Oracle® Fusion Middleware

Developer's Guide for Oracle SOA Core Extension

12*c* (12.1.3)

E37422-03

July 2015

Describes how to use SOA Core Extension to conceptualize projects. Describes how to implement new services that extend Process Integration Packs. Use the information in this guide to help ensure that the solutions you develop can be upgraded and supported.

ORACLE

Oracle Fusion Middleware Developer's Guide for Oracle SOA Core Extension, 12c (12.1.3)

E37422-03

Primary Author: Oracle Corporation

Contents

6 Designing Application Business Connector Services

7 Constructing the ABCS

8 Completing ABCS Development

11 Developing and Implementing Outbound B2B Integration Flows

12 Developing and Implementing Inbound B2B Integration Flows

15 Using Oracle Data Integrator for Bulk Processing

16 Working with Message Transformations

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

21 Oracle AIA Naming Standards for AIA Development

22 Editing Transformations Using Mapping Editor

A Delivered Oracle AIA XPath Functions

List of Tables

List of Tables

List of Tables

Preface

Welcome to . This document describes how to use SOA Core Extension to conceptualize AIA projects. Use the information in this guide to help ensure that the AIA solutions you develop can be upgraded and supported.

Audience

This guide is intended for developers and provides information to help conceptualize Oracle Application Integration Architecture (AIA) projects.

Following the guidance provided here helps ensure that your AIA solutions can follow optimal upgrade, support, and maintenance paths.

Oracle AIA Guides

In addition to this Developer's Guide, we provide the following Oracle AIA guides for this release:

- *Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack*
- *Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack*

Related Guides

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

- *Installing and Configuring Oracle SOA Suite and Business Process Management*
- *Administering Oracle SOA Suite and Oracle Business Process Management Suite*
- *Developing SOA Applications with Oracle SOA Suite*
- *Administering Web Services*
-

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 that have purchased support 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.

1

Using SOA Core Extensions for AIA Development

This chapter provides an overview of usage and features of SOA Core Extension technology for AIA development.

This chapter includes the following section

AIA Development Using SOA Core Extension

SOA Core Extension (SCE) is a feature delivered with SOA 12c to provide the necessary capability for Oracle Application Integration Architecture (AIA) based integration projects to continue operating after an upgrade to SOA 12c.

Customers who have AIA style integrations running in 11g will want to select the SOA Core Extension feature during the domain configuration/extension portion of their upgrade to SOA 12c. Selecting this feature will provide the necessary frameworks and features as required by deployed any AIA-style integrations.

SOA Core Extension provides the following features:

- **AIA Composite Application Validation System (CAVS)** - CAVS is a framework that provides a structured approach to test integration of Oracle Application Integration Architecture (Oracle AIA) services. The CAVS feature includes test initiators that simulate web service invocations and simulators that simulate service endpoints.
- **AIA Error Handling Framework (AIA-EH)** - The Oracle AIA Error Handling Framework provides error handling and logging components to support the needs of integration services operating in an Oracle Application Integration Architecture (AIA) ecosystem.
- **AIA Error Resubmission Utility** - The AIA Message Resubmission Utility enables users to resubmit error messages based on these integration milestones: Queue, Topic, Resequencer, or AQ.
- **AIA Administration Console** - A web-based application for administrators to use features such as CAVS.

Understanding AIA

Oracle Application Integration Architecture (AIA) is a framework consisting of pre-built content, templates and methodologies for orchestrating agile user-centric business processes across enterprise applications. Oracle enables best-in-class applications-Oracle and non-Oracle-to work together, leveraging industry best practices and open standards to reduce costs and increase business flexibility.

AIA Foundation Pack

Oracle Application Integration Architecture Foundation Packs provide a faster, structured and repeatable approach to process composition. AIA Foundation Pack provides the framework for Oracle's Pre-built integrations. The Oracle Application Integration Architecture Foundation Pack 11g provided a Common Object and Shared Service Library with supporting SOA programming model and best practice implementation methodology. Using Oracle AIA Foundation Packs, Oracle customers and partners can modernize their applications, consolidate systems, and both improve and extend their business processes across Oracle and non-Oracle applications following the same principles used by Oracle to deliver pre-built integrations.

AIA Pre-built Integrations and Process Integration Packs (PIPs)

Oracle Application Integration Architecture (AIA) delivers pre-built integrations as either direct integrations or process integrations packs

- **Direct Integrations (DI)** - Pre-built integrations that manage data flows and data synchronizations between Applications.
- **Process Integration Packs (PIPs)** - Pre-built integration accelerators combine one or more of the integration styles such as data-centric integration, web services, reference data query, and/or process-centric integrations. They leverage the design patterns, methodologies and common objects and services as defined within Oracle AIA Foundation Pack.

AIA Integration Types

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.

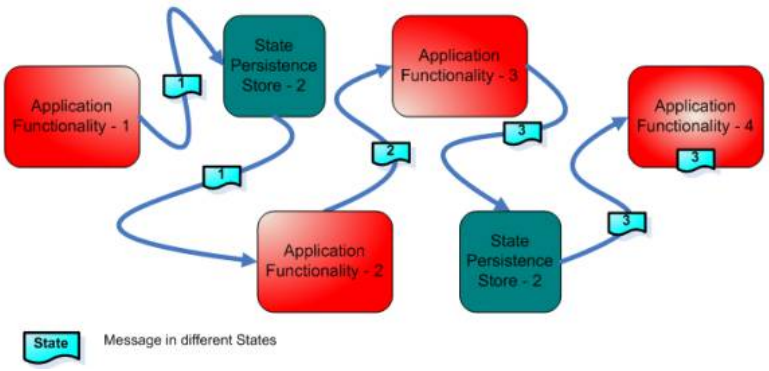
AIA Integration Flows

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 SOA Core Extension.
- Integration styles for bulk data processing.
  - Real-time data integration flow.
  - Batch data integration flow.

How to Use this Developer's 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, which provides:

- 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

This Developer's Guide assumes:

- A Functional Design Document is available.
- Access to AIA software.
- Access to all AIA-provided documents.
- You have read the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack 11g, Release 1 (11.1.1.9.0).*

This Developer's Guide provides:

- An overview of all tasks required to build an AIA integration flow.
- Details about how to develop various AIA artifacts.
- Descriptions of interactions between AIA artifacts and external artifacts.
- Discussion of various design patterns, best practices, and tuning for run-time performance.

Start with Building AIA Integration Flows and proceed to relevant chapters in this Developer's Guide as needed.

2

Building AIA Integration Flows

This chapter describes how to build AIA integration flows.

This chapter includes the following sections:

Setting up Development and Test Environments

For AIA development and testing, the setup of development and test environments consists of downloading the SOA Suite, and setting up your AIA environment.

Getting a Quick Start on your AIA Development

Start your AIA development by installing SOA Suite Quick Start and connecting to a SOA server.

1. Download and install SOA Suite 12.1.3 Quick Start.
2. Create a connection to the SOA Suite server set up for AIA Development and test using these details after Oracle Fusion Middleware is set up:
    a. Connection name: *<give a connection name for your application>*.
    b. Connection Type: use the default value - *WebLogic 12.1.3*
    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>*.

How to Set Up AIA Workstation

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

This section includes the following topics:

Prerequisites

- Install 12.*1.3* WebLogic Server with Oracle Application Development Framework.
- Install Oracle Database.
- Install SOA Suite Quick Start.
- Ensure you comply with the Supported Systems Configuration, as detailed on OTN at, `http://www.oracle.com/technetwork/middleware/fusion-middleware/overview/index.html`

Hardware should be at least 4 CPU and 32 GB of RAM with a supported Operating System.

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 $SOA_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.

For more information see Developing SOA Core Composite Applications on the OTN network.

Content of $SOA_HOME/AIAMetaData

Note:

For new 12c customers you will have the utility objects and receive any other objects through the appropriate pre-built integrations you choose to license.

AIA MetaData ($SOA_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 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

Working with AIA Components Content in $SOA_HOME/AIAMetaData

The AIA Components consist of Schemas, WSDLs, and XSLs shared among various AIA artifacts at runtime. 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:

$SOA_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:

$SOA_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 $SOA_HOME/AIAMetaData/AIAComponents/B2BServiceLibrary/Connectors/wsdls.

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

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

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

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

BusinessProcessServiceLibrary

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

$SOA_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:

$SOA_HOME/AIAMetaData/AIAComponents/EnterpriseBusinessServiceLibrary

EnterpriseObjectLibrary

Part of Oracle AIA Canonical Model

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

$SOA_HOME/AIAMetaData/AIAComponents/EnterpriseObjectLibrary

ExtensionServiceLibrary

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

$SOA_HOME/AIAMetaData/AIAComponents/ExtensionServiceLibrary

The folder structure convention followed is:

ExtensionServiceLibrary/<Application Name>

The possible values for <Application Name> are described in the Examples 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:

$SOA_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:

$SOA_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 section below:

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:

$SOA_HOME/AIAMetaData/AIAComponents/UtilityArtifacts

The folder structure convention followed is:

UtilityArtifacts/schemas

UtilityArtifacts/wsdls

How to Change an Existing File

To change an existing file:

1. From JDeveloper, open the relevant file by browsing to AIAWorkstation/$SOA_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 Updating MDS.

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/$SOA_HOME/AIAMetaData/AIAComponents/<Path to file>.

Note:

Place the file in this folder.

3. Upload to SOA-MDS > apps/AIAMetaData/AIAComponents. Refer to 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 runtime:

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.

How to Work with AIAConfigurationProperties.xml in $SOA_HOME/aia_instances/$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.

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/$SOA_HOME/aia_instances/$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 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.

How to Work with AIAEHNotification.xml in $SOA_HOME/aia_instances/$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/$SOA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/config.
2. Modify as needed.
3. Save.
4. Upload to SOA-MDS > apps/AIAMetaData/config. Refer to Updating MDS.

How to Work with Domain Value Maps in $SOA_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 Working with DVMs and Cross-References.

The DVMs are used by all the AIA Pre-Built Integrations and are shipped as part of SOA Core Extension. They are stored in $SOA_HOME/AIAMetaData/dvm.

To modify the Domain Value Maps:

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

How to Work with Cross Reference (Xref) in $SOA_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 *Developing SOA Applications with Oracle SOA Suite.*

The Cross Reference meta information as used by all the AIA Process Integration Packs are shipped as part of SOA Core Extension and are stored in $SOA_HOME/AIAMetaData/xref.

To modify the Cross Reference metadata:

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

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

The default fault policy file "fault-bindings.xml" is shipped as part of SOA Core Extension and is stored in $SOA_HOME/AIAMetaData/faultPolicies/V1.

To modify the "fault-bindings.xml":

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

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 $SOA_HOME/aia_instances/$INSTANCE_NAME/bin.
2. Source the file aiaenv.sh by executing the following command:

   ```
   source aiaenv.sh
   ```

3. Browse to the folder at $SOA_HOME/aia_instances/$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 SOA_HOME/Infrastructure/Install/config. Execute the script UpdateMetaData.xml by typing the command:

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

Note:

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

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.

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.

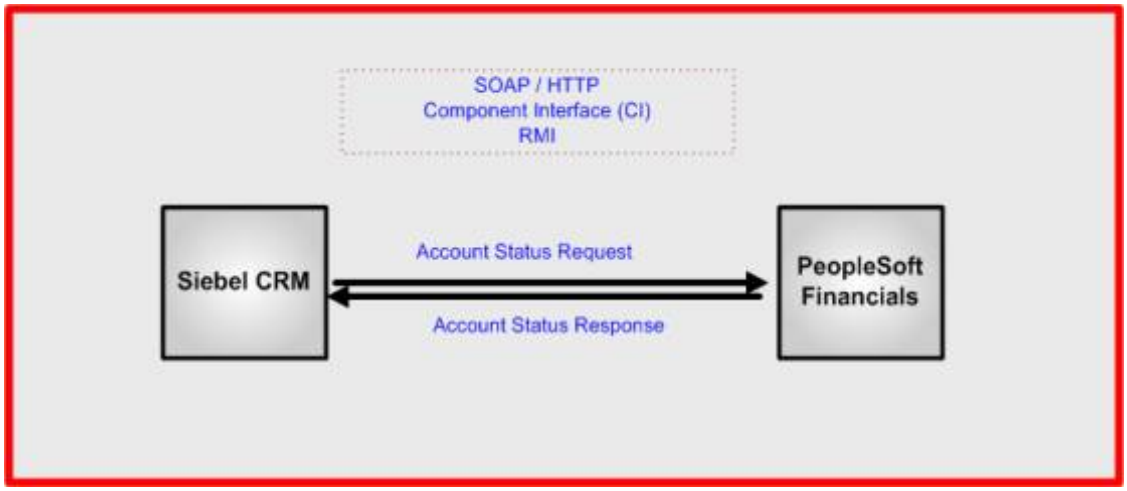
No middleware is involved in this integration.

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 2-1 illustrates how a requester application interacts directly with a provider application.

Figure 2-1 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 Establishing Resource Connectivity.

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:

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

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

For more information about different AIA service artifacts and Oracle SOA Suite components used to implement them, see *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.

Integration Flow with Requester Application Services

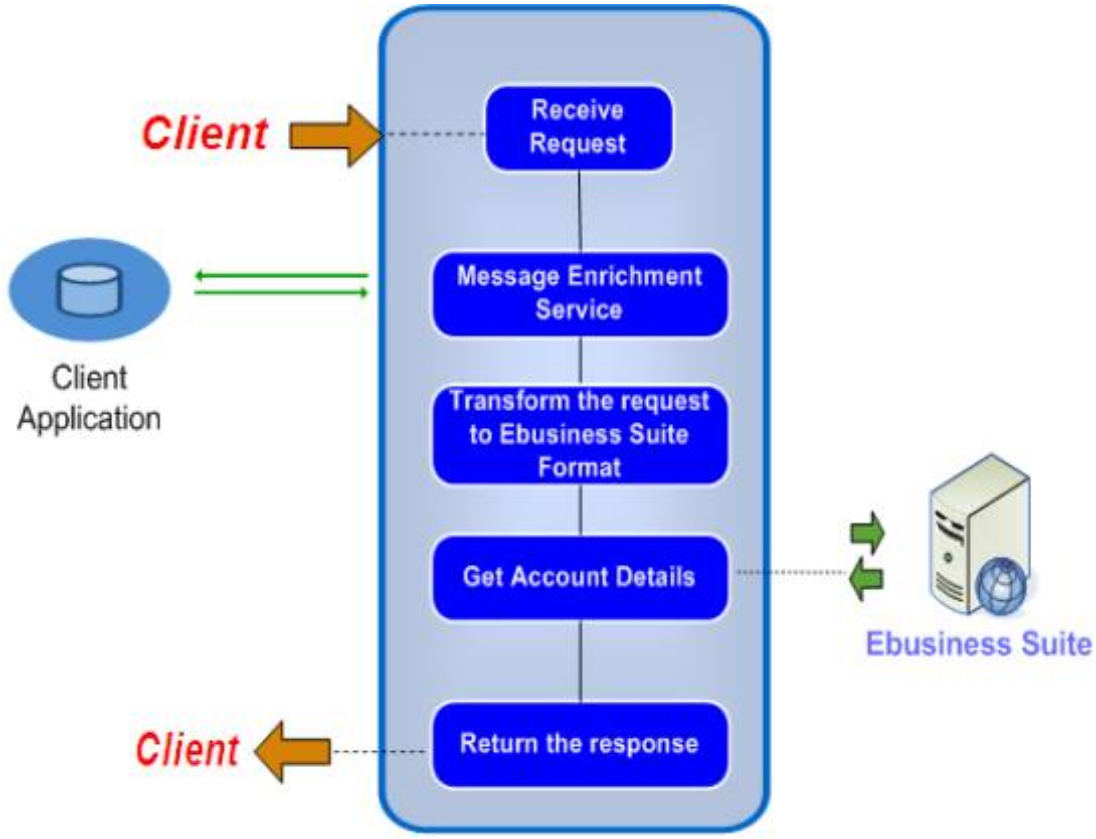
The requester application invokes a single AIA service on the middleware. The request presented by the requester application is fulfilled by the AIA service by invoking suitable APIs in the provider applications. The AIA service can accept a message in the requester application format. The AIA service 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 2-2 illustrates how a service deployed on the middleware enables integration between the requester and the provider application.

Figure 2-2 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	EBF
Asynchronous One-Way	EBS	EBF
Asynchronous Request-Delayed Response	EBS - Request EBS - Response	EBF

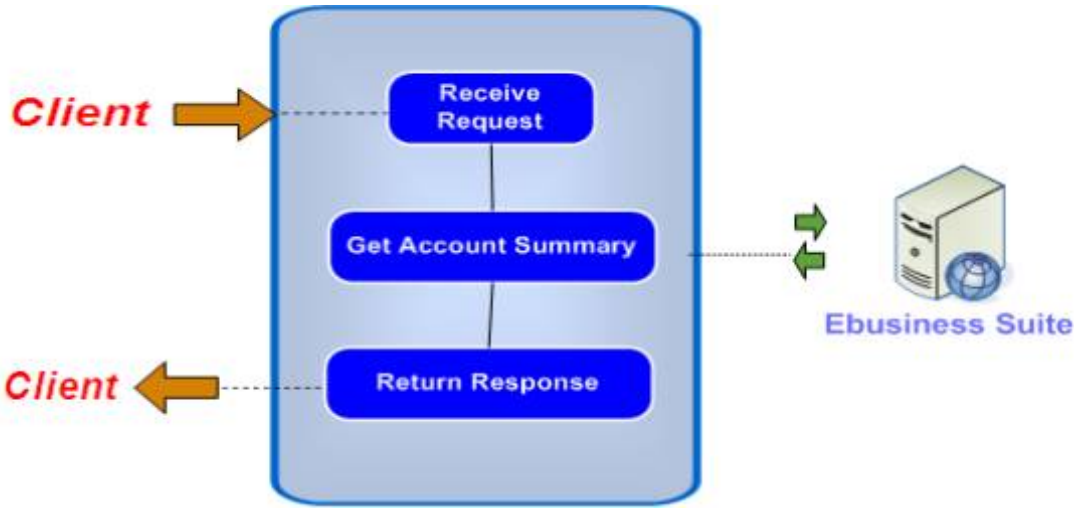
Direct Integration Through Application Web Services

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

Figure 2-3 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

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

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.

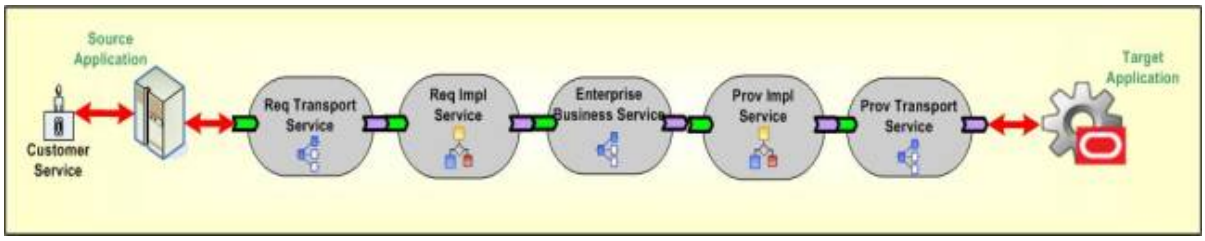
In this case, an EBS based on the Enterprise Business Objects (EBOs) and Enterprise Business Messages (EBMs) is created as a mediator service.

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 2-4 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 2-4 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

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 Using Oracle Data Integrator for Bulk Processing.

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

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

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 SOA Core Extension or construction of EBO. The EBO should incorporate all of the requirements needed by this integration.
- Identification of the EBM from the SOA Core Extension or construction of EBM, if required.

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

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 Establishing the MEP of a New Process EBS, Identifying the MEP, and 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 *Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

2. Identify the EBSs.

See Designing and Developing Enterprise Business Services.

3. Identify the EBFs.

   See Designing and Constructing Enterprise Business Flows.

4. Identify the ABCSs.

   See Designing Application Business Connector Services.

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

   See Designing and Developing Enterprise Business Services.

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

6. Identify and decide on the security model.

   See Working with Security.

   For more information, see "Understanding Security" in *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.

Understanding Oracle AIA Integration

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 Designing and Developing Enterprise Business Services.

2. Enable the participating applications.

   See Enabling and Registering Participating Applications.

3. Construct the ABCS.

   See Constructing the ABCS.

4. Construct the EBF.

   See Designing and Constructing Enterprise Business Flows.

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

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. 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 Designing and Developing Enterprise Business Services

Constructing the ABCSs

For more information, see Constructing the ABCS and Completing ABCS Development.

Enabling and Registering Participating Applications

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

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.

Managing the Oracle AIA System Registry

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

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.

SOA Core Extension provides two ways to manage data in your system registry:

- Systems page
- SystemRegistration.xml configuration file

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.
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.

Page Element	Description
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.
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.

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.

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

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 2-4.

   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 2-5:

   Actual values for the tokens in bold are derived from `$AIA_INSTANCE/config/AIAInstallProperties.xml` during process integration installation using a deployment plan.
3. To delete participating application values from the AIA application registry, use the `<delete>` element provided in the sample XML structure shown in Example 2-6 and replace the system code values in bold with the system codes of the participating applications.
4. Save the `systemRegistration.xml` file to `$SOA_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.

Example 2-1 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.xy.example.com:7001/ecommunications_enu',
SIEBEL','CRM','1.0, 'Siebel contact','1234567891','Siebelcontact@Siebel.com')
/
```

Example 2-2 Sample SQL Content Extracted by AIA Deployment Driver

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

Example 2-3 Sample SQL Content Extracted by AIA Deployment Driver

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 2-1 shows the SQL content extracted by the AIA Deployment Driver:

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 2-2 data provided in `$AIA_INSTANCE/config/AIAInstallProperties.xml`:

Example 2-4 Sample SQL Create Code

```
<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>
```

Example 2-5 Sample Participating Application Code

```
<properties>
   <participatingapplications>
      <SamplePortal>
         <server>
            <soaserverhostname>adc2180948.xy.example.com
            </soaserverhostname>
         </server>
      ...
   </participatingapplications>
</properties>
```

Example 2-6 Sample Delete Code

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

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 Designing and Constructing Enterprise Business Flows.

Testing an Oracle AIA Integration Flow

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

For more information about CAVS, see "Introduction to the Composite Application Validation System" in *Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

3

Implementing Direct Integrations

This chapter provides an overview of direct integrations and describes the procedure for implementing direct integrations.

This chapter includes the following sections:

Understanding Direct Integrations

Oracle SOA Core Extension helps organizations integrate their applications using common information model known as canonical model. Canonical Enterprise Business Objects (EBOs) work best in the integrations where integration flows involve connectivity with multiple source and destination systems. Canonical-based interfaces minimize close coupling between participating applications. This allows re-usability and increases the speed of deployment when new applications are added to the ecosystem.

However, every integration scenario does not require a canonical integration model. SOA Core Extension components also provide support for application-to-application (direct) integration.

You must consider direct integration when:

- There is only one application on each end of the integration.

- There is requirement for fewer mappings.
- The integration requires very low latency.
- The integration is transformation-intensive and requires very high throughput (peaks at over 10,000 messages per hour - or 30 messages per second).
- The integration is process-centric rather than data-centric.
- The requester and the provider applications are not expected to change.

Using the Application Business Flow Design Pattern for Direct Integrations

A simple mode of communication between a consumer and a provider application or services is allowing them to interact directly with each other using the XML-based data structures, as specified by the provider or service.

The Application Business Flow (ABF) manages the integration flow and implements the following tasks necessary for integration:

1. The service receives the request from the client application and does data enrichment, if required.

   Data Enrichment can be described as retrieving additional details from the client application.

2. The service makes calls to the provider using connectivity protocols.

   The response received from the provider application is transformed into the client application format before returning the response to the client application.

   The interface for the integration service is defined by requester / client application.

Note:

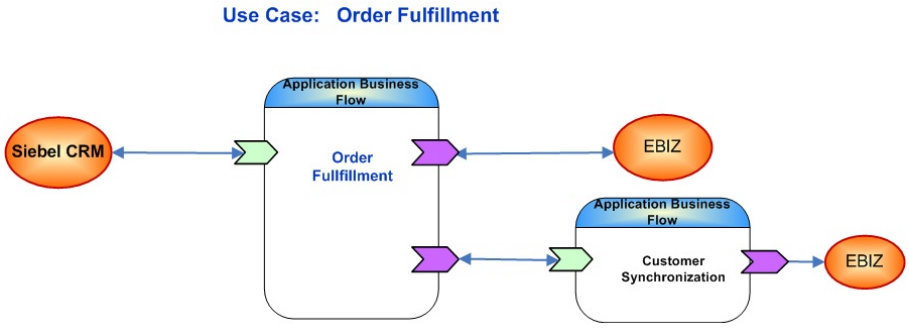
If you introduce any additional provider at a later time, you must add another ABF.

Finding the Correct Granularity

You can implement end-to-end integration flows either using a single ABF composite or multiple ABF composites, depending on your integration requirement. Consider the following when you design your integration flow.

- Define and design reusable services if the functionality of the service could potentially be reused multiple times.
- When some functionality, embedded in a service, is needed in multiple integration scenarios, re-factor it a separate ABF and create a service to reference it.
- Optionally, use an ABF to invoke another ABF, as shown in Figure 3-1, if functionality is available as a separate service.

Figure 3-1 ABF Invoking ABF



Handling Transactions

When an operation defined on an ABF service interface involves destructive writes to two or more resources then Oracle advises you to perform these tasks in a single transaction.

Building an ABF as a Transactional Composite

Design an ABF composite with transactional attributes set on it in the following scenarios:

- Two different applications need to be updated and you want to avoid problems that occur when only one update is successful leading to inconsistency.
- There is an existing global transaction context and the operations performed by your integration process service must be part of the global transaction.
- An application has to send/receive a message and also has to do an update.
- A group of related messages have to be processed in a batch mode.

Achieving Transactionality in the ABF

Create a common transaction context using two-phase commit to achieve transactionality.

The BPEL component creates a new transaction when a BPEL process is invoked. If a transaction already exists, BPEL suspends it and creates a new one. Set the transaction flag on the BPEL component to chain an execution into a single global transaction by adding bpel.config.transaction in the source of composite editor with value as required as shown inExample 3-1.

When the transaction flag is set, BPEL inherits the existing transaction. It creates a new transaction only when no similar transaction exists.

When an ABF service is invoked by another SOA composite ensure a single-threaded execution of the composite. You must implement this only when a one-way operation is being exposed by your service. To accomplish a single threaded execution of the composite, set a sync-type invocation of your composite by the caller composite by setting the property for the BPEL component in your composite as shown below:

```
bpel.config.oneWayDeliveryPolicy=sync
```

Note:

Both the settings can be set while designing your service in JDeveloper.

Example 3-1 Setting the transaction flag on the BPEL component

```
<component name="<BPELProcessName">
    <implementation.bpel  src="<BPELProcessName" >.bpel"/>
    <property name="bpel.config.transaction">required</property>
</component>
```

Enabling Outbound Interaction with Applications

This section describes how to enable outbound interactions with participating applications.

Using JCA Adapters

If JCA Adapters are available, they should be your first choice to connect to applications as they provide support for tasks associated with connection management, transaction management, and scalability in combination with the runtime environment.

Using Standard Web Service Interfaces (SOAP/HTTP, XML/HTTP)

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. Participating applications making an outbound interaction in a synchronous mode alone should send the requests using SOAP/HTTP.

Using JMS Queues

Use queues when the message submission is decoupled from processing of the message. When JMS queues are used, access them using JMS Adapter. The message is enqueued to the JMS queue by the ABF artifact and the application dequeues the message

Using Adapters in a Composite or as a Separate Composite

Develop the adapter as separate service composite only when the same transport adapter service could be used with multiple ABFs. If you do not foresee such reusability include ABF and transport adapter in the same composite.

Invoking an ABF

Communication between an ABF and the application is governed by the application capabilities. You can invoke an ABF in the following ways:

- Using standard web service interface where the client application makes a web service call to the ABF.
- Using another ABF, when the required functionality is available as a separate ABF.
- Using appropriate bindings such as JCA/JMS inside the composite for service endpoints if requester cannot make a web service call.

An ABF is built by including an adapter that can be invoked by the client application.

- Using JMS queues when message submission needs to be decoupled from processing of the message. JMS queues must be accessed using JMS Adapter. The message is en-queued to the JMS queue by the application and ABF de-queues the message. The JMS helps exchange of messages in a persistent manner for guaranteed message delivery.
- Exposing a JMS producer composite if the requester application does not have a capability to drop messages into a queue. Build a JMS consumer adapter into the ABF composite. The JMS consumer adapter dequeues the message and invokes the service provider application. The Asynchronous Queuing design pattern establishes a central queue to allow services to overcome availability issues and increase the overall robustness of asynchronous data exchange.

Handling Errors

This section describes how you can catch errors when implementing direct integration.

Using Direct Integration Services

Enable and configure direct integration services to catch and handle the runtime and business faults. Include all the business faults that you can envisage while deciding WSDL contracts and enable services to return the business fault to the caller of the ABF service.

Oracle recommends that you create a separate catch block for all possible errors and ensure that only unexpected errors go to catchAll block.

The best practice is to invoke a one-way error notification service for sending error notification and error logging.

Using the Fault Management Framework

You can leverage Error Handling framework that comes with the SOA Core Extension for exception capture and exception processing. Oracle recommends that you define a service-specific fault policy file and define both runtime and business faults in it and associate fault policies at composite level.

Use AIAAsyncErrorHandlerBPELProcess of AIA Error Handler framework with direct integration but pass the input in the format that is required by it. Construct an XML element representing an AIAFault element as string before you invoke the AIAAsyncErrorHandlerBPELProcess for business fault handling and error notification/logging.

To invoke the AIAAsyncErrorHanlderBPELProcess, follow the guidelines described in Guidelines for BPEL Catch and Catch-All Blocks in Synchronous Request-Response, except for the step where you apply XSLT EBM_to_Fault.xsl. However, the input message should have the following elements populated with data.

- Fault/FaultNotification/FaultMessage/Text
- Fault/FaultNotification/FaultingService/InstanceID
- Fault/FaultNotification/FaultingService/ECID
- Fault/FaultNotification/FaultingService/ImplementationCode

- Fault/FaultNotification/FaultingService/ID

You can get all the necessary framework behavior including email notifications, and also drill down into it using the EM console.

You cannot access the information that is part of the Fault Message which is derived from the EBM_HEADER. However, the following elements can be derived by the AIA fault handling framework for remote, binding and other faults that are configured in the fault policy xml files.

- Fault Message Code
- Fault Message Text
- Severity
- Stack
- Faulting Service ID
- Faulting Service Implementation Code
- Faulting Service Instance ID
- Corrective Action
- Reporting Date Time

Enable routing of error information through the Notification Setup screen using error code and service name.

Note:

Creation of EBM_Header variable in the BPEL process, which you use for fault handing in canonical integrations, is not relevant in direct integrations.

There is no default support for message resubmission for this release.

Securing the Service

The processes for securing the direct integration services is similar to the way AIA services are secured. The processes are:

- All the web service end points should be secured.
- Individual service components do not have to be secured.
- All services should be provided with authentication credentials.
- Applications invoking secured ABF services need to send required credentials.

Security Recommendations for OWSM Policies for Authentication

The Table 3-1 lists security recommendations for OWSM policies.

Table 3-1 Security Recommendations for OWSM Policies

Policy	Recommendations
oracle/ aia_wss_saml_or_username_token_service_policy_OPT_ON	This is a clone of oracle/wss_saml_or_username_token_service_policy with Local Optimization set to ON. Attach this OWSM security policy to the Service end point(s) of a composite. This policy is applied when authenticating the incoming requests using credentials provided in SAML tokens in the WS-Security SOAP header OR UsernameToken WS-Security SOAP header.
oracle/ aia_wss10_saml_token_client_policy_OPT_ON	This is a clone of oracle/wss10_saml_token_client_policy with Local Optimization set to ON. Attach this OWSM security policy to the Reference end point(s) of a composite. This policy inserts SAML tokens in outbound SOAP request messages.

Note:

The two custom OWSM policies are made available on the server when SOA Core Extension is installed.

Further recommendations:

- Use appropriate client-specific policies for composites that invoke protected application services.
- Use no_authentication_client_policy when invoking application services with no protection.
- Do not have to attach policies for composites calling JCA based application services.

Attaching OWSM Policies to the Composites

Services in direct integration can leverage global policy attachment support provided by the SOA Core Extension. After deployment, SOA Core Extension tools automatically attach global policies to services that have naming pattern ending with ABF.

Services that do not use SOA Core Extension have to attach OWSM policy directly (locally). Attach the policies in the composite.xml file at design time to Service and Reference endpoints. You can also attach OWSM policies to SOA composites, after deployment using EM console.

Using Cross References

Generate a custom database table for each cross reference entity. Oracle recommends that you configure the XREF entity to store the cross references in the custom cross reference table. This does not change the programming model. Wherever the cross reference data is stored, whether it is in SOA infra – XREF_DATA or custom database table, XREF look up and population are done using a single set of APIs.

The Cross Reference editor enables you to specify whether the custom database table needs to be used. (Use the optimize option while defining cross reference artifact.) It also generates the required DDL script. However, SOA does not create the table automatically.

For more information about using custom cross reference tables, see http://www.oracle.com/technetwork/middleware/foundation-pack/learnmore/aiaxref-524690.html.

Note:

The population of cross reference rows in the integration layer can be avoided in situations where the source or target applications have capabilities to manage cross references.

Naming Conventions and Standards

The ABF is modeled as a task-oriented service to capture the process logic. A practical approach is to name services based on the processing and the entity context. Oracle recommends that you use verbs and the entity in service names.

Because ABF is designed to integrate specific applications, use the names of applications as part of the service name.

The template for the name of the ABF is **[Verb][Entity][App1][App2] ABF[Vn if version > 1]**, where App1 is source and App2 is target application.

For example, *ProcessFulfillmentOrderEbizFusionABF*.

You can also add the word **To** between participating source and target applications, if required. For example, *ProcessFulfillmentOrderEbizToFusionABF*.

Additional Guidelines for Naming Services

1. Naming an ABF when invocations to enrichment services are involved

   If integration interacts with an intermediary/utility application such as GOP, Tax Computation and so on to get values to be used for integrating with target application, include the names of these intermediary/utility applications in the service names.

2. Naming an overarching ABF

   An overarching orchestration process contains multiple tasks, wherein each task may involve integration with another application. In such cases do not include all the application names in the service name. Use only the source application name.

   For example, ProcessFulfillmentOrderEbizABF creates an entry in FusionCRM, updates BRM for billing and sends a shipment notification to an external TP through B2BGateway and interface. Because the overarching process may be dictated by source application, use only the source application name.

3. Operation name

   Since the service name covers the broad context, you can have the verb followed by entity in the service operation name.

4. Message name

   The message name should be [name of the operation]Request/[name of the operation]Response

5. Composite Namespace

Composite namespace standard is http://xmlns.oracle.com/ApplicationBusinessFlow/[Functional area or pre-built integration]/[composite name minus the version]/V[n]

For example http://xmlns.oracle.com/ApplicationBusinessFlow/OrderToCash/ProcessFulfillmentOrderFusionDOOToEbiz/V1

6. Namespaces for Messages

   All namespaces must start with http://xmlns.oracle.com/followed by an entity specific and domain specific identifier. For example core, Industry. If there are sub-domains, specify them.

   For example, http://xmlns.oracle.com/ABFMessage/Core/Invoice/V1

7. Namespaces for ABF (WSDL) Interface

   http://xmlns.oracle.com/ApplicationBusinessFlow/\[Functional area or PIP]/[composite name]/V[n]

   For example, http://xmlns.oracle.com/ApplicationBusinessFlow/OrderToCash/ProcessFulfillmentOrderFusionDOOEbizABF/V1

Annotating Composites

The meta information of ABF services is used in maintaining Oracle Enterprise Repository (OER) assets. Oracle advises you to provide annotations in the composites for the exposed services and for the referenced services. Adhere to the proposed guidelines and insert these comments at development time.

Embed the annotations in the <svcdoc:AIA> element and place the <svcdoc:AIA> element inside the xml comments tags <!-- and -- > as shown in the examples in this section.

`<svcdoc:ServiceSolutionComponentAssociation>` must be the first annotation element in every annotated composite.xml file as shown in Example 3-2

This element describes a globally unique identifier (GUID) used for artifacts generation and must be placed under the root element <composite>. The value for this element should be left blank. The GUID associates the solution service component with the implemented business service composite. This association enables auto-population into the Bill Of Material.

Annotate the Service and Reference elements of the composite.xml as shown in Example 3-3 and Example 3-4.

In the previous examples 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 as shown in Example 3-5 and Example 3-6.

```
<svcdoc:Service>
    <svcdoc:ImplementationDetails>
     <svcdoc:ApplicationName>Source Application</svcdoc:ApplicationName>
     ..............................................................
     <svcdoc:ArtifactType>ApplicationBusinessFlow</svcdoc:ArtifactType>
     <svcdoc:ServiceOperation>
    ..................................................................
     </svcdoc:ServiceOperation>
    </svcdoc:ImplementationDetails>
    <svcdoc:TransportDetails>
     <svcdoc:JMSAdapter>
```

```
      <svcdoc:ResourceProvider>WLSJMS</svcdoc:ResourceProvider>
      <svcdoc:ConnectionFactory>JNDINameOfConnectionFactory</
svcdoc:ConnectionFactory>
      <svcdoc:XAEnabled>true</svcdoc:XAEnabled>
      <svcdoc:ResourceTargetIdentifier>JMSUSER1</svcdoc:ResourceTargetIdentifier>
      <svcdoc:ResourceType>Queue</svcdoc:ResourceType>
      <svcdoc:ResourceName>JMSQueueName</svcdoc:ResourceName>
      <svcdoc:ResourceFileName></svcdoc:ResourceFileName>
     </svcdoc:JMSAdapter>
    </svcdoc:TransportDetails>
   </svcdoc:Service>

<svcdoc:Reference>
       <svcdoc:ArtifactType>TransportAdapter</svcdoc:ArtifactType>
       <svcdoc:ServiceOperation>
         <svcdoc:Name>GetSalesOrderLineShippingDetailsEbizAdapter</svcdoc:Name>
        </svcdoc:ServiceOperation>
        <svcdoc:TransportDetails>
        <svcdoc:ApplicationAdapter>
        <svcdoc:ResourceProvider>OracleDB</svcdoc:ResourceProvider>
        <svcdoc:BaseVersion>12.1.3</svcdoc:BaseVersion>
        <svcdoc:ConnectionFactory>eis/Apps/OracleAppsDataSource</
svcdoc:ConnectionFactory>
        <svcdoc:ApplicationName>ebiz</svcdoc:ApplicationName>
        <svcdoc:XAEnabled>true</svcdoc:XAEnabled>
        <svcdoc:ResourceTargetIdentifier></svcdoc:ResourceTargetIdentifier>
        <svcdoc:ResourceType>Procedure</svcdoc:ResourceType>
        <svcdoc:ResourceName></svcdoc:ResourceName>
        <svcdoc:ResourceFileName>XX_BPEL_GETSALESORDERLINESHIPP.sql</
svcdoc:ResourceFileName>
       </svcdoc:ApplicationAdapter>
      </svcdoc:TransportDetails>
</svcdoc:Reference>
```

Example 3-2 First Annotation Element for Every composite.xml File

```
<composite name="SamplesCreateCustomerPartyPortalBusinessService">
.......................
<!--<svcdoc:AIA>
<svcdoc:ServiceSolutionComponentAssociation>
<svcdoc:GUID></svcdoc:GUID>
</svcdoc:ServiceSolutionComponentAssociation>
</svcdoc:AIA>-->
.................................
</composite>
```

Example 3-3 A Skeletal Service Element in composite.xml with Annotations

```
<service ui:wsdlLocation .......>
<interface.wsdl ....... />
<binding.ws ........... />
<!-- <svcdoc:AIA>
<svcdoc:Service>
. . . . . .
</svcdoc:Service>
</svcdoc:AIA> -->
</service>
```

Example 3-4 A Skeletal Reference Element in composite.xml with Annotations

```
<reference ui:wsdlLocation ........">
<interface.wsdl ...........>
```

```
<binding.ws ................./>
<!-- <svcdoc:AIA>
<svcdoc:Reference>
.................
</svcdoc:Reference>
</svcdoc:AIA> -->
</reference>
```

Example 3-5 An Example of Service Annotation Element

```
<svcdoc:Service>
                     <svcdoc:InterfaceDetails>
                        <svcdoc:ServiceName>.....</svcdoc:ServiceName>
                        <svcdoc:Namespace>.......</svcdoc:Namespace>
                        <svcdoc:ArtifactType>ApplicationBusinessFlow</
svcdoc:ArtifactType>
                        <svcdoc:ServiceOperation>
                           .......
                        </svcdoc:ServiceOperation>
                     </svcdoc:InterfaceDetails>
                     <svcdoc:ImplementationDetails>
                        <svcdoc:ApplicationName>Source Application</
svcdoc:ApplicationName>
                        .................
                        <svcdoc:ArtifactType>ApplicationBusinessFlow</
svcdoc:ArtifactType>
                        <svcdoc:ServiceOperation>
..........
                        </svcdoc:ServiceOperation>
                     </svcdoc:ImplementationDetails>
 </svcdoc:Service>
```

Example 3-6 An Example of Service Annotation Element When It Has a Transport Adapter

```
<svcdoc:Reference>
                       <svcdoc:ArtifactType>UtilityService</svcdoc:ArtifactType>
                       <svcdoc:ServiceOperation>
                       <svcdoc:Name>initiate</svcdoc:Name>
                       </svcdoc:ServiceOperation>
</svcdoc:Reference>
```

Annotating WSDLs

WSDL annotations are similar to an existing AIA service annotations. Look at Example 3-7.

Example 3-7 A Sample WSDL Annotation

```
<documentation>
<svcdoc:Service>
<svcdoc:Description></svcdoc:Description>
<svcdoc:ServiceType>ApplicationBusinessFlow</svcdoc:ServiceType>
<svcdoc:Version>2</svcdoc:Version>
</svcdoc:Service>
</documentation>
```

Service Configuration

Direct integrations services that use SOA Core Extension tools can use the existing approach for setting up service level configuration properties in an XML file. However, Oracle recommends that you use BPEL preferences by leveraging BPEL preferences/properties feature of SOA for freshly built direct integration flows. In the composite use the property and define the configurable preferences.

```
<property name="bpel.preference.myProperty">myValue</property>
```

Configuration updates to the defined properties such as add/delete/modify can be made directly through EM console. If you use SOA BPEL preferences for service configuration you do not require a separate UI for updating the properties.

There are limitations with this approach. This approach:

- Does not have support in XSL transformations
- Can be used only from BPEL. Does not have support in Mediator
- Has no support for Domain level configurations
- Has non-intuitive navigation in the EM console to the preferences

The AIA Service Configuration file can be used to configure runtime behavior of services by the direct integration flow when SOA Core Extension is installed.

Extending Direct Integrations

This section discusses the extensibility patterns for Direct Integrations. This section includes the following topics:

Extending XSLs

Develop the base transformation XSL file to provide extension capabilities using the following guidelines.

- To make transformation maps extension aware, 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.

- Define empty named templates in the extension file for every component in the message.
- Include the extension file in the core transformation file.
- Add a call-template for each component in core transformation.
- Enable the transformation to accommodate transformations for custom element as shown in Example 3-8.

Example 3-8 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>
```

Extending Services

- Mark your composite as customizable only if you have a specific customization use case.
- Mark specific scopes in your BPEL process as customizable only if there is a specific customization use case.

Guidelines for Enabling Customizations

The section discusses general guidelines for identifying the BPEL scope to be marked as customizable.

The scope of the customizable scope activity should be made as narrow as possible. For example, rather than marking the top level scope of the BPEL flow as customizable, identify a specific scope where the logic can be added to meet customization requirements.

- Avoid marking complex logic which could be partially deleted later as customizable.
- If a composite is marked as customizable, do not unmark it once it is released. If you unmark it, the customization will not survive patching.
- Do not make changes to the base version which, in a way, could cause conflict when the customized composite is patched. For example, deleting components, routing rules and activities in the customizable BPEL scope requires the customer to manually resolve the merge conflict when customized item is deleted.
- Do not delete any component or routing rule from a customizable composite and do not delete any activities in a BPEL scope marked as customizable once the composite is released.

4

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:

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.

  Example 4-1 illustrates 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 4-1 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;
}
. . .
}
```

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

Function Name

The function name must follow the standard Java method naming standards where the name should be Lower-Camel-Case.

Example: `getCurrentDate`, `getEBMHeaderValue`, ...

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`, ...

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`, ...

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`, ...

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

Function Namespace Prefix

The namespace prefix must be a readable abbreviation based on functionality.

Example: ebh for `GetEBMHeaderValueFunction`.

Supported Data Types

The XPath 1.0 and XSL 1.0 data types are supported as return or input parameters to the function, as shown in Table 4-1:

Table 4-1 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

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], as shown in Example 4-2, and register it with JDeveloper through FilePreferencesXSL Map.

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.

Example 4-2 User Defined Extension Functions Config File

