Oracle® Application Integration Architecture - Foundation Pack 2.5: Integration Developer's Guide

Release 2.5

Part No. E16465-01

December 2009

Oracle Application Integration Architecture - Foundation Pack 2.5: Integration Developer's Guide

Part No. E16465-01

Contents

Preface

The *Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide* discusses how to*:*

- Create an integration scenario.
- Define business service patterns.
- Design and develop EBSs.
- Design and develop enterprise business flows (EBFs).
- Design and construct ABC services.
- Work with message transformation, enrichment, and configuration.
- Develop custom XPath functions.
- Design and construct JMS Adapter services.
- Work with EBM headers.
- Work with message routing.
- Work with transactions.
- Develop Oracle AIA services to work with the CAVS.
- Configure Oracle AIA processes to be eligible for error handling and logging.
- Extend EBOs.
- Work with the Event Aggregation programming model.
- Work with the Publish-and-Subscribe programming model.

In addition, this guide provides Oracle AIA naming standards.

The *Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide* is a companion volume to these guides and resources discussed in this preface:

- The *Oracle Application Integration Architecture – Foundation Pack: Concepts and Technologies Guide*
- The *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide*
- Process integration pack (PIP) implementation guides
- Additional resources

Oracle Application Integration Architecture – Foundation Pack: Concepts and Technologies Guide

See the *Oracle Application Integration Architecture - Foundation Pack: Concepts and Technologies Guide* for definitions of fundamental Oracle AIA concepts and information about:

- Oracle AIA.
- Enterprise business objects (EBOs) and enterprise business messages (EBMs).
- Enterprise business services (EBSs).
- Application business connector (ABC) services.
- Interaction patterns.
- Extensibility.
- Versioning.
- Business processes.
- Batch processing.
- Infrastructure services.
- Security

Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide

See the *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide* for information about how to:

- Work with the Composite Application Validation System (CAVS).
- Work with the Business Service Repository (BSR).
- Set up and use error handling and logging.
- Work with the diagnostics framework.
- Work with Oracle AIA developer tools.

Oracle AIA PIP Implementation Guides

A PIP is a pre-built set of integrated orchestration flows, application integration logic, and extensible EBOs and EBSs required to manage the state and execution of a defined set of activities or tasks between specific Oracle applications associated with a given process.

A PIP provides everything you need to deploy a selected integrated business process area. The PIP product offering is suited to those customers seeking to rapidly implement a discreet business process.

Implementation guides are available for each PIP.

Additional Resources

These resources are also available:

Resource	Location
Oracle Application Integration Architecture - Installation and Upgrade Guide	My Oracle Support: https://metalink.oracle.com/
Known issues, workarounds, and most current list of patches	My Oracle Support: https://metalink.oracle.com/
Release notes	Oracle Technology Network: http://www.oracle.com/technology/
Documentation updates	My Oracle Support: https://metalink.oracle.com/

Chapter 1: Getting Started with the Oracle AIA Integration Developer's Guide

This chapter discusses:

- The goal of this document.
- Types of integrations addressed by Oracle Application Integration Architecture (AIA).

Goal of this document

This document defines in detail how the Oracle AIA Foundation Pack can be used to develop process integration packs (PIPs) confirming to the Oracle AIA Reference Architecture in a structured fashion based on a functional design document provided.

This document can also be used to implement custom PIPs for customers or additional functionalities in the form of new services extending the Oracle AIA PIP functionalities.

Following this document is key to ensure upgradability, supportability, and maintainability of the Oracle AIA solution built/implemented.

Types of Integrations Addressed by Oracle AIA

Oracle AIA addresses two types of integrations:

- Functional integration

  In a functional integration, the various functionalities of different participating applications, exposed as services, are weaved together in the form of processes to accomplish business tasks spanning multiple applications in any enterprise.

- Data integration

  In data integration, data in one application is moved to another application.

Part 1: Functional Integration

Chapter 2: Building an Oracle AIA Integration Scenario

This chapter discusses:

- Defining an Oracle Application Integration Architecture (AIA) integration scenario.
- Prerequisites for an Oracle AIA integration scenario.
- Designing an Oracle AIA integration scenario.
- Developing an Oracle AIA integration scenario.
- Testing an Oracle AIA integration scenario.

Defining an Oracle AIA Integration Scenario

Before you start building integration between applications, we recommend that you review the steps outlined in this chapter to design, develop, and deploy the integration. By creating an Oracle AIA integration scenario you will develop a logical process or flow for accomplishing a functional task spanning multiple applications. A functional task could be customer information retrieval, order creation and submission for processing, or invoice generation and payment processing.

The integration scenario also involves the identification and construction of the various Oracle AIA artifacts that are required for the collaboration between these applications or functions. These artifacts include application business connector (ABC) services, enterprise business services (EBSs), and enterprise business flows (EBFs).

Prerequisites for an Oracle AIA Integration Scenario

The completion of a functional design document (FDD) is the most important prerequisite for designing and developing an Oracle AIA integration scenario. The FDD should produce this output:

- Identification of the integration scenario
- Identification of the EBO(s)

Identifying the Integration Scenario

The FDD for the Oracle AIA integration scenario should include:

- Detailed use cases.

The use cases should list in detail the various usage scenarios including the exception cases with expected actions by various actors.

- Reference to the relevant activity diagrams found in Industry Reference Models.
- Details about all the participating applications.
- Details on the triggering business events.
- Details on the functional flow.
- Details about performance and scalability requirements.

Identifying the EBOs

The FDD should discuss:

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

For more information, see Extending EBOs.

Designing an Oracle AIA Integration Scenario

The design of an integration scenario 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, domain value map (DVM) types and values, error handling, and installation specifics. It also includes an outline of the unit test plans that the developer will use to validate runtime operation.

The integration scenario is not a runtime executable so a message exchange pattern cannot be attributed to the integration scenario. The design of the Oracle AIA service artifacts will need a careful analysis of the business scenario. This will lead to the decision on the message exchange patterns for each of the Oracle AIA service artifacts.

For more information about message exchange pattern decisions, see Establishing the Message Exchange Pattern of a New Process EBS, Choosing the Message Exchange Pattern for Requestor ABC Services, Choosing the Message Exchange Pattern for Provider ABC Services, or Identify the Message Pattern for 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 scenario:

1. Analyze the participating application business messages (ABMs) and map them to the enterprise business message (EBM).

   For more information, see *Oracle Application Integration Architecture – Foundation Pack: Concepts and Technologies Guide*, Understanding Enterprise Business Objects and Enterprise Business Messages

2. Identify the EBSs.

   See Chapter 3: Designing and Developing EBSs.

3. Identify the EBFs.

   See Chapter 5: Designing and Constructing EBFs.

4. Identify the ABC services.

   See Chapter 4: Designing and Constructing ABC Services.

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

   See Chapter 3: Designing and Developing EBSs.

   For more information, see *Oracle Application Integration Architecture – Foundation Pack: Concepts and Technologies Guide*, Understanding Interaction Patterns

6. Identify and decide on the security model.

   See Chapter 16: Understanding Security.

   For more information, see *Oracle Application Integration Architecture – Foundation Pack: Concepts and Technologies Guide*, Understanding Security

7. Identify and decide on the performance and scalability needs.
8. Identify and decide on the deployment strategy.

Developing an Oracle AIA Integration Scenario

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

- EBSs
- EBFs
- ABC services

Since an Oracle AIA service artifact can be part of multiple Oracle AIA integration scenarios, it is important to go through the Business Service Repository (BSR) and identify any service artifact that you could reuse. The Oracle AIA service artifacts are built with reusability in mind.

For each of the artifacts, these reusability guidelines should be followed:

- Reusability guidelines for EBSs
- Reusability guidelines for EBFs
- Reusability guidelines for ABC services

To develop an Oracle AIA integration scenario:

1. Identify and create the EBS, if needed.
2. Enable the participating applications.
3. Construct the ABC service.
4. Construct the EBF.

Each of these tasks is discussed in these sections.

For more information about the standard naming conventions see Appendix A: Oracle AIA Naming Standards.

Identifying and Creating the EBS

Each of the EBOs will have an EBS to expose the “Create, Read, Update, Delete” (CRUD) operations as well as any other supporting operations. Each of the actions present in the associated EBM will be implemented as a service operation in EBS. Creation of the EBS and the implementation of all of the service operations will be one of the critical tasks in the implementation of the end-to-end integration scenario.

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 more than one business object should naturally reside in the process EBS.

In situations where the EBS exists, check whether all of the actions necessary for acting on the EBO pertaining to this integration scenario have been implemented as service operations; if not, make changes to the existing EBS to add the additional service operations.

This table lists the high-level tasks to construct an entity based EBS. Chapter 3: Designing and Developing EBSs provides complete information about 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 Composite Application Validation System (CAVS).

For more information, see Chapter 3: Designing and Developing EBSs.

Constructing the ABC Services

For more information, see Chapter 4: Designing and Constructing ABC Services.

Enabling the Participating Applications

To enable the participating applications:

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

Identifying and Creating the EBF

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

For more information, see Chapter 5: Designing and Constructing EBFs.

Testing an Oracle AIA Integration Scenario

For more information, see Chapter 12: Developing Oracle AIA Services to be CAVS-Enabled and Provide Multi-Instance Support.

Chapter 3: Designing and Developing EBSs

This chapter discusses how to:

- Establish the need for a new enterprise business service (EBS).
- Establish the message exchange pattern of new Process EBSs.
- Define the contract for Process EBSs.
- Construct the WSDL for the Process EBS.
- Develop the EBS.
- Implement synchronous request - reply message exchange patterns in EBS.
- Implement asynchronous message exchange patterns with one-way calls in EBS.
- Handle errors in asynchronous message exchange patterns.

For more information, see Appendix A: Oracle AIA Naming Standards.

Establishing the Need for a new EBS

The Oracle Application Integration Architecture (AIA) Foundation Pack is shipped with EBS library. The EBS library is comprised of the service definitions delivered for majority of the Entity Business Objects present in the Enterprise Object Library. These are shipped as WSDL files. The service operations present in the EBS typically are the “Create, Read, Update, Delete” (CRUD) operations plus some of the operations specific to entities.

Review the sample WSDLs provided in the Oracle AIA Foundation Pack under the AIAComponents/EnterpriseServiceLibrary folder.

All the operations of the Entity EBS WSDLs 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.

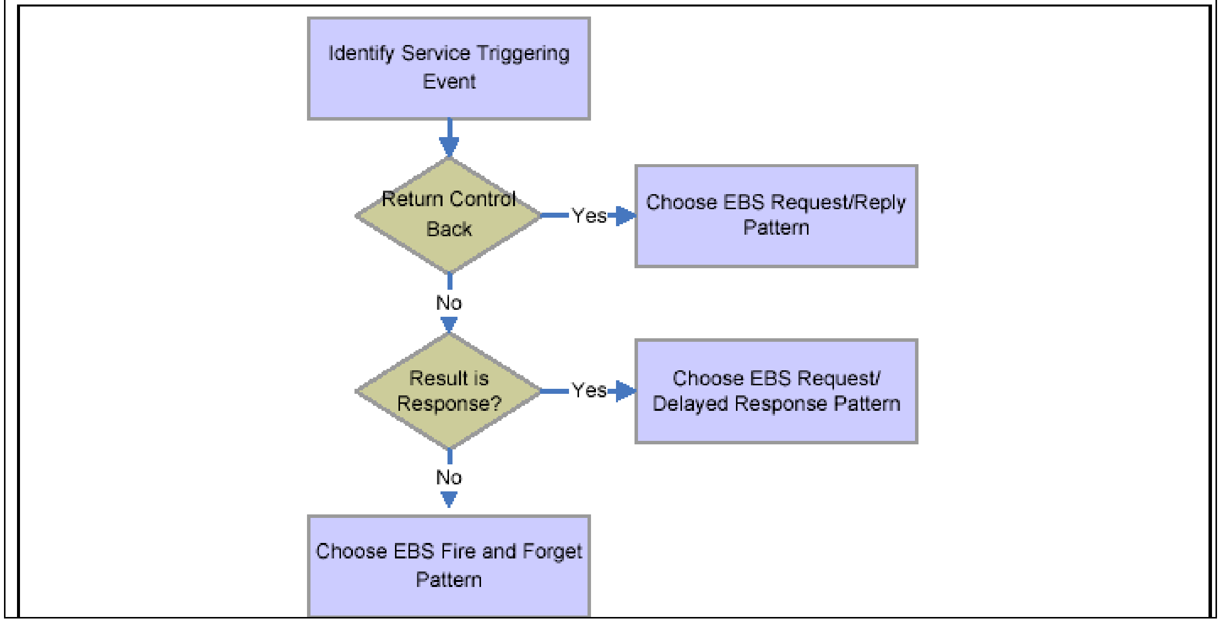
The developer needs to look at WSDLs and operations available there in, and then decide the need for a new EBS. Review each of the WSDLs, the operation's description, and the metadata before deciding to create either a new service or an operation. Any new EBS created will be a Process EBS with operations put in to meet the requirements of the business scenario for which an Oracle AIA Integration Scenario is being developed. These would be put in a different WSDL and not added to the Entity EBS WSDLs.

Establishing the Message Exchange Pattern of a New Process EBS

As mentioned in the previous section, the message exchange patterns for the Entity EBS WSDLs in the EBS library of the Oracle AIA Foundation Pack are pre-decided as shipped.

So the decision is for the Process EBS WSDLs. The EBS is modeled to have multiple operations. Each operation leads to 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.

This diagram illustrates the decision points in establishing the EBS pattern.


Identify Service Triggering Event
Return Control Back
Yes
Choose EBS Request/Reply Pattern
No
Result is Response?
Yes
Choose EBS Request/ Delayed Response Pattern
No
Choose EBS Fire and Forget Pattern

Identifying the interaction pattern for EBS operations

1. Based on the understanding of the business process requirement from the functional design, identify the triggering event for the EBS operation.
2. If the control is to be blocked until a response is returned back to the point of invocation, then choose the EBS request-reply pattern.

   This is a synchronous call. In this case, the EBS operation will have 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 on, this invocation of the EBS would be an asynchronous call.
4. Next check whether the execution of the EBS results in a response. Is there a need to correlate the request and the response?

   If the answer is yes, this is a delayed response. Use the EBS request-delayed response pattern. In this case, the EBS will have two portTypes - each of them accepting an input message only and each of them belonging to a different port.

5. If the answer is no, then choose the EBS fire-and-forget pattern.

In this case, the EBS operation will have an input message only.

The methodology for designing and implementing an EBS is contract first methodology, that is, the contract is defined and created prior to the implementation of the EBS. The contract for an EBS will be defined as a WSDL document.

For the Entity EBS, use the WSDLs from the Enterprise Service Library of the Foundation Pack.

For the Process EBS, use the TemplateEBS.wsdl available in the **Foundation Pack** and create a Process EBS WSDL. Customers wanting to create a new EBS for an EBO that was not delivered as part of Enterprise Object Library can use TemplateEBS.wsdl as a model to create the new WSDL.

- Service Operations supporting synchronous request/response message exchange pattern will be defined in one port type – Operation will have input, output and fault message defined.
- Service Operations supporting fire-and-forget message exchange pattern will be defined in one port type – operation will have input message.
- Service operations supporting asynchronous request/delayed response pattern will have two operations – one operation for sending request message and another operation for processing the response message.

  Each of these two operations will have input message alone. Both of them will NOT reside within a single port type. The service operation for processing the response message will reside in a port type having 'Response' as the suffix.

The EBS WSDLs will have two kinds of portTypes:

- PortType for all operations used for modeling Synchronous Request - Response operations and Request Only operations. **The name will not specify the 'Request'.**
- PortType for Asynchronous Response operations. **The name will specify 'Response'.**

There will be two ESB Routing Services created for each of the portType as will be described in later sections.

Defining the Contract

To define the contract for the EBS:

1. Identify the EBS and the operations it has to 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.

For more information, review the *AIAServices-TemplateSampleWSDLs* provided as part of the Foundation Pack under the AIA Samples folder.

Constructing the WSDL for the Process EBS

This section provides an overview of the WSDL and discusses how to:

- Completing the <definitions> section
- Defining message structures
- Checking for WS-I Basic Profile conformance

Understanding the WSDL for the Process EBS

When constructing the WSDL, keep these in mind:

- Oracle's JDeveloper, as well as text editing tools, can be used to create the WSDL.
- The enterprise business message (EBM) schema module for the Enterprise Business Object (EBO) needs to be referenced in the WSDL.
- The WSDL should reference the schema modules hosted on the central location.

  Each EBS should not have its local copy of the EBOs.
- Annotations should be done based on the recommendations provided in the next sections.
- You should adhere to the naming conventions for creating service, target namespace, messages, operations, port type, and so on.

For more information about naming standards, see Appendix A: Oracle AIA Naming Standards.

These sections describe how to fill in sections of the WSDL. Refer to the TemplateEBS.wsdl and SampleEBS.wsdl in the Foundation Pack build.

Completing the <definitions> Section

Use these guidelines to complete the <definitions> section:

- Name

  The service contains operations related to creation and maintenance as well as actions to be performed on an EBO.
- Namespace

  This namespace should be marked as the target namespace.
- Namespace prefixes

  Define a namespace prefix for the newly identified namespace,

  "wsdl" will be the namespace prefix for the default name space "xsd" will be the namespace prefix for XMLSchema "soap" will be the namespace prefix for SOAP namespace

A namespace prefix needs to be defined for the EBO. The namespace prefix needs to be the same as that defined by the EBO team.

Defining Message Structures

Use these guidelines to define the message structures in the WSDL types section.

- Import the appropriate schema that has all of the relevant EBMs defined.
- Use xsd:import to import the elements from a schema.

  Make sure that the namespace attribute for xsd:import element is the same as the target namespace for the schema document that is imported.

  Also, make sure that the namespace prefix is declared for that namespace so that the elements in the imported schema can be referenced using the namespace prefix.
- The attribute targetNamespace for the xsd:schema element should be the same as the targetNamespace for this WSDL.

Message Definitions

For every operation you need to 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.

Port Type Definition

The port type name should be the same as the service name. A port type name should be defined for each of the operations. The message names specified for input and output elements were defined in the message definitions section based on the pattern. When services are deployed, ESB 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. It is very important that the authors of the WSDL annotate the sections thoroughly. The annotations serve as the source of truth for populating the metadata in the Business Service Repository (BSR).

Checking for WS-I Basic Profile Conformance

The WS-I Basic Profile consists of a set of non-proprietary 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.

The WSDL templates provided ensure conformance so it is important to follow them for development of Oracle AIA artifacts.

Developing the EBS

The EBS serves these purposes:

- Provide the mediation between the requesting services and providing services.
- Provide different operations for being invoked from a requester application business connector (ABC) service, an EBS, or an enterprise business flow (EBF).
- Based on the evaluation of the various routing rules for an operation, route it to a suitable EBS, EBF, or Provider ABC service.

Oracle AIA leverages Enterprise Service Bus (ESB) technology available in Oracle SOA Suite to build the EBS. The EBS is implemented as an ESB routing service. An ESB 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 creating ESB projects in JDeveloper, see *Oracle Enterprise Service Bus Developer's Guide*.

The operations of the EBS can either be modeled as a synchronous message exchange pattern or asynchronous message exchange pattern. These sections describe the steps in detail.

- Create a new routing service using the WSDL for the EBS.
- Configure the routing rules for each of the EBS operations.
- Enable error handling and logging

Implementing Synchronous Request – Reply Message Exchange Patterns in EBS

The initiator for a synchronous request – reply pattern would be a requesting service waiting/expecting a response. The requesting service can be a participating application, requestor ABC service 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.

For enabling the ABC services (both requester and provider) and EBF, refer the relevant sections.

To implement synchronous request-reply message exchange patterns in EBS:

1. Create ESB Projects with Routing Services.

2. Create Routing Rules to route the request from the requesting service to correct providing service in the routing service of the EBS.
3. Implement error handling for logging and notification based on fault policies.

Creating ESB Projects

Follow these guidelines when creating ESB projects:

- Create two ESB 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 will reside in only one port type so there would be only one ESB Routing Service.
    - If the EBS has at least one asynchronous request/response operation, then there would be two port types –two ESB Routing Services and two ESB Projects (one for each routing service).
- Follow the naming convention detailed in Appendix A: Oracle AIA Naming Standards.

  Typical names for the ESB projects will be as:

    - **CustomerPartyEBSV2** (This will have a routing service with all operations for synchronous request – response and request – only)
    - **CustomerPartyEBSResponseV2** (This will have a routing service with all operations for asynchronous request – response)

Creating Routing Services

To create a routing service:

1. Put the EBS WSDL in the ESB project folder.
2. Create a routing service and name as per the naming convention detailed in Appendix A: Oracle AIA Naming Standards**.**
3. Select the WSDL.

   The WSDL will 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 will have all the operations specified in that portType in the EBS WSDL.

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 almost all cases, routing logic should be performed in the EBSs routing rules.

  However all routing rules in the EBS should check for and respect existing Target system ID(s) that have already been stamped in the header. EBS rules should not assume the target system ID has already been populated.
- Requestor ABC service should not determine target system(s) or stamp target system ID(s) in the EBM header.
- For any EBS operation, each possible target application system instance requires a routing rule.

  For example, if there are two Siebel provider application system instances, SEBL_01 and SEBL_02, then there must be a routing rule for each, even though both rules will target the same Siebel provider ABC service.

  An alternative to this is that 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 have to 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 ABC service that is shared between the multiple instances will be able to identify which instance to invoke and Xref to.
- If an EBS operation is a synchronous request-reply pattern or async request-delayed-response pattern, then the routing rules must be mutually exclusive given the actual topology of the Oracle AIA system.
- Customers buying process integration packs (PIPs) will see the routing rules delivered as part of ESB Routing services.

  These rules are meant to work for the delivered topology. If a customer implements any changes to the delivered topology, such as adding an additional system instance, then they will need to implement their own complete set of routing rules.
- The standard Routing Rule clause structure will be:
  - (cavs_check) and (ruleset_check) and ((target_system_identified_check) or ((target_system_absent_check) and (topology_specific_clauses))

Clause	XPath expression
`cavs_check) =`	`MessageProcessingInstruction/EnvironmentCode='PRODUCTION' or`

Clause	XPath expression
	`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'`

This table shows some of the routing rules delivered as part of Integrated Supply Chain Management PIP:

O2C2 Delivered Routing Rules for Create, Sync, Update, Get, Query,and so on		
1	*Target:*	Siebel provider ABC service
	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 already specified as Siebel, or else no Target is specified and the Sender application type is not Siebel.
2	*Target:*	Oracle EBusiness provider ABC service
	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 already specified as EBiz, or else no Target is specified and the Sender application type is not EBiz.
3	*Target:*	CAVS
	XPath Filter:	`MessageProcessingInstruction/EnvironmentCode='CAVS'`
	Transformation:	None
	Explanation:	MessageProcessingInstruction/EnvironmentCode='CAVS'

Implementing Error Handling

For more information, see Chapter 13: Configuring Oracle AIA Processes for Error Handling and Trace Logging.

Implementing Asynchronous Message Exchange Patterns with One-Way Calls in EBS

The implementation of both the asynchronous message exchange patterns – fire-and-forget and request – delayed response, need the creation of EBS WSDLs followed by creation of one or two ESB Routing Services depending upon message exchange pattern. This should be followed by the implementation of the requestor and provider services adhering to the guidelines laid out for the respective message exchange patterns.

The requesting services can be Requestor ABC service (BPEL process), EBF (BPEL process) or a participating edge application.

The providing services can be Provider ABC service (BPEL Process), EBF (BPEL process) or a participating edge application.

This section discusses:

- Implementing the Fire-and-Forget Patterns with One-Way Calls in EBSs
- Implementing the Request – Delayed Response Pattern with One-Way Calls in EBS

Implementing Fire-and-Forget Pattern With One-Way Calls in EBSs

The initiator for a fire-and-forget pattern would be a requesting service not waiting/expecting a response. The requesting service can be a participating application, Requester ABC service Impl or an EBF. In each of these cases, the request payload would be an EBM request.

For enabling the ABC service (both requester and provider) and EBF, please refer to the relevant sections.

To implement the fire-and-forget pattern with the EBS one-way calls:

1. Create EBS WSDLs.
2. Create ESB 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 Foundation Pack.

For the Process EBS, use the TemplateEBS.wsdl available in the Foundation Pack and create a Process EBS wsdl.

- Service Operations supporting synchronous request/response message exchange pattern will be defined in one port type – Operation will have input, output, and fault message defined.
- Service Operations supporting fire-and-forget message exchange pattern will be defined in one port type – operation will have input message.
- Service operations supporting asynchronous request/response pattern will have two operations – one operation for sending the request message and another operation for processing the response message. Each of these two operations will have an input message alone. Both of them will NOT reside within a single port type. The service operation for processing the response message will reside in a port type having 'Response' as the suffix.
- The EBS WSDLs will have two portTypes:
    - PortType for all operations used for modeling Synchronous Request - Response operations and Request Only operations. **The name will not specify the 'Request'.**
    - PortType for Asynchronous Response operations. **The name will specify 'Response'.**
- There will be two ESB Routing Services created for each of the portType.

Creating ESB Routing Service for Asynchronous Fire-and-Forget Patterns with One-Way Call EBS

To create ESB Routing Services:

1. Create ESB Projects.
2. Create Routing Services.
3. Create Routing Rules.

Create ESB Projects

Follow these guidelines when creating ESB projects:

- Create two ESB 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 will reside in only one port type so there would be only one ESB Routing Service.
    - If the EBS has at least one asynchronous request/response operation, then there would

be two port types – two ESB Routing Services and two ESB Projects (one for each of the routing service).

- Follow the naming conventions detailed in Appendix A: Oracle AIA Naming Standards.

  Typical names for the ESB projects will be as:

  - **CustomerPartyEBSV2** (This will have a routing service with all operations for Synchronous Request – Response and Request – Only)
  - **CustomerPartyEBSResponseV2** (This will have a routing service with all operations for Asynchronous Request – Response)

Creating Routing Services

To create routing services:

1. Put the EBS WSDL in the ESB project folder.
2. Create a routing service and name as per the naming convention detailed in Appendix A: Oracle AIA Naming Standards.
3. Select the WSDL.

   The WSDL will 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 will have all the operations specified in that portType in the EBS WSDL.

Creating Routing Rules

The routing rules for request EBS are the same as those for the Sync Request – Response section.

Implementing Error Handling

For more information, see Chapter 13: Configuring Oracle AIA Processes for Error Handling and Trace Logging.

Implementing the Request – Delayed Response Pattern with One-Way Calls in EBS

The initiator of the Request – Delayed Response pattern would be a requesting service. It would invoke the request EBS and wait for a response. The requesting service can be a participating application, Requester ABC service Impl or an EBF. In each of these cases, the request payload would be an EBM request. The response payload would be EBM response.

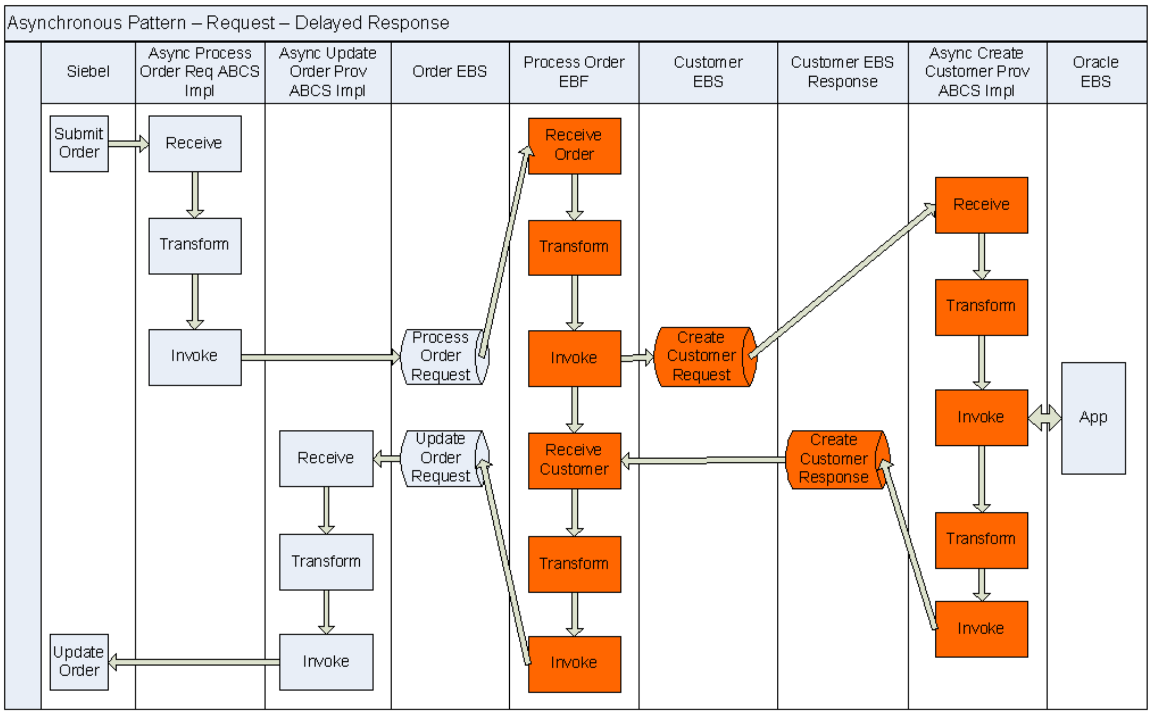
In case of an error in the providing service, a response message with error information will be constructed and returned to the requesting service for action.

For enabling the ABC service (both requester and provider) and EBF, please refer to the relevant sections in this guide.

To implement the Request – Delayed Response pattern with the two one-way calls of the EBS, These are the high level steps to be performed in addition to the regular steps required for the requesting service and the providing service:

1. Create EBS WSDLs.
2. Create ESB Routing Services for asynchronous request – delayed response patterns with two one-way call EBSs.
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.
5. Route the response message in the EBS response to the correct requesting service.

This diagram illustrates the Request – Delayed Response pattern:



Request – Delayed Response pattern

Creating EBS WSDLs

For the Entity EBS, use the WSDLs from the Enterprise Service Library of the Foundation Pack.

For the Process EBS, use the TemplateEBS.wsdl available in the Foundation Pack and create a Process EBS WSDL.

The EBS WSDLs will have two portTypes –

- PortType for all operations used for modeling Synchronous Request - Response operations and Request Only operations. **The name will not specify the 'Request'.** Service Operations supporting synchronous request/response message exchange pattern will be defined in one port type – Operation will have input, output and fault message defined.

