends Online Redefinition as the preferred method of migration as it allows the database and the table being migrated to be online during the migration process. Explaining the process is beyond the scope of this document. Please refer to 'SecureFiles Migration – An Oracle White Paper – August 2008' to get complete information.

Migration of LOB structures to SecureFiles needs due diligence. The task of migrating to SecureFiles from BasicFiles must be performed by an experienced DBA. Careful analysis and thought has to put in before implementing the SecureFiles migration. Hence, do ensure that sufficient time is given for implementing the task. It is also highly recommended that customer does analysis to decide whether adopting SecureFiles is a right strategy for their implementation before implementing it.

For more information see:: "Generating Automatic Workload Repository Reports" in Oracle Database Performance Tuning Guide

"Monitoring Performance" in Oracle Database Administrator's Guide.

30.2.11 Changing the Driver Name to Support XA Drivers

If your data sources require support for XA drivers, you must change the driver name on Oracle WebLogic Server. This is particularly true for environments in which BPEL processes assume XA is present when calling database adapters and JMS adapters.

Change the driver name using one of the following methods:

30.2.11.1 Edit in Oracle WebLogic Server Administration Console

To edit in Oracle WebLogic Server Administration Console

1. Log in to Oracle WebLogic Server Administration Console.
2. In the left pane, select *Domain Structure*.
3. Select *Services* > *JDBC* > *Data Source* > *SOADataSource* > *Connection Pool*.
4. For the **Driver Class Name**, change the value to oracle.jdbc.xa.client.OracleXADataSource.

   This provides support for the XA driver.
5. Restart the server.

30.2.11.2 Edit the SOADataSource-jdbc.xml file

To edit the SOADataSource-jdbc.xml file

1. Open the soaDataSource-jdbc.xml file on Oracle WebLogic Server.
2. Change the SOADataSource driver name from

   oracle.jdbc.OracleDriver to *oracle.jdbc.xa.client.OracleXADataSource*.

Example 30–1 soaDataSource-jdbc.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<jdbc-data-source
/. . .
. . .
/ <name>SOADataSource</name>
<jdbc-driver-params>
<url>jdbc:oracle:thin:@adc60086fems.us.oracle.com:1537:co0yd570</url>
<driver-name>*oracle.jdbc.xa.client.OracleXADataSource*</driver-name>
<properties>
<property>
<name>user</name>
<value>fusion_soainfra</value>
</property>
</properties>
/ . . .
. . ./
</jdbc-driver-params>
/. . .
. . ./
</jdbc-data-source>
```

30.2.12 Configuring Database Connections and Datasource Statement Caching

By properly configuring the connection pool attributes in JDBC data sources in your WebLogic Server domain, you can improve application and system performance. The following sections include information about tuning options for the connection pool in a JDBC data source:

30.2.12.1 JDBC Datasource Connection Pool Settings

- Statement Cache Type

  The Statement Cache Type (or algorithm) determines which prepared and callable statements to store in the cache for each connection in a data source.AIA recommends the property to set to '*LRU*'

- Statement Cache Size

  The Statement Cache Size attribute determines the total number of prepared and callable statements to cache for each connection in each instance of the data source. By caching statements, you can increase your system performance. However, you must consider how your DBMS handles open prepared and callable statements. In many cases, the DBMS maintains a cursor for each open statement. This applies to prepared and callable statements in the statement cache.

  AIA recommends keeping the default value of *10*. If you cache too many statements, you may exceed the limit of open cursors on your database server. For example, if you have a data source with 50 connections deployed on 2 servers, if

you set the Statement Cache Size to 10 (the default), you may open 1000 (50 x 2 x 10) cursors on your database server for the cached statements.

- Initial Capacity and Maximum Capacity

  For applications that use a database, performance can improve when the connection pool associated with a data source limits the number of connections. You can use the Maximum Capacity to limit the database requests from Oracle Application Server so that incoming requests do not saturate the database, or to limit the database requests so that the database access does not overload the Oracle Application Server-tier resource.

  The connection pool *MaxCapacity* attribute specifies the maximum number of connections that a connection pool allows. By default, the value of MaxCapacity is set to 15. For best performance, you should specify a value for MaxCapacity that matches the number appropriate to your database performance characteristics.

  Limiting the total number of open database connections to a number your database can handle is an important tuning consideration. You should check to ensure that your database is configured to allow at least as large a number of open connections as the total of the values specified for all the data sources MaxCapacity option, as specified in all the applications that access the database.

  AIA recommends setting Initial and Maximum Capacity to 50 to start with; and make the necessary adjustments based on the database performance.

30.2.12.2 Getting the Right Mix of Performance and Fault Tolerance

These four options can be used to get the right mix of performance and fault tolerance for your system.

- Test Frequency

  Enable periodic background connection testing by entering the number of seconds between periodic tests. . If the request is made within the time specified for Seconds to Trust an Idle Pool Connection, since the connection was tested or successfully used by an application, WebLogic Server skips the connection test.

- Test Reserved Connections

  If Test Reserved Connections is enabled on your data source, when an application requests a database connection, WebLogic Server tests the database connection before giving it to the application. If the request is made within the time specified for Seconds to Trust an Idle Pool Connection, since the connection was tested or successfully used by an application, WebLogic Server skips the connection test before delivering it to an application.

- Seconds to Trust an Idle Pool Connection

  Seconds to Trust an Idle Pool Connection is a tuning feature that can improve application performance by minimizing the delay caused by database connection testing, especially during heavy traffic. However, it can reduce the effectiveness of connection testing, especially if the value is set too high. The appropriate value depends on your environment and the likelihood that a connection becomes defunct.

- Remove Infected Connections Enabled

  Specifies whether a connection will be removed from the connection pool after the application uses the underlying vendor connection object. If you disable removing infected connections, you must ensure that the database connection is suitable for reuse by other applications.

When set to *true* (the default), the physical connection is not returned to the connection pool after the application closes the logical connection. Instead, the physical connection is closed and re-created.

When set to *false*, when the application closes the logical connection, the physical connection is returned to the connection pool and can be reused by the application or by another application.

For more information, see:

- "Generating Automatic Workload Repository Reports" in *Oracle Database Performance Tuning Guide*
- "Monitoring Performance' in *Oracle Database Administrator's Guide*
- "JDBC Data Source: Configuration: Connection Pool" in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Online Help*
- "Tuning Data Source Connection Pool Options" in *Oracle Fusion Middleware Configuring and Managing JDBC for Oracle WebLogic Server*

30.2.13 Oracle Metadata Service (MDS) Performance Tuning

For optimal performance of MDS APIs, the database schema for the MDS repository must be monitored and tuned by the database administrator. This section lists some recommended actions to tune the database repository:

- Collect schema statistics
- Increase redo log size
- Reclaim disk space
- Monitor the database performance

30.2.13.1 Using Database Polling Interval for Change Detection

MDS employs a polling thread which queries the database to gauge if the data in the MDS in-memory cache is out of sync with data in the database. This can happen when metadata is updated in another JVM. If it is out of sync, MDS clears any out of date-cached data so subsequent operations see the latest versions of the metadata.

MDS invalidates the document cache and the MDS cache, so subsequent operations have the latest version of the metadata.

The polling interval can be configured or changed post deployment through MBeans. The element maps to the ExternalChangeDetection and ExternalChangeDetectionInterval attributes of the MDSAppConfig MBean.

Before packaging the Enterprise ARchive (EAR) file, you can configure the polling interval by adding this entry in adf-config.xml:

Example 30–2 Configuring the Polling Interval in the adf-config.xml

```
<mds-config>
<persistence-config>
<external-change-detection enabled="true" polling-interval-secs="T"/>
</persistence-config>
</mds-config>
```

In Example 30–2, 'T' specifies the polling interval in seconds. The minimum value is 1. Lower values cause metadata updates that are made in other JVMs, to be seen more quickly. It is important to note, however, that a lower value can also create increased

middle tier and database CPU consumption due to the frequent queries. By default, polling is enabled ('true') and the default value of 30 seconds should be suitable for most purposes. Consider increasing the value to a higher number if the number of updates to MDS are few and far between.

For more information, see "Changing MDS Configuration Attributes for Deployed Applications" in *Oracle Fusion Middleware Administrator's Guide*.

30.2.13.2 Tuning Cache Configuration

MDS uses a cache to store metadata objects and related objects (such as XML content) in memory. MDS Cache is a shared cache that is accessible to all users of the application (on the same JVM). If a metadata object is requested repeatedly, with the same customizations, that object may be retrieved more quickly from the cache (a "warm" read). If the metadata object is not found in the cache (a "cold" read), then MDS may cache that object to facilitate subsequent read operations depending on the cache configuration, the type of metadata object and the frequency of access.

Cache can be configured or changed post deployment through MBeans. This element maps to the MaximumCacheSize attribute of the MDSAppConfig mbean.

For more information, see "Changing MDS Configuration Attributes for Deployed Applications" in *Oracle Fusion Middleware Administrator's Guide.*

Having a correctly sized cache can significantly improve throughput for repeated reading of metadata objects. The optimal cache size depends on the number of metadata objects used and the individual sizes of these objects. Before packaging the Enterprise ARchive (EAR) file, you can manually update the cache-config in adf-config.xml, by adding the entry shown in Example 30–3.

Example 30–3 Manually Updating the cache-config in adf-config.xml