```
<?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>
```

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.

5

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:

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 *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 *Developing SOA Applications with Oracle SOA Suite.*

You can model EBS operations either as synchronous or asynchronous message exchange patterns (MEPs).

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 *Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

Working with the Enterprise Business Service Library

SOA Core Extension 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 SOA Core Extension 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.

Designing the EBS

This section includes the following topics:

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 SOA Core Extension.

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.

Understanding Design Considerations

When designing EBS, consider the following:

- What is the service for?

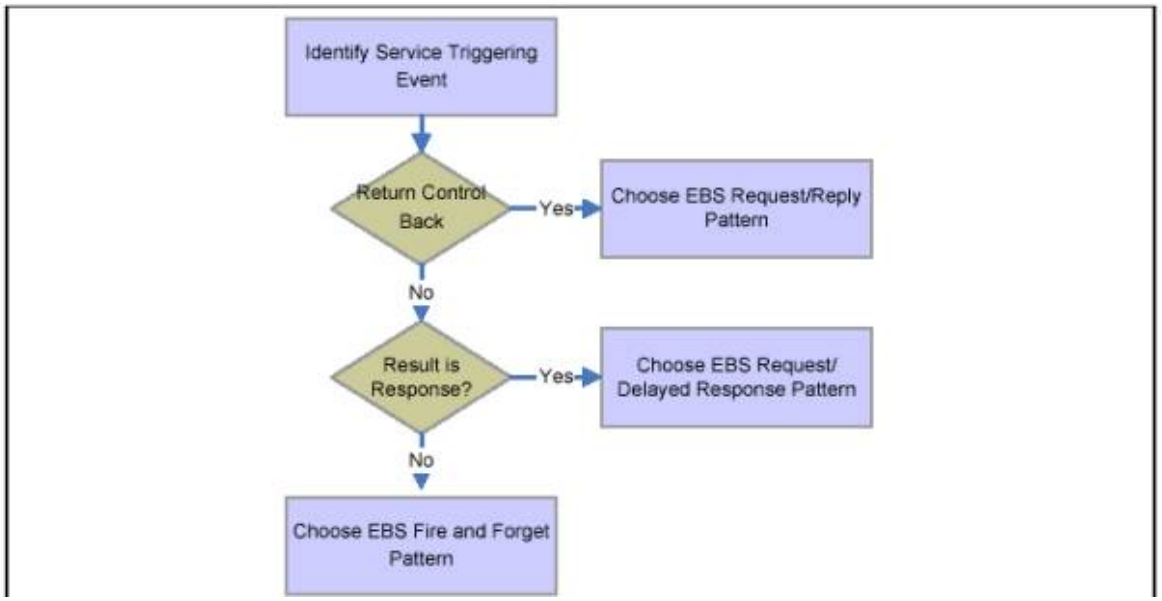
  - Your enterprise should have only one EBS for a particular business function, and it should be independent of any application.
  - The service can implement multiple business operations defined in EBS WSDL.
  - It should have implementations for all the business operations for a business object.
  - 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.

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 5-1 illustrates the decision points in establishing the EBS pattern.

Figure 5-1 Identifying the Interaction Pattern for EBS Operations



For more information about EBS types, see "Understanding Enterprise Business Services" in *Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

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.

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 Configuring Oracle AIA Processes for Error Handling and Trace Logging.

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 Working with Security.

For more information about EBS, see "Understanding Enterprise Business Services" in *Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

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.

How to Guarantee Delivery

For details about how to guarantee message delivery, see Guaranteed Message Delivery.

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.

Constructing the WSDL for the Process EBS

This section includes the following topics:

- Introduction to WSDL Construction for the Activity Service EBS
- How to Complete the <definitions> Section
- How to Define Message Structures
- How to Check for WS-I Basic Profile Conformance

These sections describe how to fill in sections of the WSDL.

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.
- You should adhere to the naming conventions for creating service, target namespace, messages, operations, port type, and so on as described in Oracle AIA Naming Standards for AIA Development.

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.

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.

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.

Working with Message Routing

This section includes the following topics:

- Creating Routing Rules
- Routing at the EBS
- Guidelines for EBS Routing Rules
- How to Identify the Target System at EBS

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 runtime, 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 runtime.

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 5-1 lists the routing rule clauses and the related XPath expressions.

Table 5-1 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 5-2 shows some routing rules delivered as part of the Integrated Supply Chain Management Pre-Built Integration.

Table 5-2 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.
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'

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.

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 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 Configuring Oracle AIA Processes for Error Handling and Trace Logging.

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.

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.

How to Develop the Oracle Mediator Service

To develop the Oracle Mediator Service:

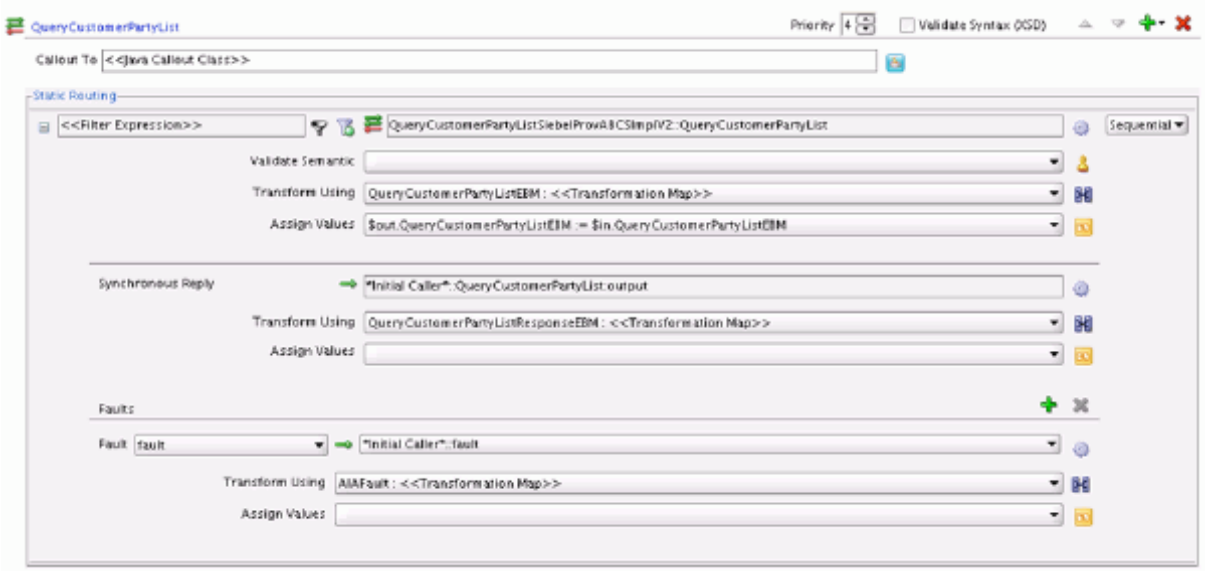
1. In JDeveloper, create an SOA composite project.
2. Open the **composite.xml** in Design mode.
3. Add a Mediator component in the components swim lane.
4. Add a Web service as external reference service in the references swim lane.

   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 5-2.

Figure 5-2 Creating an Assign



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:

- How to Implement Fire-and-Forget Pattern with EBS One-Way Calls
- Creating EBS WSDLs
- Creating Mediator Routing Services for Asynchronous Fire-and-Forget Patterns with a One-Way Call EBS
- Asynchronous Fire-and-Forget MEP Error Handling Using Compensatory Operations
- How to Invoke the Compensate Operation of EBS
- How to Enable Routing Rules in Compensate Operation Routing Service

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 Designing Application Business Connector Services, Constructing the ABCS,, and Completing ABCS Development.

For more information about enabling the EBF, see 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.

Creating EBS WSDLs

For the entity EBS, use the WSDLs from the Enterprise Service Library of the SOA Core Extension.

- 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.

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.

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.)

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 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.

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 How to Create Routing Services for the Synchronous Request-Response MEP.

How to Implement Error Handling for Asynchronous Fire-and-Forget MEP

For more information, see Configuring Oracle AIA Processes for Error Handling and Trace Logging.

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.

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.

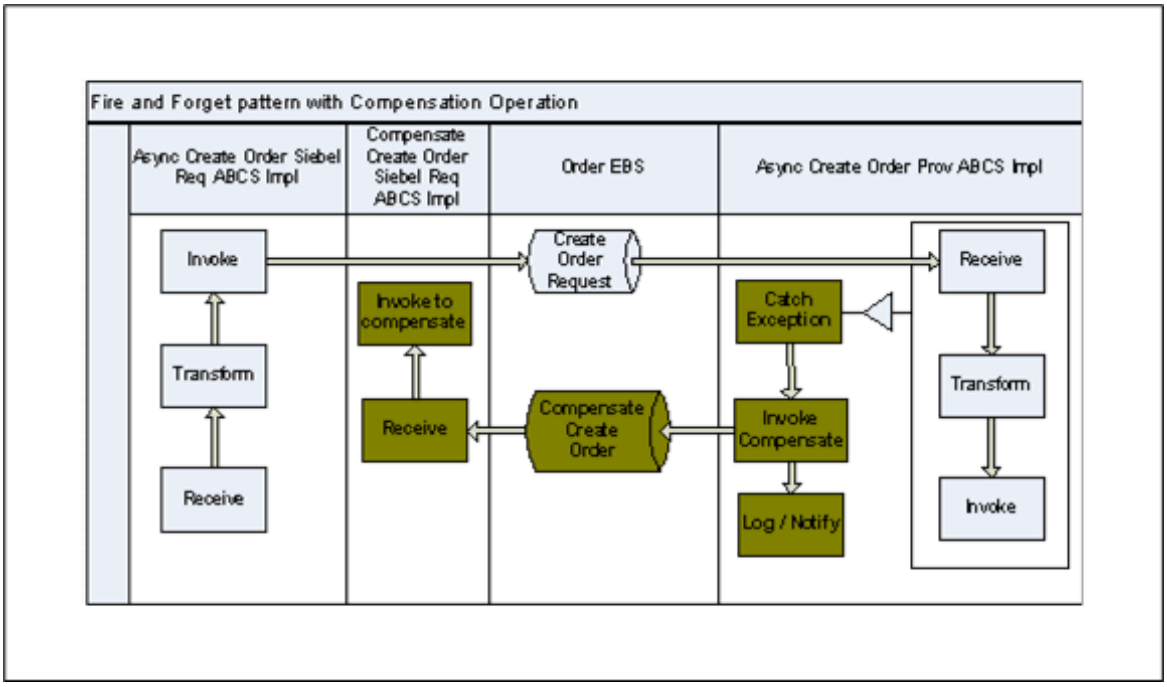
To invoke the compensate operation of the EBS:

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 5-3 illustrates the fire-and-forget pattern with the compensation operation.

Figure 5-3 Fire-and-Forget Pattern with Compensation Operation



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

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:

- How to Implement Synchronous Request-Reply Message Exchange Patterns in EBS
- How to Create Mediator Projects for the Synchronous Request-Response MEP
- How to Create Routing Services for the Synchronous Request-Response MEP
- How to Implement Error Handling for the Synchronous Request-Response MEP

For more information about enabling the ABCS (both requester and provider), see Designing Application Business Connector Services, Constructing the ABCS, and Completing ABCS Development.

For more information about enabling the EBF, see Designing and Constructing Enterprise Business Flows.

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.

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 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.)

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 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.

How to Implement Error Handling for the Synchronous Request-Response MEP

For more information, see Configuring Oracle AIA Processes for Error Handling and Trace Logging.

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:

- How to Implement the Request-Delayed Response Pattern with the Two One-Way Calls of the EBS
- Creating Mediator Routing Services for Asynchronous Request-Delayed Response Patterns with Two One-Way Call EBS
- Asynchronous Request-Delayed Response MEP Error Handling

For more information about enabling the ABCS (both requester and provider), see Designing Application Business Connector Services, Constructing the ABCS, and Completing ABCS Development.

For more information about enabling the EBF, see Designing and Constructing Enterprise Business Flows.

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:

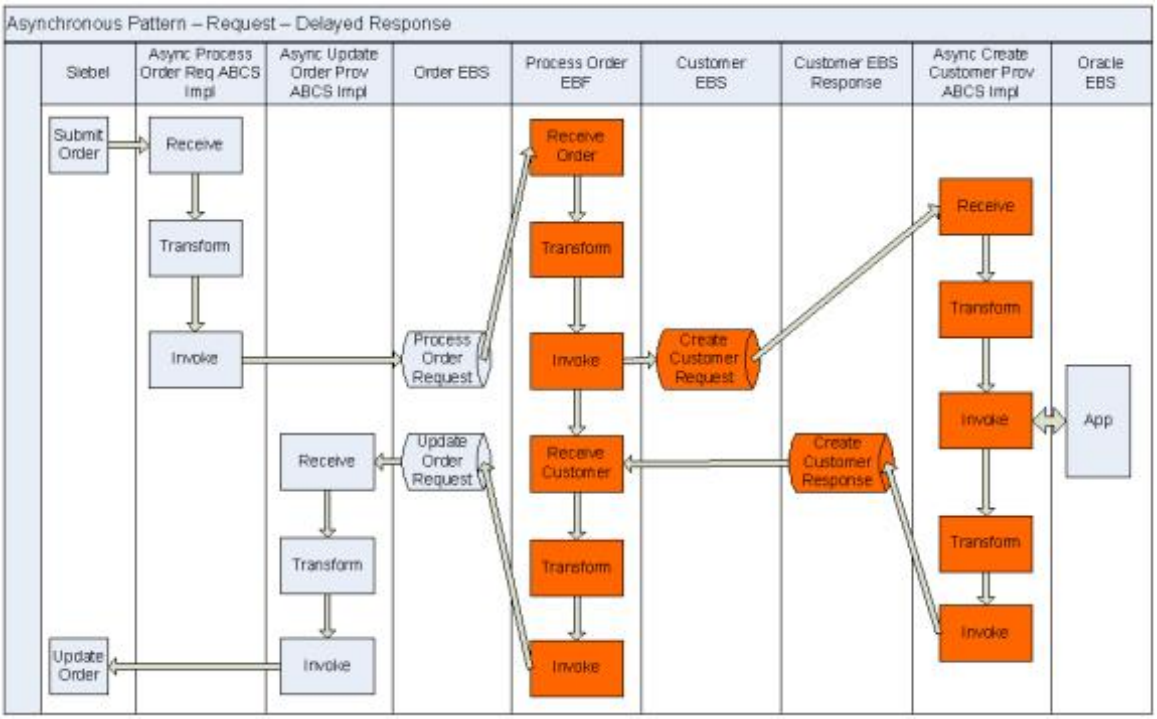
Note:

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 5-4 illustrates the request-delayed response pattern:

Figure 5-4 Request-Delayed Response Pattern



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.

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 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.)

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 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.

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 How to Create Routing Services for the Synchronous Request-Response MEP.

Routing Rules for Response EBS

There must be two routing rules.

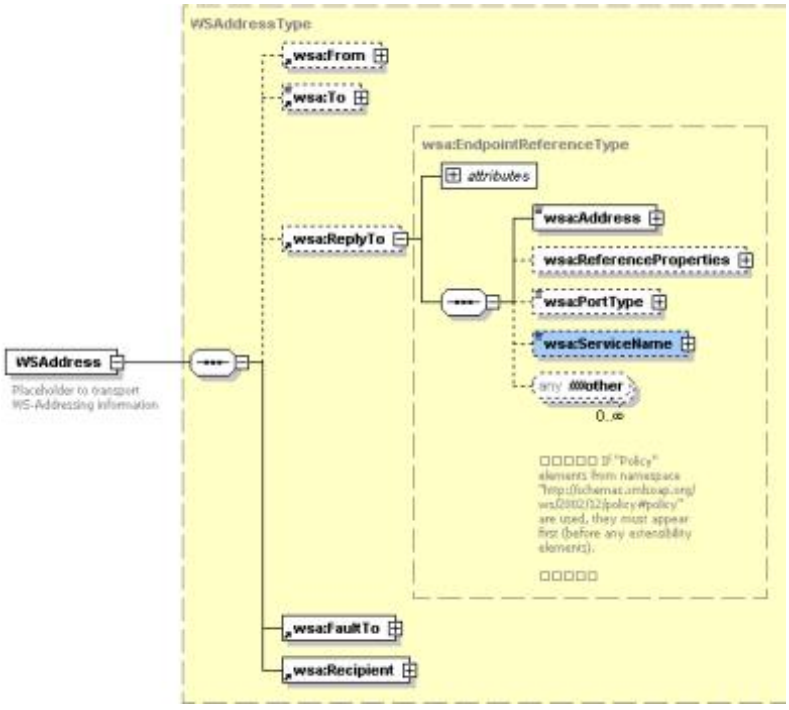
To create routing rules for the response EBS:

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 5-5, 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 5-5 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 Designing Application Business Connector Services, Constructing the ABCS, and Completing ABCS Development.

Error Handling Implementation

For more information, see Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery.

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 Designing Application Business Connector Services, Constructing the ABCS, and Completing ABCS Development.

6

Designing Application Business Connector Services

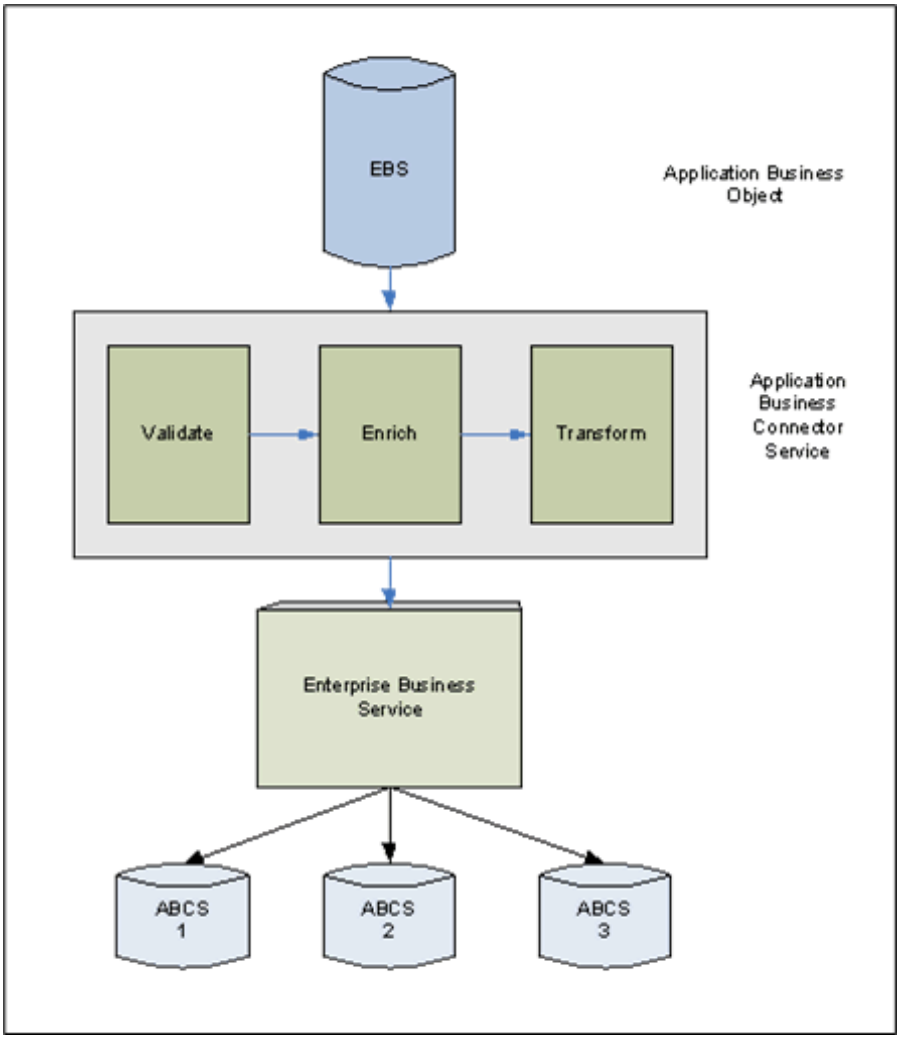
This chapter provides an overview of Application Business Connector Services (ABCS) describes how to define the ABCS contract, identify the MEP and discusses technology options for outbound interaction with the application and how to use BPEL for building ABCS.

This chapter includes the following sections:

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 6-1 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 6-1 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 *Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

ABCS Types

The two types of ABCSs are:

- Requester ABCS
- Provider ABCS

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.

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 *Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

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:

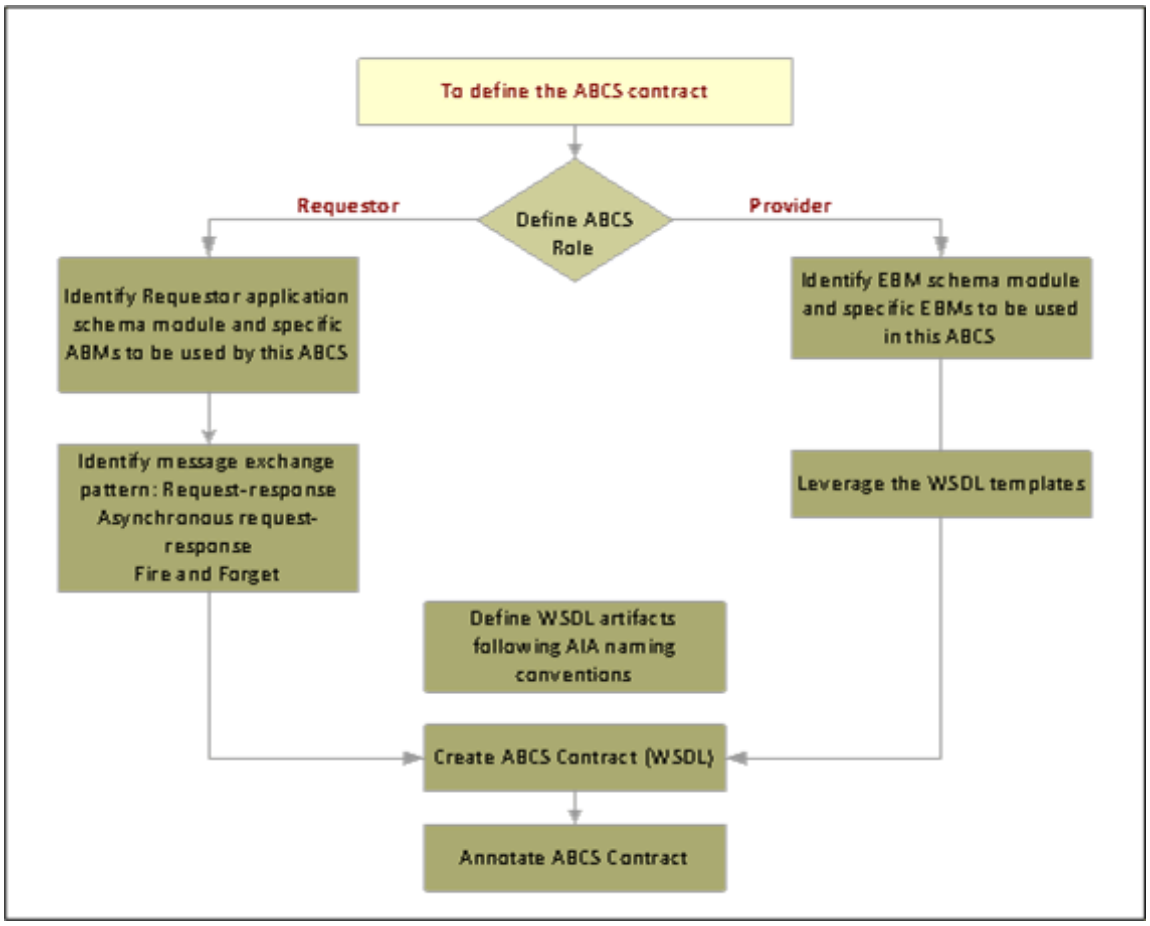
- Defining the ABCS Contract
- Identifying the MEP
- Technology Options

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 6-2 illustrates the decision flow for defining the contract (WSDL) for the ABCS.

Figure 6-2 Decision Flow for Defining the ABCS Contract



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.

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 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 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?

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 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.

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.

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 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.

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 *Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

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 *Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack.*

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 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.

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 Query.

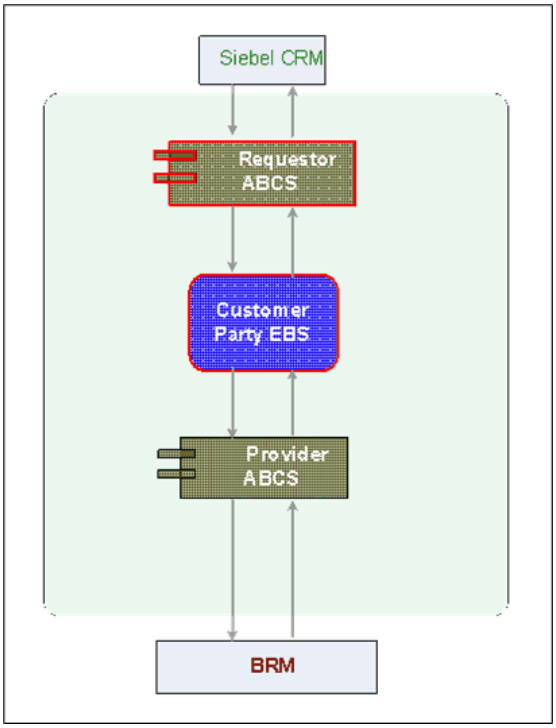
For more information about Validate, see Validate.

Use Case

CRM Application requesting account details about a customer from the billing system

Figure 6-3 illustrates the two-way communication between AIA services, requester ABCS, and CustomerPartyEBS.

Figure 6-3 GetAccountBalanceSummary Integration Scenario



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

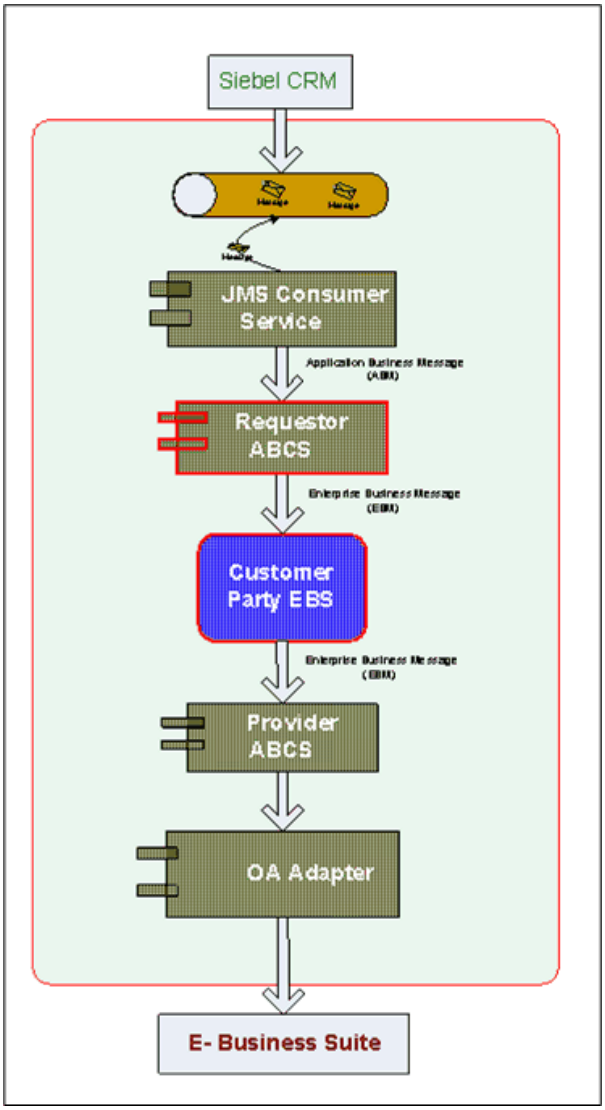
A CRM application submitting a request to the BRM application to create CustomerParty

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 6-4 illustrates the one-way invocation between AIA services, Requester ABCS, and CustomerPartyEBS.

Figure 6-4 Example of One-Way Invocation



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.

Technology Options

This section discusses technology options for:

- Outbound Interaction with the Application

  For inbound interactions see Invoking the ABCS.

- Using BPEL for Building ABCS

For more information about resource connectivity, see Establishing Resource Connectivity.

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

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 Establishing Resource Connectivity.

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.

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 *Administering JMS Resources for Oracle WebLogic Server.*

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 *Developing SOA Applications with Oracle SOA Suite.*

7

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.

Note:

Composite Business Processes (CBP) will be deprecated from next release. Oracle advises you to use BPM for modeling human/system interactions.

This chapter includes the following sections:

Constructing an ABCS

Table 7-1 lists the common tasks for constructing an ABCS and provides links to detailed information:

Table 7-1 Summary of Tasks for Constructing an ABCS

If	Refer to
You are implementing the Fire-and-Forget Message Exchange Pattern (MEP)	When to Use the Asynchronous Request Only (Fire-and Forget) MEP Implementing the Fire-and-Forget MEP To implement the MEP in EBS, see Implementing the Fire-and-Forget Message Exchange Pattern.

If	Refer to
You are implementing the Asynchronous Request Delayed Response MEP	When to Use the Asynchronous Request Delayed Response MEP Implementing the Asynchronous Request Delayed Response MEP To implement the MEP in EBS, see Implementing the Asynchronous Request-Delayed Response Message Exchange Pattern.
You are implementing the Synchronous Request - Response MEP	When to Use the Synchronous Request-Response MEP Implementing Synchronous Request-Response Message Exchange Scenario To implement the MEP in EBS, see Implementing the Synchronous Request-Response Message Exchange Pattern.
Your ABCS is invoking an Enterprise Business Service (EBS) operation	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	Working with Message Transformations.
You need information about how ABCS can be connected with the participating applications both inbound and outbound	Establishing Resource Connectivity.
You must know how clients such as applications and other services can invoke the ABCS you are developing	Invoking the ABCS
You are securing an ABCS	Securing the ABCS
You are enabling ABCS to handle errors and faults	Handling Errors and Faults
You want to allow ABCS to be extended by customers	Developing Extensible ABCS
You need guidelines on Composite Application Validation System (CAVS)-enabling the ABCS	Developing ABCS for CAVS Enablement
You want to deploy the ABCS you are developing using AIA Installation Driver	Building AIA Integration Flows.

If	Refer to
You want to access AIA Configuration Properties from within an ABCS	How to Work with AIAConfigurationProperties.xml in $SOA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/config.

Prerequisites

Before you begin constructing an ABCS, ensure that:

- The relevant development SOA server with a SOA Core Extension is installed and running.

  Refer to 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 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 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

  For more information, see 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 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 Establishing Resource Connectivity.

- Any participating application-specific requirements such as error handling or security have been identified.

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.

How Many Components Must Be Built

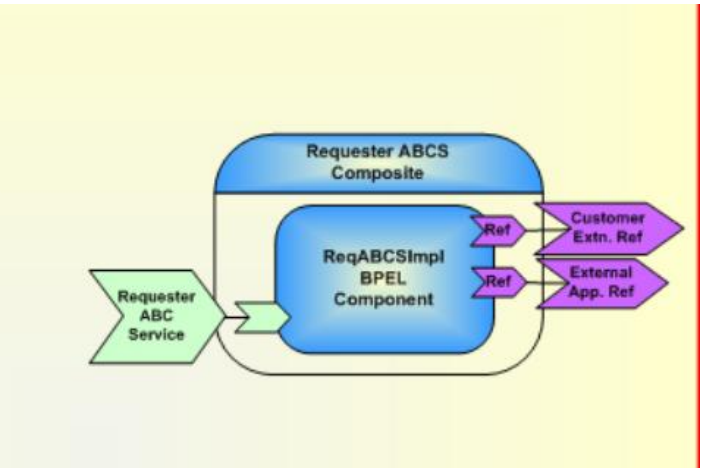
A composite application can be thought of as a composition of related SCA components that collectively provide a coarse-grained business function. A composite may contain certain components that might be required only for the internal implementation of the composite and for which interfaces might not 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 7-1 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.

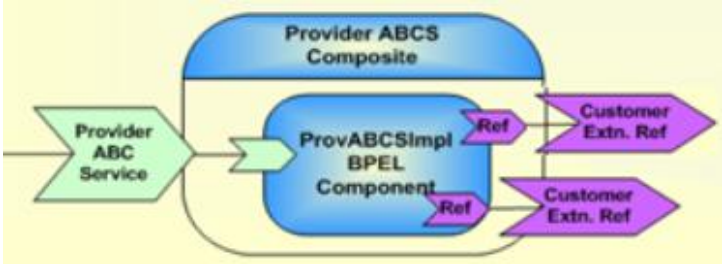
Figure 7-1 Example of a Requester ABCS Composite



Similarly, a Provider ABCS composite as shown in Figure 7-2 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 7-2 Example of a Provider ABCS Composite



Constructing an ABCS Composite Using JDeveloper

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 Using MDS in AIA.

- Develop abstract WSDL for the ABCS.

  See Designing Application Business Connector Services.

- Construct the ABCS composite using JDeveloper.

  See How to Construct the ABCS Composite Using JDeveloper.

- Develop the BPEL process.

  See Developing the BPEL Process.

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.

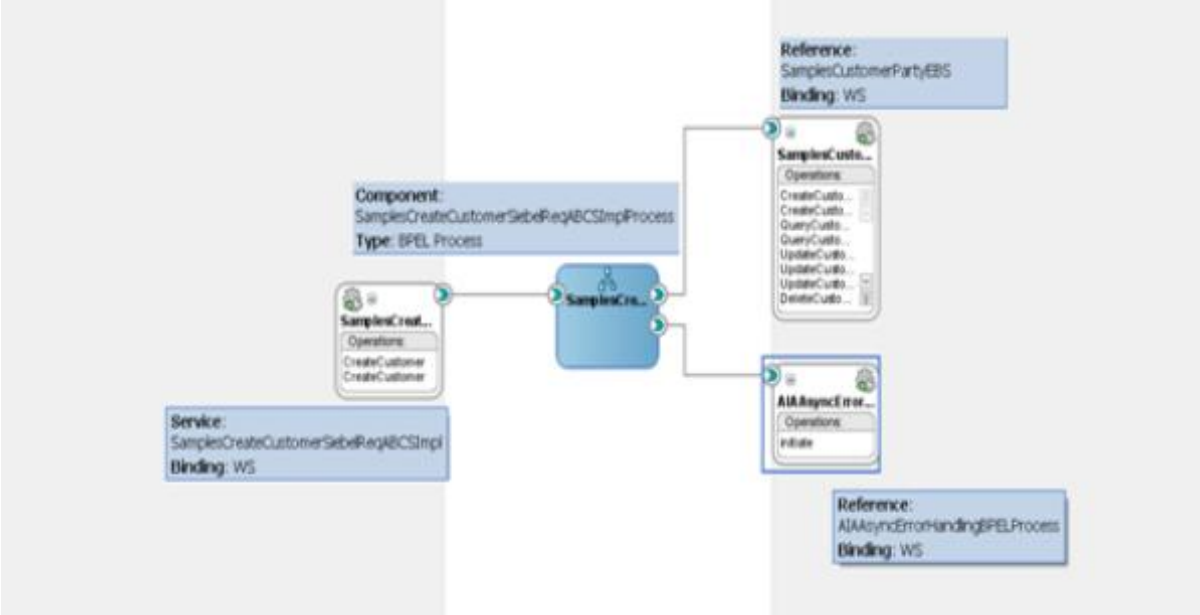
   This reference service could represent an EBS, an Adapter Service, or a participating application. An abstract WSDL of the EBS, Adapter Service, or participating application should be accessible from the MDS Repository.
5. Wire the BPEL component to the external reference component created in step 4.
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 Configuring Oracle AIA Processes for Error Handling and Trace Logging.

Figure 7-3 illustrates a requester ABCS composite.

Figure 7-3 Example of a Requester ABCS Composite



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 Completing ABCS Development.

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.

Moving Abstract Service WSDLs in MDS

SOA Suite 12c has introduced a central repository, **Metadata Service (MDS)**, that backs up your application (and hence your composites) at design time and runtime. MDS is like a version management system, where you can store and use the shared common artifacts at design and runtime.

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 Separation of Concerns.

For details of how MDS is used in AIA, refer to 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 Using MDS in AIA.

Examples:

```
AIAMetaData/AIAComponents/ApplicationConnectorServiceLibrary/Siebel/V1/
RequesterABCS
AIAMetaData/AIAComponents/ApplicationConnectorServiceLibrary/Siebel/V1/
ProviderABCS
```

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 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 Working with Message Transformations.

For more information about how to handle faults, see 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 *Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

This section includes the following topics:

- When to Use Compensating Services
- How to Invoke the Compensating Service
- Additional Tasks Required in Provider ABCS to Implement This MEP
- How to Ensure Transactions
- How to Handle Errors

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.

How to Invoke the Compensating Service

To invoke the correct compensating service from the providing service:

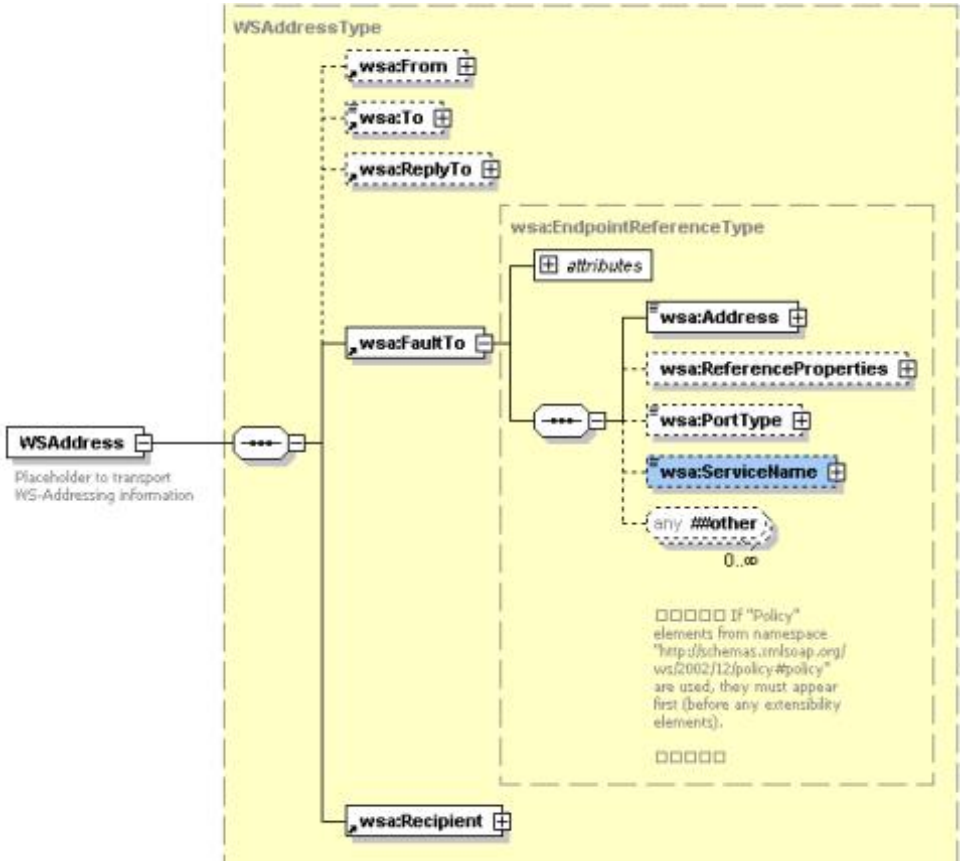
1. Populate the EBMHeader/Sender/WSAddress/wsa:FaultTo/wsa:ServiceName with the name of the compensating service in the transformation used for constructing the request EBM.

   Example of the name of the compensation service:
   CompensateCreateOrderSiebelReqABCSImpl

   For more information about naming, see Oracle AIA Naming Standards for AIA Development.

   Figure 7-4 illustrates the structure of the **WSAddressType**.

Figure 7-4 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.

Additional Tasks Required in Provider ABCS to Implement This MEP

For information about implementing this MEP, see Implementing Provider ABCS in an Asynchronous Message Exchange Scenario.

How to Ensure Transactions

For information about ensuring transactions, see How to Ensure Transactions in AIA Services.

How to Handle Errors

For information about error handling, see Configuring Oracle AIA Processes for Error Handling and Trace Logging.

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 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:

How to Populate the EBM Header for Asynchronous-Delayed Response

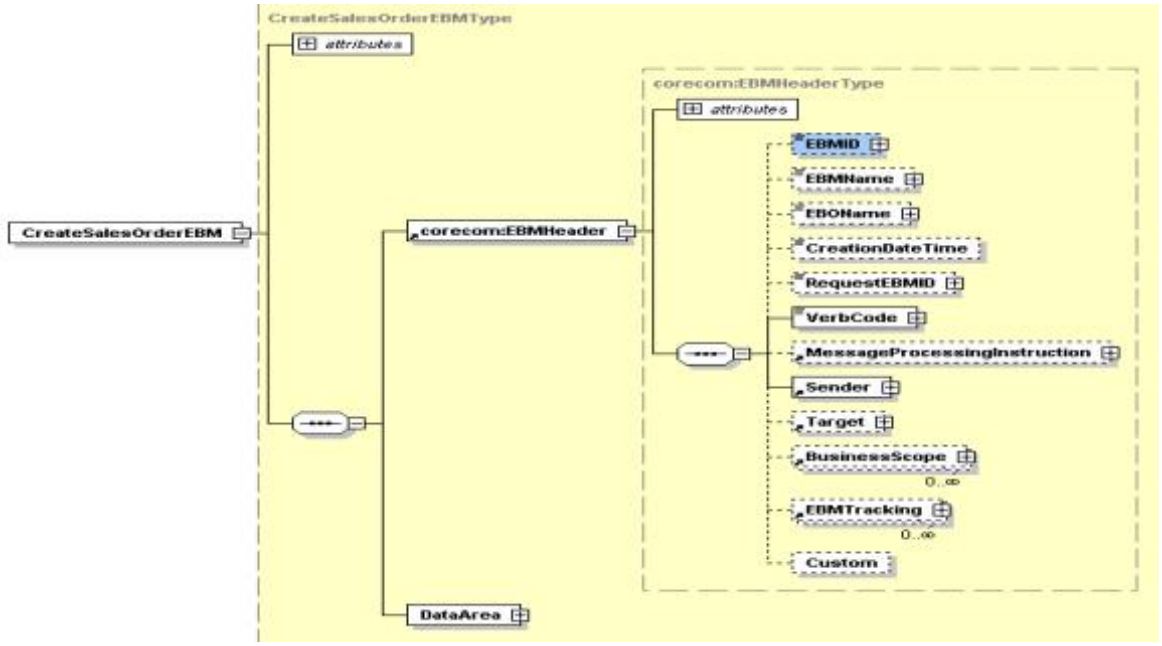
To populate the EBM header for asynchronous delayed response:

1. Populate the **EBMID**.

   This is used for correlation of the response to the correct requesting service instance.

   Figure 7-5 illustrates the structure of the **CreateSalesOrderEBMType**.

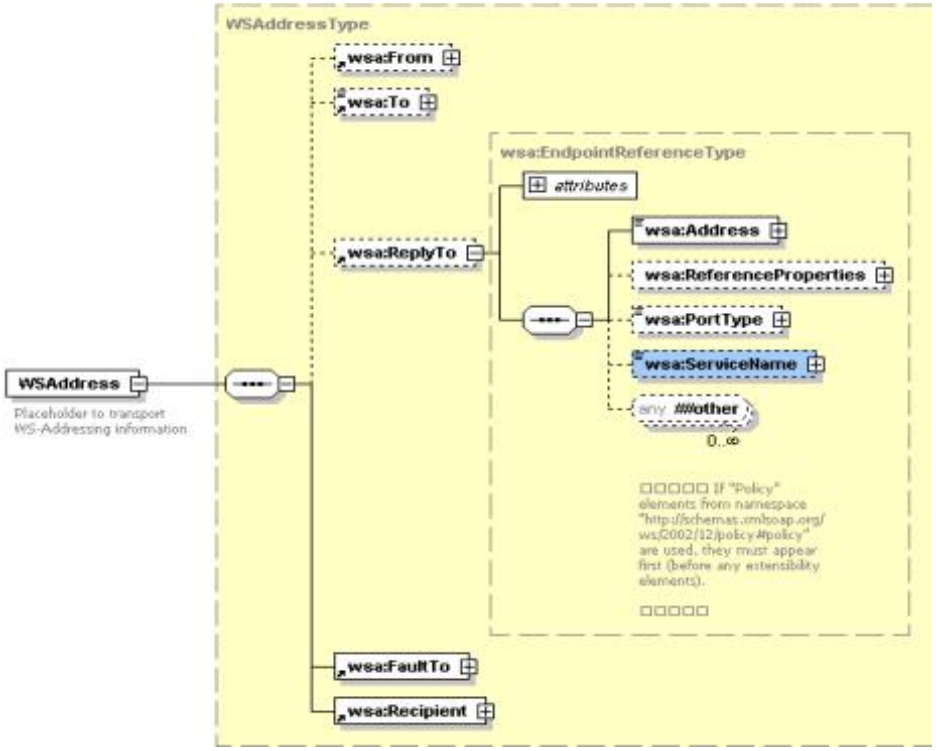
Figure 7-5 Structure of the CreateSalesOrderEBMType



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 7-6.

   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 7-6 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.

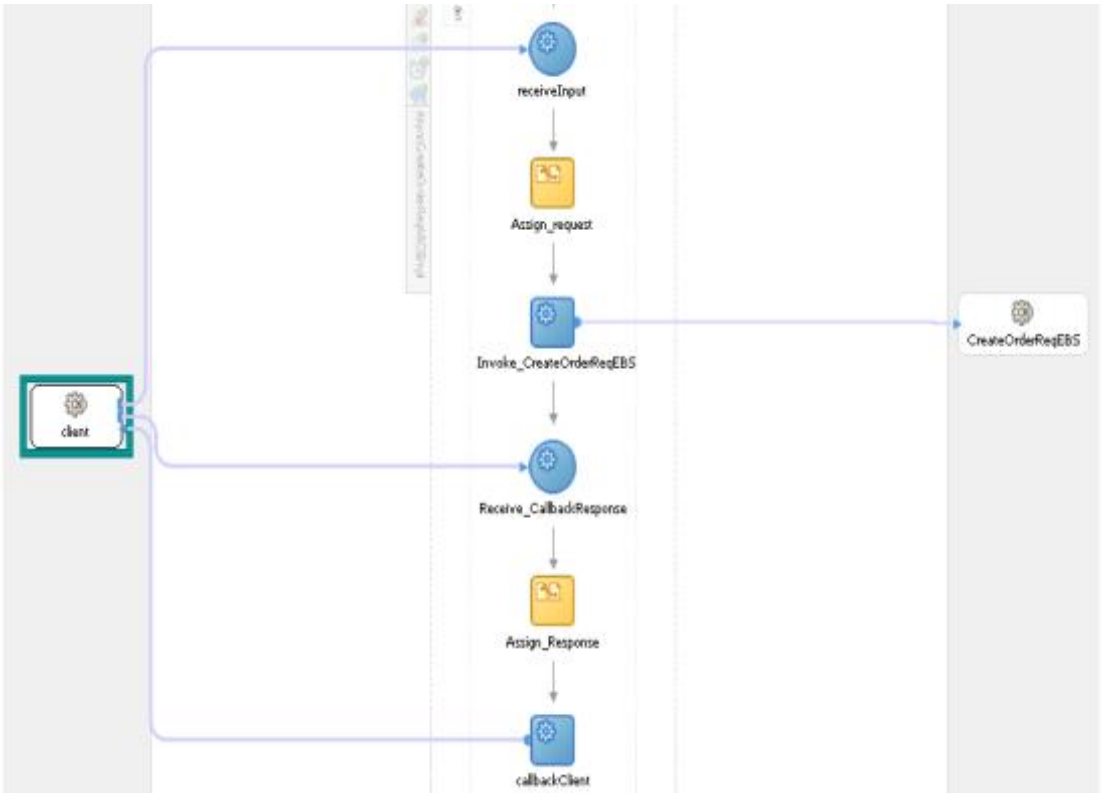
   The **responseCode** attribute of the EBM verb is set to indicate to the providing service that the requesting service is expecting a response. This is evaluated in the providing service to send back a response.

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 7-7. This is the process for which you must set the correlation IDs.

Figure 7-7 Example of Requester ABCS Process



Setting Correlation for the Asynchronous Request-Delayed Response MEP

In the process described in How to Populate the EBM Header for Asynchronous-Delayed Response 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*.

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

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.

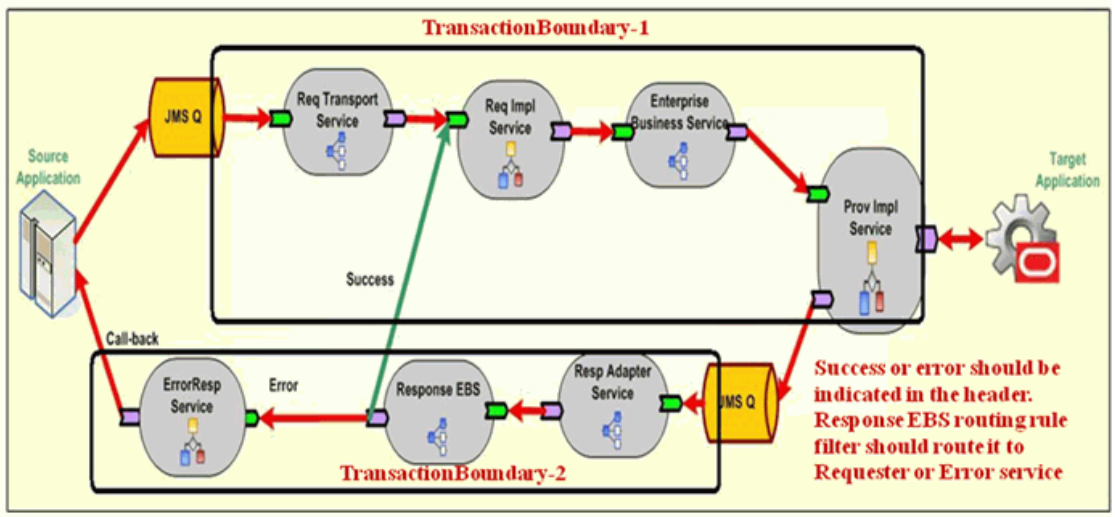
Note:

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 7-8.

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 7-8 Programming Model 1: Using a Separate Service for Error Handling



Programming Model 2: Using JMS Queue as a Milestone Between Requester ABCS and the EBS

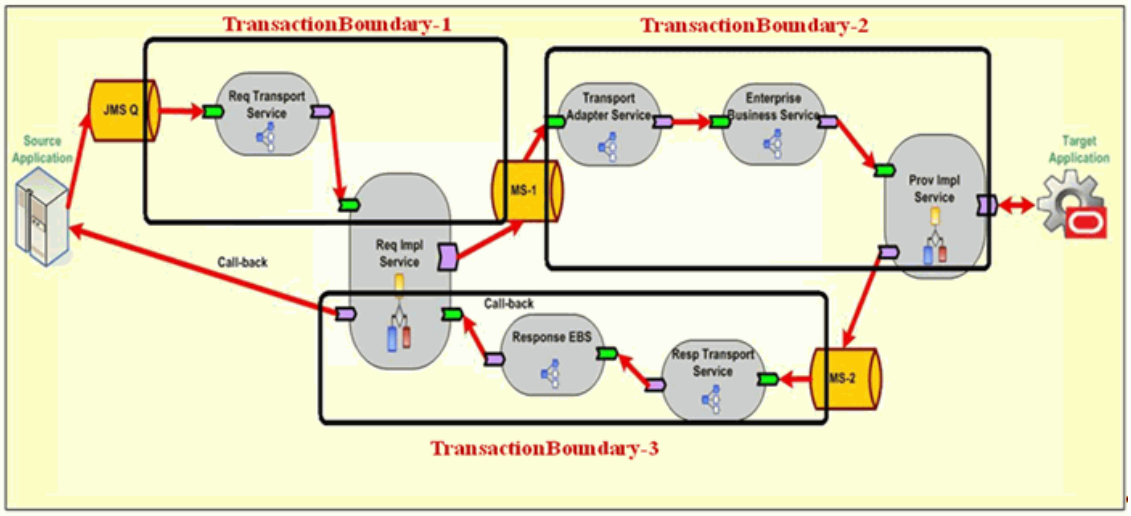
In this model, shown in Figure 7-9, the requester ABCS publishes the inbound message to a JMS milestone. The transaction starts with de-queueing of the message by the requester ABCS or the external application directly calling the requester ABCS. The transaction ends with requester ABCS publishing the 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 7-9 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.

Programming Model 3: Using a Parallel Routing Rule in the EBS

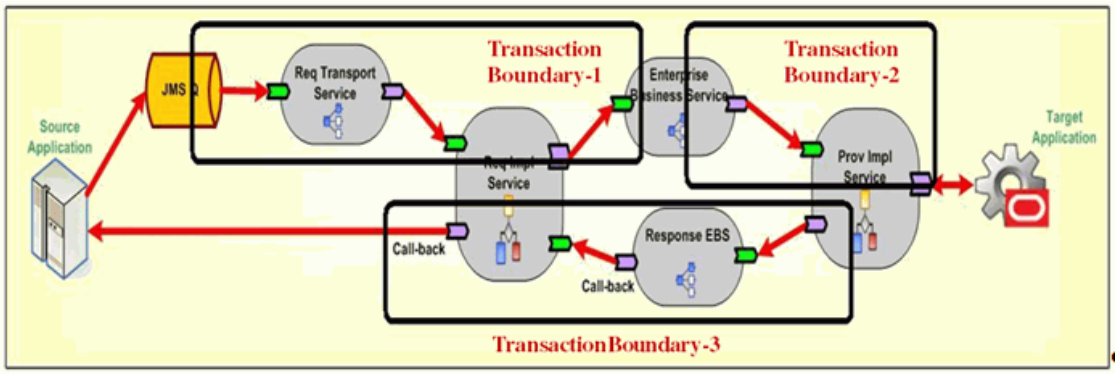
An inbound message reaching the Requester ABCS is passed to the EBS. The EBS routes the message to the Provider ABCS for the message processing, using a parallel routing rule. The transaction starts with de-queuing of the message by the requester ABCS or the external application directly calling the requester ABCS. The transaction ends when the EBS persists the inbound message for a queued execution.

Figure 7-10 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 7-10 Programming Model 3: Using a Parallel Routing Rule in the EBS



What Tasks Are Required in Provider ABCS to Implement This MEP

For details about the tasks see Implementing Provider ABCS in an Asynchronous Message Exchange Scenario.

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 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 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.

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.

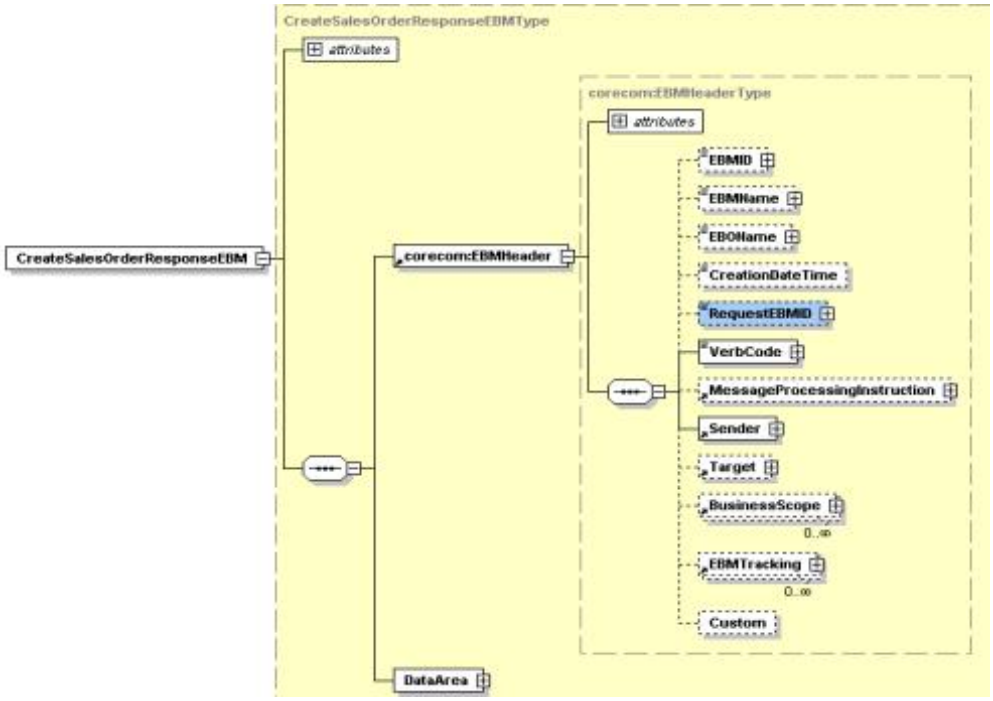
   If a particular error needs compensation service to be invoked, then the compensate operation of the EBS is invoked for routing the request to the compensation service.

   For more information, see 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 7-11.

Figure 7-11 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 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.

Using the Programming Models for the Request-Delayed Response Pattern

If you use the programming models found in 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

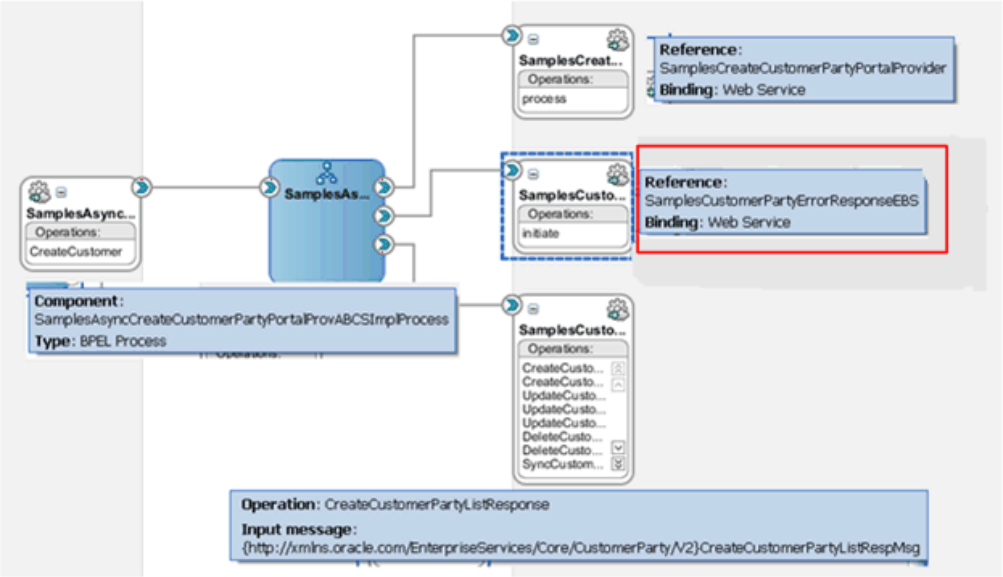
Programming Model 3: Using a Parallel Routing Rule in the EBS

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 Oracle AIA Naming Standards for AIA Development.

Figure 7-12 illustrates the asynchronous provider ABCS composite with a second reference to the ResponseEBS. The label for the second reference is *<EBSName>ErrorResponseEBS*.

Figure 7-12 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.

How to Ensure Transactions in Services

For more information about ensuring transactions, see How to Ensure Transactions in AIA Services.

How to Handle Errors in the Asynchronous Request-Delayed Response MEP

See Configuring Oracle AIA Processes for Error Handling and Trace Logging.

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 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.

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 How to Ensure Transactions in AIA Services.

How to Handle Errors in the Synchronous Request-Response MEP

See Configuring Oracle AIA Processes for Error Handling and Trace Logging.

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.

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:

- Create
- Update
- Delete

- Sync
- Validate
- Process
- 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

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.

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 7-1.

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.

Example 7-1 Update

```
<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>
```

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*.

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.

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.

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.

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 7-2.

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, as shown in Example 7-4. 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).

  In Example 7-4, 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, as shown in Example 7-5. 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.

In Example 7-5, the SalesOrder root query is sorted by OrderDateTime and Description, while the child node SalesOrderLine is sorted by price.

QueryCriteria Examples

Query with a Single QueryCriteria Element and Single Query Expression
Example 7-3 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.

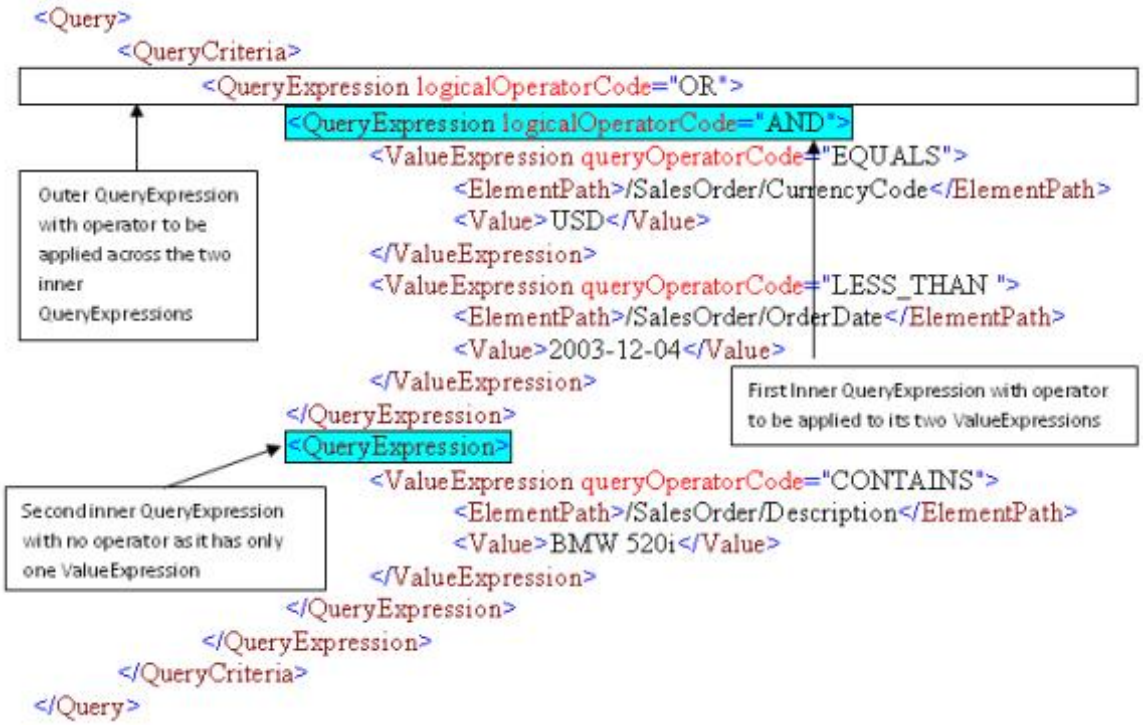
Query with a Single QueryCriteria and Nested QueryExpressions
Figure 7-13 is an example of a Query with a single QueryCriteria but with nested QueryExpressions. The query to be run is Query SalesOrders, in which SalesOrder/CurrencyCode=USD **AND** SalesOrder/OrderDateTime<2003-12-04 **OR** SalesOrder/Description **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 7-13.

Figure 7-13 Example of Query with Nested Query Expressions



Query with Multiple QueryCriteria

Example 7-6 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".

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.

Request for Single Return to QueryInvoice Message

Example 7-7 illustrates requesting only the InvoiceLine to be returned in response to a QueryInvoice message.

Request for Specific Message Return to QueryInvoice Message

Example 7-8 is an example of returning all QueryInvoice message content except for Charge and PaymentTerm.

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.

Example 7-2 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>
```

Example 7-3 Defining SortElements for Single QueryCriteria

```
<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 7-4 Defining SortElements for Multiple QueryCriteria

```
<QueryCriteria>
<QueryExpression>
         <ValueExpression>

         </ValueExpression>
   </QueryExpression>
   <SortElement sortDirection="DESC">/OrderDateTime</SortElement>
   <SortElement sortDirection="ASC">/Description</SortElement>
</QueryCriteria>
```

Example 7-5 Example of Query with Single QueryCriteria Element and Single QueryExpression

```
<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>
```

Example 7-6 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>
```

Example 7-7 Requesting a Single Return to QueryInvoice Message

```
<Query>
<QueryCriteria>

   </QueryCriteria>
         <ResponseFilter>
               <ChildComponentPath>InvoiceLine</ChildComponentPath>
         </ResponseFilter>
</Query>
```

Example 7-8 Requesting Specific Message Return to QueryInvoice Message

```
<Query>
<QueryCriteria>

   </QueryCriteria>
   <ResponseFilter>
         <ExclusionIndicator>true</ExclusionIndicator>
         <ElementPath>Charge</ElementPath>
         <ElementPath>PaymentTerm</ElementPath>
   </ResponseFilter>
</Query>
```

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:

- How to Invoke an ABCS Directly from an Application
- How to Invoke an ABCS Using Transport Adapters
- When Does an Enterprise Business Service Invoke an ABCS

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.

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 Interfacing with Transport Adapters.

The details on establishing the inbound connectivity between adapters and the ABCS are given in Establishing Resource Connectivity.

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.

8

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:

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.

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 Configuration Parameters.

Figure 8-1 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 8-1 Extending the Request-Response Interaction Style

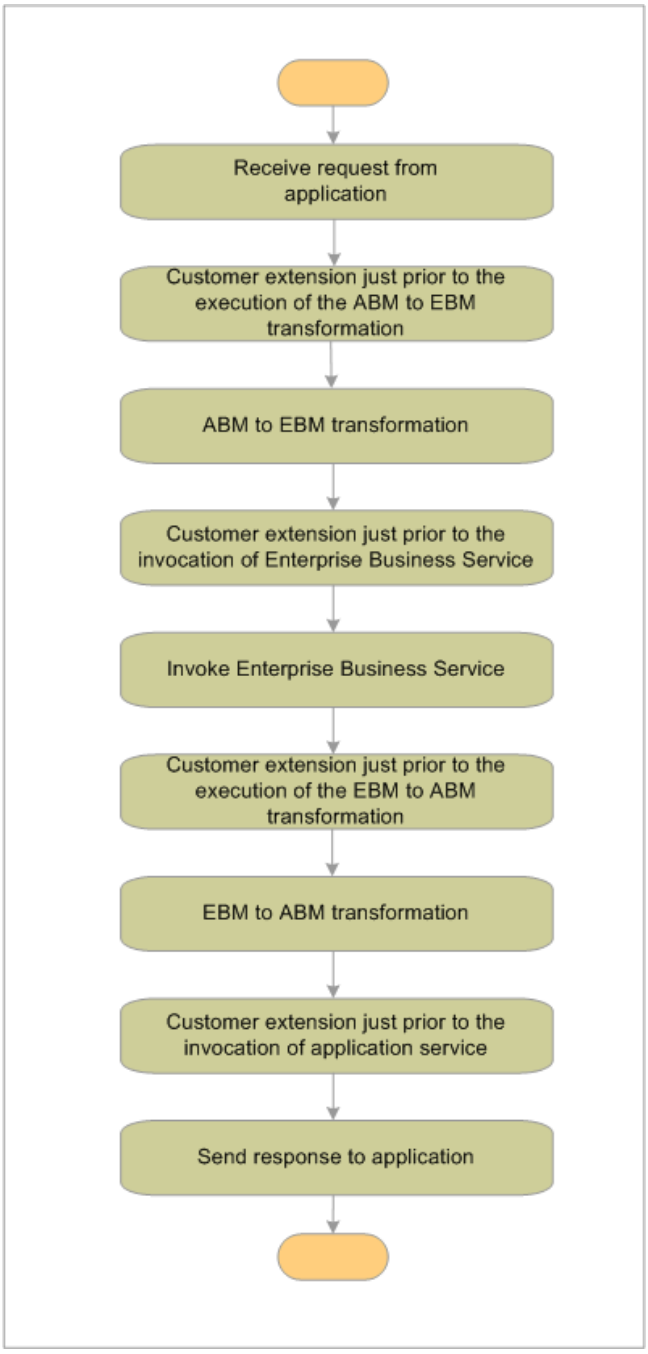
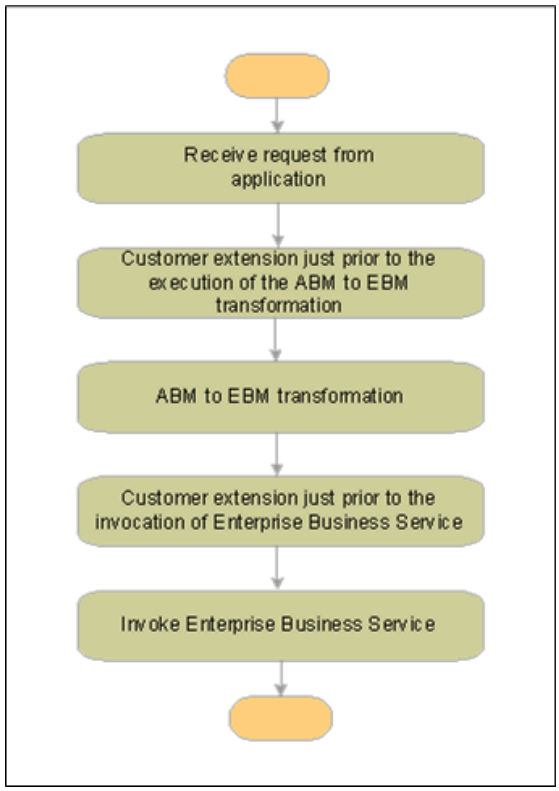


Figure 8-2 shows the high-level flow of activities in a requester-specific ABCS. The diagram assumes that the EBS with which it is interacting employs a fire-and-forget interaction style. The steps for executing the customer extension to do additional tasks are optional.

Figure 8-2 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

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 Configuration Parameters.

Figure 8-3 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 8-3 Provider-Specific ABCS Using Request-Response Interaction Style

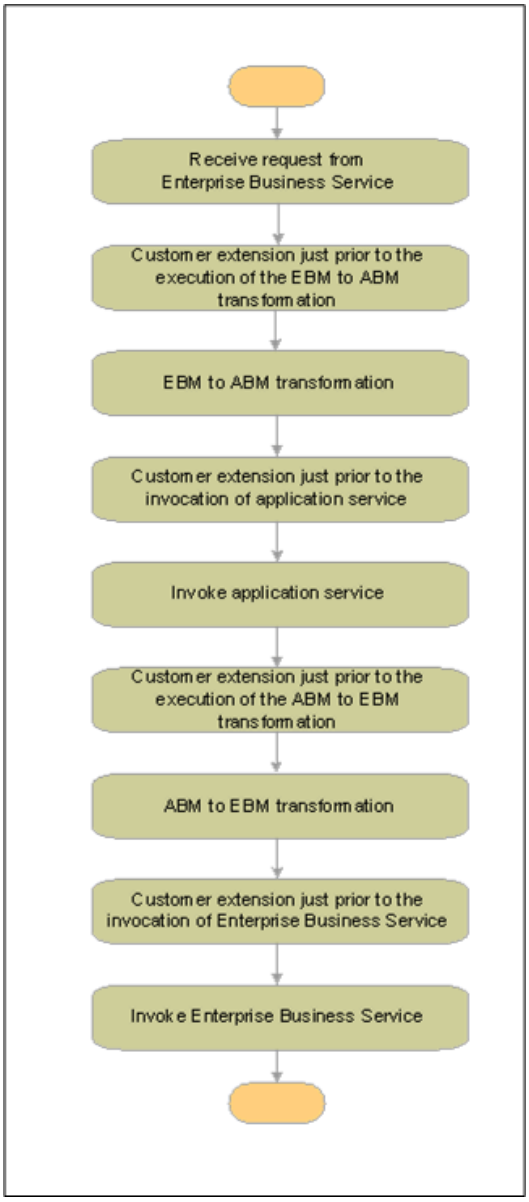
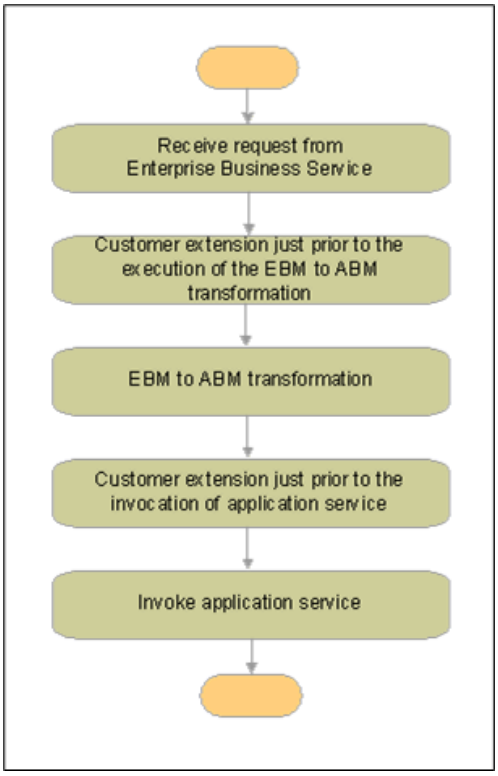


Figure 8-4 depicts the high-level flow of activities in a provider-specific ABCS. The diagram assumes that the EBS with which it is interacting employs a fire-and-forget interaction style.

Figure 8-4 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.

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 8-1 lists the service operations for the requester ABCS-specific extensibility points:

Table 8-1 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 8-2 lists the service operations for the provider ABCS-specific extensibility points:

Table 8-2 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 runtime 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 How to Ensure Transactions in AIA Services.

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.

Designing an ABCS Composite with Extension

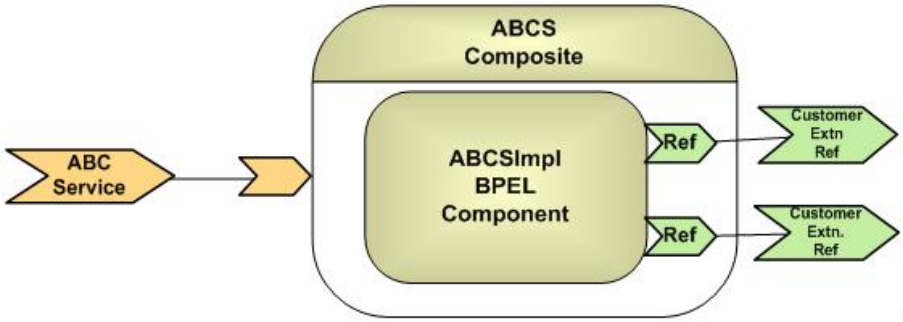
When an ABCS composite is developed as an extension-enabled service:

- 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.

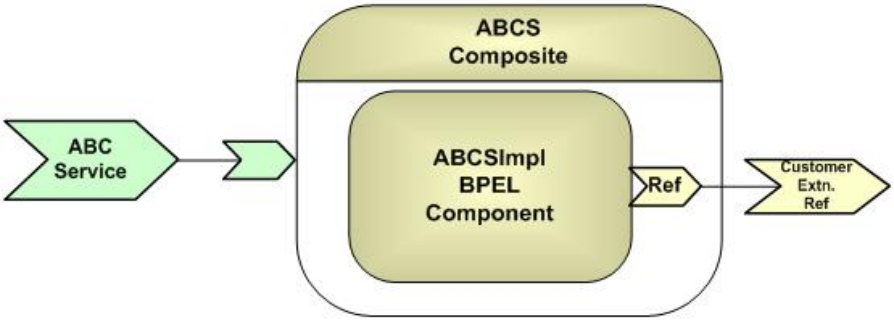
When different external extension services are referred to by the ABCS composite, multiple hooks exist from the ABCS composite to the External Service. That is, when an ABCS invokes two different services at two of its extension points, the composite must be designed to have two distinct external references, as shown in Figure 8-5:

Figure 8-5 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 8-6:

Figure 8-6 Example of Composite with One External Reference



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.

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 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 8-1.