- PortType for Asynchronous Response operations. **The name will specify 'Response'.** Service operations supporting asynchronous request/response pattern will have two operations – one operation for sending request message and another operation for processing the response message.

There will be two ESB Routing Services created for each of the portType.

Creating ESB Routing Services for Asynchronous Request – Delayed Response Patterns with two One-Way Call EBS

These are the high-level steps for creating ESB Routing Services:

- Create ESB Projects.
- Create Routing Services.
- Create Routing Rules.

Create ESB Projects

Create two ESB 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 will reside in only one port type – hence, there would be only one ESB Routing Service.
- If the EBS has at least one asynchronous request/response operation, then there will be two port types – two ESB Routing Services and two ESB Projects (one for each routing service).
- Follow the naming convention detailed in Appendix A: Oracle AIA Naming Standards.

Typical names for the ESB projects will be as:

- **CustomerPartyEBSV2** (This will have a routing service with all operations for Synchronous Request – Response and Request – Only)
- **CustomerPartyEBSResponseV2** (This will have a routing service with all operations for Asynchronous Request – Response)

Create Routing Services

To create a routing service:

1. Put the EBS WSDL created in the ESB project folder.
2. Create a routing service and name as per the naming convention detailed in Appendix A: Oracle AIA Naming Standards.
3. Select the WSDL.

The WSDL will 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 will have all the operations, including compensate operations, specified in that portType in the EBS WSDL.

The guidelines mentioned here are only the additional instructions pertaining to the implementation of asynchronous message exchanging patterns. See Chapter 3: Designing and Developing EBSs for complete instructions.

Create Routing Rules

For the asynchronous request – delayed response EBS, there will be routing rules for request and response.

Routing Rule for Request EBS

The routing rules for request EBS will be same as that explained in the sync request – response section.

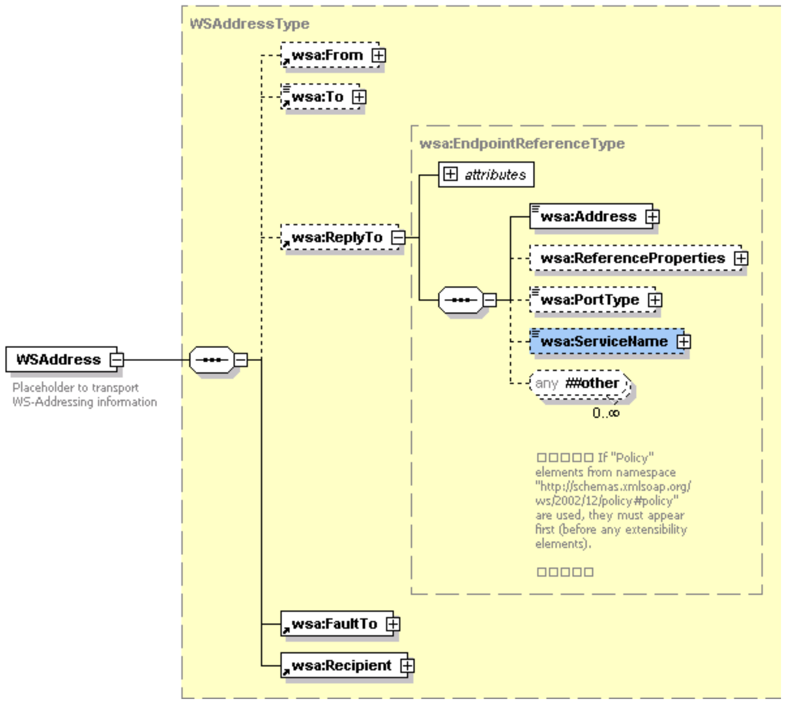
Routing Rule for Response EBS

There will be two routing rules:

- 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 there is a need to route the response to the correct requesting service.

  The EBMHeader/Sender/WSAddress/wsa:ReplyTo/wsa:ServiceName is to be set to the name of the requesting service name in the requesting service -- application business message (ABM) to EBM transformation -- before invoking the request EBS.


WSAddressType
wsa:From
wsa:To
wsa:EndpointReferenceType
attributes
wsa:Address
wsa:ReferenceProperties
wsa:ReplyTo
wsa:PortType
wsa:ServiceName
WSAddress
Placeholder to transport WS-Addressing information
any ##other
0..∞
□□□□□ If "Policy" elements from namespace "http://schemas.xmlsoap.org/ws/2002/12/policy#policy" are used, they must appear first (before any extensibility elements).
□□□□□
wsa:FaultTo
wsa:Recipient

Structure of the WSAddressType element

In the providing service, this information is transferred from the Request EBM to the Response EBM. The methodology is detailed in this guide. 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>
```

Example of the <Requesting Service Name> - CreateOrderSiebelReqABCSImplV2

The target endpoint for the evaluation of this rule should be set to the requesting service.

For every requesting service of the request EBS, waiting for a response EBS to send back response, there will be a routing rule.

- Composite Application Validation System (CAVS) Routing Rule

  The CAVS routing rules will be same as that explained in the Sync Request – Response section.

Implement Error Handling

Please refer to this section “Error Handling in Asynchronous Message Exchange Patterns”.

Error Handling in Asynchronous Message Exchange Patterns

In synchronous request – response message exchange patterns, the requesting services are waiting for response. Whenever there is an error in the provider services, an exception is raised and the fault message is propagated back to the requesting service.

In asynchronous fire-and-forget message exchange pattern, the requesting service does not expect any response. If there is an error in the providing service, there may be a need for compensation. In such situations, the compensatory operations in EBS need to be used for triggering compensations.

In asynchronous request –delayed response message exchange pattern, 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.

Asynchronous Fire-and-Forget Message Exchange Pattern Error Handling Using Compensatory Operations

For the one-way calls, operations can be put in the EBS (not provided out of the box) to trigger compensation in the requesting services to offset the effects of errors in the provider services. This can be achieved by having compensatory operations in the EBS.

Compensatory operations are modeled as one-way calls. These are modeled as separate operations. For each request - only operation in the request portType, there will be an operation for triggering compensation.

For example, CompensateCreateCustomer, CompensateCreateOrder

The Compensatory operations will be invoked in cases where a business exception is likely to result in a fatal error. This is a situation where the conventional retry and resubmit is not possible and the correction is required to be made in the requesting service.

This would call for suitable compensatory services to be implemented taking advantage of the participating applications compensatory action web services or APIs.

Invoking Compensate Operation of EBS

As part of Error Handling, for some errors, there may be a need to ensure the compensatory actions are taken. From the providing service, the compensate operation of the EBS will be invoked. The compensate operation of the EBS will route to the correct compensating service.

To invoke the compensate operation of the EBS:

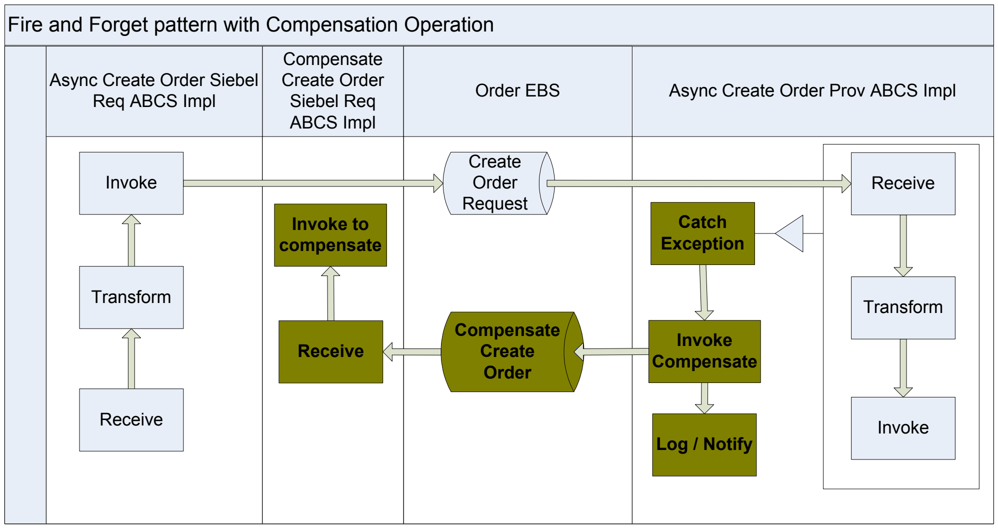
1. In the providing service, in case of an error, raise an exception and catch it in the catch block.
2. In the catch block, construct the Request EBM along with fault component in the EBM header.
    a. Create a transform step and select the input variable representing the Request EBM and the compensate variable, also representing the Request EBM.
    b. Pass the fault message generated from the exception as a variable into the input variable to compensate variable XSLT. Please refer to this guide for the technique.

c. Map to the compensate variable these:

- Standard EBM header content from the Request EBM
- Data Area from the Request EBM
- Fault message

d. Set the 'InvokeCompensate' step to invoke the corresponding compensate operation in the request EBS routing service.

e. Route the compensate request to a suitable compensating service.


Fire and Forget pattern with Compensation Operation
Async Create Order Siebel Req ABCS Impl
Compensate Create Order Siebel Req ABCS Impl
Order EBS
Async Create Order Prov ABCS Impl
Invoke
Transform
Receive
Invoke to compensate
Receive
Create Order Request
Compensate Create Order
Catch Exception
Invoke Compensate
Log / Notify
Receive
Transform
Invoke

Fire-and-forget pattern with compensation operation

Enable Routing Rule in Compensate Operation Routing Service

There will be two routing rules:

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 - CAVS at present does not have feature to accept an asynchronous response for a request initiated from CAVS.

Asynchronous Request – Delayed Response Message Exchange Pattern Error Handling

In the asynchronous request – delayed response message exchange pattern, 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.

Follow these steps:

1. In the providing service, in case of an error, 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.
    a. Create a transform step and select the input variable representing the Request EBM and the output variable representing the Response EBM.
    b. Pass the fault message generated from the exception as a variable into the ‘input variable to fault variable’ XSLT. Please refer to this guide for the technique.
    c. Map to the output variable:
        - Standard EBM header content from the Request EBM including the correlation information
        - Fault message
    d. Set the ‘Invoke’ step to invoke the response operation in the response EBS routing service.
    e. Route the response from the providing service to the correct requesting service.

Chapter 4: Designing and Constructing ABC Services

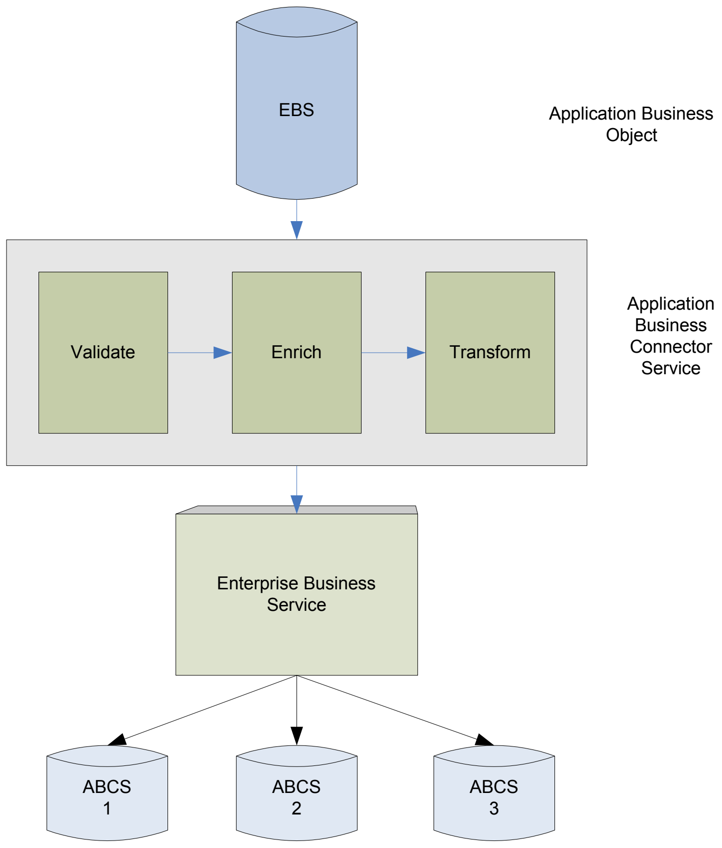
This chapter provides an overview of application business connector (ABC) services and discusses how to:

- Design the ABC service.
- Develop the ABC service contract.
- Construct ABC services.
- Describe verbs used to define enterprise business service (EBS) operations.
- Implement ABC services.
- Work with the Artifacts Generator.
- Interact with applications – inbound and outbound.
- Test ABC services.
- Recognize I18N considerations.

For information about using the Artifacts Generator to autogenerate much of the common code needed to create ABC service implementations in BPEL, see Working with the Artifacts Generator.

Understanding ABC Services

An ABC service provides an implementation of a participating application's API for core functionality and facilitates interaction with the EBS.


EBS
Application Business Object
Validate
Enrich
Transform
Application Business Connector Service
Enterprise Business Service
ABCS 1
ABCS 2
ABCS 3

ABC services

The ABC service allows participating applications to become service providers as well as service consumers. It also allows applications having non-standard connectivity to expose their functionality as web services.

Core Tasks

The core tasks of an ABC service are:

- Establishing connectivity with the participating application

  The mode of ABC service interaction with participating application can be either thru web services, JCA adapters, or queue based. Each mode provides opportunities for implementing various patterns. Absence of some modes also sometimes forces usage of a particular pattern.

- Content validation

In most of the situations validations are performed in the participating applications before submitting the message to the ABC service for processing. However, there will still be situations when certain validations on the message or validation of content of the message need to be performed in the ABC service, upon receipt of the message.

- Message enrichment

  In certain situations, the message received by the ABC service would have references to related entities but the full information may not be there. For example, the Order message has references to Customer but not enough information to create a Customer. In such a situation, the ABC service can query back the participating application to get the information and enrich the message before submitting to EBS.

- Population of enterprise business message (EBM) headers

  In the ABC service, the sender's information, security information, and correlation information are set in EBM headers.

- Message Transformation

  The EBS interfaces are EBMs. In the ABC service, the transformation from the application specific format to application independent format and vice versa is done.

- Invocation of EBS or application services

  The ABC service either presents the request to the EBS or services the request received from the EBS by invoking the application services.

- Manages referencing of instance identifiers and static values

  The Cross Reference utility is used for managing the entity references across applications participating in the integration. The Domain Value Mapping (DVM) utility is used to manage the static values across applications participating in the integration. In the ABC service, XPath functions are used to manage the Cross Reference utility and DVM utility as part of the transformations.

- Internationalization and localization of message content

  In the ABC service, the locale dependent strings for error messages are managed.

ABC Service Types

There are two types of ABC services specified in Oracle Application Integration Architecture (AIA):

- Requestor ABC services
- Provider ABC services

Each participating application will develop an ABC service for each of the business tasks depending on the role it is playing in the integration. The ABC service is deployment agnostic – one ABC service could support multiple instances of an application.

Requestor ABC Service

The Requester ABC service is provided by requesting application to interface with an EBS for performing a single business task. The service interfaces between Requesting/Source application and EBS.

The Request ABC service:

- Is a service provided by the requesting application to invoke the EBS
- Receives application business message (ABM) as payload
- Returns (optional) ABM as the response

The ABM can be enriched by having additional conversations with the Requestor application. The ABM will be transformed into an EBM using XSL. Responses received from EBS (EBM) will be transformed into ABM.

Non-SOAP payloads will be handled by having transport adapters in between requestor. Requestor Applications need to pass system information for the ABC service to have subsequent conversations.

Provider ABC Service

The Provider ABC service is supplied by the provider application to interface with an EBS for performing a single business task. The service interfaces between the EBS and Provider or Target application.

The Provider ABC service:

- Is a service provided by provider application to interface with the provider application
- Receives EBM as payload
- Returns (optional) EBM as the response

The EBM will be transformed into an ABM using XSL. ABM might be enriched by having additional conversations with the provider application. Responses received from application (ABM) will be transformed into EBM

Non-SOAP connectivity with the provider application will be handled by having transport adapters in between the Provider ABC service and the Provider application

Designing the ABC Service

This section provides an overview of the ABC service design process and discusses how to:

- Analyze the functional design document (FDD)
- Analyze the participating application integration capabilities
- Choose the implementation style
- Choose message exchange pattern

Understanding the ABC Service Design Process

The goal of this phase is the creation of the technical design document (TDD).

Key architectural principles to consider:

- ABC service should be absolutely decoupled from the target system.
- Requestor ABC service should make no assumptions about the target system.
- Requestor ABC service could end up invoking a provider ABC service that was not built.
- Requestor ABC service could end up being replaced by another requestor ABC service.
- ABC service is not process integration pack (PIP)-specific.
- ABC service design decisions should not be based on the PIP for which it is currently built.

  For example, EBM population should not be based on a target system or a PIP.
- ABC service is not bound to a specific application version.

  An ABC service can be used to interact with multiple application versions.
- ABC service is transport neutral. ABC service should be communicating inbound as well as outbound only using SOAP and ESB bindings.
- BSR, if used, should be consulted regarding existence of ABC service.
- Provide the necessary input to the development team that might already be currently developing the ABC service to incorporate your requirements.
- Concurrent multiple versions of an ABC service should be avoided.
- More than one service for an application performing a single business activity using a specific role (requestor/provider) is not allowed.
- Multiple teams involved in producing/consuming an ABC service for a specific application and a business activity should come to consensus regarding the contracts.
- Having multiple versions for a single service with semantically and technically incompatible contracts among versions is not acceptable.
  - Version 1 owned by Team A; Team B does not like the contract – creates Version 2 with a different MEP/Contract; Team A creating Version 3 with a contract contradicting the previous one.

Analyzing the Functional Design Document

Analyze the functional design document and clearly identify the integration points as well as the flow scenarios. After completing this step, you should be able to:

- Identify enterprise business objects (EBOs), operations, and EBMs and make sure they are available, and ready for use.
- Identify the participating applications.

- Identify all integration touch points.
- Identify functional flows with a clear understanding of scope, business logic, and algorithms.
- Understand all applicable scenarios/use cases including the ones detailing the exception scenarios.

Analyzing the Participating Application Integration Capabilities

After identifying the participating applications, you need to analyze the integration capabilities of each participating application and how each application exposes its data.

After completing this step, you should be able to:

- Work with the application providers (developers) to decide the best way to interact with the application to achieve the needed functionality. The interactions can be inbound or outbound. The interaction mechanism can be one of these:
  - SOAP WebService
  - Events/Queues
  - JCA
  - Database Adapter
- Obtain relevant details from the applications in situations where the interactions between the participating applications and the ABC service happen through a message queue adapter, database adapter, or JCA Adapter. These details will be used in configuring the adapters. The configuration will result 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.
- Identify the available functionality within the participating applications as well as the mapping between EBMs and application business objects (map operations). There may be a one-to-one, one-to-many, or many-to-one mapping between the two.
- Tabulate the mapping between the EBO attributes/elements and the application objects attributes/elements in a spreadsheet. Keep these points in mind when creating the transformation maps:
- In the case of 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 where the ABM does not have all of the content to populate all of the data elements in the EBM, you need to work with the respective requester application teams to identify the application services that could be used to populate the content in EBM.
  - For example, a Siebel application invoking the CreateOrder ABCs might populate 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 the all the information regarding the Customer. So it becomes the responsibility of the CreateOrder ABCs to invoke 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, it is highly recommended that you work with the participating application to ensure that the payload being passed to the ABC is as close to the shape of the EBM as possible in order to minimize the chattiness.

- Identify any unique requirements or rules that must be followed when interacting with the participating application. Areas of focus could be:

  - Synchronous versus asynchronous

    Identify how the participating applications interact with the ABC services for inbound as well as outbound interactions.

    Whether the requester application needs to be in a call-waiting state until it has received the response from the provider application will influence the mode of invocation. Waiting until the response is received from the provider application will necessitate the requester application to invoke the ABC service 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 agent cannot perform any task until the response is received. On the other hand, 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 may leverage different kinds of integration technologies for synchronous versus asynchronous invocations. You need to select the most appropriate technology based on the situation.

    Note: It is highly recommended that participating applications making an outbound interaction in an asynchronous mode use message queues to publish the requests. This approach is recommended since it allows for high scalability as well as a better end-user experience.
    Participating applications making an outbound interaction in a synchronous mode alone should send the requests using SOAP/HTTP.

  - Error/fault handling

    Determine how faults are handled and passed by the participating application. You need to make sure the application error handling capabilities are in line with the integration platform error handling capabilities as described in detail in the Error Handling chapter.

    For more information, see Chapter 13: Configuring Oracle AIA Processes for Error Handling and Trace Logging.

  - Security credentials

    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 it is a custom mechanism, is that using header or as part of payload?

For more information, see Chapter 16: Understanding Security.

- Application context

  Do you need to set an application context after successful authentication? This may include information such as:

  - Language of operation
  - RunAs role responsibility similar to Oracle eBusiness Suite.
  - Setting up logging and diagnostic switches to ensure end-to-end logging with participating applications logging as well.

Choosing the Implementation Style

To choose the correct ABC service implementation style, research the capabilities of the participating application APIs and map them to the capabilities of the EBS. The complexity of the ABC service will increase with the degree of incompatibility between the application capabilities and the EBS capabilities.

These are common ABC service implementation styles:

One Provider/Requester ABC Implementation Service per EBS Operation

For each EBS, the standard is to have one ABC implementation service for each of the provider applications and one ABC implementation service for each of the requester applications. In this case, there is an implicit assumption that the granularity of the service request made by the requester application can be directly equated with that of the EBS operation; and the granularity of the service provided by the provider application is the same as that specified by EBS operation. An implementation service was chosen per operation for the sake of simplicity. There is no value in combining the implementations of all operations into a single ABC implementation service.

Provider ABC Implementation Service Invoking Multiple Application Operations

There are cases where there is no direct mapping between the participating application API and the EBS operations. In this case the ABC implementation service has to aggregate or disaggregate data depending on the service role as a requester or provider service.

This implementation style is used when:

- The message payload needs to be consolidated from different systems.

  An example is convergent billing, where billing information from different systems is consolidated in one message.

- The provider application API is more granular than that of the EBS operation.

  In this case multiple APIs need to be called on the participating application. The ABC implementation needs to have a chatty conversation with the application and also needs to manage state and transactions for these calls.

- Some of the EBS operations support multiple instances of an object to be passed as the payload.

  In this scenario, the provider ABC service implementing this operation can pass the entire payload to the provider application's API. This is possible only when the provider application has the ability to accept multiple instances and process them natively. In situations where the provider application does not have the ability, the ABC service implementation needs to loop through the payload and call the application API for each instance of the payload.

Choosing the Message Exchange Pattern for Requestor ABC Services

Synchronous Request – Response Message Exchange Pattern

Most of the cases involving Query and Validation would need this pattern. In this case the participating application that initiated the request is waiting for the response. For all the other cases, the demands of the response time and possibility of meeting those requirements based on the amount of processing to be done, will decide the suitability of this pattern.

In this case, the message exchange pattern between Requester ABC service and EBS can be synchronous or asynchronous.

Asynchronous Message Exchange Pattern

One-Way Service Call

This pattern is used when the Requestor ABC service receives the request message from the participating application and ends with the invoking of the EBS. There is no response to the participating application that initiated the request.

Request Delayed Response

In this pattern, the Requestor ABC service 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.

In this case, the message exchange pattern between Requester ABC service and EBS can be synchronous or asynchronous.

Choosing the Message Exchange Pattern for Provider ABC Services

Synchronous Request – Response Message Exchange Pattern

Most of the cases involving Query and Validation would need this pattern. In this case the EBS is waiting for the response. For all the other cases, the demands of the response time and possibility of meeting those requirements based on the amount of processing to be done, will decide the suitability of this pattern.

In this case, the message exchange pattern between Provider ABC service and participating application servicing the request can either be synchronous or asynchronous.

Asynchronous Message Exchange Pattern

One-Way Service Call

This pattern is used when the Provider ABC service receives the request message from the EBS and ends with the invoking of the provider participating application API. There is no response to the EBS that initiated the request.

Request Delayed Response

In this pattern, the Provider ABC service receives the request message from the EBS Request routing service, processes the message, and finally responds to the requesting service -- Requestor ABC service or enterprise business flow (EBF) -- by invoking the response operation of the EBS Response routing service. The EBS Request routing service is not waiting for the response.

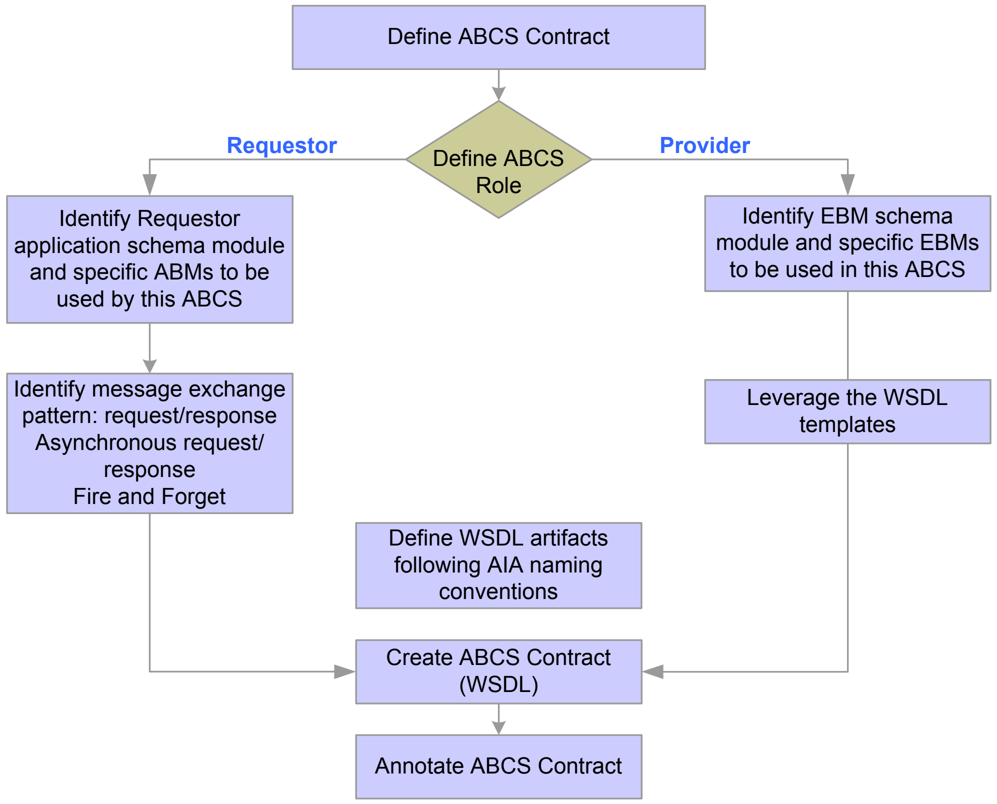
In this case, the message exchange pattern between Provider ABC service and the provider participating application can be synchronous or asynchronous.

Developing the ABC Service Contract

The ABM schemas are generated from the participating applications. Following points need to be considered while constructing the ABMs:

- Messages need to be durable and long lasting
- Frequent changing of ABM definition will put duress on ABC service development
- Reusability of business components across multiple ABMs is critical. Promotes reusability of transformation maps
- Application Business Messages should be designed to pass these system data
    - System Instance – connection information that helps in identifying System ID registered in BSR
    - Locale – language code in which the request is sent
    - Sender Information – contextual details pertaining to the origination of the message
- ABM should be extensible

The ABC service contract (WSDL) specifies how a participating application interacts with its Business Connector Service. Its MEP does not have to be identical to EBS MEP. This diagram depicts the decision flow for defining the contract (WSDL) for the ABC service.


Define ABCS Contract
Define ABCS Role
Requestor
Provider
Identify Requestor application schema module and specific ABMs to be used by this ABCS
Identify message exchange pattern: request/response Asynchronous request/ response Fire and Forget
Identify EBM schema module and specific EBMs to be used in this ABCS
Leverage the WSDL templates
Define WSDL artifacts following AIA naming conventions
Create ABCS Contract (WSDL)
Annotate ABCS Contract

Defining the ABC service Contract

The ABC service WSDL can be developed using the ABC service template WSDLs provided in the Foundation pack at **"AIA_HOME/samples/AIASamples/AIAServices-TemplateSampleWSDLs.zip"**. Follow the naming convention as mentioned in the template WSDLs and use the sample WSDL to check.

Constructing an ABC Service

Key principles to follow when constructing an ABC service:

- The relevant Oracle AIA Foundation Pack is installed and the development SOA server is up and running.
- The application entities' schemas (ABM schemas) are accessible from a central location. They should not be part of each ABC service project.
- Enterprise Object Library containing EBOs and EBMs should be accessed from a central location.

  They should not be part of each ABC service project.
- Except the ABC service WSDL and any EBS or Participating App reference WSDLs that define PartnerLink types, all the abstract EBS or Participating App WSDLs should be

imported from AIAComponents resource location created as part of Oracle AIA Foundation pack installation.

- Dynamic PartnerLink construct should be used for all PartnerLinks defined to invoke services except the EBS.

  This approach is followed since the target end point could be either the Composite Application Validation System (CAVS) or a concrete end point for one of the application instances.

For information about using the Artifacts Generator to autogenerate much of the common code needed to create ABC service implementations in BPEL, see Working with the Artifacts Generator.

ABC Service Implementation Technology

Though Oracle AIA does not prevent any technology to be used for developing an ABC service, BPEL process implementation will be used in most of the situations:

- Long-lived process

  In scenarios where you have a long-lived process that may take hours or even days, BPEL is the best technology choice. In this case, BPEL will persist (dehydrate) the process at certain hook points and then bring the process to life when needed.

- Complex orchestration

  In scenarios that require complex orchestration, parallel processing, and multiple conversations with participating applications, as well as with integrated services such as BPA, BAM, Workflow, and Rules Engine.

- Content augmentation and validation that cannot be done using XSLT

  In cases where the decision-making and validation is not simple enough to perform using XSLT, other means will be needed such as the standard BPEL procedural constructs or even calling out to the Rules Engine. BPEL enables you easily to perform fairly complicated decision trees and route the results to different partners.

- Aggregation or disaggregation

  In scenarios where the ABC needs to 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 will also be responsible for maintaining the state and handling transactions in between invocations.

- Augmenting the participating application functionality

There are scenarios where there is no matching operation on the application side for an EBS operation. For example, EBS may have a SyncCustomer operation with no matching operation at the application level. In this case, the SyncCustomer ABC implementation service needs to fill the gap by analyzing the content of the SyncCustomer message, querying the Customer object from the application side, comparing the two objects, and then deciding which child objects need to be inserted, updated, or deleted.

Describing Verbs Used to Define EBS Operations

This section provides brief explanations for each of these verbs used to define enterprise business service (EBS) operations:

- Create
- Update
- Delete
- Sync
- Validate
- Process
- Query

This content can help further your understanding of EBS operations from an invocation and implementation perspective.

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 a new instance of a business object.

Operations that use the Sync verb *must* create content and *should not* be an orchestration of other operations.

Generally speaking, a business process would invoke a create operation mainly in cases where the source event that triggers the invocation is *not* the creation 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, usage of Sync would also be conditional to the service provider having an implementation of a “Sync” operation.

An example of a typical usage of a “Create” operation would be a front end customer management system 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 in the case of “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 will have only two messages with a Create operation (single and List).

Verb Attributes

The Create Verb has an optional “ResponseCode” attribute that is intended to communicate 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).

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 for 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 is 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/ update content and *should not* be an orchestration of other operations.

Similar to the Create, a business process would invoke an “Update” operation mainly in cases where the source event that triggers the invocation is *not* the updation 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, usage of Sync would also be conditional to the service provider having an implementation of a “Sync” operation.

An example of an “Update” operation would be 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/ other information.

The content included in the business payload an EBM using the “Update” verb is assumed to be *only* the fields that have to 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 in the case of hierarchical, multi-level 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 in conjunction with the “Update” verb, to communicate the processing action to be applied to that business component.

Note that the “ActionCode” applies only if the object has child business components – if not, there is no need for an “ActionCode” and the verb alone is sufficient to convey this information.

As an example of usage of “ActionCode”, consider a scenario where a Requisition is updated in a self service requisitioning system, and this update information needs to be communicated to the Purchasing system which also maintains a copy of the Requisition.).

If a user updates a requisition and does these (in a single transaction)

- Update the “Description” in the Requisition Header
- Add a New Requisition Line (Line No. 4)
- Modify the item on a requisition line (Line No. 3)
- Delete a requisition Line (Line 2)
- Modify the accounting distribution of a requisition line, and add a new accounting distribution (Line 1)

The content of the DataArea business payload in the instance XML document that communicates the changes will be as:

```
<UpdateRequisition actionCode=”Update”>   Root Action Code not
processed
   <Description>New Description</Description>
   <RequisitionLine actionCode=”Update”> Indicates that some
property or association of this line is being updated. In this
example, the accounting distribution Percentage has been updated for
Line 1, and a new AccountingDistribution line has been added
         <Identification>
               <ID>1</ID>
         </Identification>
         < RequisitionAccountingDistribution actionCode=”UPDATE”>
               <Identification>
                     <ID>1</ID>
               </Identification>
               <AccountingDistribution>
                     <Percentage>15</Percentage>
               </AccountingDistribution>
         < /RequisitionAccountingDistribution>
         < RequisitionAccountingDistribution actionCode=”ADD”>
               <Identification>
                     <ID>3</ID>
               </Identification>
               <AccountingDistribution>
                     <Percentage>15</Percentage>
               </AccountingDistribution>
         < /RequisitionAccountingDistribution>
   </RequisitionLine>
   <RequisitionLine actionCode=”DELETE”> Indicates this line has
been deleted
         <Identification>
               <ID>2</ID>
```

```
            </Identification>
      </RequisitionLine>
      <RequisitionLine actionCode="UPDATE">
            <Identification>
                  <ID>3</ID>
            </Identification>
   <ItemReference>
   <Identification>
                        <ID>1001</ID>
                  </Identification>
   </ItemReference>
      </RequisitionLine>
      </RequisitionLine>
      <RequisitionLine actionCode="ADD"> Indicates this line has been
   added
            <Identification>
                  <ID>4</ID>
            </Identification>
   <ItemReference>
   <Identification>
                        <ID>1005</ID>
                  </Identification>
   </ItemReference>
      </RequisitionLine>
   </UpdateRequisition>
```

Content Payload

The payload of an operation that uses an "Update" Verb is typically the entire EBO and in general every business object will have only two messages with an Update operation (single and List).

There may be situations where there is a need to update subsets of an EBO, in which case it is possible that there may be multiple update messages, each with a distinct payload. An example is a possible UpdateSalesOrderLineEBM message with a payload that only contains 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).

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 for 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.

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. It 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, we do not support using "Delete" for components of a business object that is Delete Purchase Order Line is allowed.

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 in the case of "Delete" is "ID" (response payload is expected to be the Identifier of the object that was created).

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 for 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. It is used where applications provide operations that have the ability to accept the payload of the operation and create/ update business objects as necessary.

Operations that use the Sync verb *must* create/ update content and *should not* be an orchestration of other operations.

The primary usage scenario for “Sync” is generally batch transactions where 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 “Sync “operation implies that the object exists in both the source and the target system, and the end result of “Sync” is that both the source and target will have the same content. Sync is different from the other verbs in that it assumes a dual processing instruction – sync may both create as well as 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 where change is not frequent, and it may not be practical or necessary from an operational point of view to synchronize data on a realtime 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 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: 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 will be only one Sync message per EBO (with a single and List implementation), and the payload of the message will be the entire EBO.

Sync should always be used to synchronize the entire business object. Multiple Sync messages may exist in cases where there are different named views of the business object, but never for synchronizing a specific component of a business object.

Note: Unlike the OAGIS implementation of Sync, we have opted not to have specific attributes in Sync to indicate “Add”, “Change”, “Replace” and “Delete”. The 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 e the entire object that was created/ updated) and "ID" (response is only the ID of the object that was created/ updated)

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 for 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 where 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.

Operations that use the validate verb *do not* create/ update business objects, and may internally be implemented as an orchestration of other operations. For example, validating a Purchase Order for approval may involve validating if there is available budget, 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.

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, and/ or intended to achieve a specific business end-result.

“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, in order to avoid a proliferation of verbs for specific business object actions.

Operations that use the “Process” verb *must* always result in creation/ 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. Generally speaking 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 an equipment,, and so on.

Since there can be multiple operations performed by the “Process” verb, there can be potentially multiple “Process” verbs used in any given EBS.

Note: Operations that implement the “Process” verb may be implemented as synchronous or asynchronous – this is a deviation from the other verbs where there is a consistent implementation pattern that applies across all operations that use them.

Payload

The nature of the operation performed by a Process Verb may require properties/ 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.

The payload of each Process operation is not restricted to content from the EBO definition. This is unlike all the other EBOs where the EBM business content must be the same as or a subset of the EBO content.

Note: Currently, we do 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, there may be both a single and List implementation of a Process operation, or just one or the other. Unlike the other generic verbs where single and list are both consistently created for all EBOs, in the case of Process this is driven by the requirements of the corresponding operation.

Verb Attributes

Not applicable.

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 execute 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 requestor has the ability to optionally request for a specific subset of the response payload.

Similar to the other verbs, the “single” and the “List” pattern applies to queries also. The usage of “Query” for each of these patterns has been listed in the next sections.

Note: 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 allows callers to lookup an EBO by its identifier. It is intended to request for 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 more than one object, and the response payload for the simple query is restricted to a single instance of the object being queried.

The Single Object Query contains these elements:

QueryCode within the Query element (optional)

The use of the QueryCode in a single object Query is 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 in cases where there is a need to communicate more than the ID to the query service provider in order to successfully execute 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.

Note: In order for this to happen, the service provider should implement the processing of such a code. Currently, we do not pre-define 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, 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 only provide 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.

Note: In order for this to happen, the service provider should implement the processing of such a code. Currently, we do not pre-define 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.

```
<QueryAccountBalanceAdjustmentEBM>
   <corecom:EBMHeader>

   </corecom:EBMHeader>
   <DataArea>
         <Query>
         </Query>
         <QueryAccountBalanceAdjustment>
               <corecom:Identification>
                     <corecom:ID>1005</corecom:ID>

               </corecom:Identification>
               <Custom/>
         </QueryAccountBalanceAdjustment>
   </DataArea>
</QueryAccountBalanceAdjustmentEBM>
```

Verb Attributes

The simple Query can have an optional getAllTranslationsIndicator to indicate if 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

There are two possible ways 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. There is a single "Customer" Query service, but in order to successfully execute the query, the service provider needs to 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 these 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.

Note that in order for this to happen, the service provider should implement the processing of such a code. Currently, we do not pre-define 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 only provide 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.

Note that in order for this to happen, the service provider should implement the processing of such a code. Currently, we do not pre-define any generic Request or Response Codes as part of the EOL.

QueryCriteria (1 to n instances)

The QueryCriteria element provides the ability for a user to build a complex query statement *on a specific node* of the object being queried. There must be at least one QueryCriteria element specified for a List Query.

There can be more than one QueryCriteria element present in a Query*, if the query spans multiple nodes* of the object being queried. Multiple Query criteria is similar to a sub-select in that the result set of one Query Criteria is filtered by the second to arrive at a smaller sub-set of records.

Each QueryCriteria element consists of these:

QualifiedElementPath (0 or 1 instance): This provides the ability for 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 NULL value, or if it has a value of "/", it implies that 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 supports the ability to define complex queries. Each QueryExpression consists of these:

- 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 1 or more QueryExpressions or 1 or more ValueExpressions.
    - A QueryExpression may contain other QueryExpressions when there is a need for combining 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 these:

- 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. There is no explicit way to indicate if the ElementPath contains a code or an Xpath expression
- **Value** element that contains the value assigned to the ElementPath, for example, Approved
- A **queryOperatorCode** attribute that specifies the operator applicable to the Value assigned to the ElementPath. The possible operators are EQUALS, NOT_EQUALS, GREATER_THAN, GREATER_THAN_EQUALS, LESS_THAN, LESS_THAN_EQUALS, CONTAINS, DOES_NOT_CONTAIN, LIKE, NOT_LIKE, LIKE_IGNORE_CASE, NOT_LIKE_IGNORE_CASE, IS_BLANK, IS_NOT_BLANK, BETWEEN,

NOT_BETWEEN, IN, NOT_IN

SortElement (0 or more instances): Each QueryCriteria can have one or more SortElements defined. A SortElement is used to request for 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).

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

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

In the example of SortElement (0 or more instances), the result set of the QueryExpression is to be sorted by Descending Order of OrderDateTime and ascending order of Description.

If there are multiple QueryCriteria, then each can have its own SortElement – so the result set of one QueryCriteria is sorted, and then once the next QueryCriteria filter is applied, the resultant sub-set is sorted using the SortElement specified for that QueryCriteria.

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

         </ValueExpression>
   </QueryExpression>
   <SortElement sortDirection=”DESC”>/OrderDateTime</SortElement>
   <SortElement sortDirection=”ASC”>/Description</SortElement>
</QueryCriteria>
<QueryCriteria>
   <QualifiedElementPath>/SalesOrderLine/SalesOrderLineBase</Qualifi
edElementPath>
   <QueryExpression>
   ...
   </QueryExpression>
   <SortElement>/SalesOrderLine/SalesOrderLineBase/ListPrice</SortEl
ement>
</QueryCriteria>
```

In this example, the SalesOrder root query is sorted by OrderDateTime and Description, while the child node SalesOrderLine is sorted by price.

QueryCriteria Examples

Example 1: This is an example of a query with a single QueryCriteria element and a single QueryExpression element which contains only ValueExpression elements. The query to be executed is: Query SalesOrder where “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.

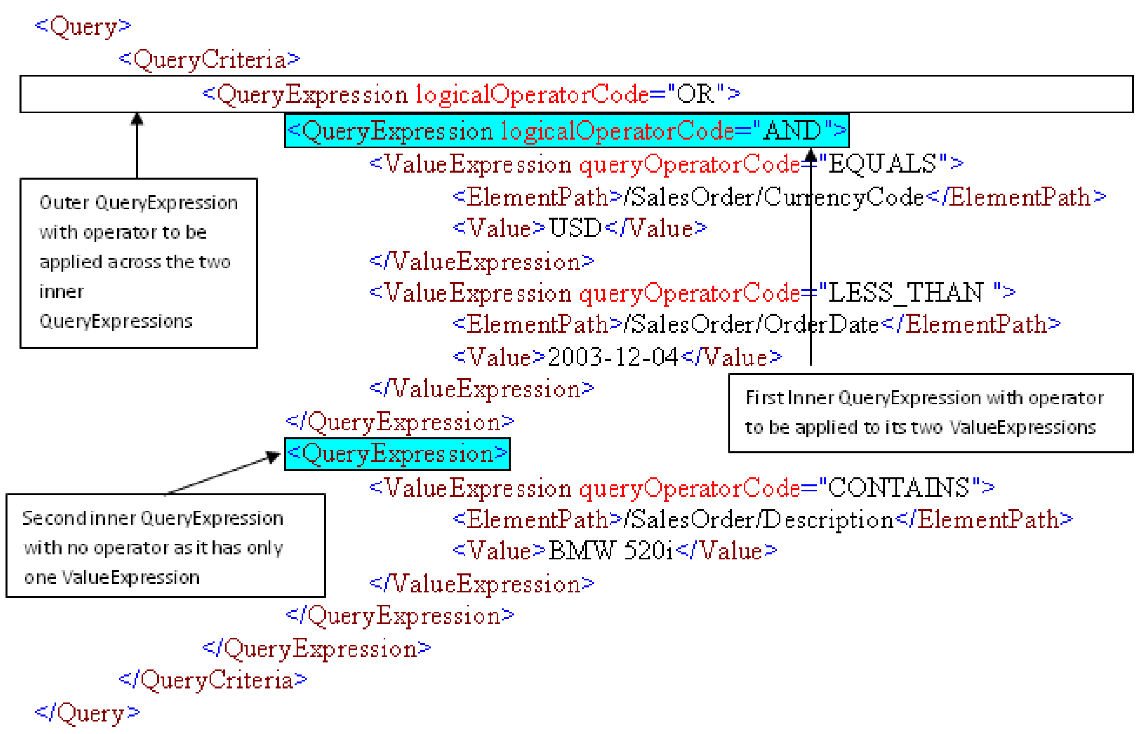
```
<Query>
<QueryCriteria>
         <QueryExpression logicalOperatorCode="AND">
               <ValueExpression queryOperatorCode="EQUALS">

   <ElementPath>/SalesOrderBase/CurrencyCode</ElementPath>
                     <Value>USD</Value>
               </ValueExpression>
               <ValueExpression
queryOperatorCode="GREATER_THAN_EQUALS">

   <ElementPath>/SalesOrderBase/OrderDateTime</ElementPath>
                     <Value>2003-12-04</Value>
               </ValueExpression>
         </QueryExpression>
   </QueryCriteria>
</Query>
```

Example 2: This is an example of a Query with a single QueryCriteria but with nested QueryExpressions. The query to be executed is: Query SalesOrders where 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 there are nested QueryExpressions, 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



```
<Query>
    <QueryCriteria>
        <QueryExpression logicalOperatorCode="OR">
            <QueryExpression logicalOperatorCode="AND">
                <ValueExpression queryOperatorCode="EQUALS">
                    <ElementPath>/SalesOrder/CurrencyCode</ElementPath>
                    <Value>USD</Value>
                </ValueExpression>
                <ValueExpression queryOperatorCode="LESS_THAN ">
                    <ElementPath>/SalesOrder/OrderDate</ElementPath>
                    <Value>2003-12-04</Value>
                </ValueExpression>
            </QueryExpression>
            <QueryExpression>
                <ValueExpression queryOperatorCode="CONTAINS">
                    <ElementPath>/SalesOrder/Description</ElementPath>
                    <Value>BMW 520i</Value>
                </ValueExpression>
            </QueryExpression>
        </QueryExpression>
    </QueryCriteria>
</Query>
```

Outer QueryExpression with operator to be applied across the two inner QueryExpressions
First Inner QueryExpression with operator to be applied to its two ValueExpressions
Second inner QueryExpression with no operator as it has only one ValueExpression

Example 3: This is an example of a Query with multiple QueryCriteria. Note that when there are multiple QueryCriteria, there is no logical operation that is applicable across them – they are the equivalent of executing 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 executed on the CustomerParty root node.

The result set of this query is then filtered by applying the second Query Criteria. This will query the CustomerParty/Account node to get all CustomerParty Accounts that have STATUS = "ACTIVE".

```
<Query>
   <QueryCriteria>
         <QueryExpression>
               <ValueExpression queryOperatorCode="EQUALS">
                     <ElementPath>/Type</ElementPath>
                     <Value>GOLD</Value>
               </ValueExpression>
         </QueryExpression>
         <SortElement>/LastName</SortElement>
   </QueryCriteria>
   <QueryCriteria>

   <QualifiedElementPath>/CustomerAccount</QualifiedElementPath>
         <QueryExpression>
               <ValueExpression queryOperatorCode="EQUALS">
```

```
      <ElementPath>/CustomerAccount/Status/Code</ElementPath>
                        <Value>ACTIVE</Value>
                  </ValueExpression>
            </QueryExpression>
      </QueryCriteria>
   </Query>
```

ResponseFilter (0 to 1 instance)

The ResponseFilter provides the ability for a requester to indicate which child nodes they are interested or not interested in. The excluded child nodes will not be populated by the application when building the response to the query. If supported by participating applications, this feature will improve performance by not querying the nodes that are excluded from the query.

Example 1: Requesting for only the InvoiceLine to be returned in response to a QueryInvoice message

```
   <Query>
      <QueryCriteria>

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

Example 2: Returning all QueryInvoice message content except for Charge and PaymentTerm

```
   <Query>
      <QueryCriteria>

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

Note: In the absence of a very comprehensive framework for processing queries, this option is difficult to implement practically, as in theory there are infinite ways in which the query can be built, and the service provider will not be able to process all possible ways in which the “QueryCriteria” is specified.

Content Payload

There is no content payload for a List Query – 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 if 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 will return 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 101^{st} to the 200^{th} record of the result set.

- **recordSetCount (optional):** 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 requestor may only be able to process 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 requestor can only process 100 records at a time. The service requestor can use the maxItems attribute to instruct the service provider to return only a 100 records in the response.

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

Implementing ABC Services

In the ABC service, these tasks are accomplished:

- Message enrichment
- Transform message content
- Populate message headers
- Error handling and logging
- Extensibility
- Multi-instance support and CAVS enablement
- Implement security context

These sections provide guidelines for developing the ABC service.

For information about using the Artifacts Generator to autogenerate much of the common code needed to create ABC service implementations in BPEL, see Working with the Artifacts Generator.

Message Enrichment

- The APIs exposed by the participating applications are granular and sometimes all the information needed to generate the EBM for an EBS is not available in a Requester ABC service.

  In such situations, the ABC service queries back the required information from the requesting application, and uses this information to enrich the EBM in the ABM > EBM transformation.

- The PartnerLink(s) defined should be dynamic and be capable for CAVS and multiple application instances.

For more information, see Chapter 12: Developing Oracle AIA Services to be CAVS-Enabled and Provide Multi-Instance Support.

Transform Message Content

- In the Requestor ABC service, the ABM is transformed to an EBM prior to invoking the EBS operation.

  If the operation returns a response, the requestor ABC service will be responsible for transforming EBM to ABM before returning the message to the requestor application.

- In the Provider ABC service, the EBM is transformed to an ABM prior to invoking the services provided by provider application.

  In case of EBS operations expecting a response, the provider ABC service will transform the ABM returned by the application into EBM. This would be sent back as the response.

- These are the best practice recommendations:
  - Transformations from ABM --> EBM should be comprehensive mapping of all available attributes (All ABM Attributes supported in EBM should be mapped)
  - Requester ABC services should NOT stamp any target system information in the EBM header before invoking the EBS
  - Provider ABC services should not assume that the target system ID has been populated in EBM header
  - Instance identifying information needs to be populated with as much information as possible in every Identification/IdentificationType element
- Transformation should be extension enabled to accommodate transformations for custom element

- EBM header needs to be populated fully
- Every EBM needs to have languageCode & versionID populated
  - languageCode – specifies the locale in which the request is sent
  - versionID- specifies the major & minor version of the schema used
- Verb element in data area needs to be populated appropriately
- Processing in Provider ABC services is influenced by the content

  For example, the EBM for Create operation can have Create element with ResponseCode attribute set to either ‘ID’ or ‘OBJECT’. This will indicate what type of response is expected by requestor ABC services from provider ABC services after the successful completion of the task.

- Language sensitive data elements need to have languageCode attribute populated if different from the one set at EBM root element

```
<xsd:element name="QueryAccountBalanceAdjustmentEBM" type="QueryAccountBalanceAdjustmentEBMType"/>
<xsd:complexType name="QueryAccountBalanceAdjustmentEBMType">
      <xsd:sequence>
            <xsd:element ref="corecom:EBMHeader"/>
            <xsd:element name="DataArea" type="QueryAccountBalanceAdjustmentDataAreaType"/>
      </xsd:sequence>
      <xsd:attribute name="languageCode" type="corecom:LanguageCodeType"/>
      <xsd:attribute name="versionID" type="corecom:StringType"/>
</xsd:complexType>
```

- No unnecessary or redundant aia configuration lookup calls
- Retrieve static configuration properties once in the very beginning of transformation
- "Push" rather than "Pull" approach should be used when appropriate for performance
- Transformation logic and XPath should be designed with the possibility of missing or empty elements

```
<xsl:if test="normalize-space(Line3/text())">
    <Line3>
        <xsl:value-of select="Line3/text()"/>
    </Line3>
</xsl:if>
```

- No unnecessary or redundant Xref calls
- ABC services and XSL Scripts should not be coded to work against one specific logical application instance
- No hardcoded system Ids in XSL scripts and ABC services

For more information about standards for transforming documents, using cross references, using Domain Value Maps and the use of Oracle AIA Configuration file, see Chapter 6: Understanding Message Transformation, Enrichment, and Configuration.

Populate Message Headers

For more information about various components and elements in the EBM header and their purpose, see Chapter 9: Working with EBM Headers.

Error Handling and Logging

For more information about error handling and logging, see Chapter 13: Configuring Oracle AIA Processes for Error Handling and Trace Logging.

Extensibility

For more information about extensibility, see Chapter 15: Extensibility for Oracle AIA Artifacts.

ABC Service Multi-Instance Support and CAVS Enablement

ABC services must be capable of supporting multiple instances of a participating application. You will use dynamic PartnerLinks to provide CAVS enablement and multi-instance support for ABC services.

For more information about CAVS enablement and multi-instance support, see Chapter 12: Developing Oracle AIA Services to be CAVS-Enabled and Provide Multi-Instance Support.

Implement Security Context

Work with edge applications so that the application can send and receive authorization information in XACML format.

```
<soap:Envelope xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
xmlns:rep='http://www.w3.org/2004/08/representation'
xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
  <soap:Header>
   <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>

    </soap:Header>
    <soap:Body>
    </soap:Body>
</soap:Envelope>
```

Determine the implementation strategy for transform appcontext service and implement the service for all the applications involved in scenario

In the Requestor ABC service:

- Map the application authorization info in XACML format if it is not received in XACML format from edge application.
- Call the security service with application info in XACML format and receive the standard authorization info in XACML format.

  The response XACML is represented using Application neutral attributes described in section 3.5.1.2.

- Insert the standard authorization info in EBM header and provide a sample XSL snippet that will include request element in EBM header.

  Also provide a screen shot of EBM header showing the sender system with request details.

In Provider ABC services:

- Call the security service with standard authorization info in XACML format and receive application authorization info in XACML format
- Send the authorization info in XACML format if the edge application supports else set the application security context using XACML structure.

Apart from these, there are some typical activities that need to be done in the ABC service for some of the patterns. Depending on the type of pattern used for ABC service, please follow the next sections.

Implementing Asynchronous Message Exchange Pattern in Requestor ABC Services

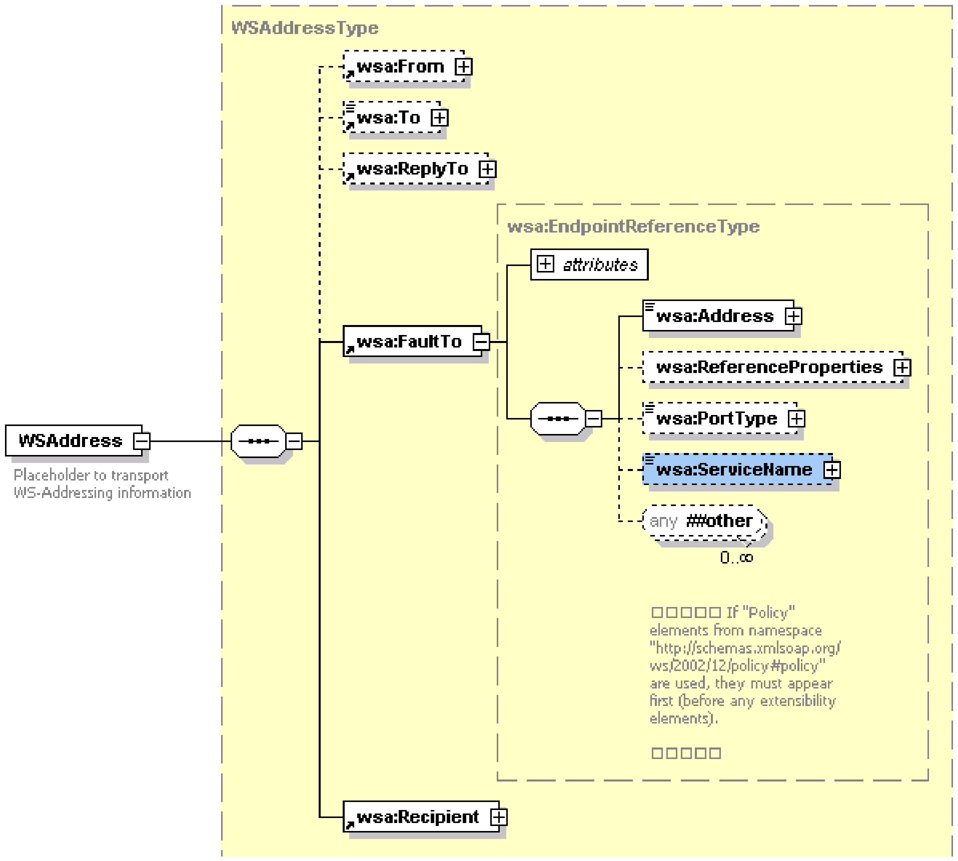
One-Way Service Call

This pattern is used when the Provider ABC service receives the request message from the EBS and ends with the invoking of the provider participating application API. There is no response to the EBS that initiated the request.

Before an EBS is invoked, various elements of the EBM header are populated.

In case of the fire-and-forget pattern, there is no response. In situations where the provider ABC service is not able to successfully complete the task, the retrying as well as notifications will be done automatically. But there are times, where either an automatic correction of data or reversal of what has been done in requestor service will be needed. In these situations, the requestor application can implement the compensation service operation; and pass the name of the service operation to Provider the ABC service. The Provider ABC service invokes this compensation service operation in case of errors. There may be a need to implement a compensating service for the requesting service. In order for the correct compensating service to be invoked from the providing service, these need to be done:

- 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 (follow Appendix A: Oracle AIA Naming Standards): CompensateCreateOrderSiebelReqABCSImpl


WSAddressType
wsa:From
wsa:To
wsa:ReplyTo
wsa:EndpointReferenceType
attributes
wsa:Address
wsa:FaultTo
wsa:ReferenceProperties
wsa:PortType
wsa:ServiceName
any ##other
0..∞
WSAddress
Placeholder to transport WS-Addressing information
□□□□□ If "Policy" elements from namespace "http://schemas.xmlsoap.org/ws/2002/12/policy#policy" are used, they must appear first (before any extensibility elements).
□□□□□
wsa:Recipient

Structure of the WSAddressType

This information will be used in the compensate operation of the EBS to route the request for compensation to the correct compensating service.

Request Delayed Response

In this pattern, the Provider ABC service receives the request message from the EBS Request routing service, processes the message, and finally responds to the requesting service (Requestor ABC service or EBF) by invoking the response operation of the EBS Response routing service. The EBS Request routing service is not waiting for the response.

In this case, the message exchange pattern between Provider ABC service and the provider participating application can be synchronous or asynchronous.