```
<mds-config>
<cache-config>
<max-size-kb>200000</max-size-kb>
</cache-config>
</mds-config>
```

For more information about tuning the MDS and the database, see "Optimizing Instance Performance" in *Oracle Database Performance Tuning Guide,* and "Oracle Metadata Service (MDS) Performance Tuning" in *Oracle Fusion Middleware Performance and Tuning Guide.*

30.3 Configuring the Common SOA Infrastructure

This section discusses properties that impact the entire SOA infrastructure.

30.3.1 Configuring SOA Infrastructure Properties

These settings apply to all SOA Composite applications running in the SOA infrastructure. The properties set at this level impact all deployed SOA composite applications, except those composites for which you explicitly set different audit level values at the composite application or service engine levels. AIA recommends configuring at least the following properties.

auditLevel

This property sets the audit trail logging level. This property controls the amount of audit events that are logged by a process.

AIA recommends the following value to be set to this property:

- **Production:** The BPEL service engine does not capture the payload. The payload details are not available in the flow audit trails. Payload details for other BPEL activities are collected, except for assign activities. This level is optimal for most normal operations and testing.

captureCompositeInstanceState

Enabling this option may result in additional run time overhead during instance processing. This option provides for separate tracking of the running instances. All instances are captured as either running or not running. This information displays later in the *State* column of the composite instances tables for the SOA Infrastructure and SOA composite application, where it shows the counts of running instances versus total instances. You can also limit the view to running instances only

Valid states are *Running*, *Completed*, *Faulted*, *Recovery Needed*, *Stale*, *Terminated*, *Suspended*, and *State Not Available*.

The *Running* and *Completed* states are captured only if this check box is selected. Otherwise, the state is set to *Unknown*. The conditional capturing of these states is done mainly to reduce the performance overhead on SOA Infrastructure run time.

Note: If this property is disabled and you create an instance of a SOA composite application, an instance is created, but the instance does not display as running, faulted, stale, suspended, terminated, completed, or requiring recovery in the table of the Dashboard page of the composite application. This is because capturing the composite state of instances is a performance-intensive process.

AIA recommends disabling this property.

PayloadValidation

This property validates incoming and outgoing XML documents. If set to *True*, the SOA Infrastructure applies schema validation for incoming and outgoing XML documents.

This property is applicable to both durable and transient processes. The default value is *False*. This value can be overridden to turn on schema validation based on business needs. Turning on schema validation both at SOA Infrastructure and at the BPEL Process Service engine causes the schema validation to be done twice.

30.3.2 Disabling HTTP Logging

To disable HTTP logging from Weblogic Console for Admin, SOA and BAM servers

1. Login to Weblogic Console
2. Environments -> Servers -> Admin Server -> Logging -> HTTP -> Uncheck HTTP access log file enabled option

To change log level to error (From Oracle Enterprise Manager):

1. Login to EM
2. SOA Infrastructure -> Logs -> Log Configuration

30.4 BPEL - General Performance Recommendations

This section includes the following topics:

30.4.1 Configuring BPEL Process Service Engine Properties

The basic BPEL Process Manager performance tuning properties can be configured using WLST or Oracle Enterprise Manager. These properties must be configured:

auditLevel

This property sets the audit trail logging level. This property controls the amount of audit events that are logged by a process.

AIA recommends the following value to be set to this property.

- **Production:** The BPEL service engine does not capture the payload. The payload details are not available in the flow audit trails. Payload details for other BPEL activities are collected, except for assign activities. This level is optimal for most normal operations and testing.

instanceKeyBlockSize

This property controls the instance ID range size. Oracle BPEL Server creates instance keys (a range of process instance IDs) in batches using the value specified. After creating this range of in-memory IDs, the next range is updated and saved in the ci_id_range table. For example, if instanceKeyBlockSize is set to 100, Oracle BPEL Server creates a range of instance keys in-memory (100 keys, which are later inserted into the cube_ instance table as cikey). To maintain optimal performance, ensure that the block size is larger than the number of updates to the ci_id_range table.

The default value is *10000*. AIA recommends the default value to be used as is.

auditDetailThreshold

This property sets the maximum size (in kilobytes) of an audit trail details string before it is stored separately from the audit trail. If an audit trail details string is larger than the threshold setting, it is not immediately loaded when the audit trail is initially retrieved; a link is displayed with the size of the details string. Strings larger than the threshold setting are stored in the audit_details table, instead of the audit_trail table. The details string typically contains the contents of a BPEL variable. In cases where the variable is very large, performance can be severely impacted by logging it to the audit trail.

The default value is *50000* (50 kilobytes). AIA recommends retaining the default value.

LargeDocumentThreshold

This property sets the large XML document persistence threshold. This is the maximum size (in kilobytes) of a BPEL variable before it is stored in a separate location from the rest of the instance scope data. This property is applicable to both durable and transient processes. Large XML documents impact the performance of the entire Oracle BPEL Server if they are constantly read in and written out whenever processing on an instance must be performed.

The default value is 10000 (100 kilobytes). AIA recommends changing the value to *50000*.

PayloadValidation

This property validates incoming and outgoing XML documents. If set to *True*, the Oracle BPEL Process Manager applies schema validation for incoming and outgoing XML documents.

This property is applicable to both durable and transient processes. The default value is *False*. This value can be overridden to turn on schema validation based on business needs. Turning on schema validation both at SOA Infrastructure and at the BPEL Process Service engine causes the schema validation to be done twice.

DispatcherInvokeThreads (dspInvokeThreads in 10*g*)

This property specifies the total number of threads allocated to process invocation dispatcher messages. Invocation dispatcher messages are generated for each payload received and are meant to instantiate a new instance. If the majority of requests processed by the engine are instance invocations (as opposed to instance callbacks), greater performance may be achieved by increasing the number of invocation threads. Higher thread counts may cause greater CPU utilization due to higher context switching costs.

The default value is 20. AIA recommends this value as the starting point. This parameter can be used throttle the requests to avoid overloading the mid-tier, database tier, and participating application systems and tune for best performance.

DispatcherEngineThreads (dspEngineThreads in 10*g*)

This property specifies the total number of threads allocated to process engine dispatcher messages. Engine dispatcher messages are generated whenever an activity must be processed asynchronously. If the majority of processes deployed are durable with a large number of dehydration points (mid-process receive, onMessage, onAlarm, and wait activities), greater performance may be achieved by increasing the number of engine threads. Higher thread counts can cause greater CPU utilization due to higher context switching costs.

The default value is 30. Since the majority of AIA flows do not have dehydration points, AIA recommends the value to be set to *20*. This parameter can be used to throttle the requests to avoid overloading the mid-tier, database tier, and participating application systems and tune for best performance.

DispatcherSystemThreads (dspSystemThreads in 10*g*)

This property specifies the total number of threads allocated to process system dispatcher messages. System dispatcher messages are general clean-up tasks that are typically processed quickly by the server (for example, releasing stateful message beans back to the pool). Typically, only a small number of threads are required to handle the number of system dispatch messages generated during run time.

The default value is 2. AIA recommends the default value is kept as is.

Disable BPEL Monitors and Sensors

Select this check box to disable all BPEL monitors and sensors defined for all BPEL components across all deployed SOA composite applications.

syncMaxWaittime

This property sets the maximum time the process result receiver waits for a result before returning. Results from asynchronous BPEL processes are retrieved synchronously by a receiver that waits for a result from Oracle BPEL Server. This property is applicable to transient processes.

The default value is 45. Value of syncMaxWaittime parameter mainly depends on the scenario and the number of concurrent processes. Even though one can increase the time to avoid the transaction time out, one has to realize that end user experience will be adversely impacted after increasing the value beyond a certain point.

StatsLastN

This property sets the size of the most-recently processed request list. After each request is finished, statistics for the request are kept in a list.

A value less than or equal to 0 disables statistics gathering. This property is applicable to both durable and transient processes.

For more information, see "Configuring BPEL Process Service Engine Properties" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

30.4.2 Configuring BPEL Properties Inside a Composite

This section lists the config properties of some sections of the deployment descriptor.

For each configuration property parameter, a description is given, and the expected behavior of the engine when it is changed. All the properties set in this section affect the behavior of the component containing the BPEL composite only. Each BPEL process can be created as a component of a composite. These properties are modified through WLST.

inMemoryOptimization

This property indicates to Oracle BPEL Server that this process is a transient process and dehydration of the instance is not required. When set to True, Oracle BPEL Server keeps the instances of this process in memory only during execution. This property can only be set to True for transient processes or processes that do not contain any dehydration points such as mid-process receive, wait, onMessage and onAlarm activities. This property has the following values:

- *False* (default): instances are persisted completely and recorded in the dehydration store database.
- *True*: Oracle BPEL Process Manager keeps instances in memory only.

AIA recommends setting the value to *True* for transient BPEL processes. BPEL processes that have request-response pattern with no dehydration points should have this property set to *True*. Setting this property to true and setting completionPersistPolicy to *faulted* ensures that successfully completed transient processes will not be persisted.

completionPersistPolicy

This property configures how much of instance data should be saved. It can only be set at the BPEL component level.

AIA recommends setting the property to *Off* for transient services. In that situation, no instances of process are saved. It persists only the faulted instances.

PayloadValidation

This property is set at the partnerlink level. This property validates incoming and outgoing XML documents.

If set to *True*, the Oracle BPEL Process Manager applies schema validation for incoming and outgoing XML documents. When set to True the engine validates the XML message against the XML schema during <receive> and <invoke> for this

partnerLink. If the XML message is invalid then bpelx:invalidVariables run time BPEL Fault is thrown. This overrides the domain level validateXML property.

The default value is *False*. This value can be overridden to turn on schema validation based on business needs.

30.4.3 How to Monitor the BPEL Service Engine

The request and thread statistics for all BPEL process service components running in the service engine must be monitored regularly to check whether the above properties must be tweaked. The statistics page for the BPEL engine provides valuable information. These details help determine how the BPEL engine is performing:

- Details on Pending requests in the service engine
- Active requests in the service engine
- Thread statistics for the service engine

The statistics page also provides information about the count, minimum, average, and maximum processing times for each of the tasks. This information helps the developer and the administrator identify the hot spots.

For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite.*

30.5 Oracle Mediator: General Performance Recommendations

This section includes the following topics:

- Section 30.5.1, "Configuring Mediator Service Engine Properties"
- Section 30.5.2, "How to Monitor the Mediator Service Engine"

30.5.1 Configuring Mediator Service Engine Properties

You can configure the properties of Mediator Service Engine by setting the parameters mentioned in this section. These parameters can be set by setting the values of the parameters in the Mediator Service Engine Properties page. These properties must be configured:

auditLevel

This property sets the audit trail logging level. This property controls the amount of audit events that are logged by a process.

AIA recommends the following value to be set to this property.

- **Production:** All events are logged. All audit details, except the details of assign activities, are logged. Instance tracking information is collected, but payload details are not captured and these details are not available in the flow audit trails. This level is optimal for most typical operations and testing.

metricsLevel

Administrator can set the Mediator-specific flag metricsLevel for configuring the Dynamic Monitoring Service (DMS) metrics level. DMS metrics are used to measure the performance of application components. The possible values of this flag are:

- *Enabled* - Enables DMS metrics tracking
- *Disabled* - Disables DMS metrics tracking

AIA recommends disabling the DMS metrics tracking in production. Consider turning this on when there is a need for tracking DMS metrics.

DeferredWorkerThreadCount

Specifies the number of deferred dispatchers for processing messages in parallel. For higher loads consider increasing this parameter to have more outbound threads for deferred processing as each parallel rule is processed by one of the DeferredWorkerThreads.

Default value is 4 threads. AIA recommends changing the value to a higher number only when there are a significant number of mediator services having parallel routing rules.

DeferredMaxRowsRetrieved

When the Mediator routing rule type is set to *'Parallel'*, DeferredMaxRowsRetrieved sets the number of maximum rows (maximum number of messages for parallel routing rule processing) that are retrieved from Mediator store table (that stores messages for parallel routing rule) for processing.

Ensure that each message retrieved in this batch is processed by one worker thread at a time. The default value is 20 threads. AIA recommends altering the value only when there are a significant number of mediator services having parallel routing rules.

DeferredLockerThreadSleep

For processing parallel routing rules, Oracle Mediator has a daemon locker thread that retrieves and locks messages from Mediator store database. The thread polls the database to retrieve messages for parallel processing. When no messages are available, the locker thread "sleeps" for the amount of time specified in the DeferredLockerThreadSleep and prevents round trips to database.

Default value is 2 seconds. Consider increasing the value to a higher number, say 120 seconds, for integrations having no mediator components with parallel routing rules.

During the specified time, no messages are available for parallel routing in either of the following cases:

- There are no Mediator components with parallel routing rules deployed.
- Mediator component(s) with parallel routing rule is deployed, but there are no continuous incoming messages for such components.

SyncMaxWaitTime

The maximum time a request and response operation takes before timing out.

Default value is 45 seconds.

30.5.2 How to Monitor the Mediator Service Engine

The efficiency level of the Mediator service engine can be assessed by monitoring the following:

- Routing statistics
- Request breakdown statistics.
- Instance statistics

The request breakdown statistics provide information about the count and the time taken for processing each of the following actions:

- Invoke one-way
- Enqueues
- Transformation
- Invoke request-response
- Publish
- Condition evaluation
- Message validation

Keeping the number of time consuming actions such as enqueues, message validations, transformations as minimal as possible is critical for improving the performance and scalability.

For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite.*

30.6 Tuning Oracle Adapters for Performance

Oracle technology adapters integrate Oracle Application Server and Oracle Fusion Middleware components such as Oracle BPEL Process Manager (Oracle BPEL PM) or Oracle Mediator components to file systems, FTP servers, database queues (advanced queues, or AQ), Java Message Services (JMS), database tables, and message queues (MQ Series).

This section describes how to tune Oracle Adapters for optimal performance.

This section focuses only on the tuning aspect for technology adapters that are used by AIA to integrate with applications.

For more information about Oracle Adapters, see *Oracle Fusion Middleware User's Guide for Technology Adapters.*

30.6.1 How to Tune JMS Adapters

This section describes some properties that can be set for the Oracle SOA JMS Adapter to optimize performance.

For more information, see "Introduction to the Oracle JMS Adapter" in *Oracle Fusion Middleware User's Guide for Technology Adapters.*

If during stress it is taking time to read the messages off the queue, then the JMS adapter thread shown in Example 30–4 can be increased to get the desired output. The following parameter must be set in the bpel.xml property file to achieve the above purpose:

To improve performance, the *adapter.jms.receive.threads* property can be tuned for an adapter service. The default value is 1, but multiple inbound threads can be used to improve performance. When specified, the value of *adapter.jms.receive.threads* is used to spawn multiple inbound poller threads.

Be very cautious about increasing the value for this property. Increasing the value increases the load for the Fusion Middleware Server and puts additional stress on the downstream systems such as application systems and databases. Consider increasing the value only when the CPU is underutilized.

Example 30–4 shows the parameter that must be set in the composite.xml property file to achieve the above purpose.

Example 30–4 Tuning JMS Adapters Using the composite.xml Property File