   This code example was reproduced from the composite.xml, where the extension service is defined using an abstract WSDL:

Example 8-1 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>
```

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 SOA Core Extension.

You should push the concrete WSDL into the MDS repository to the folder 'ExtensionServiceLibrary'.

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 runtime.

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 Using MDS in AIA.

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.

How to Set Up the Extension Point Pre-ProcessABM

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

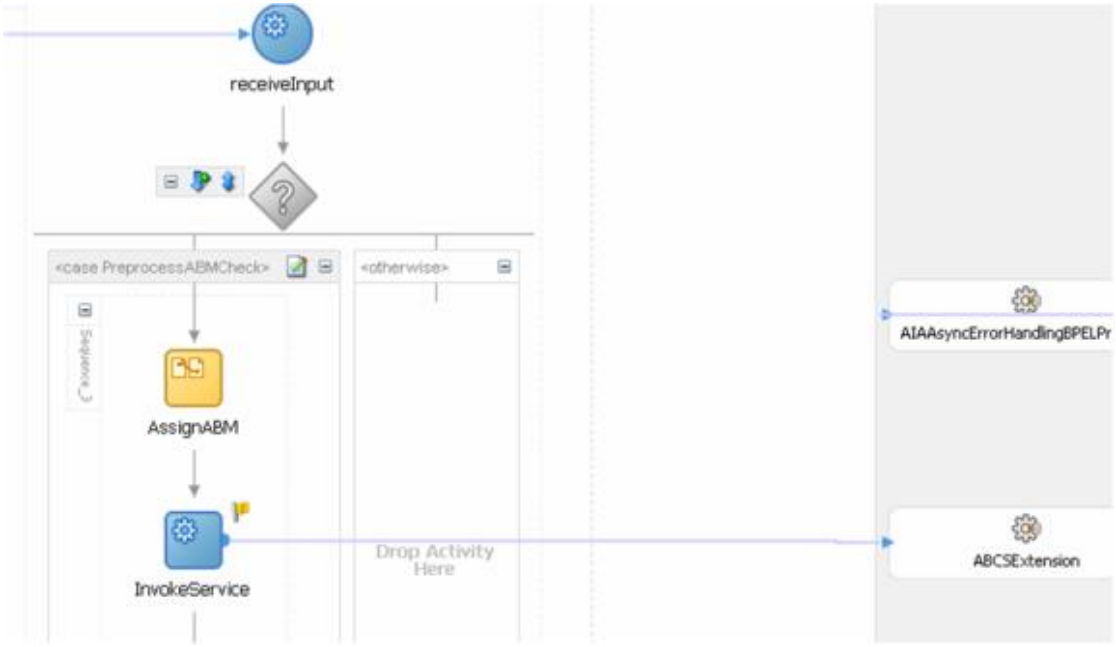
     The Invoke activity makes the service-call on the partner-link. The programming model for the ABCS extension points sets up a contract on the service to return the same payload that it receives.

   - 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 8-7 illustrates the sequence:

Figure 8-7 Setting up the Extension Point Pre-ProcessABM



How to Set Up the Extension Point Pre-ProcessEBM

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

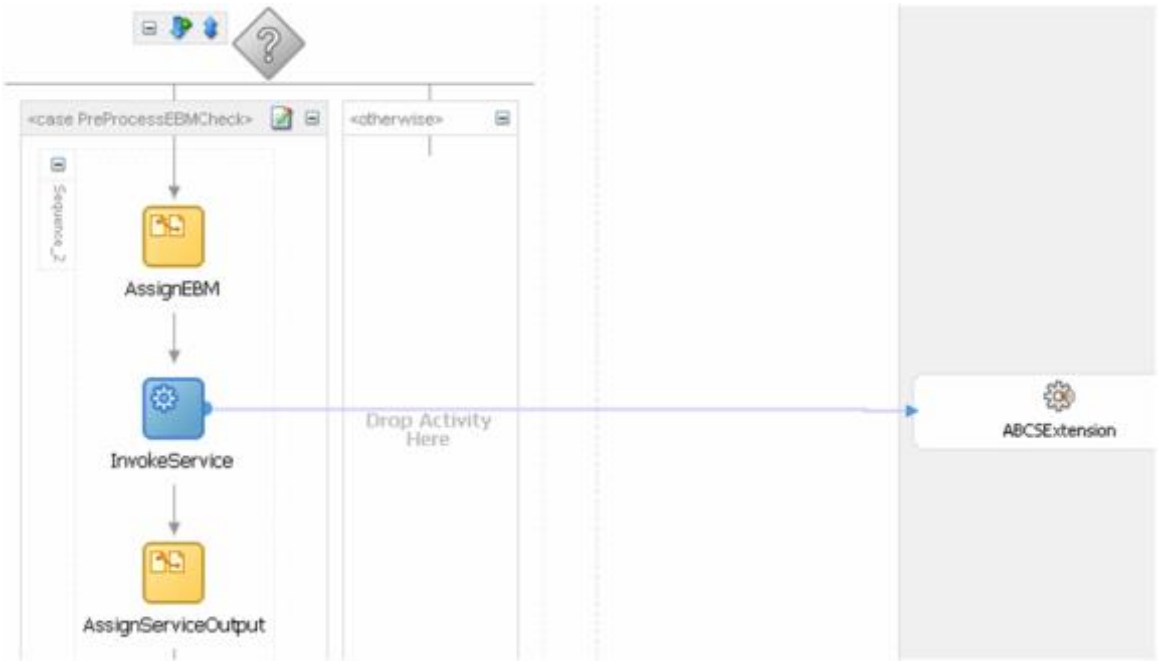
     The Invoke activity makes the service-call on the partner-link. The programming model for the ABCS extension points sets up a contract on the service to return the same payload that it receives.

   - Assign

     The Assign activity assigns the value of the response payload from the service invocation to variable of the process activity that follows.

   Figure 8-8 illustrates the sequence:

Figure 8-8 Setting up the Extension Point Pre-process EBM



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 8-2) 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.

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

Example 8-2 Servlet to Test Extension

```
<service name="CreateCustomerSiebelReqABCSImplExt">
<port name="CreateCustomerSiebelReqABCSImplExt"
binding="tns:CreateCustomerSiebelReqABCSImplExt_Binding">
              <soap:address
location="http://{host}:{port}/MirrorServlet/mirror"/>
          </port>
    </service>
```

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.

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 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 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 Configuring Oracle AIA Processes for Error Handling and Trace Logging.

Working with Adapters

This section discusses working with transport and version adapters.

This section includes the following topics:

- Interfacing with Transport Adapters
- How to Develop Transport Adapters
- When to Put Adapters in a Single Composite
- Planning Version Adapters
- How to Configure a Version Adapter

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 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 Working with AIA Design Patterns.

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 *Administering JMS Resources for Oracle WebLogic Server*.

How to Develop Portal DB Adapter

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.

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.

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.

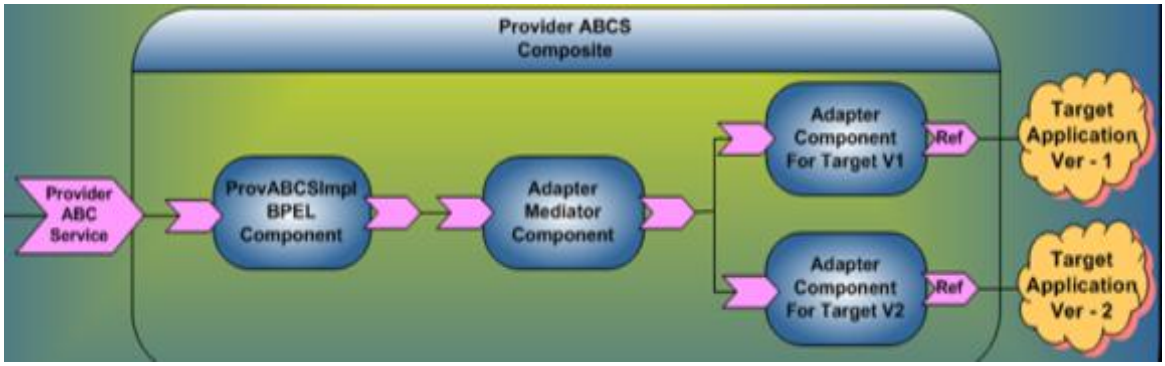
Examples:

- 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 8-9:

Figure 8-9 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.

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.

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:

- How to CAVS Enable Provider ABCS
- How to CAVS Enable the Requester ABCS
- Introduction to the CAVSEndpointURL Value Designation
- 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 "Introduction to the Composite Application Validation System" in *Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

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 8-3:

   The CAVSEndPointURL value is set at design time.

   For more information about the design-time designation of the CAVSEndPointURL, see Introduction to the CAVSEndpointURL Value Designation.

2. Ensure that you have the following namespace prefixes defined in your BPEL process as shown in Example 8-4:

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 8-5 as the first step in the BPEL process. Be sure to replace the tokens appropriately:

6. Add the following <scope> as shown in Example 8-6 **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 8-3 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>
```

Example 8-4 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
```

Example 8-5 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>
```

Example 8-6 Adding 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>
```

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 8-7.

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.

Example 8-7 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>
```

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

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.

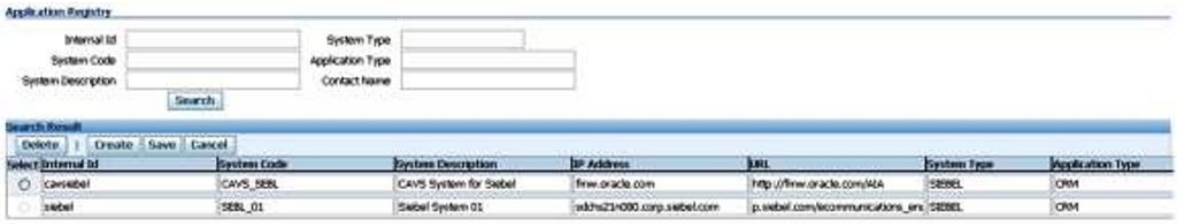
This purge is accomplished as follows:

1. Pre-Built Integrations that are delivered to work with Oracle SOA Core Extension 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 8-10 illustrates the system and CAVS-related entries.

Figure 8-10 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.

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.

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 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.

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.

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.

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.

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).

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.

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 8-8:

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 8-9 shows how transaction properties should be set on the BPEL adapter interface:

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.

Example 8-8 Populating the wsa:EndpointReference Element

```
<component name="SamplesCreateCustomerSiebelReqABCSImplProcess">
    <implementation.bpel
src="SamplesCreateCustomerSiebelReqABCSImplProcess.bpel"/>
    <property name="bpel.config.transaction">required</property>
</component>
```

Example 8-9 Populating the PEL Endpoint Element

```
<component name="SamplesCreateCustomerSiebelJMSConsumerProcess">
     <implementation.bpel
src="SamplesCreateCustomerSiebelJMSConsumerProcess.bpel"/>
        <property name="bpel.config.transaction">required</property>
</component>
```

How to Transaction-Enable AIA Services

The following sections provide details on how to transaction enable AIA services in different scenarios:

- Synchronous Request-Response Scenarios
- AIA Services in Asynchronous MEP

- Asynchronous Operation from an ABCS in the Same Thread but in a Different Transaction

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.

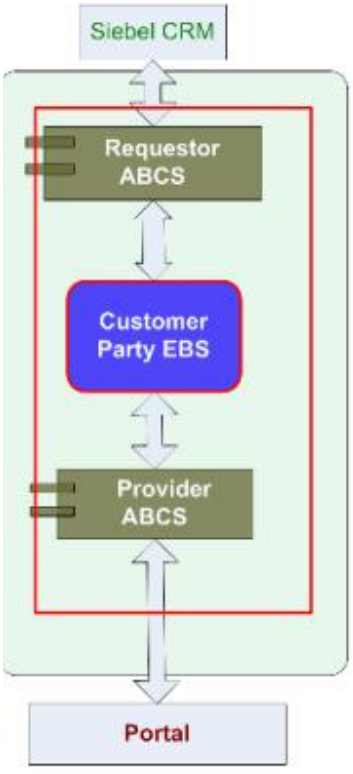
Hence, the transaction settings in the ABCS services should be as shown in Example 8-10:

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 8-11 illustrates the transaction boundary.

Figure 8-11 Transaction Boundary



Example 8-10 Transaction Settings for Asynchronous MEP

```
<component name="SamplesCreateCustomerSiebelJMSConsumerProcess">
     <implementation.bpel
src="SamplesCreateCustomerSiebelReqABCSImplProcess.bpel"/>
        <property name="bpel.config.transaction">required</property>
</component>
```

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

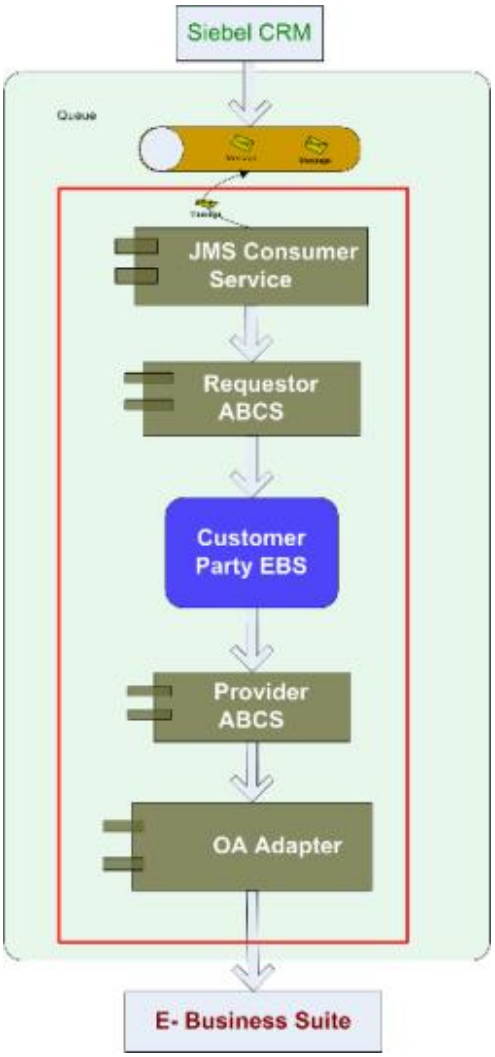
Therefore, the transaction settings in the ABCS services should be as shown in Example 8-11:

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 8-12 shows how transaction properties are set on the adapter component:

Figure 8-12 illustrates the transaction boundary.

Figure 8-12 Transaction Boundary in Asynchronous MEP



Example 8-11 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>
```

Example 8-12 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>
```

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.

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.

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 Oracle AIA Naming Standards for AIA Development.

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.

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.

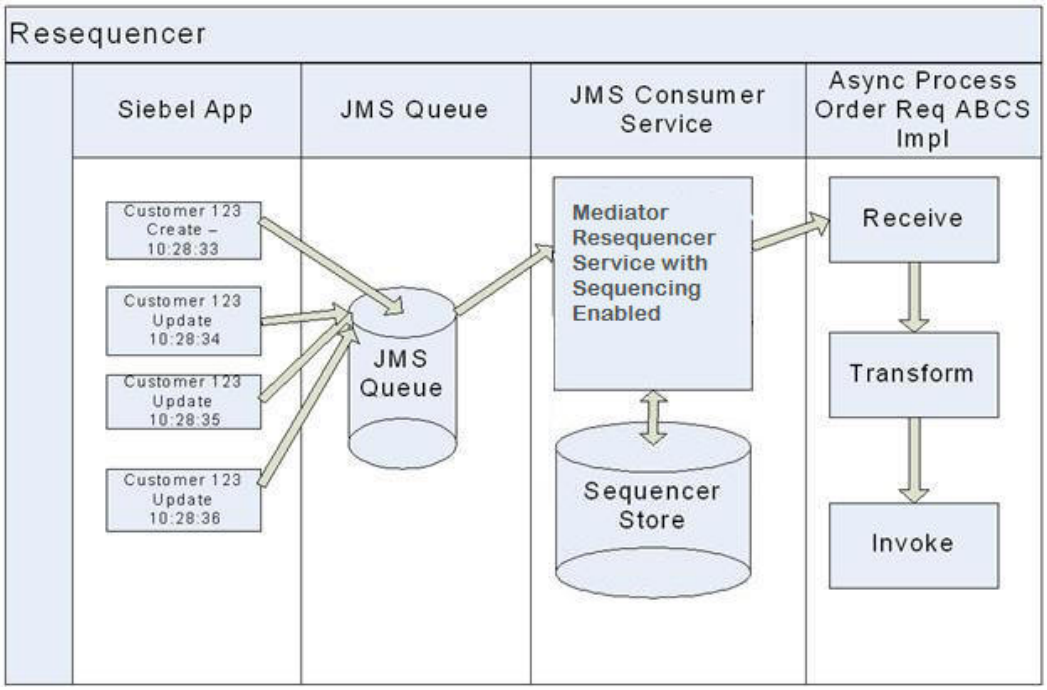
A Resequencer is used to rearrange a stream of related but out-of-sequence messages back into order. It sequences the incoming messages that arrive in a random order and then send them to the target services in an orderly manner.

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 8-13 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 8-13 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

For more information about the Oracle Mediator Resequencer feature, see "Resequencing in Oracle Mediator" in *Developing SOA Applications with Oracle SOA Suite.*

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 8-14 is the screen shot of a composite mplan where the Resequence level is selected at operation level.

Figure 8-14 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 groupID. 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.

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 8-15 is the screenshot that shows Resequence types.

Figure 8-15 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.

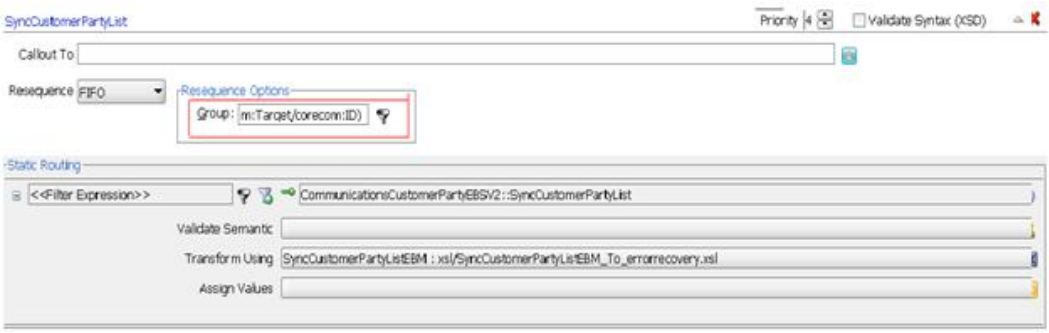
      For example, CustomerId, OrderId, ProductId, Account ID and so forth.

      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 8-16 is the screenshot that shows XPath expression in the **Group** field.

Figure 8-16 Resequence Options



b. Determine and set the ID

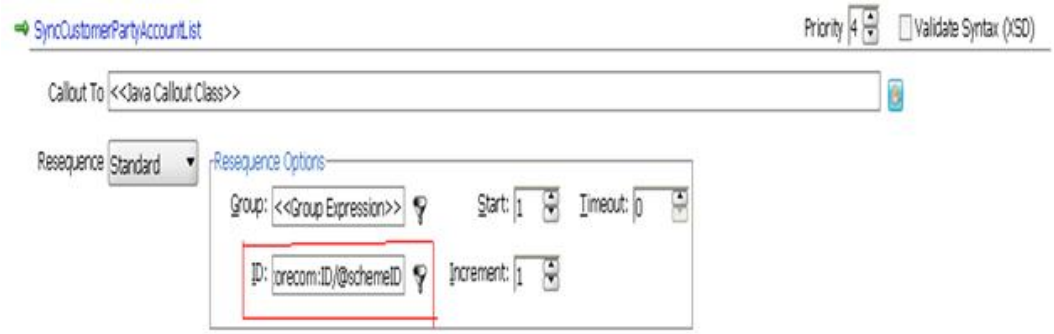
The ID is a parameter (field or attribute) on the incoming message that indicates the position of the message in the sequence.

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 8-17 is the screenshot that shows XPath expression in the **ID** field.

Figure 8-17 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 8-18 is the screenshot that shows the option dateTime selected.

Figure 8-18 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.

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.

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.

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 Populating Message Resubmission Values.

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 "Resequencer Based Resubmission" in *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.

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:

You cannot turn off the resequencer at runtime after you turn it on for a composite.

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 *Developing SOA Applications with 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.

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.

Customizing the Customer Version

If you want to customize customer version of the SOA composite application, see "Customizing the Customer Version" in *Developing SOA Applications with Oracle SOA Suite.*

Applying patches after customization

To apply patches after you have customized, see "Upgrading the Composite" in *Developing SOA Applications with Oracle SOA Suite.* Instructions here help you to apply layers to the new base composite.

9

Designing and Constructing Enterprise Business Flows

This chapter provides an overview of Enterprise Business Flows (EBF) describes how to define and implement EBFs.

Note:

Composite Business Processes (CBP) will be deprecated from next release. Oracle advises you to use BPM for modeling human/system interactions.

This chapter includes the following sections:

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 9-1 and Figure 9-2 illustrate how some EBFs in the Order to Cash Process Integration Pack Pre-Built Integration are implemented to leverage existing capabilities.

Figure 9-1 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 9-1 Example of EBF Orchestrating a Flow from Source to Target

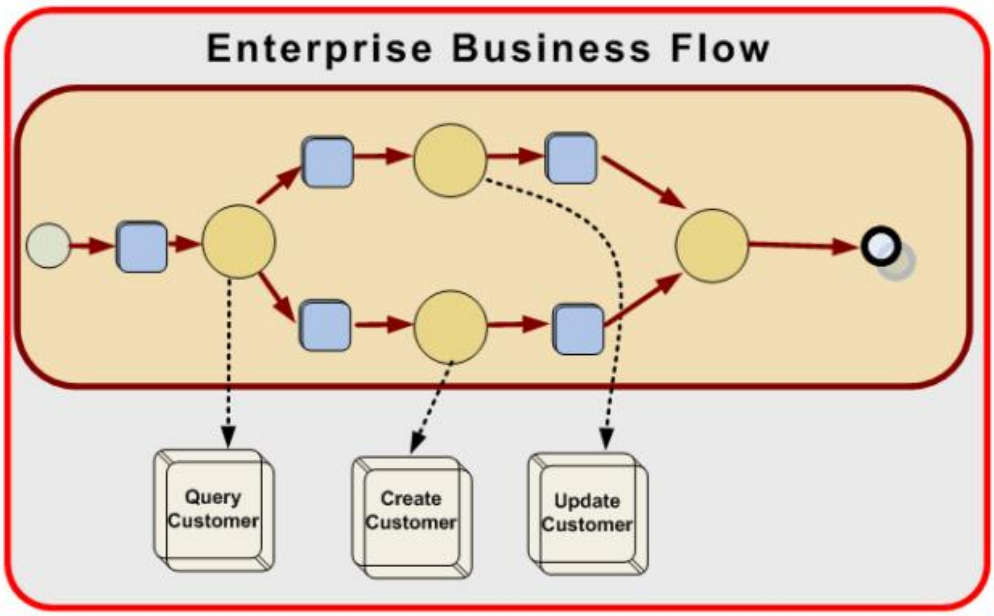
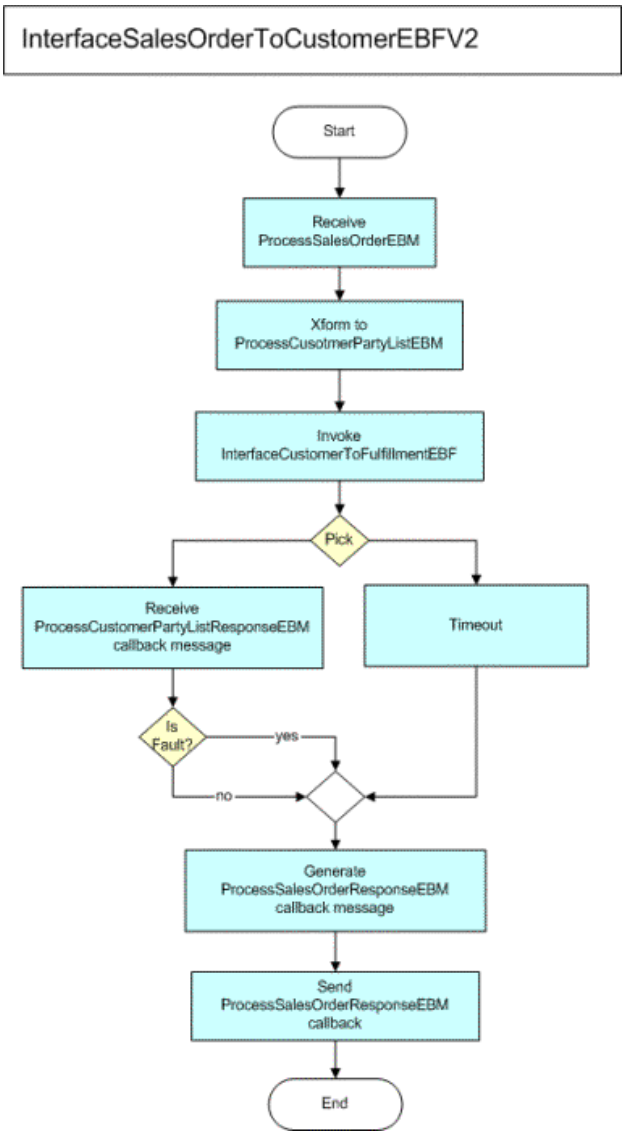


Figure 9-2 demonstrates a flow that is receiving the sales order and orchestrating across customer creation, fulfillment, and update back in the source application with response.

Figure 9-2 Example of Sales Order Flow



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).

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.

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. See Example 9-1.

   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.

Example 9-1 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>
```

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.

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.

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.

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 "Getting Started with Oracle BPEL Process Manager" in *Developing SOA Applications with Oracle SOA Suite.*

   Follow 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 Configuring Oracle AIA Processes for Error Handling and Trace Logging.

4. Enable extensibility points in the EBF.

10

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 SOA Core Extension infrastructure for B2B.

This chapter includes the following sections:

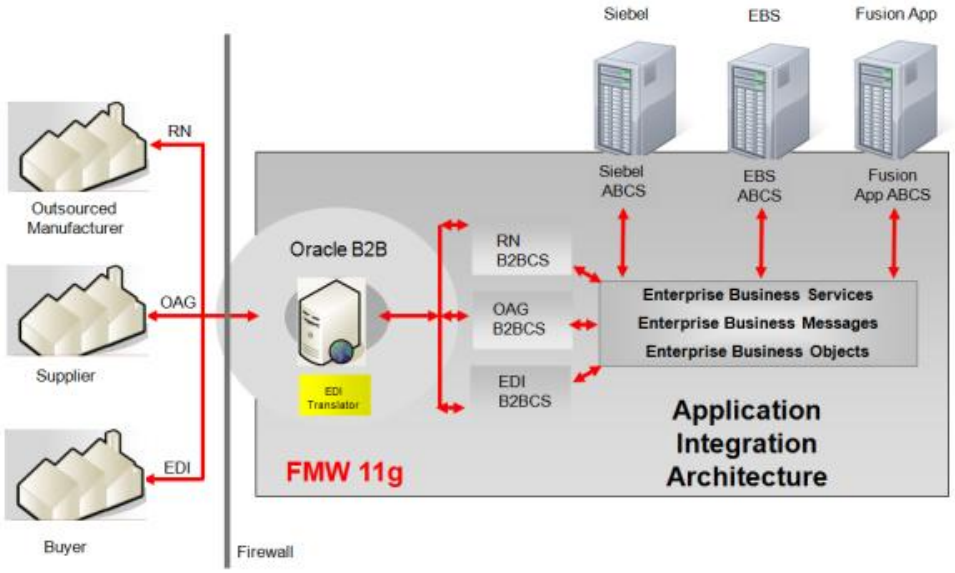
Overview of B2B Integration Using AIA

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 10-1 illustrates how the canonical-based integration architecture of AIA can be applied to meet the B2B integration needs of large enterprises.

Figure 10-1 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 SOA Core Extension, 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.

Understanding B2B Document Flows

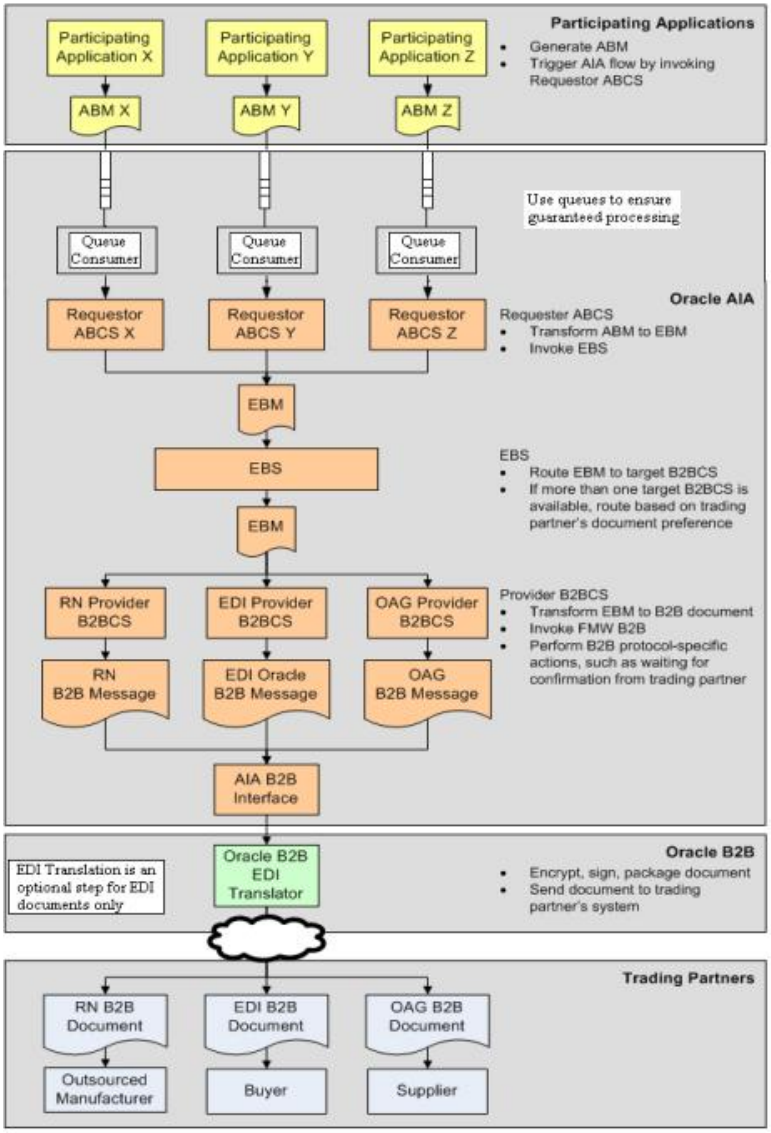
This section includes the following topics:

- Describing Outbound B2B Document Flows Built Using AIA
- Describing Inbound B2B Document Flows Built Using AIA

Describing Outbound B2B Document Flows Built Using AIA

Figure 10-2 illustrates a sample outbound B2B flow implemented using AIA:

Figure 10-2 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.

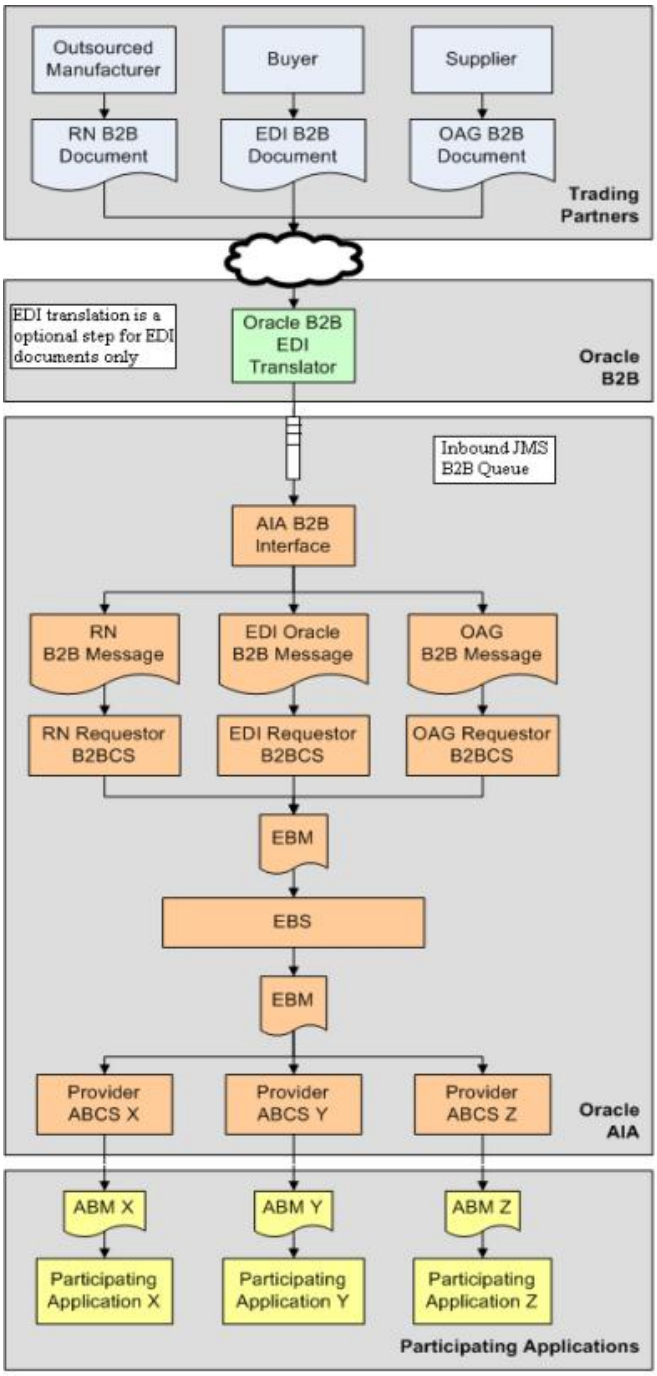
3. The EBS introspects the EBM and routes it to the correct provider B2BCS based on the trading partner's B2B document preference.

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.

Describing Inbound B2B Document Flows Built Using AIA

Figure 10-3 illustrates a sample inbound B2B flow implemented using AIA:

Figure 10-3 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.

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: How AIA Complements Oracle Fusion Middleware Oracle B2B.

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 *User's Guide for Oracle B2B*.

Understanding the SOA Core Extension Infrastructure for B2B

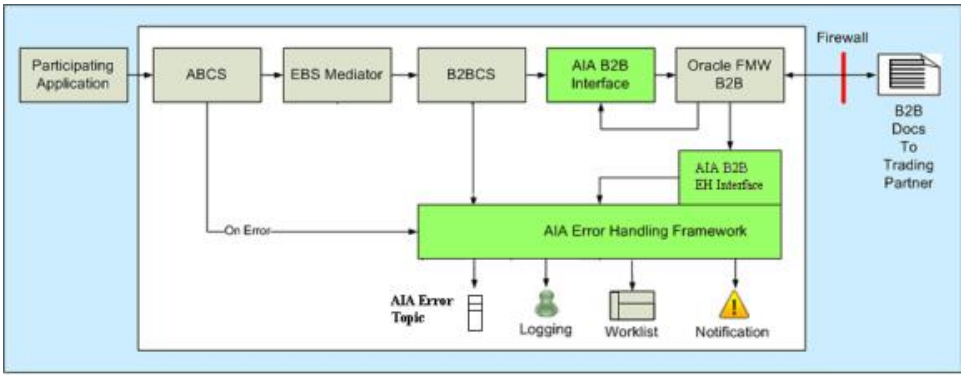
This section includes the following topics:

- B2B Support in AIA Error Handling Framework
- AIA B2B Interface

Along with the architecture and programming model, additional infrastructure components are delivered with SOA Core Extension in support of B2B integrations.

This section provides a high-level description of the B2B infrastructure delivered with SOA Core Extension. Developing and Implementing Outbound B2B Integration Flows and Developing and Implementing Inbound B2B Integration Flows provide more details on how to use the infrastructure components shown in Figure 10-4 when you develop and implement B2B flows using AIA.

Figure 10-4 SOA Core Extension B2B Infrastructure Components



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 SOA Core Extension.*

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 *User's Guide for Oracle B2B.*

11

Developing and Implementing Outbound B2B Integration Flows

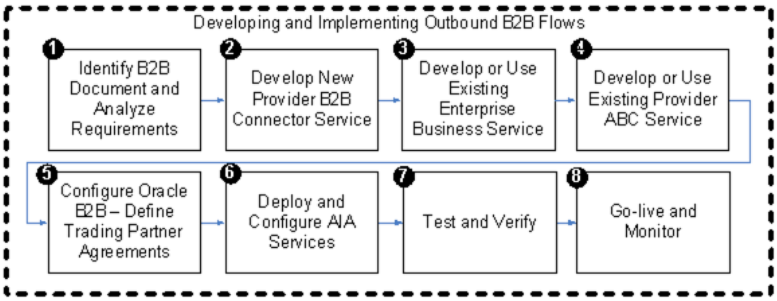
This chapter provides an overview of developing and implementing outbound B2B integration flows and describes how to identify the B2B document and analyze requirements, develop new provider B2B connector service, develop or extend an existing Enterprise Business Service, develop or extend an existing 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:

Introduction to Developing and Implementing Outbound B2B Integration Flows

Figure 11-1 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 11-1 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.

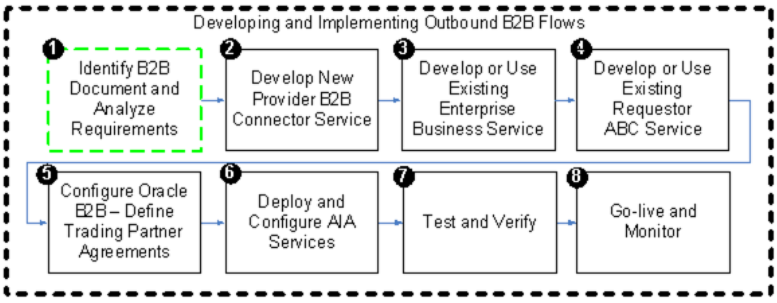
Step 1: Identifying the B2B Document and Analyzing Requirements

This section includes the following topics:

- How to Identify the B2B Document Protocol
- How to Identify the B2B Document Type and Definition
- How to Define the Document in Oracle B2B
- How to Define the Document in AIA
- How to Identify the EBO_ EBS_ and EBM to Be Used
- How to Design Mappings for the B2B Document
- 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 11-2, you must also analyze the requirements for the AIA services that must be built or extended to support the flow.

Figure 11-2 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.

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.

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.

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.*

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.

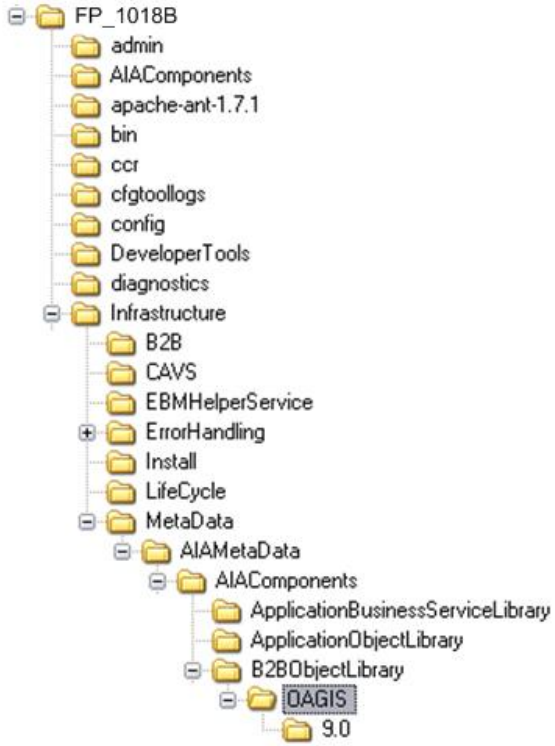
To deploy the B2B documents to the metadata store:

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 11-3, create directory *OAGIS* to which to copy OAG business object definitions.

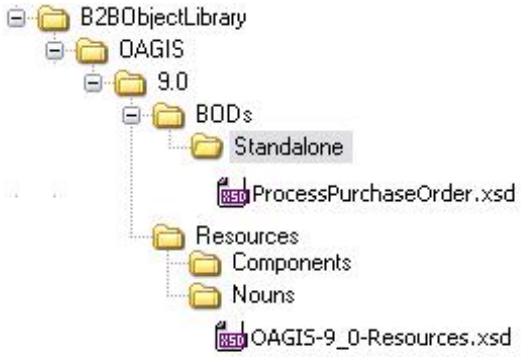
Figure 11-3 OAGIS Directory Created to Store OAG Business Object Definitions



3. Copy the XML schema files that correspond to the B2B document type under this folder.

   For example, as shown in Figure 11-4, copy the OAGIS ProcessPurchaseOrder.xsd file and all of its supporting schemas to this folder location, while also preserving the relative directory structure.

Figure 11-4 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 11-5, to specify that the directory B2BObjectLibrary has to be updated in Oracle Metadata Services (MDS).

   Figure 11-5 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.

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

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:

- ABCSExtension.PreInvokeABS
- ABCSExtension.PostInvokeABS
- ABCSExtension.PostXformABMtoEBM

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.

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.

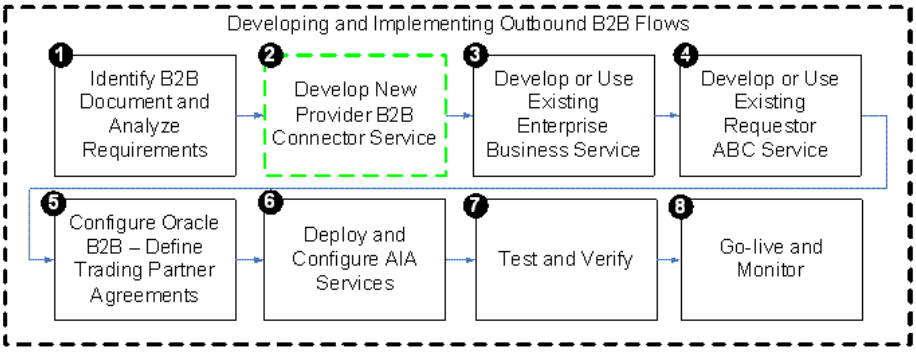
Step 2: Developing a New Provider B2B Connector Service

This section includes the following topics:

- Introduction to a Provider B2B Connector Service
- How to Identify the Message Exchange Pattern
- How to Develop a B2BCS Service Contract
- How to Store a WSDL in the Oracle Metadata Repository
- How to Develop a B2B Connector Service
- How to Customize the AIA B2B Interface
- How to Annotate B2B Connector Services
- How to Support Trading Partner-Specific Variants
- How to Enable Error Handling

The next step, as shown in Figure 11-6, is to develop the provider B2BCS to support the outbound B2B integration flow.

Figure 11-6 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.

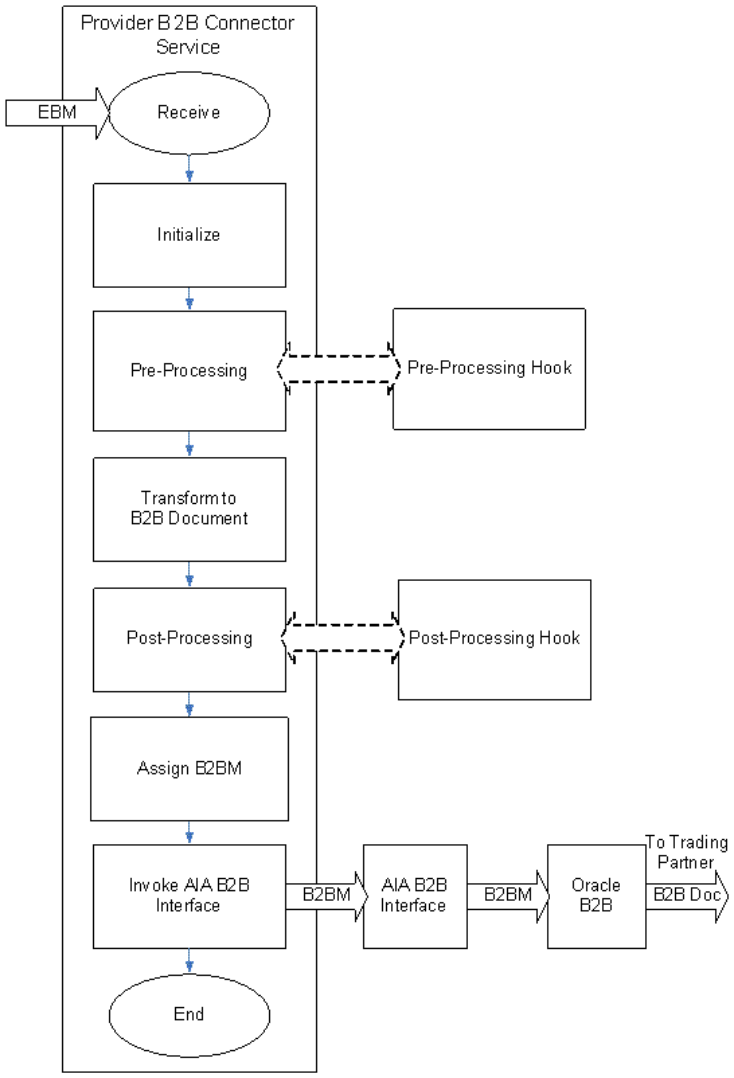
Introduction to a Provider B2B Connector Service

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 11-7 illustrates the processing that takes place in a simple fire-and-forget message exchange pattern-based provider B2BCS.

Figure 11-7 Processing Flow for a Fire-and-Forget Provider B2BCS



Step-by-step instructions for developing B2BCSs are provided in the following sections.

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 Acknowledgment B2B document.

In this use case, even if a response is expected from the supplier in the form of the Purchase Order Acknowledgment 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 Acknowledgment document to the original purchase order because the Vendor is expected to provide the same purchase order identification used in the request in the Acknowledgment 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 Identifying the MEP.

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 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 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

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

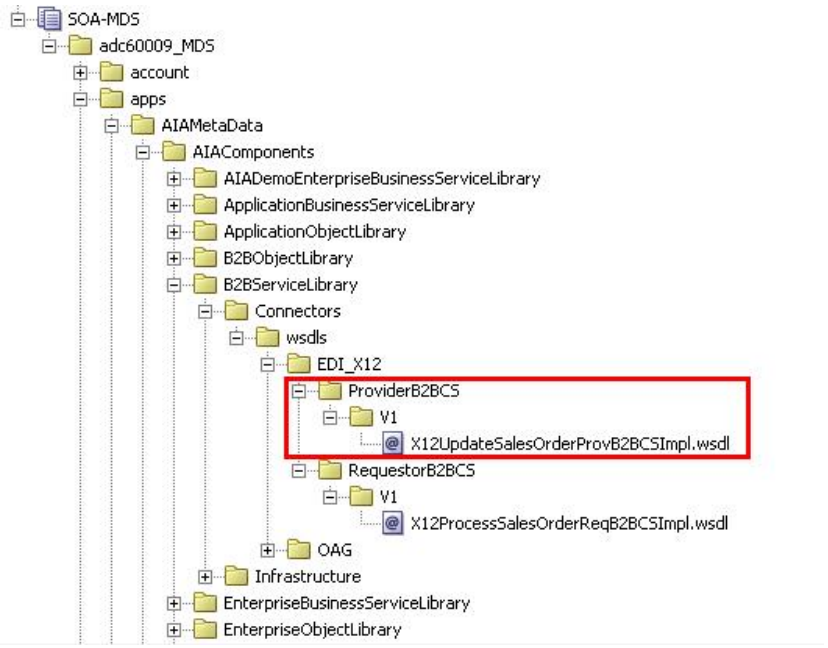
To store a WSDL in MDS:

1. Copy the handcrafted WSDL to the following location under AIA_HOME: $AIA_HOME/aia_instances/$INSTANCE_NAME/AIAMetaData/AIAComponents/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 11-8.

Figure 11-8 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

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 11-1.

Table 11-1 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

Element	Value	Example
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 11-2.

Table 11-2 <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 11-3.

Table 11-3 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 SOA Core Extension.

13. Test the service by using sample EBM payloads as input.

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.

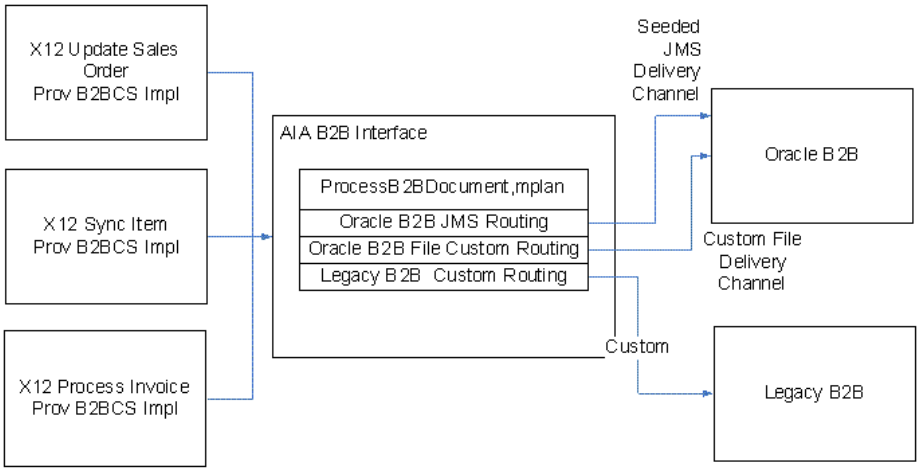
If the choice of internal delivery channel to be used for communication between B2B and AIA/SOA has to be changed at a later stage in the implementation, the B2BCS code need not be modified.

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.

This figure illustrates the use of the AIA B2B Interface to support custom internal delivery channels (file) and legacy B2B connectivity.

Figure 11-9 AIA B2B Interface Support for Custom Internal Delivery Channels and Legacy B2B Connectivity



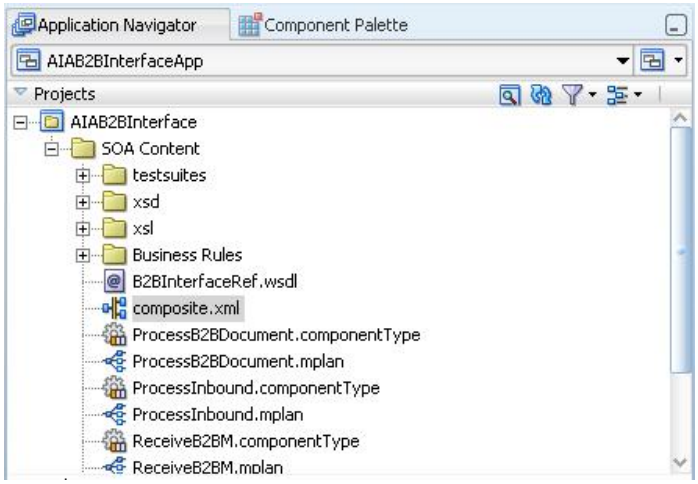
For example, you might have outbound Order documents being routed to a Fusion Middleware Oracle B2B component, and outbound Invoice documents being routed to an older B2B instance. To support this use case, define two routing rules: one for JMS invocation of Oracle B2B and another for Oracle Advanced 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:

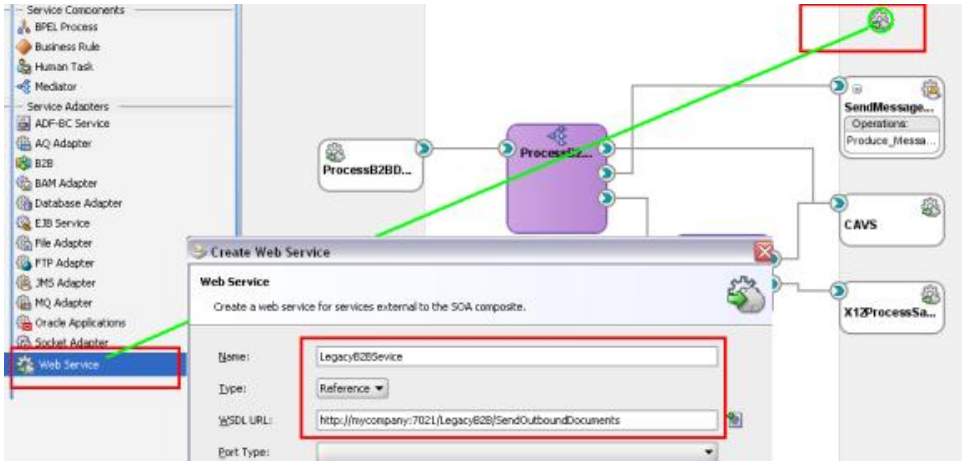
1. Open the SOA application, $AIA_HOME/Infrastructure/B2B/src/AIAB2BInterfaceApp, using Oracle JDeveloper.
2. Edit the AIAB2BInterface/composite.xml file using the Composite Editor, as shown in Figure 2.

Figure 11-10 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 3.

Figure 11-11 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 4 and 5.

Figure 11-12 AIAB2BInterface/ProcessB2BDocument.mplan in the Composite Editor

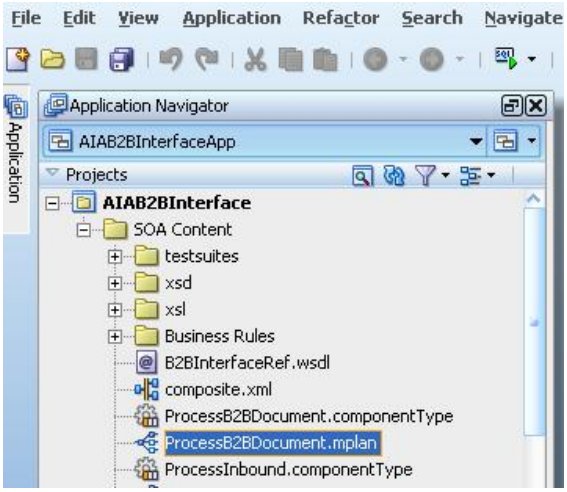
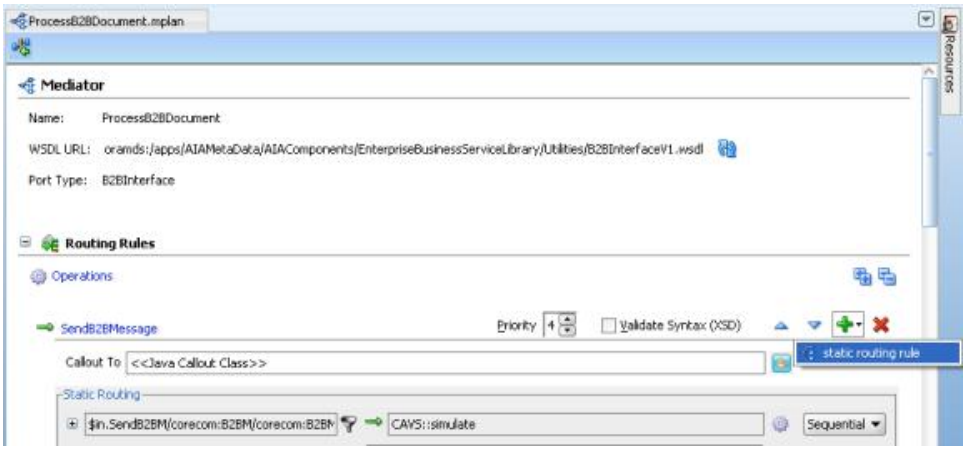


Figure 11-13 Addition of Custom Routing Rules to AIAB2BInterface/ ProcessB2BDocument.mplan

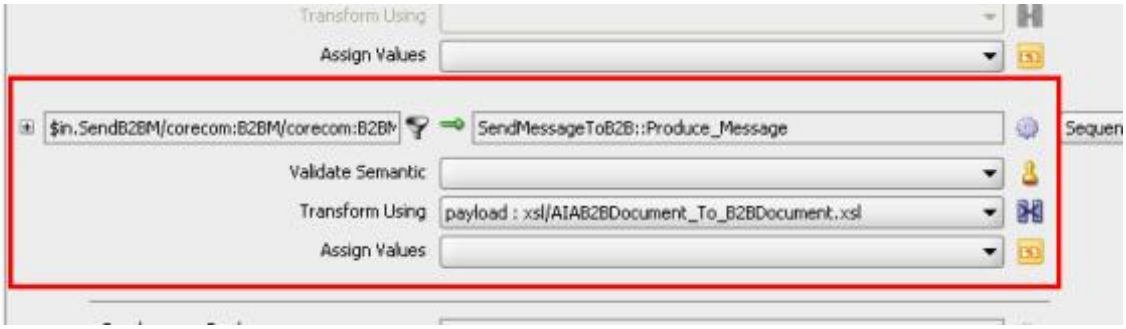


5. Use a condition expression, as shown in example code below on the GatewayID element to identify which B2B documents must be routed to the third-party B2B software.

```
<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 below. Support for Oracle B2B is prebuilt and does not require any modifications.

Figure 11-14 Transformation Step to Provide Input to B2B Software



7. The input B2BM variable is received from the provider B2BCSs. This table provides the information from the B2BM variable that can be used to provide input to the B2B software.

Table 11-4 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.

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.

To annotate B2B Connector Services:

1. Table 11-5 lists the annotation elements that must be added to the service element composite.xml, as described subsequently.

Table 11-5 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 11-15, is an example of an annotated B2BCS called X12ProcessSalesOrderReqB2BCSImpl composite.xml.

Figure 11-15 Example Snippet of an Annotated B2BCS Called X12ProcessSalesOrderReqB2BCSImpl composite.xml

```
<interface.wsdl interface="http://xmlns.oracle.com/B2BCSImpl/Core/X12UpdateSalesOrderProvB2BCSImpl/V1#wsdl.
<binding.ws port="http://xmlns.oracle.com/B2BCSImpl/Core/X12UpdateSalesOrderProvB2BCSImpl/V1#wsdl.endpoint(
<!--
            <svcdoc:AIA>
                <svcdoc:Service>
                  <svcdoc:InterfaceDetails>
                     <svcdoc:ServiceName>SalesOrderEBS</svcdoc:ServiceName>
                     <svcdoc:Namespace>http://xmlns.oracle.com/EnterpriseServices/Core/SalesOrder/V2</svcdo
                     <svcdoc:ArtifactType>EnterpriseBusinessService</svcdoc:ArtifactType>
                     <svcdoc:ServiceOperation>
                        <svcdoc:Name>UpdateSalesOrder</svcdoc:Name>
                     </svcdoc:ServiceOperation>
                  </svcdoc:InterfaceDetails>
                  <svcdoc:ImplementationDetails>
                     <svcdoc:ApplicationName></svcdoc:ApplicationName>
                     <svcdoc:BaseVersion></svcdoc:BaseVersion>
                     <svcdoc:DevelopedBy>Oracle</svcdoc:DevelopedBy>
                     <svcdoc:OracleCertified>Yes</svcdoc:OracleCertified>
                     <svcdoc:ArtifactType>ProviderB2BCSImplementation</svcdoc:ArtifactType>
                     <svcdoc:ServiceOperation>
                        <svcdoc:Name>UpdateSalesOrder</svcdoc:Name>
                     </svcdoc:ServiceOperation>
                     <svcdoc:B2BDocument>855</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. Table 11-6 lists AIA B2B interface utility service annotation elements in composite.xml.

Table 11-6 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 11-16.

Figure 11-16 AIA B2B Interface Utility Service Annotation Elements in composite.xml Snippet

```
<reference name="AIAB2BInterfaceService"
           ui:wsdlLocation="oramds:/apps/AIAMetaData/AIAComponents/Enterprise
  <interface.wsdl interface="http://xmlns.oracle.com/EnterpriseObjects/Core/
  <binding.ws port="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2
              location="http://ap6060fems.us.oracle.com:8001/soa-infra/servic
     <xs:annotation>
        <xs:documentation xml:lang="en">Details about invocation of AIAB2B
           <xs:appInfo>
                 <svcdoc:AIA>
                    <svcdoc:Reference>
                       <svcdoc:ArtifactType>UtilityService</svcdoc:Art
                       <svcdoc:ServiceOperation>
                          <svcdoc:Name>ProcessB2BDocument</svcdoc:Nam
                       </svcdoc:ServiceOperation>
                    </svcdoc:Reference>
                 </svcdoc:AIA>
           </xs:appInfo>
     </xs:annotation>
</reference>
```

3. The $AIA_HOME/Infrastructure/B2B/src/B2BConnectors/EDIX12B2BCSApp/X12Update SalesOrder ProvB2BCSImpl/composite.xml file has sample provider B2BCS Impl annotations for your reference.

How to Support Trading Partner-Specific Variants

This section includes the following topics:

- Supporting Trading Partner-Specific Custom Extensions
- Supporting Trading Partner-Specific XSLTs
- 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.

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.

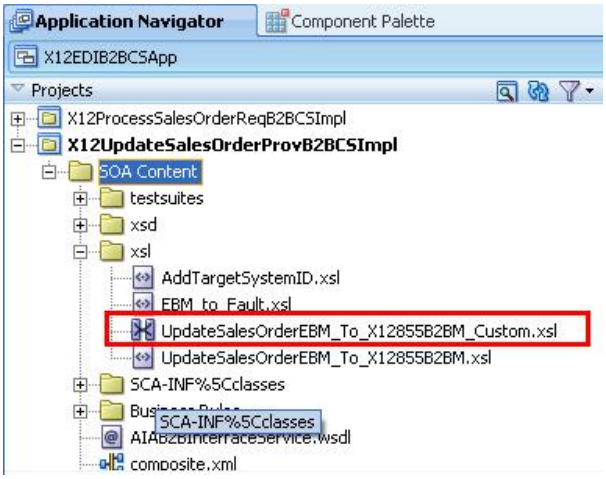
For more information about support for XSLT extension using callouts to custom XSLT templates, see Developing Extensible ABCS.

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 11-17.

Figure 11-17 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 11-18, which is included in the main XSLT.

Figure 11-18 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 11-19.

Figure 11-19 Trading Partner ID Passed as an Input to the Custom XSLT Template Call

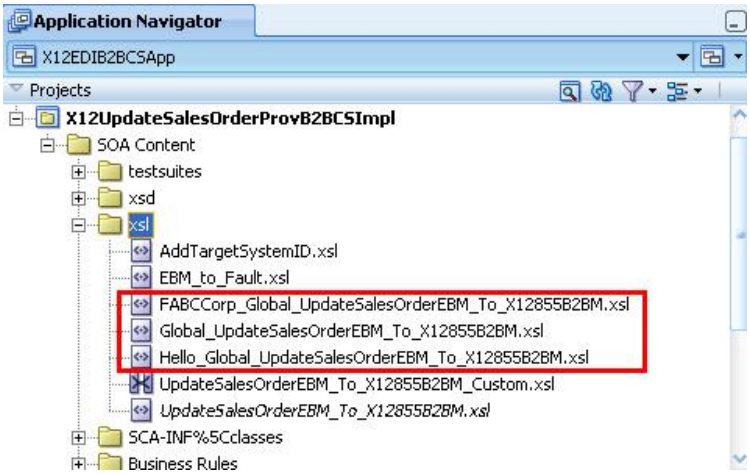
```
<xsl:template name="Segment-ST_ext">
    <xsl:param name="currentNode">
    <xsl:param name="$TradingPartnerID">
    <xsl:if test='$TradingPartnerID= "ABCCorp"'>
       <!-- Add custom mappings to Segement-ST for trading partner "ABC Corp" here.
    </xsl:if>
    <xsl:if test='$TradingPartnerID= "Hello Inc"'>
       <!-- Add custom mappings to Segement-ST for trading partner "Hello Inc" here.
    </xsl:if>
    <xsl:if test='$TradingPartnerID= "Global"'>
       <!-- Add custom mappings to Segement-ST for trading partner "Global" here.
    </xsl:if>
</xsl:template>
```

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.

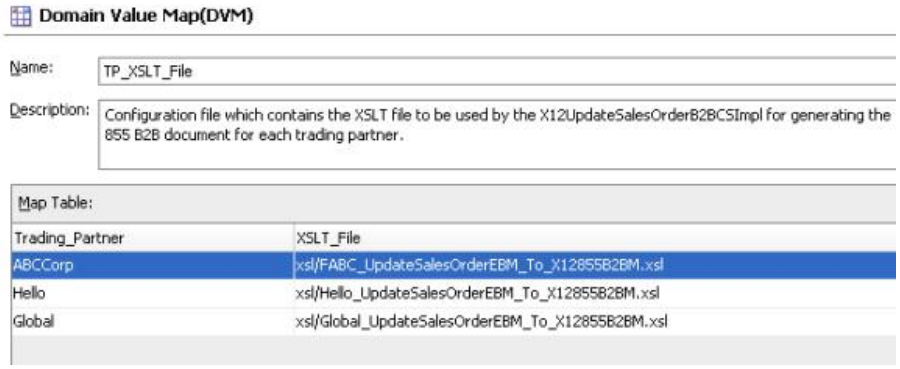
To meet such requirements, you can decide to develop one XSLT file for each trading partner that needs custom mappings. You can make a copy of a shipped XSLT file, edit it to include partner-specific mapping logic, and save it by using a partner prefix in the filename, as shown in Figure 11-20.

Figure 11-20 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 11-21.

Figure 11-21 DVM File Used to Map Trading Partners to XSLT Files



In the provider B2BCS Impl BPEL process, as shown in Figure 11-22, you can do a lookup on this DVM file, as shown in Figure 11-23, to obtain the XSLT filename, as shown in Figure 11-24.

Figure 11-22 Provider B2BCS Implementation BPEL Process Requiring Trading Partner-Specific XSLT

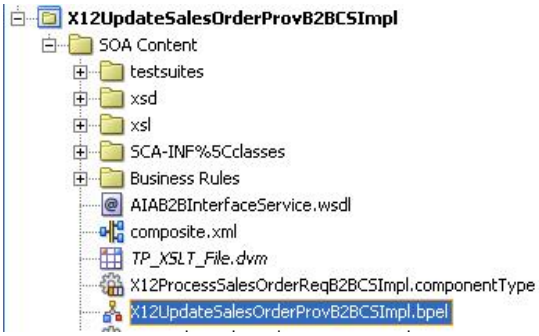


Figure 11-23 DVM Lookup to Obtain XSLT Filename

```
<assign name="GetXSLTFileName">
  <copy>
    <from expression='dvm:lookupValue("TP_XSLT_File.dvm","TradingPartner",
    <to variable="xslt_file_name"/>
  </copy>
</assign>
```

Figure 11-24 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>
```

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 11-25. This information can be looked up by the BPEL process to populate the /B2BMHeader/B2BDocumentType/TypeCode and /B2BMHeader/B2BDocumentType/Version attributes.