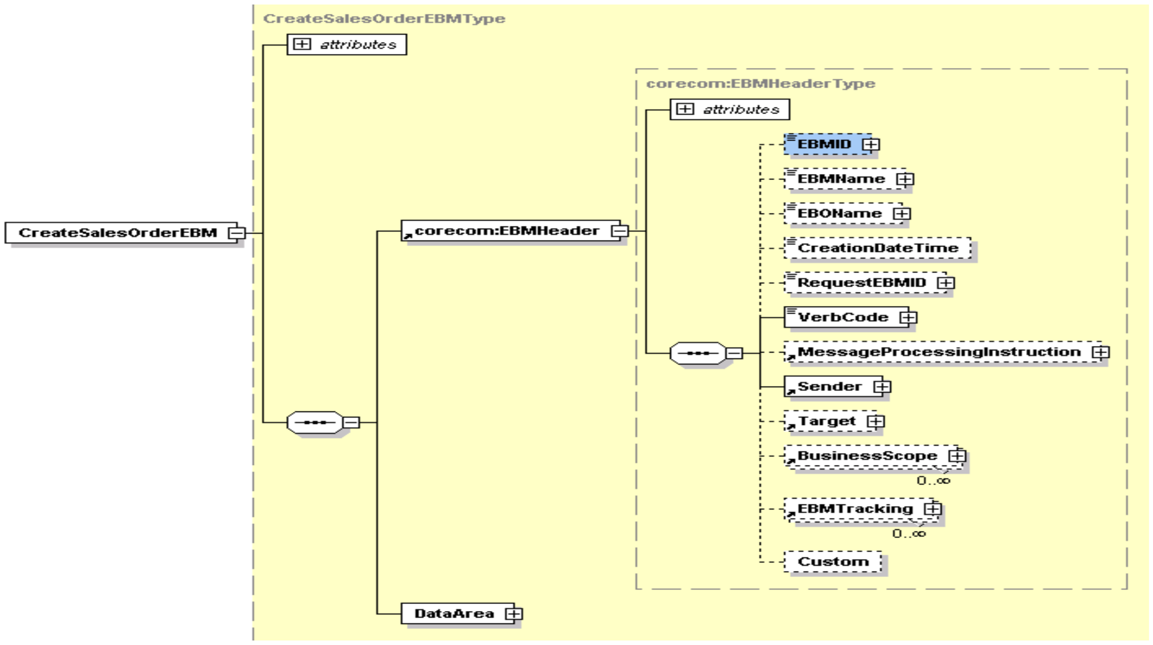
Before an EBS is invoked, three tasks need to be done:

Populate EBM Header

These are the entries needed in the EBM header in the requesting service:

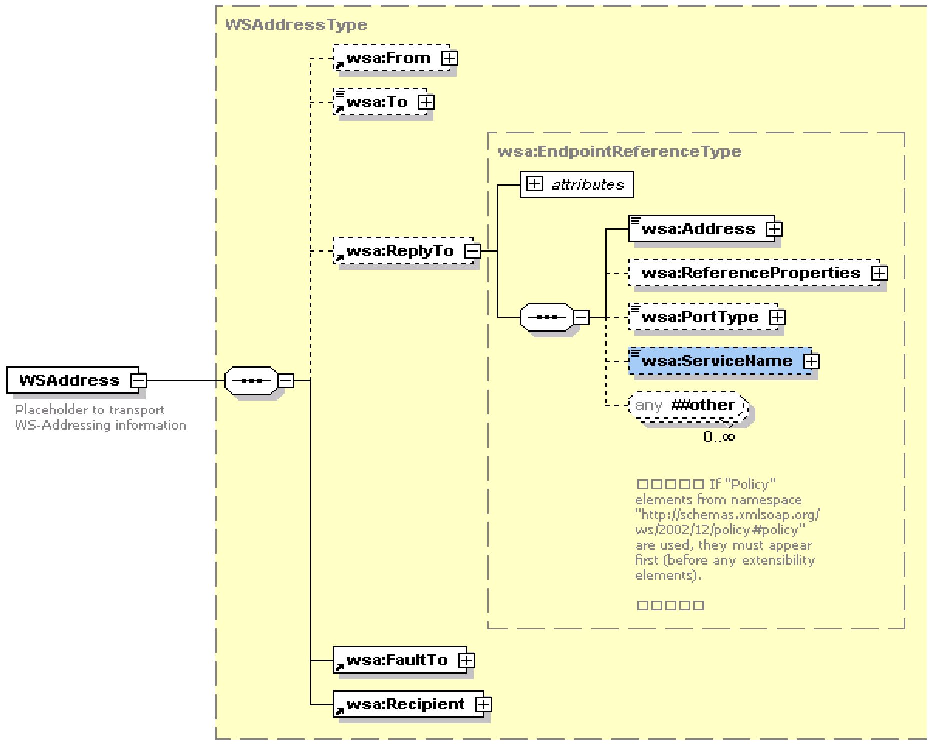
1. Populate EBMID: This will be used for correlation of the response to the correct requesting service instance.

   Set the EBMID to guide as detailed in Chapter 9: Working with EBM Headers.



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. This will be the name of the service that needs to be invoked by EBS for processing the response message. In most of the situations, it will be the same Requestor ABC service that would also be responsible for processing the response message coming from provider ABC service.


WSAddressType
wsa:From
wsa:To
wsa:EndpointReferenceType
attributes
wsa:Address
wsa:ReplyTo
wsa:ReferenceProperties
wsa:PortType
wsa:ServiceName
WSAddress
Placeholder to transport WS-Addressing information
any ##other
0..∞
□□□□□ If "Policy" elements from namespace "http://schemas.xmlsoap.org/ws/2002/12/policy#policy" are used, they must appear first (before any extensibility elements).
□□□□□
wsa:FaultTo
wsa:Recipient

Structure of the WSAddressType

This information will be used in the response operation of the EBS to route the response from the providing service to the correct requesting service.

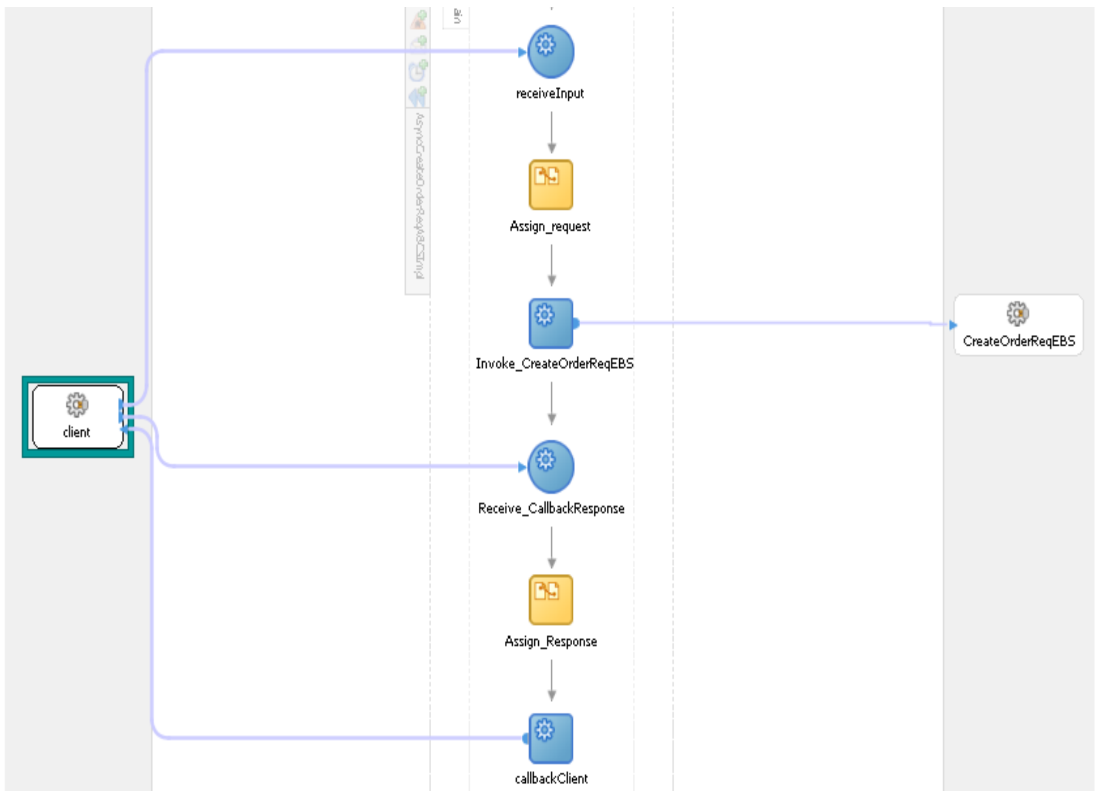
Set the "responseCode" attribute of EBM Verb

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.

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. Once the 'Invoke' step is performed in the requesting service and the process moves to 'Receive' step, the BPEL process instance is suspended and dehydrated. This will restart after the response is received. In order to facilitate this behavior, correlation set has to be specified on the PartnerLink for the 'Receive'.

The requester ABC service process would look like this. This is the process where you need to set the correlation ids.



Example of requestor ABC service process

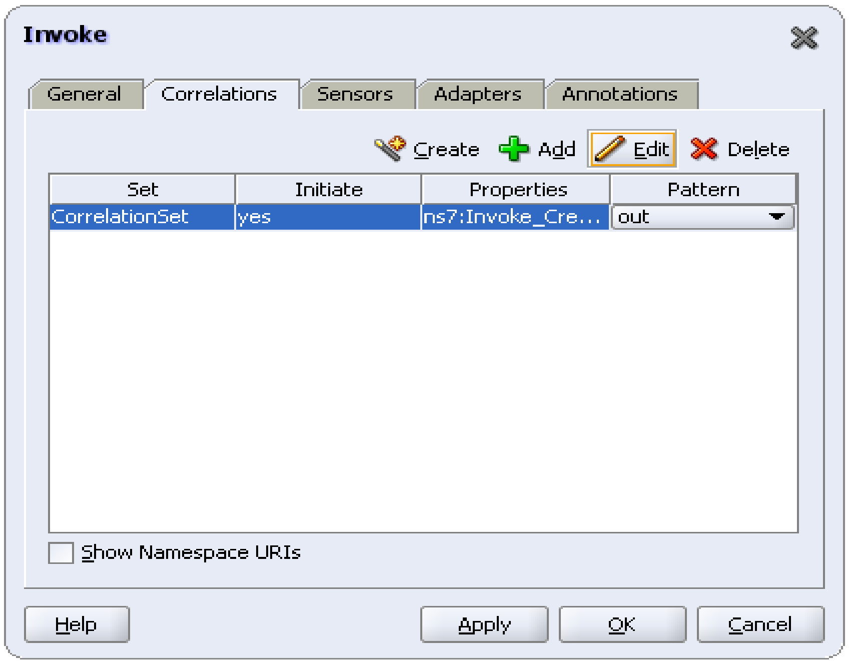
In the preceding process, there are two places where you need to set the correlation.

- Correlation Set
  - Add a correlation ID and create a correlation set for the “Invoke” activity where the process to go into 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. The property should always be defined as “correlation = correlationSet”.

To create the correlation set for the Invoke/Receive activities and correlation property for the PartnerLinks:

1. Create Correlation Set for the “Invoke” Activity as illustrated in the following diagram.

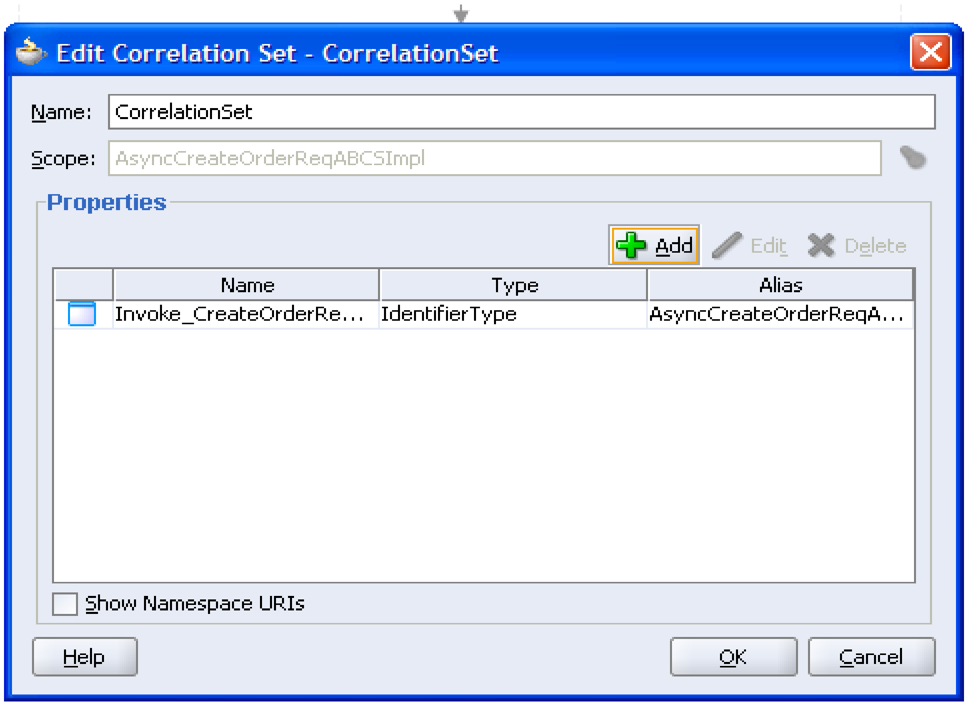
   Select the “pattern = out” and the “initiate = yes”.



Set	Initiate	Properties	Pattern
CorrelationSet	yes	ns7:Invoke_Cre...	out

Creating correlation set for the invoke activity

2. Click Add to select the correlation set variable.



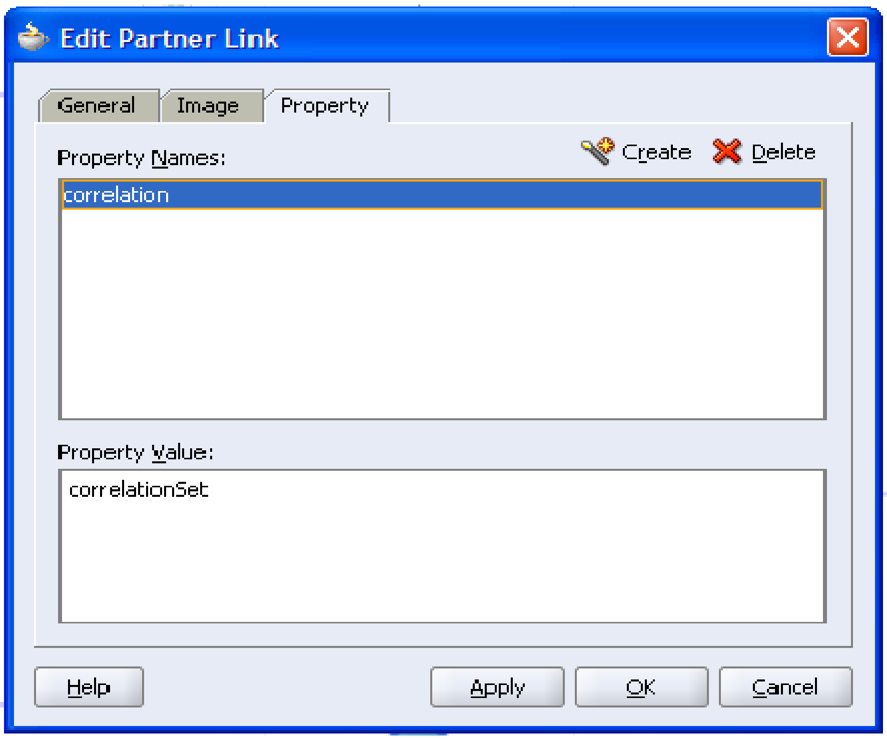
Name: CorrelationSet

Scope: AsyncCreateOrderReqABCSImpl

	Name	Type	Alias
	Invoke_CreateOrderRe...	IdentifierType	AsyncCreateOrderReqA...

Selecting the correlation set variable

3. Set the correlation property in the PartnerLink to be invoked with “correlation = correlationSet”



Property Names:

correlation

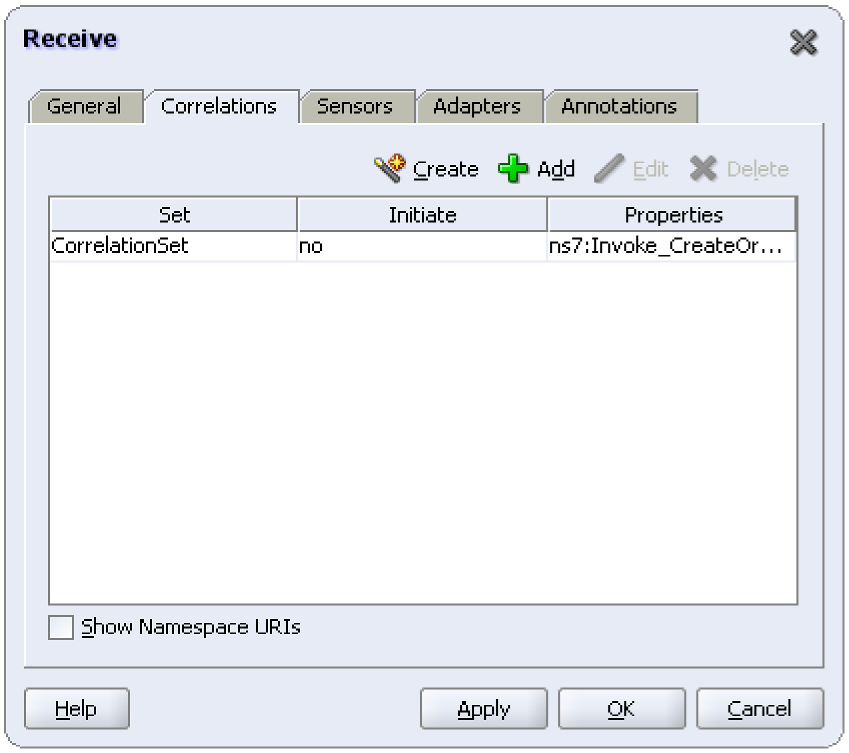
Property Value:

correlationSet

Setting the correlation property

4. Set the correlation ID in the “Receive” activity.

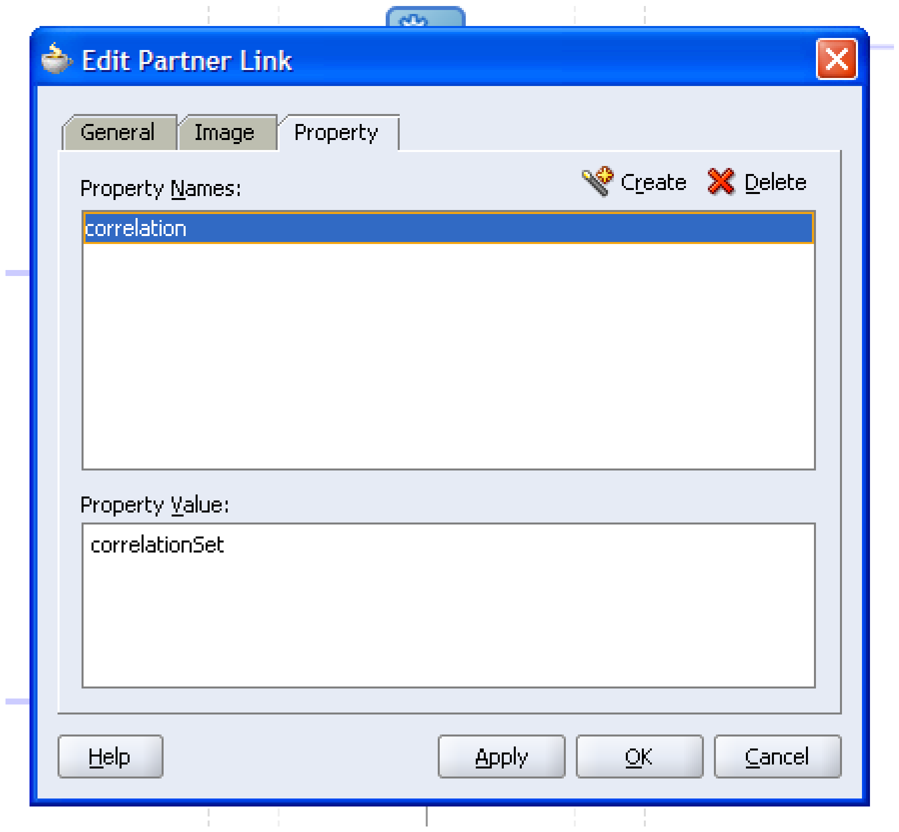
Make sure to select the “Initiate = no”.



Set	Initiate	Properties
CorrelationSet	no	ns7:Invoke_CreateOr...

Setting the correlation ID

5. Set the PartnerLink property for the client PartnerLink with “correlation = correlationSet”

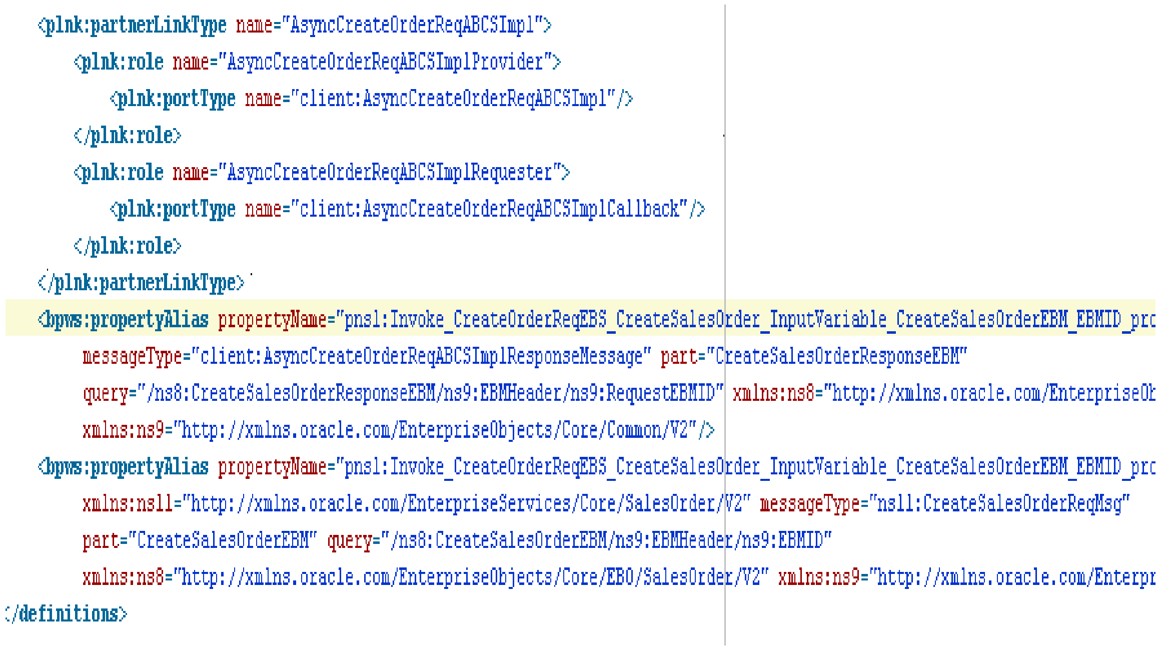


Property Names: correlation

Property Value: correlationSet

Setting the PartenerLink property

6. Verify your requester ABC service WSDL file to check whether the correlation sets are created properly or not.



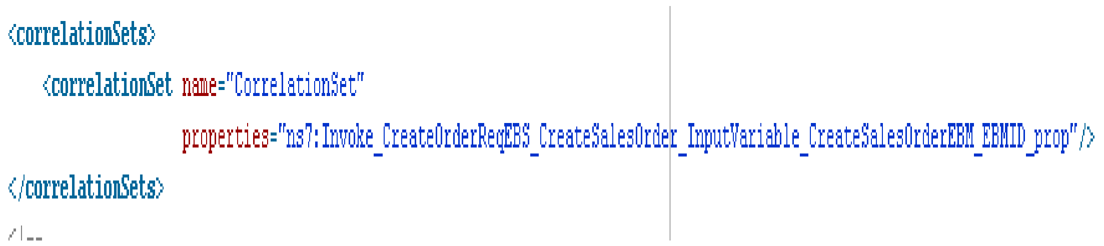
```
  <plnk:partnerLinkType name="AsyncCreateOrderReqABCSImpl">
    <plnk:role name="AsyncCreateOrderReqABCSImplProvider">
      <plnk:portType name="client:AsyncCreateOrderReqABCSImpl"/>
    </plnk:role>
    <plnk:role name="AsyncCreateOrderReqABCSImplRequester">
      <plnk:portType name="client:AsyncCreateOrderReqABCSImplCallback"/>
    </plnk:role>
  </plnk:partnerLinkType>
  <bpws:propertyAlias propertyName="pns1:Invoke_CreateOrderReqEBS_CreateSalesOrder_InputVariable_CreateSalesOrderEBM_EBMID_pro
     messageType="client:AsyncCreateOrderReqABCSImplResponseMessage" part="CreateSalesOrderResponseEBM"
     query="/ns8:CreateSalesOrderResponseEBM/ns9:EBMHeader/ns9:RequestEBMID" xmlns:ns8="http://xmlns.oracle.com/EnterpriseOb
     xmlns:ns9="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"/>
  <bpws:propertyAlias propertyName="pns1:Invoke_CreateOrderReqEBS_CreateSalesOrder_InputVariable_CreateSalesOrderEBM_EBMID_pro
     xmlns:ns11="http://xmlns.oracle.com/EnterpriseServices/Core/SalesOrder/V2" messageType="ns11:CreateSalesOrderReqMsg"
     part="CreateSalesOrderEBM" query="/ns8:CreateSalesOrderEBM/ns9:EBMHeader/ns9:EBMID"
     xmlns:ns8="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesOrder/V2" xmlns:ns9="http://xmlns.oracle.com/Enterpr
</definitions>
```

Verifying the requester ABCS WSDL file

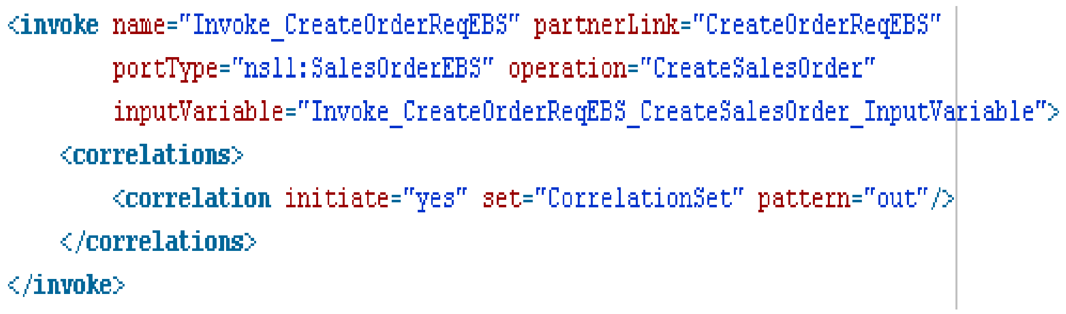
The values will look like this:

```
   <bpws:propertyAlias
propertyName="pns1:Invoke_CreateOrderReqEBS_CreateSalesOrder_InputVa
riable_CreateSalesOrderEBM_EBMID_prop"
messageType="client:AsyncCreateOrderReqABCSImplResponseMessage"
part="CreateSalesOrderResponseEBM"

query="/ns8:CreateSalesOrderResponseEBM/ns9:EBMHeader/ns9:RequestEBM
ID"
xmlns:ns8="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesO
rder/V2"

xmlns:ns9="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
/>
    <bpws:propertyAlias
propertyName="pns1:Invoke_CreateOrderReqEBS_CreateSalesOrder_InputVa
riable_CreateSalesOrderEBM_EBMID_prop"
xmlns:ns11="http://xmlns.oracle.com/EnterpriseServices/Core/SalesOrd
er/V2" messageType="ns11:CreateSalesOrderReqMsg"
         part="CreateSalesOrderEBM"
query="/ns8:CreateSalesOrderEBM/ns9:EBMHeader/ns9:EBMID"

xmlns:ns8="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/SalesO
rder/V2"
xmlns:ns9="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
/>
```

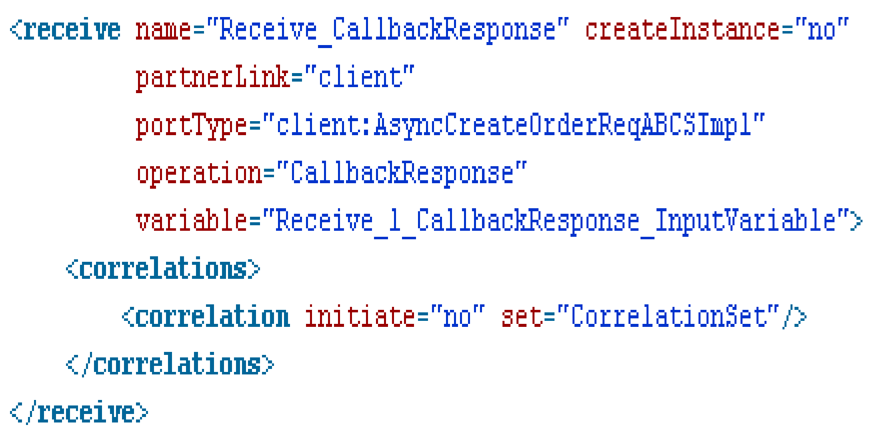
7. Verify the BPEL source to see the correlation set was created correctly. You should see the code snippets like this:



```
<correlationSets>
   <correlationSet name="CorrelationSet"
                 properties="ns7:Invoke_CreateOrderReqEBS_CreateSalesOrder_InputVariable_CreateSalesOrderEBM_EBMID_prop"/>
</correlationSets>
<!--
```



```
<invoke name="Invoke_CreateOrderReqEBS" partnerLink="CreateOrderReqEBS"
        portType="ns11:SalesOrderEBS" operation="CreateSalesOrder"
        inputVariable="Invoke_CreateOrderReqEBS_CreateSalesOrder_InputVariable">
    <correlations>
        <correlation initiate="yes" set="CorrelationSet" pattern="out"/>
    </correlations>
</invoke>
```



```
<receive name="Receive_CallbackResponse" createInstance="no"
         partnerLink="client"
         portType="client:AsyncCreateOrderReqABCSImpl"
         operation="CallbackResponse"
         variable="Receive_1_CallbackResponse_InputVariable">
    <correlations>
        <correlation initiate="no" set="CorrelationSet"/>
    </correlations>
</receive>
```

In a typical ABC service (or EBF), it is very common to invoke two different target services in an asynchronous delayed response pattern. To keep track of the business process instance while interacting with different target systems, a correlation ID used in different correlation sets.