```
<activationAgents>
<activationAgent className="…" partnerLink="MsgQueuePL">
... <property name="adapter.jms.receive.threads">5</property>
</activationAgent>
</activationAgents>
<activationAgents>
<activationAgent className="…" partnerLink="MsgQueuePL">
... <property name="adapter.jms.receive.threads">5</property>
</activationAgent>
```

30.6.2 How to Tune AQ Adapters

If during stress it is taking time to read the messages off the queue, then the AQ adapter thread shown in Example 30–5 can be increased to get the desired output. The following parameter must be set in the bpel.xml property file to achieve the above purpose:

To improve performance, the *adapter.aq.receive.threads* property can be tuned for an adapter service. The default value is *1*, but multiple inbound threads can be used to improve performance. When specified, the value of *adapter.aq.receive.threads* is used to spawn multiple inbound poller threads.

Be very cautious about increasing the value for this property. Increasing the value increases the load for the Fusion Middleware Server and puts additional stress on the downstream systems such as application systems and databases. Consider increasing the value only when the CPU is underutilized.

Example 30–5 shows the parameter that must be set in the composite.xml property file to achieve the above purpose.

Example 30–5 Tuning AQ Adapters Using the composite.xml Property File

```
<service name="dequeue" ui:wsdlLocation="dequeue.wsdl">
<interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/aq/raw/raw/dequeue/#wsdl.interfa
ce(Dequeue_ptt)"/>
<binding.jca config="dequeue_aq.jca">
<property name="adapter.aq.dequeue.threads" type="xs:string"
many="false">5</property>
</binding.jca>
</service>
<service name="dequeue" ui:wsdlLocation="dequeue.wsdl">
```

minimumDelayBetweenMessages

There could be situations where downstream / back office application will not be able to keep up pace with the SOA Server – and this could result in more requests being sent to the application than it could process. Hence, this could result in longer time being taken to process the requests, invariably resulting in transaction timeouts. In these situations, the following parameter could be used for throttling - the tuning parameter that you should use is:

<property name="**minimumDelayBetweenMessages**">1000</property>

This value can be tuned (increase/reduce) further based on the actual time required to complete your transaction.

This property is available for JMS based end points only within BPEL based services.

30.6.3 How to Tune Database Adapters

Consider the following properties when tuning database adapters:

- MaxTransactionSize

  This property controls the number of records processed per transaction by each thread. This should be set to a reasonable value, such as 100.

- NumberOfThreads

  This property controls how many threads the DB adapter uses to process records. The default is 1.

  Reducing the numProcessorThread (setting numProcessorThread=4) may improve performance in Core2 Duo boxes but not for Netburst boxes.

- UseBatchDestroy

  This property controls how the processed records are updated (for example, Deleted for DeletePollingStrategy, MarkedProcessed for LogicalDeleteStrategy). If set, only one update or delete is executed for all the rows part of that transaction. The number of rows in a transaction is controlled by the MaxTransactionSize option.

- MaxTransactionSize
  - If set to a large value such as 1000, turning on the UseBatchDestroy option could have a negative impact on performance.
  - Setting a large MaxTransactionSize and a small MaxRaiseSize could have negative impact on performance.
  - Maintaining 10:1 ratio could give better performance.

30.6.4 Throttling Inbound Message Flows

The setting in *11g* should be configured as a binding property in the composite.xml for the corresponding (inbound) <service> as shown in Example 30–6.

Example 30–6 Binding Property in the composite.xml

```
<service name="Inbound">
interface.wsdl interface="http://xmlns.oracle.com/pcbpel/demo1#wsdl.interface
(SampleInbound_PortType)"/>
<binding.jca config="Dequeue_jms.jca">

<property name="minimumDelayBetweenMessages">1000</property>

</binding.jca>
</service>
```

This setting ensures that there is at least 1000 milli seconds delay between two consecutive messages being posted to the BPEL process.

> **Note:** This setting pertains only to one adapter polling thread. If multiple adapter polling threads (for example, multiple JMS dequeuer threads) have been configured, this setting controls the speed of each thread, not the combined speed.

30.7 Purging the Completed Composite Instances

Data growth from auditing and dehydration can have a significant impact on database performance and throughput. See the sections on **auditLevel** for audit configuration and **inMemoryOptimization** for dehydration configuration.

These tables are impacted by instance data growth:

- Audit_trail
- Audit_details
- Cube_instance
- Cube_scope
- Dlv_message
- Dlv_subscription
- Document_ci_ref
- Document_dlv_message_ref
- Schema_md
- Task
- Work_item
- Xml_document
- Header_properties

The database administrator must continuously monitor the growth of these tables and make sure these tables are configured properly to give the optimal performance.

In addition, AIA highly recommends having a process in place to archive and clean up the completed composite instances after a specified period.

- Synchronous request -response based completed composite instances should be purged from SOA dehydration data store after a month.
- Data synchronization related composite instances should be purged 3 - 4 months after their completion.
- Composite Business Processes should be purged 1 - 2 years after their completion.

Even though this duration is subject to change based on your company's policies, attempts should be made to do the purging at the earliest possible time. Purging must be preceded by the archiving of the instance data.

For more information about performing this task, refer to "Deleting Large Numbers of Instances with the Purge Script" in *OracleFusion Middleware Administrator's Guide for Oracle SOA Suite* to get complete information on how to perform this task.

30.8 Tuning Java Virtual Machines (JVMs)

This section discusses how to optimize the JVM heap.

30.8.1 How to Optimize the JVM Heap - Specifying Heap Size Values

The goal of tuning your heap size is to minimize the time that your JVM spends doing garbage collection while maximizing the number of clients that the Fusion Middleware stack can handle at a given time.

Specifically the Java heap is where the objects of a Java program live. It is a repository for live objects, dead objects, and free memory. When an object can no longer be reached from any pointer in the running program, it is considered "garbage" and ready for collection. A best practice is to tune the time spent doing garbage collection to within 5% of execution time.

The JVM heap size determines how often and how long the virtual system spends collecting garbage. An acceptable rate for garbage collection is application-specific and should be adjusted after analyzing the actual time and frequency of garbage collections. If you set a large heap size, full garbage collection is slower, but it occurs less frequently. If you set your heap size in accordance with your memory needs, full garbage collection is faster, but occurs more frequently.

In production environments, set the minimum heap size and the maximum heap size to the same value to prevent wasting virtual system resources used to constantly grow and shrink the heap. Ensure that the sum of the maximum heap size of all the JVMs running on your system does not exceed the amount of available physical RAM. If this value is exceeded, the Operating System starts paging and performance degrades significantly. The virtual system always uses more memory than the heap size. The memory required for internal virtual system functionality, native libraries outside of the virtual system, and permanent generation memory (memory required to store classes and methods) is allocated in addition to the heap size settings.

To tune the JVM garbage collection options you must analyze garbage collection data and check for the frequency and type of garbage collections, the size of the memory pools, and the time spent on garbage collection.

Before you configure JVM garbage collection, analyze the following data points:

- How often is garbage collection taking place? Compare the time stamps around the garbage collection.
- How long is a full garbage collection taking?
- What is the heap size after each full garbage collection? If the heap is always 85% free, for example, you might set the heap size smaller.
- Do the young generation heap sizes (Sun) or Nursery size (Jrockit) need tuning?

You can manually log garbage collection and memory pool sizes using verbose garbage collection logging:

Sun JVM command line options:

```
-verbose:gc
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
```

Jrockit JVM command line options:

```
-XXverbose:gc
```

For example, you can use the following JVM options to tune the heap:

- If you run out of heap memory (not due to a memory leak), increase -Xmx.
- If you run out of native memory, you must decrease -Xmx.
- For Sun JVM, modify -Xmn to tune the size of the heap for the young generation.
- For Oracle JRockit, modify -Xns:<nursery size> to tune the size of the nursery.

- If your AIA service instance runs on a dedicated host, set the heap size value as high as possible but make sure it does not exceed the available physical RAM.

It is highly recommended that the testing and production environment have 64-bit JVM running on 64-bit hardware with atleast 8 GB of memory. The following configurations are based on assumption that the environment has atleast 6 GB of memory. It also assumes that the host system has sufficient physical RAM installed and that other OS processes are not consuming a significant portion of the available memory.

Table 30–3 JVM Heap Size Values

JVM Parameters	Recommended Starting Value for Testing	Notes
Initial heap size	4096	It is recommended that Initial Heap and Maximum Heap are set to the same value for the proper memory utilization and minimize major garbage collection. A small heap becomes full quickly and must be garbage collected more often. It is also prone to more fragmentation, making object allocation slower. A large heap introduces a slight overhead in garbage collection times. A heap that is larger than the available physical memory in the system must be paged out to disk, which leads to long access times or even application freezes, especially during garbage collection. In general, the extra overhead caused by a larger heap is smaller than the gains in garbage collection frequency and allocation speed, as long as the heap does not get paged to disk. Thus a good heap size setting would be a heap that is as large as possible within the available physical memory.
Maximum heap size	4096	
Enable J2SE 5.0 Platform Mbeans	Enable	Keep it enabled, so that memory and cpu real time monitoring can be done and further tuning can be done if required.
-XX:PermSize	256	This setting is used to allocate memory outside of heap space to hold java classes. Note: JRockit JVM does not have this capability.
-XX:MaxPermSize	256	
-XX:AppendRatio	3	None

Table 30–3 (Cont.) JVM Heap Size Values

JVM Parameters	Recommended Starting Value for Testing	Notes
-XX:NewSize	1638	Another important heap configuration is the garbage collector's generational settings. The garbage collector optimizes collection by classifying objects by how long they live. Most of the BPEL engine's objects are short lived, thus they live in the "Eden" space. AIA recommends sizing the "Eden" space to be 40-50% of the total maximum heap size. The following command line starts Java with a Eden sizing that is 60% of the maximum heap size. Also If you are using 2 or more than 2 CPUs and it is a non-Windows system, it is recommended to use -XX:+ AggressiveOpts jvm flag. The -XX:+ AggressiveOpts option inspects the system resources (size of memory and number of processors) and attempts to set various parameters to be optimal for long-running, memory allocation-intensive jobs. Note this flag does not affect Windows performance. In JRockit JVM, young (new) generation is called nursery size - and it is specified using -Xns<size>
-XX:MaxNewSize	1638	
-XX:SurvivorRatio	6	
-XX:+AggressiveOpts	Set in the JVM argument	

Example 30–7 shows sample JVM configurations for 64-bit Sun JVM running on 64-bit Linux on 64-bit hardware having 16GB RAM. Settings such as heap space (-mx & -ms), young generation (MaxNewSize & NewSize), permanent size (MaxPermSize), ParallelGCThreads must be adjusted according to the memory and processors available in your environment. Similarly, directory paths for several of the directives listed below must be adjusted accordingly.

Example 30–7 Sample JVM Configurations

```
<data id="java-options" value="
-d64
-server -Doc4j.jms.implementation=oracle.j2ee.jms
-Dcom.sun.management.jmxremote
-mx8192M
-ms8192M
-XX:MaxPermSize=512M
-XX:MaxNewSize=2450M
-XX:NewSize=2450M
-Xmn2500M
-XX:SurvivorRatio=6
-XX:AppendRatio=3
-verbose:gc
-XX:+AggressiveOpts
-XX:+HeapDumpOnOutOfMemoryError
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
-XX:+UseParallelOldGC
-XX:ParallelGCThreads=8
-Xloggc:/export/oracle/product/10.1.3.1/OracleAS_1/opmn/logs/j2eegc.log
-Djava.security.policy=$ORACLE_HOME/j2ee/OC4J_SOA/config/java2.policy
-Djava.awt.headless=true
-Dhttp.webdir.enable=false
```