Figure 11-25 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.

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 SOA Core Extension*.

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 11-26 and Figure 11-27.

Figure 11-26 B2BCS Composite Defined to Invoke AIAAsyncErrorHandlingBPELProcess

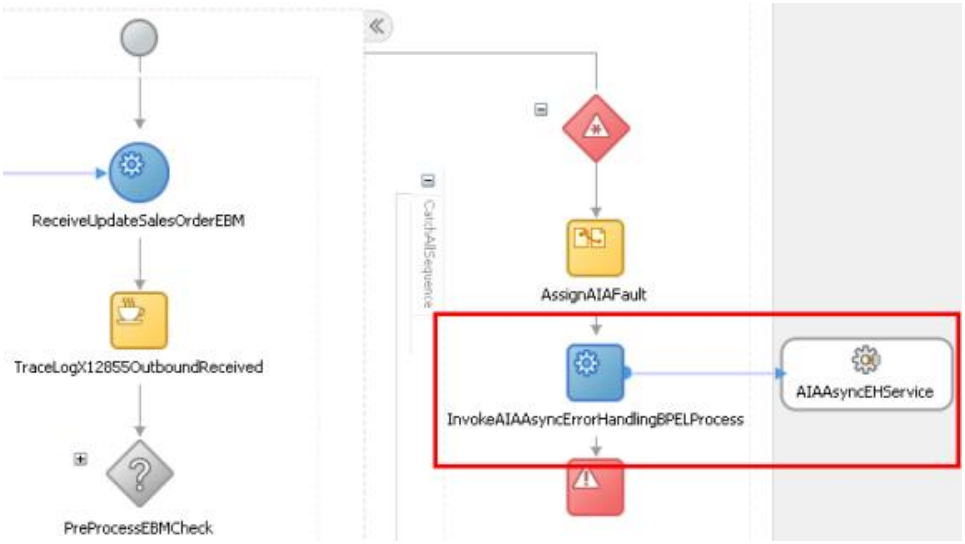


Figure 11-27 Invocation of the AIAAsyncEHService

```
    </assign>
          <invoke name="InvokeAIAAsyncErrorHandlingBPELProcess"
                  portType="aiaasynceh:AIAAsyncErrorHandlingBPELProcess" inputVariable="AIAA
                  partnerLink="AIAAsyncEHService"
                  operation="initiate"/>
          <throw name="ThrowAIAFault" faultName="corecom:Fault" faultVariable="AIAFaultMessa
       </sequence>
    </catchAll>
</faultHandlers>
```

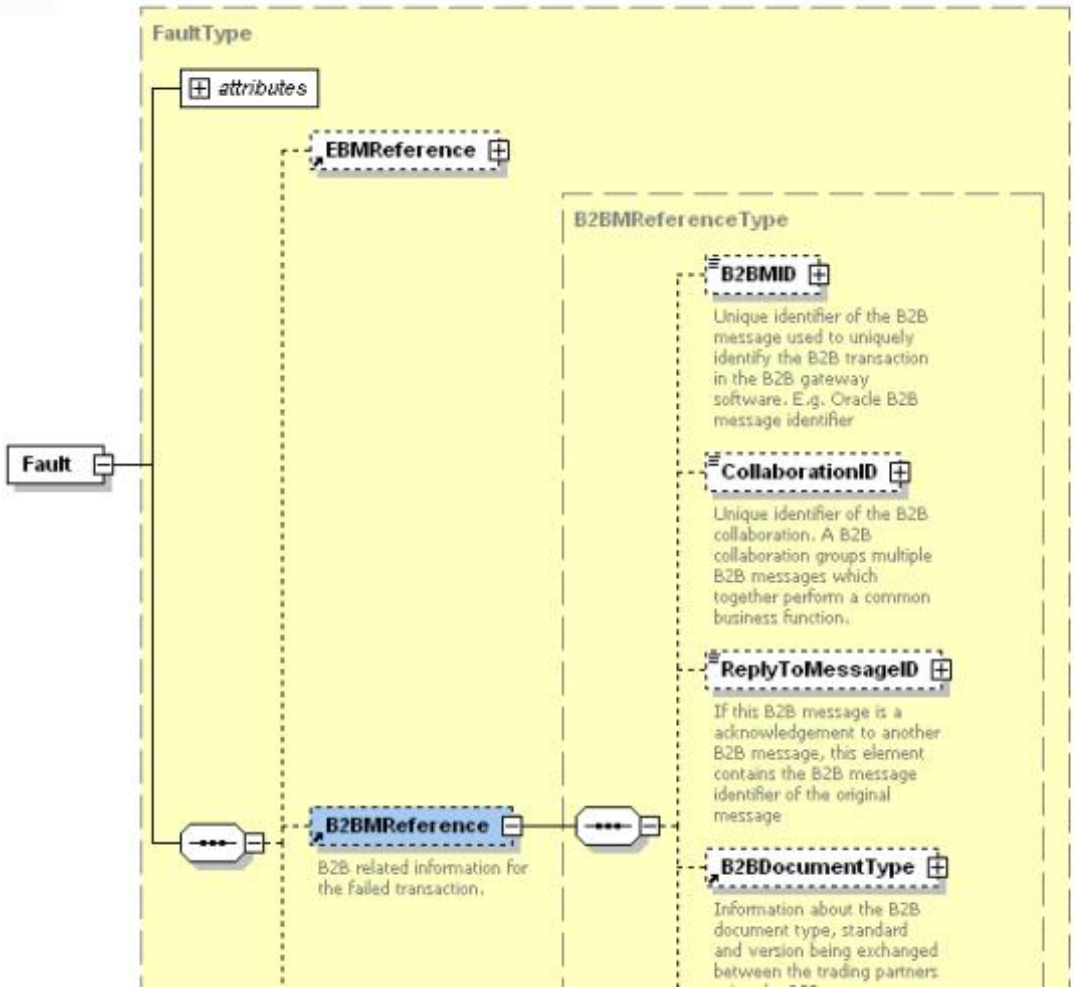
While invoking the AIAAsyncErrorHandlingBPELProcess, the following B2B-specific elements in the fault schema can be populated as described in Table 11-7.

Table 11-7 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 11-28 provides the B2B-specific elements in the corecom:Fault.

Figure 11-28 B2B-Specific Elements in the corecom:Fault

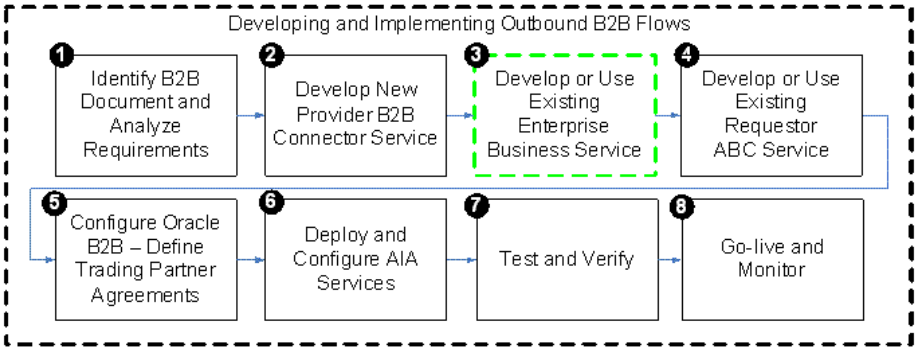


The B2B details of the failed AIA service that are available in the fault instance are logged and available for debugging the failed flow.

Step 3: Developing or Extending an Existing Enterprise Business Service

The next step, as shown in Figure 11-29, 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 11-29 Step 3: Developing or Extending an Existing Enterprise Business Service



For more information about creating a new EBS, see Designing and Developing Enterprise Business Services.

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.

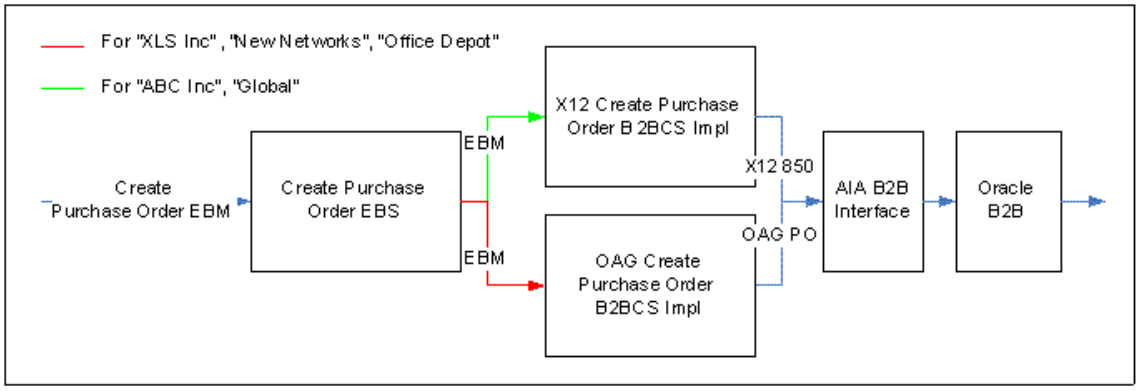
For example, suppliers XLS Inc., New Networks, and Office Systems may require that new purchase orders be sent using the OAG Process Purchase Order document format, while suppliers ABC Inc. and Global may require that new purchase orders be sent using the X12 850 document format.

To support the needs of this implementation, two B2BCSs must be developed, as shown in Figure 11-30:

- 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 11-30 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 11-31.

Figure 11-31 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
                        ($in.CreatePurchaseOrderEBM/EBMHeader/B2BProfile/ReceiverTradingPartnerID='Office Depot'))
            </expression>
         </condition>
         <action>
            <transform/>
            <invoke reference="OAGCreateSalesOrderProvB2BCSImpl"
                    operation="CreatePurchaseOrder"/>
         </action>
      </case>
      <case executionType="direct"
            name="X12CreatePurchaseOrderProvB2BCS.CreatePurchaseOrder">
         <condition language="xpath">
            <expression>(($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 11-32.

Figure 11-32 DVM Used to Store Trading Partner B2B Preferences

Domain Value Map(DVM)

Name: CreatePurchaseOrder_TP_B2BConfig_DVM

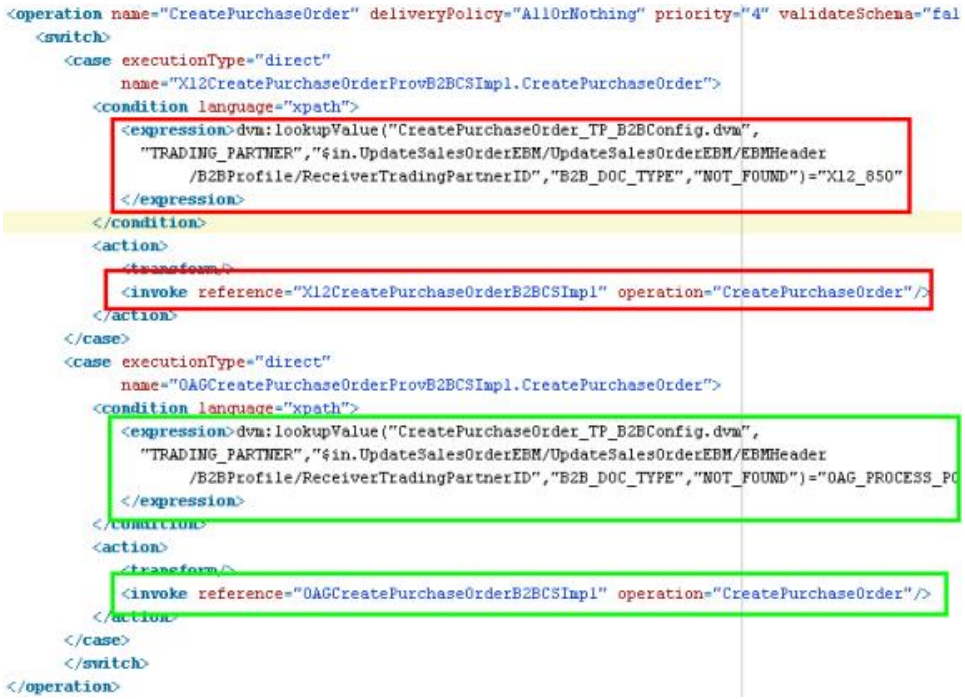
Description: This DVM file contains the B2B document format supported by different trading partners to receive new Purchase Orders.

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 runtime by the EBS implementation to determine the target provider B2BCS to be invoked, as shown in Figure 11-33.

Figure 11-33 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.

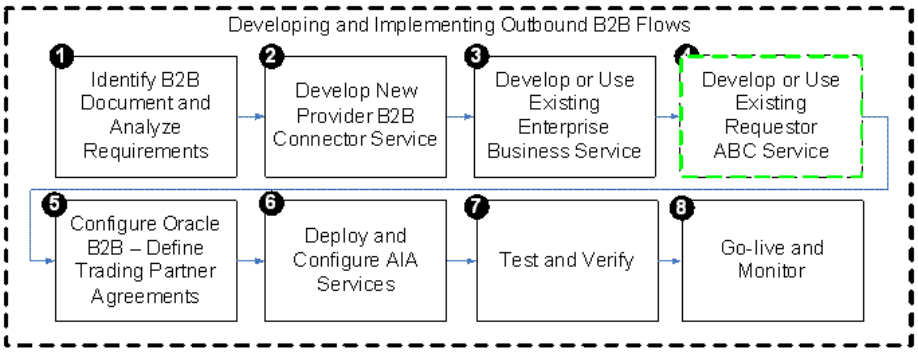
Step 4: Developing or Extending an Existing Requester ABCS

This section includes the following topics:

- What You Must Know About Message Exchange Patterns
- What You Must Know About Transformations

The next step, as shown in Figure 11-34, is to develop a new requester ABCS or extend an existing requester ABCS.

Figure 11-34 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 Designing Application Business Connector Services and 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.

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.

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 11-8 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 11-8 EBM Header Elements that Must be Mapped to Enable Trading Partner-Specific Routing in the EBS Layer

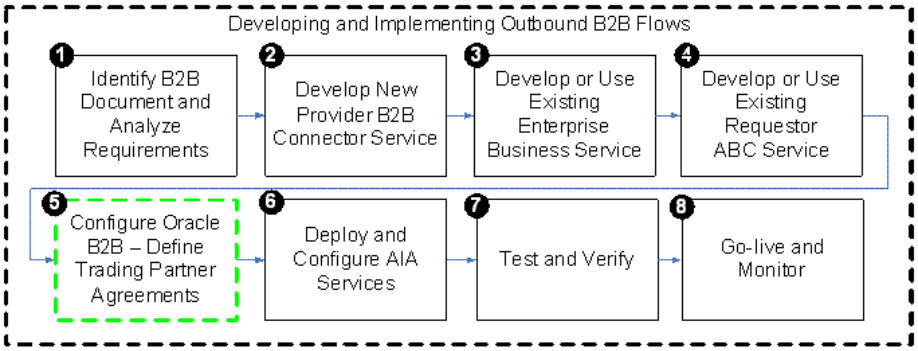
EBM Header Element	Description	Example Value
/EBMHeader/B2BProfile/ SenderTradingPartner/ TradingPartnerID	ID of the sending trading partner as defined in Oracle B2B. For outbound flows, this is the host" trading partner.	*MyCompany*
/EBMHeader/B2BProfile/ ReceiverTradingPartner/ TradingPartnerID	ID 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.

Step 5: Configuring Oracle B2B and Defining Trading Partner Agreements

The next step, as shown in Figure 11-35, is to create trading partner agreements in Oracle B2B.

Figure 11-35 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)

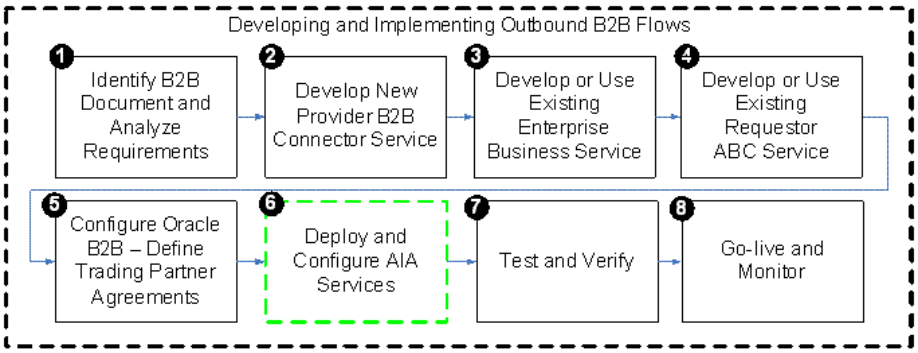
If the values provided by the source application for these fields and the trading partner names in Oracle B2B do not match, you can maintain a DVM file that contains the mapping between the source application's trading partner IDs and trading partner names of Oracle B2B.

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.

Step 6: Deploying and Configuring AIA Services

The next step, as shown in Figure 11-36, is to deploy the AIA services. You can deploy the services to a target Oracle SOA server using Oracle JDeveloper.

Figure 11-36 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.

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 SOA Core Extension.*

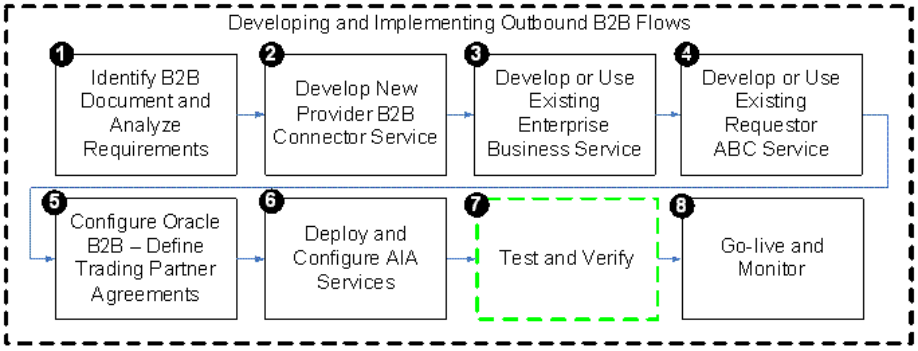
Step 7: Testing and Verifying

This section includes the following topics:

- How to Test Using CAVS
- How to Test Using Dummy Trading Partner Endpoints

The next step, as shown in Figure 11-37, is to test your newly developed or deployed AIA services that constitute the B2B integration flow.

Figure 11-37 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.

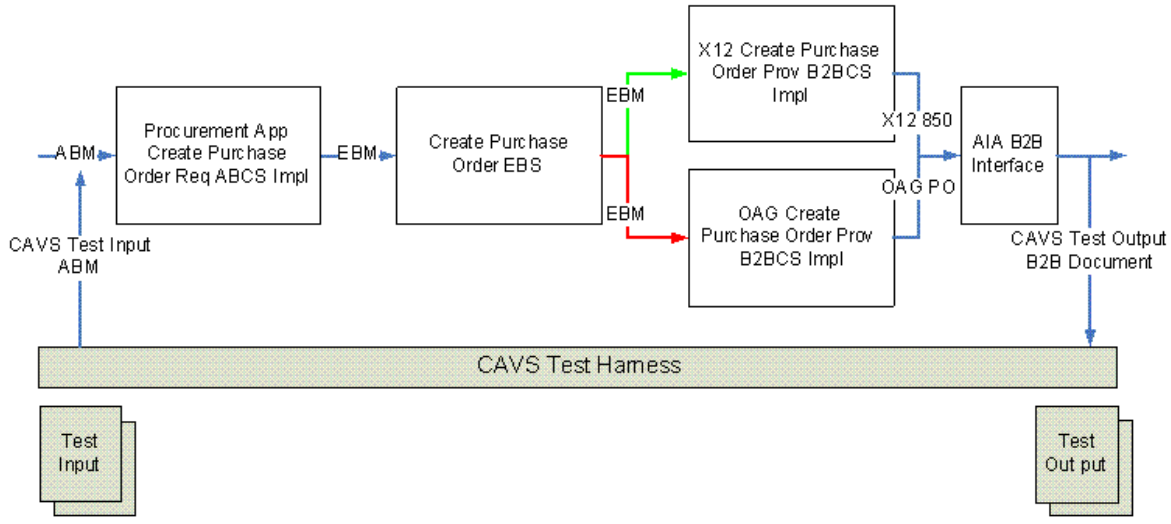
How to Test Using CAVS

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 11-38, is to verify that the AIA services are properly developed, deployed, and configured.

Figure 11-38 B2B Integration Flow Test Using CAVS



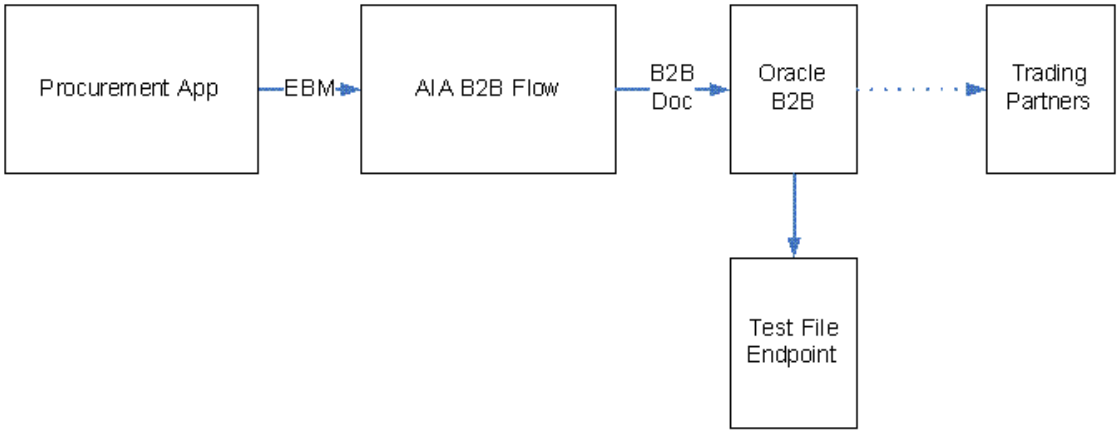
How to Test Using Dummy Trading Partner Endpoints

This section includes the following topics:

- How to Test Using the Production Code Value Set to "Test"
- 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 11-39. 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 11-39 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.

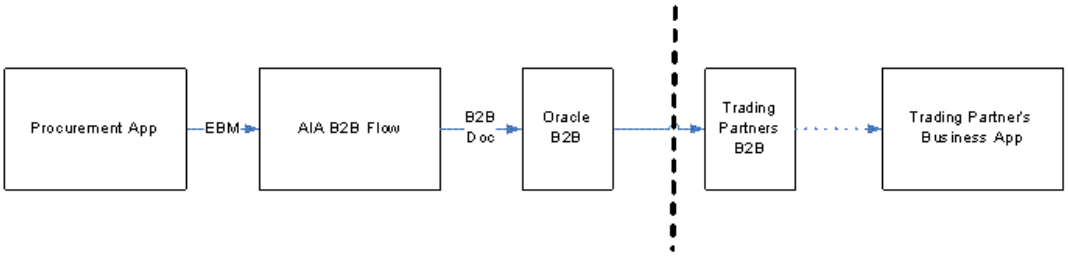
- The objective of this testing is to verify the integration between AIA services, the application, and B2B and to verify that the configuration across these layers is consistent.
- This testing can also take place without any knowledge of your trading partners.

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 back-end application, as shown in Figure 11-40.

Figure 11-40 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 acknowledgment, encryption, packaging, partner and document identification, and so forth are verified.

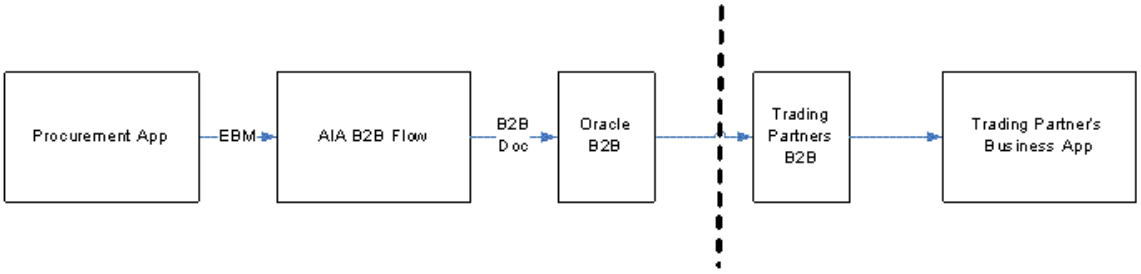
This approach requires that this test mechanism be discussed and agreed upon with each of your trading partners.

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 11-41, however, the trading partner's business users or rules discard the order request.

Figure 11-41 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.

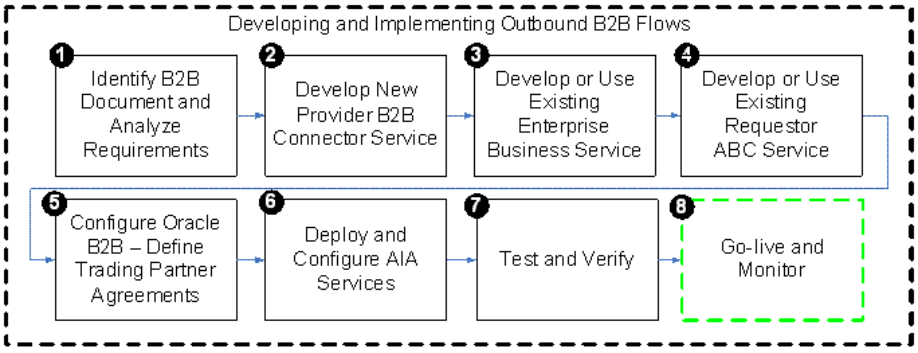
Step 8: Going Live and Monitoring

This section includes the following topics:

- Monitoring Using Oracle B2B Reports
- Monitoring Using Oracle Enterprise Manager Console
- Monitoring Using Error Notifications

The final step, as shown in Figure 11-42, 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 11-42 Step 8: Going Live and Monitoring



Monitoring Using Oracle B2B Reports

Oracle B2B's built-in reporting allows for the creation of the reports listed in Table 11-9. 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 11-9 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.*

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.

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

12

Developing and Implementing Inbound B2B Integration Flows

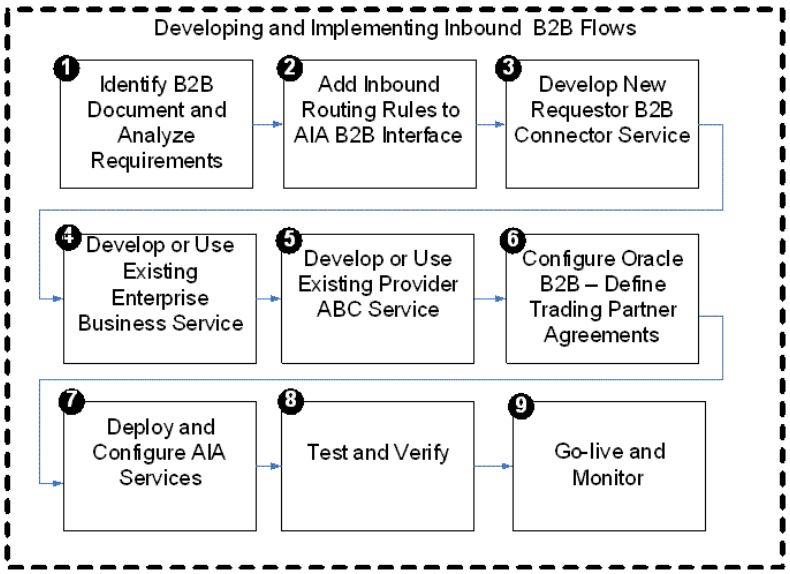
This chapter provides an overview of developing and implementing inbound B2B integration flows and describes how to identify the B2B document and analyze requirements, add inbound routing rules to an AIA B2B interface, develop new requester B2B connector service, develop or extend an existing Enterprise 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:

Introduction to Developing and Implementing Inbound B2B Integration Flows

Figure 12-1 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 12-1 High-Level Steps to Develop and Implement a Simple Inbound B2B Flow

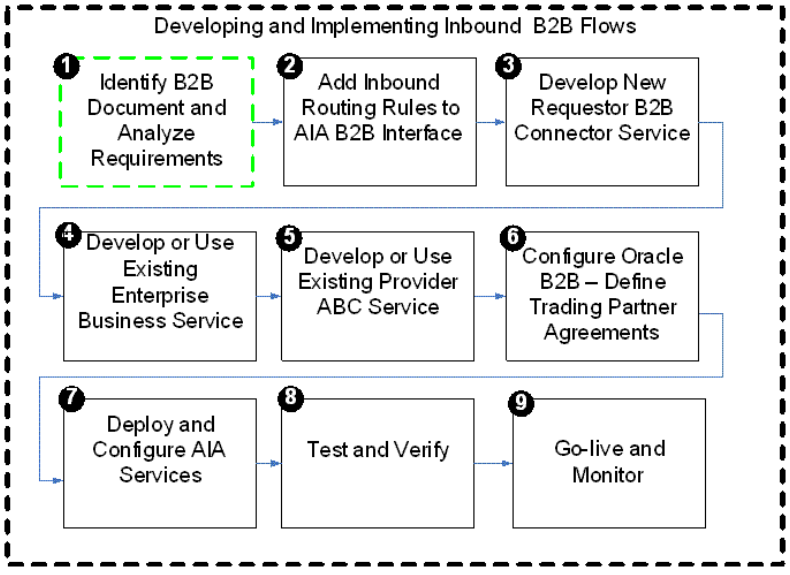


Most of the preceding steps are described inDeveloping and Implementing Outbound B2B Integration Flows. This chapter is referred to whenever applicable.

Step 1: Identifying the B2B Document and Analyzing Requirements

The first step in building an inbound B2B flow, as shown in Figure 12-2, 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 12-2 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:

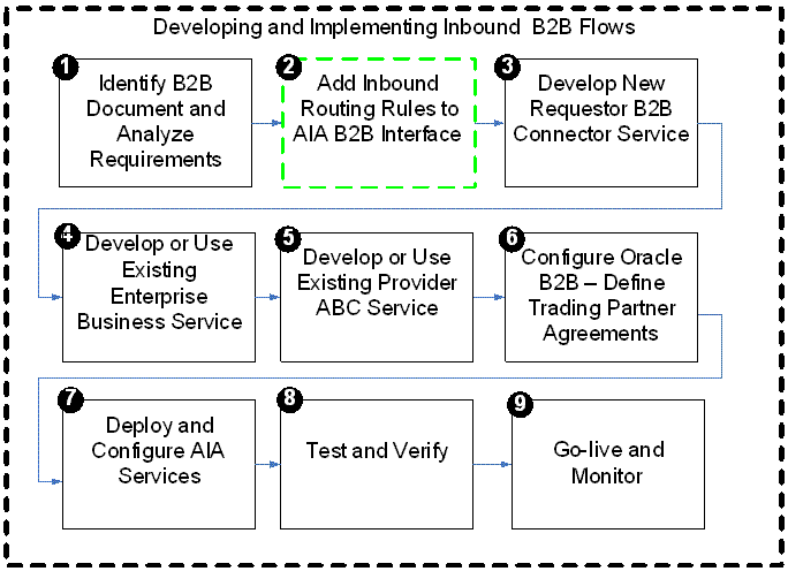
1. The source and targets in the mapping are reversed. The B2B document received from trading partners is the source of the mapping. The AIA Enterprise Business Message (EBM) is the target of the mapping.
2. At the end of this step:
   a. The B2B document is defined in Oracle B2B.
   b. The XML schema 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 Step 1: Identifying the B2B Document and Analyzing Requirements.

Step 2: Adding Inbound Routing Rules to an AIA B2B Interface

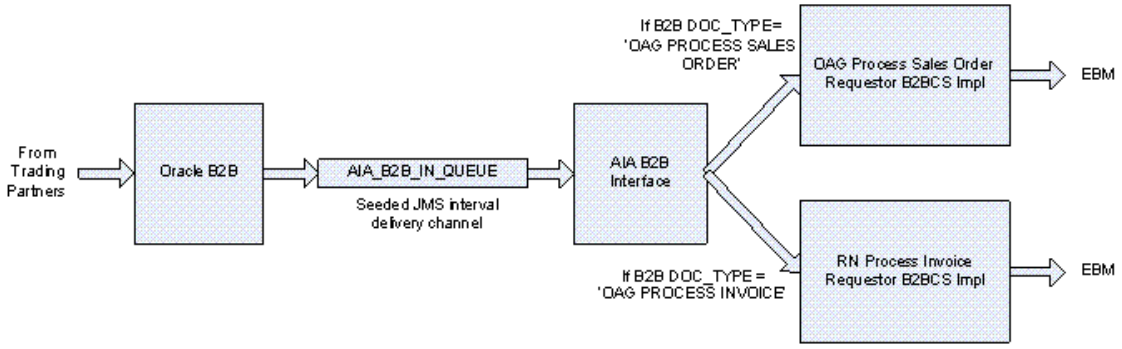
The next step in building an inbound B2B flow, as shown in Figure 12-3, is to add routing rules to the AIAB2BInterface Infrastructure service.

Figure 12-3 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 12-4.

Figure 12-4 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.

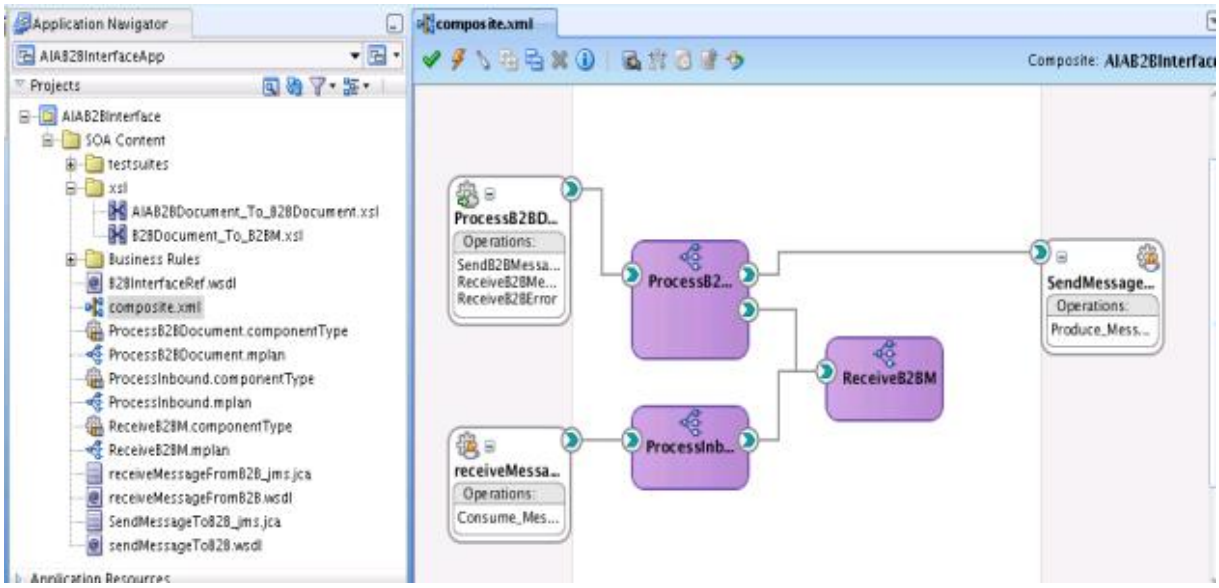
    The value is the trading partner name as defined in Oracle B2B.
  - Example: Acme
- /corecom:B2BM /corecom:Payload
  - Mapped from: JMS payload
  - Description: The actual JMS text payload that contains the B2B document sent by the trading partner.
  - Example: <?xml version="1.0"?> <PROCESS_INVOICE_002> <CONTROLAREA>…

How to Add a New Routing Rule to the AIA B2B Interface

To add a new routing rule to the AIA B2B Interface:

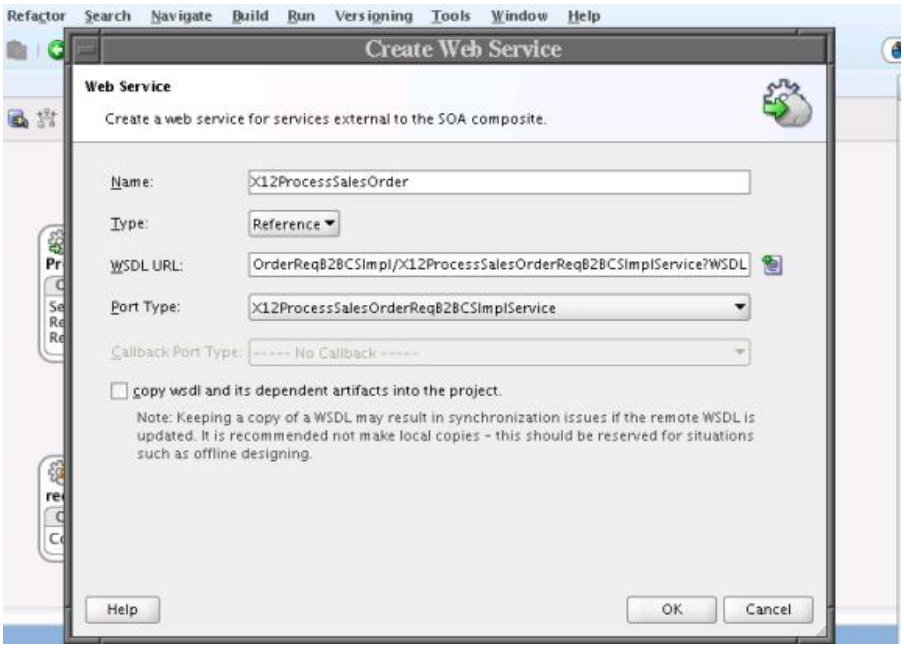
1. Open the $AIA_HOME/Infrastructure/B2B/AIAB2BInterfaceApp/ AIAB2BInterfaceApp.jws application using Oracle JDeveloper.
2. Open the composite.xml file using Oracle JDeveloper, as shown in Figure 12-5.

Figure 12-5 composite.xml in Oracle JDeveloper



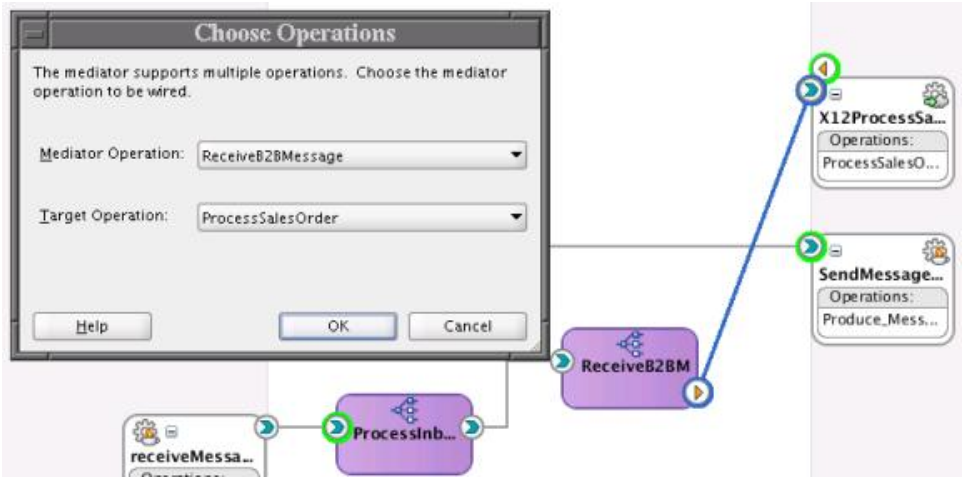
3. Add a new service reference to the requester B2BCS that processes the new inbound B2B document type as shown in Figure 12-6. The next section of this document explains how to develop requester B2BCSs.

Figure 12-6 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 12-7.

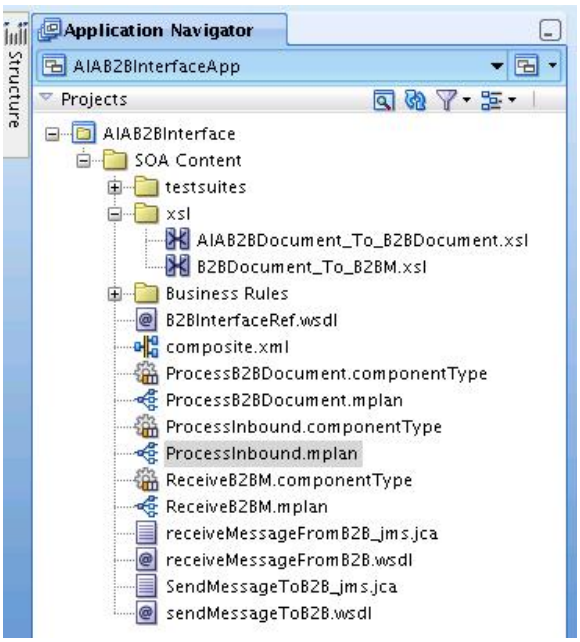
Figure 12-7 Wiring Added to the Newly Added Service from the Receive B2BM Mediator Component in the Composite



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.

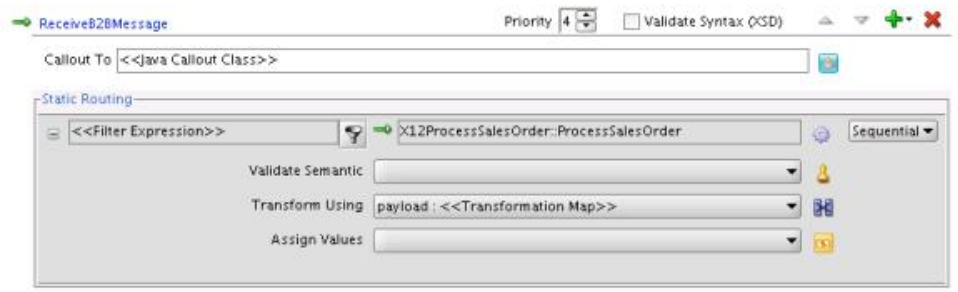
6. Open the file ProcessInbound.mplan as shown in Figure 12-8.

Figure 12-8 ProcessInbound.mplan



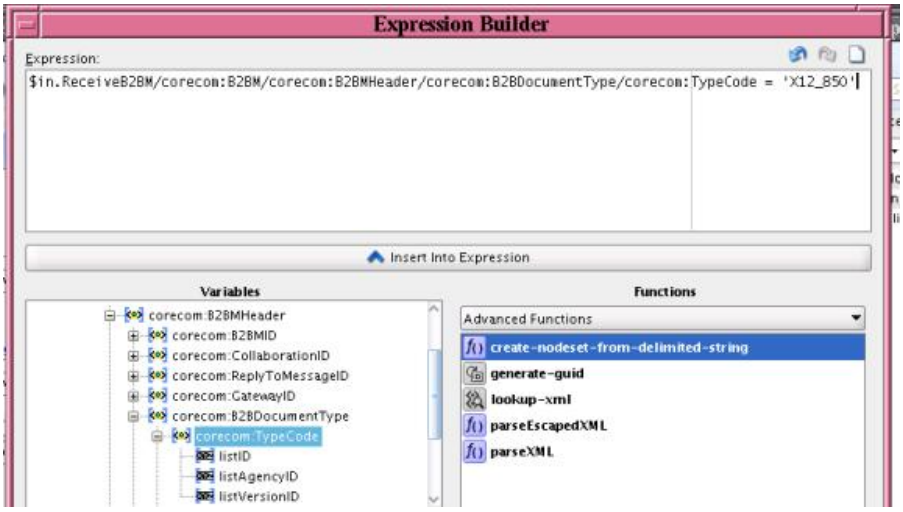
7. Add a new routing rule to invoke the requester B2BCS using the service reference added to the file composite.xml, as shown in Figure 12-9.

Figure 12-9 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 12-10 ensures that the EDI X12 Process Sales Order documents are routed to the X12 ProcessSalesOrder B2BCS Implementation.

Figure 12-10 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.

Step 3: Developing a New Requester B2B Connector Service

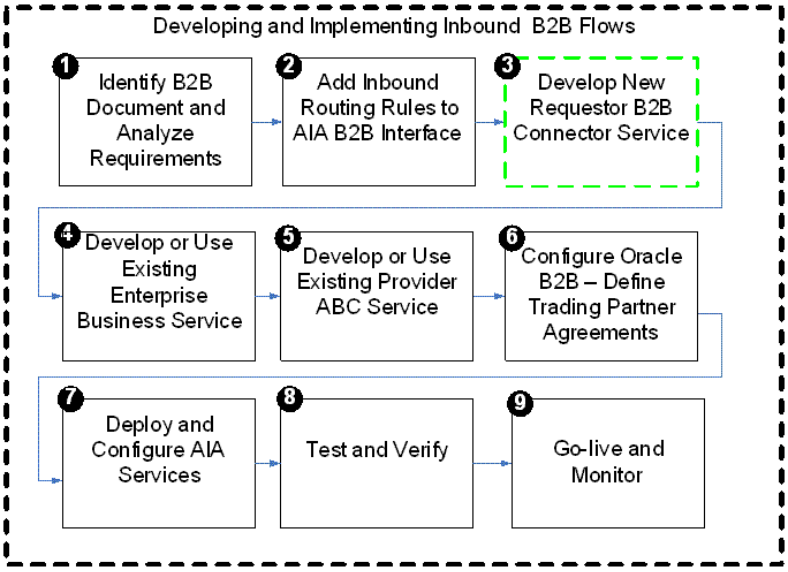
This section includes the following topics:

- Introduction to Requester B2B Connector Services
- How to Identify the Message Exchange Pattern
- How to Develop a B2BCS Service Contract
- How to Store a WSDL in Oracle Metadata Services Repository
- How to Develop a B2B Connector Service

- How to Annotate B2B Connector Services
- How to Support Trading Partner-Specific Variants
- How to Enable Error Handling

The next step, as shown in Figure 12-11, is to develop the requester B2BCS to support the outbound B2B integration flow.

Figure 12-11 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 Constructing the ABCS.

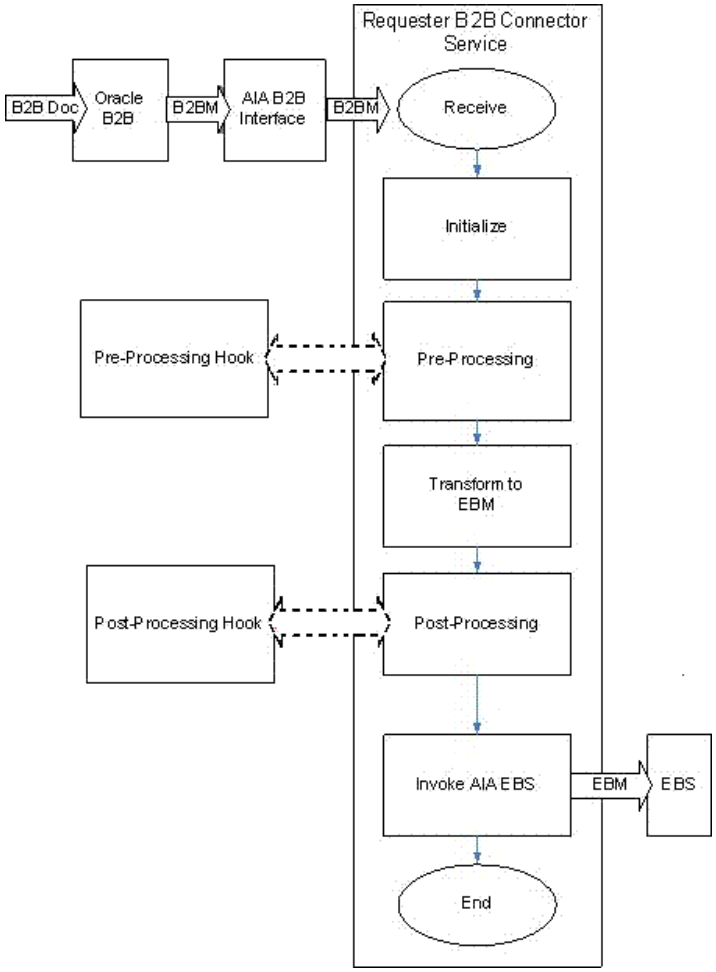
Introduction to Requester B2B Connector Services

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 12-12 illustrates the processing that takes place in a simple fire-and-forget message exchange pattern-based provider B2BCS.

Figure 12-12 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.

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 acknowledgment message to trading partners.

For more information, see How to Identify the Message Exchange Pattern.

For more information about identifying message exchange patterns, see Identifying the MEP.

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 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 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

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

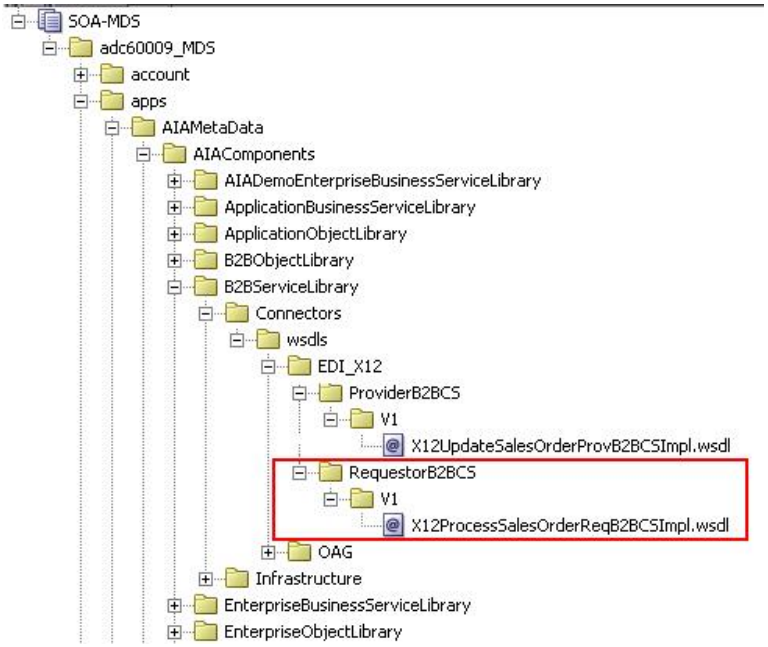
   /Connectors/wsdls/<B2B_STANDARD>/RequesterB2BCS

   /<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 12-13.

   Figure 12-13 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

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.

You should use Oracle JDeveloper to develop your 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 12-1.

Table 12-1 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 SOA Core Extension installed.

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.

To annotate requester B2B Connector Services:

1. Table 12-2 lists the annotation elements that must be added to the service element composite.xml, as described subsequently.

Table 12-2 Service Annotation Elements in composite.xml

Annotation Element	Description	Example
AIA/Service/InterfaceDetails/ServiceOperation	<Verb><EBOName>	ProcessSalesOrder
AIA/Service/ImplementationDetails/ArtifactType	RequesterB2BCSImplementation	RequesterB2BCSImplementation
AIA/Service/ImplementationDetails/ServiceOperation/Name	<Verb><EBOName>	ProcessSalesOrder
AIA/Service/ImplementationDetails/B2BDocument	B2B Document Type processed by this B2BCS	850
AIA/Service/ImplementationDetails/B2BDocumentVersion	B2B Document Version processed by this B2BCS	4010
AIA/Service/ImplementationDetails/B2BStandard	B2B Standard processed by this B2BCS	X12
AIA/Service/ImplementationDetails/B2BStandardVersion	B2B Standard Version processed by this B2BCS	4010

The following XML snippet, as shown in Figure 12-14, is an example of an annotated B2BCS called X12ProcessSalesOrderReqB2BCSImpl composite.xml.

Figure 12-14 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 12-3.

Table 12-3 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 12-15.

Figure 12-15 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.

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 How to Support Trading Partner-Specific Variants.

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 *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 12-16 and Figure 12-17.