To enable the two correlation sets to work in a single BPEL process:

These steps need to be executed for ABCSImpl or EBF, based on the requirements:

1. Create only one Alias-Variable/Property

Create only one alias property in the **<your process>_properties.wsdl** file as illustrated here:

```
<bpws:property
name="Invoke_1_CreateCustomerParty_InputVariable_CreateCustomerParty
EBM_prop" type="xsd:normalizedString"/>
```

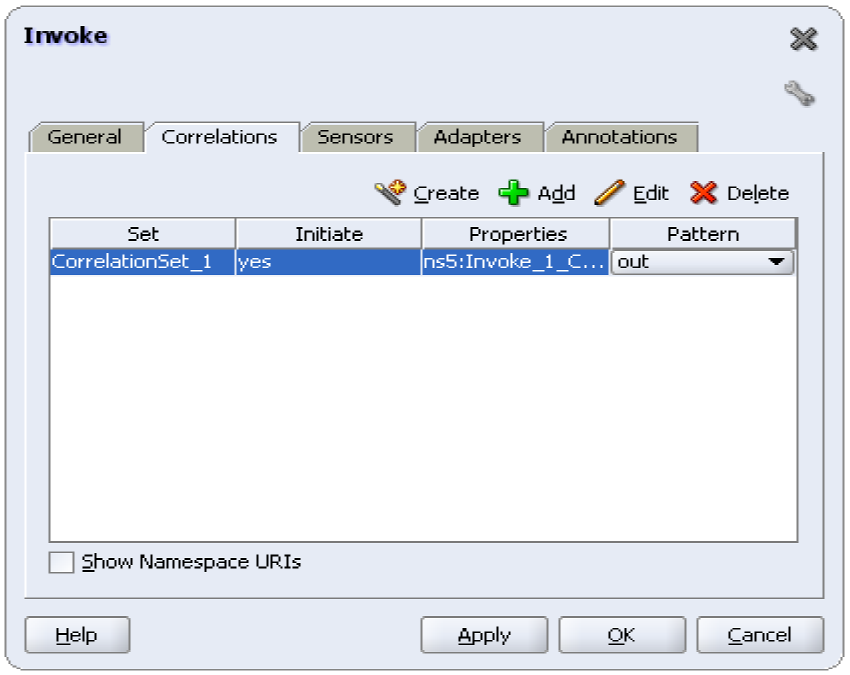
2. Create two Correlation-Sets

In your BPEL process **<your process>.bpel** file, create two correlation sets pointing to same alias property as illustrated here:

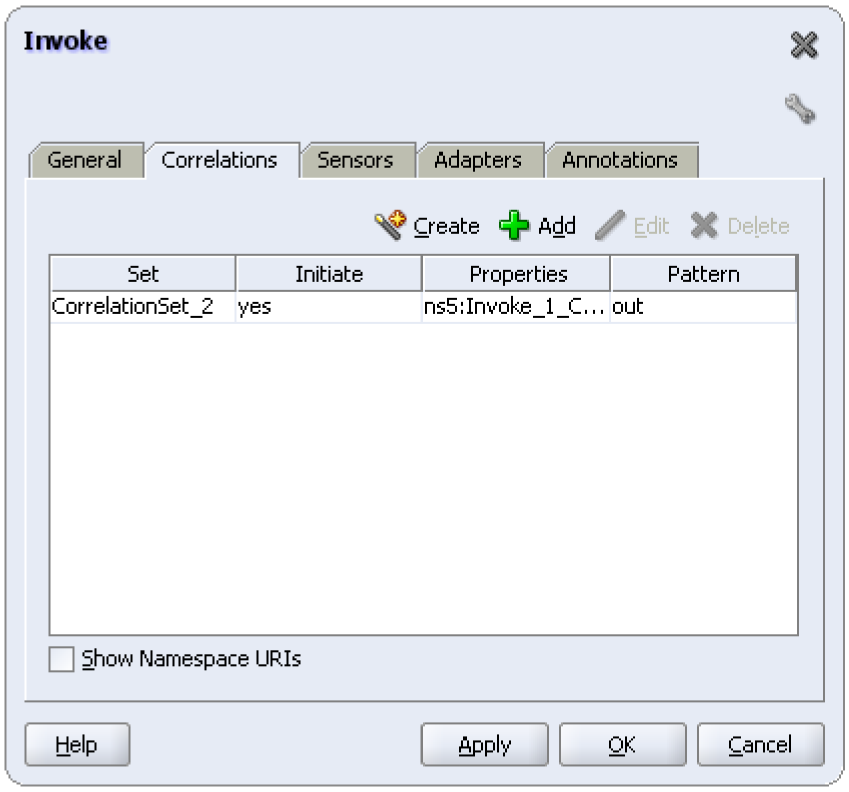
```
<correlationSets>
       <correlationSet name="CorrelationSet_1"

properties="ns5:Invoke_1_CreateCustomerParty_InputVariable_CreateCus
tomerPartyEBM_prop"/>

        <correlationSet name="CorrelationSet_2"

properties="ns5:Invoke_1_CreateCustomerParty_InputVariable_CreateCus
tomerPartyEBM_prop"/>
    </correlationSets>
```

3. Create all the message types with XPath expressions pointing to the same alias property

In your invoke activities, use the same alias property while mapping the correlation ID using the XPath expression. Make sure to select “initiate = yes” and “pattern = out”



Set	Initiate	Properties	Pattern
CorrelationSet_1	yes	ns5:Invoke_1_C...	out

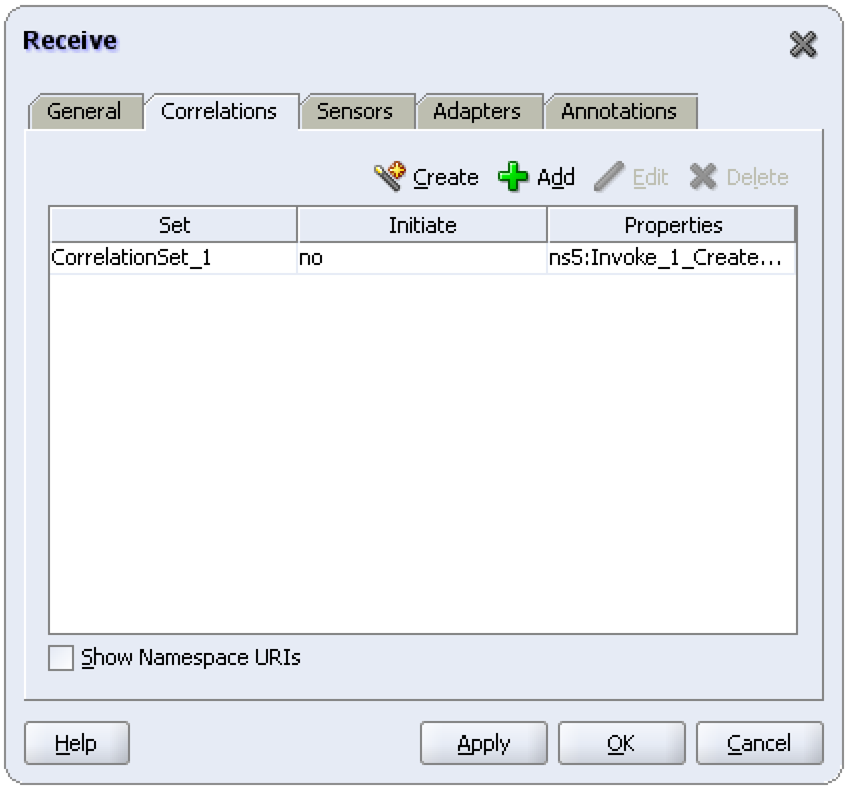


Set	Initiate	Properties	Pattern
CorrelationSet_2	yes	ns5:Invoke_1_C...	out

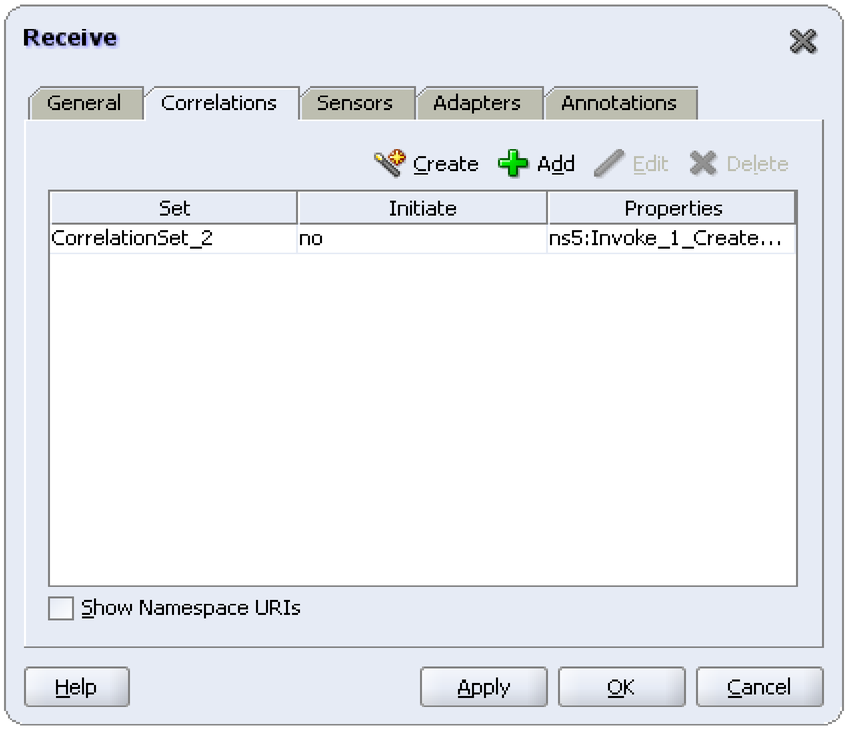
☐ Show Namespace URIs

In your receive activities, make sure to use corresponding correlation set that was defined in the invoke activities.

Do not select the “create instance” check box and set the “initiate=no” as illustrated here:



Set	Initiate	Properties
CorrelationSet_1	no	ns5:Invoke_1_Create...



Set	Initiate	Properties
CorrelationSet_2	no	ns5:Invoke_1_Create...

These steps should result in these “bpws:propertyAlias” changes in **<your process>.wsdl** file:

```
<bpws:propertyAlias
```

```
propertyName="pns1:Invoke_1_CreateCustomerParty_InputVariable_Create
CustomerPartyEBM_prop"

xmlns:ns1="http://xmlns.oracle.com/EnterpriseServices/Core/CustomerP
arty/V2" messageType="ns1:CreateCustomerPartyReqMsg"
         part="CreateCustomerPartyEBM"
query="/ns2:CreateCustomerPartyEBM/ns3:EBMHeader/ns3:EBMID/@schemeID
"

xmlns:ns2="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Custom
erParty/V2"

xmlns:ns3="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
/>

<bpws:propertyAlias
propertyName="pns1:Invoke_1_CreateCustomerParty_InputVariable_Create
CustomerPartyEBM_prop"
         messageType="client:TestCorrelReceiveMessage"
part="CreateCustomerPartyResponseEBM"

query="/ns2:CreateCustomerPartyResponseEBM/ns3:EBMHeader/ns3:EBMID/@
schemeID"
xmlns:ns2="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Custom
erParty/V2"

xmlns:ns3="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2"
/>
```

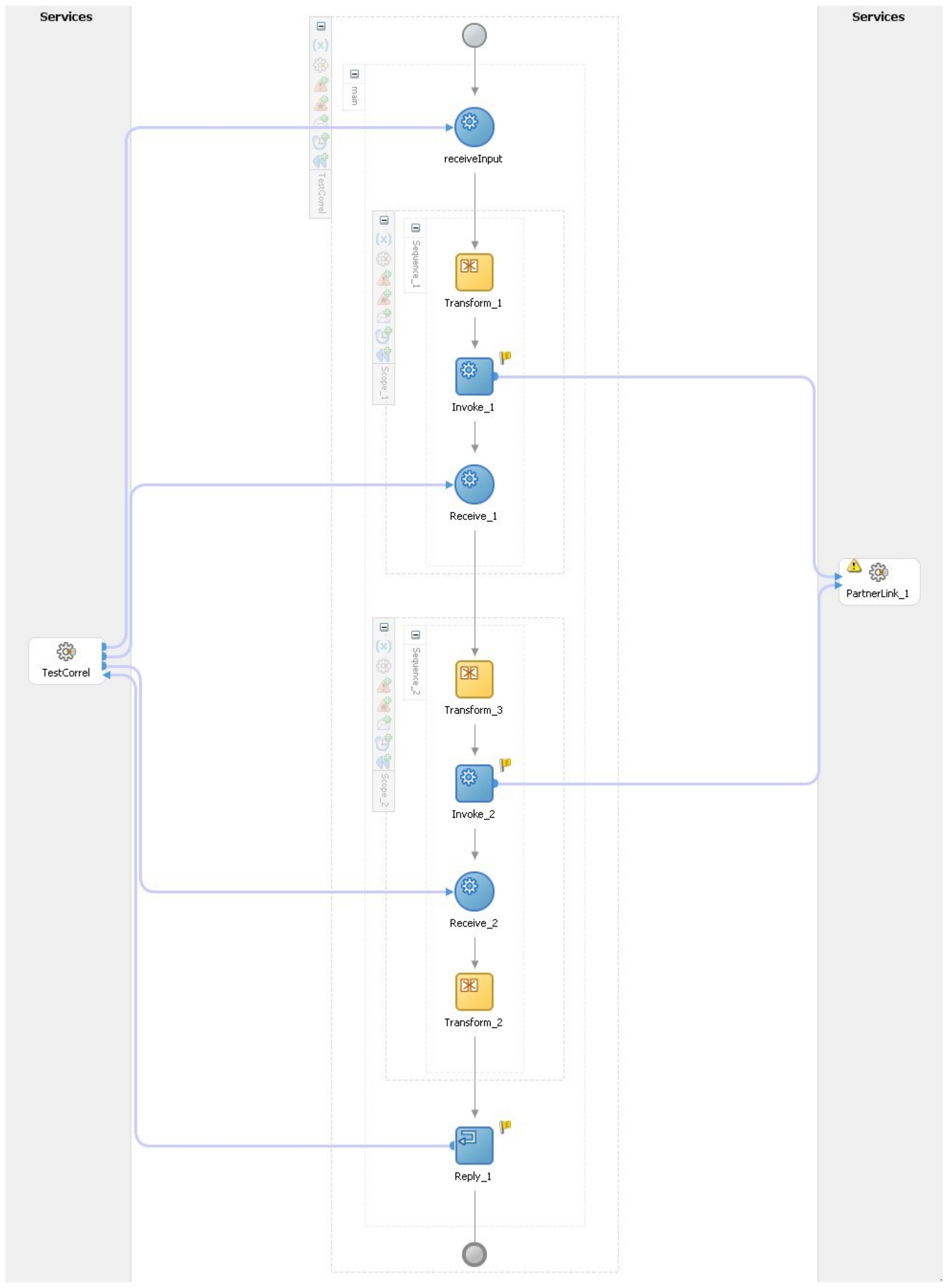
These steps should result in this code generation in **<your process>.bpel** file:

```
  <invoke name="Invoke_1" partnerLink="PartnerLink_1"
                        portType="ns1:CustomerPartyEBS"
                        operation="CreateCustomerParty"

inputVariable="Invoke_1_CreateCustomerParty_InputVariable">
                    <correlations>
                        <correlation initiate="yes"
set="CorrelationSet_1"
                                     pattern="out"/>
                    </correlations>
                </invoke>
                <receive name="Receive_1" createInstance="no"
                         variable="Receive_1_respond_InputVariable"
                         partnerLink="TestCorrel"
portType="client:TestCorrel"
                         operation="respond">
                    <correlations>
                        <correlation initiate="no"
set="CorrelationSet_1"/>
                    </correlations>
                </receive>

          <invoke name="Invoke_2" partnerLink="PartnerLink_1"
                        portType="ns1:CustomerPartyEBS"
                        operation="CreateCustomerParty"
```

```
inputVariable="Invoke_2_CreateCustomerParty_InputVariable">
                    <correlations>
                        <correlation initiate="yes"
set="CorrelationSet_2"
                                     pattern="out"/>
                    </correlations>
                </invoke>
                <receive name="Receive_2" createInstance="no"
                         variable="Receive_2_respond_InputVariable"
                         partnerLink="TestCorrel"
portType="client:TestCorrel"
                         operation="respond">
                    <correlations>
                        <correlation initiate="no"
set="CorrelationSet_2"/>
                    </correlations>
                </receive>
```

4. Separate the "invoke + receive" activities combination into sub-scopes in the main scope of the BPEL process.

   Put the "invoke + receive" activities along with any transformations required under one scope. That is, as you have to invoke two target systems, you need to have two scopes in your BPEL process.



Example of sub-scopes

Implementing Asynchronous Message Exchange Pattern in Provider ABC Services

The one-way service call pattern is used when the Provider ABC service receives the request message from the EBS and ends with the invoking of the provider participating application API. There is no response to the EBS that initiated the request.

In the request delayed response pattern, the Provider ABC service receives the request message from the EBS Request routing service, processes the message, and finally responds to the requesting service (Requestor ABC service or EBF) by invoking the response operation of the EBS Response routing service. The EBS Request routing service is not waiting for the response.

The Provider ABC service is implemented to either behave as a one-way service call pattern or request delayed response pattern.

All the Provider ABC services (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. The "responseCode" attribute on the verb element of the EBM is to be evaluated that the requesting service is expecting a response.

Complete these in the Provider ABC service to behave as both the patterns:

- Populate EBM header of request message in the providing service with fault information

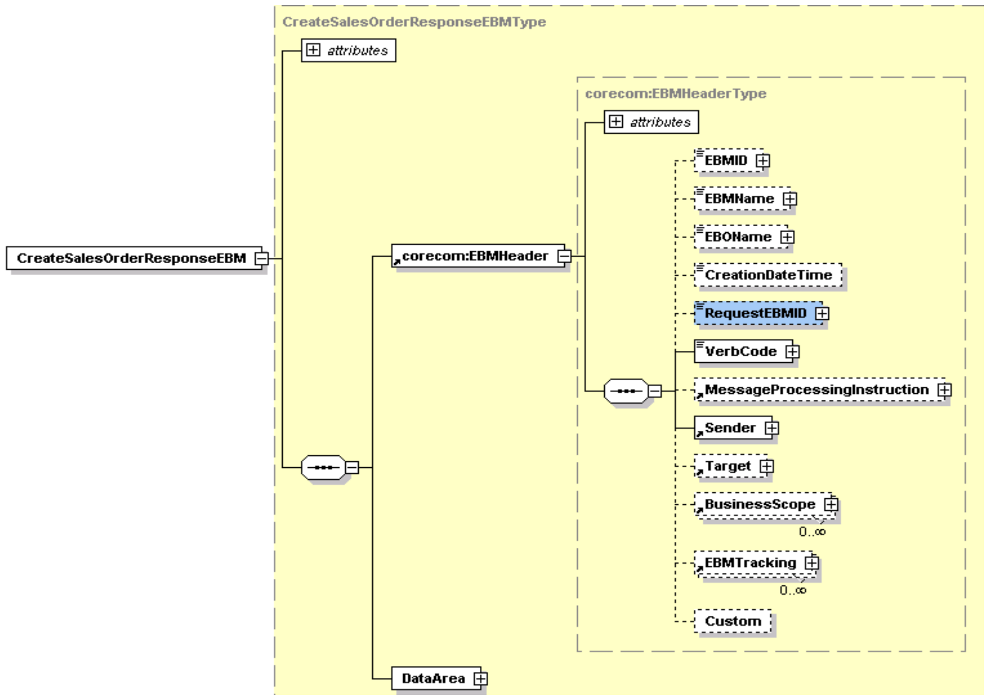
  If there is an error in the provider service before evaluation of the requirement of a response, and an exception is thrown, populate the fault information in the request message EBM header. This will facilitate in passing the entire request EBM along with fault message to the catch block for processing.

- Implement Error Handling in 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. The details are in the 'Error Handling and Compensation' section.

- Populate the correlation information in the EBM header in providing service

  The EBMID in the EBM header is to be used 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.



Structure of CreateSalesOrderResponseEBMType element

The technique of passing the EBMHeader of the request message in a variable into the ABM to EBM XSLT is detailed in Chapter 6: Understanding Message Transformation, Enrichment, and Configuration. This is required to copy the relevant information from the EBM header of the Request EBM to the EBM header of the Response EBM.

- Populate EBM header of response message in the providing service with fault information

  If there is an error in the provider service after evaluation of the requirement of a response, there is a need to populate the fault information in the response message EBM header fault component. This will facilitate in passing the response EBM along with fault message to the catch block for processing.

  The requesting service will receive a response with fault elements populated.

Error Handling in Asynchronous Message Exchange Patterns

In Synchronous Request – Response message exchange patterns, the requesting services are waiting for response. Whenever there is an error in the provider services, an exception is raised and the fault message is propagated back to the requesting service.

In asynchronous fire-and-forget message exchange pattern, the requesting service does not expect any response. If there is an error in the providing service, there may be a need for compensation. In such situations, the compensatory operations in EBSs need to be used for triggering compensations.

In Asynchronous Request –Delayed Response message exchange pattern, 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.

Asynchronous Fire-and-Forget Message Exchange Pattern Error Handling Using Compensatory Operations

For the one-way calls, operations can be put in the EBS (not provided out of the box) to trigger compensation in the requesting services to offset the effects of errors in the provider services. This can be achieved by having compensatory operations in the EBS.

Compensatory operations are modeled as one-way calls. These are modeled as separate operations. For each request - only operation in the request portType, there will be an operation for triggering compensation. For example, CompensateCreateCustomer, CompensateCreateOrder

The Compensatory operations will be invoked in cases where a business exception is likely to result in a fatal error. This is a situation where the conventional retry and resubmit is not possible and the correction is required to be made in the requesting service.

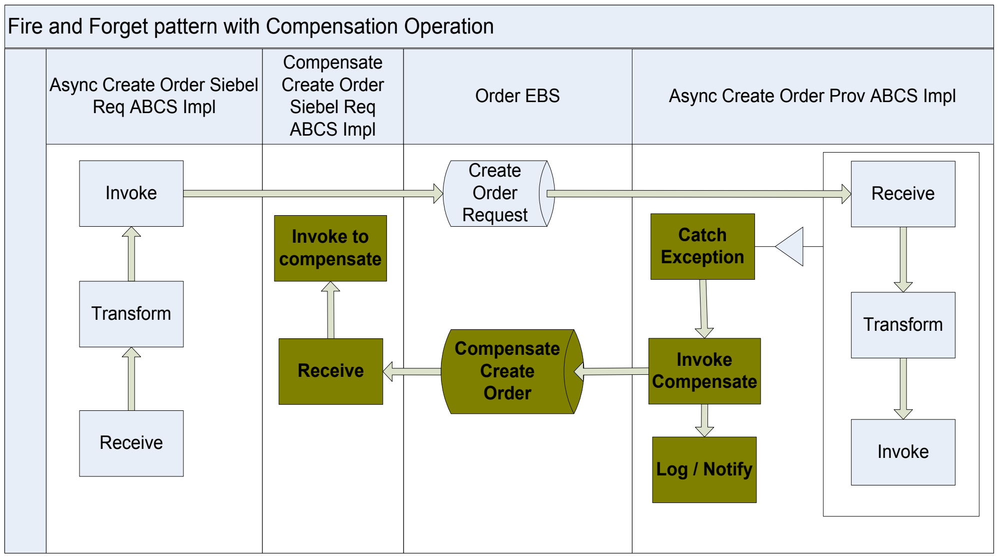
This would call for suitable compensatory services to be implemented taking advantage of the participating applications compensatory action web services or APIs.

Invoking Compensate operation of EBS

As part of Error Handling, for some errors, there may be a need to ensure the compensatory actions are taken. From the providing service, the compensate operation of the EBS will be invoked. The compensate operation of the EBS will route to the correct compensating service.

Follow these steps:

1. In the providing service, in case of an error, raise an exception and catch it in the catch block.
2. In the catch block, construct the Request EBM along with fault component in the EBM header.
    a. Create a transform step and select the input variable representing the Request EBM and the compensate variable, also representing the Request EBM.
    b. Pass the fault message generated from the exception as a variable into the input variable to compensate variable XSLT. Please refer to this guide for the technique.
    c. Map to the compensate variable:
        - Standard EBM header content from the Request EBM
        - Data Area from the Request EBM
        - Fault message
    d. Set the ‘InvokeCompensate’ step to invoke the corresponding compensate operation in the request EBS routing service.
    e. Route the compensate request to a suitable compensating service.


Fire and Forget pattern with Compensation Operation
Async Create Order Siebel Req ABCS Impl
Compensate Create Order Siebel Req ABCS Impl
Order EBS
Async Create Order Prov ABCS Impl
Invoke
Transform
Receive
Invoke to compensate
Receive
Create Order Request
Compensate Create Order
Catch Exception
Invoke Compensate
Log / Notify
Receive
Transform
Invoke

Fire-and-forget pattern with compensation operation

Enable Routing Rule in Compensate Operation Routing Service

There will be two routing rules:

- 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>

- Routing rule for CAVS - CAVS at present does not have the capability to accept an asynchronous response for a request initiated from CAVS.

Asynchronous Request – Delayed Response Message Exchange Pattern Error Handling

In Asynchronous Request –Delayed Response message exchange pattern, 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.

Follow these steps:

1. In the providing service, in the case of an error, 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 these to the output variable:
   a. Standard EBM header content from the Request EBM including the correlation information
   b. Fault message
   c. Set the 'Invoke' step to invoke the response operation in the response EBS routing service.
   d. Route the response from the providing service to the correct requesting service.

Working with the Artifacts Generator

You can use the Artifacts Generator to generate much of the common code needed to create ABC service implementations in BPEL.

For more information about the Artifacts Generator, see *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* "Working with the Artifacts Generator."

This section discusses how to:

- Use the Artifacts Generator
- Create the Artifacts Generator input.xml file
- Complete Artifacts Generator ABC service development

Using the Artifacts Generator

To use the Artifacts Generator:

1. Copy the AIA Service generator folder from the Foundation Pack installation to your development machine.

   The AIA Service generator is located in FPHome\Infrastructure\DeveloperTools\ArtifactsGenerator.

2. Download Freemarker Template Engine version 2.3, minor version 13 or higher from http://www.freemarker.org/freemarkerdownload.html.

3. From the archive, extract lib\freemarker.jar

4. Copy freemarker.jar into AIAServiceGenerator\lib folder

5. Open the command line window and set the path to ***JDK 1.5*** and ***ant 1.6.***

6. Navigate to the AIA Service generator folder.

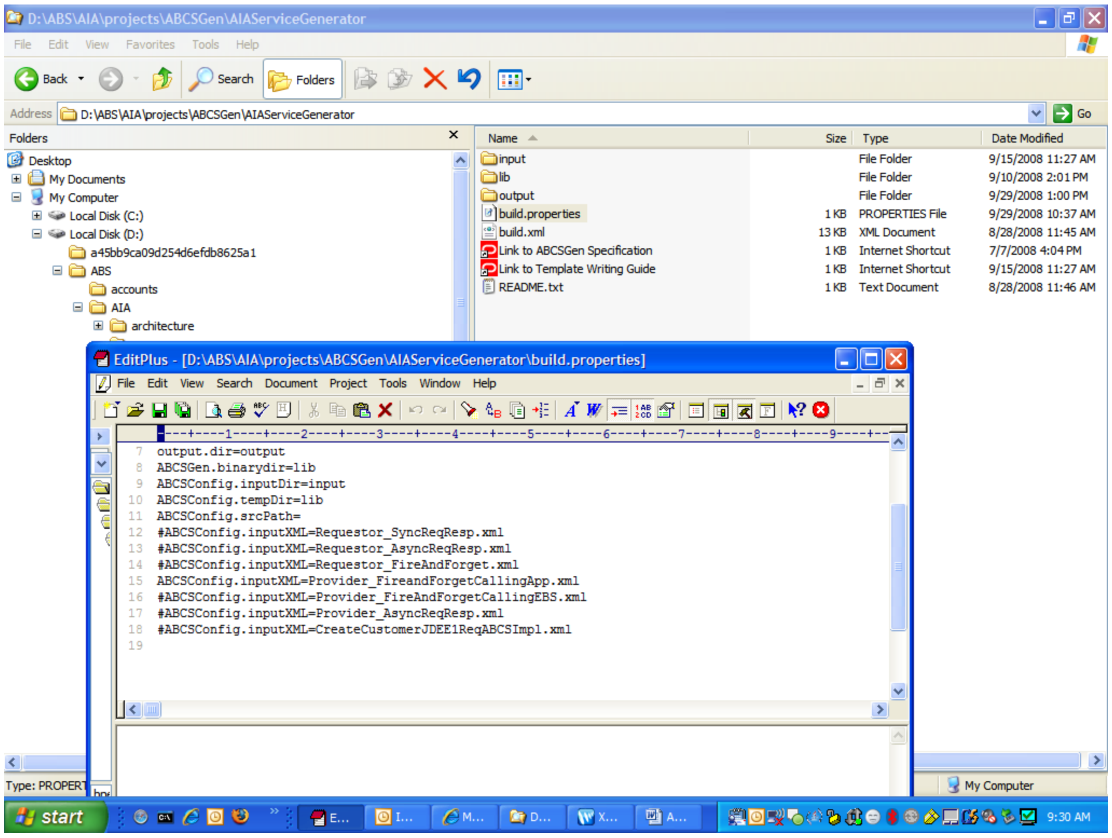
7. Using a text editor, open a sample input.xml located in the AIA Service Generator\input folder.

8. Create an input.xml file as described in Creating the Artifacts Generator input.xml File.

9. Save the input.xml file.

10. Navigate to the AIA Service Generator folder and open the build.properties file using a text editor.

11. Change the uncommented ***ABCSConfig.inputXML= <Your input file name>*** value to point to the input.xml file you created in step 5.



D:\ABS\AIA\projects\ABCSGen\AIAServiceGenerator

Address D:\ABS\AIA\projects\ABCSGen\AIAServiceGenerator

Name	Size	Type	Date Modified
input		File Folder	9/15/2008 11:27 AM
lib		File Folder	9/10/2008 2:01 PM
output		File Folder	9/29/2008 1:00 PM
build.properties	1 KB	PROPERTIES File	9/29/2008 10:37 AM
build.xml	13 KB	XML Document	8/28/2008 11:45 AM
Link to ABCSGen Specification	1 KB	Internet Shortcut	7/7/2008 4:04 PM
Link to Template Writing Guide	1 KB	Internet Shortcut	9/15/2008 11:27 AM
README.txt	1 KB	Text Document	8/28/2008 11:46 AM

EditPlus - [D:\ABS\AIA\projects\ABCSGen\AIAServiceGenerator\build.properties]

```
output.dir=output
ABCSGen.binarydir=lib
ABCSConfig.inputDir=input
ABCSConfig.tempDir=lib
ABCSConfig.srcPath=
#ABCSConfig.inputXML=Requestor_SyncReqResp.xml
#ABCSConfig.inputXML=Requestor_AsyncReqResp.xml
#ABCSConfig.inputXML=Requestor_FireAndForget.xml
ABCSConfig.inputXML=Provider_FireandForgetCallingApp.xml
#ABCSConfig.inputXML=Provider_FireAndForgetCallingEBS.xml
#ABCSConfig.inputXML=Provider_AsyncReqResp.xml
#ABCSConfig.inputXML=CreateCustomerJDEE1ReqABCSImpl.xml
```

Input.xml file name defined in the build.properties file

12. Save the build.properties file.

13. From the command line window in the AIA Service Generator folder, run Ant buildBPEL.

14. The output of the AIA Service Generator is written to the output directory as a compilable project.

15. Open the project from step 11 in JDeveloper and compile it.

16. Although the service created by the Artifacts Generator is compilable, it is not in an executable form.

    You must first review all of the artifacts generated by the utility and ensure that the code generated addresses requirements.

17. Once you have reviewed all of the artifacts, complete the tasks in Completing Artifacts Generator ABC Service Development to create a fully functional ABC service.