```
-Doc4j.userThreads=true
-Doracle.mdb.fastUndeploy=60
-Doc4j.formauth.redirect=true
-Djava.net.preferIPv4Stack=true
-Dorabpel.home=/export/oracle/product/10.1.3.1/OracleAS_1/bpel
-Xbootclasspath/p:/export/oracle/product/10.1.3.1/OracleAS_1/bpel/
   lib/orabpel-boot.jar
-Dhttp.proxySet=false
-Doraesb.home=/export/oracle/product/10.1.3.1/OracleAS_1/integration/esb
-Dhttp.proxySet=false -Daia.home=/export/oracle/AIA" />
```

30.8.1.1 Java Performance Analysis Tools

A profiler is a performance analysis tool that enables you to reveal hot spots in the application that result in either high CPU utilization or high contention for shared resources. Some common profilers are:

- OptimizeIt Java Performance Profiler at http://www.borland.com/optimizeit/optimizeit_profiler/index.html from Borland, a performance debugging tool for Solaris and Windows
- JProbe Profiler with Memory Debugger at http://www.quest.com/jprobe/, a family of products that provide the capability to detect performance bottlenecks, perform code coverage and other metrics
- Hewlett Packard JMeter at http://www.hp.com/products1/unix/java/hpjmeter/, a tool for analyzing profiling information
- VTune Performance Analyzer at http://www.intel.com/software/products/vtune/, a tool to identify and locate performance bottlenecks in your code
- PerformaSure at http://www.quest.com/performasure/, a tool to detect, diagnose, and resolve performance problems in multitier J2EE applications

30.9 Tuning Weblogic Application Server

This section discusses how to optimize the Weblogic server for high performance.

30.9.1 Domain Startup Mode - Production

The Weblogic server must be created using production mode. This sets many properties like log level to minimum to optimize the Weblogic server performance.

30.9.2 Work Manager - default

To handle thread management and perform self-tuning, WebLogic Server implements a default Work Manager. This Work Manager is used by an application when no other Work Managers are specified in the application's deployment descriptors.

In many situations, the default Work Manager WebLogic Server's thread-handling algorithms assign each application its own fair share by default. Applications are given equal priority for threads and are prevented from monopolizing them.

30.9.3 Tuning Network I/O

Download and enable performance pack for the specific operating system where Weblogic server is installed.

Additional Reading

- Document on Tuning Garbage Collection with the 5.0 Java[tm] Virtual Machine
- An article on Monitoring and Managing Java SE 6 Platform Applications
- Oracle Database VLDB and Partitioning Guide 11.1 B32024-1
- SecureFiles Migration – An Oracle White Paper – August 2008

31

Oracle AIA Naming Standards for AIA Development

This chapter provides an overview of guidelines for naming standards for various AIA components for AIA development.

This chapter includes the following sections:

31.1 General Guidelines

The goal is to ensure that the intent of each artifact is clear, and that the text associated with the artifact conveys as much information as possible given the space constraints. The following should be applied whenever applicable:

- Avoid abbreviations

  Abbreviations may be ambiguous. The names used must be spelled out. Do not abbreviate unless the object name becomes too long.

- Artifact names must be alphanumeric

  Names must be composed only of alphanumeric character with these rules:

  Word comprising a name should be concatenated without spaces in an upper camel-case fashion.

  Example: *Purchase Order*

Avoid using numeric characters in the name unless it is required to convey some business meaning.

No special characters such as spaces, '-', '_', '.', '$', '%', '#', []

- Indicate artifact type in the name to reduce ambiguity

  When the same name is used for different artifact types, append a suffix to indicate its type. It makes it easier to distinguish these artifacts by identifying their types:

  <Artifact Name><Type Suffix>

  Example: *InvoiceEBO, InvoiceEBOType, InvoiceEBM, InvoiceEBMType*.

- The total path length to a named artifact must not exceed 17000 characters.

  Some operating systems such as Windows have a limit of 255 characters for file names. Some room is left for prefixing with complete network directory or URL.

31.1.1 XML Naming Standards

The following standards are based on UN/CEFACT - XML Naming and Design Rules.

31.1.1.1 General Naming Standards

Follow these general naming standards:

- Lower-Camel-case must be used for naming attributes.

  Example: *<xsd:attribute name="unitCode"/>*

- Upper-Camel-case must be used for naming elements and types.

  Example: *<xsd:element name="UnitOfMeasure"/> <xsd:complexType name="InvoiceEBOType"/>*

- Names must be singular unless the concept itself is plural.

  For example repeating elements must have a singular name.

- Names must not contain special characters such as: space, '-', '_', '.', '$', '%', '#', ...

- Avoid having numeric characters in the name.

  There are cases were using a numeric character is required to convey some significance.

- Complex type names should end with the 'Type' suffix to help recognize types from elements.

  Example: *<xsd:complexType name="InvoiceEBOType"/>*

- The name of a simple type definition should be the name of the root element with the 'ContentType' suffix.

  Example: *<xsd:simpleType name="PhoneNumberContentType">*

31.1.1.2 General Namespace Naming Standards

These are the general namespace naming rules. More detailed rules are described in the following sections, especially naming rules for EBS and ABCS.

- All namespaces must start with **http://xmlns.oracle.com/**.

- Namespaces used by Enterprise Business Objects (EBOs) and Enterprise Business Messages (EBMs) start with **http://xmlns.oracle.com/EnterpriseObjects/**.

Example: *http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V1*

- Namespaces used for externally facing services must start with **http://xmlns.oracle.com/EnterpriseServices/**.

  Examples: *http://xmlns.oracle.com/EnterpriseServices/Core/Invoice/V1*
  http://xmlns.oracle.com /EnterpriseServices/Industry/Telco/Invoice/V1

- Namespaces for versioned artifacts must have the major version number as a suffix with 'V' as an abbreviation for 'version'.

  Example: *http://xmlns.oracle.com/ EnterpriseObjects/Core/EBO/Invoice/V1*

- The namespace structure should closely map to the taxonomy of the types it encapsulates.

  Example: Horizontal: *http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V1*.

  Telco: *http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/EBO/Invoice/V1*.

- Namespaces for artifacts generated within ABCSs must start with: **http://xmlns.oracle.com/ABCS/**.

- When importing or including schema in a schema file, the schema location must always use relative path.

  Example:
  <xsd:importnamespace="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V1" schemaLocation="../../../../Core/EBO/Invoice/InvoiceEBO .xsd"/>

- Namespace prefixes must be a minimum of six (6) lowercase characters abbreviation of the namespace.

  The abbreviation must be descriptive and unambiguous within the context where it is being used.

- Namespace prefixes for EBOs and EBMs must adhere to the following standard wherever used regardless of the applications or technology used.

  Auto-generated prefixes such as ns1, ns2 must not be used. Auto-generated prefixes for standard namespaces such as xsd, xsi are acceptable.

Table 31–1 provides details on the namespace patterns and files associated with namespace prefixes.

Table 31–1 Namespace Prefixes

Prefixes	Namespace Pattern	Files
corecomcust	http://xmlns.oracle.com/EnterpriseObjects/Core/Custom/Common/V1	CustomCommonComponents, CustomReferenceComponents
coreinvcust	http://xmlns.oracle.com/EnterpriseObjects/Core/Custom/EBO /Invoice/V1	CustomInvoiceEBO
corecom	http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V1	CommonComponents. Reference Components
coreinv	http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V1	InvoiceEBO
coreinv	http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V1	InvoiceEBM

Table 31–1 (Cont.) Namespace Prefixes

Prefixes	Namespace Pattern	Files
telcocomcust	http://xmlns.oracle.com/EnterprisObjects/Industry/Telco/Custom/Common/V1	CustomCommonComponents, CustomReferenceComponents
telcoinvcust	http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/Custom/EBO/Invoice/V1	CustomInvoiceEBO
telcocom	http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/Common/V1	CommonComponents. Reference Components
telcoinv	http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/EBO/Invoice/V1	InvoiceEBO
telcoinv	http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/EBO/Invoice/V1	InvoiceEBM

31.1.1.3 Participating Applications Names

When participating application names are part of an artifact name, upper-camel-case short names should be used. For cases where an abbreviation is needed, an upper-case abbreviation should be used.

Table 31–2 provides a list of short names and abbreviations used for participating applications.

Table 31–2 Short Names and Abbreviations for Participating Application Names

Application	Short Name	Abbreviation
Oracle E-Business Suite	Ebiz	EBIZ
Oracle Siebel	Siebel	SEBL
Oracle PeopleSoft	PeopleSoft	PSFT
Oracle JD Edwards Enterprise One	JDEOne	JDE1
Oracle JD Edwards World	JDEWorld	JDEW
Oracle Transportation Management Suite	Logistics	LOGIS
Oracle Telephony @Work	Telephony	TELE
Oracle Demantra	Demantra	DMTR
Oracle Communications Billing and Revenue Management	Portal	PORTAL
Oracle Retail Applications	Retek	RETEK

31.2 Composites

A **composite** is unit of deployment for SCA and contains service components.

The name of the composite is the same as the name of the Oracle Application Integration Architecture (AIA) artifact.

The composites are created for the following AIA artifacts:

- Composite Business Process
- Enterprise Business Flow
- Enterprise Business Service
- Application Business Connector Service
- Utility Services

Examples: *SalesOrderOrchestrationProcess*, *InterfaceCustomerToFulfillmentEBF*, *CreateCustomerEBS*, *CreateOrderSiebelReqABCSImpl*, *AsyncErrorHandlerService*

31.3 Composite Business Process

Composite business processes are long running SOA BPEL orchestration processes.

Table 31–3 provides details about the naming standards for composite business processes.

Table 31–3 Naming Standards for Composite Business Processes

Artifact	Name	Example
Service Name (in the WSDL)	[Optional Industry] [Verb][EBO Name]CBP	OrchestrateSalesOrderCBP, TelcoResolveComplaintCBP
Namespace	http://xmlns.oracle.com/CompositeProcess/{Core/ or Industry/[Industry Name]}/{EBO name}/V{VersionNumber}	http://xmlns.oracle.com/CompositeProcess /Core/SalesOrder/V1 http://xmlns.oracle.com/CompositeProcess /Industry/Telco/SalesOrder/V1
Namespace Prefix	Follow the namespace prefix standard.	coreordprocess, telordprocess, …
WSDL	[Service Name].wsdl	OrchestrateSalesOrderCBP.wsdl TelcoResolveComplaintCBP.wsdl
WSDL Message	Request message: [Verb][EBO Name]ReqMsg Response message: [Verb][EBO Name]RespMsg	OrchestrateSalesOrderReqMsg, OrchestrateSalesOrderRespMsg
WSDL Port Type	[Service Name]Service	OrchestrateSalesOrderCBPService, TelcoResolveComplaintCBPService
WSDL Port Type Operations	[Verb][EBO Name]	OrchestrateSalesOrder
BPEL Flow Name	[Service Name]V[Version Number]	ProcessSalesOrderCBP, TelcoVerifyCustomerCBPV2

31.4 Enterprise Business Services

Enterprise Business Services (EBS) are SOA Mediator Routing Services with routing rules to invoke the appropriate composite business processes, Enterprise Business Flows, and Application Business Connector Services.

Table 31–4 provides details about the naming standards for Enterprise Business Services.

Table 31–4 Naming Standards for Enterprise Business Services

Artifact	Name	Example
Service Name [in the WSDL]	[Optional Industry Name][EBO Name]EBS	OrderEBS, CustomerEBS, TelcoInvoiceEBS
Namespace	http://xmlns.oracle.com/EnterpriseServices/{Core/ or Industry/[Industry Name]}/{EBO name}/V{VersionNumber}	http://xmlns.oracle.com/EnterpriseServices/Industry/Telco/Invoice/V1 http://xmlns.oracle.com/EnterpriseServices/Core/Invoice/V2
Namespace Prefix	Follow the namespace prefix standard.	coreinv, telcoinv, …