Figure 12-16 B2BCS Composite Defined to Invoke AIAAsyncErrorHandlingBPELProcess

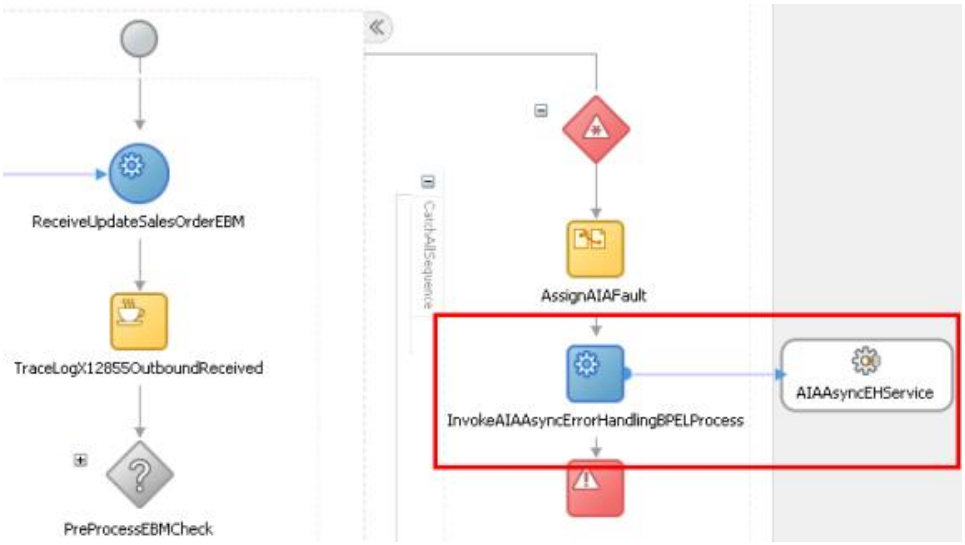


Figure 12-17 Invocation of the AIAAsyncEHService

```
    </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 12-4.

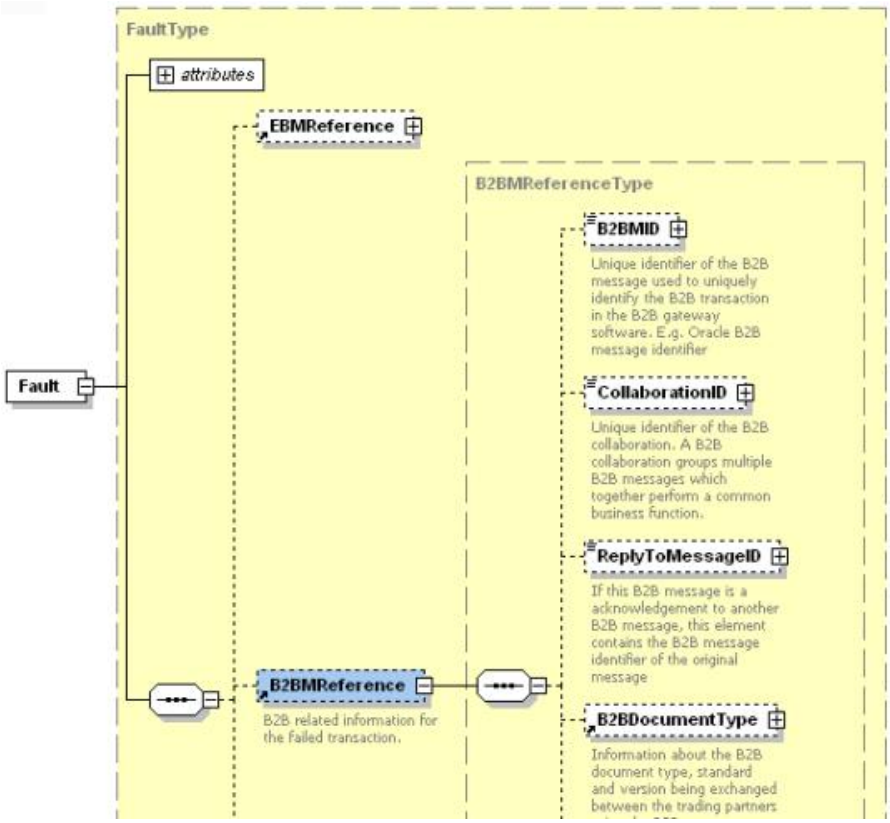
Table 12-4 B2B-Specific Elements in the Fault Schema That Can Be Populated by AIAAsyncErrorHandlingBPELProcess

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

Fault Element Schema	Description	Example
Fault/B2BMReference/ ReceiverTradingPartner/ TradingPartnerID	Receiver trading partner, mapped from the B2BM	Global

Figure 12-18 provides the B2B-specific elements in the corecom:Fault.

Figure 12-18 B2B-Specific Elements in corecom:Fault

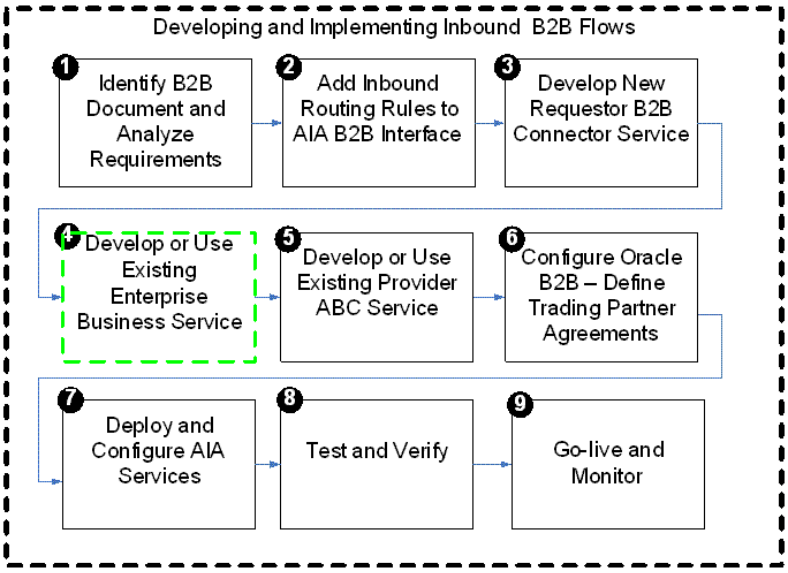


The B2B details of the failed AIA service that are available in the fault instance are logged and available for debugging the failed flow.

Step 4: Developing or Extending an Existing Enterprise Business Service

The next step, as shown in Figure 12-19, is to develop a new EBS or use an existing EBS that is invoked by the requester B2BCS.

Figure 12-19 Step 4: Developing or Extending an Existing Enterprise Business Service

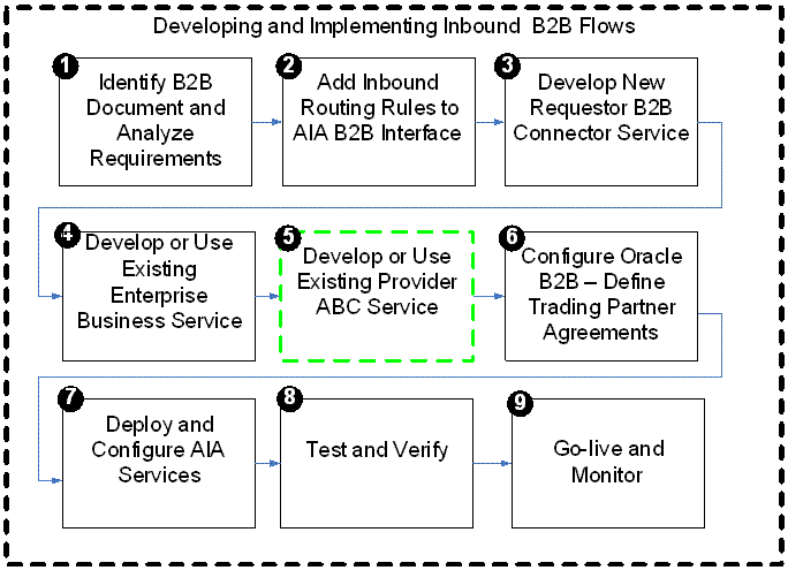


For more information about creating a new EBS, see Designing and Developing Enterprise Business Services.

Step 5: Developing or Extending an Existing Provider ABCS

The next step, as shown in Figure 12-20, 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 12-20 Step 5: Developing or Extending an Existing Provider ABCS



For more information about how to design and construct a provider ABCS, see Designing Application Business Connector Services and Constructing the ABCS.

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 12-5.

Table 12-5 Sender and Receiver Trading Partner Fields in the EBM

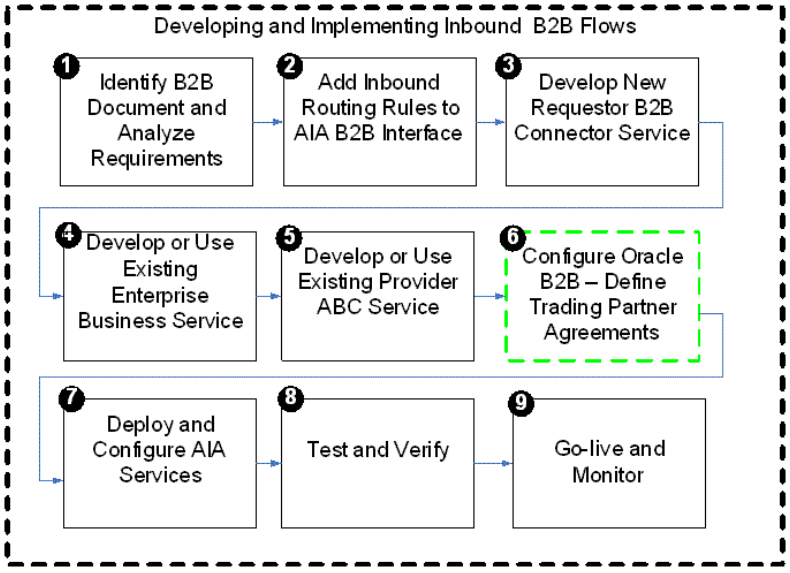
EBM Header Element	Description	Example Value
/EBMHeader/B2BProfile/ SenderTradingPartner/ TradingPartnerID	ID of the sending trading partner as defined in Oracle B2B. For inbound flows, this is the remote trading partner.	GlobalChips
/EBMHeader/B2BProfile/ ReceiverTradingPartner/ TradingPartnerID	ID 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.

Step 6: Configuring Oracle B2B and Defining Trading Partner Agreements

The next step, as shown in Figure 12-21, is to create trading partner agreements in Oracle B2B.

Figure 12-21 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 *User's Guide for Oracle B2B*.

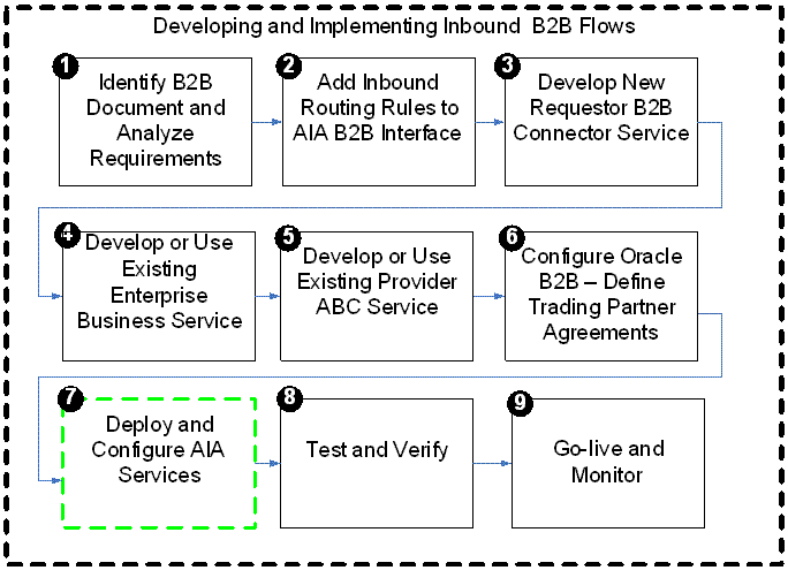
In addition, for EDI-based outbound B2B flows, Oracle B2B can be configured for batch processing of outbound documents. Parameters such as the batch size and time-

out can be configured in Oracle B2B without having to make any changes to the AIA layer.

Step 7: Deploying and Configuring AIA Services

The next step, as shown in Figure 12-22, is to deploy the AIA services. You can deploy the services to a target Oracle SOA server using Oracle JDeveloper.

Figure 12-22 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 Implementing Direct Integrations.

For more information about generating deployment plans, see Implementing Direct Integrations.

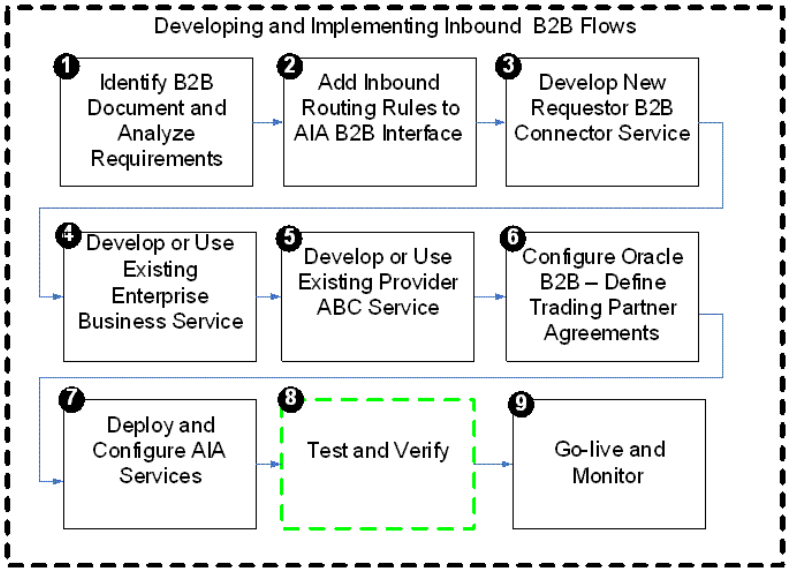
In addition, configure the AIA Error Handling framework and set up appropriate roles to be notified of errors in AIA flows.

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

Step 8: Testing and Verifying

The next step, as shown in Figure 12-23, 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 12-23 Step 8: Testing and Verifying

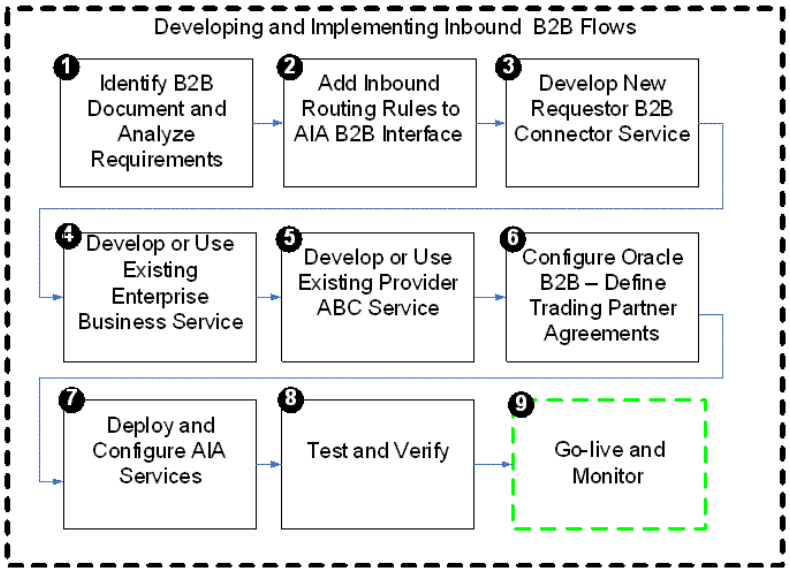


For more information, see Step 7: Testing and Verifying.

Step 9: Going Live and Monitoring

The final step, as shown in Figure 12-24, 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 12-24 Step 9: Going Live and Monitoring



For more information, see Step 8: Going Live and Monitoring.

13

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:

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.

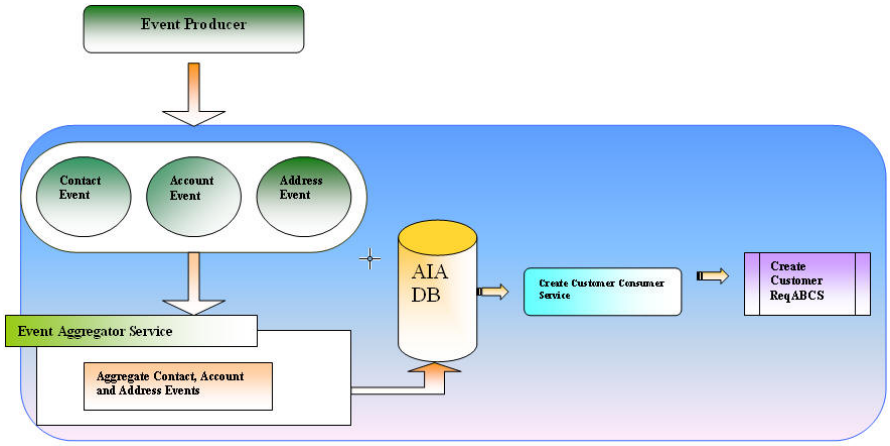
The Event Aggregation programming model provides:

- 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 13-1 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 13-1 Event Aggregation Service Raising a Single Coarse-Grained Event from Multiple Fine-Grained Events



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.

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 recommends 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.

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.

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.

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.

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 13-2.

Figure 13-2 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 13-3.

Figure 13-3 Example for Creating a Stored Procedure Object

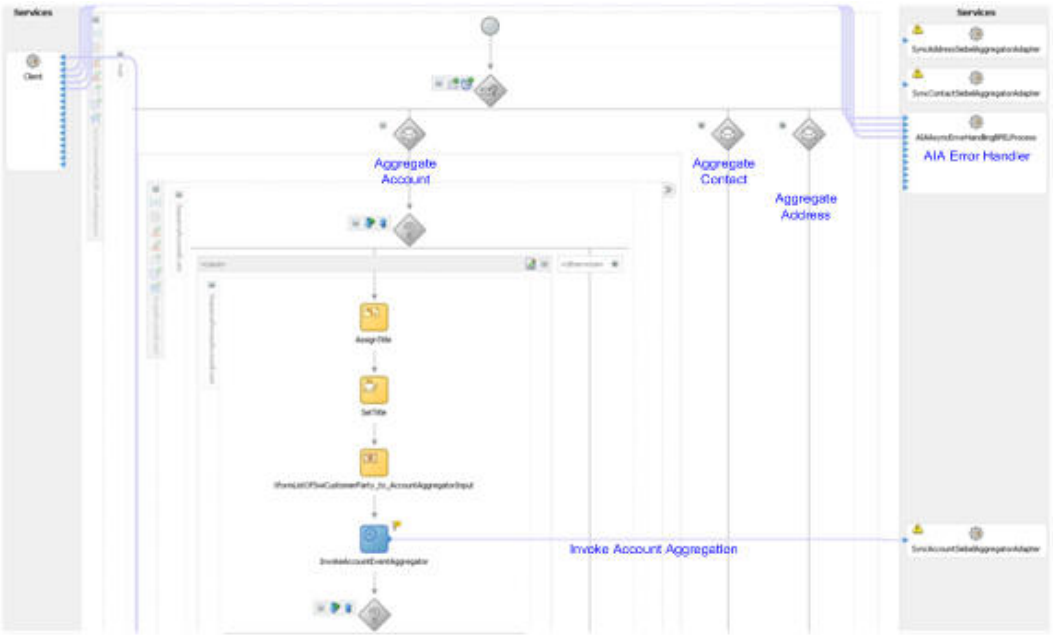
```
PROCEDURE SIEBEL_AGGREGATE_ACCOUNT(ACCOUNT_ID  IN VARCHAR2
                                 , CONTACT_IDS IN AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                 , ADDRESS_IDS IN AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                 , EP_SERVER_ID IN 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);
```

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 13-4.

Figure 13-4 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.

For more information, see 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.

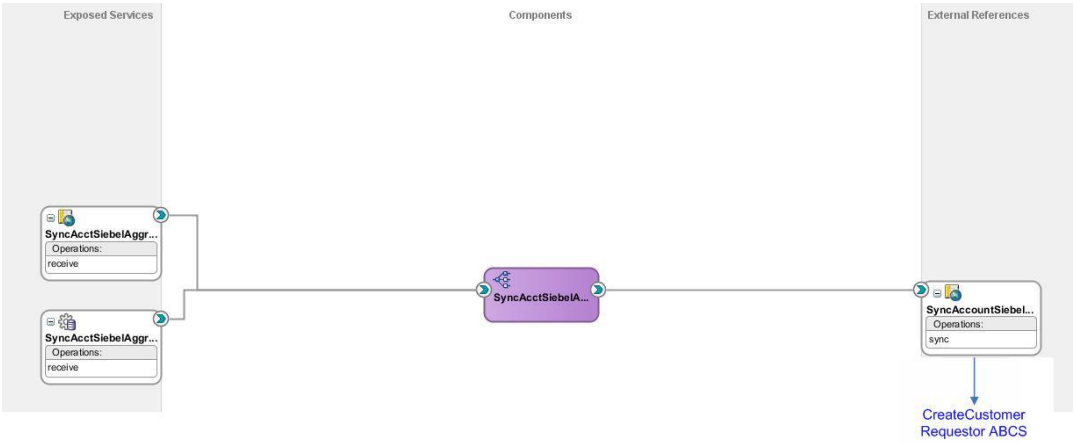
Creating Consumer Service

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 13-5 illustrates how to implement Consumer Service using Mediator Composite.

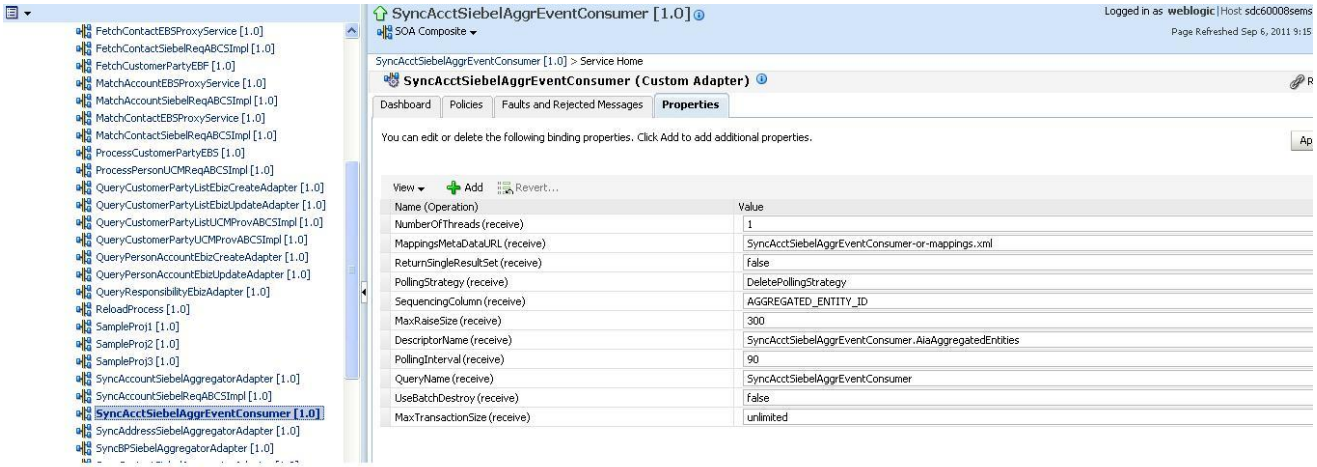
Figure 13-5 Consumer Service with Mediator Composite



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 13-6 illustrates how to configure time interval for polling on the consumer service.

Figure 13-6 Configure the Time Interval for Polling on the Consumer Service



For information on how to configure polling interval and other configurable properties, see "Configuring PollingInterval, MaxTransactionSize, and ActivationInstances" in *Understanding Technology Adapters.*

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.

For more information about the Error Handling framework, see "Setting Up Error Handling" in *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.

14

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:

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.

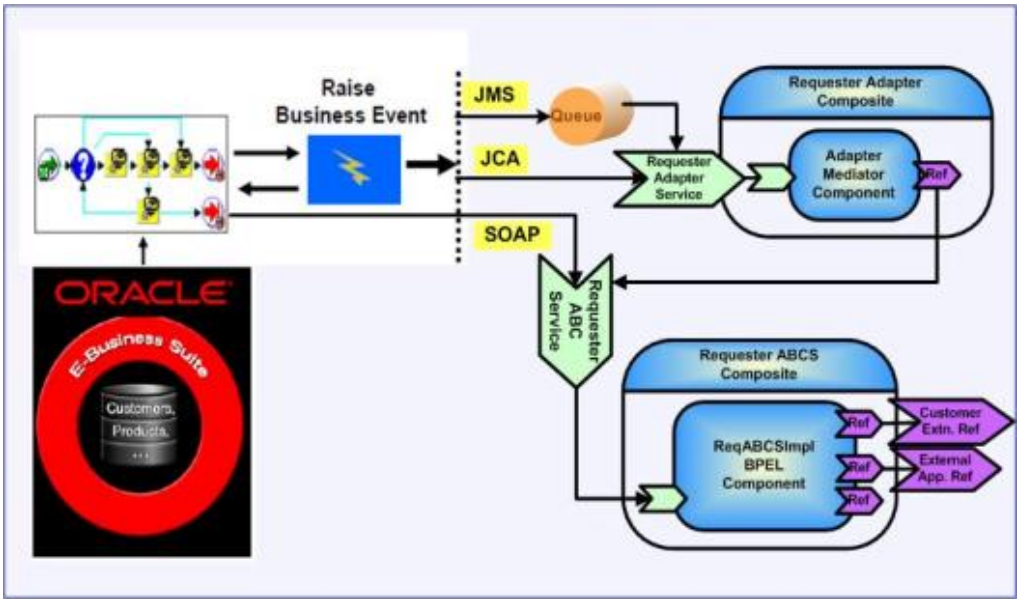
Inbound Connectivity

Inbound means inbound into the AIA layer, as shown in Figure 14-1.

Figure 14-1 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 14-1 Example of Inbound Connectivity



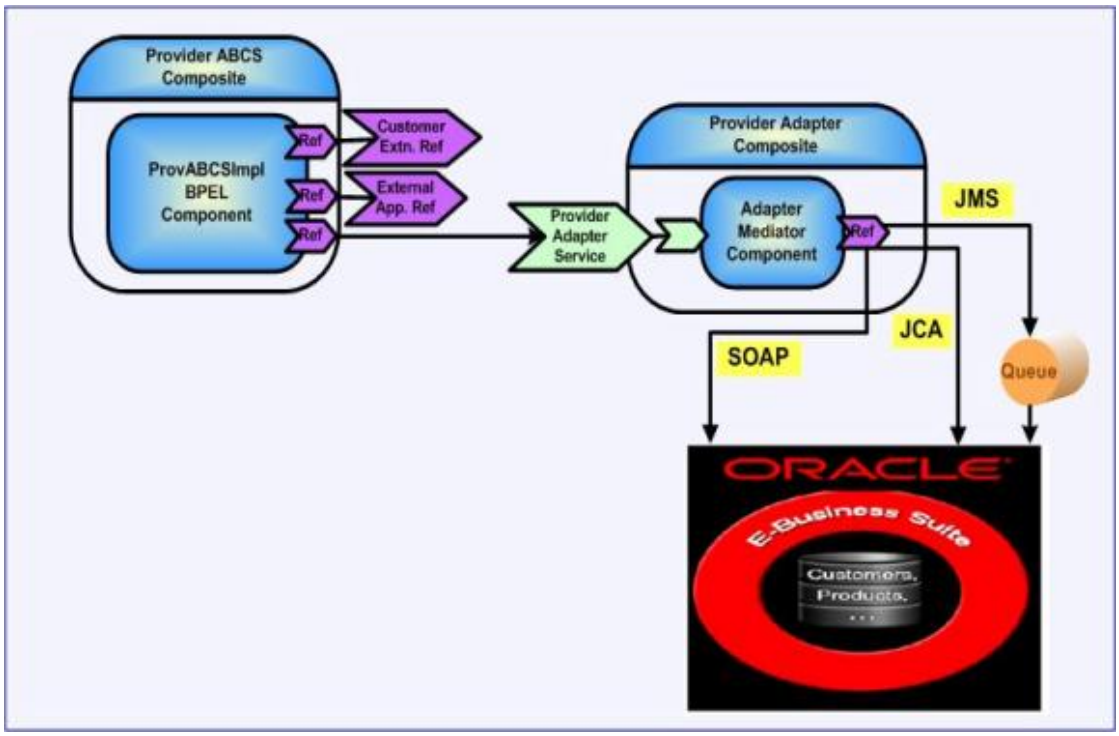
Outbound Connectivity

Outbound means outbound from the AIA layer, as shown in Figure 14-2.

- 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 14-2 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 14-2 Example of Outbound Connectivity



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:

- Web Services with SOAP/HTTP
- When to Use Web Services with SOAP/HTTP
- Session Management for Web Services with SOAP/HTTP
- Error Handling for Web Services with SOAP/HTTP
- Security for Web Services with SOAP/HTTP
- Message Propagation Using Queues or Topics
- Ensuring Guaranteed Message Delivery
- When to Use JCA Adapters

Note:

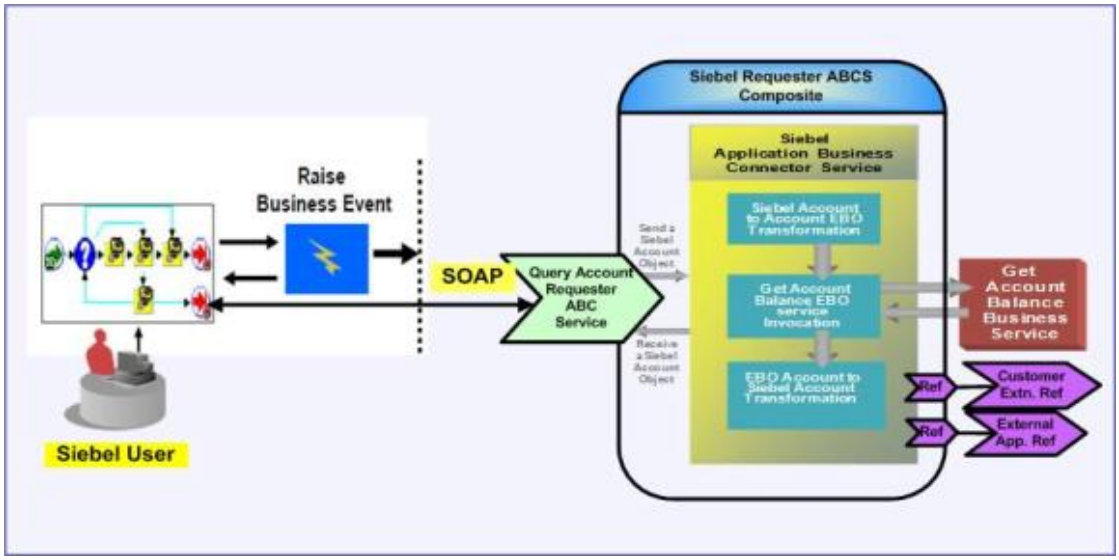
The contents of this section are general guidelines and recommendations only. Programming methodologies for AIA services can vary based on a participating application's capabilities and currently, some participating applications may not have the capabilities discussed in this section. Refer to the relevant application-specific guides for the facilities provided.

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 14-3, 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 14-3 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.

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

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.

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.

Advantages of Using Web Services with SOAP/HTTP

Web services with SOAP/HTTP are:

- Platform independent
- Language independent

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 Session Management for Web Services with SOAP/HTTP.

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.

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:

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.

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 14-1, Example 14-2, and Example 14-3 provide the URL and SOAP header for the Siebel application examples.

The SOAP header looks like this:

The response looks like this:

Example 14-1 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
```

Example 14-2 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>
```

Example 14-3 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>
```

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.

Error Handling for Web Services with SOAP/HTTP

This section discusses error handling for Web services with SOAP/HTTP:

- For Inbound Connectivity
- For Outbound Connectivity
- Request-Response and Request-Only System Errors
- Request-Response Business Errors

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).

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.

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.

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.

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.

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.

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.

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.

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

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 SOA Core Extension installation driver automatically creates the error queues for each queue.
- To generate the error queue name, the SOA Core Extension 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 SOA Core Extension 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.
- SOA Core Extension 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 SOA Core Extension 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.

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 14-4 and Figure 14-5.

Figure 14-4 Example of Multiple Milestones in a Business Process (1 of 2)

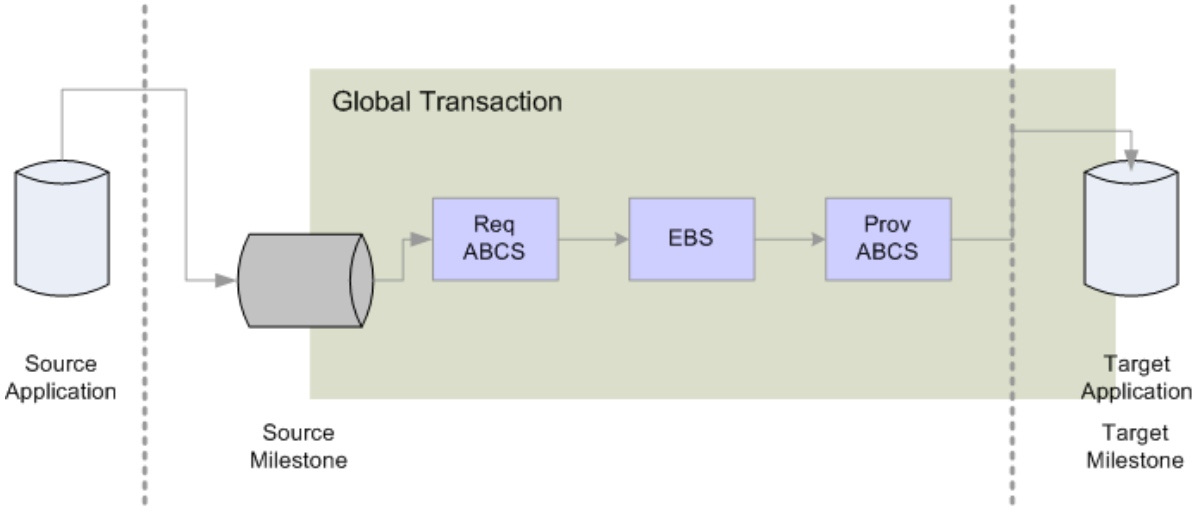
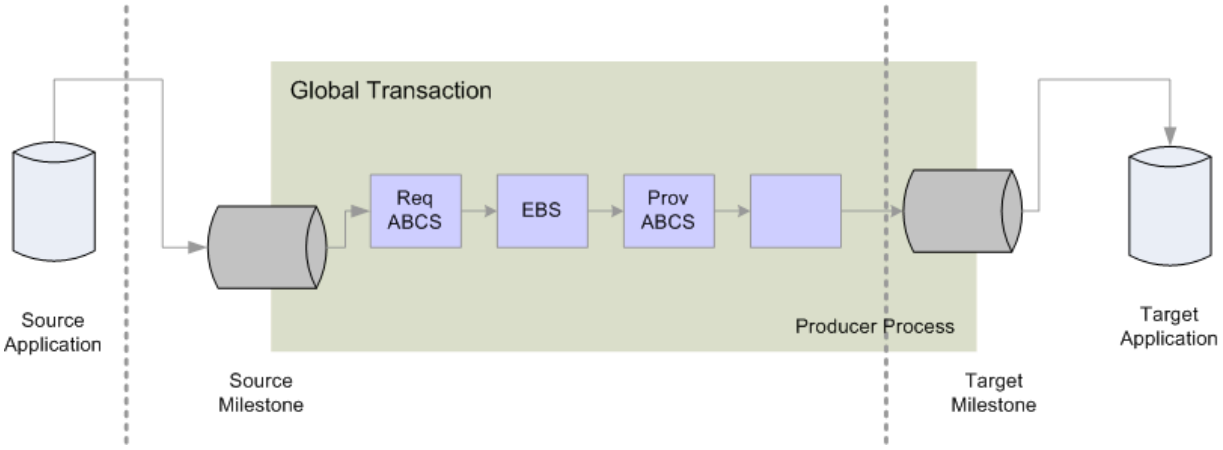


Figure 14-5 Example of Multiple Milestones in a Business Process (2 of 2)



- 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.

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.

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 *Administering JMS Resources for Oracle WebLogic Server.*

Siebel Application-Specific Connectivity Guidelines

The following sections discuss how to establish inbound and outbound connectivity with Siebel applications:

- Inbound: Siebel Application Interaction with AIA Services
- Web Services with SOAP/HTTP
- Creating JMS Consumers to Consume Siebel Messages from JMS Queues/Topics
- Outbound - Siebel Application Interaction with AIA Services
- Web Services with SOAP/HTTP

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*.

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 runtime. 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.

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.

Developer Tasks in the Service Design and Construction Phase

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 14-4 provides an example of version 1.
   b. Custom Attributes - Attributes in Example 14-5 are required:

      A sample of custom attributes is provided in Example 14-6.
5. Construct a requester ABCS.
6. Provide the WSDL from this service to the Siebel Application development team.

Example 14-4 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
```

Example 14-5 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"/>
```

Example 14-6 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>
```

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.

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.

Developer Tasks in Service Design and Outline Construction Phase

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 Implementing the Event Aggregation Programming Model.

5. Harvest to Oracle Enterprise Repository.

   For more information, see Implementing Direct Integrations.

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.

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.

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.

For more information, see:

- *Integration Platform Technologies: Siebel Enterprise Application Integration*, "Web Services," Examples of Invoking Web Services
- *Integration Platform Technologies: Siebel Enterprise Application Integration*, "Web Services," Mapping the xsd:any Tag in the XML Schema Wizard
- *Oracle Application Integration Architecture Pre-Built Integrations: Utilities Guide*, "Session Pool Manager"

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.

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:

- Inbound: E-Business Suite Application Interaction with AIA Services
- Concurrent Program Executable
- Business Event Subscription (JCA Connectivity Using OAPPS Adapter)
- Outbound: Oracle E-Business Suite Application Interaction with AIA Services

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).

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 runtime. 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.

Best Practices and Design Asynchronous Patterns

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:
4. Analyze the E-Business Suite requester ABS definition provided by solution architect.
5. Work with E-Business Suite development and discuss the possible design.
6. Finalize the content of the message from E-Business Suite.
7. 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 14-7 provides an example of version 0.
8. Construct a requester ABCS.

   Provide the WSDL from this service to the E-Business Suite development team.

Example 14-7 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
```

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 14-8 provides an example of version 1:
5. Construct a requester ABCS.
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 Implementing the Event Aggregation Programming Model.
14. Harvest to Oracle Enterprise Repository.

    For more information, see Implementing Direct Integrations.

Example 14-8 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
```

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 *Developing Fusion Web Applications with 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.

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 Working with AIA Design Patterns.

15

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 *Developing Integration Projects with Oracle Data Integrator*.

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:

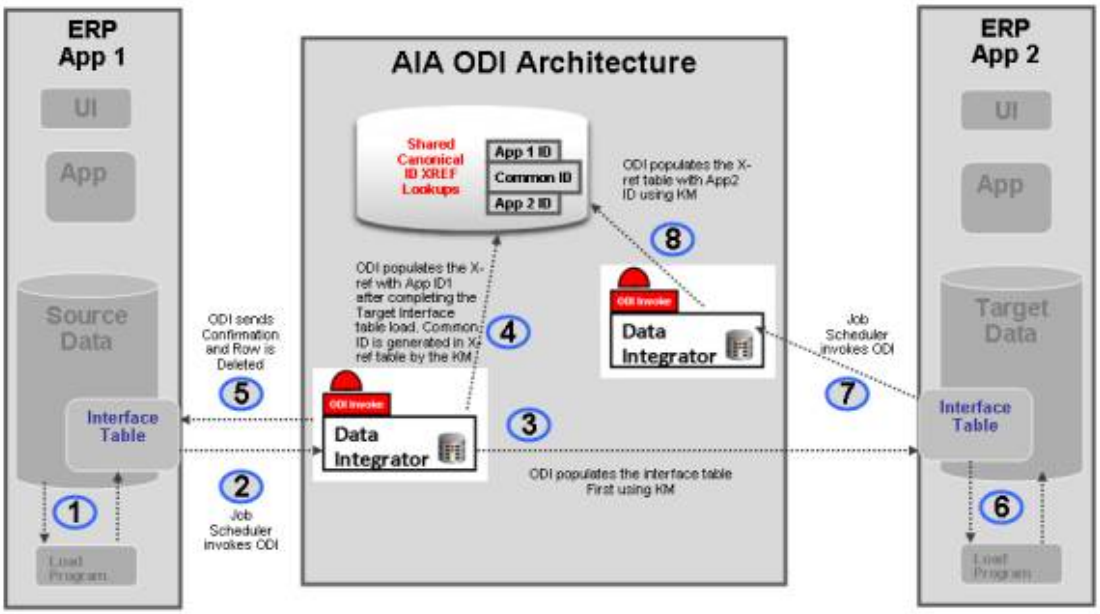
- High Volume Transactions with Xref Table
- Intermittent High Volume Transactions

Initial Data Loads

In this design pattern, shown in Figure 15-1, 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 15-1 Initial Data Loads



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 15-1.

---

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 *Developing SOA Applications with Oracle SOA Suite.*

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.

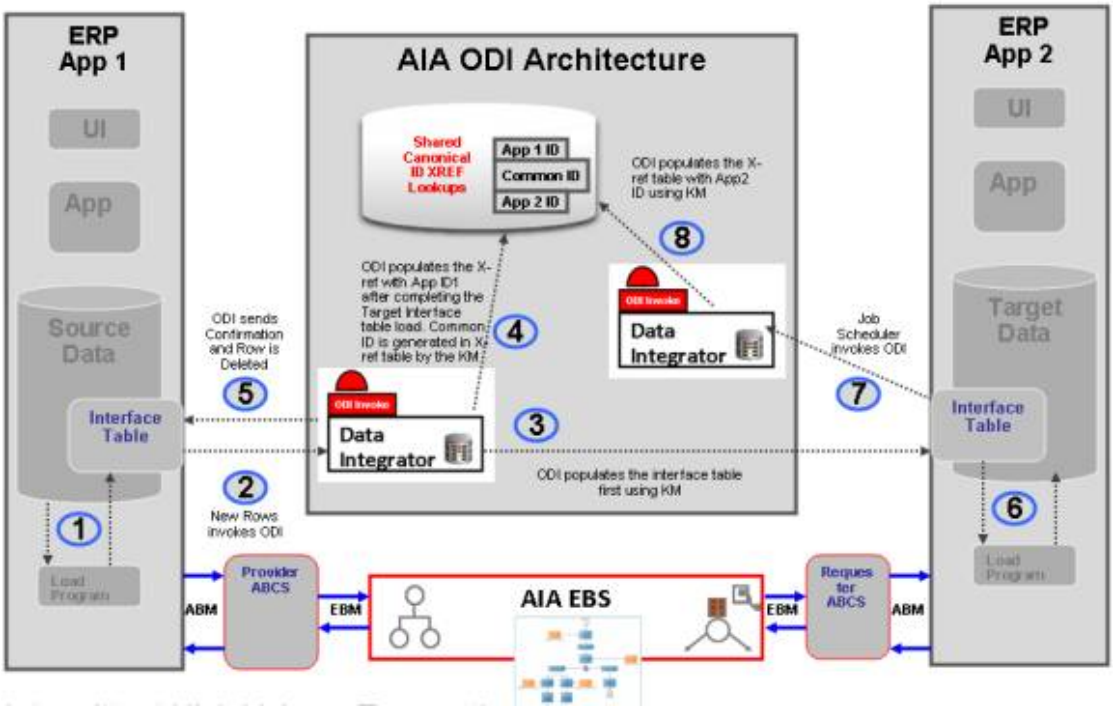
For details about how to load data, see How to Perform the Oracle Data Integrator Data Load.

AIA recommends that the interface tables on the source side have a mechanism to indicate processed rows.

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 15-2.

Figure 15-2 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 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.

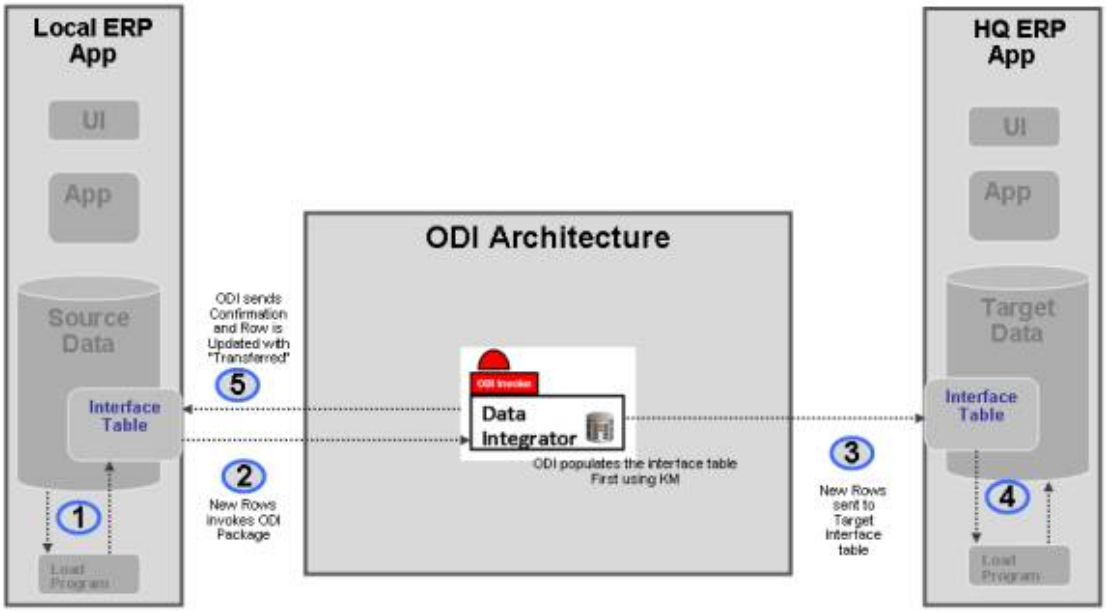
In this design pattern, do not use the AIA Oracle Data Integrator approach as an alternate route to send data when Oracle Fusion Middleware is unavailable. Instead, messages should be queued using an appropriate message queuing technology such as JMSQ, and handled using the guaranteed message technology recommended by AIA.

For more information about guaranteed messages, see Guaranteed Message Delivery.

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 15-3.

Figure 15-3 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 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.

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 runtime.

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 *Developing SOA Applications with 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.

Using the XREF Knowledge Module

In Oracle Data Integrator, the creation of XREF data is a two-step process. Each step is an interface. The overall process is illustrated in Figure 15-4.

- 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.

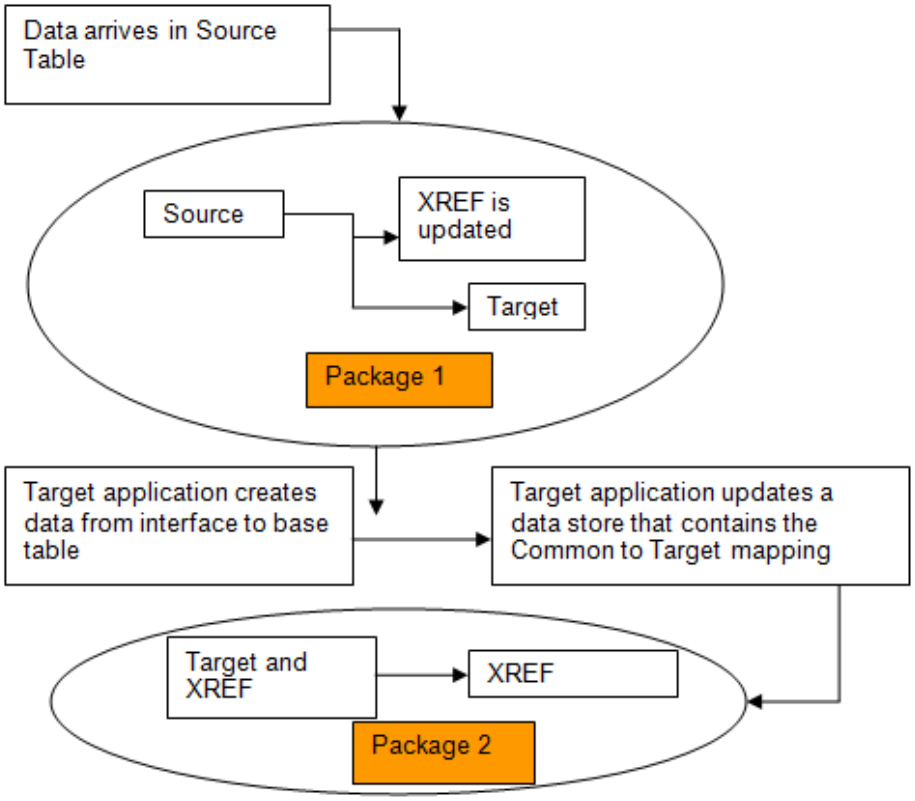
  Note:

  If 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 15-4 Using the XREF Knowledge Module



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 Working with DVMs and Cross-References.

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*Developing Integration Projects with Oracle Data Integrator.*

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 *Developing Integration Projects with 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 15-5, has the appropriate SQL to get the value from the source, depending on the implementation.

   Figure 15-5 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.

How to Create the First Interface (Source to Target)

To create the first interface:

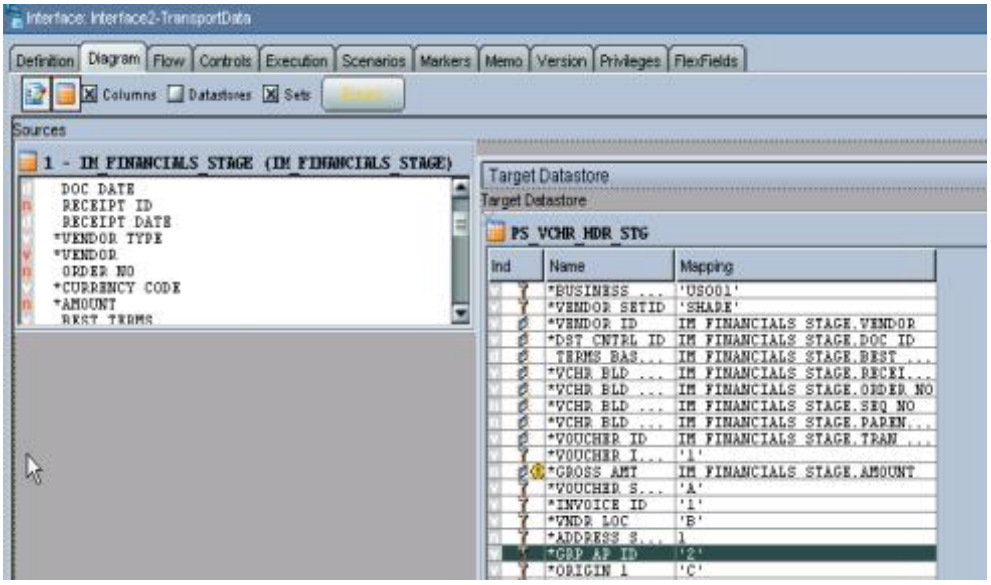
1. Create an interface.

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 15-6.

Figure 15-6 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 15-1:

Table 15-1 Required Options in Customized Knowledge Module IKM SQL to SQL

Term	Description
XREF_TABLE_NAME	Name of your XREF table.

Term	Description
XREF_COLUMN_NAME	Name of the source column in the XREF table. Enter the variable name that you defined earlier (#GetSourceColumnName) in this option.
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 runtime, 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.

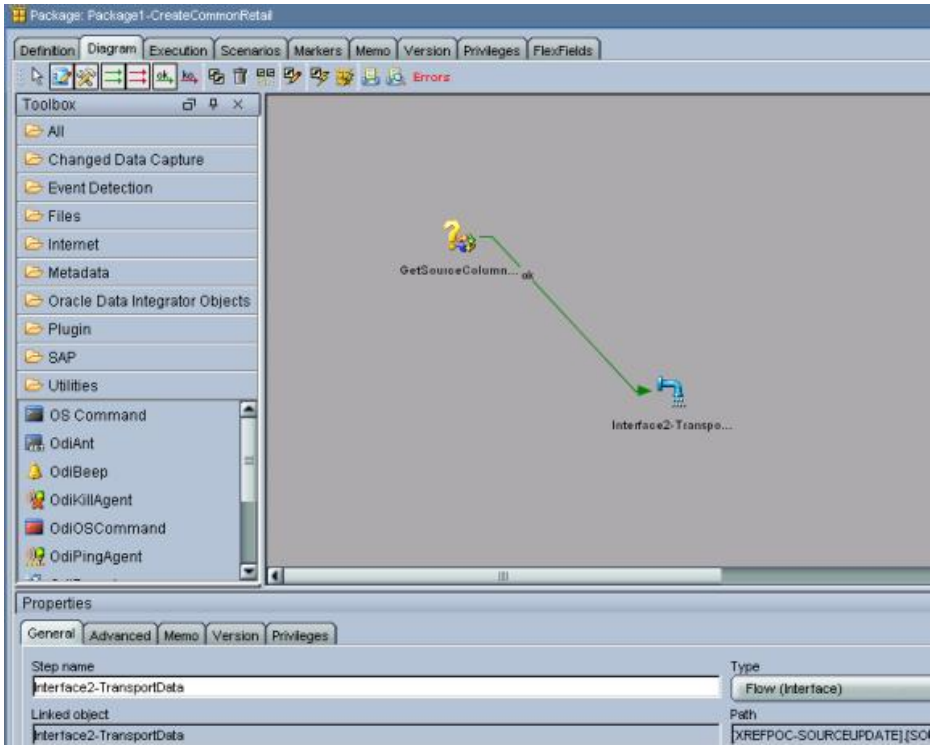
7. Validate and save the interface.

How to Create a Package for the First Interface

To create a package for the first interface:

1. Create a package to run the interfaces, as shown in Figure 15-7.

Figure 15-7 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.

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 15-1.

Note:

Construct this view for each implementation.

Example 15-1 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;
```

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.

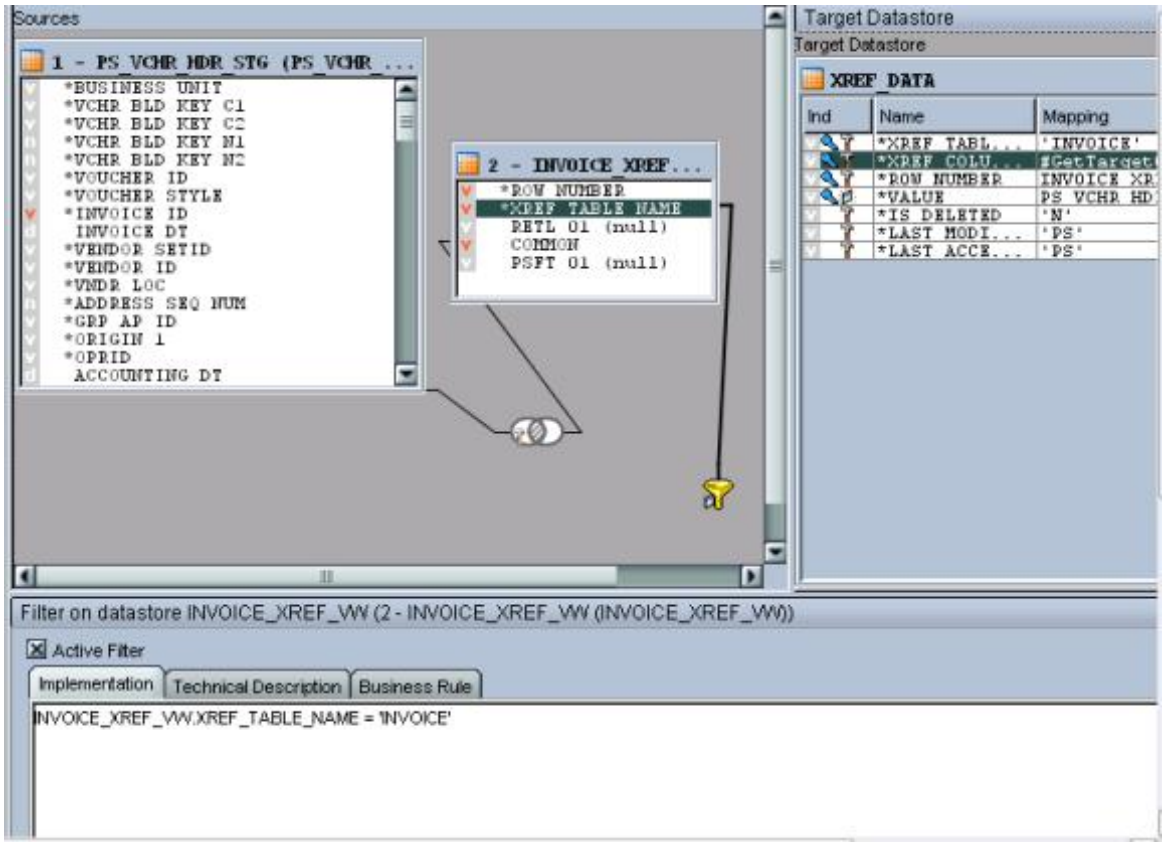
To create the second interface:

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 15-8. For example, `XREF_VW.XREF_TABLE_NAME='INVOICE'`,if you are using this for INVOICE.

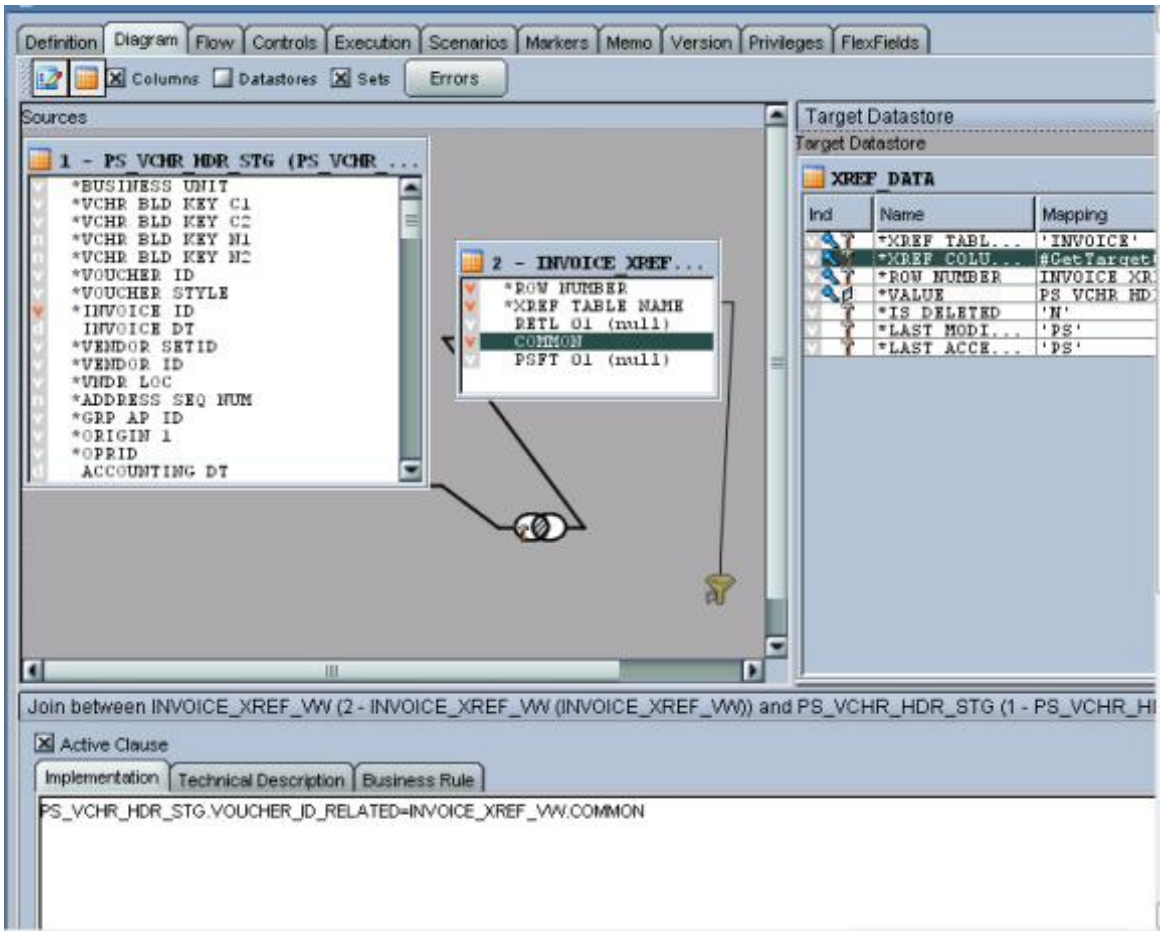
Figure 15-8 Filter for XREF_VW with a WHERE Clause to Filter Data from Your Table Name Only



3. Join the mapping data store and XREF_VW with the columns that store the common (or source) identifier, as shown in Figure 15-9.

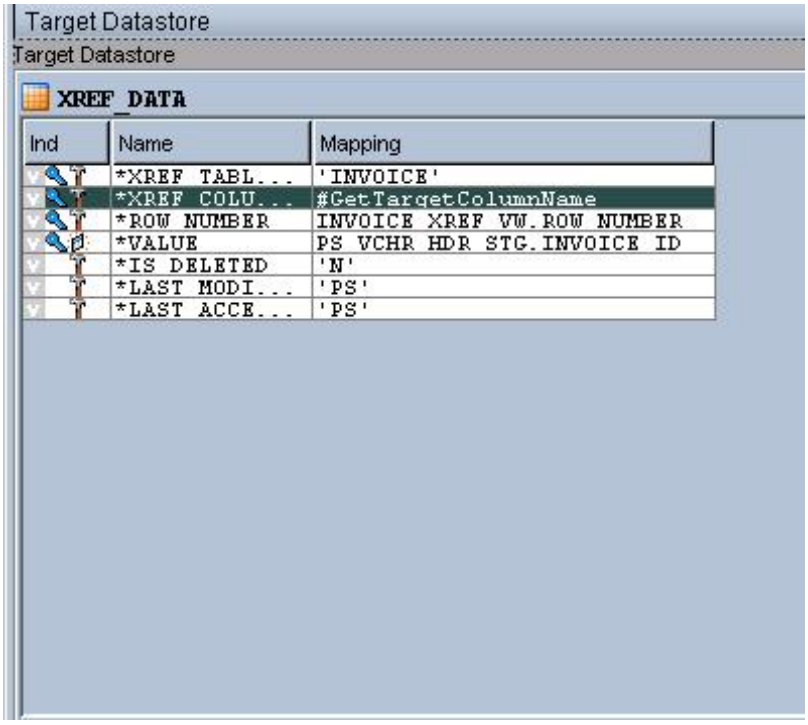
   In this example, the column of the PeopleSoft interface table that stores common data is `VOUCHER_ID_RELATED`.

Figure 15-9 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 15-10.

Figure 15-10 XREF_TABLE_NAME, XREF_COLUMN_NAME and VALUE Marked as Keys



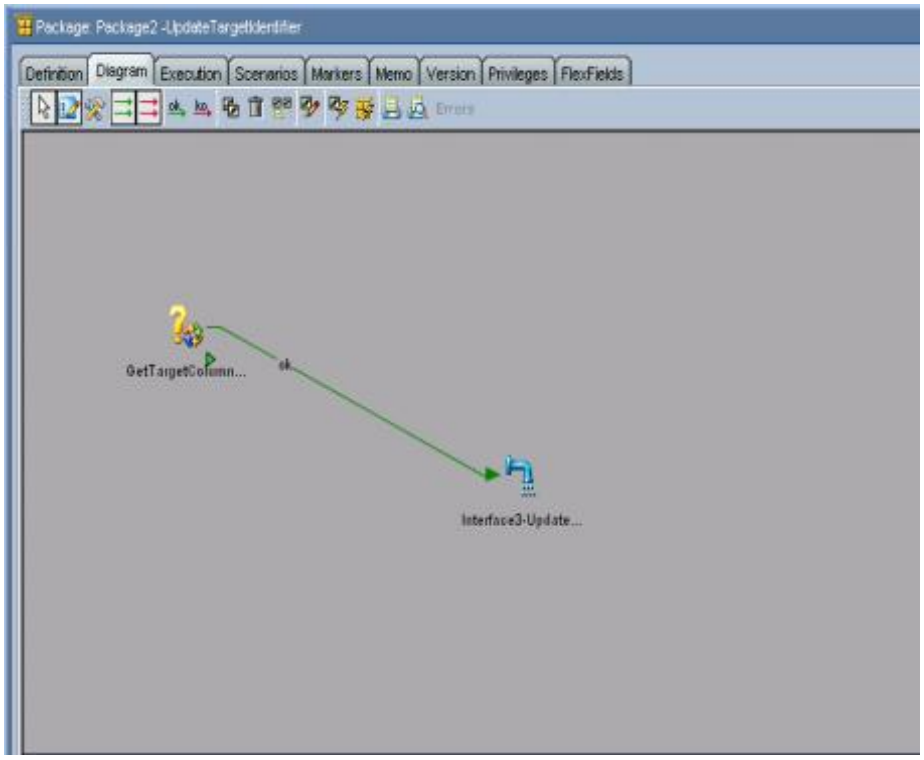
11. On the Flow tab, use the load knowledge module, `LKM SQL to Oracle`.
12. Use the integration knowledge module, `IKM Oracle incremental update`.
13. On the `Controls` tab, use the check knowledge module, `CKM Oracle`.
14. Validate and save the interface.

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 15-11.

Figure 15-11 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.

Working with Domain Value Maps

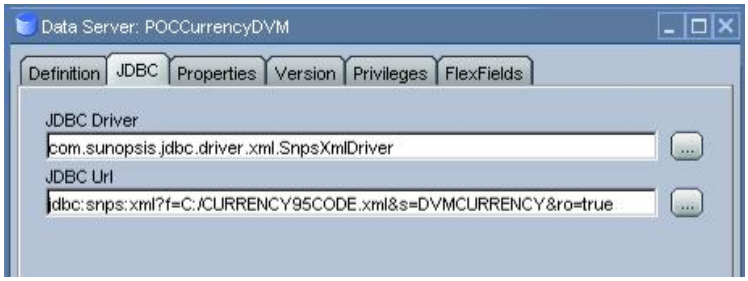
The Domain Value Maps (DVM) are available as XML files and can be used as delivered.

To use the DVM:

1. Reverse-engineer the DVM and it results in multiple relational tables.

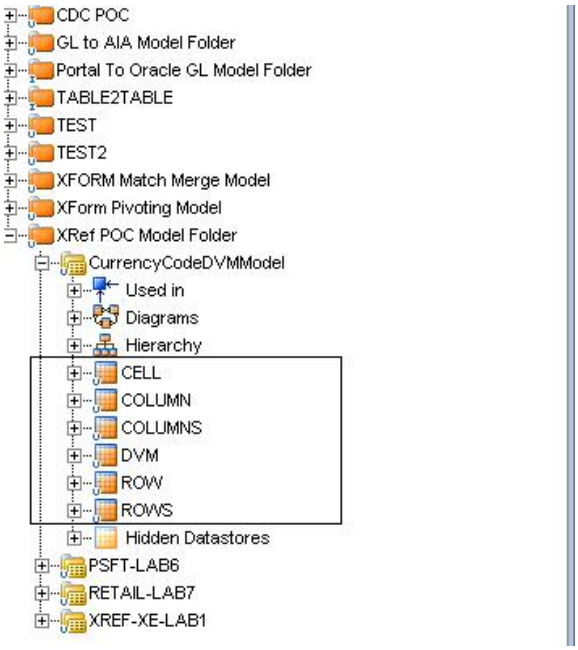
   Here is how the DVM XML is converted after reverse-engineering. We have used the XML itself in the JDBC description for the data server and not any schema for reverse engineering, as shown in Figure 15-12.

Figure 15-12 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 15-13.

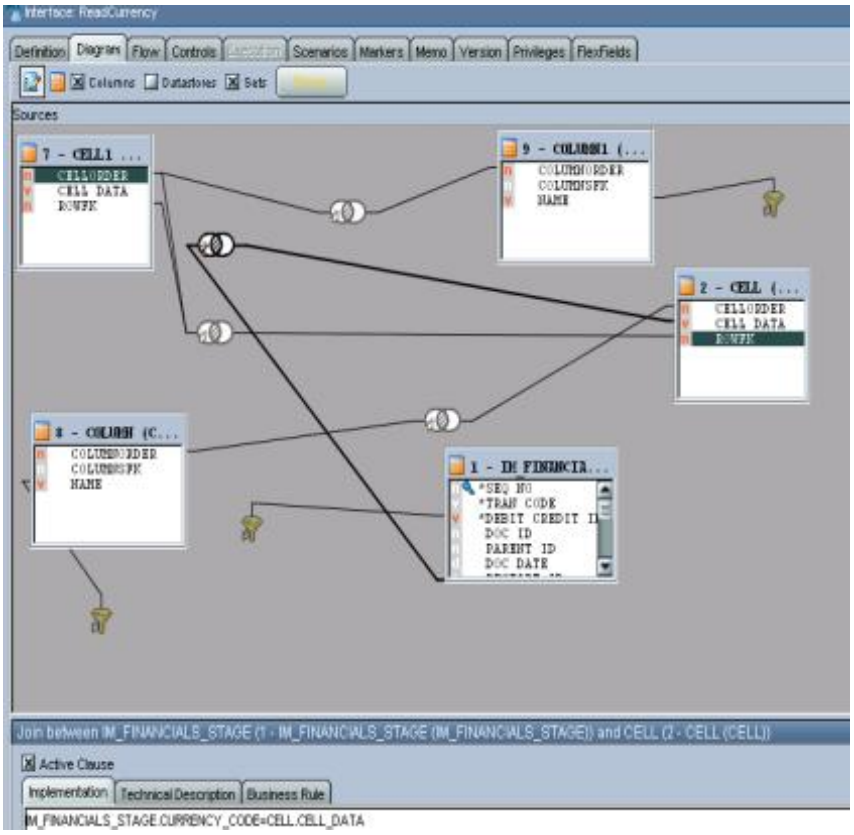
Figure 15-13 DVM XML Results in Six Tables Following Reverse-Engineering



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 15-14.

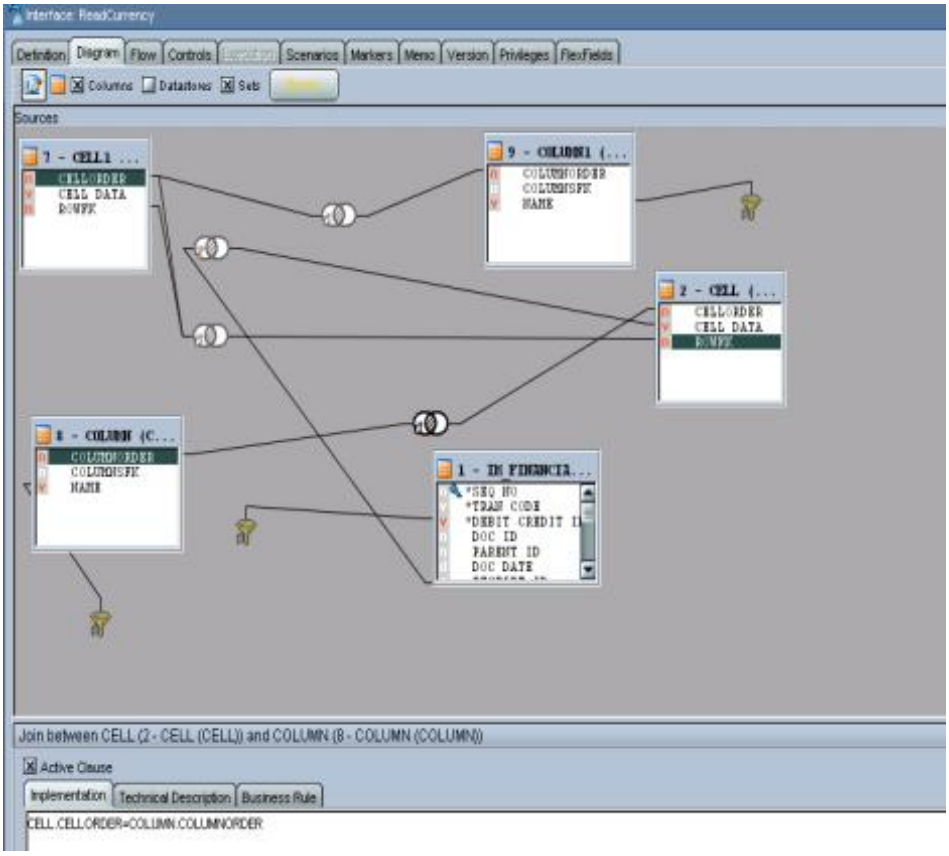
   Use the column in your main source table for DVM in the join.