Creating the Artifacts Generator input.xml File

You must create the input.xml file to include the information needed by the Artifacts Generator to create the ABC service. The information you need to create your input.xml file should be available to you once your ABC service technical design is complete. You can use one of the delivered sample input.xml files as a template.

Edit the values in the input.xml to suit the ABC service you are creating using the Artifacts Generator. The schema used for constructing this XML document is available in AIA Service Generator\lib\xsd. This xsd contains detailed documentation of each element. Each input.xml element is described in this table.

Name	Details
ABCSType	Role played by the ABC service - Requestor or Provider.
Verb	Operation being implemented by the ABC service, such as Create, Update,, and so on.
Industry	Industry for which this service is generated.
ParticipatingAppShortName	Short name of the participating application for which this service is generated. **For more information** about guidelines for creating participating application short names, see Participating Applications Names.
ParticipatingAppLongName	Long name of the participating application for which this service is generated.
ParticipatingAppDefaultID	Default SystemID of the participating application.
ProductPillar	Family or product line to which this product belongs. For example, CRM, Financials, HCM, SCM, Retail, Communications, or Financial Services.
ProductFamily	Family or product line to which this product belong. For example, Siebel, Ebiz, PeopleSoft, Fusion, JDEOne, JDEWorld, CRM on Demand, Retail, OracleServicesManager, or BRM.
ProductCode	Product code of the product family for which the ABC service is being developed. This should be the same as the product family's application module.
ABCSVersionNumber	Version of the ABC service.
EOLLocation	URL pointing to the location of the Enterprise Object Library using the format [hostName]:[portName].
GenerateErrorHandlingIndicator	Indicates whether the process-level error handling code

Name	Details
	needs to be generated. Values are: • True • False
EnableExtensionIndicator	Indicates whether the ABC service extension-enabling code needs to be generated. This will indicate whether or not the extension wsdl, pre-transformation extension, and post-transformation extension need to generated. Values are • True • False
ExtensionWSDLRuntimeLocation	Location of the concrete WSDL for the ABC service's extension service.
ServiceObjectName	EBO or Application Business Object (ABO) for which this service is implemented.
ServiceObjectVersion	Version of the service object.
ServiceObjectNamespacePrefix	Namespace prefix for the target namespace of this business object.
ServiceObjectNamespace	Name of the target namespace for this business object.
ServiceObjectSchemaLocation	Location of the schema needed for the service object.
FaultNamespacePrefix	Namespace prefix for the namespace of this fault object
FaultNamespace	Name of the target namespace for the fault element
FaultSchemaLocation	Location of the schema needed for the service object
ServiceMEP	Message exchange pattern implemented by this specific service operation. Values are: • SyncReqResp • FireAndForget • AsyncReqResp
EnableDynamicEndpointIndicator	Indicates whether CAVS-enabling related code needs to be generated. Values are: • True • False
GenerateScopeErrorHandlingIndicator	Indicates whether error handling code needs to be generated for this BPEL scope. Values are: • True • False
ScopeName	Name of the BPEL scope in which the BPEL activities for

Name	Details
	invoking this specific service's PartnerLink will be included.
ServiceNamespacePrefix	Namespace prefix for the target namespace of this service.
ServiceNamespace	Name of the target namespace for this service.
ServiceWSDLLocation	Location of the wsdl for this target service.
Operation	Name of this service's operation. The service could be an ABC service, EBS, or application service.
PortTypeName	Name of this service's port type. The service could be an ABC service, EBS, or application service. The relevant WSDL should contain the port type.
PartnerRole	Set the value to the partner role in the partner link type definition in the target WSDL. This is required only if GenerateReferenceWSDLFileIndicator is set to false.
PartnerLinkType	Set the value to the name of the partner link type definition in the target WSDL. This is required only if GenerateReferenceWSDLFileIndicator is set to false.
InputMessage	Name of the input message used as input for this service operation.
InputMessageElement	Name of the element bound to the input message and used as the input for this service operation.
OutputMessage	Name of the output message used as output for this service operation.
OutputMessageElement	Name of the element bound to the output message and used as the output for this service operation.
FaultMessage	Name of the fault message used as the fault for this service operation.
FaultMessageElement	Name of the element bound to the fault message and used as the fault for this service operation.

Completing Artifacts Generator ABC Service Development

To complete development of an ABC service whose artifacts were generated by the Artifacts Generator:

1. Rearrange service invocations, if necessary.

   The Artifacts Generator generates relevant definitions for all external service invocations in a sequence in the order specified in the input xml. If your ABC service requires that the invocations be done either in parallel or based on certain other conditions, rearrange the services as needed using Oracle BPEL Designer.

2. Complete the business payload transformation.

   The Artifacts Generator generates transformations only if the target document is an enterprise business message and generates root element business transformations only. It generates the transformation mapping pertaining to most of the header elements, as well as the XSL code pertaining to the business payload that is common to all of the ABC services.

   Use Oracle BPEL Designer to develop the transformation code for transforming remaining parts of the message document to complete the business payload and header transformation.

3. Configure correlation properties for asynchronous application service invocation.

   The Artifacts Generator does not generate correlation properties for any application target service that is being invoked using an asynchronous request/response message exchange pattern.

   Use Oracle BPEL Designer to configure the correlation upon the invoke and subsequent receive.

4. Create invocations for other operations when a service is called multiple times.

   If a service is called multiple times, the Artifacts Generator does not generate code for multiple invocations. Copy and paste the invocation code generated for the first service invocation from the same process and rename operations and variables names.

5. Alter deployment descriptor properties in the bpel.xml file.

   The Artifacts Generator generates deployment descriptor properties in the bpel.xml file with default values. Modify these default values based on the integration scenario’s requirements.

6. Alter policies in the fault policy.

   The Artifacts Generator generates fault policies with default values. Modify these default values based on the service’s requirements.

7. Alter properties in the configuration properties file snippet and add it to your properties.

   The Artifacts Generator generates a configuration file snippet for service-specific properties. Copy this property segment from the snippet and add it to the Oracle AIA Configuration Properties file in the generated project.

Interacting with Participating Applications

Communication between ABC services and the application is governed by the application capabilities. Communication with applications can be done using any of these technologies:

- JCA adapters, when available and authorized for use.
- Standard Web Service Interfaces (SOAP/HTTP, XML/HTTP).
- Adapters, such as database, file, JMS, or AQ.

- Oracle ESB Resequencer.

Native JCA Adapters

The JCA Adapter is the most effective means to communicate with applications because it natively invokes participating applications in a co-located fashion. JCA Adapters should be the first choice for developers to connect to applications; however other factors may affect the decision of using JCA Adapters such as:

- Availability of the JCA Adapter with Oracle.
- Availability of third-party JCA Adapters and the willingness to pay license fees.
- The functionality available through the JCA Adapter.

Web Service Interface

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.

JMS Adapter with AQ

In situations when the message submission is decoupled from the processing of the message, usage of queues is suggested. When guaranteed delivery of the message is a requirement, usage of Advanced Queuing (AQ) for persistence is suggested.

When JMS queues are used, they have to be accessed using JMS Adapter. If the participating application has capabilities to use JMS adapter, then the message will be enqueued to the JMS queue by the participating application and a JMS Consumer Service would dequeue the message.

In the case of participating applications not having the capability to use JMS adapter, a JMS Producer Service will be invoked by the participating application, which will enqueue the message.

For more information, see Chapter 8: Designing and Constructing JMS Adapter Services.

Oracle Enterprise Service Bus Resequencer

This section provides an overview of the Oracle ESB Resequencer feature and discusses how to set up and use it in the Oracle AIA asynchronous message exchange pattern.

Overview

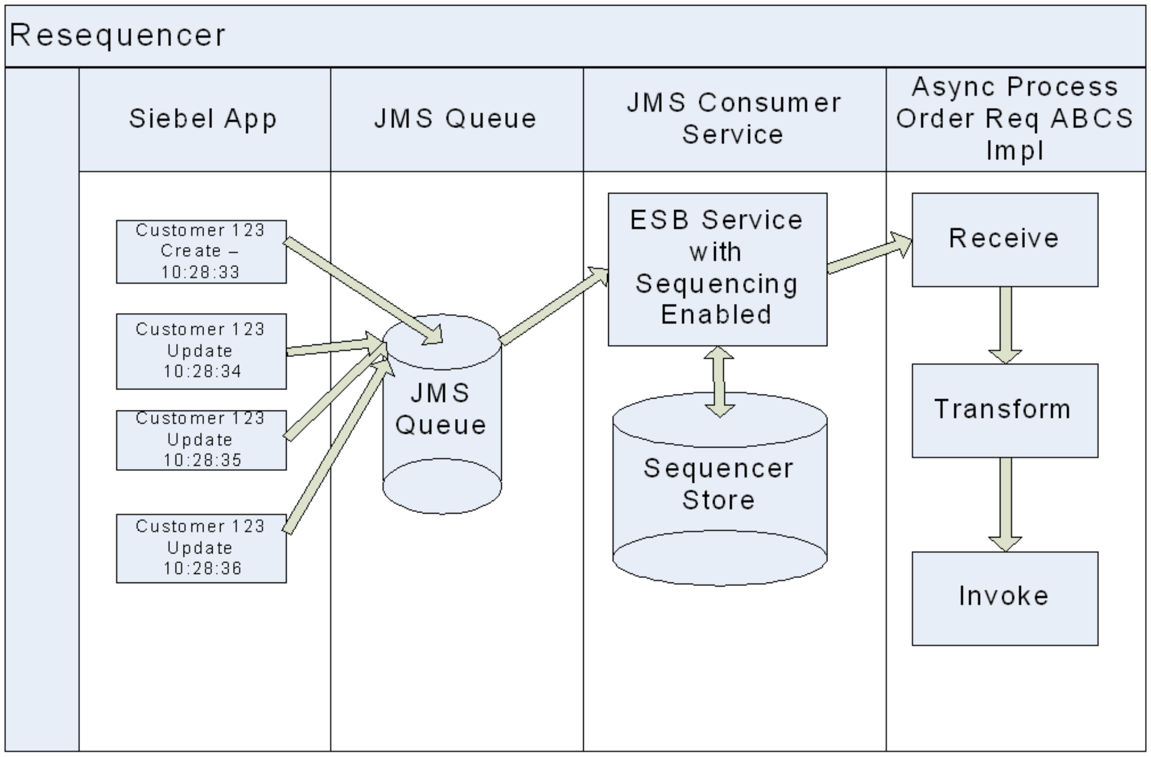
In the Oracle AIA asynchronous message exchange pattern, participating applications push messages to the Oracle AIA layer where they are processed in a store-and-forward fashion.

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. Additionally, there is a need to ensure guaranteed delivery of messages to the destination.

The Oracle SOA Suite 10.3.4 ESB Resequencer helps to maintain correct processing sequence. 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.

For more information about the Oracle ESB Resequencer feature, see Support for Resequencing.

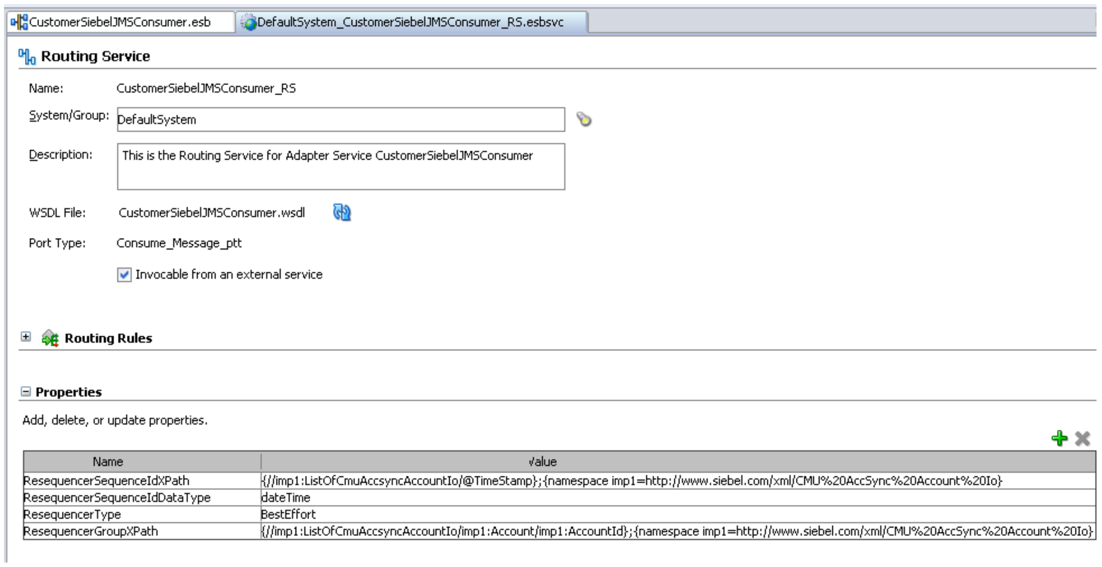
As this is a feature of the Oracle ESB product, an ESB service must be in place for it to work as designed.


Resequencer
Siebel App
JMS Queue
JMS Consumer Service
Async Process Order Req ABCS Impl
Customer 123 Create – 10:28:33
Customer 123 Update 10:28:34
Customer 123 Update 10:28:35
Customer 123 Update 10:28:36
JMS Queue
ESB Service with Sequencing Enabled
Sequencer Store
Receive
Transform
Invoke

Required ESB service configuration for use with the Oracle ESB Resequencer

Configuring the Oracle ESB Routing Service to Use the Resequencer

This section discusses how to configure the Oracle ESB routing service to enable use of the Oracle ESB Resequencer feature.



CustomerSiebelJMSConsumer.esb | DefaultSystem_CustomerSiebelJMSConsumer_RS.esbsvc

Routing Service

Name:	CustomerSiebelJMSConsumer_RS
System/Group:	DefaultSystem
Description:	This is the Routing Service for Adapter Service CustomerSiebelJMSConsumer
WSDL File:	CustomerSiebelJMSConsumer.wsdl
Port Type:	Consume_Message_ptt
	✓ Invocable from an external service

Routing Rules

Properties

Add, delete, or update properties.

Name	Value
ResequencerSequenceIdXPath	{//imp1:ListOfCmuAccsyncAccountIo/@TimeStamp};{namespace imp1=http://www.siebel.com/xml/CMU%20AccSync%20Account%20Io}
ResequencerSequenceIdDataType	dateTime
ResequencerType	BestEffort
ResequencerGroupXPath	{//imp1:ListOfCmuAccsyncAccountIo/imp1:Account/imp1:AccountId};{namespace imp1=http://www.siebel.com/xml/CMU%20AccSync%20Account%20Io}

ESB routing service configurations set to enable use of the Oracle ESB Resequencer

To configure the Oracle ESB routing service to enable use of the Oracle ESB Resequencer:

1. Determine and set the group ID.

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, and so forth. The group ID is declared by an XPath expression for the ***ResequencerGroupXPath*** property on the ESB routing service.

It should follow this format:

Name	ResequencerGroupXPath
Value	{//imp1:ListOfCmuAccsyncAccountIo/imp1:Account/imp1:AccountId};{namespace imp1=http://www.siebel.com/xml/CMU%20AccSync%20Account%20Io}

Note: A wild card (*) can be used in the XPath expression. This will allow the ESB service to accept messages with different root elements. However, the namespaces of all of the messages must be the same. For example: {//*/*/imp1:AccountId};{namespace imp1=http://www.siebel.com/xml/CMU%20AccSync%20Account%20Io}

2. Determine and set the resequencer ID.

The resequencer ID is a parameter (field or attribute) on the incoming message that indicates the position of the message in the sequence. This can be an integer or a date-time data type. The resequencer ID is declared by an XPath expression for the ***ResequencerSequenceIdXPath*** property on the ESB routing service. If the data type is of date-time, it must be declared as a ***ResequencerSequenceIdDataType*** property.

It should follow one of these formats:

Name	ResequencerSequenceIdXPath
Value	{//imp1:ListOfCmuAccsyncAccountIo/@TimeStamp};{namespace imp1=http://www.siebel.com/xml/CMU%20AccSync%20Account%20Io}

Note: A wild card (*) can be used in the XPath expression. This will allow the ESB service to accept messages with different root elements. However, the namespaces of all of the messages must be the same. For example: {//*/@TimeStamp};{namespace imp1=http://www.siebel.com/xml/CMU%20AccSync%20Account%20Io }

Name	ResequencerSequenceIdDataType
Value	dateTime

3. Determine and set the resequencing type.

   There are three resequencing types:

   - 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.

4. Configure the Resequencer for transaction and multi-threading.

   The Oracle ESB Resequencer initiates a global transaction when it picks up a message from a particular group and processes it. For this purpose, the routing rule execution should be set to ***Asynchronous.*** This ensures that the target service is invoked with a global transaction in place.

The Resequencer provides the ***ResequencerWorkerThreadPoolSize*** property that should be set for each routing service. Set it to a suitable number to manage the load of incoming messages.

Describing Oracle ESB 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 retryable errors.

The processing of messages for other groups continues without interruption.

If new messages arrive for the errored group, messages are stored in the Oracle ESB Resequencer database so that no messages are lost. When failed messages are resubmitted from the Oracle ESB Console or another tool, the associated errored group is unlocked and normal processing resumes for that group.

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

Testing ABC Services

For more information, see Chapter 12: Developing Oracle AIA Services to be CAVS-Enabled and Provide Multi-Instance Support.

Recognizing I18N Considerations for ABC Services

In general, applications publish and receive data in a locale neutral format, however this cannot be always guaranteed. In cases where the ABO is locale dependent, the ABC services that pass the EBOs to the actual service providers will be responsible for translating the document into a locale-specific ABO. Similarly, the ABC service will be responsible for translating the locale-specific ABO into locale-agnostic EBO.

The ABC service deciphers the locale based on the locale preferences of the end user of the relevant participating application. This data will provide information about how the locale-specific ABO needs to be constructed, as well as how the locale-specific ABO needs to be interpreted.

The ABC service should specify the locale in which the response needs to be provided by the real service provider. For example, the ABC service for Get Product Details EBS needs to specify the locale in which the product details need to be provided by the Oracle eBusiness Suite application. If the requester wants the product details in Spanish, the Get Product Details EBS needs to instruct the real service provider that the product details need to be returned according to the Spanish locale.

Examples of Locale Impedance Mismatch

It is not very common for applications to send and receive applications in a locale depended fashion, however there may be cases where this may occur. Here are some examples of how you may have impedance mismatch between data provided by an application and the canonical ISO representation of data in EBMs:

- Number Impedance Mismatch

  The application is running in French and sends a number in the French locale representation as 123,45 where a comma is used as the decimal point symbol. In this case, the ABC service implementation is responsible for converting the number into its canonical representation as 123.45 to make it locale neutral.

- Date Impedance Mismatch

  The application is running in United Kingdom locale and sends a date in the UK locale representation as 12-Jan-2007. In this case, the ABC service implementation is responsible for converting this date into its canonical representation as 2007-01-12 to make it locale neutral.

Chapter 5: Designing and Constructing EBFs

This chapter discusses how to:

- Define and create the contract for an enterprise business flow (EBF).
- Implement the enterprise business flow as a BPEL service.

Defining and Creating 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 needs to be defined and created prior to the implementation of the EBF.

To define the contract for an EBF:

1. Identify the EBF.
2. Identify the pattern for the EBF.
3. Identify enterprise business messages (EBM) to be used for the requests and responses (if any).

To create the contract for an enterprise business flow:

1. Identify the EBM.
2. Construct the WSDL for the EBF.
3. Annotate the service interface.
4. Ensure WS-1 basic profile conformance.

Identifying the EBF

The first task involved in designing a new service is to verify whether it is necessary. There might be situations where other services exist which are already providing some or all of the features identified. Each of the services as well as the operation's description and any metadata should be reviewed before deciding to create either a new service or an operation.

An EBF is needed when an enterprise business service (EBS) operation needs to be implemented with a set of tasks and involves invoking of multiple services.

An EBF can invoke only another EBS. In no situation should an EBF invoke an application business connector (ABC) service directly.

Identify the Message Pattern for EBF

The enterprise business flow is modeled to implement a single operation.

To identify the pattern for an EBF:

1. Based on the understanding of the business process requirement from the FDD, identify the triggering event for the EBF.
2. If the control is to be blocked until a response is returned back to the point of invocation, then choose EBF Request - Reply Pattern. This would be a synchronous call.
3. 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.
4. Next check whether the execution of the EBF results in a response. Is there a need to correlate the request and the response?
5. 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 EBF fire-and-forget pattern.
6. Any EBF operation invoked as a result of a subscription to a publish event would use the EBS Subscribe Pattern

Constructing the WSDL for the EBF

Since the EBF development starts with constructing a WSDL, the end result of the technical design process is an EBF WSDL.

Please use the Template EBF wsdl and Sample EBF wsdl from “AIAServices-TemplateSampleWSDLs” creating a WSDL.

Implementing the EBF as a BPEL Service

The EBF:

- Provides implementation with multiple steps for being invoked from an EBS.
- Invokes another EBS.

For more information about the details of creating BPEL projects in JDeveloper**,** see the *Oracle BPEL Process Manager Developer's Guide*.

To create an EBF:

1. Create a new 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 process.

   Please follow Chapter 4: Designing and Constructing ABC Services for details. The difference is that the EBF deals with EBMs.

3. Enable error handling and logging.

   EBSs should handle errors to allow clients or administrators to resubmit or re-trigger processes. This will be done through a central error handler.

For more information, see Chapter 13: Configuring Oracle AIA Processes for Error Handling and Trace Logging.

4. Enable extensibility points in EBF.

For more information, see Chapter 15: Extensibility for Oracle AIA Artifacts.

Chapter 6: Understanding Message Transformation, Enrichment, and Configuration

This chapter discusses:

- Standards for transforming documents.
- Cross-references.
- Domain value maps (DVMs).
- Oracle Application Integration Architecture (AIA) configuration file.

Standards for Transforming Documents

A set of best practice standards should be followed when transforming documents. These standards will ensure extensibility so that customers can easily switch over to a different transformation logic or provide transformation extensions for the custom area.

This section discusses:

- Naming standards
- XSLT extensibility

Naming Standards

Each transformation type should be placed in a separate core XSL file. The file name should closely follow this convention where possible:

```
<Source Schema Type>_to_<Destination Schema Type>.xsl
```

Example:

CreateOrderEBM_to_CreateOrderSiebelABO.xsl

XSLT Extensibility

Every component in the enterprise business message (EBM) including the EBM header contains a custom area that can be configured by customers to add the necessary elements. To allow customers to act on this content, enrich it, or transform it, you should develop the core transformation XSL file to provide extension capabilities using these guidelines:

- Create an extension XSL file for every transformation.

There will be one extension XSL file for every core XSL file name, for example:

```
<XSL file name>_Custom.xsl
```

- Define empty XSL templates in the extension file for every component in the message.
- Include the extension file in the core transformation file.
- Add call-template for each component in core transformation.

An example of a core transformation XSL file:

PurchaseOrder_to_PurchaseOrder.xsl

Its custom extension file: PurchaseOrder_to_PurchaseOrder_Custom.xsl.

PurchaseOrder_to_PurchaseOrder.xsl snippet

This XSL file transforms PurchaseOrder to PurchaseOrder. It includes the PurchaseOrder_to_PurchaseOrder_ Custom .XSL and then calls out to templates defined in the custom XSL file for each of the components transformed in the core file.

```
<!-Include Extension file for Customer Extensions -- >
<xsl:include href="PurchaseOrder_to_PurchaseOrder_Custom.xsl"/>

<xsl:template match="/">
<po:purchaseOrder>
<shipTo>
<name>
<xsl:value-of select="/po:purchaseOrder/shipTo/name"/>
</name>
<street>
<xsl:value-of select="/po:purchaseOrder/shipTo/street"/>
</street>
<city>
<xsl:value-of select="/po:purchaseOrder/shipTo/city"/>
</city>
<state>
<xsl:value-of select = "/po:purchaseOrder/shipTo/state" />
</state>
<zip>
<xsl:value-of select="/po:purchaseOrder/shipTo/zip"/>
<zip>
<xsl:call-template name="shipTo_ext">
<xsl:with-param name="currentNode" select="."/>
</xsl:call-template>
</shipTo>
<billTo>
  <name>
<xsl:value-of select="/po:purchaseOrder/billTo/name"/>
</name>
<street>
<xsl:value-of select="/po:purchaseOrder/billTo/street"/>
</street>
<city>
<xsl:value-of select="/po:purchaseOrder/billTo/city"/>
</city>
<state>
```

```
<xsl:value-of select="/po:purchaseOrder/billTo/state"/>
</state>
<zip>
<xsl:value-of select="/po:purchaseOrder/billTo/zip"/>
<zip>
<xsl:call-template name="billTo_ext">
<xsl:with-param name="currentNode" select="."/>
</xsl:call-template>
<billTo>

<CreditCard>
  <FirstName>
<xsl:value-of select="/po:purchaseOrder/CreditCard/FirstName"/>
</FirstName>
<LastName>
<xsl:value-of select="/po:purchaseOrder/CreditCard/LastName"/>
</LastName>
<CardNumber >
<xsl:value-of select="/po:purchaseOrder/CreditCard/CardNumber"/>
</CardNumber>
<xsl:call-template name="credit_ext">
<xsl:with-param name="currentNode" select="."/>
</xsl:call-template>
</CreditCard>

<items>
<xsl:for-each select="/po:purchaseOrder/items/item">
<item>
<productName>
<xsl:value-of select="productName"/>
</productName>
<quantity>
<xsl:value-of select="quantity"/>
</quantity>
<USPrice>
<xsl:value-of select="USPrice"/>
</USPrice>
<po:comment>
<xsl:value-of select="comp:comment"/>
</po:comment >
<shipDate>
<xsl:value-of select="shipDate"/>
</shipDate>
<xsl:call-template name="item_ext">
<xsl:with-param name="currentNode" select="."/>
</xsl:call-template>
</item>
</xsl:for-each>
</items>
</po:purchaseOrder>
</xsl:template>
</xsl:stylesheet>
```

PurchaseOrder_to_PurchaseOrder_Custom.xsl

This custom extension XSL file contains empty templates for all of the components being transformed in the core XSL file. You can modify this file to add transformations to the empty templates to append extra transformations to the original core XSL file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<?oracle-xsl-mapper <!--
SPECIFICATION OF MAP SOURCES AND TARGETS, DO NOT MODIFY. -- >
<mapSources>
<source type="XSD">
<schema location="http://machine:port/schemas/
Extension/core/PO.xsd"/>
<rootElement name="purchaseOrder" namespace="
http://www.example.com/PO"/>
</source>
</mapSources>
<mapTargets>
<target type="XSD">
<schema location="http://machine:port/schemas/
Extension/custom/NativeOrder.xsd"/>
<rootElement name="purchaseOrder" namespace=
"http://www.example.com/PO”/>
</target>
</mapTargets>

?>
<xsl:stylesheet version="1.0"
xmlns:bpws="http://schemas.xmlsoap.
org/ws/20 03/03/business-process/”
xmlns:ehdr=
"http://www.oracle.com/XSL/Transform/java/oracle.tip.
esb.server.headers.ESBHeaderFunctions"
xmlns:ns0="http://www.w3.org/20 01/XMLSchema"
xmlns:hwf="http://xmlns.oracle.com/bpel/
workflow/xpath"
xmlns:comp="http://www.example.com/Components"
xmlns:xp2 0=
"http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.
services.functions.Xpath2 0"
xmlns:xref=
"http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.
xpath.XRefXPathFunctions"
xmlns:po="http://www.example.com/PO"
xmlns:xsl="http://www.w3.org/19 99/XSL/Transform"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:po="http://www.example.com/PO"
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"
exclude-result-prefixes="xsl ns0 comp
po po bpws ehdr hwf xp2 0 xref ora ids orcl">

<xsl:template name="shipTo_ext">
<xsl:param name="currentNode"/>
```

```
<!- Customers add transformations here-- >
</xsl:template>

<xsl:template name="billTo_ext">
<xsl:param name="currentNode"/>
<!- Customers add tranformations here-- >
</xsl:template>

<xsl:template name="credit_ext">
<xsl:param name="currentNode"/>
<!- Customers add tranformations here-- >
</xsl:template>

<xsl:template name="item_ext">
<xsl:param name="currentNode"/>
<!- Customers add tranformations here-- >
</xsl:template>