WSDL	[Service Name].wsdl	CreateOrderEBS.wsdl, UpdateCustomerEBS.wsdl, TelcoProcessInvoiceEBS.wsdl
WSDL Message	Request message: [Verb][EBO Name]ReqMsg Response message: [Verb][EBO Name]RespMsg	CreateOrderReqMsg CreateOrderRespMsg
WSDL Port Type	Request - Response EBS or Request Only EBS : [Industry name][ServiceName][version number] Response EBS in one-way call: [industry name][Service Name]Response[Version number]	CreateOrderEBS, CreateOrderEBSResponse, TelcoProcessInvoiceEBS, TelcoProcessInvoiceEBSResponse
WSDL Port Type Operations	Request - Response EBS or Request Only EBS : [Verb][EBO Name] Response EBS in one-way call: [Verb][EBO Name]Response	CreateOrder, CreateOrderResponse
Mediator Routing Service Name	[Service Name]V[Version Number]	CreateOrderEBS, CreateOrderEBSV2, UpdateCustomerEBSV2, TelcoProcessInvoiceEBSV2

31.5 Enterprise Business Flows

Table 31–5 provides details about the naming standards for BPEL flows used primarily to interact with multiple Enterprise Business Services.

Table 31–5 Naming Standards for Enterprise Business Flows

Artifact	Name	Example
Service Name (in the WSDL)	[Optional Industry] [Verb][EBO Name]EBF	ProcessOrderEBF, TelcoVerifyCustomerEBF

Table 31–5 (Cont.) Naming Standards for Enterprise Business Flows

Artifact	Name	Example
Namespace	http://xmlns.oracle.com/EnterpriseFlows/{Core/ or Industry/[Industry Name]}/{EBO name}/V{VersionNumber}	http://xmlns.oracle.com/EnterpriseFlows /Core/Order/V1 http://xmlns.oracle.com/EnterpriseFlows /Industry/Telco/Order/V1
Namespace Prefix	Follow the namespace prefix standard.	coreordflow, telordflow, …
WSDL	[Service Name].wsdl	ProcessOrderEBF.wsdl, TelcoVerifyCustomerEBF.wsdl
WSDL Message	Request message: [Verb][EBO Name]ReqMsg Response message: [Verb][EBO Name]RespMsg	ProcessOrderReqMsg, ProcessOrderRespMsg
WSDL Port Type	[Service Name]Service	ProcessOrderEBFService, VerifyAccountEBFService
WSDL Port Type Operations	[Verb][EBO Name]	ProcessOrder
BPEL Flow Name	[Service Name]V[Version Number]	ProcessOrderEBF, TelcoVerifyCustomerEBFV2

31.6 Application Business Connector Service

Application Business Connector Services are of two categories: Requester and Provider.

31.6.1 Requester Application Business Connector Service

Requester ABCS are the services that serve requests coming from client applications and process these requests and delegate them to the EBSs.

Table 31–6 provides details about the naming standards for requester ABCS.

Table 31–6 Naming Standards for Requester ABCS

Artifact	Name	Example
Service Name (in the wsdl)	[Verb][App Entity Name][Short Application Name][Optional Industry]ReqABCSImpl	CreateOrderSiebelReqABCSImpl UpdateOrderSiebelTelcoReqABCSImpl CreateCustomerPortalTelcoReqABCSImpl
Namespace	http://xmlns.oracle.com/ABCSImpl/{Participating Application Name}/{Core/ or Industry/[Industry Name]}/{App Entity Name}/V{version}	http://xmlns.oracle.com/ABCSImpl/Siebel/Core/Order/V1 http://xmlns.oracle.com/ABCSImpl/Portal/Industry/Telco/ Customer/V2
Namespace Prefix	Follow the namespace prefix standard.	abcsimplsieblelinv, abcsimplportalprod, …

Table 31–6 (Cont.) Naming Standards for Requester ABCS

Artifact	Name	Example
WSDL	[Service Name].wsdl	CreateOrderSiebelReqABCSImpl.wsdl UpdateOrderSiebelTelcoReqABCSImpl.wsdl CreateCustomerPortalTelcoReqABCSImpl.wsdl
WSDL Message	Request message: Verb[ABO Name]ReqMsg Response message: Verb[ABO Name]RespMsg	CreateOrderReqMsg, CreateOrderRespMsg UpdateOrderLinesReqMsg, UpdateOrderLinesRespMsg
WSDL Port Type	BPEL: [Service Name]Service Mediator: [Service Name]	CreateOrderSiebelReqABCSImplService UpdateOrderSiebelReqABCSImpl
WSDL Port Type Operations	Verb[ABO Name] Should be self-describing to reflect the functionality needed by the application.	UpdateOrder, GetOrderLines, …
BPEL Flow Name/ Mediator Routing Service Name	[Service Name]V[Version Number]	CreateOrderSiebelReqABCSImplV2Process UpdateOrderSiebelTelcoReqABCSImplProcess CreateCustomerPortalTelcoReqABCSImplProcess

31.6.2 Provider Application Business Connector Services

Provider ABCS are the implementation of Enterprise Business Services.

Table 31–7 provides details about the naming standards for provider ABCS.

Table 31–7 Naming Standards for Provider ABCS

Artifact	Name	Example
Service Name (in the WSDL)	[Verb][App Entity Name][Short Application Name][Optional Industry]ProvABCSImpl	CreateOrderSiebelProvABCSImpl UpdateOrderSiebelTelcoProvABCSImpl CreateCustomerPartyTelcoPortalProvABCSImpl
Namespace	http://xmlns.oracle.com/ABCSImpl/{Participating Application Name}/{Core/ or Industry/[Industry Name]}/{App Entity Name}/V{version}	http://xmlns.oracle.com/ABCSImpl/Siebel/Core/Order/V1 http://xmlns.oracle.com/ABCSImpl/Portal/Industry/Telco/ CustomerParty/V2
Namespace Prefix	Follow the namespace prefix standard.	abcsimplsieblelinv, abcsimplportalprod, …
WSDL	[Service Name].wsdl	CreateOrderSiebelProvABCSImpl.wsdl UpdateOrderSiebelTelcoProvABCSImpl.wsdl CreateCustomerPartyTelcoPortalProvABCSImpl.wsdl

Table 31–7 (Cont.) Naming Standards for Provider ABCS

Artifact	Name	Example
WSDL Message	Request message: [Verb][ABO Name]ReqMsg Response message: [Verb][ABO Name]RespMsg	CreateOrderReqMsg, CreateOrderRespMsg
WSDL Port Type	BPEL: [Service Name]Service Mediator: [Service Name]	CreateOrderSiebelProvABCSImplService UpdateOrderSiebelProvABCSImpl
WSDL Port Type Operations	[Verb][ABO Has one operation which should match the operations exposed by the corresponding EBS.	UpdateOrder, getOrderLines, …
BPEL Flow Name/ Mediator Routing Service Name	[Service Name]V[Version Number]	CreateOrderSiebelProvABCSImplV2Process UpdateOrderSiebelTelcoProvABCSImplProcess CreateCustomerPortalTelcoProvABCSImplProcess

31.7 JMS and Adapters

Table 31–8 provides details about the naming standards for JMS and Adapters.

Table 31–8 Naming Standards for JMS and Adapters

Artifact	Name	Example
JMS Store, JMS Server, JMS Module	Use the default configuration created by AIA foundation pack	N/A
JNDI Name for Queue or Topic Connection Factory	eis/wlsjms/AIA<Application Name>CF OR eis/wlsjms/AIACommonCF (For common queues or topics used as milestones before/after EBS or EBF) For XA connection factory, follow the below convention: eis/wlsjms/AIA<Application Name>XACF	eis/wlsjms/AIAPeopleSoftCF For XA: eis/wlsjms/AIAPeopleSoftXACF Oracle AIA Foundation Pack creates the JMS module level connection factory automatically by appending "jms/aia" to the given connection factory name. Example:jms/aia/AIAPeopleSoftCF

Table 31–8 (Cont.) Naming Standards for JMS and Adapters

Artifact	Name	Example
Queue or Topic Name	AIA_<ApplicationName><EBO / ABO Name><Optional Industry>JMSQueueV<Version Number >	AIA_PeopleSoftCurrencyExchangeJMSQueue AIA_SiebelCustomerPartyJMSTopicV1 Notes: ■ Here the "JMSQueue" indicates "WLS JMS Queue" by default. For other JMS providers, a prefix should be added like AqJMSQueue, TibJMSQueue...and so on. ■ While coming up with the queue/topic names, you could use the meaningful short names or abbreviations in the following use cases: Long Application/Industry name. Long EBO/ABO name. JMS provider having a restriction on the number of characters. For example: AQ JMS provider has 24 characters limitation on queue name. ■ Use short form "JMSQ" instead of "JMSQueue" and "JMST" in the above cases. Based on the JMS provider's maximum characters limitation on their queue name, it is recommended to come up with a suitable & meaningful queue or topic name by following short names or abbreviations of application/industry and EBO/ABO names. Example: AIA_PSFTCurExchangeJMSQ, AIA_SEBLCustPartyJMSTV1 ■ Oracle AIA Foundation Pack creates the JNDI names for queue or topic under JMS module during deployment by appending "jms/aia". Example: jms/aia/AIA_SiebelCustomerPartyJMSTopicV1
	AIA_<ApplicationName><EBO / ABO Name><Optional Industry>JMSTopicV<Version Number >	
Error Queues	<Queue Name>_ErrorQ	FP scripts use the resource name from annotations to get the queue/topic name and then append "_ErrorQ" as suffix. Example:- For "AIASalesOrderQueue", FP generates "AIASalesOrderQueue_ErrorQ"
Table Name for AQJMS provider	AIA_<App><ABO/EBO><optional industry>JMSQTABV[Version Number]	AIA_SiebelCustomerProvJMSQTAB AIA_EbizOrderProvJMSQTABV2

31.7.1 AQ JMS (Additional Attributes)

Table 31–9 provides details about the naming standards for AQ JMS additional attributes.

Table 31–9 Naming Standards for AQ JMS Additional Attributes

Artifact	Name	Example
Database User/Schema for queues and queue tables creation in AQJMS provider	JMSUSER	N/A
Table Name for AQJMS provider	AIA_<App><ABO/EBO><optional industry>JMSQTABV[Version Number]	AIA_SiebelCustomerProvJMSQTAB AIA_EbizOrderProvJMSQTABV2

31.7.2 Adapter Services Naming

Table 31–10 provides details about the naming standards for adapter services.

Table 31–10 Naming Standards for Adapter Services

Artifact	Name	Example
Transport adapter services (File/FTP)	<Optional Application Short Name><optional verb><EBO/ABO Name><Optional Industry><Resource Provider><Operation>V<Version Number>	E1PurchaseOrderFileReadAdapter E1PurchaseOrderFileWriteAdapterV1 CRMODPOFtpGetAdapter CRMODPOFtpPutAdapterV1
AQJMS /WLSJMS Consumer Adapter Service	<Optional Application Short Name><optional verb><EBO/ABO Name><Optional Industry><Resource Provider><Operation>V<Version Number>	PSFTupdateCurrencyExchange AQJMSCons SiebelCustomerPartyJMSConsumerV1
AQJMS /WLSJMS Producer Adapter Service	<Optional Application Short Name><optional verb><EBO/ABO Name><Optional Industry><Resource Provider><Operation>V<Version Number>	PSFTupdateCurrencyExchange AQJMSProd SiebelCustomerPartyJMSProducerV1
Apps/DB Adapter	<optional verb><EBO/ABO Name><Optional Industry><Application Short Name><Resource Provider>V<Version Number>	SyncCurrencyExchangeListEBiz DBAdapter CreateCustomerEBizAppAdapterV1 UpdatePriceListEBizEventAdapter

31.7.3 Participating Application Service

Table 31–11 provides details about the naming standards for participating application services.

Table 31–11 Naming Standards for Participating Application Services

Artifact	Name
Name	Registered in Mediator with the same name as exposed by the application. For example, BRMCUSTService, Account_BS
Namespace	Follows the namespace used by the application.
Namespace Prefix	Follow the namespace prefix standard.
WSDL	Follows the WSDL name as exposed by the application.
WSDL Message	Follows the message names exposed in the application WSDL.
WSDL Port Type	Follows the port types exposed in the application WSDL.
WSDL Port Type Operations	Follows the operation names exposed in the application WSDL
WSDL Binding	Follows the WSDL
WSDL Service	Follows the service name exposed in the application WSDL

31.8 DVMs and Cross References

This section includes the naming standards for:

- Section 31.8.1, "DVMs"
- Section 31.8.2, "Cross References"

31.8.1 DVMs

When creating DVMs, the following naming standards should be followed:

31.8.1.1 Map Name

Map names:

- Must start with the object name.

  This enables you to identify maps that belong to a certain object. The object name should be equivalent to the EBO name.
- Should be followed by the element name that needs domain value mapping.
- Must be uppercase.

Pattern: **{Object Name}_{Element Name}**

Examples: *CUSTOMERPARTY_ACCOUNTTYPECODE, INVOICE_REJECT_REASON, SALESORDER_CARRIER_TYPECODE*

DVM File Name Examples: *CUSTOMERPARTY95ACCOUNTTYPECODE, INVOICE95REJECT95REASON, SALESORDER95CARRIER95TYPECODE*

31.8.1.2 Map Column Names

Map column names:

- Must be set to the participating application instance name abbreviation that the column value represents. This name can be the application name and its version, or an instance name in case two similar applications of the same version are integrated. The name must be a unique identifier for the application instance across the integration platform in the form: {Application Abbreviated Name}_{Sequence Number}. The sequence number uniquely identifies multiple instances of the same application.
- Must be uppercase.
- A column named COMMON must be always added. This column contains the values used in the EBOs within the platform.

Examples: *COMMON*, *EBIZ_01*, *PSFT_01*, *SEBL_02*, *SEBL_03*, *PORTAL_01*, *IFLEX_01*

31.8.2 Cross References

When creating cross-reference virtual tables in the cross reference tables, the following naming standard should be followed:

31.8.2.1 Table Name