Figure 15-14 Join of IM_FINANCIALS_STAGE with the CELL Table



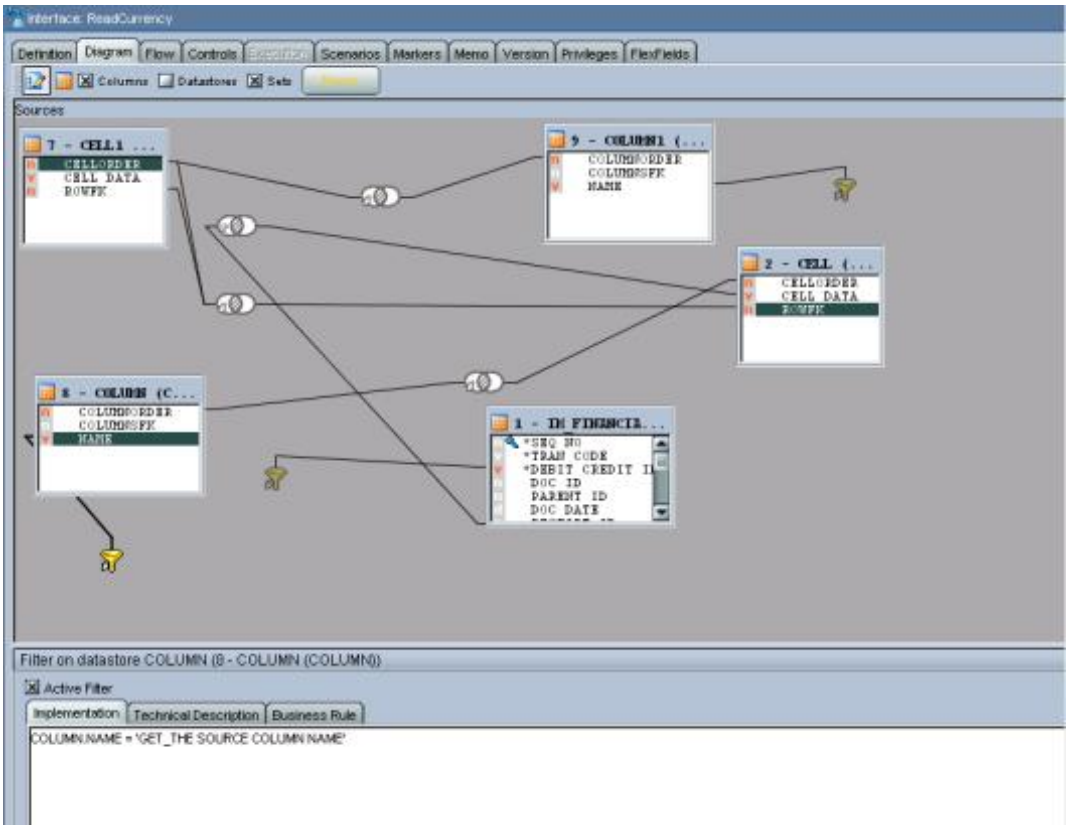
4. Join the CELL table with COLUMN table, as shown in Figure 15-15.

Figure 15-15 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 15-16. This filter can use a variable that holds the source column name.

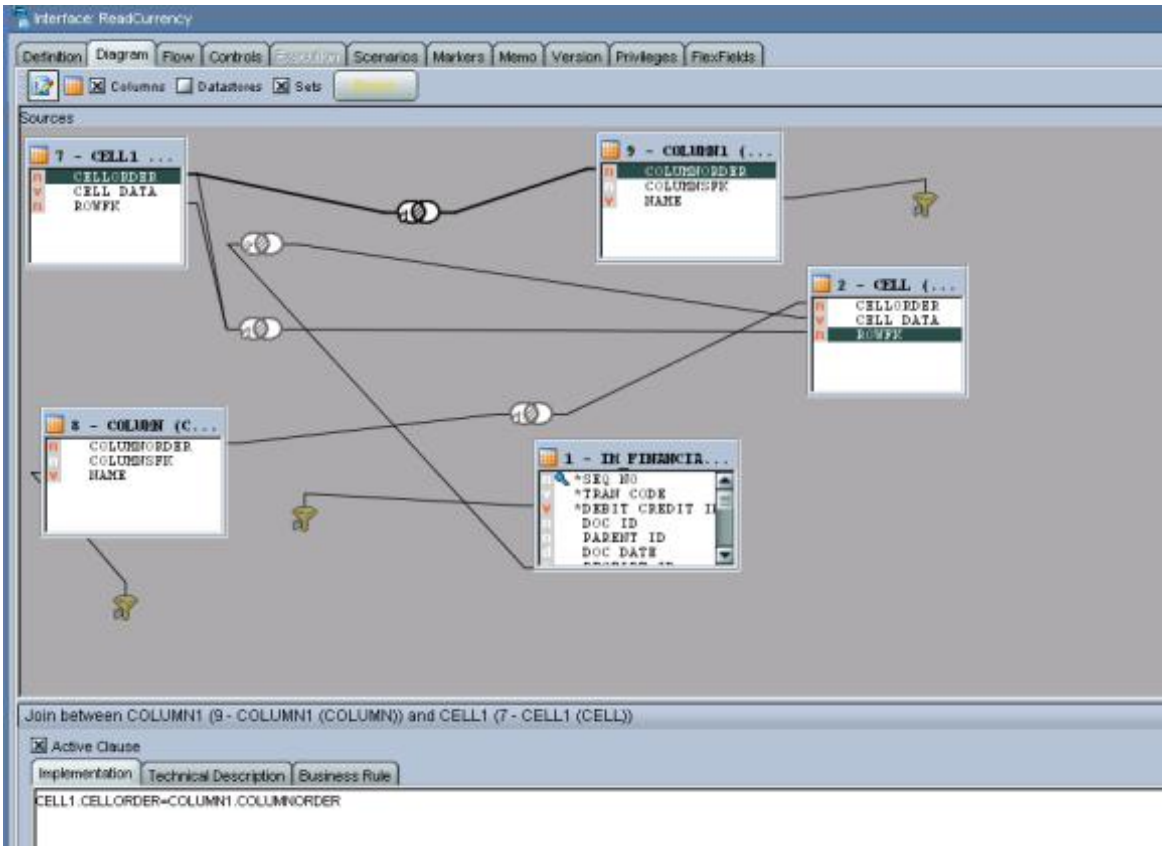
Figure 15-16 Filter Added to the COLUMN Table



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 15-17.

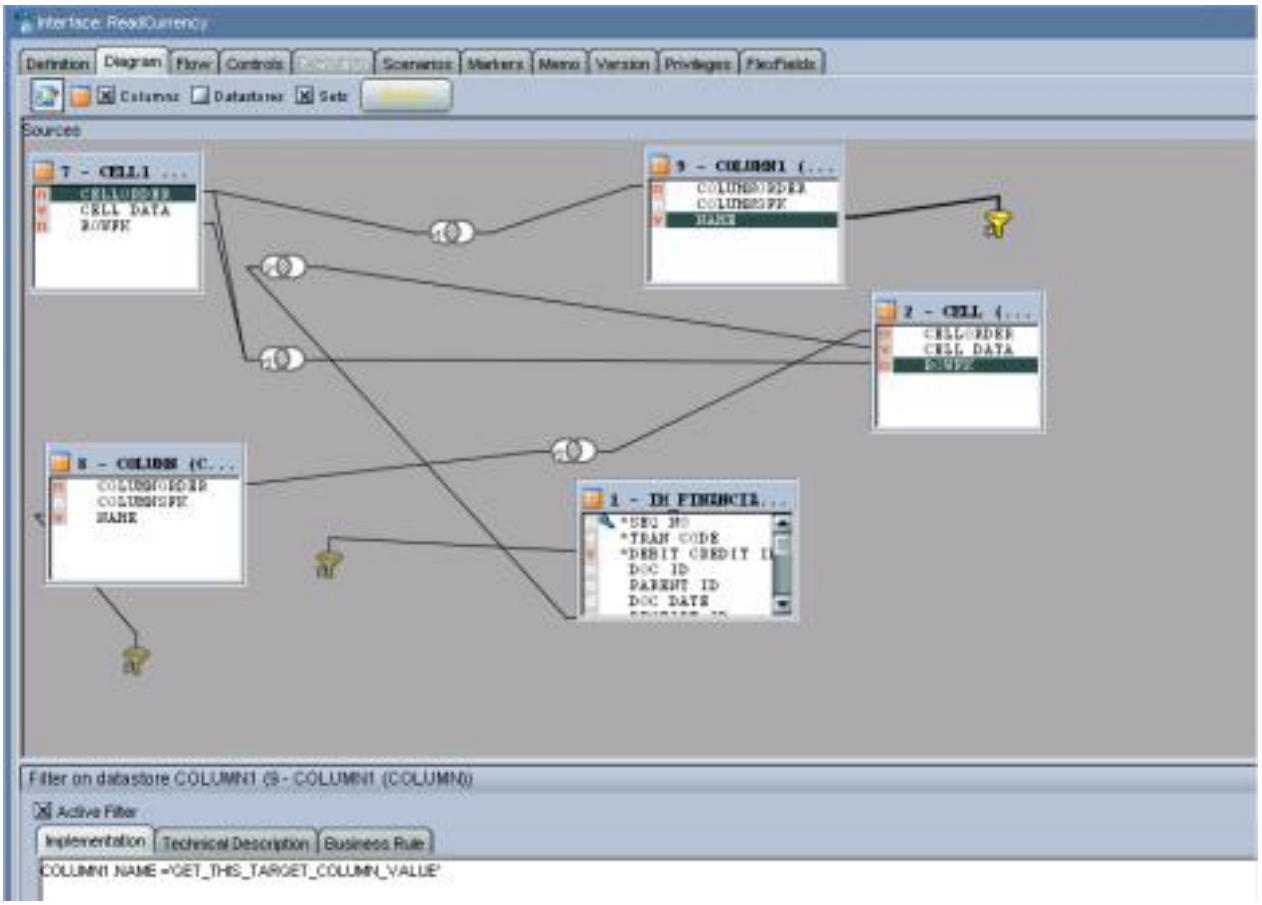
Figure 15-17 Join of CELL1 with the COLUMN1 Table



8. Add a filter on the COLUMN1 table (to determine the target column name), as shown in Figure 15-18.

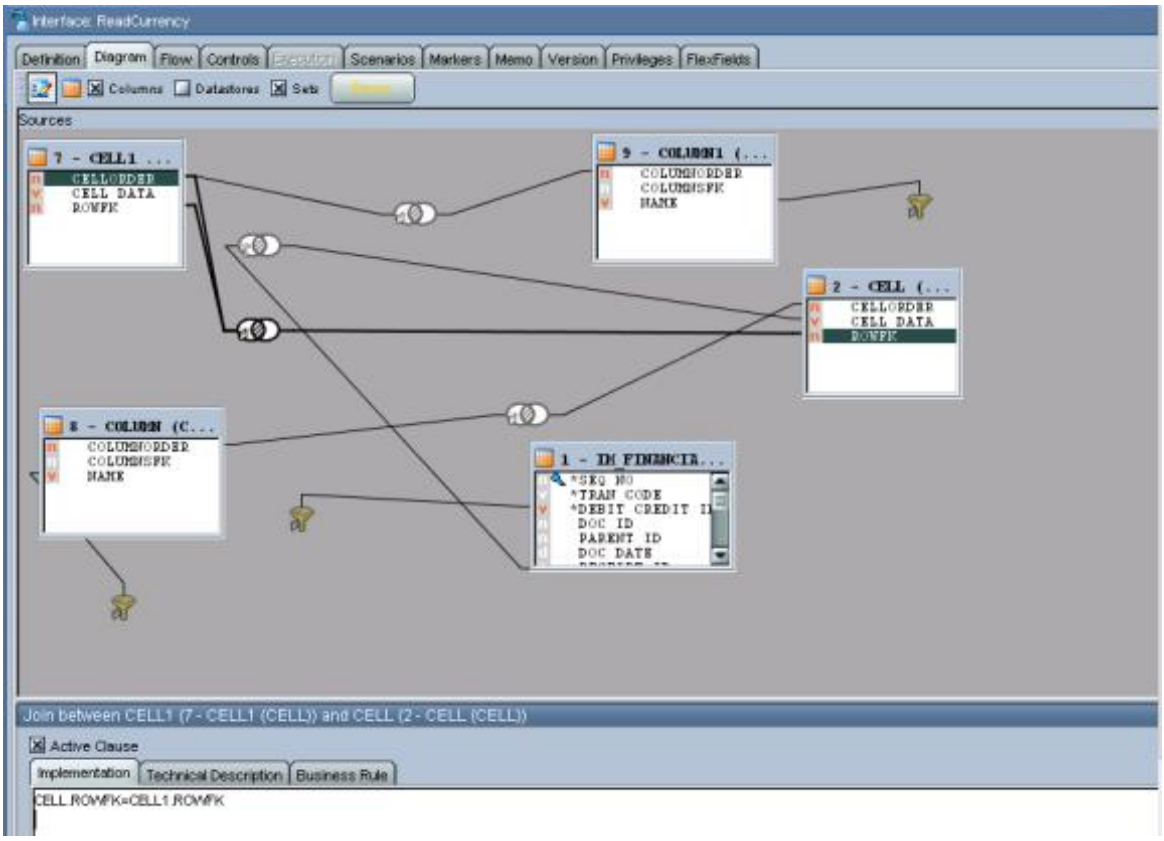
   This filter can use a variable that holds the source column name.

Figure 15-18 Filter Added on the COLUMN1 Table



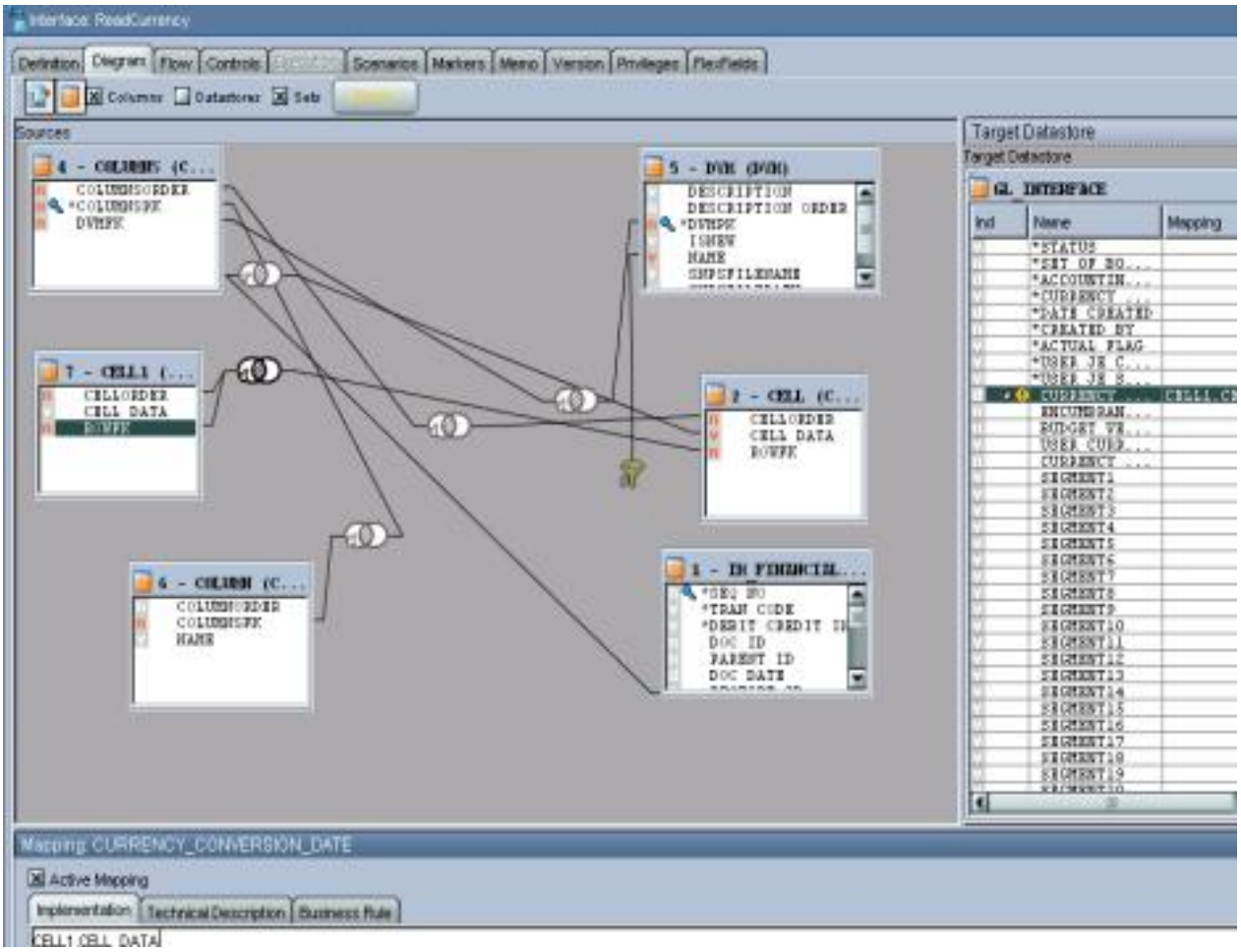
9. Finally, self-join the CELL table with another CELL (CELL1) table, as shown in Figure 15-19.

Figure 15-19 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 15-20.

Figure 15-20 Second CELL Data in the Target Interface Mapping



Note:

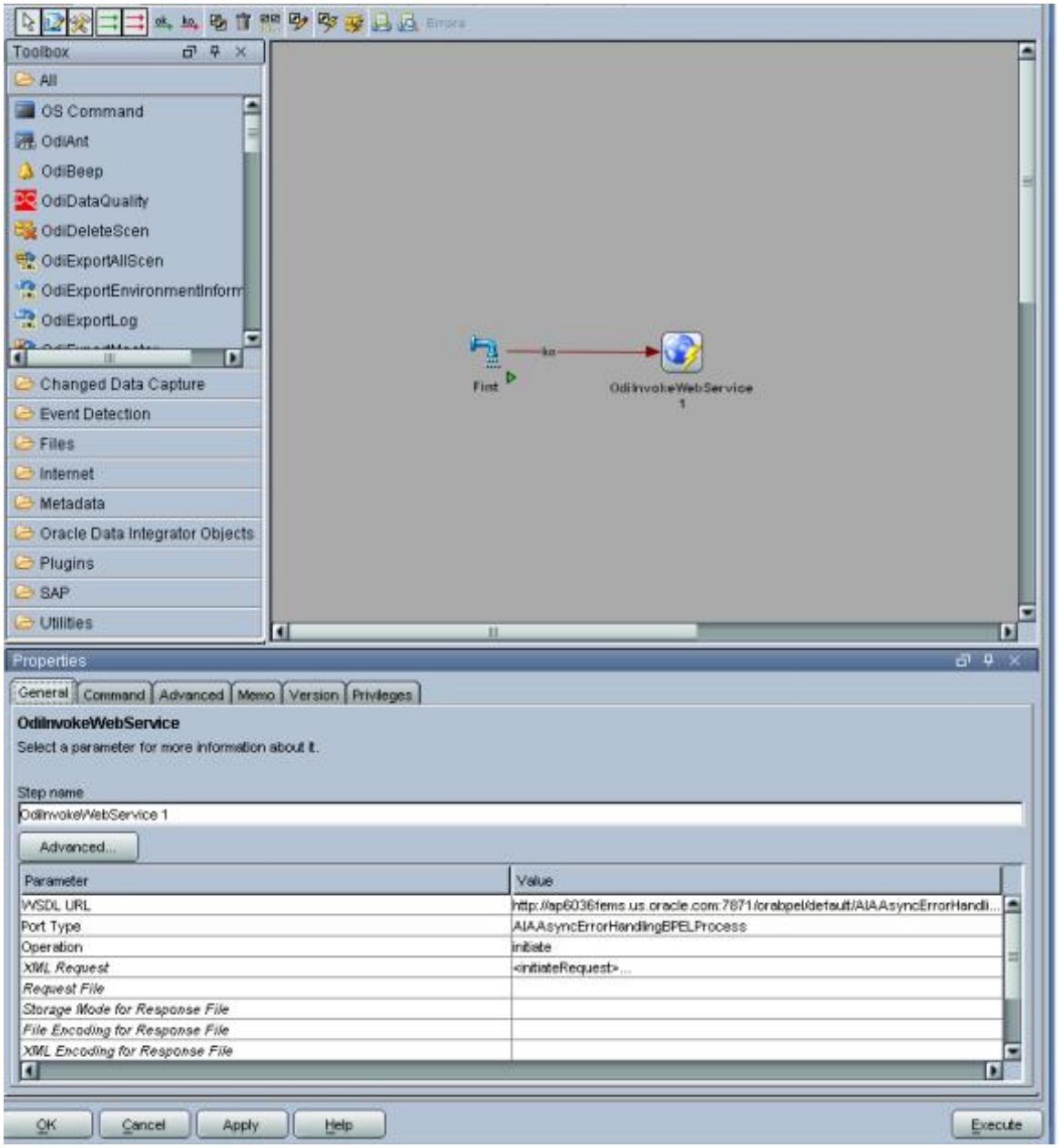
You must repeat all joins for each DVM-centric column.

Using Error Handling

To use error handling for the Oracle Data Integrator flows:

The package shown in Figure 15-21 is a sample interface that invokes the **AIAAsyncErrorHandlingBPELProcess** when it ends in error.

Figure 15-21 Sample Interface that Invokes AIAAsyncErrorHandlingBPELProcess When it Ends in Error



The XML Request Input for this process is provided in Example 15-2.

Example 15-2 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>
```

Oracle Data Integrator Ref Functions

Table 15-2 lists the Oracle Data Integrator Ref functions with a description of each.

Table 15-2 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.

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 (`http://www.oracle.com/technetwork/middleware/ias/downloads/utilsoft-090603.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/technetwork/database/enterprise-edition/jdbc9201-092698.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 15-3.

   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 15-22.

   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:

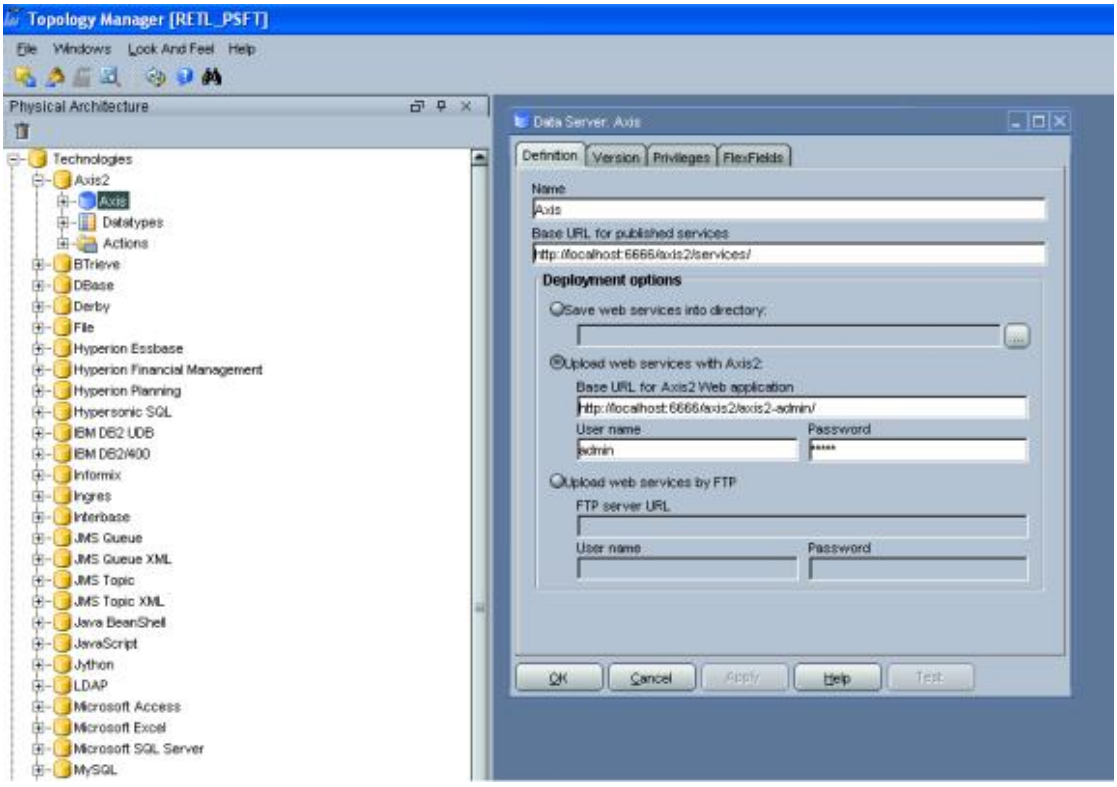
- **Name**: Name of the dataserver as it appears in Oracle Data Integrator.
- **Base URL for published services**: `http://<Host>:<HTTP port>/axis2/services`.
- Select the option corresponding to the chosen method of deployment.
- **Web Service Upload**: Specify the root URL for the Axis2 application, 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 15-22 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 15-23.

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.

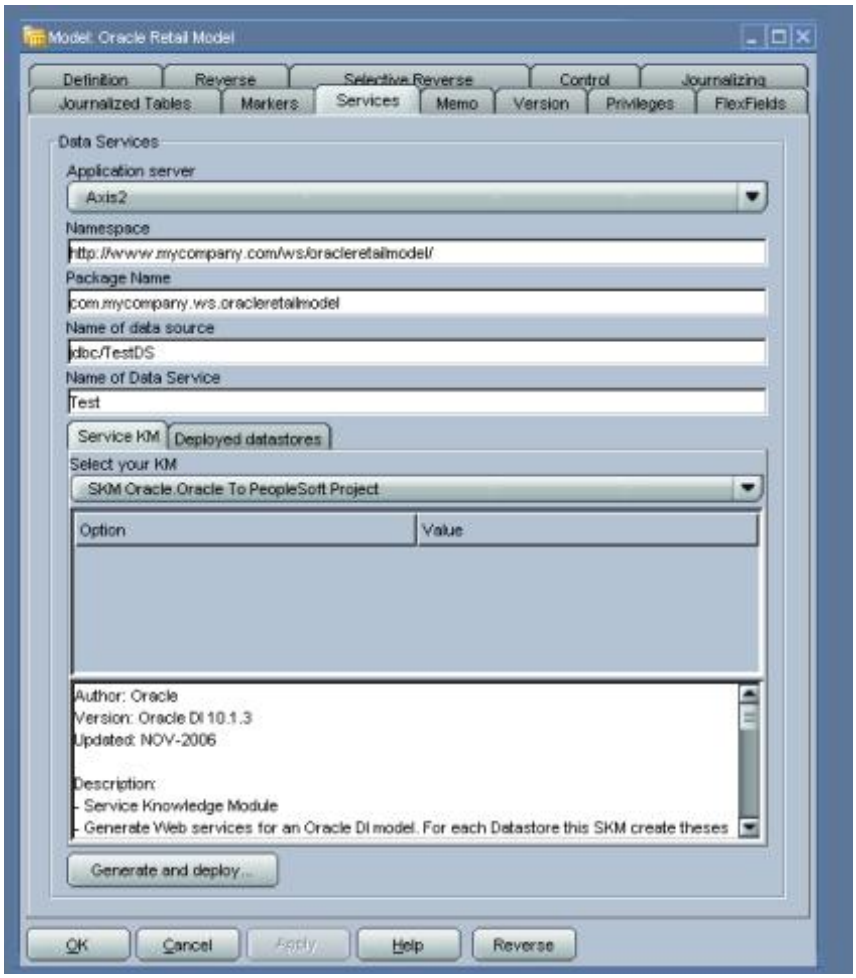
j. Click **OK** to save your changes.

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 15-23 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 15-4 is a sample SOAP request for this web service.

   The scenario execution returns a SOAP response as shown in Example 15-5.

Example 15-3 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>
```

Example 15-4 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>
```

Example 15-5 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>
```

16

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:

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.

Connecting Applications to Implement Business Processes

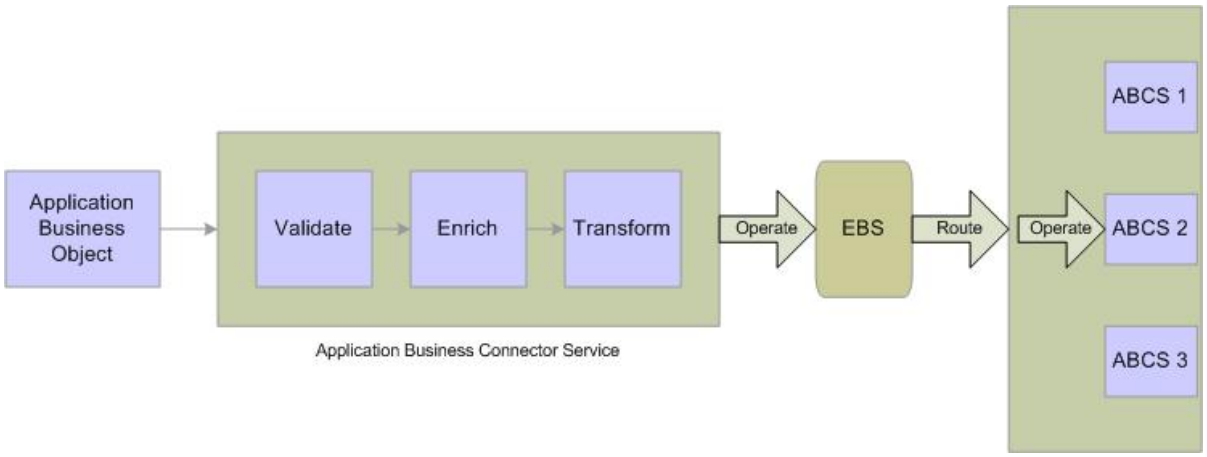
Oracle Application Integration Architecture (AIA) leverages canonical patterns and direct patterns to connect applications for implementing business processes.

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 16-1 illustrates how a canonical pattern is implemented in AIA.

Figure 16-1 Canonical Pattern Implemented in AIA



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 Using Oracle Data Integrator for Bulk Processing.

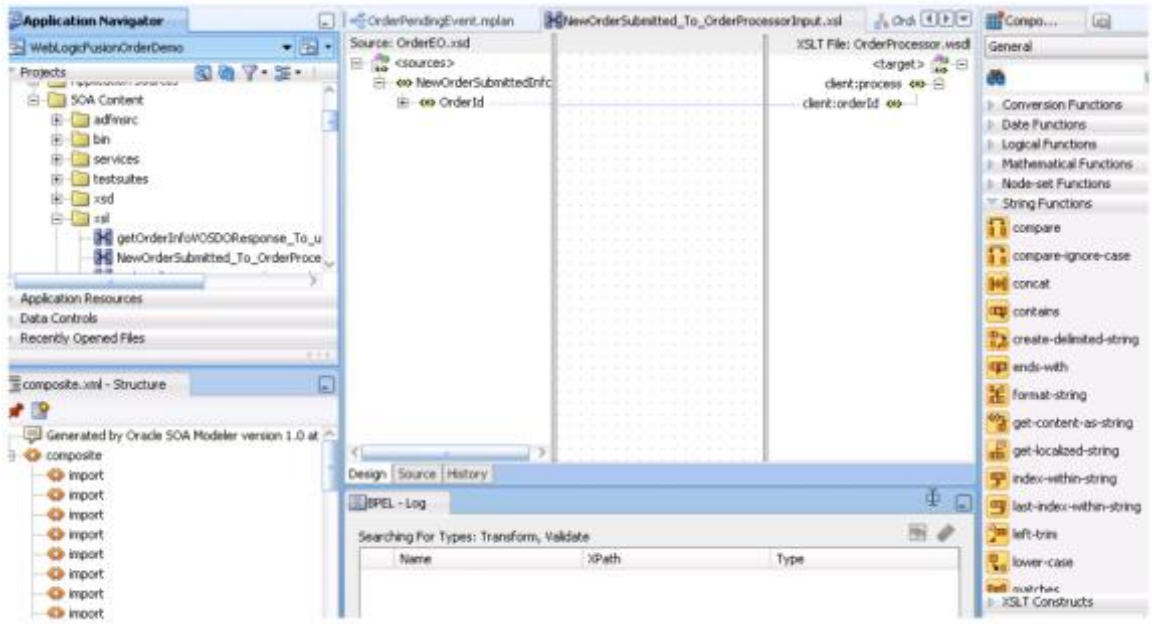
Using Tools and Technologies to Perform Message Transformations

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 16-2 shows the layout of the XSLT Mapper.

Figure 16-2 Layout of the XSLT Mapper



For more information about the XSLT Mapper, see "Creating Transformations with the XSLT Mapper" in *Developing SOA Applications with Oracle SOA Suite.*

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.

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 How to Work with AIAConfigurationProperties.xml in $SOA_HOME/aia_instances/$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 16-1 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 16-1 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>
```

Handling Missing or Empty Elements

Transformation logic and xpath should be designed with the possibility of missing or empty elements, as shown in Example 16-2.

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.

Example 16-2 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>
```

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 16-3, and the source node does not exist in the input document, then an empty node is created in the target document.

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 16-4.

Example 16-3 Statement Without xsl:If

```
<portal:PHONE>
           <xsl:value-of
select="corecom:Contact/corecom:ContactPhoneCommunication[1]/corecom:
```

```
PhoneCommunication/corecom: WorkContactNumber"/>
          </portal:PHONE>
```

Example 16-4 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>
```

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 16-5. No hard-coded system IDs should exist in XSL Scripts and ABCS.

The recommended approach is to load the system IDs dynamically, as shown in Example 16-6.

Example 16-5 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>
```

Example 16-6 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>
```

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.

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.

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`

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 Introducing EBM Header Concepts.

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 16-7.

Example 16-7 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>
```

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 16-8 shows the industry extensible transformation template.

Example 16-8 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>
```

Working with DVMs and Cross-References

This section discusses DVMs, cross-references, and naming standards, and when to use them.

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 runtime, 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 SOA Core Extension.

For more information about naming standards, see Oracle AIA Naming Standards for AIA Development.

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.

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 16-1 lists a cross referencing configuration implemented in the Product synchronization scenario between the Portal BRM and Siebel CRM systems

Table 16-1 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.

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 Oracle AIA Naming Standards for AIA Development.

For more information about cross-references, see "Working with Cross References" in *Developing SOA Applications with Oracle SOA Suite.*

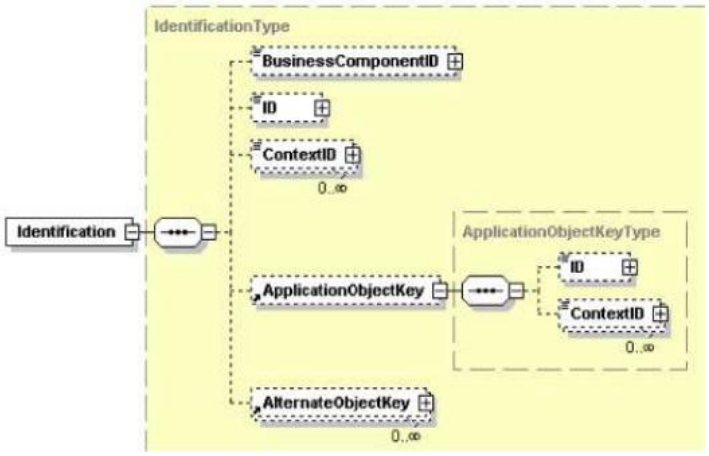
Mapping and Populating the Identification Type

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 16-3 illustrates the schema definition for Identification Type.

Figure 16-3 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 16-2 describes each of the identifiers:

Table 16-2 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 16-3 describes the purpose of each of the attributes.

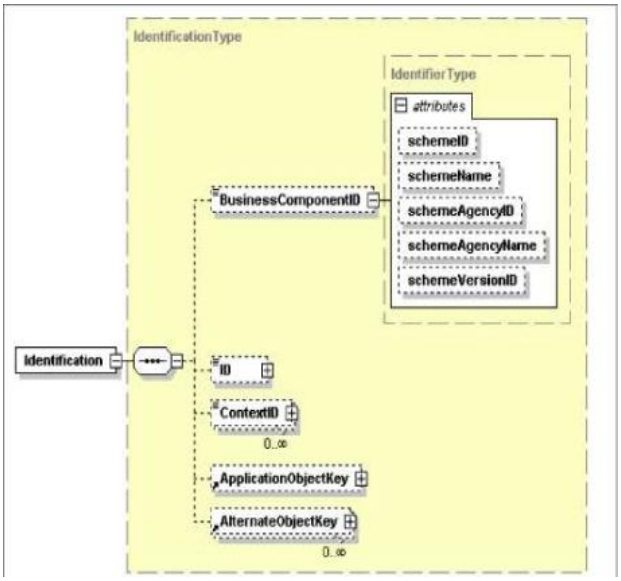
Table 16-3 Key Context Attributes

Name	Type	Description
BusinessComponentID/ID ApplicationObjectKey/ID AlternateObjectKey/ID	Element	The element is used to store the actual object ID.

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 16-4 illustrates the Business Component ID:

Figure 16-4 Structure of the Business Component ID



Example 16-9 provides an example of how the BusinessComponentID is populated in the EBM.

Example 16-9 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>
```

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.

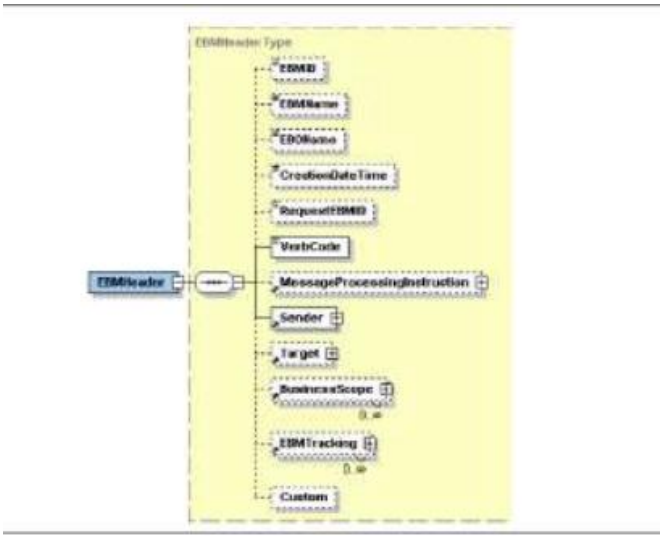
Introducing EBM Header Concepts

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 16-5 illustrates the components of an EBM header:

Figure 16-5 EBM Header Components



The following sections provide details about each of the components.

Standard Elements

This section discusses the standard elements in a message header.

EBMID

This element contains a GUID to uniquely identify the EBM, as shown in Example 16-10. The GUID is generated using a provided standard XPath function.

XPath function to generate GUID: `orcl:generate-guid()`

Example 16-10 EBMID

```
<EBMHeader>
<EBMID>33303331343032313534393138393830</EBMID>
. . .
</EBMHeader>
```

EBOName

This element contains the fully qualified name of the EBO in the notation: `{namespaceURI}localpart`, as shown in Example 16-11.

Example 16-11 EBO Name

```
<EBMHeader>
<EBOName>{http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V1}Query
Invoice</EBOName>
. . .
</EBMHeader>
```

RequestEBMID

This element contains the originating request GUID that uniquely identifies the originating request ID, as shown in Example 16-12. The EBS populates this field in the response message by extracting it from the request message.

Example 16-12 RequestEBMID

```
<EBMHeader>
<RequestEBMID>33303331343032313534393138393830</RequestEBMID>
. . .
</EBMHeader>
```

CreationDateTime

This element contains a timestamp that reflects when the EBM was created, as shown in Example 16-13. 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 16-13 CreationDateTime

```
<EBMHeader>
<CreationDateTime>200 7-04-2 7T12:22:17-08:00</CreationDateTime>
. . .
</EBMHeader>
```

VerbCode

This element contains the verb represented by the EBM, as shown in Example 16-14.

Example 16-14 VerbCode

```
<EBMHeader>
<VerbCode >Query</VerbCode>
. . .
</EBMHeader>
```

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`.

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.

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 16-15.

2. These fields are currently populated by the CAVS.

   Usually, the fields are populated in the SOAP Header before initiating a testing request.

Example 16-15 Message Processing Instruction Population

```
<corecom:MessageProcessingInstruction>
    <corecom:EnvironmentCode>
 <xsl:text disable-output-escaping="no">Production</xsl:text>
    </corecom:EnvironmentCode >
</corecom:MessageProcessingInstruction>
```

Sender

The Sender contains information about the originating application system, as shown in Figure 16-6.

Figure 16-6 Structure of the Sender Element

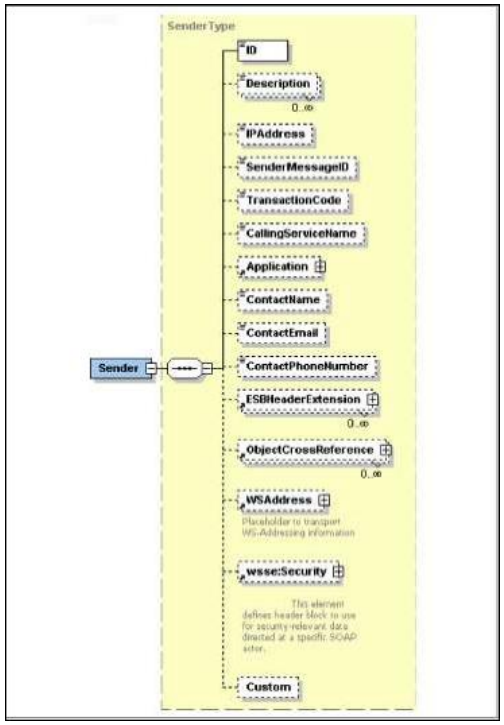


Table 16-4 provides details about each element in the Sender element.

Table 16-4 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 16-16 to populate this value.
Description	This is the long description of the Sender System. ReqABCSImpl should call the API detailed in Example 16-16 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.

Element	Description
CallingServiceName	Name for the calling ABCS. Populated by source ABCS.
Application	Holds information about the originating application.
Application/ID	The sender system can designate the originating application code in this element. ReqABCSImpl should call the API detailed in Example 16-16 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 16-16 to populate this value.

Example 16-16 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>
```

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.

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.

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.

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.

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 16-16 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.

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.

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.

ESBHeaderExtension

This element, shown in Figure 16-7, 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 16-7 Structure of the ESBHeaderExtension Element

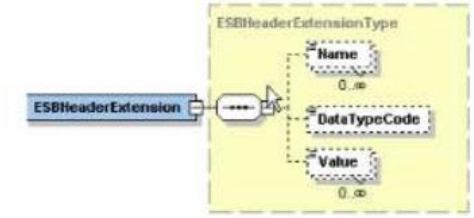


Table 16-5 describes the elements in the ESBHeaderExtension element.

Table 16-5 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 16-8 illustrates the structure of the ESBHeaderExtension element.

Figure 16-8 Structure of the ESBHeaderExtension Element

ObjectCrossReference

This component stores the identifier information of the sender objects and the corresponding cross-reference identifier, as shown in Figure 16-9. 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 16-9 Structure of the ObjectCrossReference Element

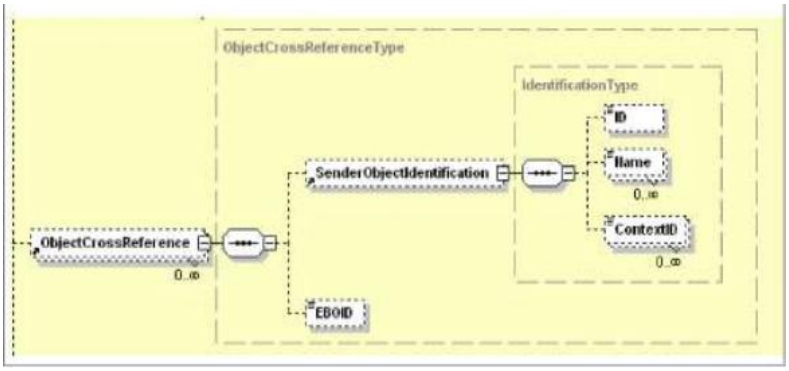


Table 16-6 describes the elements in the ObjectCrossReference element.

Table 16-6 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.

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.

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.

Custom

This element is the complex type that you can extend to add your own elements.

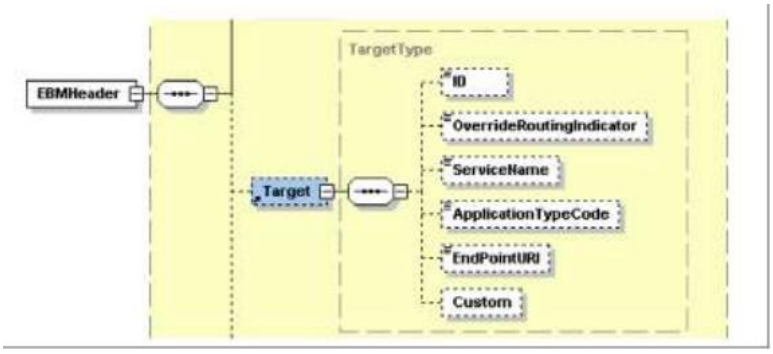
For more information about specific usage of this element, see Implementing the Fire-and-Forget Message Exchange Pattern.

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 16-10.

Figure 16-10 Structure of the Target Element



ID

Use this element, shown in Example 16-17, 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.

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 16-17, 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.

Custom

This is the complex type that you can extend to add your own elements when needed.

Example 16-17 shows an example of target element code.

Example 16-17 Target Element Code Sample

```
<corecom:Target>
<corecom:ID>PORTAL_01</corecom:ID>
<corecom:ApplicationTypeCode>PORTAL_01</corecom:ApplicationTypeCode>
</corecom:Target>
```

BusinessScope

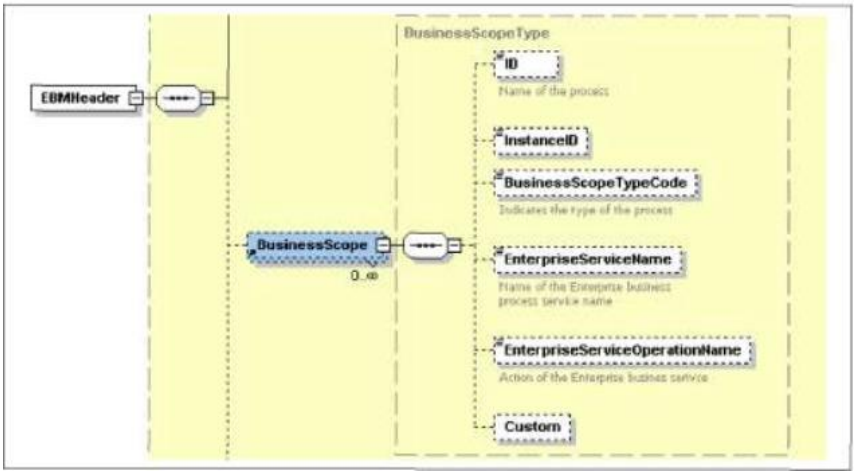
This section captures business scope specification related information defined in UN/CEFACT Standard business definition header.

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 16-11 describes how this works.

Figure 16-11 Structure of the BusinessScope Element



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.

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.

BusinessScopeTypeCode

This element indicates the kind of business scope. Values are:

- `BusinessScope` (UMM)
- `BusinessService` (for ebXML)
- `Message`

ReqABCSImpl populates this value.

EnterpriseServiceName

Name of the EBS where this message belongs. Known to the message creator, be it ABCS or EBS. ReqABCSImpl populates this value.

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.

Custom

A complex type for you to extend to add extra elements.

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.

Use Case: Request-Response

GetAccountBalance: Sends a request message with account number as input and receives account balance as response, as shown in Figure 16-12.

Figure 16-12 Use Case: Request-Response

Request EBM

Example 16-18 shows two rows: one for the Process and one for the Request EBM Message involved in the process.

Example 16-18 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>
```

Response EBM

Example 16-19 shows two rows: one for the process and one for the Response EBM Message involved in the process.

Example 16-19 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>
```

Use Case: Asynchronous Process

SyncProduct: Portal sends an async message to Siebel to sync the product details, as shown in Figure 16-13.

Figure 16-13 Use Case: Asynchronous Process

Request EBM

Example 16-20 shows two rows: one for the process and one for the Request EBM Message involved in the process.

Example 16-20 Request EBM for Request-Response 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
```

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 16-14.

Figure 16-14 Synchronous Process with Spawning Child Processes

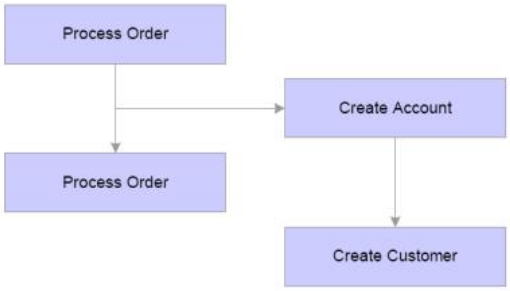
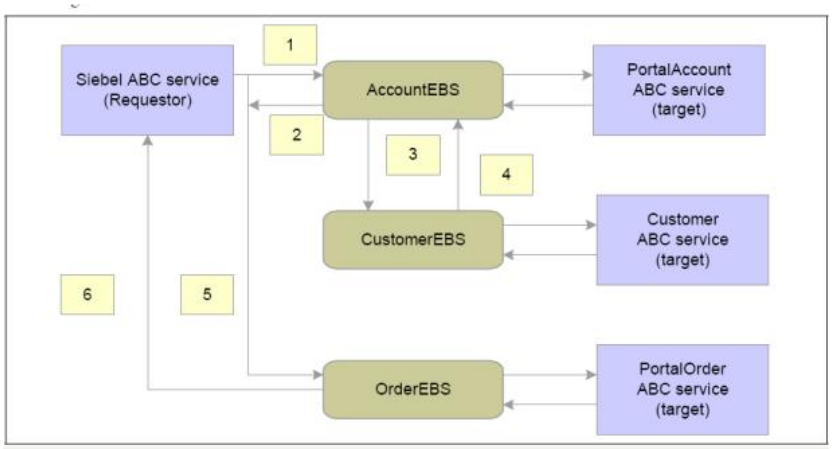


Figure 16-15 illustrates the ProcessOrder flow.

Figure 16-15 ProcessOrder flow



Message 1: Create Account Request EBM

Example 16-21 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)

Message 2: Create Account Response EBM

Example 16-22 shows two rows:

- One for the process
- One for the Create Account Response Message in the process

Message 3: Create Customer Request EBM

Example 16-23 shows two rows:

- One for the process
- One for the Create Customer request message in the process

Message 4: Create Customer Response EBM

Example 16-24 shows two rows:

- One for the process
- One for the Create Customer response message in the process

Message 5: ProcessOrder Request EBM

Example 16-25 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)

Message 6: Process Order Response EBM

Example 16-26 shows two rows:

- One for the process
- One for the Process Order Response in the process

Example 16-21 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>
```

Example 16-22 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>
```

Example 16-23 Create Account Response 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>
```

Example 16-24 Create Customer Request 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>
```

Example 16-25 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>
```

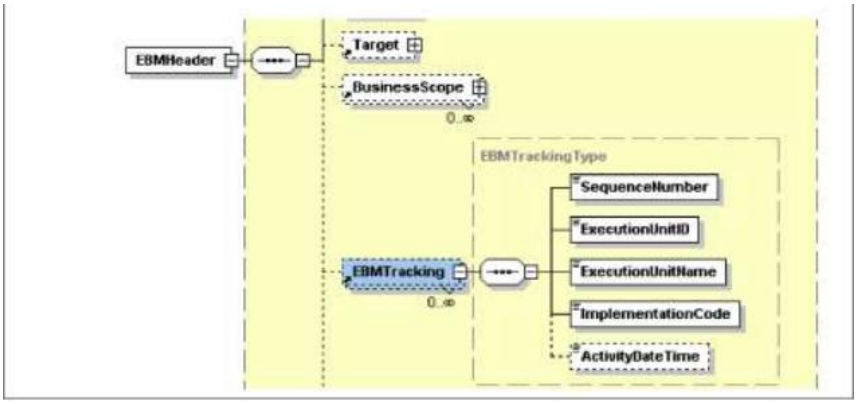
Example 16-26 Process Order Response EBM

```
<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>
```

EBMTracking

EBMTracking contains tracking information about each node that the EBM has been through, a shown in Figure 16-16. EBMTracking may appear multiple times, once for each execution unit it passes through.

Figure 16-16 Structure of the EBMTracking element



SequenceNumber

This element contains the sequence number of the node the EBM has been through.

ExecutionUnitID

This element contains the ID of the execution unit, node, or process ID.

ExecutionUnitName

This element contains the fully qualified name of the execution unit, node, or process ID.

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.

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)?`

When to Populate EBM Tracking Information

Add an EBMTracking entry, as shown in Example 16-27, 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 16-27 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>
```

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.

17

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 Introduction to B2B Integration Using AIA.

Overview of Oracle BPEL and Mediator Process Error Handling

This section includes the following topics:

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.

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 Guidelines for Configuring Mediator for Handling Business Faults.

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, Release 11.1.1.9.*

Enabling AIA Processes for Fault Handling

This section includes the following topics:

- What You Must Know About Fault Policy Files
- How to Implement Fault Handling in BPEL Processes

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 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.

SOA Core Extension 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 17-1 added to it.

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.

For more information, see "Schema Definition File for Fault-policies.xml" and "Schema Definition File for Fault-bindings.xml" in *Developing SOA Applications with Oracle SOA Suite.*

Example 17-1 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>
```

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 17-2.

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).

In the fault-bindings.xml file, the association is done as shown in Example 17-3.

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.

Example 17-2 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>
```

Example 17-3 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>
```

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 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 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 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 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 Guidelines for BPEL Catch and Catch-All Blocks.

Implementing Error Handling for the Synchronous Message Exchange Pattern

This section includes the following topics:

- Guidelines for Defining Fault Policies
- Guidelines for BPEL Catch and Catch-All Blocks in Synchronous Request-Response
- Guidelines for Configuring Mediator for Handling Business Faults

Guidelines for Defining Fault Policies

This section includes the following topics:

- Defining a Fault Policy XML File for Handling Run-time Faults
- Defining a Fault Policy XML File for Handling Business Faults

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 17-4.

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 17-5.

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 Guidelines for BPEL Catch and Catch-All Blocks in Synchronous Request-Response.

Example 17-4 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>
```

Example 17-5 subLanguageExecutionFault Fault Handling

```
<faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"name="bpelx:
 subLanguageExecutionFault">
   <condition>
      <action ref="aia-ora-java"/>
   </condition>
</faultName>
```

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 below:

```
<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 below:

```
<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 Guidelines for BPEL Catch and Catch-All Blocks in Synchronous Request-Response.

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.

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 17-6.

   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.

Example 17-6 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>
```

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.

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 17-7.

   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 *Developing SOA Applications with Oracle SOA Suite.*

Example 17-7 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>
```

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.

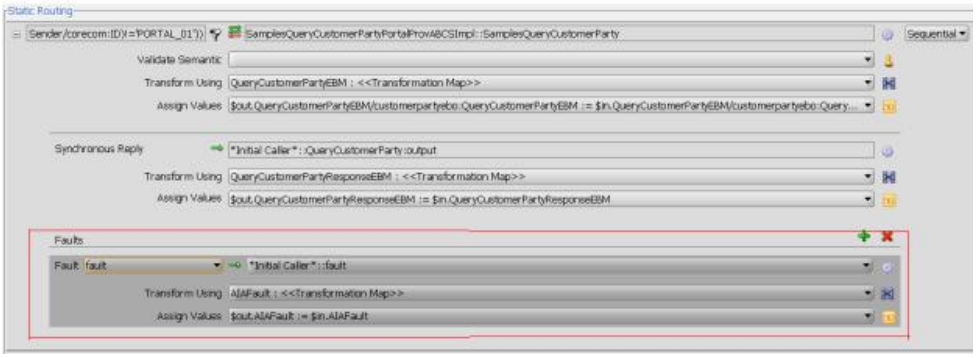
Here are some points to consider:

- 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 17-1.

Figure 17-1 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 *Developing SOA Applications with Oracle SOA Suite.*

Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery

This section includes the following topics:

- Overview
- Configuring Milestones
- Configuring Services Between Milestones

- Guidelines for BPEL Catch and Catch-All Blocks
- Guidelines for Defining Fault Policies
- Configuring Fault Policies to Not Issue Rollback Messages
- Using the Message Resubmission Utility API

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 acknowledgment 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, Release 11.1.1.9.*

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 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 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 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 Configuring Milestones and 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, Release 11.1.1.9.*

For more information about the Message Resubmission Utility API, see Using the Message Resubmission Utility API.

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 17-2 and Figure 17-3 illustrate a few possible milestone locations across an integration flow.

Figure 17-2 Integration Flow in Which the Receiver Target Milestone is the Target Participating Application

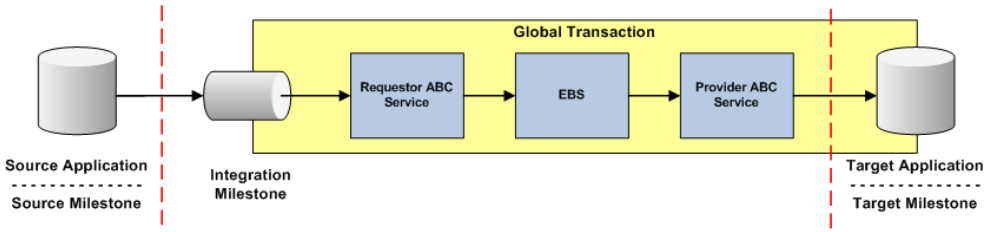
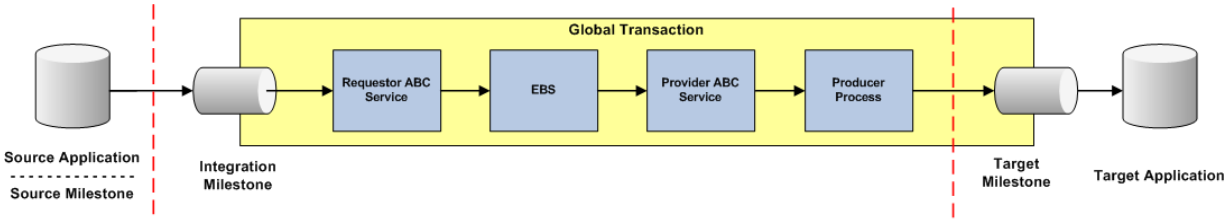


Figure 17-3 Integration Flow in Which the Receiver Target Milestone is Within the Global Transaction Space



Configuring Services Between Milestones

This section includes the following topics:

- Populating Message Resubmission Values
- 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.

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 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 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 17-8 provides sample code snippet from the EBM-to-EBM XSL.

Example 17-8 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>
```

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 17-9 illustrates how to assign the JMS Message ID to the ABM.

Example 17-9 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 17-10 illustrates how to concatenate the data and assign it to the ABM.

Example 17-10 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>
```

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 17-11.

Example 17-11 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 17-12 provides a code snippet that extracts the resubmission values and assign them to EBM header elements.

Example 17-12 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 *Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

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 Designing and Developing Enterprise Business Services, Designing Application Business Connector Services, Constructing the ABCS, and Designing and Constructing Enterprise Business Flows.

Guidelines for BPEL Catch and Catch-All Blocks

This section includes the following topics:

- Handling Run-time Faults Defined in the Fault Policy File
- Handling Run-time Faults Not Defined in the Fault Policy File

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.