</xsl:stylesheet>
```

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 as well as command line utilities to import and export data.
- XPath functions to query, add, and delete cross-references.

For more information about cross-references, see the *Oracle Cross Reference User Guide*.

Setting Up Cross-References

For more information about setup instructions, see the *Oracle Cross Reference User Guide*.

Populating Cross-References

The cross-references feature has an import/export utility to help with initial data loads. The Cross Reference Import Export Tool Reference can be found in the *Oracle Cross Reference User Guide.*

A command line admin tool is also available to help configure the cross-references virtual table. The Cross Reference Admin Tool Reference can be found in the *Oracle Cross Reference User Guide*.

Cross-Reference APIs

There are three different types of operations that can be performed against the Xref data:

- Populate Xref
- Lookup Xref
- Delete Xref

Each uses a separate XPath or XSL extension function. The functions are:

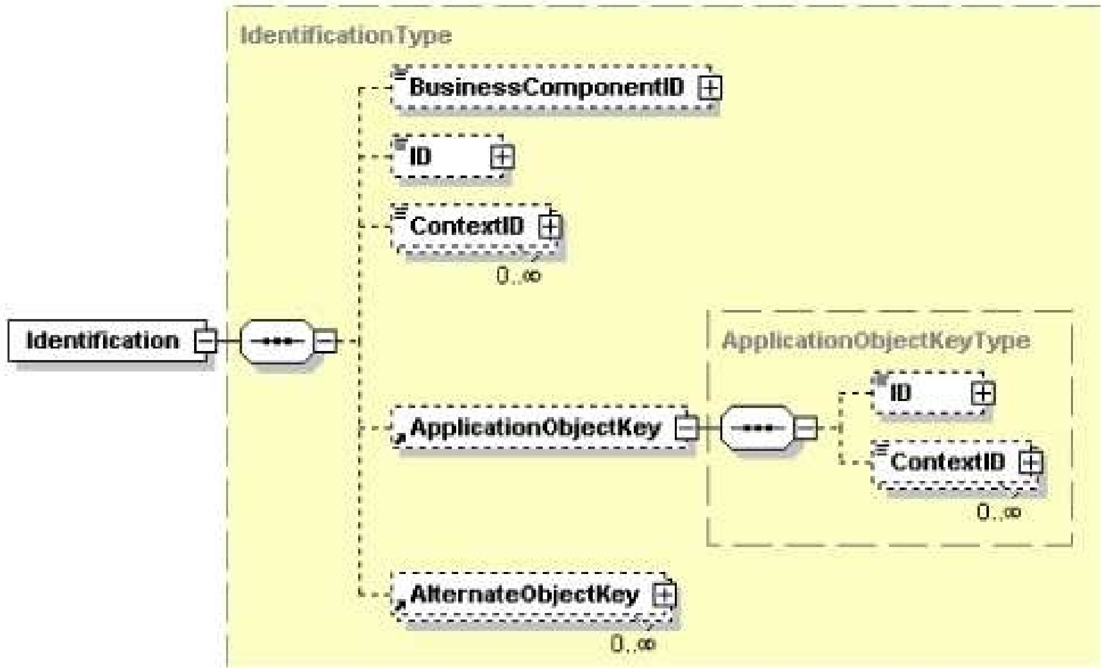
```
xref:populateXRefRow( xrefTableName as string,
xrefReferenceColumnName as string,
xrefReferenceValue as string,
xrefColumnName as string,
xrefValue as string,
mode as string) return xrefValue as string

xref:populateXRefRow1M( xrefTableName as string,
xrefReferenceColumnName as string,
xrefReferenceValue as string,
xrefColumnName as string,
xrefValue as string,
mode as string) return xrefValue as string

xref:lookupXRef( xrefTableName as string,
xrefReferenceColumnName as string,
xrefReferenceValue as string,
xrefColumnName as string,
needAnException as boolean) return as string

xref:lookupXRef1M(  xrefTableName as string,
xrefReferenceColumnName as string,
xrefReferenceValue as string,
xrefColumnName as string) return as node-set

xref:markForDelete( xrefTableName as string,
   xrefColumnName as string,
xrefValueToDelete as string) return asboolean
```

Populating the EBO Object Identification

Each of the EBMs will have at least one element to hold an object's instance identifying information. An EBM containing details about multiple objects might have elements dedicated for 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 as well as 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 that of the top-level instance (for example - Order Instance Identification) or for that of a grandchild instance (for example - Schedule Line Item Identification).

The commonly used identification theme 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 as well as at the integration layer. Each of the representations will be uniquely identified using identifying keys. For example, Order Number can uniquely identify either an Order ID or an Order instance in Siebel application; whereas in a PeopleSoft application, the order instance will be 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 as well as the systems are quite different. The Identification Type allows you to capture the identifying keys of these representations for a single object instance. This diagram illustrates the schema definition for Identification Type.



Structure of the Identification Type element

The Identification Type allows us to capture three specific representations and one general mechanism to capture all other representations for a single object. This table describes each of the representations.

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/ContextID	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

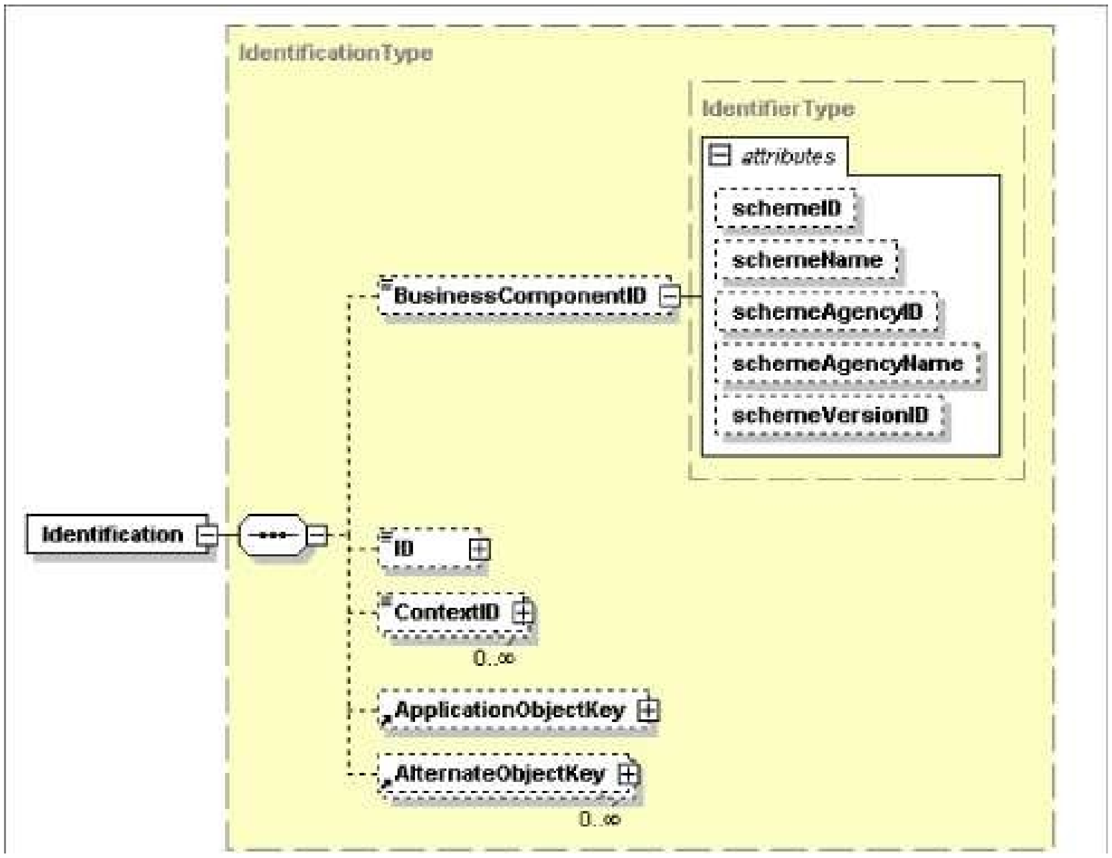
Name	Purpose	Details
		examples.
ApplicationObjectKey	Participating application specific internally generated unique identifier for this object instance	Business documents generated by Oracle AIA applications populate this information. This represents the primary key of the object at the participating 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 environment in which an identifier is valid, a set of attributes that describes the validity of the key is supported in addition to the actual key value.

This table describes the purpose of each of the attributes:

Name	Type	Description
BusinessComponentID ID ApplicationObjectKey/ID AlternateObjectKey/ID	Element	The element is used to store the actual object ID.
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 will have AIA 01 populated in this attribute. For identifiers generated by participating applications, the short code for that of the participating application will be stored in this attribute.

This diagram illustrates the Business Component ID:



Structure of the Business Component ID

Example:

```
<telcoitem:Identification>
<corecom:BusinessComponentID
mlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common/
V1" schemeID="ITEM_ID_COMMON"
schemeAgencyID="AIA_20">31303838373539343330313937393531</corecom:Bu
sinessComponentID>
<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>
```

Following is the template to populate identifications. This takes care of first doing lookupXref and if not found then doing populateXref. In the Identification component, populate the attributesschemeID/schemeAgencyID as well as ApplicationObjectKey element.

Resulting code is:

```
<corecom:BusinessComponentID
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common
/V1"
schemeID="SALESORDER_ID"
schemeAgencyID="COMMON">2d3 53 23 83 73 83 63 93 03 43 03 63 63 53
63 6</corecom:Business^>
ComponentID>
<corecom:ID
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common
/V1"
schemeID="SALESORDER_NUMBER"
schemeAgencyID="SEBL_01">OrdNum1</corecom:ID>
<corecom:ContextID
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common
/V1"
schemeID="ORGANIZATION_ID"
schemeAgencyID="SEBL_01">2d3836303130363634343033373631931</corecom:C
ontextID>
<corecom:ApplicationObjectKey
xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common
/V1">
<corecom:ID schemeID="SALESORDER_ID"
schemeAgencyID="SEBL_01">OrdId1</corecom:ID>
</corecom:ApplicationObjectKey>
```

Follow these guidelines everywhere that has corecom:Identification or any of its derivations.

1. SchemeAgencyID for Common is COMMON.
2. 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.
3. Scheme ID for BusinessComponentID is the same as XrefTableName in lookupXref/populateXref.

Example:

```
<telcosord:Identification>
<xsl:call-template name="PopulateEBMIdentification">
<xsl:with-param name="SchemeAgencyID"
select="$SenderInstanceColumnName"/>
<xsl:with-param name="SchemeID" select="'SALESORDER_ID'"/>
<xsl:with-param name="ApplicationObjectKeyID"
select="/sordabo:SiebelMessage/sordabo:ListOfOrder/sordabo:ListOfOrd
erHeader/sordabo:Order/sordabo:Id"/>
<xsl:with-param name="CreateCommonIDFlag" select="'Y'"/>
<xsl:with-param name="UserKeyID"
select="/sordabo:SiebelMessage/sordabo:ListOfOrder/sordabo:ListOfOrd
erHeader/sordabo:Order/sordabo:OrderNumber"/>
<xsl:with-param name="UserKeySchemeID"
select="'SALESORDER_NUMBER'"/>
<xsl:with-param name="ContextID"
select="/sordabo:SiebelMessage/sordabo:ListOfOrder/sordabo:ListOfOrd
erHeader/sordabo:Order/sordabo:PrimaryOrganizationId"/>
<xsl:with-param name="ContextSchemeID" select="'ORGANIZATION_ID'"/>
</xsl:call-template>
```

```
</telcosord:Identification>
1. SchemeAgencyID - Source System Instance ID. Used for LookupXref
ReferenceColumn
as well as in schemeAgencyID attributes
2. ScemeID - Used as xref table name in LookupXref and also to
populate
schemeID attributes
3. ApplicationObjectKeyID - Used for ReferenceValue in LookupXref
and
as well as populating ApplicationObjectKey/ID
4. CreateCommonIDFlag(Optional default 'N') - To indicate if the
template should populateXref
if lookupXref finds no value
5. UserKeyID(Optional) - used to populate Identification/ID
6. UserKeySchemeID(Optional) - Used to populate schemeID attribute
in
Identification/ID
7. ContextID(Optional) - used to populate Identification/ContextID
8. ContextSchemeID(Optional) - used to populate schemeID in
Identification/ContextID
<xsl:call-template name="PopulateEBMIdentification">
<xsl:with-param name="SchemeAgencyID" select=""/>
<xsl:with-param name="SchemeID" select=""/>
<xsl:with-param name="ApplicationObjectKeyID" select=""/>
<xsl:with-param name="CreateCommonIDFlag" select="'Y'"/>
<xsl:with-param name="UserKeyID" select=""/>
<xsl:with-param name="UserKeySchemeID" select=""/>
<xsl:with-param name="ContextID" select=""/>
<xsl:with-param name="ContextSchemeID" select=""/>
</xsl:call-template>
— ->
<xsl:template name="PopulateEBMIdentification">
<xsl:param name="SchemeAgencyID"/>
<xsl:param name="SchemeID"/>
<xsl:param name="ApplicationObjectKeyID"/>
<xsl:param name="CreateCommonIDFlag" select="'N'"/>
<xsl:param name="UserKeyID" select="''"/>
<xsl:param name="UserKeySchemeID" select="''"/>
<xsl:param name="ContextID" select="''"/>
<xsl:param name="ContextSchemeID" select="' ' "/>
<corecom:BusinessComponentID>
<xsl:attribute name="schemeID">
<xsl:value-of select="$SchemeID"/>
</xsl:attribute>
<xsl:attribute name="schemeAgencyID">
<xsl:text>COMMON</xsl:text>
</xsl:attribute>
<xsl:variable name="CommonId">
<xsl:value-of select="xref:lookupXRef($SchemeID,
$SchemeAgencyID, $ApplicationObjectKeyID,
'COMMON', false)"/>
</xsl:variable>
<xsl:choose>
<xsl:when test="$CreateCommonIDFlag = 'Y' and $CommonId = ''">
<xsl:value-of select="xref:populateXRefRow($SchemeID,
$SchemeAgencyID,
$ApplicationObjectKeyID,
'COMMON', orcl:generate-guid(),'ANY')"/>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="$CommonId"/>
</xsl:otherwise>
</xsl:choose>
</corecom:BusinessComponentID>
<xsl:if test="$UserKeyID != ''">
<corecom:ID>
<xsl:attribute name="schemeID">
<xsl:value-of select="$UserKeySchemeID"/>
</xsl:attribute>
<xsl:attribute name="schemeAgencyID">
<xsl:value-of select="$SchemeAgencyID"/>
</xsl:attribute>
<xsl:value-of select="$UserKeyID"/>
</corecom:ID>
</xsl:if>
<xsl:if test="$ContextID != ''">
<corecom:ContextID>
<xsl:attribute name="schemeID">
<xsl:value-of select="$ContextSchemeID"/>
</xsl:attribute>
<xsl:attribute name="schemeAgencyID">
<xsl:value-of select="$SchemeAgencyID"/>
</xsl:attribute>
<xsl:variable name="CommonId">
<xsl:value-of select="xref:lookupXRef($ContextSchemeID,
$SchemeAgencyID, $ContextID,
'COMMON', false)"/>
</xsl:variable>
<xsl:choose>
<xsl:when test="$CommonId = ''">
<xsl:value-of select="xref:populateXRefRow($ContextSchemeID,
$SchemeAgencyID, $ContextID,
'COMMON', orcl:generate-guid(),'ANY')"/>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="$CommonId"/>
</xsl:otherwise>
</xsl:choose>
</corecom:ContextID>
</xsl:if>
<corecom:ApplicationObjectKey>
<corecom:ID>
<xsl:attribute name="schemeID">
<xsl:value-of select="$SchemeID"/>
</xsl:attribute>
<xsl:attribute name="schemeAgencyID">
<xsl:value-of select="$SchemeAgencyID"/>
</xsl:attribute>
<xsl:value-of select="$ApplicationObjectKeyID"/>
</corecom:ID>
</corecom:ApplicationObjectKey>
</xsl:template>
```

Cross-References for Multi-Part Keys

The Cross-References API does not support multi-part keys. In such cases where the application does not have a single unique key, we recommend key concatenation using this notation:

```
{Key1}::{Key2}::Key3
```

Hierarchical Cross References for Child Objects

Oracle AIA does not support hierarchical cross-references. Child object common keys are unique on their own without being under the parent's context.

This means that child objects will have their own entry within the cross-references table with a GUID used for the common ID. For application instance IDs, if the child key is not global, then it should be concatenated with its parent key to form a global multi-part key.

Transactional Considerations

The cross-reference XPath functions that add, modify, or delete data are transactional in nature and follow the transactional behavior of the data source used by that cross-reference.

Usually these cross-reference functions work in the context of the current transaction, meaning that it should join the existing transaction semantics. To ensure this behavior, the data source in data-source.xml used by the cross-references must be configured as:

- The managed-data-source element must have its tx-level attribute set to global.
- Or, the managed-data-source element must not have the tx-level attribute populated (default is global).

Example:

```
<managed-data-source connection-pool-name="soademo_ppol" jndi-
name="jdbc/AIADatasource" name="AIADatasource" tx-level="global" />
```

or

```
<managed-data-source connection-pool-name="soademo_ppol" jndi-
name="jdbc/AIADatasource" name="AIADatasource" />
```

When using a BPEL process with synchronous sub-processes, if you want the cross-references in the main process to be visible to the sub-processes, you have to make sure that the sub-processes are executed under the same transaction context (join the same transaction). This way the cross-references in the parent process as well as sub-processes will all be either committed or rolled back together under the same transaction.

This can be configured by setting transaction property to participate on the partner link of the sub-process to indicate that it should participate in the existing transaction.

Example:

```
"transaction=participate"
```

BPEL Scope and Transaction Scope

Cross-references created in different scopes do not imply that they are isolated under different transactions for each scope. You need to manually manage cross-reference cleanup within fault blocks. BPEL will not automatically roll back or clean up cross-references when a fault is thrown.

Utility XSL Template for XREF Lookup-or-Populate Logic

Most requestor ABC services will need to use this 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:
    a. Generate a new ID.
    b. Populate the Xref table with the new ID.
3. Use the found or new common ID in the transformation.

There may be several different places in the transformation to perform this logic. Put the following template into your transformation.

```
<!-- -------------------------------------------------------------

lookupOrPopulateXRef template
1. first attempt xref:lookupXRef()
2. if no value is found, then generate a value and
xref:populateXRefRow()
3. returns the xref value.
------------------------------------------------------------------ →
<xsl:template name="lookupOrPopulateXRef">
<xsl:param name="xrefTableName"/>
<xsl:param name="xrefReferenceColumnName"/>
<xsl:param name="xrefReferenceValue"/>
<xsl:param name ="xrefColumnName"/>
<xsl:variable name="xrefValue"
elect="xref:lookupXRef($xrefTableName,$xref^>
ReferenceColumnName,$xrefReferenceValue,$xrefColumnName,false())"/>
<xsl:choose>
<xsl:when test="$xrefValue!=''">
<xsl:value-of select="$xrefValue"/>
</xsl:when> <xsl:otherwise>
<xsl:value-of
select="xref:populateXRefRow($xrefTableName,$xrefReferenceColumnName
,$xrefReferenceValue,$xrefColumnName,orcl:generate-guid (),'ANY')"/>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
```

Here is an example for use in transformations:

```
<xsl:variable name="commonAccountId">
<xsl:call-template name="lookupOrPopulateXRef">
<xsl:with-param name="xrefTableName"
select="'CUSTOMERPARTY_ACCOUNTID'"/>
<xsl:with-param name="xrefReferenceColumnName" select="'SEBL_01'"/>
```

```
<xsl:with-param name="xrefReferenceValue"
select="$siebelAccountId"/> <xsl:with-param name="xrefColumnName"
select="'COMMON'"/>
</xsl:call- template>
</xsl:variable>

<corecom:AccountIdentification>
<corecom:BusinessComponentID>
<xsl:value-of select="$commonAccountId"/>
</corecom:BusinessComponentID>
</corecom:AccountIdentification>
```

DVMs

This section discusses:

- Populating domain values
- Domain values lookups
- When to use DVMs

For more information, see the *Oracle Enterprise Service Bus Developer's Guide* available on oracle.com.

Populating Domain Values

DVMs are stored in the Enterprise Service Bus (ESB) database and are maintained using the ESB Console user interface.

First, you create a new DVM and then add rows and columns to store the data values.

Domain Values Lookups

Domain values are to be used only for static lookups. They must not be used 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.

When to Use DVMs

DVMs can be categorized into three groups:

- DVMs where 100% 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.

- DVMs where all or a few samples are seeded, based on the seeded values found in the participating applications.

  You can add more to the participating apps and can add more to the DVMs in the middle to accommodate.

  Typically, logic is not added to these values; there is a match done to get the value and pass it on to the receiving application. You must enhance the list to include all possible values that you have defined in your participating applications.

- Custom DVMs.

  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 impacted during an upgrade.

Oracle AIA Configuration File

While developing EBS services as well as ABC services and flows, you often need to parameterize certain values to allow customers to configure the functionality without the need to modify code. For this reason, you can externalize name-value pairs at the system, module, Enterprise Business Object (EBO), and service levels in a configuration file that can be modified by customers when needed.

Configuration Schema

The integration architecture configuration (AIAConfiguration) schema is comprised of two main components:

- AIASystemConfiguration:

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

- EBOConfiguration:

  Contains configuration name-value pairs for services belonging to a certain EBO.

Reading Configuration Name-Value Pairs

These XPath functions are provided to access the configuration name-value pairs at the different levels:

```
Namespace aiacfg =
http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.config
.ConfigurationXPathFunctions
aiacfg:getSystemProperty( propertyName as string,
needAnException as boolean,
) return propertyValue as string
aiacfg:getSystemModuleProperty( moduleName as string,
propertyName as string,
needAnException as boolean
) return propertyValue as string
aiacfg:getServiceProperty( EBOName as string,
serviceName as string,
propertyName as string,
needAnException as boolean
) return propertyValue as string
```

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

In all three functions, if a matching property is not found, the result will depend upon the value of the needAnException argument. If needAnException is true, then a PropertyNotFoundException is thrown, otherwise an empty string is returned.

Adding a new property to the production environment will need refreshing of Oracle AIA Config cache by doing these using the Business Service Repository (BSR) interface:

1. Log on to http://[SOA host]:[Port]/AIA/
2. Go to ‘Setup’ and then to the ‘Configuration’ tab
3. Click Reload. The changes made to the Oracle AIA Configuration file are reloaded.

Sample AIAConfigurationProperties.xml

The sample AIAConfiguration.xml code:

```
<?xml version="1.0" encoding="UTF-8"?>
<AIAConfiguration xmlns="http://xmlns.oracle.com/aia/core/config">
<!-- System-wide configuration properties -->
<SystemConfiguration>
<Property name="isProductionTransaction">TRUE</Property>
<Property name="logLevel">3</Property>
<Property name="debugMode">FALSE</Property>
<ModuleConfiguration moduleName="ErrorHandler">
<Property name="logLevel">2</Property>
<Property name="logDirectory">temp</Property>
</ModuleConfiguration>
</SystemConfiguration>
<!-- SalesOrder related service configurations  -->
<ServiceConfiguration serviceName="ProcessOrderSiebelRequestorABCS">
<Property
name="TransformationFile">ProcessOrderSiebelToEBM</Property>
</ServiceConfiguration>
<ServiceConfiguration serviceName="OrderEBS">
<Property name="String">String</Property>
</ServiceConfiguration>
<!--  Product related service configurations  -->
<ServiceConfiguration serviceName="SyncProductRequestorPortalABCS">
<Property
name="TransformationFile">SyncProductPortalToEBM</Property>
```

```
<Property name="String">String</Property>
</ServiceConfiguration>
<ServiceConfiguration serviceName="ProductEBS">
<Property name="String">String</Property>
<Property name="String">String</Property>
</ServiceConfiguration>
<ServiceConfiguration serviceName="SyncProductConsumerSiebelABCS">
<Property name="String">String</Property>
<Property name="String">String</Property>
</ServiceConfiguration>
</AIAConfiguration>
```

Chapter 7: Developing Custom XPath Functions

This chapter provides an overview of custom XPath functions and discusses how to:

- Implement the custom XPath function.
- Deploy the XPath/XSL function in JDeveloper.
- Deploy the XPath/XSL function in the application server.

Understanding Custom XPath Functions

To create and use an XPath function:

1. Implement your function in Java as a Java XPath extension class.
2. Deploy your custom XPath function in JDeveloper.
3. Deploy your custom XPath function in the application server.

XPath Function Implementation Java 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 will be 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:

  com.oracle.bpel.xml.XPath.IXPathFunction

  This interface has the method:

  Object call(IXPathContext context, List args) throws XPathFunctionException that gets called by BPEL when the XPath function is invoked. When the function is called, the runtime engine will pass the XPathContext and list of arguments passed to the function. IXPathContext allows you to get access to the BPEL variable values within the execution context. The implementation of this method should naturally delegate to the public static method representing the function.

Here is a custom function implementation to get the current date. The implementation takes care of supporting both XSL as well as XPath. The function also has an optional parameter to show how overloaded methods are implemented:

```
package oracle.apps.aia.core;•
import java.text.SimpleDateFormat;
import java.util.Date; import
java.util.List;
import com.oracle.bpel.xml.XPath.IXPathContext;
import com.oracle.bpel.xml.XPath.IXPathFunction;
import com.oracle.bpel.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("Need to pass one argument.");
}
}
public static String getCurrentDate(String formatMask)
{
SimpleDateFormat df = new SimpleDateFormat(formatMask)
String s = df. format (new Date());
return s;
}
. . .
}
```

Implementing the Custom XPath Function

This section discusses:

- Naming standards
- Supported data types

Naming Standards

These naming standards are enforced:

- Function name
- 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 exactly the same as the function name. The parameters data types should match the function parameter data types. If there are optional parameters in the function, you should a different overloaded method to take care of 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 if you want 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 as well as 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:

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 as well as in the XSL Mapper. To do that, developers have to provide a User Defined Extension Functions config file and register it with JDeveloper through FilePreferencesXSL Map.

This config file looks like:

```
<extension-functions xmlns:pcui="http://www.oracle.com/PC/ui">
<functions
xmlns:utl="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.
core.DateUtils">
<function name="utl:getCurrentDate" as="string">
<param name="formatMask" as="string"/>
</function>
<function ...>
. . .
</function>
</functions>
<functions xmlns:ora="http://schemas.oracle.com/XPath/extension">
<function ...>
. . .
</function>
</functions>
</extension-functions>
```

The implementation classes must be zipped/jared and dropped at: [jdev_home]\jdev\lib\patches. This way, these classes will be placed in the classpath of JDeveloper when the embedded OC4J is used.

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 XPath-functions.xml file, which is found at: SOA_Oracle_Home\bpel\system\config. 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 these attributes:
  - id: Defines the XPath function name
  - arity: Defines the number of required arguments that the function can accept (for example, a function takes five arguments, but the last two arguments are optional; in this case, the value for arity is three). It is recommended that you do not use functions with optional parameters.
- <classname>: Defines the XPath implementation class name. In this case, it is the inner class name.
- <comment>: Used to define documentation.
- Specify these initialization properties for each function:
  - namespace-uri — Associate each XPath with a namespace. (See the Naming Standards section).
  - namespace-prefix — Specify a namespace prefix for this XPath function. (See the Naming Standards section).

```
<function id="getCurrentDate" arity="0">
<classname>
oracle.apps.aia.core.DateUtils$GetCurrentDateFunction</classname>
<comment>Gets the current date with optional format mask.</comment>
<property id="namespace-uri">
<value>http://www.oracle.com/XSL/Transform/java/
oracle.apps.aia.core.DateUtils</value>
</property>
<property id="namespace-prefix">
<value>utl</value>
</property>
</function>
```

The implementation classes must be dropped at: [soa_home]\bpel\system\classes. This way, these classes will be placed in the classpath of the J2EE container and available to the classloader.

Chapter 8: Designing and Constructing JMS Adapter Services

This chapter provides an overview of JMS Adapter and discusses how to configure JMS Adapter.

Understanding JMS Producer Adapter Service and JMS Consumer Adapter Service

For fire-and-forget and asynchronous request/response interaction patterns, queues are leveraged to ensure guaranteed delivery. Even though the default behavior is to make the usage of queues transparent to the requester application, the requester application can decide to push the messages directly into the queue.

Oracle Advanced Queuing (AQ) is used as the JMS provider and is used as the persistence layer for the messages. Although the default approach is to leverage the BPEL dehydration data store for creating the queues, it is not mandatory. You can decide to create the queues in other data stores; and the client applications -- BPEL and Enterprise Service Bus (ESB) services -- can be configured to point to the new data stores. Similarly, you also have the option to use a third-party product as the JMS provider. In some situations, participating applications, such as BRM, can leverage their own data stores to create the queues.

JMS Producer Adapter is used to consume the SOAP messages and enqueue them to the JMS provider. This producer adapter can either be implemented as BPEL or ESB, however the preferred approach is ESB. Ensure that the adapter refers to JNDI name to make the location transparent.

JMS Consumer Adapter is used to dequeue the message and invoke the application business connector (ABC) service. This adapter has to be implemented as ESB service.

In situations where the participating applications decide to push the messages directly into the queue, only the JMS consumer adapter needs to be built.

The architecture mandates that the ABC services have only ESB/SOAP based entry points. Hence, the non-SOAP adapters have to sit between the participating applications and the ABC services.

Configuring JMS Adapter

To configure the JMS Adapter using AQ for persistence:

1. Configure the database.

   This includes creating the user, assigning privileges, creating the requiring queue tables for each entity to be handled, creating queues, and starting the queues.

2. Configure the application.xml and oc4j-ra.xml.

3. **Configure the JMS adapter for the producer.**

4. **Configure the JMS adapter for the consumer.**

5. **Configure Connection Factory in the Application Server.**

Configuring the Database

To configure the database:

1. **Create user and assign privileges.**

   Run this sql command:

```
DROP USER jmsuser CASCADE ;
GRANT connect,resource,AQ_ADMINISTRATOR_ROLE TO jmsuser
IDENTIFIED BY jmsuser ;
GRANT execute ON sys.dbms_aqadm TO jmsuser;
GRANT execute ON sys.dbms_aq TO jmsuser;
GRANT execute ON sys.dbms_aqin TO jmsuser;
GRANT execute ON sys.dbms_aqjms  TO  jmsuser;

connect jmsuser/jmsuser;
```

2. **Create the table that handles the OEMS JMS destination (queue).**

   Queues use a queue table. This SQL example creates a single table, AIA SALESORDERJMSQTAB, for a queue. The multiple_consumers parameter specifies whether there are multiple consumers. Set multiple_consumers to false for a queue.

```
Begin
DBMS_AQADM.CREATE_QUEUE_TABLE(Queue_table
'AIA_SALESORDERJMSQTAB',Queue_payload_type 'SYS.AQ$_JMS_MESSAGE',
sort_list 'PRIORITY,ENQ_TIME', multiple_consumers = >false,
compatible '8.1.5');•
End;
```

3. **Create the JMS destination.**

   This SQL example creates a queue called AIA SALESORDERJMSQUEUE within the queue table AIA SALESORDERJMSQTAB and then starts the queue.

```
Begin
DBMS_AQADM.CREATE_QUEUE(Queue_name'AIA_SALESORDERJMSQUEUE',
Queue_table'
SALESORDERJMSQTAB');
End;
```

4. **Start Queue.**

   EXEC DBMS_AQADM.START_QUEUE(queue_name'AIA_SALESORDERJMSQUEUE');

Configuring the XML Files

To configure the application.xml and oc4j-ra.xml files:

1. Add resource provider information to %ORACLE_HOME%/j2ee/home/config/application.xml.