Table names:

- Must not exceed 48 characters.
- Must start with the object name.

  This enables you to identify cross-references that belong to a certain object. The object name should be equivalent to the EBO name.
- Must be followed by the element name that needs cross-referencing.

  If exceeds 48 characters, it should be properly abbreviated.
- Must be uppercase.

Pattern: **{Object Name}_{Element Name}**

Examples: *ORDER ORDERID*, *INVOICE INVOICEID*, *CUSTOMER ID*

31.8.2.2 Column Names

Column names:

- Must not exceed 48 characters.
- Must be set to the participating application instance name abbreviation that the column value represents.

  The name must be a unique identifier for the application instance across the integration platform in the form: {Application Abbreviated Name}_{Sequence Number}. The sequence number uniquely identifies multiple instances of the same application.
- Must be uppercase.
- Must have a column named COMMON added.

  This column contains the values used in the EBOs within the platform.

Examples: *COMMON*, *EBIZ_01*, *PSFT_01*, *SEBL_02*, *SEBL_03*, *PORTAL_01*, *IFLEX_01*

31.9 BPEL

This section discusses naming standards for:

- Section 31.9.1, "BPEL Activities"
- Section 31.9.2, "Other BPEL Artifacts"

31.9.1 BPEL Activities

This section provides naming standards for the following BPEL activities:

- Assign
- Compensate
- Flow
- FlowN
- Invoke
- Java Embedding
- Pick
- Receive
- Scope
- Sequence
- Switch
- Case
- Terminate
- Throw
- Transform
- Wait
- While

31.9.1.1 BPEL Process Name and Namespace

The BPEL process JDeveloper project name should match the BPEL process name (use default project setting).

Name standards

- The name should follow the general standard naming standards depending on whether it is being used for EBS, ABCS Impl, or Adapter Service.
- The name should clearly describe the process and action/verb being performed.

Namespace standards:

- The namespace should follow the general namespace standards depending on whether it is being used for EBS, ABCS Impl, or Adapter Service.
- The namespace must reflect the taxonomy of the process.
- The namespace must include the major version number where appropriate.
- BPEL composite's reference component name should follow the general naming standards based on the type of AIA artifacts it is calling.

31.9.1.2 Assign

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with the **Assign** prefix.
- Followed by a name describing what is being assigned. If what is assigned is a message, then use the message name.
- In case there are multiple assignments, provide a name that describes the group of assignments if possible.

Pattern: **Assign<Name of what is being assigned>**

Example: *AssignPaymentEBM, AssignOrderInitialValues*

31.9.1.3 Compensate

Follow these guidelines:

- The name should follow the general standard naming standards.
- Start with the **Compensate** prefix.
- Followed by the scope encapsulating the tasks to be compensated.

Pattern: **Compensate<scope name>**

Example: *CompensateProcessCreditCheckMilestone, Compensate TranseferFundsScope*

31.9.1.4 Flow

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts by a name describing the tasks being run concurrently.
- Ends with the **Flow** suffix.

Pattern: **<Name describing concurrent tasks>Flow**

Example: *CallManufacturersFlow, GetQuotesFlow*

31.9.1.5 FlowN

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts by a name describing the dynamic tasks being run concurrently.
- Ends with the **FlowN** suffix.
- The index variable name should be the flow name with **Index** as suffix.

Pattern: **name = <Name describing concurrent tasks>FlowN, index variable = <Name describing concurrent tasks>FlowNIndex**

Example: *ActivateUsersFlowN (ActivateUsersFlowNIndex), CheckSuppliersFlowN (CheckSuppliersFlowNIndex)*

31.9.1.6 Invoke

Follow these guidelines:

- The name should follow the general standard naming standards.

- Starts with the **Invoke** prefix.
- Followed by the partner link to be invoked.
- Followed by **Call** if synchronous invocation or **Start** if asynchronous invocation.
- Followed by the operation name within the partner link.

Pattern: **Invoke<Partner Link Name>{Call/Start}<Operation>**

Example: *InvokeCustomerServiceCallGetCustomer, InvokeNotificationServiceStartNotifyByEmail*

31.9.1.7 Java Embedding

Follow these guidelines:

- The name should follow the general standard naming standards.
- The name should be similar to a Java method Name with lower-camel-case.

Pattern: **<A name describing the functionality>**

Example: *getDiscountPrice*

31.9.1.8 Pick

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with the **Pick** prefix.
- Followed by a name describing as accurate as possible all branches (onMessage and onAlarm) within the pick activity.

Pattern: **Pick<Name describing the branches to pick from>**

Example: *PickOrderAckOrTimeout*, *PickFirstQuote*

31.9.1.9 Receive

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with the **Receive** prefix.
- Contains the name of the message it is receiving.

Pattern: **Receive<Message Name>**

Example: *ReceiveUpdateInvoiceEBM*

31.9.1.10 Scope

Follow these guidelines:

- The name should follow the general standard naming standards.
- Including brief information about transaction type may be appropriate.
- Use **Milestone** as the suffix if the scope is a candidate for end-user monitor.
- If it is not intended for the end-user monitor, use **Scope** as the suffix.

Pattern: **<Name describing the Scoped Tasks>{ Scope |Milestone}**

Examples: *GetCreditRatingScope*, *GetLoanOfferScope*, *ProcessCreditCheckMilestone*

31.9.1.11 Sequence

Follow these guidelines:

- The name should follow the general standard naming standards.
- The sequence name should describe the steps performed in the sequence.
- The sequence name should end with **Sequence** suffix.

Pattern: **<Name describing the Sequenced Tasks>Sequence**

Example: *GetCustomerInfoSequence*

31.9.1.12 Switch

Follow these guidelines:

- The name should follow the general standard naming standards.
- Start with the **Switch** prefix.
- Followed by what is being evaluated

Pattern: **Switch<Name of what is being evaluated>**

Example: *SwitchCreditRating*

31.9.1.13 Case

Follow these guidelines:

- The name should follow the general standard naming standards.
- Start with the **Case** prefix.
- Followed by the evaluated value.

Pattern: **Case<Name evaluated value>**

Example: *CaseBadCredit*, *CaseApprovalRequired*

31.9.1.14 Terminate

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with the **Terminate** prefix.
- Followed by a name describing the termination reason.

Pattern: **Terminate<reason of termination>**

Example: *TerminateTimeout*, *TerminateEndOfProcess*

31.9.1.15 Throw

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with the **Throw** prefix.
- Followed by the fault name.
- The fault variable name is typically named the same as the fault name.

Pattern: **Throw<fault name>**

Example: *ThrowExceededMaxAmount*, which uses ExceededMaxAmount variable.

Note: When defining a Catch in the Scope activity, the displayed catch name is the fault name.

31.9.1.16 Transform

Follow these guidelines:

The name should follow the general standard naming standards.

- Starts with the **Xform** prefix.
- Followed by the source name.
- Followed by **To**.
- Followed by the destination name.

Pattern: **Xform<source>To<destination>**

Example: *XformBillToPortal80Bill*

31.9.1.17 Wait

Follow these guidelines:

The name should follow the general standard naming standards.

- Starts with the **Wait** prefix.
- Followed by a name describing the reason for waiting.

Pattern: **Wait<Name describing the waiting reason>**

Example: *WaitOrderAcknowledgeTimeout, WaitWarmUpTime*

31.9.1.18 While

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with the **While** prefix.
- Followed by a name describing the loop condition.

Pattern: **While<Name describing the loop condition>**

Example: *WhileAllMsgsSent*

31.9.2 Other BPEL Artifacts

Follow these guidelines for other BPEL artifacts:

31.9.2.1 Variables

Follow these guidelines:

- The name should follow the general standard naming standards.
- Use lower-camel-case for variable names.
- The data type must not be part of the variable name.

Example: *accountBalance, invoiceAmount*.

31.9.2.2 Properties

Property names follow the general BPEL variables naming standards.

31.9.2.3 Correlation Sets

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with a name describing the correlation set.
- Ends with **CorSet** suffix.

Pattern: **<Name describing the correlation>CorSet**

Example: *PurchaseOrderCorSet*

31.9.2.4 Correlation Set Properties

The correlation set property names follows the general BPEL variables naming standards.

31.10 Custom Java Classes

Custom Java classes must follow the standard Java coding practices. The Java code components names types must conform to the Oracle Corporate Java Coding Standards: http://bali.us.oracle.com/bali/ojcs/front.html.

All of AIA custom java code must exist in a sub-package:

oracle.apps.aia.<lba>... where **lba** stands for logical business area.

Externally facing services implemented in Java must have the version number part of the package to be in line with our namespace naming standards. This also enables you to publish the same service under different versions at the same time.

oracle.apps.aia.<lba>... v<version> where lba stands for logical business area.

> **Tip:** To avoid collisions, 'aia' must be defined as an application in the Fusion Application.

Examples:

- *oracle.apps.aia.util.logging*: contains util java classes used for logging.
- *oracle.apps.aia.security.siebel.login*: contains java modules to login into Siebel.
- *oracle.apps.aia.order.siebel.V1*: contains an order query service implemented in Java.
- *oracle.apps.aia.item.pricediscount.v3*: contains a Java Service price discount engine.

31.11 Package Structure

The complete source control and package structure related information is described in the *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack*.

31.12 Deployment Plans

Table 31–12 provides details about the naming standards for Deployment Plans.

Table 31–12 Naming Standards for Deployment Plans

Artifact Name	Artifact File Name	Purpose	Mandatory / Optional	Details
Deployment Plan (Auto Generated)	<PIP Code>DP.xml	Deployment instructions for the artifacts harvested by AIA Harvester	Every Pre-Built Integration must deliver a plan.	The ODI and the main deployment plan can coexist. In the command argument of the deployment plan generator you must give **"-DODIinput=ODIBOM.xml"** as an additional argument to generate the combined deployment plan. You can also generate deployment plan for the ODI alone by skipping the **"-Dinput"** and **"-DharvesterSettings"** arguments. Deployment Plan generator automatically generates <PIP Code>HS.xml
Deployment Plan holding customer extensions	<PIP Code>CustomDP.xml	Deployment instructions for the artifacts created by the customer	"<ProjectCode>CustomDP.xml :re-generated deployment plan on customer sites using PLW and DPG. It contains only native artifacts	This is similar to the main deployment plan with all the Preinstall, Configurations, Deployments and PostInstall section.
Deployment Plan holding non-native artifacts	<PIP Code>SupplementaryDP.xml	Deployment instructions for non-native artifacts (J2EE application, java class, web services and every other artifact that could not be harvested)	Pre-Built Integrations having non-native artifacts alone should deliver this plan	This is similar to the main deployment plan with all the Preinstall, Configurations, Deployments and PostInstall section.

A

Delivered Oracle AIA XPath Functions

This appendix provides details about the XPath functions that are delivered with the Oracle Application Integration Architecture (AIA) Foundation Pack for use in your integration flows.

This appendix includes the following sections:

> **Note:** These functions can be called from BPEL and XSLT, and Mediator routing rule filters.

A.1 aia:getSystemProperty()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.
  xpath.AIAFunctions"
string aia:getSystemProperty (string propertyName, boolean needAnException)
```