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 17-13.
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 *Developing SOA Applications with Oracle SOA Suite.*

Example 17-13 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>
```

Guidelines for Defining Fault Policies

For more information, see Defining a Fault Policy XML File for Handling Run-time Faults.

Configuring Fault Policies to Not Issue Rollback Messages

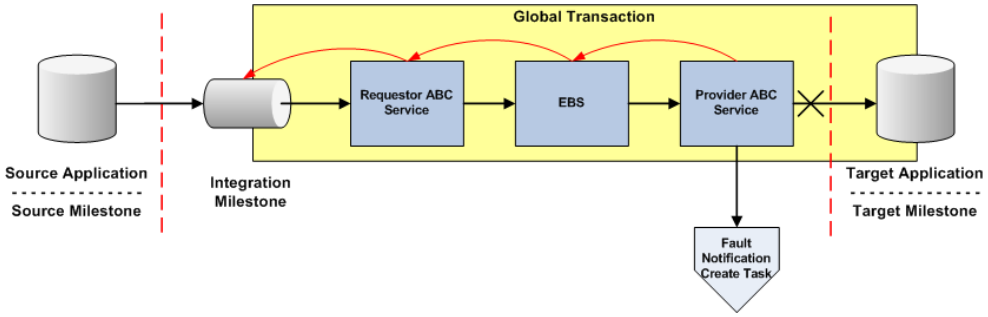
According to the guaranteed message delivery programming model, when a message cannot be delivered to a service or component in the flow of a global transaction, the

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 17-4, 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 17-4 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 17-15.
- 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 17-14.

Example 17-14 Java Snippet to Invoke the Oracle AIA Error Handler

```
<?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>
```

Example 17-15 Sample Fault Policy Using the aia-no-action No-op Action

```
<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>
```

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.

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:

- Defining Corrective Action Codes
- 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, Release 11.1.1.9.*

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 -.

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 -.

Describing the Oracle AIA Fault Message Schema

This section includes the following topics:

- Describing the EBMReference Element
- Describing the B2BMReference Element
- 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 17-5 and Figure 17-6. Fault elements are described in Table 17-1.

Figure 17-5 Fault Element and Its Child Elements (1 of 2)

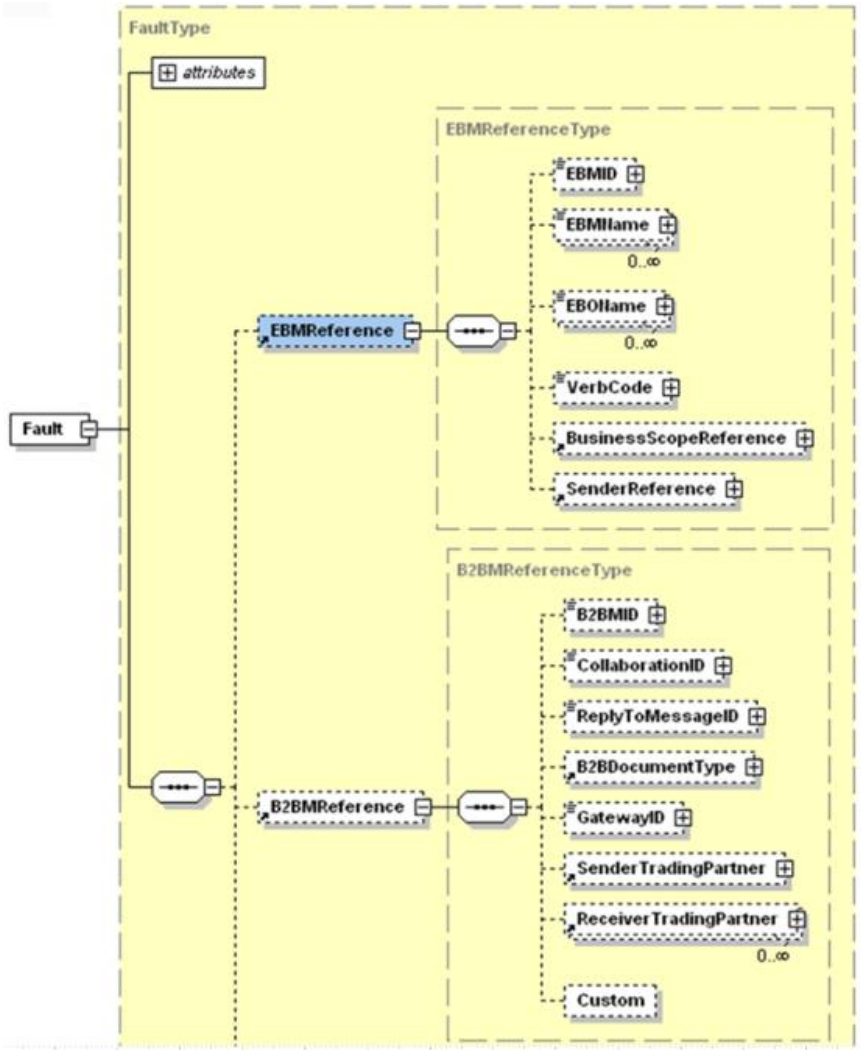


Figure 17-6 Fault Element and Its Child Elements (2 of 2)

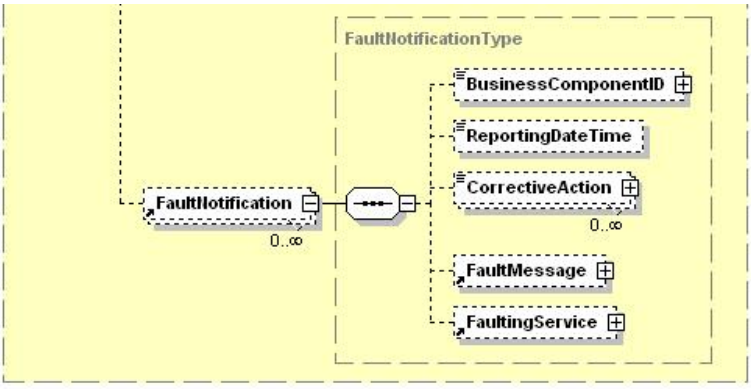


Table 17-1 Fault Elements

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 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 Describing the B2BMReference Element.
FaultNotification	Provides actual details of the fault.	For more information, see Describing the FaultNotification Element.

Describing the EBMReference Element

This section provides details about the `EBMReference` element in the Oracle AIA fault message schema, as shown in Figure 17-7. EBM Reference elements are discussed in Table 17-2.

Figure 17-7 EBMReference Element and Its Child Elements

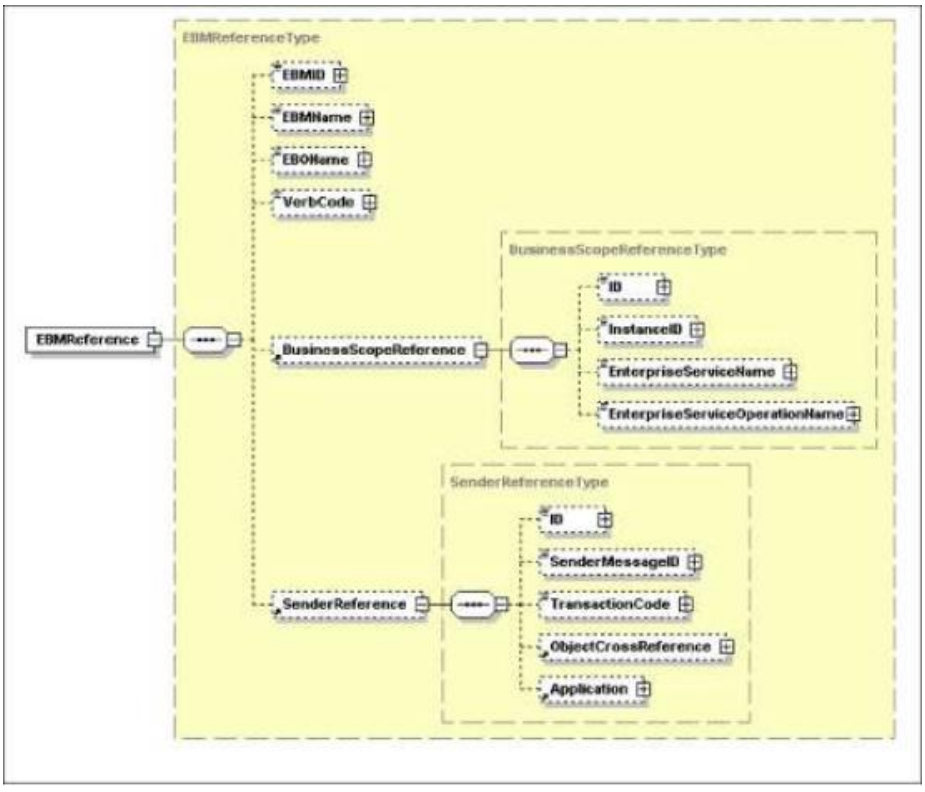


Table 17-2 EBMReference Elements

Name	Purpose
EBMID	Provides the EBMID in the message.

Name	Purpose
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 Introducing EBM Header Concepts.

Describing the B2BMReference Element

This section provides details about the `B2BMReference` element in the Oracle AIA fault message schema, as shown in Figure 17-8. B2BM reference elements are discussed in Table 17-3.

Figure 17-8 B2BMReference Element and Its Child Elements

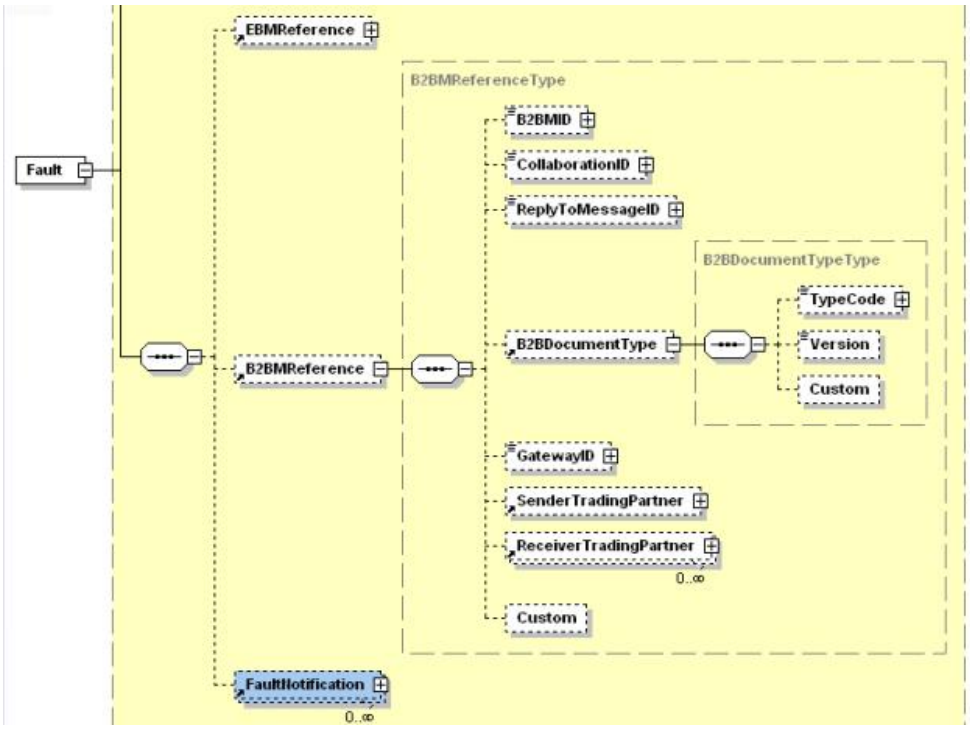


Table 17-3 B2BMReference Elements

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 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 Introduction to B2B Integration Using AIA.
ReplyToMessageID	Provides the ID of the reply-to message.	For more information, see 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 Introduction to B2B Integration Using AIA.
B2BDocumentType/ Version	Provides the document type version of the failed transaction in Oracle B2B.	For more information, see Introduction to B2B Integration Using AIA.
SenderTradingPartner/TradingPartnerID	Provides the name of the sending trading partner in the B2B flow.	For more information, see Introduction to B2B Integration Using AIA.
ReceiverTradingPartner/ TradingPartnerID	Provides the name of the receiving trading partner in the B2B flow.	For more information, see 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 Introduction to B2B Integration Using AIA.

Describing the FaultNotification Element

This section includes the following topics:

- FaultMessage Element
- IntermediateMessageHop Elements
- FaultingService Element

This section provides details about the `FaultNotification` element in the Oracle AIA fault message schema, as shown in Figure 17-9. Fault notification elements are discussed in Table 17-4.

Figure 17-9 FaultNotification Element and Its Child Elements

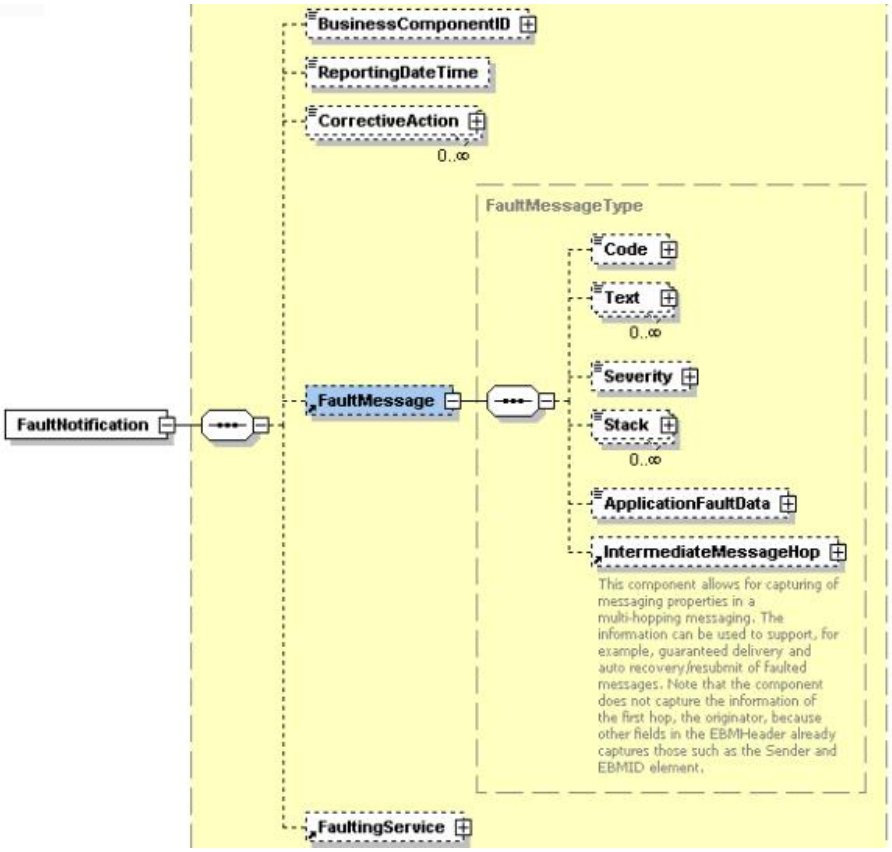


Table 17-4 FaultNotification Elements

Name	Purpose	Details
BusinessComponent ID	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 FaultMessage Element.
FaultingService	Provides details of the faulting service.	For more information see FaultMessage Element.

FaultMessage Element

Table 17-5 discusses fault message elements.

Table 17-5 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 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 Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery.

IntermediateMessageHop Elements

Figure 17-10 illustrates `IntermediateMessageHop` elements. Intermediate message hop elements are discussed in Table 17-6.

Figure 17-10 IntermediateMessageHop Elements

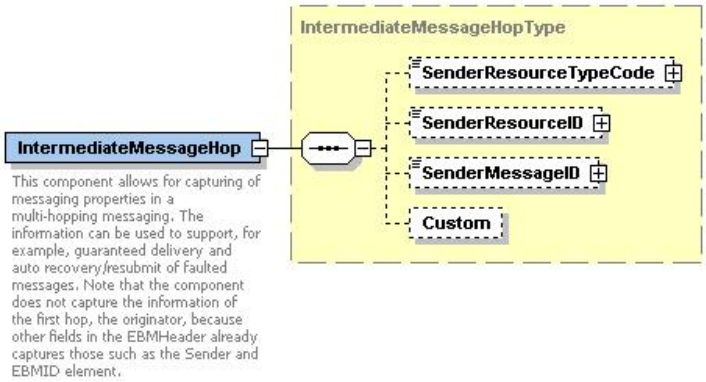


Table 17-6 IntermediateMessageHop Elements

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.

FaultingService Element

Table 17-7 discusses the FaultingService element.

Table 17-7 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.

Extending Fault Messages

This section includes the following topics:

- Introduction to Extending Fault Messages
- Extending a Fault Message

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.

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 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.

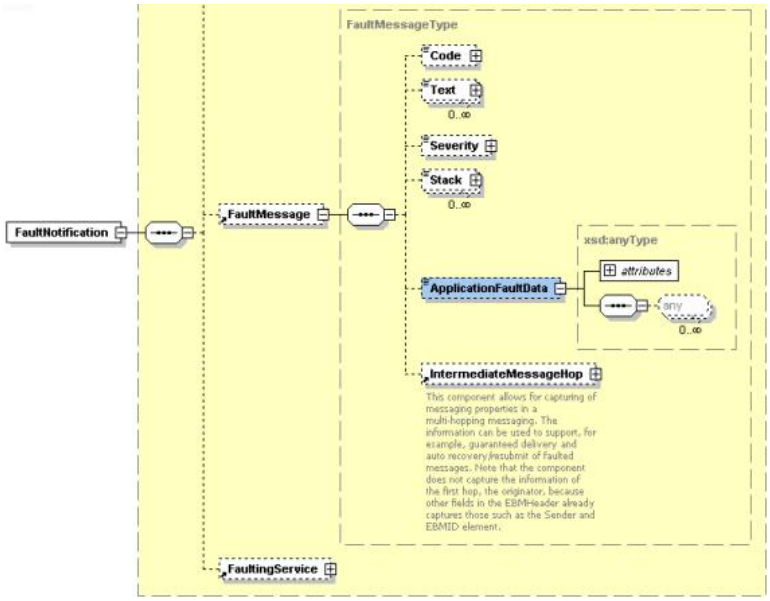
For more information about extending error handling, see Extending Error Handling.

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.

Extending a Fault Message

Extending a fault message uses the `ApplicationFaultData` element, highlighted in Figure 17-11, of type xsd:anyType in the FaultType message schema definition in Meta.xsd.

Figure 17-11 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 runtime 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, Release 11.1.1.9.*

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 17-12. The value for this property is the fully qualified class path of the handler.

   Figure 17-12 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 17-16 and Example 17-17 illustrate the interface structure.

   You can view extended field values in the error logs accessed in Oracle Enterprise Manager.

   You should also be able to view them in email notifications if they have been configured appropriately.

Example 17-16 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 17-17 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);
}
```

Extending Error Handling

This section includes the following topics:

- Introduction to Extending Error Handling

- Implementing an Error Handling Extension

This section provides an overview of error handling extension and discusses how to implement an error handling extension.

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.

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 Extending Fault Messages.

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.

   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.

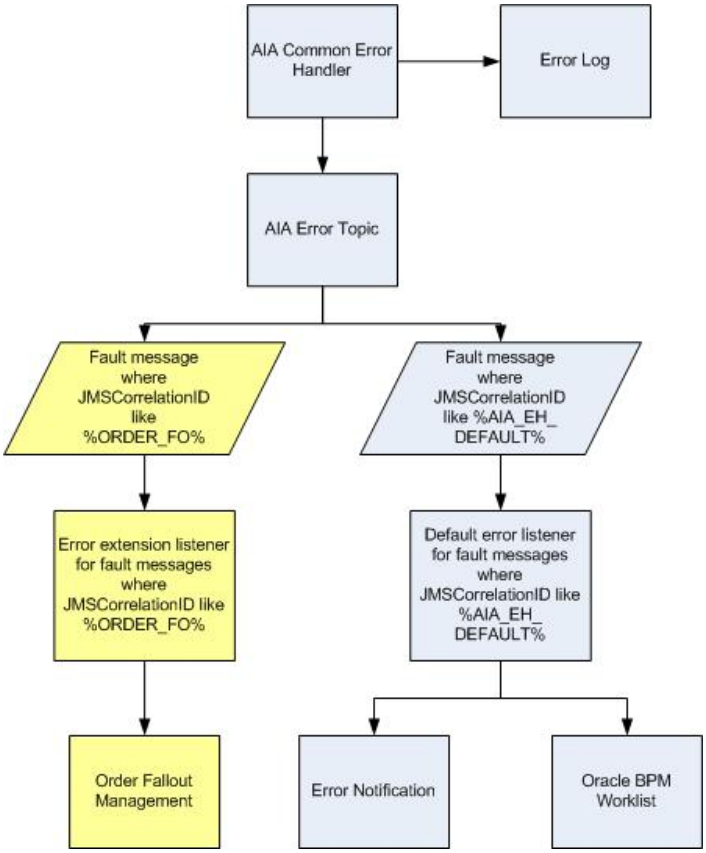
   Configure an error extension listener to filter fault messages based on the JMS Header - JMSCorrelationID value defined by the error type you created in step 1.

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 17-13.

Figure 17-13 Sample Extended Error Handling Flow Alongside a Default Error Handling Flow



Configuring Oracle AIA Processes for Trace Logging

This section includes the following topics:

- Describing Details of the isTraceLoggingEnabled Custom XPath Function
- Describing Details of the logTraceMessage Custom XPath Function
- 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, Release 11.1.1.9.*

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.

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.

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.

18

Working with AIA Design Patterns

This chapter provides an overview of AIA message processing patterns, AIA assets centralization patterns, AIA assets extensibility patterns.

Note:

Composite Business Processes (CBP) will be deprecated from next release. Oracle advises you to use BPM for modeling human/system interactions.

This chapter includes the following sections:

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:

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

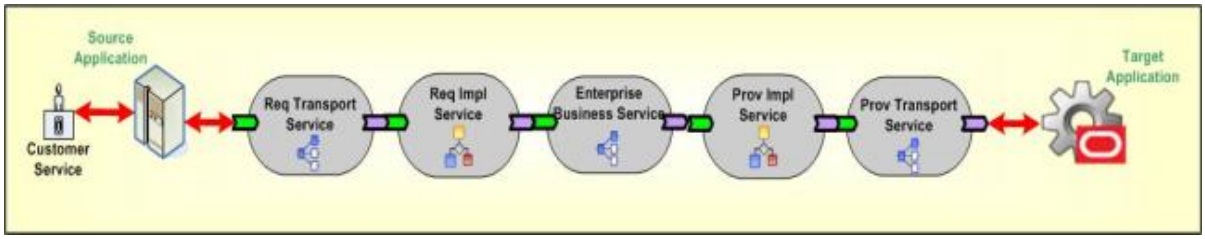
AIA recommends using the synchronous request and response service interface in all the composites involved in processing or connecting the back-end systems. There should not be any singleton service in the services involved in the query processing. The general recommendation is for all of the intermediary 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 18-1 illustrates how a task implemented as an Enterprise Business Service (EBS) is invoked synchronously by the requester application.

Figure 18-1 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.

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 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.

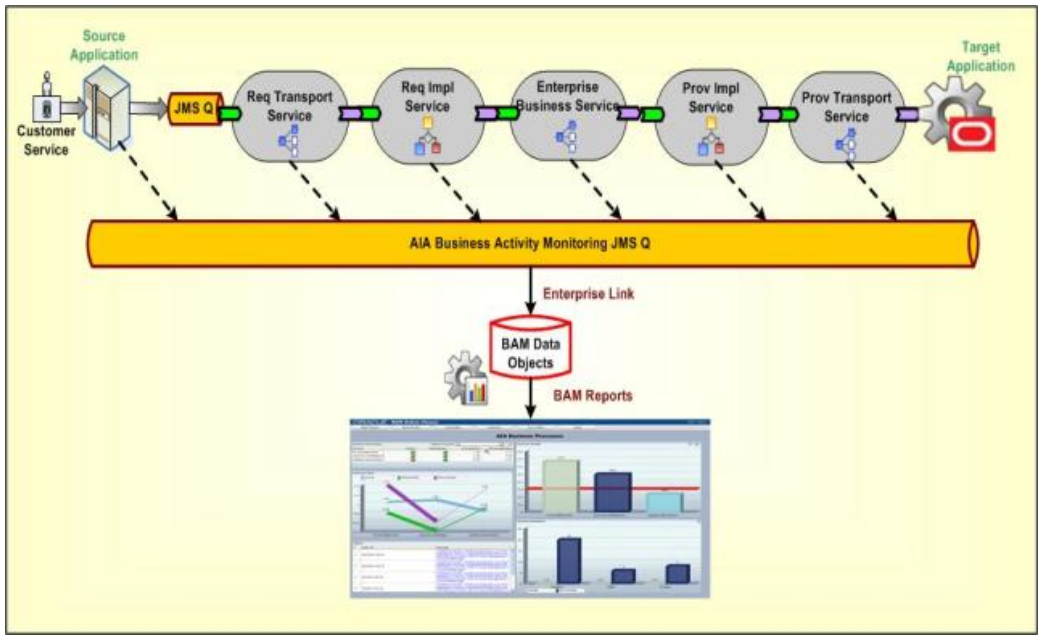
AIA recommends having a queue per business object. All requests emanating from a requester application for a business object use the same queue.

Figure 18-2 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 18-2 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 18-2 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 Designing Application Business Connector Services and Constructing the ABCS.

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.

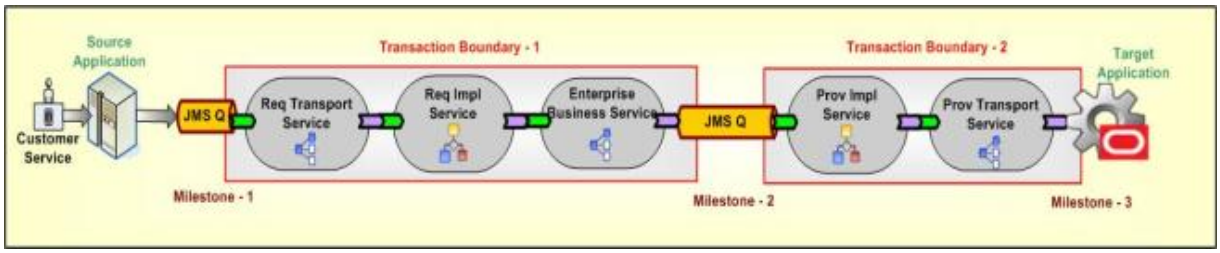
Solution

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 18-3 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 18-3 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.

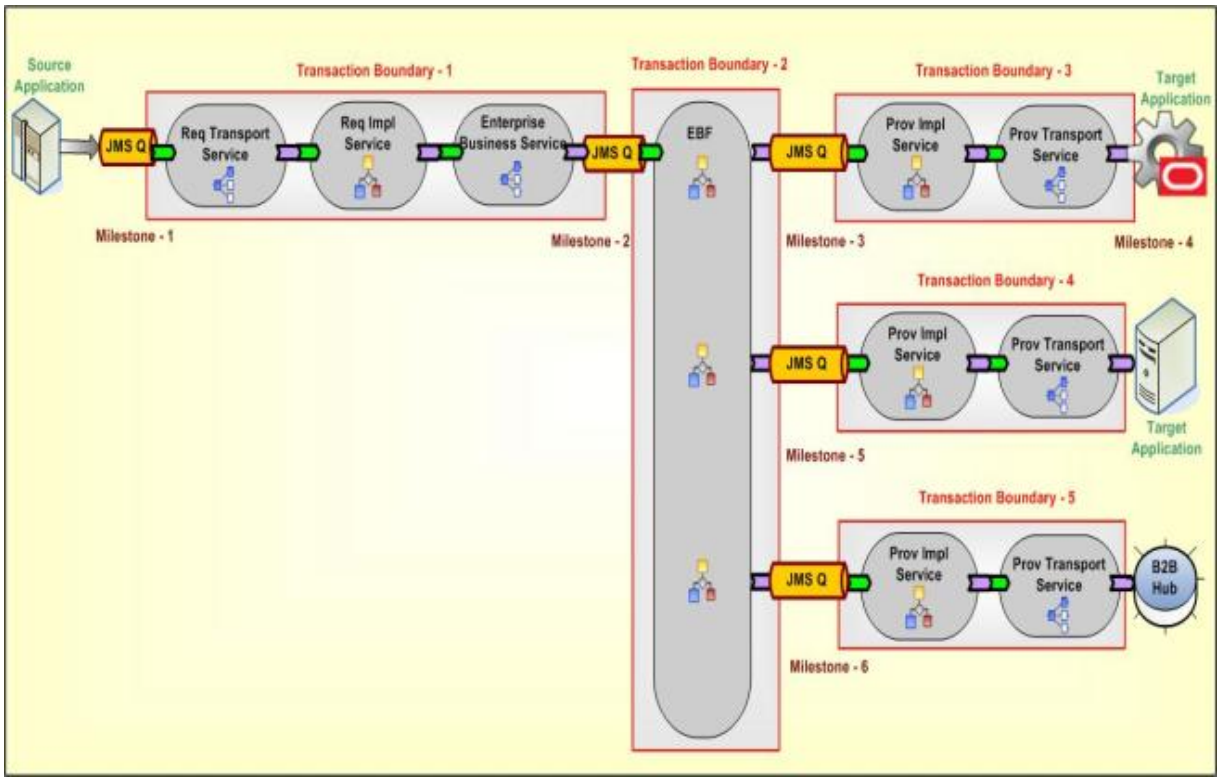
The provider implementation service picks up the billing profile information and ensures the message is delivered to the target application. The target application may or may not have the capability of participating in the transaction between Milestone-2 and Milestone-3 so the implementation service 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 18-4 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 18-4 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 runtime 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.

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.

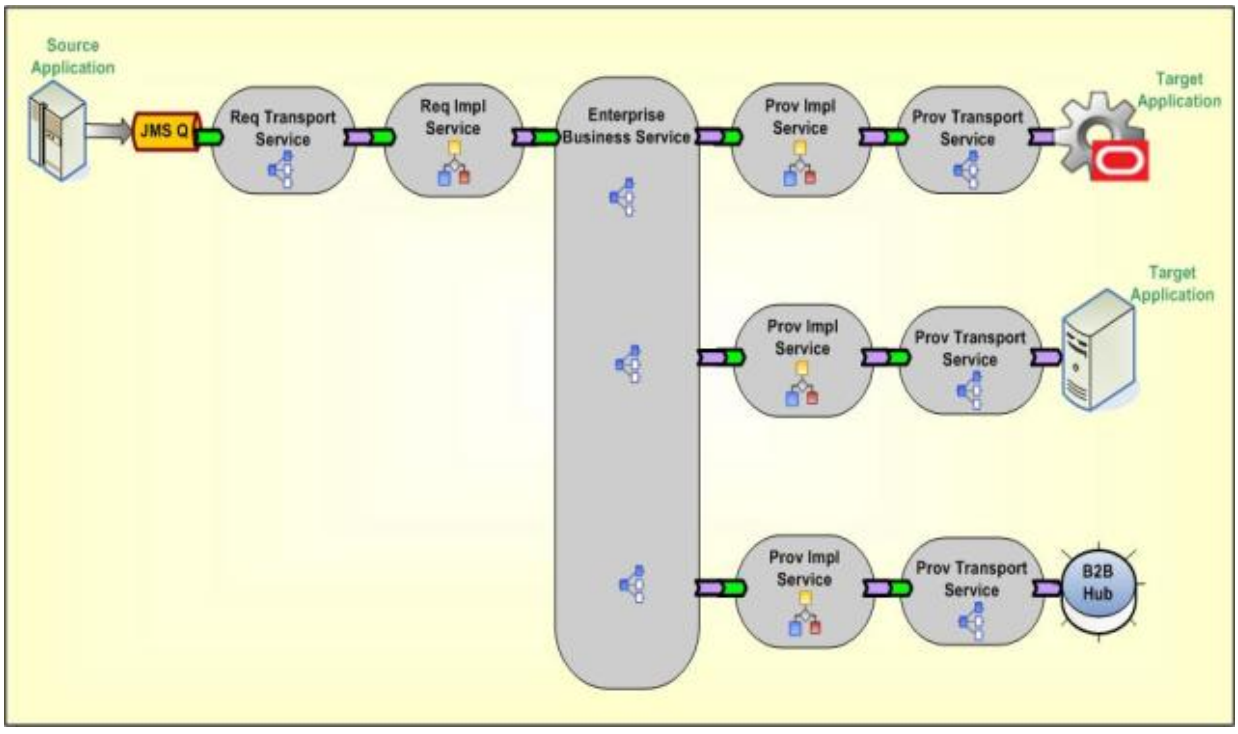
Solution

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 18-5 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 18-5 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.

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.

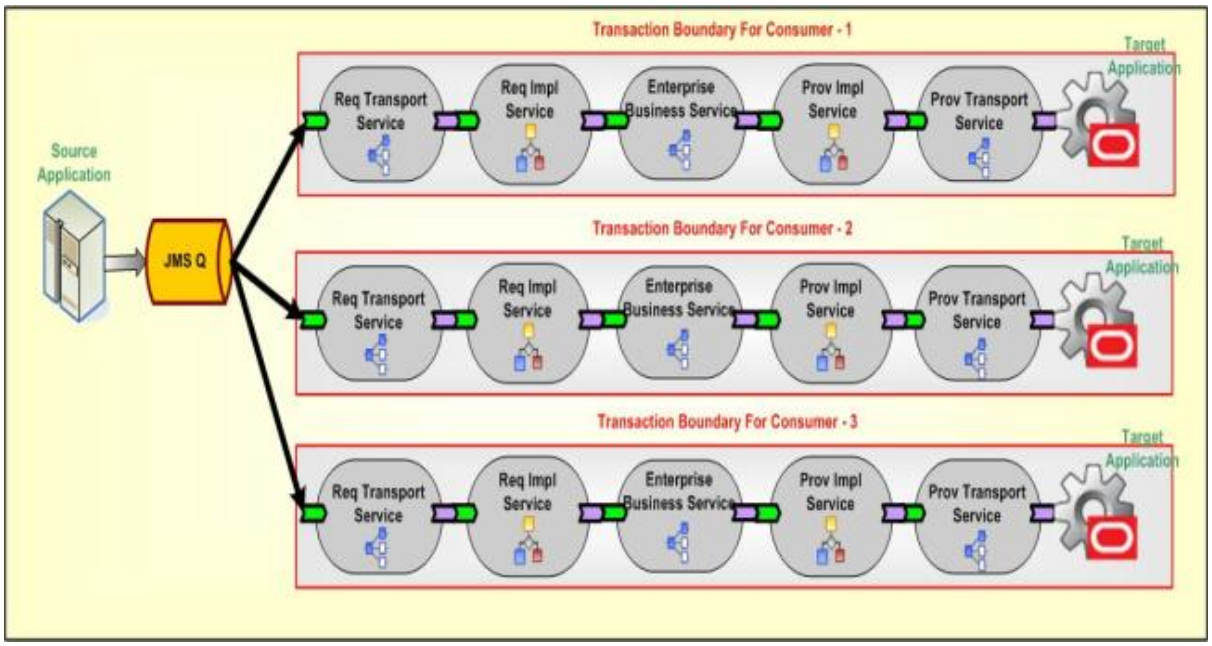
Solution

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 Asynchronous Fire-and-Forget Pattern for more details.

Figure 18-6 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 18-6 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.

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.

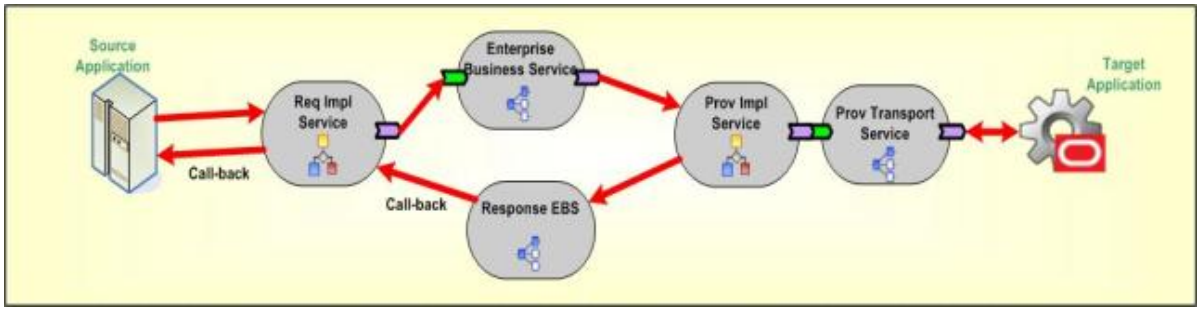
Solution

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 18-7, 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 18-7 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.

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.

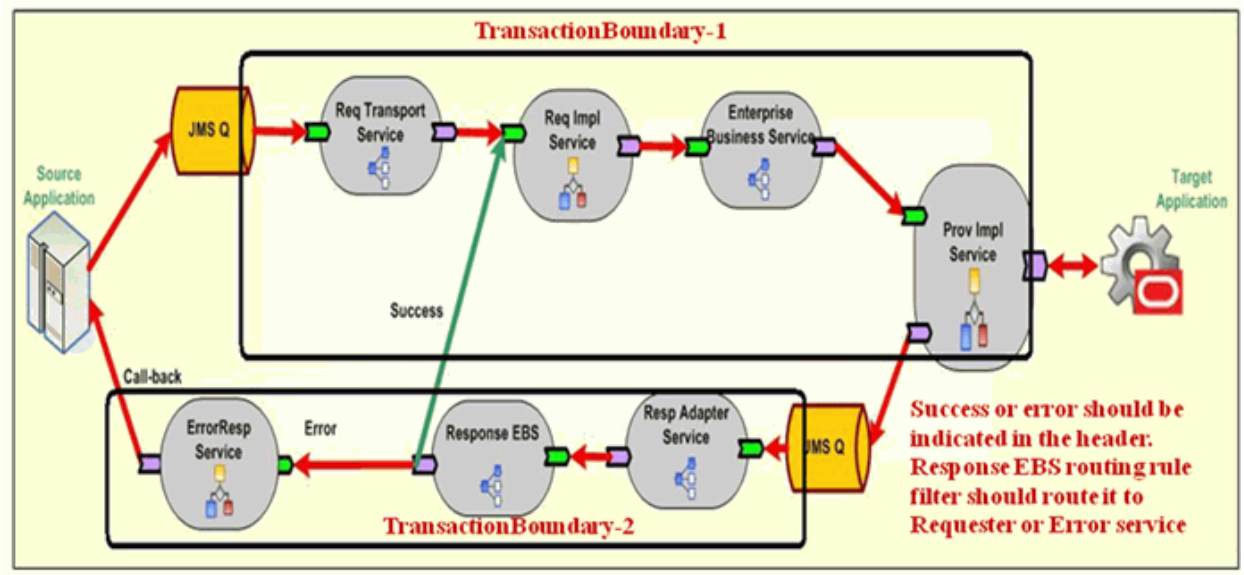
Solution: Success scenario calls the same requester but separate service for the Error Handling

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 18-8 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.

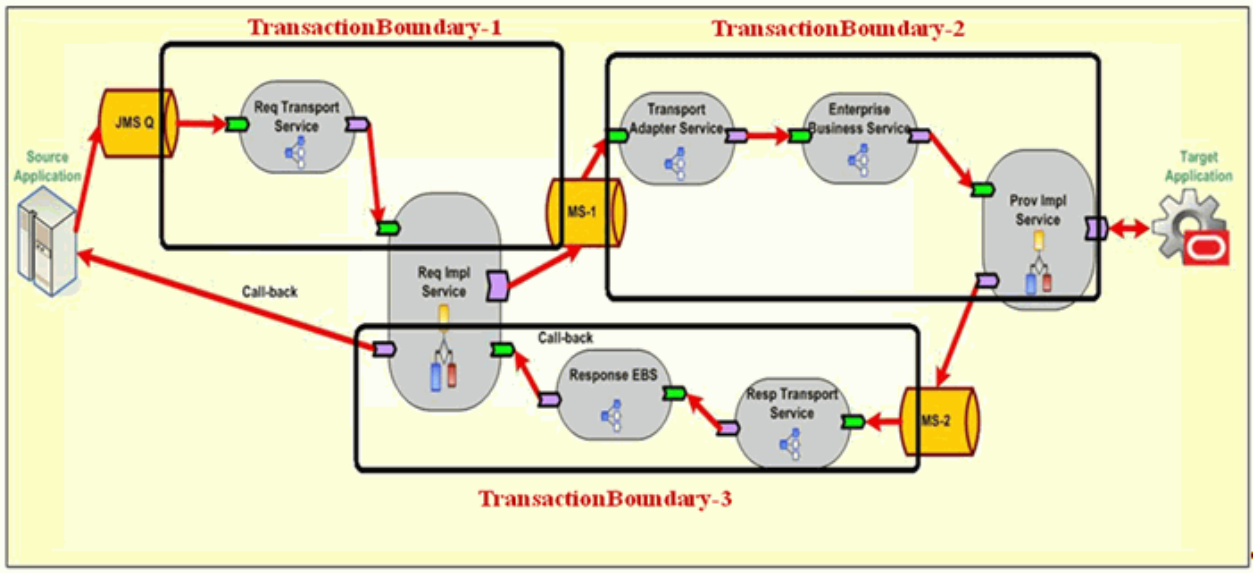
Figure 18-8 Example of Interaction Divided into Two Transactions: Error Handling and Requester ABCS



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 18-9 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 18-9 Error Scenario when Using JMS Queue as a Milestone

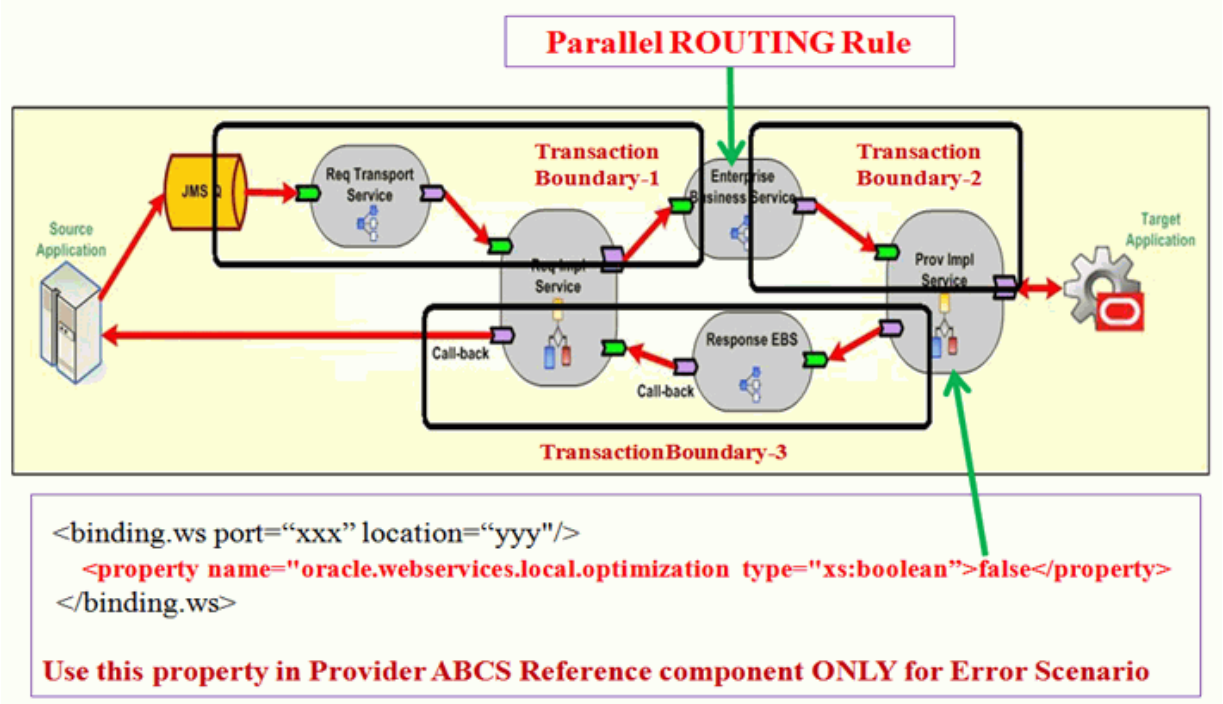


Solution: Using parallel routing rule and Web Service call

In this solution, the interaction is divided into three transactions. Figure 18-10 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 18-10 Error Scenario Using Parallel Routing Rule and Web Service Call



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:

- How to Avoid Redundant Data Model Representation in AIA
- How to Avoid Redundant Service Contracts Representation in AIA

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 runtime.

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.

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 runtime.

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.

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:

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 18-1 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 18-2 shows the Custom Schema - A: "AIAComponents \EnterpriseObjectLibrary\Industry\Telco\Custom\ EBO\CustomerParty \V2\CustomCustomerPartyEBO.xsd"

Example 18-1 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 18-2 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"/>
```

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.

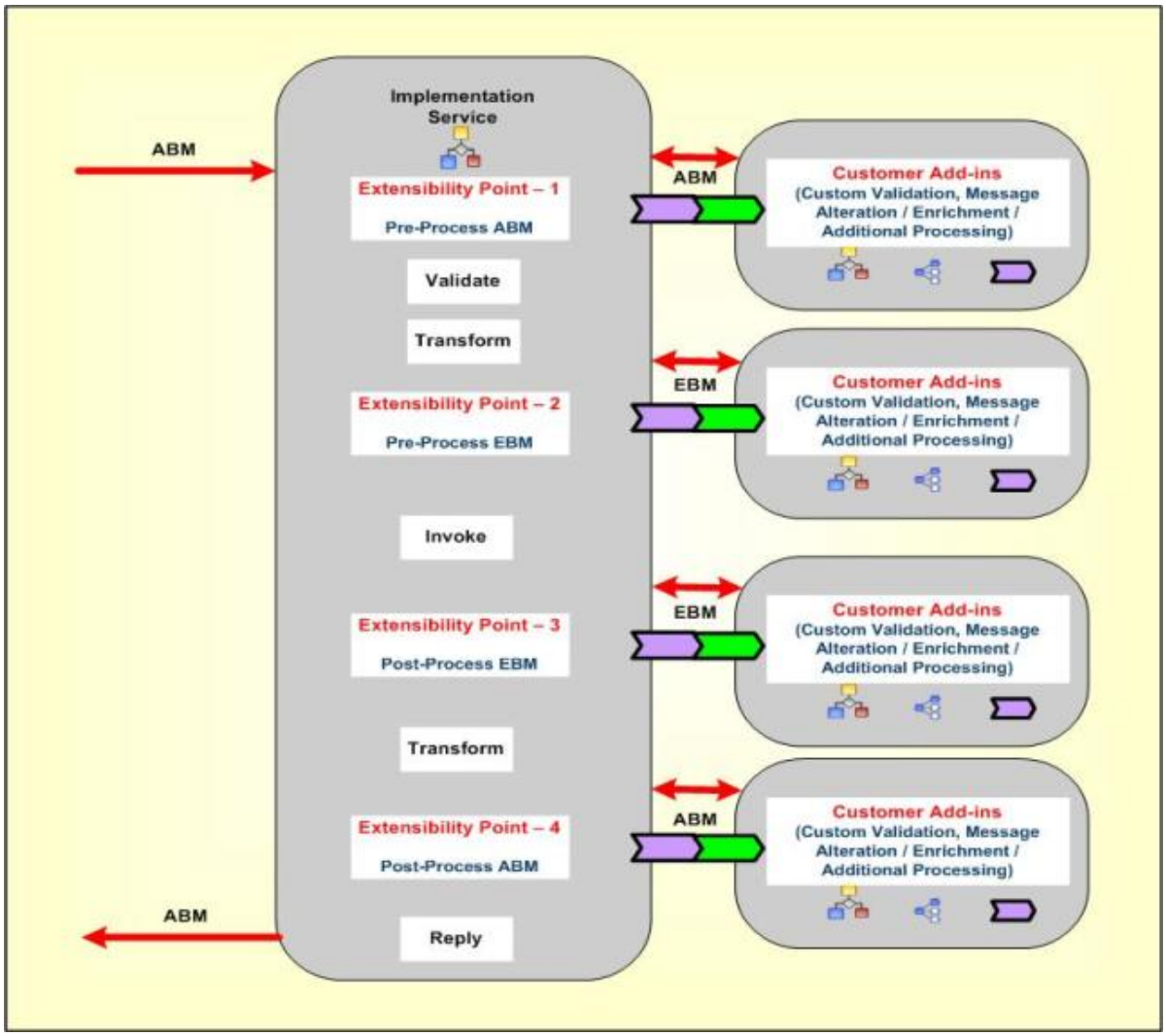
Solution

AIA recommends introducing various extension points at logical points during the service implementation. Figure 18-11 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 18-11 Extending AIA Services



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 18-3 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 18-4 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 18-3 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 18-4 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>
```

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.

19

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:

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.

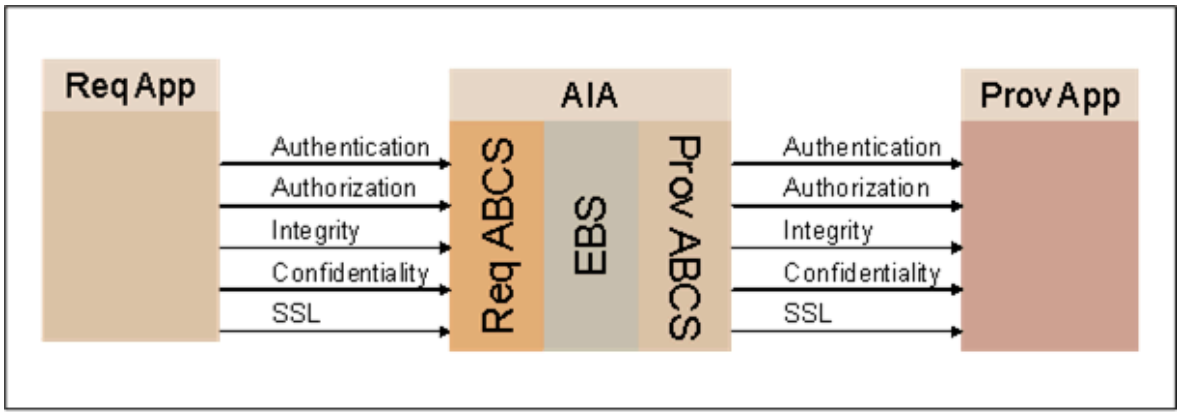
Securing Service to Service Interaction

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 19-1 illustrates the high-level security structure for AIA.

Figure 19-1 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.

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.

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 *Administering 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 *Administering Web Services*

Implementing Security

This section includes the following topics:

- Enabling Security for AIA Services
- Invoking Secured Application Services
- Overriding Policies Using a Deployment Plan
- Testing Secured Services using CAVS.

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 *Administering 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 *Administering Web Services.*

6. Diagnose problems.

   For more information about how to diagnose problems, see "Diagnosing Problems" in *Administering Web Services.*

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.

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 *Administering 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:

   a. Configure credential store.

   b. Add the UserID and password associating with a key into the store.

   c. Use the key in the policy.

   For more information about how to configure each policy, see "Configuring Policies" in *Administering Web Services.*

6. Diagnose problems.

   For more information about how to diagnose problems, see "Diagnosing Problems" in *Administering Web Services.*

Overriding Policies Using a Deployment Plan

SOA Core Extension 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 AIA Security Configuration Properties .

2. Place it in the project folder.

Testing Secured Services using CAVS

To test secured services using CAVS, the element <cavs:CAVSRequestInput_1> should have the element shown in Example 19-1 under the <soap:Envelope>.

If you are using a default user in the Identity store, then [user name] = weblogic and [pwd] = weblogic#1.

Example 19-1 <soap:Envelop> 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>
```

Security for Applications

This section contains the following topics:

- Enabling Security in Application Services
- Invoking Secured AIA Services

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.

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 19-2 is a sample of a security header for authentication.

If the AIA service requires SSL, then the application should configure SSL for both one way and two-way SSL.

Example 19-2 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>
```

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 SOA Core Extension 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.

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.

Policy Naming Conventions

This section includes the following topics:

- Naming Conventions for Global Policy Sets
- Naming Conventions for Overriding Config Params

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

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.

How Does SOA Core Extension Help in Securing AIA Services?

SOA Core Extension automatically:

- Creates the recommended global policy sets.
- Attaches the local policies for the composites when required.

This section includes the following topics:

What Default Policies are Attached to a Service?

The deployment of global policies is handled during the SOA Core Extension installation. A set of global services and client policies are attached to the WLS domain at the time of SOA Core Extension installation. The result is automatic securing of all services (matching the pattern) deployed on that server.

SOA Core Extension installer creates both service-specific and client-specific global policy sets for the composite name patterns:

- ABCS
- EBS
- EBF
- Adapter

- Producer
- Consumer

SOA Core Extension 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

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 19-3.

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 SOA Core Extension tool. These composites must have an associated xml-based security configuration file. This file is named **AIASecurityConfigurationProperties.xml**.

The SOA Core Extension 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 19-3 shows a sample AIASecurityConfigurationProperties.xml.

> **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.

Example 19-3 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>
```

Application Security Context

This section includes the following topics:

- Introduction to Application Security
- How To Exchange Security Context Between Participating Applications and ABCS

- Mapping Application Security Context in ABCS To and From Standard Security Context
- Using the AppContext Mapping Service
- Understanding the Structure for Security Context
- Using Attribute Names
- Propagating Standard Security Context through EBS and EBF
- Implementing Application Security Context

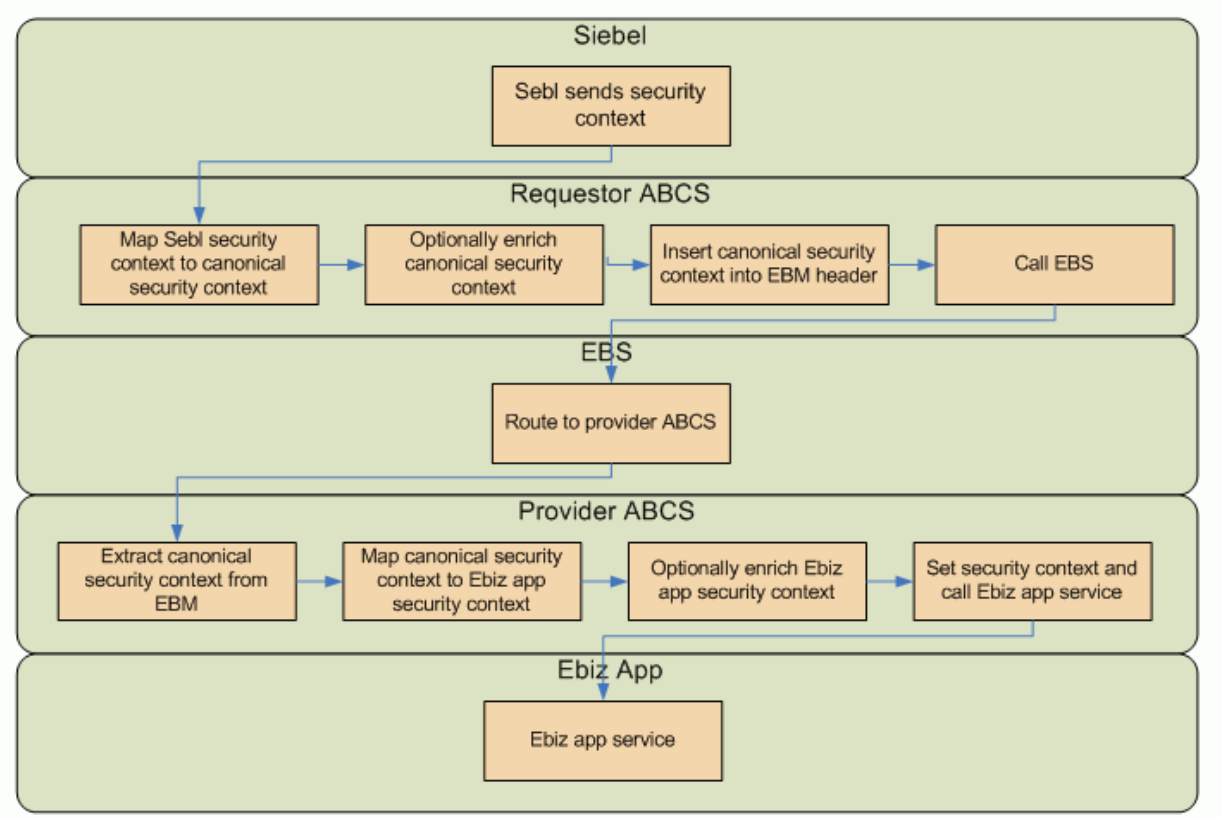
Introduction to Application Security

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 19-2 illustrates the high-level security functional flow.

Figure 19-2 Security Functional Flow



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.

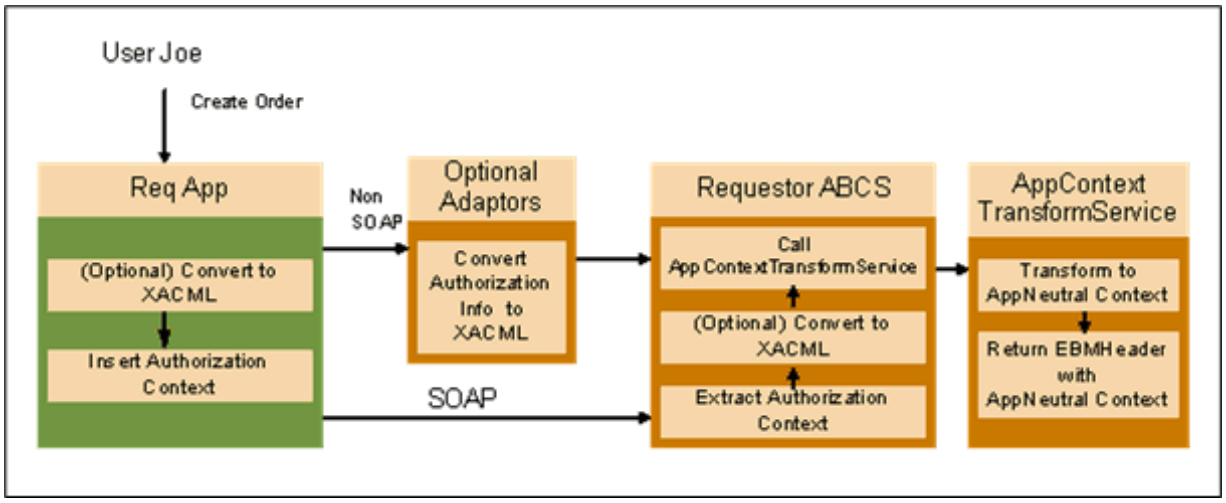
Requester Applications

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 19-3 illustrates the requester application flow.

Figure 19-3 Requester Application Flow



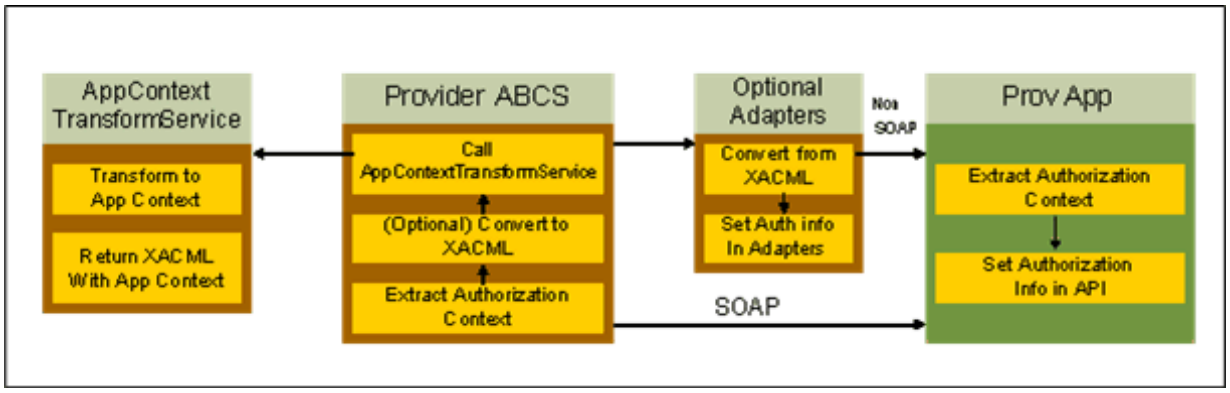
Provider Applications

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 19-4 illustrates the provider application flow.

Figure 19-4 Provider Application Flow



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.

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 19-4 shows a sample of the AppContextMappingService.

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.

Example 19-4 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>
```

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 19-5 illustrates the structure of XACML Request.

Figure 19-5 Structure of XACML Request

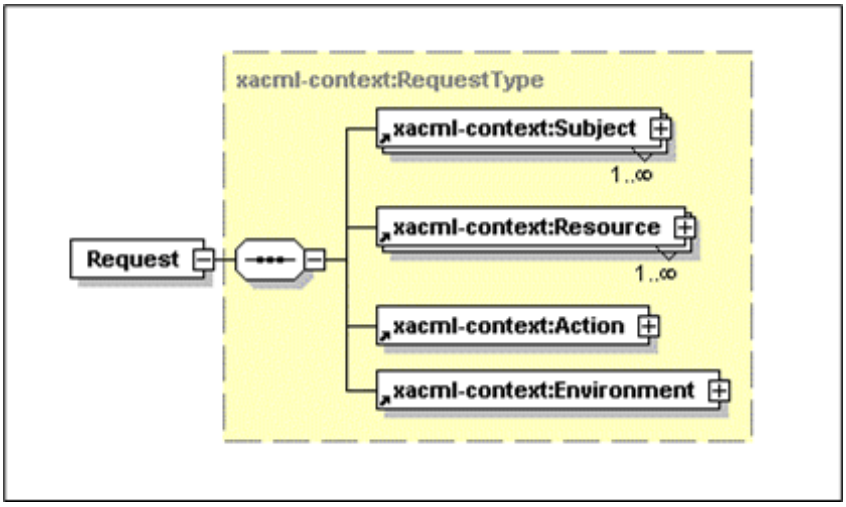


Figure 19-6 illustrates the structure of XACML Subject.

Figure 19-6 Structure of XACML Subject

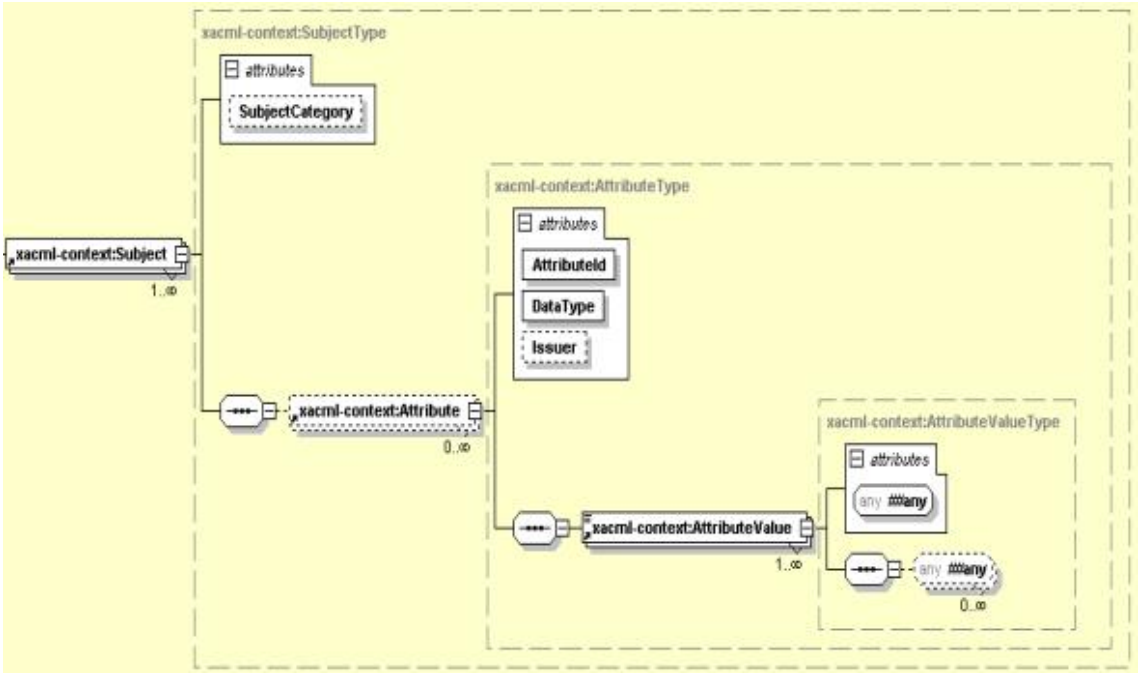


Figure 19-7 illustrates the structure of XACML Resource.

Figure 19-7 Structure of XACML Resource

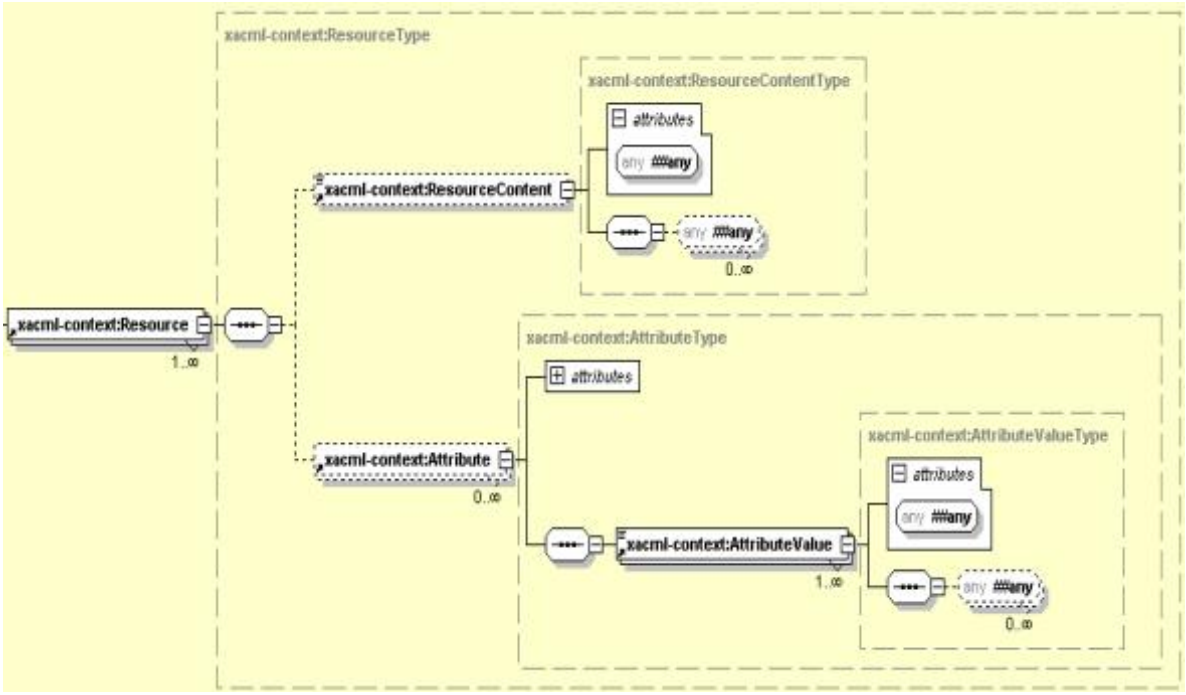


Figure 19-8 illustrates the structure of XACML Action.

Figure 19-8 Structure of XACML Action

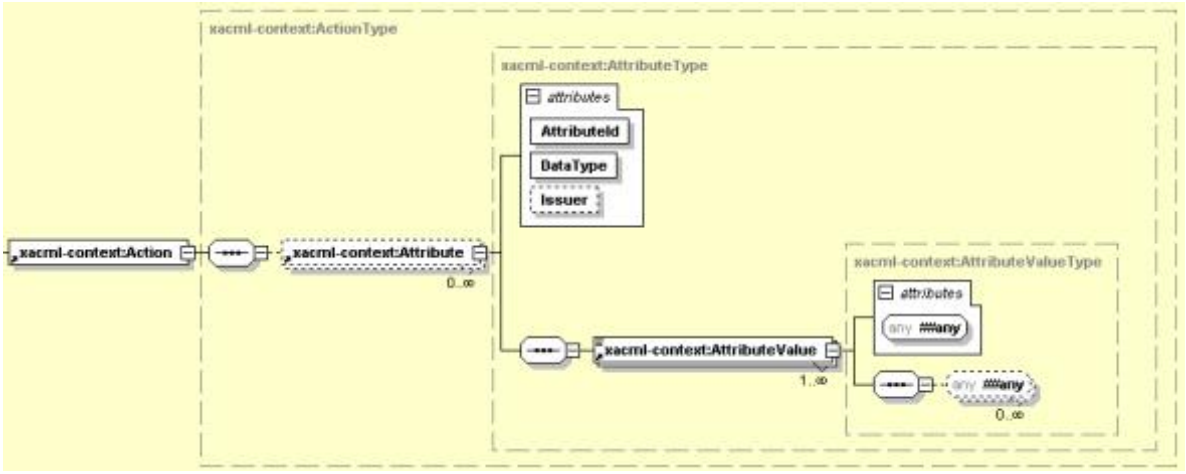
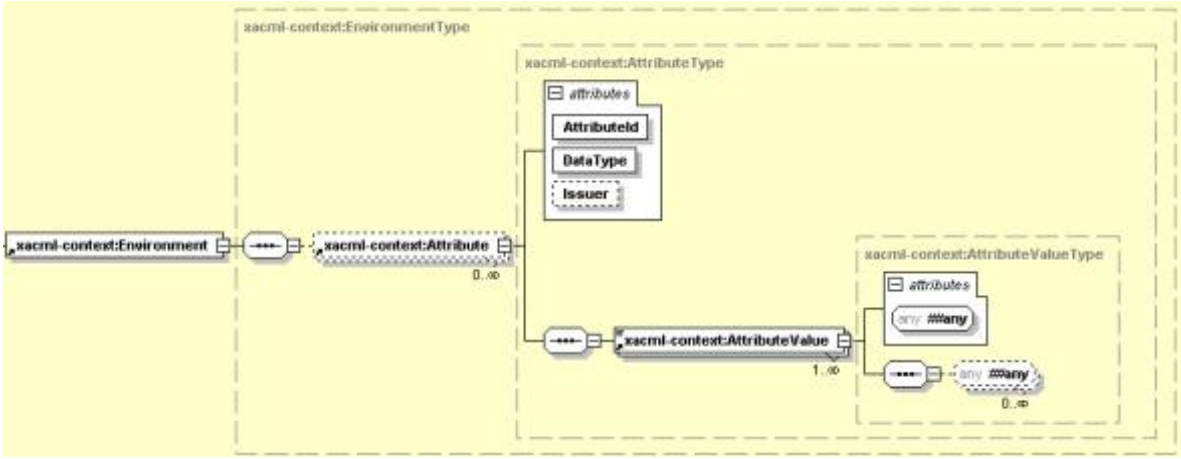


Figure 19-9 illustrates the structure of XACML Environment.

Figure 19-9 Structure of XACML Environment



Example 19-5 shows the SEBL AppContext information that is sent to the security service.

Example 19-5 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>
```

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

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

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.

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.

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.

20

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.

Note:

Composite Business Processes (CBP) will be deprecated from next release. Oracle advises you to use BPM for modeling human/system interactions.

This chapter includes the following sections:

- General Guidelines for Design_ Development_ and Management of AIA Processes
- Building Efficient BPEL Processes

General Guidelines for Design, Development, and Management of AIA Processes

This section includes the following topics:

- Interpreting Empty Element Tags in XML Instance Document
- Purging the Completed Composite Instances
- Syntactic / Functional Validation of XML Messages
- Provide Provision for Throttling Capability
- Artifacts Centralization
- Separation of Concerns
- Adapters Inside ABCS Composite OR as Separate Composite
- AIA Governance

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 20-1 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 20-2 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 20-3 illustrates how a message containing significant elements can be processed.

Use the construct in Example 20-1 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.

Use the construct in Example 20-2 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.

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 20-3.

Example 20-1 ABM -> EBM Transformation

```
~--> <xsl:if test="ABMSourceElement/text()"> <EBMTargetElement> <xsl:value-of
select="ABMSourceElement"/> </EBMTargetElement> </xsl:if>
```

Example 20-2 ABM -> EBM Transformation

```
~--> <xsl:if test="ABMSourceElement/text()"> <EBMTargetElement> <xsl:value-of
select="ABMSourceElement"/> </EBMTargetElement> </xsl:if>
```

Example 20-3 EBM -> ABM Transformation

```
~--> <xsl:if test="EBMSourceElement"> <ABMTargetElement> <xsl:value-of
select="EBMSourceElement"/> </ABMTargetElement> </xsl:if>
```

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 "Managing Database Growth" in *Administering Oracle SOA Suite and Oracle Business Process Management Suite.*

Syntactic / Functional Validation of XML Messages

This section discusses:

- Syntactic Validation
- Data / Functional Validation

Syntactic Validation

With document style, a web service endpoint can use the capabilities of a validating parser and the runtime 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 runtime 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 *Developing SOA Applications with Oracle SOA Suite.*

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.

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 AIA Message Processing Patterns.

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 Using MDS in AIA.

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 runtime 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.

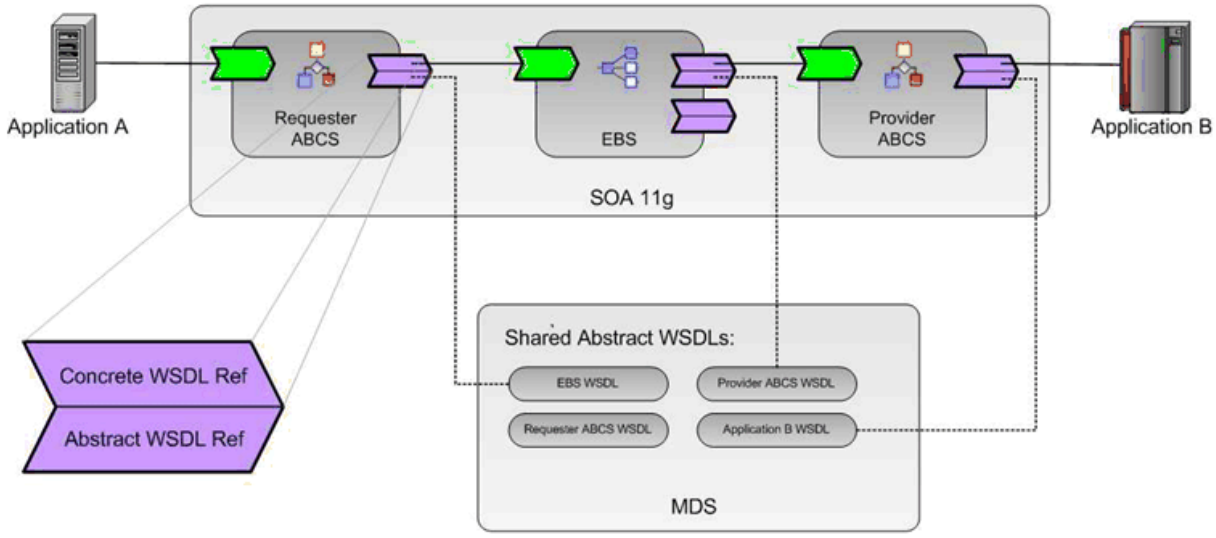
- Do not use the concrete WSDLs directly in your consuming artifacts. If the Service Provider (not as in ProviderABCS) changes, then you must redeploy the consumer.
- Do not reference the deployed concrete WSDL of a Service Provider. If Provider is unavailable, then consumer cannot be compiled or be 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 20-1 illustrates that references between SOA composites actually consist of two parts: an abstract WSDL (design time) and a concrete WSDL (runtime).

Figure 20-1 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.

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 runtime. MDS is like a version management system, where you can store and use the shared common artifacts at design and runtime.

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 20-1:

Table 20-1 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
B2BObjectLibrary	Business-to-business (B2B) schemas

Location	Artifacts
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 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 Getting a Quick Start on your 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 Populating the binding.ws Element in the composite.xml.

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?

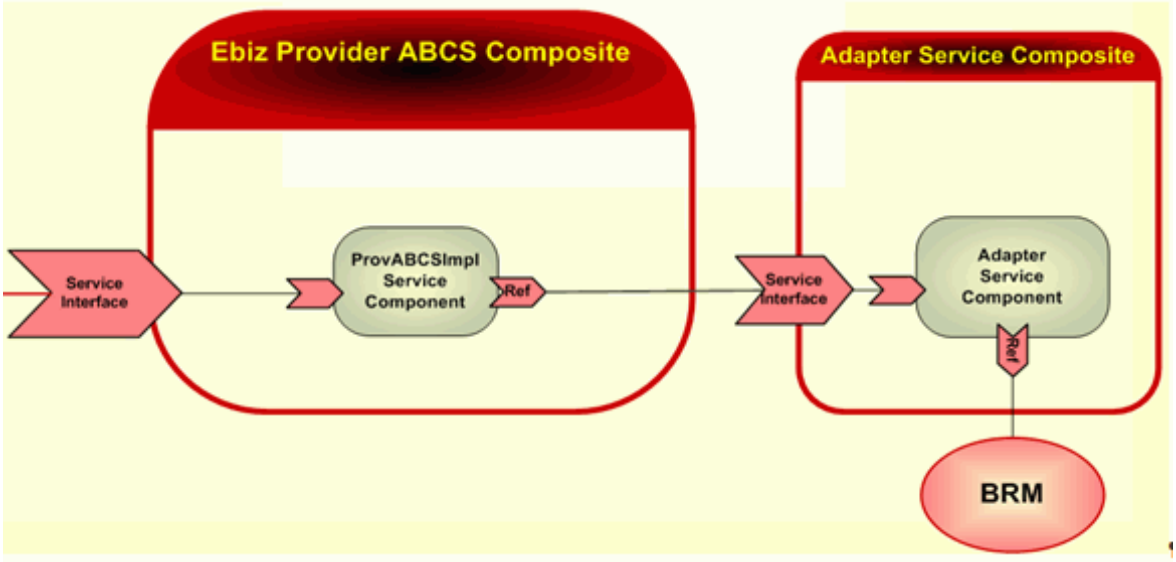
An architectural decision should be made on how you build the SCA. With SOA/SCA it is all about re-usability. Think about this - are you designing your SCA based on the business flow or are you designing our SCA from technical point of view? The answer lies in finding the correct balance.

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.

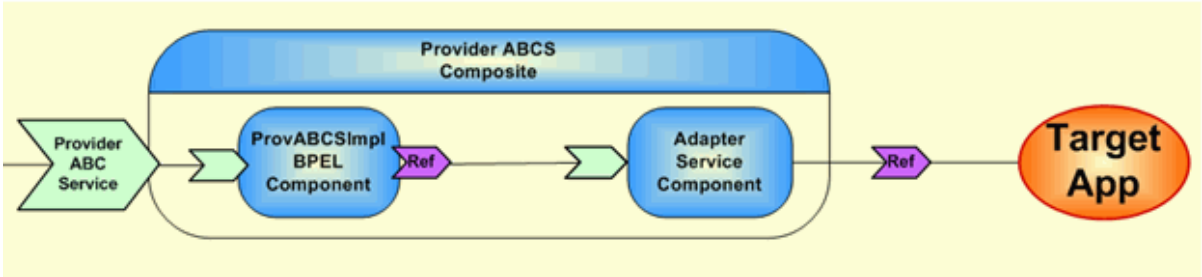
Going back to the question of whether the adapters should be inside the ABCS composite or should be developed as a separate composite, as a best practice, AIA recommends that you put adapters that are interfaced with ABCS in a different composite. as shown in Figure 20-2. This is also the preferred way when the same transport adapter service could be used with multiple ABCS.

Figure 20-2 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 20-3.

Figure 20-3 Example of ABCS and Transport Adapter Service in the Same Composite



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 runtime.

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:

- Using BPEL as "Glue"_ Not as a Programming Language
- Avoiding Global Variables Wherever Possible
- Avoiding Large FlowN
- Controlling the Persistence of Audit Details for a Synchronous BPEL Process
- Using Non-Idempotent Services Only When Absolutely Necessary
- Defining the Scope of the Transaction
- Disabling the Audit for Synchronous BPEL Process Service Components
- Including No Break-Point Activity in a Request-Response Flow

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.

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.

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 runtime when an instance document having 100 repeating instances is processed.

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 20-4 illustrates the use of local variables.

Example 20-4 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>
```

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*.

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.

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 *Developing SOA Applications with Oracle SOA Suite*

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.

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 20-5 shows how to set the bpel.config.auditLevel property to an appropriate value in the composite.xml file of your SOA project.

Example 20-5 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>
```

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.

21

Oracle AIA Naming Standards for AIA Development

This chapter provides an overview of guidelines for naming standards for various AIA components for AIA development.

> **Note:**
>
> Composite Business Processes (CBP) will be deprecated from next release. Oracle advises you to use BPM for modeling human/system interactions.

This chapter includes the following sections:

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.

XML Naming Standards

The following standards are based on UN/CEFACT - XML Naming and Design Rules.

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">*

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 21-1 provides details on the namespace patterns and files associated with namespace prefixes.

Table 21-1 Namespace Prefixes

Prefixes	Namespace Pattern	Files
corecomcust	http://xmlns.oracle.com/EnterpriseObjects/Core/Custom/Common/V1	CustomCommonComponents, CustomReferenceComponents

Prefixes	Namespace Pattern	Files
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
telcocomcust	http://xmlns.oracle.com/EnterpriseObjects/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

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 21-2 provides a list of short names and abbreviations used for participating applications.

Table 21-2 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

Application	Short Name	Abbreviation
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

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*

Composite Business Process

Composite business processes are long running SOA BPEL orchestration processes.

Table 21-3 provides details about the naming standards for composite business processes.

Table 21-3 Naming Standards for Composite Business Processes

Artifact	Name	Example
Service Name (in the WSDL)	[Optional Industry] [Verb] [EBO Name]CBP	OrchestrateSalesOrderCBP, TelcoResolveComplaintCBP

Artifact	Name	Example
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

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 21-4 provides details about the naming standards for Enterprise Business Services.

Table 21-4 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

Artifact	Name	Example
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

Enterprise Business Flows

Table 21-5 provides details about the naming standards for BPEL flows used primarily to interact with multiple Enterprise Business Services.

Table 21-5 Naming Standards for Enterprise Business Flows

Artifact	Name	Example
Service Name (in the WSDL)	[Optional Industry] [Verb][EBO Name]EBF	ProcessOrderEBF, TelcoVerifyCustomerEBF
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

Artifact	Name	Example
WSDL Port Type Operations	[Verb][EBO Name]	ProcessOrder
BPEL Flow Name	[Service Name]V[Version Number]	ProcessOrderEBF, TelcoVerifyCustomerEBFV2

Application Business Connector Service

Application Business Connector Services are of two categories: Requester and Provider.

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 21-6 provides details about the naming standards for requester ABCS.

Table 21-6 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, …
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

Artifact	Name	Example
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

Provider Application Business Connector Services

Provider ABCS are the implementation of Enterprise Business Services.

Table 21-7 provides details about the naming standards for provider ABCS.

Table 21-7 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
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

Artifact	Name	Example
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

JMS and Adapters

Table 21-8 provides details about the naming standards for JMS and Adapters.

Table 21-8 Naming Standards for JMS and Adapters

Artifact	Name	Example
JMS Store, JMS Server, JMS Module	Use the default configuration created by SOA Core Extension	N/A
JNDI Name for Queue or Topic Connection Factory	eis/wlsjms/ AIA<Application Name>CFOReis/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/AIAPeopleSoftCFFor XA: eis/wlsjms/AIAPeopleSoftXACFOracle SOA Core Extension creates the JMS module level connection factory automatically by appending "jms/aia" to the given connection factory name. Example:jms/aia/AIAPeopleSoftCF

Artifact	Name	Example
Queue or Topic Name	AIA_<ApplicationName><EBO / ABO Name><Optional Industry>JMSQueueV<Version Number >AIA_<ApplicationName><EBO / ABO Name><Optional Industry>JMSTopicV<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 SOA Core Extension creates the JNDI names for queue or topic under JMS module during deployment by appending "jms/ aia". Example: jms/aia/ AIA_SiebelCustomerPartyJMSTopicV1
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

AQ JMS (Additional Attributes)

Table 21-9 provides details about the naming standards for AQ JMS additional attributes.

Table 21-9 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

Adapter Services Naming

Table 21-10 provides details about the naming standards for adapter services.

Table 21-10 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

Participating Application Service

Table 21-11 provides details about the naming standards for participating application services.

Table 21-11 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

DVMs and Cross References

This section includes the naming standards for:

- DVMs
- Cross References

DVMs

When creating DVMs, the following naming standards should be followed:

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*

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*

Cross References

When creating cross-reference virtual tables in the cross reference tables, the following naming standard should be followed:

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*

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*

BPEL

This section discusses naming standards for:

- BPEL Activities
- Other BPEL Artifacts

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

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.

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*

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*

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*

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)*

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*

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*

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*

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*

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*

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*

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*

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*

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*

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.

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*

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*

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*

Other BPEL Artifacts

Follow these guidelines for other BPEL artifacts:

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*.

Properties

Property names follow the general BPEL variables naming standards.

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*

Correlation Set Properties

The correlation set property names follows the general BPEL variables naming standards.

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.xy.example.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.

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*.

Deployment Plans

Table 21-12 provides details about the naming standards for Deployment Plans.

Table 21-12 Naming Standards for Deployment Plans

Artifact Name	Artifact File Name	Purpose	Mandatory / Optional	Details
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.

22

Editing Transformations Using Mapping Editor

This chapter discusses the process for editing transformations using the Mapping Editor.

This chapter includes the following sections:

Overview of Mapping Editor

The Mapping Editor is a browser-based application that enables you to customize prebuilt integration mappings. Changes are saved as layered customizations, which protects them from being overridden when applying patches or upgrades. The application also enables you to merge the customizations and deploy the impacted composite services to the runtime environment.

Administering the Mapping Editor

This section discusses the following Mapping Editor administration tasks:

Enabling the Mapping Editor

To enable the Mapping Editor, the **AIAMappingCustomizer** Enterprise Role must be assigned to the appropriate user(s).

The Oracle WebLogic Administrator must enable the role to the users.

The AIAMappingCustomizer Enterprise Role becomes available in the Oracle WebLogic console after the AIAHome application is deployed.

To assign the AIAMappingCustomizer Enterprise Role to a user:

1. Open Oracle WebLogic Server Administration Console.
2. Navigate to **Security Realms**, **myrealm**, **Users and Groups** and select the user.
3. In the **Settings for <User>** screen, select **Groups** tab.
4. Move **AIAMappingCustomizer** from **Available** to **Chosen**.
5. Click the **Save** button.

   The Mapping Editor panel becomes available in the Oracle SOA Core Extension console for that user.

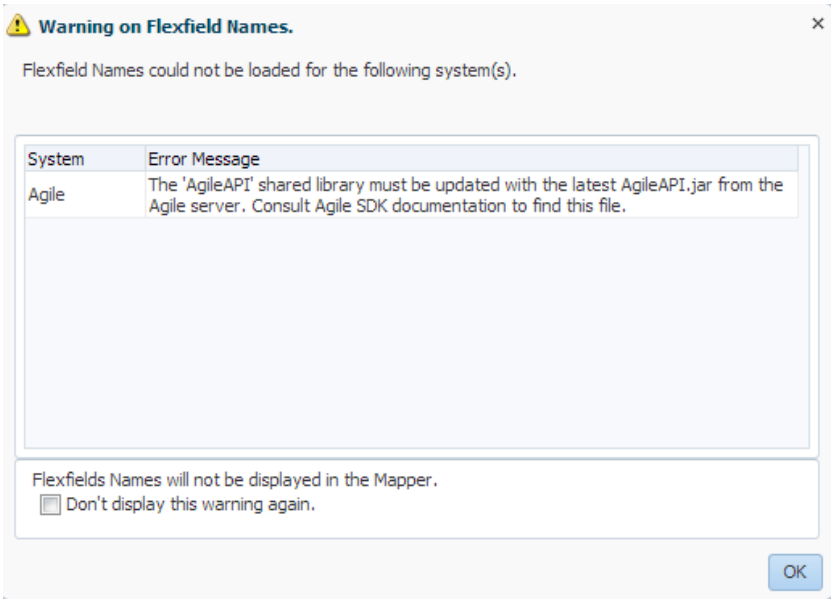
Deploying the AgileAPI.jar File as a Shared Library

Deploy the AgileAPI.jar as a shared library to enable users to retrieve user-defined names for Agile flexfields and display them for editing on the Mapping Editor page as described in **Show Flexfield Names** in Table 22-2.

The Mapping Editor uses the `AgileAPI.jar` file to retrieve the flexfield names for transformations whose source or target system is Agile.

Out-of-the-box, a stub version of the `AgileAPI.jar` file is deployed on the SOA server under the AgileAPI 1.0, 0.0 shared library. However, at this point, if you try to edit an Agile XSLT file in Mapping Editor, the **Warning on Flexfield Names** error displays, as shown in Figure 22-1.

Figure 22-1 Warning on Flexfield Names Error



Select the **Don't display this warning again** option if you don't want this error to display again during the current session for the system(s) listed in the **Warning on Flexfield Names** dialog.

To enable flexfield name functionality, you must deploy the `AgileAPI.jar` file as a shared library to the WebLogic server. The `AgileAPI.jar` file uses the `wlsauth.jar` and `crypto.jar` files. These three JAR files must be available at runtime. In addition, the `wlsauth.jar` and `crypto.jar` files must be a part of the classpath when you start the SOA server.

You can access the `AgileAPI.jar`, `wlsauth.jar`, and `crypto.jar` files from the following locations on the Agile server:

- `<AGILE_HOME>/integration/sdk/lib/AgileAPI.jar`
- `<AGILE_HOME>/<AGILE_DOMAIN>/lib/wlsauth.jar`
- `<AGILE_HOME>/<AGILE_DOMAIN>/lib/crypto.jar`

The location of *<AGILE_HOME>* depends on your installation.

The Mapping Editor supports only AgileAPI.jar file versions that are binary-compatible with Agile 9.3.1.2, 9.3.2, and 9.3.3 versions.

For more information about using Agile APIs, see the *SDK Developer Guide - Using Agile APIs* available in the Agile PLM 9 Documentation library.

To deploy the AgileAPI.jar file as a shared library:

1. Deploy the `AgileAPI.jar` file as a shared library in the WebLogic server.

   The deployment sequence can be `370`. The deployment sequence must be prior to `505`.

   For information about how to deploy a shared library, see "Deploying Shared Java EE Libraries and Dependent Applications" in *Oracle Fusion Middleware Developing Applications for Oracle WebLogic Server*.

2. Redeploy the `AIAHomeApp.ear` file.

3. To make the `wlsauth.jar` and `crypto.jar` a part of the classpath used by the SOA server at runtime, access the `setDomainEnv.sh` file located in your Fusion Middleware domain: `<FMW_HOME>/user_projects/domains/soainfa/bin`.

   Search for the declaration of the `PRE_CLASSPATH` variable. Append the absolute paths to the wslauth.jar and crypto.jar files to the `PRE_CLASSPATH`.

   For your reference, this is the default `PRE_CLASSPATH` configuration:

   ```
   PRE_CLASSPATH="${SOA_ORACLE_HOME}/soa/modules/user-patch.jar${CLASSPATHSEP}${SOA_ORACLE_HOME}/soa/modules/soa-startup.jar${CLASSPATHSEP}${PRE_CLASSPATH}"
   ```

4. Save the `setDomainEnv.sh` file.

5. Restart the SOA server.

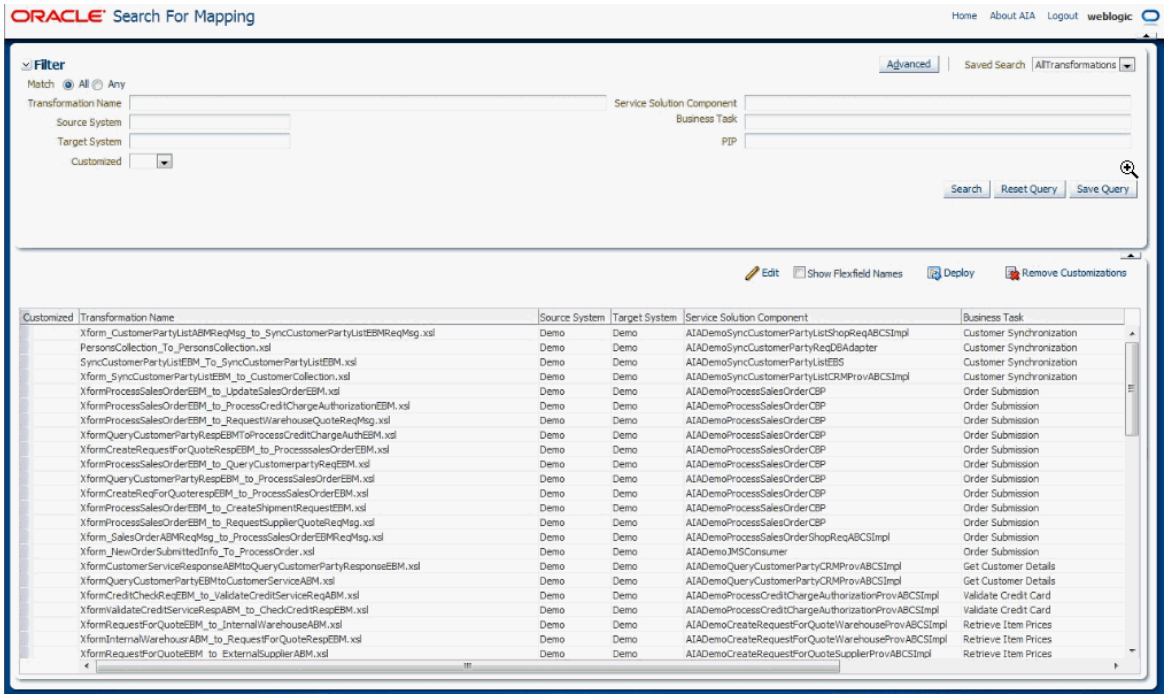
Users can now view user-defined flexfield names for Agile transformations on the Search For Mapping Page, as well as edit Agile transformations on the Mapping Editor page.

Working with the Search For Mapping Page

To launch the Search For Mapping page, navigate to the Oracle SOA Core Extension console and click the **Go** button in **AIA Mapping Editor** section.

The **Search For Mapping** page is displayed as illustrated in Figure 22-2. This page has two sections. The top section enables you to enter search criteria. The bottom section lists transformation files (.xsl) that are available to customize. Transformation files contain the individual mappings that convert data from the source to target in the integration. The list shown is derived from the *MapperConfiguration.xml* file.

Figure 22-2 Search For Mapping Page



Use the filters provided in the **Filter** area to help you locate the transformation file you want to customize. Table 22-1 lists the filters and their descriptions.

Table 22-1 Filters Area on the Search For Mapping Page

Field	Description
Match	Enables you to retrieve transformations based on All or Any of the search parameters.
Transformation Name	Name of the transformation file (.xsl) in the file system.
Service Solution Component	Name of the Service Solution Component in Project Lifecycle Workbench.
Source System	The originating application system.
Business Task	Name of the Business Task in Project Lifecycle Workbench.
Target System	The receiving application system.
PIP	Name of the Process Integration Pack (prebuilt integration provided by Oracle) in Project Lifecycle Workbench.
Customized	Filters transformations based on whether they have been customized or not.

Enter search parameters in any of the **Filter** fields and click **Search**.

Use the controls describes in Table 22-2 to act upon mapping search results.

Table 22-2 Controls on the Search For Mapping Page

Control	Description
Edit	Click to edit the selected transformation to apply custom mappings.
Show Flexfield Names	Used in combination with the **Edit** button. When selected, the system dynamically queries the source or target application metadata to retrieve user-defined names for flexfields and display them as the *business alias* on the Mapping Editor page. **Note:** E-Business Suite and Agile are the supported applications in this release. If you get a **Warning on Flexfield Names** message when editing Agile mappings, see Deploying the AgileAPI.jar File as a Shared Library. If you get a warning message when editing E-Business Suite mappings, then you should troubleshoot using the warning message content and SOA server logs.
Deploy	After you customize the mappings, click to merge the customization layer with the base file and deploy the impacted composite services to the runtime server.
Remove Customizations	Click to remove all customized mappings from the selected transformation.

Editing Transformations

To edit the transformations:

1. Select a transformation.
2. Optionally, if the source or target system is E-Business Suite or Agile, check the **Show Flexfield Names** option to view user-defined names for flexfields.

   Note:

   When you select this option, loading the mappings may take longer.

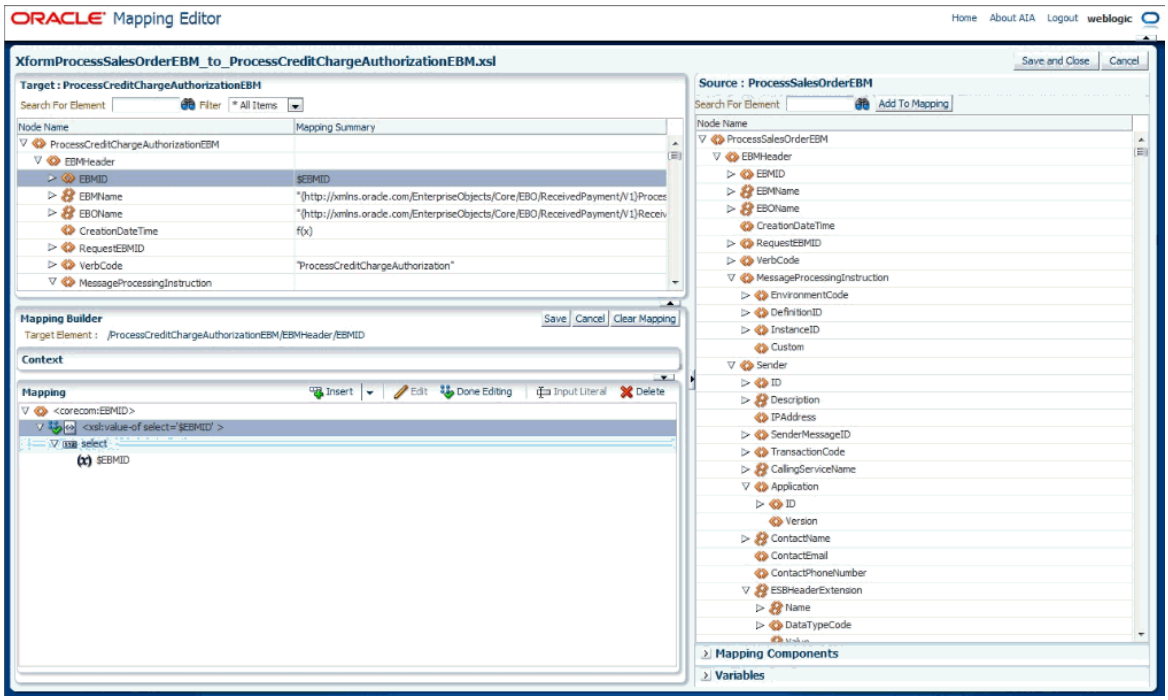
3. Click the **Edit** button.

   The **Mapping Editor** page is displayed.

Mapping Editor Page

The Mapping Editor page, as shown in Figure 22-3, enables you to view existing mappings and extend them. The Mapping Editor is preloaded with existing mappings from the selected Transformation. It enables you to add, change, and delete individual mappings. The Mapping Editor page is divided into multiple frames.

Figure 22-3 Mapping Editor Page



Page Header

- Displays the name of the transformation file (.xsl) being edited.
- **Save and Close:** This button saves mapping changes to the customization layer, and returns to the **Search for Mapping** page.
- **Cancel:** This button returns you to the Search for Mapping page without saving mapping changes.

Target Frame

The first frame you see on the top left is the Target frame. It has target schema name and displays target schema elements. The Mapping Editor is designed to enable you to build the *output document*. The page is arranged this way so that you select target elements on the left and drag and drop elements to build the mapping from the right.

- Target elements are displayed in tree structure. Elements whose children are only attributes are not initially expanded. Expand them to view attribute elements. The tree is initially expanded up to the sixth child. If the sixth child has children underneath it, those children are no longer expanded. The MaxDepth value can be configured in the MapperConfiguration.xml file.
- **Search for Element** enables you to search for an element in the tree. Type the name of the element to search and press **Enter** or click the icon to initiate the search. The next element that contains the entered text is retrieved. Keep pressing **Enter** or click the icon multiple times to find the next matching elements.
- **All Items**, **Mapped** and **Not Mapped** options enable you to filter elements.
  - **All Items:** Displays all the elements of the Target schema.

  - **Mapped:** Filters the tree to display only those Target elements that are mapped. When a child element is mapped, its parent is displayed to keep the Target schema structure intact.

  - **Not Mapped:** Filters the tree to display only those Target elements that are not currently mapped. When a child element is unmapped, its parent is displayed to keep the Target schema structure intact even if the parent is mapped.

- Right click on any element to **Show as Top**, **Collapse All**, or **Expand All**.

- This frame is divided into two columns. They are **Node Name** and **Mapping Summary**.

  - **Mapping Summary** column is the summary of what has been mapped to the target element. It displays Source elements, Variables, Text and Functions. When a Function is used in mapping, it normally contains a few parameters which involve a combination of Source elements, Variables, Texts, Operators and so on. Since the Mapping Summary column has a limited length which cannot accommodate all the different combinations used in a function, an abbreviated format is displayed to convey that a function was used and the parameters passed to it.

  - Target Elements with customized mappings are marked with a blue dot indicator.

  - When you select the target element, the complete expression is displayed in **Mapping** section of **Mapping Builder** frame.

Source Frame

This frame on the right displays three sections inside an accordion control: **Source Schema**, **Mapping Components** and **Variables**. All the three sections in the frame have search fields.

- **Source Schema**

  - Source elements are displayed in tree structure. Elements whose children are only attributes are not initially expanded. You can expand to view attribute elements under them. The tree is initially expanded up to the sixth child. If the sixth child has children underneath it, those children are no longer expanded. The MaxDepth value can be configured in the MapperConfiguration.xml file.

  - Source elements have check marks to indicate that those elements have already been mapped to target.

  - Search for Element enables you to search for an element in the tree. After typing the name of the element to search, press Enter or click on the icon to initiate the search. The next element which contains the entered text is retrieved. Keep pressing Enter or click the icon multiple times to find the next matching elements.

  - You can drag and drop Source elements to an unmapped Target element or into the Mapping builder. Alternatively, you can click the Add To Mapping button to insert the selected source element into the Mapping builder.

Note:

When Source elements are inserted into a mapping, the appropriate relative or absolute path is automatically calculated for you based on the current execution context.

- **Mapping Components:** Mapping Components consist of XSL Elements, Functions and Operators.
- **XSL Elements** are a subset of the most frequently used elements in the XSL language. All Mapping Statements must start with an XSL Element. The following elements are supported:
  - **Attribute:** Use this when an attribute is needed to complete the mapping. For example while setting the nil attribute to `true`.
  - **Choose:** Use this to specify a condition with multiple cases. The `When` element is automatically inserted along with `Choose`.
  - **For-each:** Use this to map a repeating source element to a repeating target element. For-each statements change the execution context, which may result in automatic adjustment of the relative paths of Source and Variables inside child statements.
  - **If:** Use this to specify a condition with a single case.
  - **Otherwise:** Use this statement to specify the final condition under a `Choose` statement.
  - **Text:** Use this to specify literal values.
  - **Value-of:** Use this primary statement to perform the actual mapping assignment.
  - **When:** Use this to specify a condition under a `Choose` statement. If `Choose` is not already a parent statement, than it is automatically inserted along with `When`.
- **Functions** consist of standard XPATH functions such as Date functions, advanced functions, string functions, and so on. There are also frequently used AIA functions such as DVM, XREF and so on.
- **Operators** consist of Addition, And, Division, Greater Than, Less Than and so on.
- The **Mapping Component** section also supports **Search** and **Add To Mapping**.
- **Variables:** Displays variables that can be used for a selected Target element. Variables are context sensitive.
  - When a Target element is first selected, any applicable variable is displayed only if the Target element has not been mapped. It can be dragged and dropped to the Target element as a value-of select statement.
  - Variables are also displayed when a row has been selected in the Mapping panel. As long as the variable is applicable for the selected Mapping row, it is displayed.

– Certain Variables can have a tree structure. You can expand these Variables to see its children and continue expanding until it reaches a leaf node.

Note:

When variables elements are inserted into a mapping, the appropriate relative or absolute path is automatically calculated for you based on the current execution context.

– The Variables section also supports **Search** and **Add To Mapping**.

Mapping Builder Frame

This frame provides a mapping building utility that enables users to create and edit complex mappings. This frame is on the bottom left. This frame has the following sections.

Mapping Builder Header

- **Target Element:** Displays the full path to the selected target element (above in the target frame).
- **Save:** Saves the changes made in the Mapping Builder to an internal document in memory. Changes must be saved in one target element before selecting another target element.
- **Cancel:** Discards the changes made in the Mapping Builder. The target mapping reverts back to its previous state.

  Clear Mapping: Erases all the mappings made on the Target element.

Context:

Displays relevant contextual information to help the user understand under which conditions and code paths the currently selected target element shall be mapped. Context is not editable. The execution context may include apply templates, match templates, call templates, for-each, choose/when/otherwise and if statements that must be satisfied for the mapping to be executed. This pane is initially collapsed. Click the Restore Pane icon for it to display its content.

Mapping

Displays the complete collection of statements used in the mapping of a Target element. The mapping includes the mapping assignment itself as well as conditional statements which affect the target. Users can insert, edit or delete statements using the following toolbar options:

- **Insert:** Provides options to `Insert Parent`, `Insert Sibling Before`, `Insert Sibling After` and `Insert Child` statements in the **Mapping** pane. These options are also available in a context menu when a user right clicks on a statement in the pane.
- **Edit:** Enables the user to expand the selected statement to display subcomponents that make up the statement. Once expanded, these subcomponents can then be edited. This button and the **Edit** option in the context menu are only enabled for statements that can be edited. For example value-of select, if, when and so on. Editable statements in the pane are preceded with the pencil icon.

- **Done Editing:** This option is enabled after the **Edit** button is clicked and the selected statement is expanded. After you complete making changes to any of the subcomponents, click this button to complete the change and collapse the statement. Clicking the **Save** button automatically clicks the **Done Editing** button.
- **Input Literal:** This option is enabled when an expanded subcomponent contains an expression that can be modified to a literal.
- **Delete:** This option is enabled when a selected statement or expression is eligible for deletion.

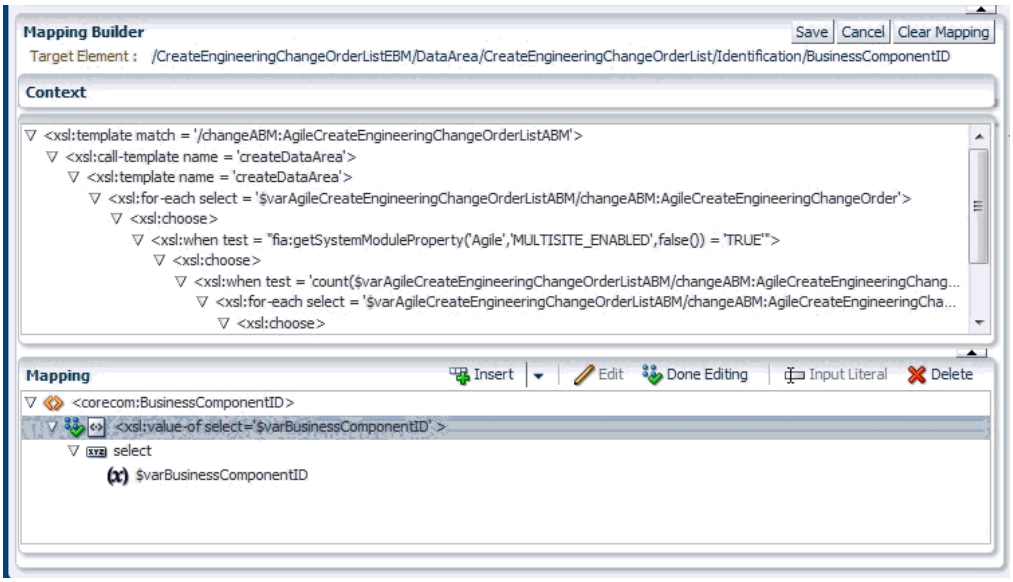
Building Mappings (Examples)

This section, using simple examples, shows you how to use the Mapping Editor page.

Building Simple Mappings

You can do simple mapping when no complex conditions are required. For example, you need to map a product ID in source system to a product ID in target system and no conditions are involved, do the simple mapping.

Figure 22-4 Mapping Expression Builder Screen



To do a simple mapping of elements:

1. Select the element in target frame that need to be mapped.
2. Select the element you want to map in the source frame.
3. Drag the source element into the target element row that needs to be mapped.

   Mapping summary column of that row gets refreshed with the source element and the row is highlighted with a blue dot indicating that the element is customized.

   Mapping Builder frame is refreshed with the complete mapping statement for the simple mapping just created. The `value-of` statement, which is the XSL equivalent of a simple assignment, is automatically added for you.

   You can also map a Variable into the target this way. However, Mapping Components can only be mapped using the Mapping builder.

Note:

To remove the mapping that you have just made, select the element in the target frame and click the **Clear Mappings** button.

Building Complex Mappings

To do more complex mapping that requires conditions, functions, and so on, use Mapping builder.

To create mapping using Mapping Builder:

1. Select the element in the **Target** frame.
   - If the selected element is unmapped, an empty row in the Mapping section is added with the text *Drag and Drop or Type here*.
   - If the selected element is mapped, the complete set of mapping statements for that target is displayed in the Mapping section. You can create an empty row and add a new statement or edit an existing statement before continuing. For more information, see *To add a new mapping statement row* below.
2. Select and drag the source element and drop it into the empty row in the Mapping section.
   - The `value-of` statement, which is the XSL equivalent of a simple assignment, is added for you.
   - Alternatively you can select a source element and click the **Add to Mapping Expression** button.
3. Click the **Done Editing** button once you are done with changes.
4. Click **Save**.

Adding a New Mapping Statement Row

To edit a mapping statement row:

1. Select the row you want to edit in the **Mapping** section.
2. Click **Edit** and make necessary changes.

   This statement is expanded to display its subcomponents. Edit existing subcomponents by inserting Source, Function, Operator, or Variables onto the existing subcomponent row.
3. Click **Done Editing** button once you are done with changes.
4. Click **Save**.

To save all mapping changes:

Once you are done with all the edits, click **Save and Close** button. This takes you back to **Transformations** search page. This page helps you deploy the customized transformations.

To discard all mapping changes, click the **Cancel** button.

Refer to Editing Rules for Mapping Editor section before you proceed with editing expressions.

Editing a Mapping Statement Row

To add a new mapping statement row:

1. Click the **Insert** button and choose either a parent, sibling before, sibling after, or child row.

   An empty row is inserted in the desired position with the text `Drag and Drop or Type` here.

2. From the Mapping Components section, drag and drop an XSL Element onto the new row. For example, select `if`.

3. This statement is expanded to display the following required subcomponents.

   - A child row for **Test** which is a required attribute of the **if** XSL Element.
   - A new empty child row under **Test** has been defaulted for you with the text `Drag and Drop or Type here… for Test`.

4. Complete the new XSL Element by inserting a valid Source, Function, Operator, or Variable onto the new empty child row. If a Function/Operator is inserted, it is once again expanded to display its required subcomponents. Complete them too.

5. Click the **Done Editing** button once you are done with changes.

6. Click **Save**.

Editing Rules for Mapping Editor

This section discusses rules for editing transformations. This section also discusses in detail editing rules for each field.

Definitions

- **Empty Row:** A row without a Target Tag, Source element, Mapping Component, Variable or Literal in it. Empty row says “-- Available for D&D or Type ..."
- **Target tag:** XML sentence defining the tag for a Target Element.
- **Attribute-Value Template Mapping:** The expressions shown in the Expression Builder window for an Attribute-Value Template.
- **Regular Mapping:** A mapping that is not for Attribute-Value Template.
- **Mapping Component:** A component listed under Mapping Components accordion panel.
- **Sentence:** An XML Element or an XML complete sentence.
- **Ancestor Sentence:** Parent, grand parent or grand-grand parent or so on sentence excluding shared sentences.
- **Sub-component:** Part of a sentence or part of a subcomponent.
- **Shared Sentence:** An XSLT sentence which has two or more sibling target elements under its scope.
- **OOTB:** Out Of The Box.

Read-only Sentences

The following sentences are read-only:

- xsl:copy
- xsl:copy-of
- xsl:element
- xsl:attribute
- xsl:for-each
- Target tag with attribute-value template
- An OOTB sentence with a function or operation not listed on the Mapping Components panel

Drag and Drop or Typing

Table 22-3 lists various Drag and Drop (D & D) or typing activities that you can do while editing transformations.

Table 22-3 Drag and Drop or Typing

			D&D	D&D	D&D	D&D	D&D	D&D	Type
Mapping Type	**Parent**	**Destination**	**Source Element**	**XSLT Element**	**XSLT Attribute**	**Function**	**Operator**	**Variable**	**Literal**
N/A	N/A	Unmapped Target Element In Target Tree	Yes	No	No	No	No	Yes	No
N/A	N/A	Mapped Target Element In Target Tree	No	No	No	No	No	No	No
Regular	Target tag	Empty Row	Yes	Yes	No	No	No	Yes	Yes
Regular	XSLT Element	Empty Row	No	Yes	Yes	No	No	No	(1) Yes
Regular	XSLT Attribute	Empty Row	Yes	No	No	Yes	Yes	Yes	Yes
Regular	Function	Empty Row	Yes	No	No	Yes	Yes	Yes	Yes
Regular	Operator	Empty Row	Yes	No	No	Yes	Yes	Yes	Yes
Regular	Literal	Empty Row	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible

			D&D	D&D	D&D	D&D	D&D	D&D	Type
Mapping Type	**Parent**	**Destination**	**Source Element**	**XSLT Element**	**XSLT Attribute**	**Function**	**Operator**	**Variable**	**Literal**
Attribute-Value Template	Any	Empty Row	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible
Any	Any	Target Tag	No	No	No	No	No	No	No
Regular	Any	Source Element	Yes	No	No	Yes	Yes	Yes	Yes
Attribute-Value-Template	Attribute	Source Element	Yes	No	No	No	No	Yes	Yes
Regular	Any	Variable	Yes	No	No	Yes	Yes	Yes	Yes
Attribute-Value-Template	Attribute	Variable	Yes	No	No	No	No	Yes	Yes
Regular	Any	XSLT Element	No	No	No	No	No	No	No
Regular	Any	XSLT Attribute	No	No	No	No	No	No	No
Regular	Any	Function	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Regular	Any	Operator	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Regular	Target	Literal	No	No	No	No	No	No	Yes
Regular	Text XSL Element	Literal	No	No	No	No	No	No	Yes
Regular	Select attribute of value-of XSL Element	Literal	Yes	No	No	Yes	Yes	Yes	Yes
Attribute-Value-Template	Any	Function	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible
Attribute-Value-Template	Any	Operator	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible
Attribute-Value-Template	Any	Literal	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible	(2)Not possible
Any	Any	Any Read Only	No	No	No	No	No	No	No

(1) Yes: Valid for <xsl:text>. Other xslt sentences will error at runtime

(2) Not Possible: User cannot create an empty row for that parent

Note:

When you add more than one child element with different attributes, the Mapping Editor saves both the child elements. When you add two child elements with the same attribute, the Mapping Editor saves only one attribute. This is the expected behavior.

Inserting a Row

The Table 22-4 discusses situations when a new row can be inserted.

Table 22-4 Rules while Inserting a Row

			Insert as	Insert as	Insert as	Insert as
Mapping Type	**Parent of a Selected Row**	**Selected Row**	**Parent**	**Child**	**Sibling Before**	**Sibling After**
Regular	None	Any	Yes	Yes	No	No
Regular	Any	Target tag	Yes	Yes	Yes	Yes
Regular	Any	XSLT Element	Yes	Yes	Yes	Yes
Regular	XSLT Element	XSLT Attribute	No	(1) Yes	Yes	Yes
Regular	Any	Function	No	Yes	No	No
Regular	Any	Operator	No	Yes	No	No
Regular	Sub-component	Literal as subcomponent	No	No	No	No
Regular	XML Sentence	Literal as value of a XML Sentence	Yes	No	No	No
Regular	Any	Source Element	No	No	No	No
Regular	Any	Variable	No	No	No	No
Attribute-Value Template	Any	Any	No	No	No	No

(1) Yes: Valid for <xsl:text>. Other xslt sentences will error at runtime

Deleting

This section discusses rules while deleting transformations.

Deleting all Mapping rules (Remove All)

- When the target element is a leaf element, target tag and all child sentences are deleted including variable declarations.
- When the target element is a parent element, mapping editor does not allow the parent to be deleted until all child target mappings are deleted. You must delete child first and then parent.
- When the Expressions UI shows sentences which are ancestors of the target element tag, the ancestor sentences are not shared with other target and that is the reason they are shown. These ancestor sentences are along with ancestor variables are deleted too.

Deleting an individual saved sentence

- Individual saved sentences excludes sentences for “Attribute-Value Template sentences".
- Deleting whole mappings have precedence over deleting individual sentences.
- A sentence that is created but not saved can be deleted. No rules apply for these.
- If a sentence to be deleted defines the source execution context (for example, for each), it will be temporally flagged read only, so cannot be deleted.

Deleting Sentences in Target Element Tag

- If the target tag sentence is a Parent Target tag, and the Child Target(s) have mapping(s) then it cannot be deleted.
- If the target tag sentence has children sentences it cannot be deleted.
- If the target tag has ancestor sentences, it cannot be deleted.

  Target tag should be the only sentences without any child, parent or ancestor to be deleted.

Deleting Attribute-Value Templates

Example of a Attribute-Value template: deleting action="CREATE" of <changeEBO:CreateEngineeringChangeOrderList actionCode="CREATE"> sentence.

Attribute-Value template can be deleted as a whole mapping. Individual rows cannot be deleted.

Deleting Sub-Components

- Deleting an Attribute, Parameter, Operand: Only optional sub-components can be deleted. Required subcomponents cannot be deleted.
- Deleting a Function, Operator, Literal, Source Xpath, Variable Xpath: They can be deleted.

Editing Rules for For-Each

This section discusses editing rules for for-each.

Definitions

- **Execution Content:** Execution context is a scope concept. When an execution context is defined, each child XSL sentence inherits the execution context. The execution context is mainly defined by the following XSLT sentences:
  - <xsl:template match='xxxx'>
  - <xsl:apply-template select='xxx'/>
  - <xsl:for-each select='xxx'/>

    “xxx" is the XPath expression defining the execution context.
- **Execution Context Definer:** An XSL sentence that defines an execution context.
- **Execution context based on the Payload or a Variable:** An execution context can be defined with a path (relative/absolute) to the payload (source) or to a Variable. The XPath expressions with relative paths access the information from the current execution context. In other words, when the execution context is based on a variable, the relative paths access the variable content and not the payload content.
- **Execution Context Dependent (ECD) sentence:** An ECD sentence is a sentence with an XPath which has any relative paths.

  Attribute-Value Template is not XSLT. However, for this document they are considered as ECD sentence if the value assigned to the attribute uses a relative path.
- **Unknown Execution Context:** It happens when a sentence which defines the Execution Context (for-each, apply-template and so on.) has an invalid XPath or a valid XPath but is using a complex XPath which the mapper cannot decipher. The Execution Context is flagged as Unknown for that sentence and its scope.
- **Not Editable Sentence:** A sentence which is either not displayed on the Expression builder, like:
  - Variable declarations
  - Template invocations
  - Template declarations
  - A ReadOnly sentence
  - A sentence with wildcard XPath (//xxx)

Allowing Changing Execution Context

- **Out of the Box (OOTB) for-each:** If the OOTB for-each scope involves Not Editable ECD sentences, the <xsl:for-each> sentence is set as ReadOnly.

  If the OOTB for-each has an Unknown Execution Context, the <xsl:for-each> sentence is set as ReadOnly.
- **OOTB Unknown Execution Context:** When a context definer sentence defines an Unknown Execution Context, all ECD child sentences are set as Read Only.

- **Drag and Drop <xsl:for-each> or Editing an existing <xsl:for-each>:** A <xsl:for-each> can be dragged and dropped or edited as long as:
  - There is no Not Editable ECD sentence under the scope of the new for each. If there is any Not Editable ECD sentence, an error pops-up.

    When inserting a <xsl:for-each> as an ancestor sentence of the tag sentence of a Parent Target, then the mapping sentences for the target's child mappings are within the scope of the new for-each. Those sentences are also evaluated as Not Editable ECD sentence.
  - There is no child sentence in edit mode. If a child sentence is in edit mode, an error pops-up.
- Removing< xsl:for-each>: An xsl:for-each can be removed except when:
  - It is Read Only.
  - The <xsl-for-each> scope involves Not Editable ECD sentences.

Adjusting Relative Paths when Execution Context Changes

When a <xsl:for-each> is added, changed or deleted the execution context is changed.

When the Execution Context changes the ECD child sentences (sentences under the new <xsl:for-each> scope> are adjusted to keep accessing the same data as before the new context was introduced. Mapper will try to keep relative paths as much as possible.

When the new execution context changes from payload to a variable (or backwards) or from a variable to another variable, the ECD child sentences are adjusted to use absolute paths in order to keep them accessing the same data as before the execution context change.

When the new execution context changes from one location to another within the same payload or variable, the ECD child sentences are adjusted keeping relativity paths. This is done by using the XPath navigation commands ("../", ".", "/")

Sentences with absolute paths are not adjusted on an execution context change.

For ECD sentences displayed on the Expression Builder, a user sees the relative paths adjusted immediately. The ECD child sentences for child target mappings are auto-adjusted when the **Save** button is clicked.

The Table 22-5 lists a few examples of relative path getting adjusted when execution context changes.

Table 22-5 Examples of Relative Path Getting Adjusted when Execution Context Changes

Execution Context changed from -> to (next row)	XPath to employee first name is adjusted from -> to (next row)
/company/employee/employee/name	first_name
/company/employees/employee	name/first_name
$department	/company/employee/employee/name/first_name

/company/employees/employee	/company/employee/employee/name/first_name
Execution Context changed from -> to (next row)	**XPath to Company's phone number name is adjusted from -> to (next row)**
/company/employee/employee/name	../../../phone_number
/company/employees/employee	../../phone_number
$department	/company/phone_number
/company	/company/phone_number
Execution Context changed from -> to (next row)	**Absolute XPath to Company's city address. It is not adjusted.**
/company/employee/employee/name	/company/address/city
/company/employees/employee	/company/address/city
$department	/company/address/city
Execution Context changed from -> to (next row)	**XPath to Department Name from the variable $department. It is not adjusted because it is an absolute path.**
/company/employee/employee/name	$department/name
/company/employees/employee	$department/name
$department	$department/name

Understanding Customization Layer

- There is no need to put custom code in the _Custom.xsl which was recommended previously as the Mapping Editor leverages MDS Layered Customizations.
- Using the Mapping Editor, you can now modify the existing transformation and changes are saved in the new mdssys folder.
- The mdssys folder contains subfolders with *mapperui*. This subfolder contains an xml file whose name matches the xsl that was modified. For example, AgileCreateEngineeringChangeOrderListABM_to_CreateEngineeringChangeOrderListEBM.xsl.xml.
- The xml file contains the mapping changes made to the original xsl file. These mapping changes are marked with unique xml:id="mapperui_xxx. The mappings in this file are merged with the out-of-the-box and displayed accordingly in the Mapping Editor.
- The mappings in the Customized Layer are not affected during pre-built integration upgrade or while applying patches.

Deploying Edited Transformations

The transformations that are customized have a mark in the customized column. To deploy the customized transformation, select the transformation and click the **Deploy** button.

The system does the following when you click the **Deploy** button:

1. Merges the Customization Layer with the base XSL file to create a Merged file.
2. Generates a custom Deployment Plan for the selected composite services.
3. Runs the custom Deployment Plan using the SOA Core Extension AID utility, which redeploys the selected composite(s) to the runtime server.

Removing Customizations

To undo the customization that you have done to the transformation, select the customized transformation and click the **Remove Customizations** button.

A

Delivered Oracle AIA XPath Functions

This appendix provides details about the XPath functions that are delivered with the Oracle SOA Core Extension 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.

aia:getSystemProperty()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.
  xpath.AIAFunctions"
string aia:getSystemProperty (string propertyName, boolean needAnException)
```

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.

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.

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>
```

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)
```

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.

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.

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>
```

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)
```

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.

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.

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>
```

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)
```

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 Managing the Oracle AIA System Registry.

Returns

Returns a node set of system information, entered in the Oracle AIA system registry for the given System Code or Internal ID.

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>
```

aia:getSystemType()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
string aia:getSystemType (string systemCode)
```

Parameters

`systemCode`: System code as entered in the Oracle AIA system registry.

For information about maintaining the Oracle AIA system registry, see Managing the Oracle AIA System Registry.

Returns

Returns the System Type associated with a System Code as entered in the Oracle AIA system registry.

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>
```

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)
```

Parameters

- `errorCode`

  Error code.

- `localeString`

  Delimited locale string.

- `Delimiter`

  Delimiter used in the localeString.

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.

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>
```

aia:getCorrectiveAction()

```
namespace aia="g aia:getCorrectiveAction (string correctiveActionCode,
  string localeString, string delimiter)
```

Parameters

- `correctiveActionCode`

  Corrective action code.

- `localeString`

  Delimited locale string.

- `Delimiter`

  Delimiter used in the localeString.

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.

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>
```

aia:isTraceLoggingEnabled()

```
namespace aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
  core.xpath.AIAFunctions"
string aia:isTraceLoggingEnabled (string logLevel, string processName)
```

Parameters

- `logLevel`

  Log level that you wish to log.

- `processName`

  Name of the process.

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.

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>
```

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.

Parameters

- `ebmHeader`

  The Enterprise Business Message (EBM) header providing context for the error message.

- `Message`

  Message to log.

Returns

Returns an empty string.

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>
```

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)

Parameters

- `Level`

  Level of the trace message.

- `ebmHeader`

  The EBM Header providing context for the trace message.

- `Message`

  Message to log.

Returns

Returns an empty string.

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>
```

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)
```

Parameters

- `systemId`

  System ID.

- `errorCode`

  Error code.

- `serviceName`

  Name of the errored service.

- `processName`

  Name of end-to-end process.

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 Managing the Oracle AIA System Registry.

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>
```

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)
```

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 runtime 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>
```

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.

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>
```

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)
```

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.

Returns

This function returns the converted date as string without time zone in yyyy-MM-dd'T'HH:mm:ss.SSS format.

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>
  ```

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)
```

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.

Returns

This function returns the converted date as string without time zone in the input date format.

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:

AddTargetSystemID.xsl

Example B-1 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 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>
```

SetCAVSEndpoint.xsl

Example B-2 provides XSL text that should be used to develop CAVS-enabled requester ABCSs.

For more information about how to use this XSL, see 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>
```