A.1.1 Parameters

- `propertyName`

  Name of the system property for which to retrieve the value.

- `needAnException`

  Used to specify whether to throw an exception if the property is not found, otherwise return empty string.

A.1.2 Returns

Returns the string value of the system property identified by propertyName. If the property is not found, either an exception is thrown or an empty string is returned, depending on the value of the needAnException parameter.

A.1.3 Usage

- XSLT example:

```
<xsl:variable name="activeRuleset" select="aia:getSystemProperty('Routing.
  ActiveRuleset',true())"/>
```

- BPEL example:

```
<assign name="AssignVar">
   <copy>
      <from expression="aia:getSystemProperty('Routing.    ActiveRuleset',
       true())"/>
      <to variable="ActiveRuleset"/>
   </copy>
</assign>
```

A.2 aia:getSystemModuleProperty()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.
  xpath.AIAFunctions"
string aia:getSystemModuleProperty (string moduleName, string propertyName,
 boolean needAnException)
```

A.2.1 Parameters

- `moduleName`

  Module for which to retrieve the property.

- `propertyName`

  Name of the property for which to retrieve the value.

- `needAnException`

  Used to specify whether to throw an exception if the property is not found, otherwise return empty string.

A.2.2 Returns

Returns the string value of the module property identified by moduleName and propertyName. If the module property is not found, then the system property of the same name is returned. If the system property with the same name is not found, then either an exception is thrown or an empty string is returned, depending on the value of the needAnException parameter.

A.2.3 Usage

- XSLT example:

```
<xsl:variable name="errHdlrImpl" select="aia:getSystemModuleProperty
  ('ErrorHandler','COMMON.ERRORHANDLER.IMPL',true())"/>
```

- BPEL example:

```
<assign name="AssignVar">
   <copy>
      <from expression="aia:getSystemModuleProperty('ErrorHandler',
       'COMMON.ERRORHANDLER.IMPL',true())"/>
      <to variable="ErrorHandler"/>
   </copy>
</assign>
```

A.3 aia:getServiceProperty()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.
  xpath.AIAFunctions"
string aia:getServiceProperty (string serviceName, string propertyName,
 boolean needAnException)
```

A.3.1 Parameters

- `serviceName`

  Service for which to retrieve the property.

- `propertyName`

  Name of the property for which to retrieve the value.

- `needAnException`

  Used to specify whether to throw an exception if the property is not found, otherwise return empty string.

A.3.2 Returns

Returns the string value of the service property identified by serviceName and propertyName. If the service property is not found, then the system property of the same name is returned. If the system property with the same name is not found, then either an exception is thrown or an empty string is returned, depending on the value of the needAnException parameter.

A.3.3 Usage

- XSLT example:

```
<xsl:variable name="defaultSystemID" select="aia:getServiceProperty
  ('{http://xmlns.oracle.com/ABCSImpl/Siebel/Core/UpdateCustomerParty
  SiebelReqABCSImpl/V2}UpdateCustomerPartySiebelReqABCSImplV2','Default.
  SystemID',true())"/>
```

- BPEL example:

```
<assign name="AssignVar">
   <copy>
```

```
        <from expression= "aia:getServiceProperty('{http://xmlns.oracle.com
         /ABCSImpl/Siebel/Core/UpdateCustomerPartySiebelReqABCSImpl/V2}
         UpdateCustomerPartySiebelReqABCSImplV2','Default.SystemID',true())"/>
        <to variable="DefaultSystemID"/>
      </copy>
    </assign>
```

A.4 aia:getEBMHeaderSenderSystemNode()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.
  xpath.AIAFunctions"
node-set aia:getEBMHeaderSenderSystemNode (string senderSystemCode, string
  senderSystemID)
```

A.4.1 Parameters

- `senderSystemCode`

  System Code as defined in the Oracle AIA system registry.

- `senderSystemID`

  System Internal Id as defined in the Oracle AIA system registry.

For information about maintaining the Oracle AIA system registry, see Section 2.4.5.2, "Managing the Oracle AIA System Registry."

A.4.2 Returns

Returns a node set of system information, entered in the Oracle AIA system registry for the given System Code or Internal ID.

A.4.3 Usage

Given the set up on the Systems page shown in Figure A–1, both aia:getEBMHeaderSenderSystemNode('SEBL_01', '') and aia:getEBMHeaderSenderSystemNode('', 'siebel') would return the node set provided in Example A–1.

Figure A–1 Systems Page Entry and aia:getEBMHeaderSenderSystemNode()

Example A–1 Node Set Returned for aia:getEBMHeaderSenderSystemNode()

```
<ID xmlns="">SEBL_01</ID>
<Description xmlns="">Siebel Instance 01</Description>
<IPAddress xmlns="">xxxxx.siebel.com</IPAddress>
<ContactName xmlns="">Siebel contact</ContactName>
<ContactPhone xmlns="">1234567891</ContactPhone>
```

```
   <ContactEmail xmlns="">Siebelcontact@Siebel.com</ContactEmail>
   <Url xmlns="">http://xxxxx.siebel.com:80/ecommunications_enu</Url>
   <Application xmlns="">
   <ID>CRM</ID>
   <Version>8.0</Version>
</Application>
```

An XSLT example for aia:getEBMHeaderSenderSystemNode() is provided in Example A–2.

> **Tip:** This example requires `<xsl:stylesheet version="2.0" ...>`

Example A–2 XSLT Example for aia:getEBMHeaderSenderSystemNode()

```
<xsl:variable name="senderNodeVariable">
</xsl:variable>

<corecom:Sender>
   <corecom:ID>
      <xsl:value-of select="$RequestTargetSystemID"/>
   </corecom:ID>
   <corecom:Description>
         <xsl:value-of select="$senderNodeVariable/Description"/>
   </corecom:Description>
   <corecom:IPAddress>
      <xsl:value-of select="$senderNodeVariable/IPAddress"/>
   </corecom:IPAddress>
   <corecom:CallingServiceName> ...
   </corecom:CallingServiceName>
   <corecom:Application>
      <corecom:ID>
         <xsl:value-of select="$senderNodeVariable/Application/ID"/>
   </corecom:ID>
   <corecom:Version>
      <xsl:value-of select="$senderNodeVariable/Application/Version"/>
      </corecom:Version>
   </corecom:Application>
   <corecom:ContactName>
      <xsl:value-of select="$senderNodeVariable/ContactName"/>
   </corecom:ContactName>
   <corecom:ContactEmail>
      <xsl:value-of select="$senderNodeVariable/ContactEmail"/>
   </corecom:ContactEmail>
   <corecom:ContactPhoneNumber>
      <xsl:value-of select="$senderNodeVariable/ContactPhone"/>
   </corecom:ContactPhoneNumber>
...
</corecom:Sender>
```

A.5 aia:getSystemType()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
string aia:getSystemType (string systemCode)
```

A.5.1 Parameters

`systemCode`: System code as entered in the Oracle AIA system registry.

For information about maintaining the Oracle AIA system registry, see Section 2.4.5.2, "Managing the Oracle AIA System Registry."

A.5.2 Returns

Returns the System Type associated with a System Code as entered in the Oracle AIA system registry.

A.5.3 Usage

Given the set up on the Systems page shown in Figure A–2, the result of aia:getSystemType('SEBL_01') would be the string `SIEBEL`.

Figure A–2 Systems Page Entry and aia:getSystemType()

Internal Id	System Code	System Description	IP Address	URL	System Type
siebel	SEBL_01	Siebel Instance 01			SIEBEL

- XSLT example:

```
<xsl:variable name="systemType" select="aia:getSystemType('SEBL_01')"/>
```

- BPEL example:

```
<assign name="AssignVar">
   <copy>
      <from expression= "aia:getSystemType('SEBL_01')"/>
      <to variable="SystemType"/>
   </copy>
</assign>
```

A.6 aia:getErrorMessage()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.
  xpath.AIAFunctions"
string aia:getErrorMessage (string errorCode, string localeString, string
 delimiter)
```

A.6.1 Parameters

- `errorCode`

  Error code.

- `localeString`

  Delimited locale string.

- `Delimiter`

  Delimiter used in the localeString.

A.6.2 Returns

This function looks up the error message resource bundle and return a localized string for the input errorCode. The localeString is a concatenated string of LanguageCode, CountryCode, and Variant delimited by the delimiter specified. If no locale is found with the input parameters, the system default locale is used.

For more information, see java.util.Locale documentation: http://download.oracle.com/javase/1.5.0/docs/api/java/util/class-use/Locale.html.

A.6.3 Usage

- XSLT example:

```
<xsl:variable name="errMsg" select="aia:getErrorMessage('AIA_ERR_AIAO2C2_
  1007','','')"/>
```

- BPEL example:

```
<assign name="Assign_Fault">
   <copy>
      <from expression="'AIA_ERR_AIAO2C2_1007'"/>
      <to variable="AIAFaultMsg" part="AIAFault"query="/corecom:
       Fault/corecom:FaultNotification/corecom:FaultMessage/corecom:Code"/>
   </copy>
   <copy>
      <from expression="aia:getErrorMessage('AIA_ERR_AIAO2C2_1007','','')"/>
      <to variable="AIAFaultMsg" part="AIAFault"query="/corecom:
       Fault/corecom:FaultNotification/corecom:FaultMessage/corecom:Text"/>
   </copy>
</assign>
```

A.7 aia:getCorrectiveAction()

```
namespace aia="g aia:getCorrectiveAction (string correctiveActionCode,
  string localeString, string delimiter)
```

A.7.1 Parameters

- `correctiveActionCode`

  Corrective action code.

- `localeString`

  Delimited locale string.

- `Delimiter`

  Delimiter used in the localeString.

A.7.2 Returns

This function looks up the Corrective Action Code resource bundle and returns a localized string for the input correctiveActionCode.

The localeString is a concatenated string of LanguageCode, CountryCode, and Variant delimited by the delimiter specified. If no locale is found with the input parameters, the system default locale is used.

For more information, see java.util.Locale documentation: http://download.oracle.com/javase/1.5.0/docs/api/java/util/class-use/Locale.html.

A.7.3 Usage

- XSLT example:

```
<xsl:variable name="corrAction" select="aia:getCorrectiveAction('SAMPLECODE'
  ,'','')"/>
```

- BPEL example:

```
<assign name="Assign_Fault">
   <copy>
      <from expression="aia:getCorrectiveAction('SAMPLECODE','','')"/>
      <to variable="AIAFaultMsg" part="AIAFault" query="/corecom:Fault/
       corecom:FaultNotification/corecom:CorrectiveAction"/>
   </copy>
</assign>
```

A.8 aia:isTraceLoggingEnabled()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
string aia:isTraceLoggingEnabled (string logLevel, string processName)
```

A.8.1 Parameters

- `logLevel`

  Log level that you wish to log.

- `processName`

  Name of the process.

A.8.2 Returns

Boolean indicating whether the specified logLevel is enabled for the specified process.

This function does two things:

1. Checks whether logging is enabled or disabled for the input process name from the AIAConfigurationProperties.xml file.
2. Checks if the trace logger log level is set to log the input log level.

This function returns true only when both the above conditions return true, otherwise it returns false. If the logLevel parameter is null or if it is incorrectly specified, this function returns false. If processName is null or empty strings, this function returns null.

A.8.3 Usage

- XSLT example:

```
<xsl:variable name="testVal" select="aia:isTraceLoggingEnabled('INFO',
  'SamplesCreateCustomerSiebelReqABCSImpl')" />
```

- BPEL example:

```
<assign name="Assign_Fault">
   <copy>
      <from expression="aia:isTraceLoggingEnabled('INFO',
       'SamplesCreateCustomerSiebelReqABCSImpl')"/>
      <to variable="isTraceLoggingEnabledVal"/>
   </copy>
</assign>
```

A.9 aia:logErrorMessage()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
string aia:logErrorMessage (node-set ebmHeader, string message)
```

This function logs an error message. If the ebmHeader parameter is null or not passed, the message is logged without any supplemental attributes. An error message is always logged with the level SEVERE.

A.9.1 Parameters

- `ebmHeader`

  The Enterprise Business Message (EBM) header providing context for the error message.

- `Message`

  Message to log.

A.9.2 Returns

Returns an empty string.

A.9.3 Usage

- XSLT example:

```
<xsl:variable name="testVal"  select="aia:logErrorMessage
  (oraext:parseEscapedXML('<test></test>'),'LogError:: Your Error message
  goes here.............................')"/>
```

- BPEL example:

```
<assign name="Assign_Fault">
   <copy>
      <from expression="aia:logErrorMessage(oraext:parseEscapedXML
       ('<test></test>'),'LogError from XSL::Your Error Message goes
       here..........')"/>
      <to variable="testXpathVal"/>
   </copy>
</assign>
```

A.10 aia:logTraceMessage()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
```

```
string aia:logTraceMessage (string level, node-set ebmHeader, string message)
```

This function logs a trace message.

If the ebmHeader parameter is null or not passed, the message is logged without any supplemental attributes.

If the level parameter is null or if it is incorrectly specified, a message is logged with level `INFO`. The level should be `java.util.logging.Level`.

The levels in descending order are:

- SEVERE (highest value)
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST (lowest value)

A.10.1 Parameters

- `Level`

  Level of the trace message.

- `ebmHeader`

  The EBM Header providing context for the trace message.

- `Message`

  Message to log.

A.10.2 Returns

Returns an empty string.

A.10.3 Usage

- XSLT example:

```
<xsl:variable name="testVal"  select="aia:logTraceMessage('ERROR',
  oraext:parseEscapedXML('<test></test>'),'LogTrace :: Your Trace message
  goes her.............')" />
```

- BPEL example:

```
<assign name="Assign_Fault">
   <copy>
      <from expression="aia:logTraceMessage
       ('ERROR',ora:parseEscapedXML('<test></test>'),'LogTraceError::
       Your trace message goes here.........')"/>
      <to variable="testXpathVal"/>
   </copy>
</assign>
```

A.11 aia:getNotificationRoles()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
node-set aia:getNotificationRoles (string systemId, string errorCode,
 string serviceName, string processName)
```

A.11.1 Parameters

- `systemId`

  System ID.

- `errorCode`

  Error code.

- `serviceName`

  Name of the errored service.

- `processName`

  Name of end-to-end process.

A.11.2 Returns

This function queries the Oracle AIA system registry with input values and returns a node-set containing the actor role and the FYI role corresponding to the input errorCode.

For example:

```
<actor>seblAdmin</actor>
<fyi>seblCSR</fyi>
```

If serviceName is null or empty strings or if no value is found from in Oracle AIA system registry, this function returns the default roles specified in the AIAConfigurationProperties.xml file.

For information about maintaining the Oracle AIA system registry, see Section 2.4.5.2, "Managing the Oracle AIA System Registry."

A.11.3 Usage

- XSLT example:

```
<xsl:variable name="testVal"  select="aia:getNotificationRoles
  ('AIADEMO','AIADEMO_ORDER_FALLOUT','AIADemoProcessSalesOrderCBP',
  'AIADemoProcessSalesOrderCBP')" />
```

- BPEL example:

```
<assign name="Assign_Fault">
   <copy>
      <from expression="aia:getNotificationRoles('AIADEMO','AIADEMO_ORDER_
       FALLOUT','AIADemoProcessSalesOrderCBP','AIADemoProcessSalesOrderCBP')
       "/>
      <to variable="testXpathVal"/><to variable="NotificationRole"/>
   </copy>
</assign>
```

A.12 aia:getAIALocalizedString()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
string aia:getAIALocalizedString (string resourceBundleId, string key,
 node params)
string aia:getAIALocalizedString (string resourceBundleId, string key,
 string language, string country, node params)
```

A.12.1 Parameters

- `resourceBundleId`

  Identifies the AIA resource bundle from which to retrieve the localized string.

- `key`

  The key whose value has to be picked up from the resource bundle.

- `language`

  Language, according to java.util.Locale, for which to retrieve the localized string.

- `Country`

  Country, according to java.util.Locale, for which to retrieve the localized string.

- `Params`

  Node supplying the values for any bind variables within the localized string.

The AIAConfigurationProperties.xml file contains a set of module configuration properties that provide the mapping of resourceBundleId values to resource bundle class names, as shown in Example A–3.

This mapping is used at run time to determine which resource bundle class to use for looking up the localized string.

Example A–3 Module Configuration Properties

```
<ModuleConfiguration moduleName="ResourceBundle">
   <property name="Telco/BillingManagement">oracle.apps.aia.core.
    i18n.AIAListResourceBundle</property>
   <property name="Telco/ProductLifeCycle">oracle.apps.aia.core.
    i18n.AIAListResourceBundle</property>
   <property name="Telco/SalesOrder">oracle.apps.aia.core.i18n.
    AIAListResourceBundle</property>
   <property name="Telco/CustomerParty">oracle.apps.aia.core.i18n.
    AIAListResourceBundle</property>
</ModuleConfiguration>
```

A.12.2 Returns

This function returns the localized string for the passed key from an AIA resource bundle, identified by resourceBundleId. If language and country are omitted, the default locale is assumed.

A.12.3 Usage

Example A–4 provides a BPEL usage example for aia:getAIALocalizedString().

Example A–4 BPEL Usage Example for aia:getAIALocalizedString()

```
<assign name="Assign_1">
   <copy>
      <from variable="inputVariable" part="payload" query="/sordabo:
       ListOfSWIOrderIO/sordabo:SWIOrder/sordabo:OrderNumber"/>
      <to variable="localizedStringParams" query="/bpelcom:parameters/
       bpelcom:item/bpelcom:value"/>
   </copy>
   <copy>
      <from expression="aia:getAIALocalizedString ('Telco/SalesOrder','ORDER_
       NUMBER_MESSAGE',bpws:getVariableData(&quot;localizedStringParams&quot;))
       "/>
      <to variable="internationalizedstring"/>
   </copy>
</assign>
```

A.13 aia:getConvertedDate()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
string getConvertedDate(String dateTimeTz, String timeZone, boolean
 needAnException)
```

A.13.1 Parameters

- `dateTimeTz`

  DateTime as String, in standard W3C and RFC date format. For example: yyyy-MM-dd'T'HH:mm:ss.SSSZ.

- `timeZone`

  Target time zone, as 3 characters code. For example, IST or as GMT offset +05:30.

- `needAnException`

  Boolean flag: `true` throws exception for any Parse error and `false` returns the input date.

A.13.2 Returns

This function returns the converted date as string without time zone in yyyy-MM-dd'T'HH:mm:ss.SSS format.

A.13.3 Usage

- XSLT example:

  ```
  <xsl:value-of select="aia:getConvertedDate(ns0:
   CreateEngineeringChangeOrderList/ns0:ImplementationDate,$ebizTimeZone,false
   ())"/>
  ```

- BPEL example:

  ```
  <copy>
     <from expression="aia:getConvertedDate("2010-04-26T16:25:00+05:30",
      "-08:00", false()) "/>
     <to variable="getConvertedDateVal"/>
  </copy>
  ```

A.14 aia:getConvertedDateWithTZ()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
string getConvertedDateWithTZ(String dateTimeTz, String timeZone,
 boolean needAnException)
```

A.14.1 Parameters

- `dateTimeTz`

  DateTime as String, in standard W3C and RFC date format. For example: yyyy-MM-dd'T'HH:mm:ss.SSSZ

- `timeZone`

  Target time zone, as 3 characters code. For example, IST or as GMT offset +05:30.

- `needAnException`

  Boolean flag: `true` throws exception for any Parse error and `false` returns the input date.

A.14.2 Returns

This function returns the converted date as string without time zone in the input date format.

A.14.3 Usage

- XSLT example:

```
<xsl:value-of select="aia:getConvertedDateWithTZ(ns0:
  CreateEngineeringChangeOrderList/ns0:ImplementationDate,$ebizTimeZone,
  false())"/>
```

- BPEL example:

```
<copy>
   <from expression="aia:getConvertedDateWithTZ('2010-04-26T14:30:00
    +00:00','+05:30', false())"/>
   <to variable="getConvertedDateWithTZVal"/>
</copy>
```

B

XSL for Developing CAVS-Enabled Oracle AIA Services

This appendix provides XSL text that should be used in developing Composite Application Validation System (CAVS)-enabled Oracle Application Integration Architecture (AIA) services.

This appendix includes the following sections:

- Section B.1, "AddTargetSystemID.xsl"
- Section B.2, "SetCAVSEndpoint.xsl"

B.1 AddTargetSystemID.xsl

This section provides XSL text that should be used to develop CAVS-enabled provider Application Business Connector Services (ABCSs).

For more information about how to use this XSL, see Section 16.4, "Developing ABCS for CAVS Enablement."

Example B–1 AddTargetSystemID.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
 xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
 xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
 xmlns:ehdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.
  headers.ESBHeaderFunctions"
 xmlns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"
 xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.
  functions.Xpath20"
 xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.
  XRefXPathFunctions"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:ora="http://schemas.oracle.com/xpath/extension"
 xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
 xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.
  functions.ExtFunc"
 xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
 xmlns:aia=http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.
  xpath.AIAFunctions exclude-result-prefixes="xsl plnk coresalesorder ns0 ns3
  ns5 ns1 client corecustcom ns4 corecom bpws ehdr aia hwf xp20 xref ora ids
  orcl">
   <xsl:param name="ConfigServiceName">{[ABCServiceNamespace]}[ABCServiceName]
   </xsl:param>
   <xsl:param name="ConfigPropertyName">Default.SystemID</xsl:param>
```

```
<xsl:template match="/*">
   <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
   </xsl:copy>
</xsl:template>

<xsl:template match="corecom:EBMHeader">
   <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
   </xsl:copy>
</xsl:template>

<xsl:template match="corecom:EBMHeader/corecom:Sender">
   <xsl:copy-of select="."/>
   <xsl:if test="not(following-sibling::corecom:Target)">
      <corecom:Target>
         <xsl:variable name="TargetID" select="aia:getServiceProperty
          ($ConfigServiceName,$ConfigPropertyName,true())"/>
         <corecom:ID>
            <xsl:value-of select="$TargetID"/>
         </corecom:ID>
         <corecom:ApplicationTypeCode>
            <xsl:value-of select="aia:getSystemType($TargetID)"/>
         </corecom:ApplicationTypeCode>
      </corecom:Target>
   </xsl:if>
</xsl:template>

<xsl:template match="corecom:EBMHeader/corecom:Target">
   <corecom:Target>
      <xsl:copy-of select="@*"/>
      <xsl:variable name="TargetID">
         <xsl:choose>
            <xsl:when test="corecom:ID/text()">
               <xsl:value-of select="corecom:ID/text()"/>
            </xsl:when>
            <xsl:otherwise>
               <xsl:value-of select="aia:getServiceProperty
                ($ConfigServiceName,$ConfigPropertyName,true())"/>
            </xsl:otherwise>
         </xsl:choose>
      </xsl:variable>
   <corecom:ID>
      <xsl:copy-of select="corecom:ID/@*"/>
      <xsl:value-of select="$TargetID"/>
   </corecom:ID>
   <xsl:copy-of select="corecom:OverrideRoutingIndicator"/>
   <xsl:copy-of select="corecom:ServiceName"/>
   <corecom:ApplicationTypeCode>
      <xsl:copy-of select="corecom:ApplicationTypeCode/@*"/>
      <xsl:choose>
         <xsl:when test="corecom:ApplicationTypeCode/text()">
            <xsl:value-of select="corecom:ApplicationTypeCode/text()"/>
         </xsl:when>
         <xsl:otherwise>
            <xsl:value-of select="aia:getSystemType($TargetID)"/>
         </xsl:otherwise>
      </xsl:choose>
   </corecom:ApplicationTypeCode>
```

```
      <xsl:copy-of select="corecom:EndPointURI"/>
      <xsl:copy-of select="corecom:Custom"/>
   </corecom:Target>
   </xsl:template>

   <xsl:template match="@*|node()">
      <xsl:copy-of select="."/>
   </xsl:template>

</xsl:stylesheet>
```

B.2 SetCAVSEndpoint.xsl

This section provides XSL text that should be used to develop CAVS-enabled requester ABCSs.

For more information about how to use this XSL, see Section 16.4, "Developing ABCS for CAVS Enablement."

Example B–2 SetCAVSEndpoint.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"xmlns:xsl=http://www.w3.org/1999/XSL/Transform
 xmlns:ehdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.
 headers.ESBHeaderFunctions" xmlns:jhdr="http://xmlns.oracle.com/esb"
 xmlns:corecom=http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2
 exclude-result-prefixes="xsl corecom ehdr jhdr">
   <xsl:template match="/">
      <xsl:copy-of select="/*"/>
      <xsl:variablename="Endpoint"select="/*/corecom:EBMHeader/corecom:Message
       ProcessingInstruction/corecom:DefinitionID"/>
         <xsl:if test="$Endpoint!=''">
            <xsl:variable name="SetEndpoint"select="ehdr:setOutboundHeader
             ('/jhdr:ESBHeader/jhdr:location',$Endpoint,'jhdr=http://xmlns.
             oracle.com/esb;')"/>
         </xsl:if>
   </xsl:template>
</xsl:stylesheet>
```

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X