```
<resource-provider class="oracle.jms.OjmsContext" name="DBConnect">
<description>AIA JMS reference</description>
<property name="url"
value="jdbc:oracle:thin:@localhost:1521:archorcl" />
<property name="username" value="jmsuser" />
<property name="password" value="jmsuser" />
</resource-provider>
```

2. Add connection factory information to oc4j-ra.xml in <SOA Home>\j2ee\home\application-deployments\default\JmsAdapter

```
<connector-factory location="eis/Jms/DBConnect" connector-name="Jms
Adapter">
<config-property name="connectionFactoryLocation" value="
java:comp/resource/DBConnect/QueueConnectionFactories/AIA_Queue" />
<config-property name="factoryProperties" value="" />
<config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE" />
<config-property name="isTopic" value="false" />
<config-property name="isTransacted" value="true" />
<config-property name="username" value="jmsuser" />
<config-property name="password" value="jmsuser" />
</connector-factory>
```

Configuring the JMS Adapter for the Producer

To configure the JMS Adapter for the producer, follow the steps in the Adapter Configuration Wizard:

1. Enter a Service Name and description.
2. On the JMS Provider page, select the JMS provider.

   Select Database for the Oracle Enterprise Messaging Service (OEMS).

3. On the Service Connection page, select a database connection already defined in your project.

   Specify the JNDI name for the database.

4. On the Operation page, select Produce Message as the Operation Type and enter an operation name.
5. On the Produce Operation Parameters page, browse and select the queue that you created in the queue table.
6. On the Messages page, browse and select the schema file location, and select the schema element that defines the message.

Configuring the JMS Adapter for the Consumer

To configure the JMS adapter for the consumer:

1. Enter a Service Name and description.
2. On the JMS Provider page, select the JMS provider.

   Select Database for the OEMS.

3. On the Service Connection page, select a database connection already defined in your project.

   Specify the JNDI name for the database.

4. On the Operation page, select Consume Message as the Operation Type and enter an operation name.
5. On the Consume Operation Parameters page, browse and select the queue that you created in the queue table.
6. On the Messages page, browse and select the schema file location and select the schema element that defines the message.

   A Schema File is required if you would like to validate the message content. For example, if you would like to consume the message only if it is confined to any particular XSD, then you need to choose a schema and the message type/schema element. The Schema should be available in your local folders with in the project folder. If validation is not required, choose the Native format translation.

Configuring Connection Factory in the Application Server

To configure Connection Factory in the Application Server:

1. Open the Application Server Control.
2. Go to Oc4j home/applications.
3. On the Applications tab, select Standalone Resource Adapters from the available options.
4. Select the OracleASjms adapter.
5. Select Connection Factories and click Create.
6. On the Create Connection Factory: Select Interface page, select javas.jms.QueueConnectionFactory as the Connection Factory Interface.
7. In the JNDI Location field, enter eis/Jms/DBConnect. Select No Connection Pool and click Finish.

How Can an Application Publish JMS Messages Directly to the JMS Queue Configured on AIA?

You can publish the JMS messages directly from the participating application to a JMS queue that is available on the AIA instance. To achieve this, you need to configure the resource adapter on the application server and the client setup.

For more information about connecting to AIA from your participating application, see the *Connect to AIA from your Enterprise via JMS* white paper, available on My Oracle Support.

Chapter 9: Working with EBM Headers

This chapter provides an overview of enterprise business message (EBM) header components and discusses:

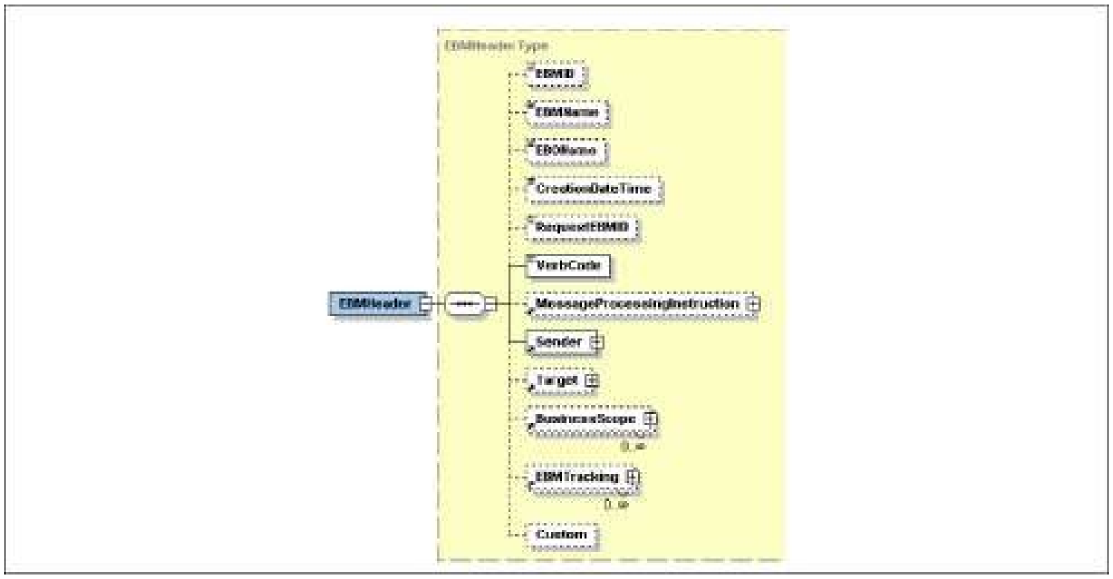
- Standard elements.
- Sender.
- Target.
- BusinessScope.
- EBMTracking.
- Custom.
- Populating EBM header components.

Understanding Header Components

Every EBM that is processed by application business connector (ABC) services and enterprise business services (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 for these tasks:

- Tracking important information
- Auditing
- Indicating source and target systems
- Error handling and tracing

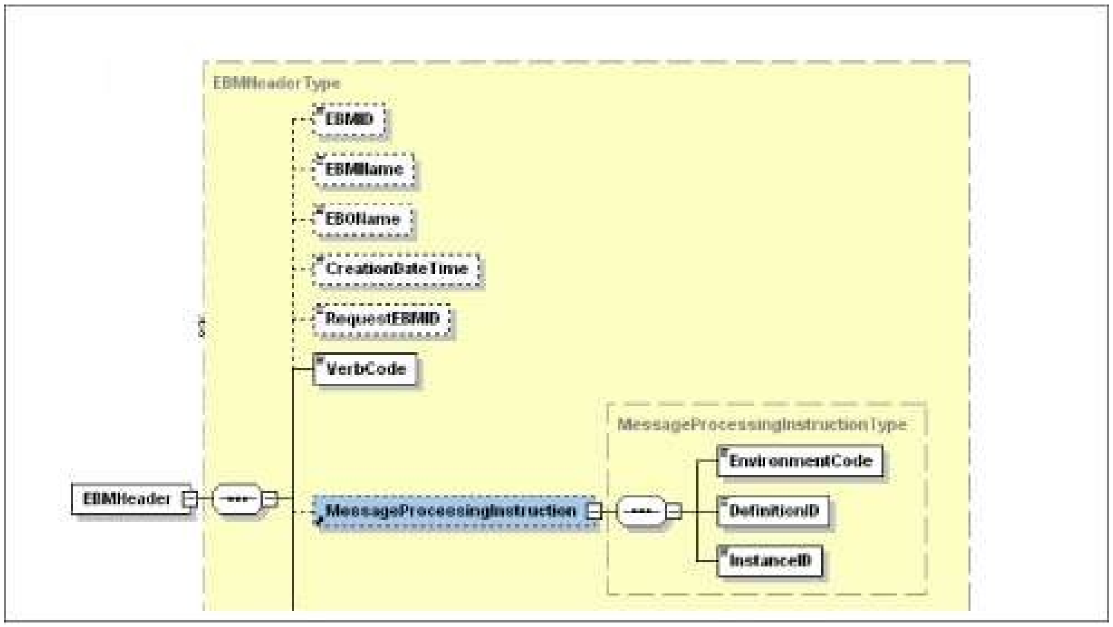
The EBM header consists of these components:



EBM header components

Standard Elements

This section discusses the standard elements in a message header.



Standard elements in a message header

EBMID

This element contains a GUID to uniquely identify the EBM. The GUID is generated using a standard XPath function provided.

Example:

```
<EBMHeader>
<EBMID>2134 8 6yio534 8oidsus4 32 54 5</EBMID>
. . .
</EBMHeader>
```

EBOName

This element contains the enterprise business object's (EBO's) fully qualified name in the notation: {namespaceURI}localpart.

Example:

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

RequestEBMID

This element contains the originating request GUID that uniquely identifies the originating request ID. The EBS populates this field in the response message by extracting it from the request message.

Example:

```
<EBMHeader>
<RequestEBMID>213 4 8 6yio53 4 8oidsus43 2 54 5</RequestEBMID>
. . .
</EBMHeader>
```

CreationDateTime

This element contains a timestamp that reflects when the EBM was created. 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)?
```

Example:

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

VerbCode

Contains the verb represented by the EBM.

Example:

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

```
</EBMHeader>
```

MessageProcessingInstruction

Contains instructions on how the message should be processed. This section is used to indicate 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/simulation path.

MessageProcessingInstruction contains three fields:

- EnvironmentCode:

  Can be set to either CAVS or PRODUCTION. The default is PRODUCTION. Setting the value to PRODUCTION indicates that the request needs to be sent to a concrete endpoint. Setting the value to CAVS indicates that the request needs to be sent to CAVS Simulator.

- DefinitionID:

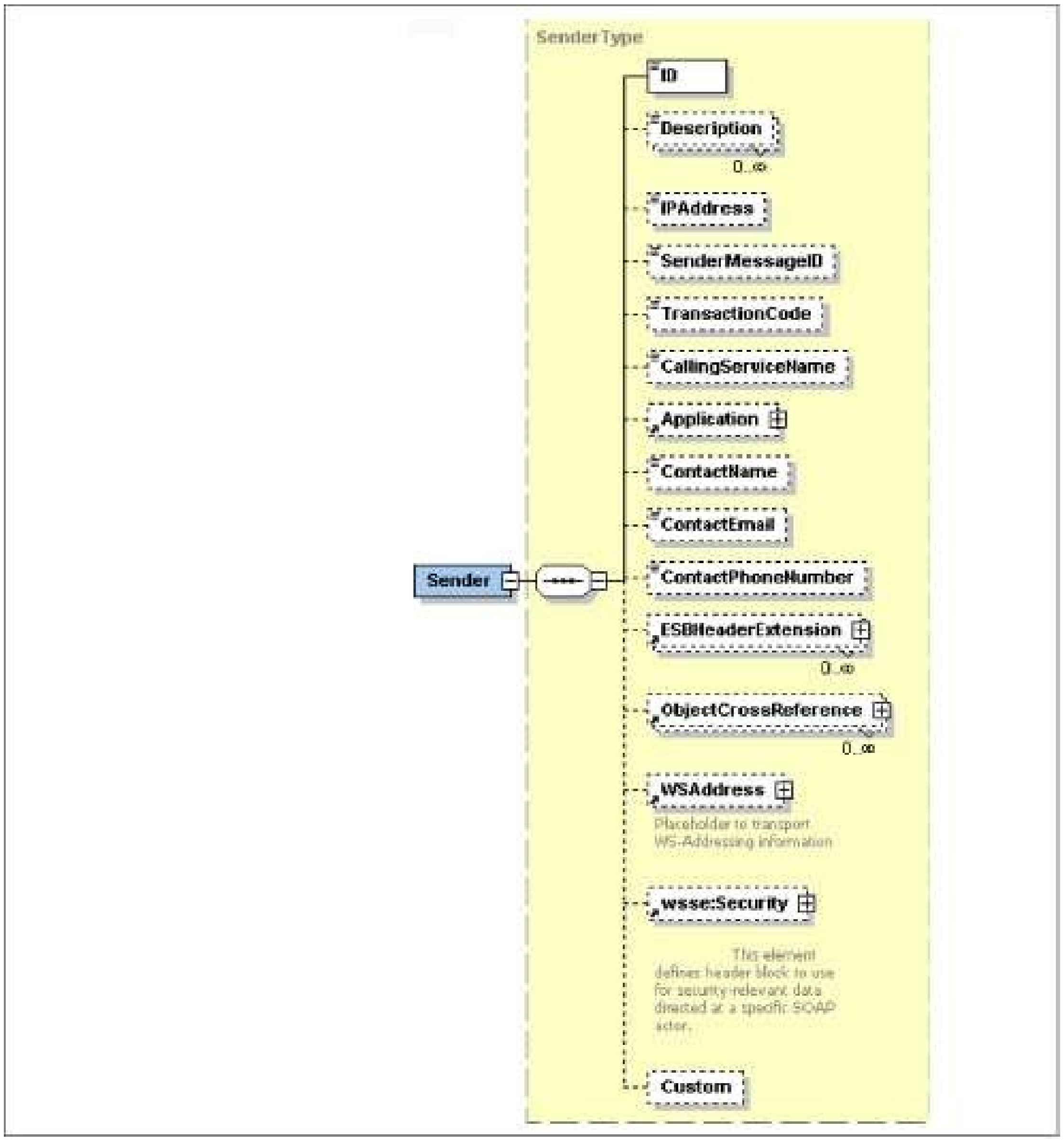
  Used to specify the test definition ID.

  It is the value of the property "Routing.[PartnerlinkName].CAVS.EndpointURI" in AIAConfigurationProperties.xml.

  It should be populated when the "Routing.[PartnerlinkName].RouteToCAVS" property is set to “true” in AIAConfigurationProperties.xml.

Sender

The Sender contains information about the originating application system.

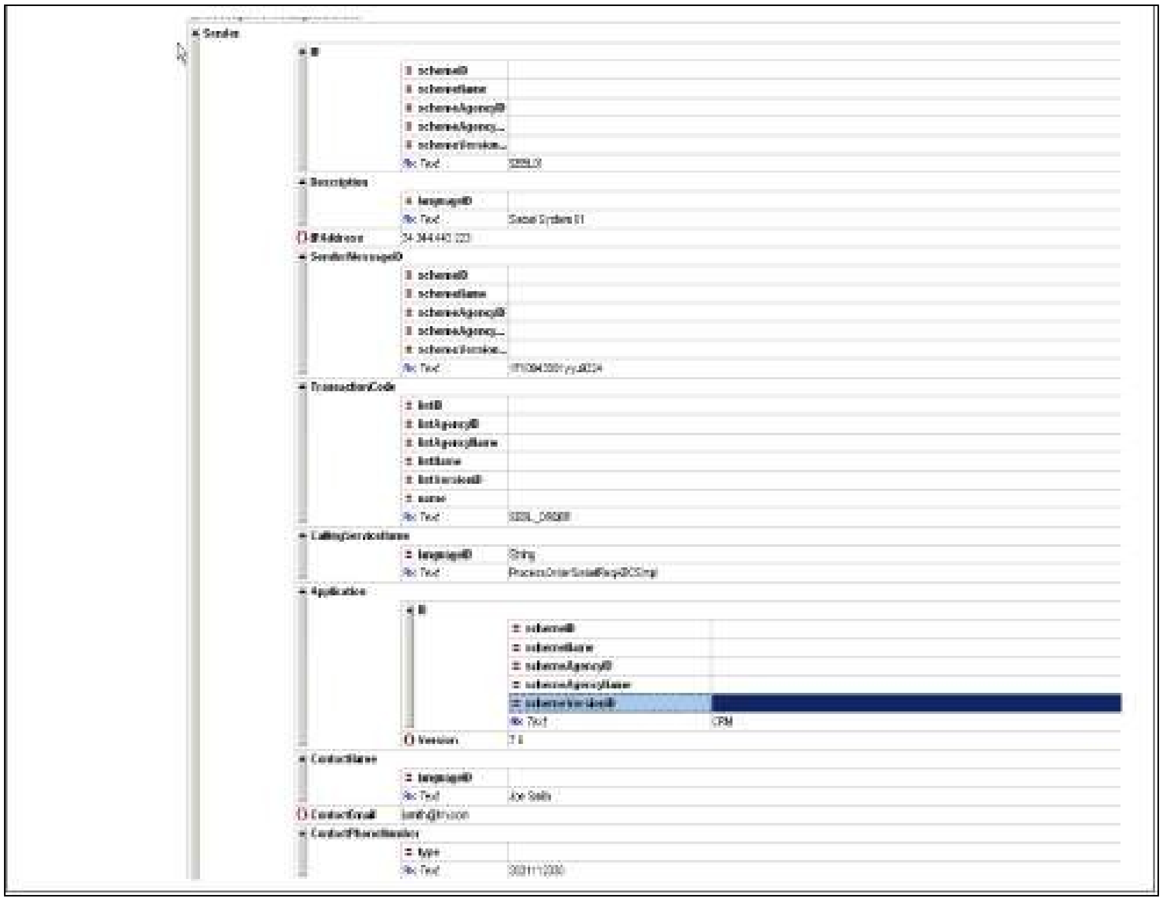


Structure of the Sender element

Element	Description
ID	Contains the sender system identifier code as seeded in the Business Service Repository (BSR). This is a mandatory element. This element represents the unique identifier of each source system. This value is stored in the BSR. ReqABCSImpl should call the API to populate this value.
Description	This is the long description of the Sender System. This value is stored in the BSR. ReqABCSImpl should call the API to populate this value.
IPAddress	Represents the IP address of the sender system. This value is stored in the BSR. ReqABCSImpl can call the API to populate this value.
SenderMessageID	This is the ID that can uniquely identify the message sent by the sender

Element	Description
	system. ReqABCSImpl may have this information and that information can be used to populate this element.
Transaction Code	This is the task code in the sender system that is used to generate this message. ReqABCSImpl has this information and should use that while preparing the EBM.
CallingServiceName	Name for the calling ABC service. Will be populated by source ABC service.
Application	Holds information about the originating application.
Application/ID	The sender system can designate the originating application code in this element. This will be stored in the BSR. ReqABCSImpl should call the API to populate this value.
Application/Version	The sender system can designate the version of the originating application in this element. This will be stored in the BSR. ReqABCSImpl should call the API to populate this value.

This diagram illustrates a sample Sender element:



```
Sender
    ID
        schemeID
        schemeName
        schemeAgencyID
        schemeAgency...
        schemeVersion...
        Text                 SEBL01
    Description
        languageID
        Text                 Siebel System 01
    IPAddress            34-344-443-223
    SenderMessageID
        schemeID
        schemeName
        schemeAgencyID
        schemeAgency...
        schemeVersion...
        Text                 [illegible]
    TransactionCode
        listID
        listAgencyID
        listAgencyName
        listName
        listVersionID
        name
        Text                 SEBL_ORDER
    CallingServiceName
        languageID           String
        Text                 ProcessOrderSiebelReqABCSImpl
    Application
        ID
            schemeID
            schemeName
            schemeAgencyID
            schemeAgencyName
            schemeVersionID
            Text                 CRM
        Version              7.8
    ContactName
        languageID
        Text                 Joe Smith
    ContactEmail         jsmith@[illegible]
    ContactPhoneNumber
        type
        Text                 [illegible]
```

Structure of the Sender element

SenderMessageID

This uniquely identifies the message that was 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.

TransactionCode

Used to identify the operation in the sender system that generated the sender message.

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

Name of the contact of the sender system. The value will be retrieved from BSR by doing a lookup. ReqABCSImpl should call the API to populate this value. This information could be made available in the fault message or in the request message to be sent to external businesses by the downstream services.

ContactEmail

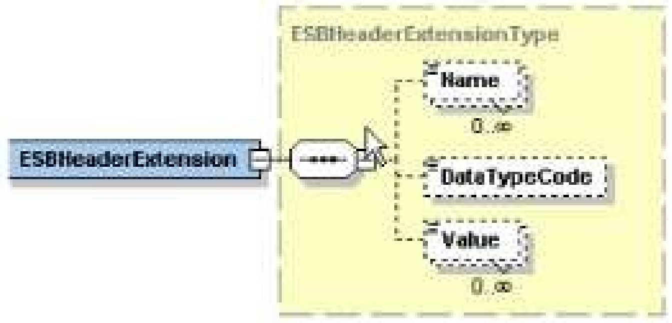
Email ID of the contact of the sender system. The value will be retrieved from BSR by doing a lookup. ReqABCSImpl should call the API to populate this value. This information could be made available in the fault message or in the request message to be sent to external businesses by the downstream services.

ContactPhoneNumber

Phone number of the contact of the sender system. The value will be retrieved from BSR by doing a lookup. ReqABCSImpl should call the API to populate this value. This information could be made available in the fault message or in the request message to be sent to external businesses by the downstream services.

ESBHeaderExtension

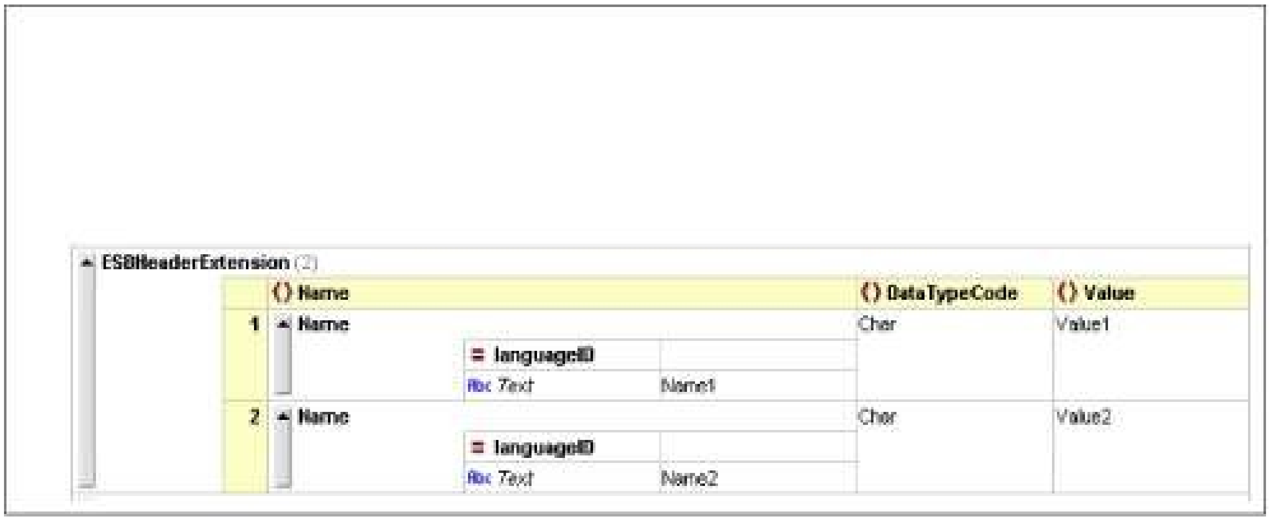
This element provides an additional facility to accommodate the transmission of additional information from source application. Some data passed from sender system must be transported through the EBM message life cycle and needed for identifying and processing the target system. The Target application 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 will be used to hold that information. ESB has name/value pair and this component can have as many such elements as possible.



Structure of 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 ABC service.
ESBHeaderExtension/Value	ESB Header element value is populated by source ABC service. 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.

This diagram illustrates a sample ESBHeaderExtension:



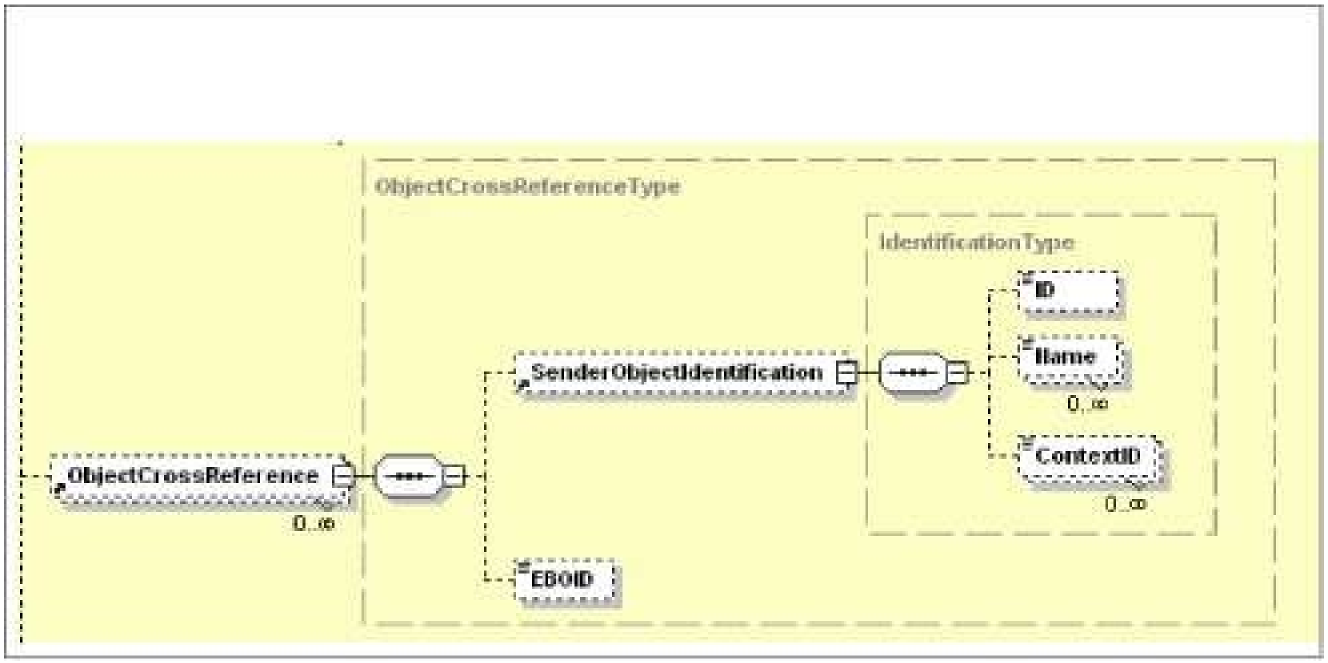
ESBHeaderExtension (2)

	Name		DataTypeCode	Value
1	Name		Char	Value1
	languageID			
	Text	Name1		
2	Name		Char	Value2
	languageID			
	Text	Name2		

Structure of the ESBHeaderExtension element

ObjectCrossReference

This component will store the identifier information of the sender objects and the corresponding cross-reference identifier. Since the EBM can be used for bulk processing this element can be repeated as well. This data may be repeated in data area as well but to maintain uniform XPath location information about them, they are maintained here as well. ReqABCSImpl is supposed to populate this value.



Structure of the ObjectCrossReference element

Element	Description
ObjectCrossReference/SenderObjectIdentification	Contains all the key field name and key field values for the object. The data is provided by the sender system. ReqABCSImpl will have this information and populate this value.
ObjectCrossReference/SenderObjectIdentification/ID	Identifies the object type of the sender system. For example, ORDER ReqABCSImpl will have this information and populate this value
ObjectCrossReference/SenderObjectIdentification/Name	This will identify the description of the sender system, for example, Purchase Order. ReqABCSImpl will have this information and populate this value.
ObjectCrossReference/SenderObjectIdentification /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 the Key value should be stored in the element itself. ReqABCSImpl will have this information and populate this value.
ObjectCrossReference/EBOID	This is the corresponding cross-reference identifier stored in the integration layer. ReqABCSImpl will have this information and populate this value.

WS Address

This element holds all WS-Addressing elements. This will be used to transport WS-Addressing elements and exchange WS-Addressing elements between requester and target system. A local version of WS-Address schema is stored in a specified directory. ReqABCSImpl populates this. This element needs to be populated in the request EBM only when the provider application will send the response in an asynchronous mode. Provider application sending an asynchronous response will leverage this element's value to identify the callback address.

For more information about specific usage of this element, see Implementing Asynchronous Message Exchange Patterns with One-Way Calls in EBS.

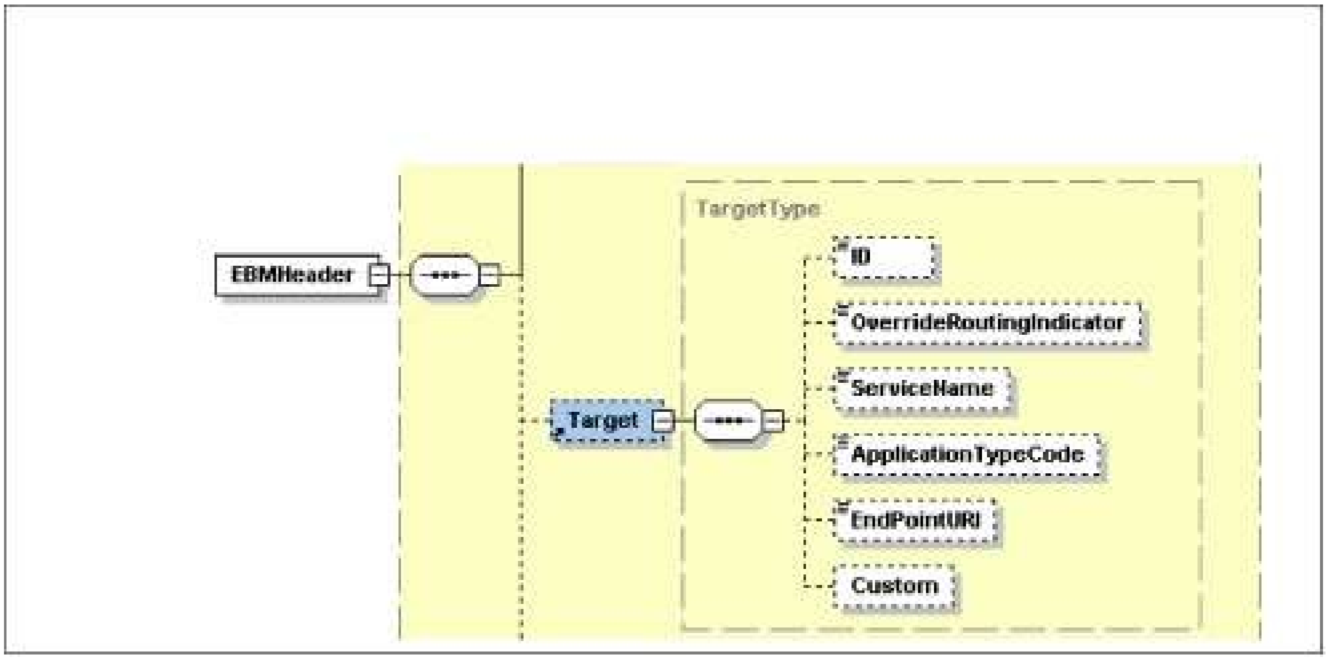
Custom

This is the complex type that customers can extend to add their own elements.

Target

The Target section is populated only when there is a need to override the routing rules for the target system defined in ESB. For bulk processing it is assumed that all objects reach the same target system defined in a routing rule. In that scenario you must define the appropriate target system information in this element if you need to override the routing rule. The overriding target system information is applicable to all the objects in the data area. It has to be noted that the requestor ABCS should never populate the target system. Enterprise Business Flow or an Enterprise Business Service alone can populate the details.

The Target element contains information regarding the target/receiving system and has these elements:



Structure of the Target element

ID

Used to identify target systems to route to when the routing rules are overridden. Populated by EBS when the target system must be overridden. The value of the participating application instance code as registered in the BSR. Other target system information can be looked up from BSR using the ID as a key.

ApplicationTypeCode

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 more than one version is supported on the system.

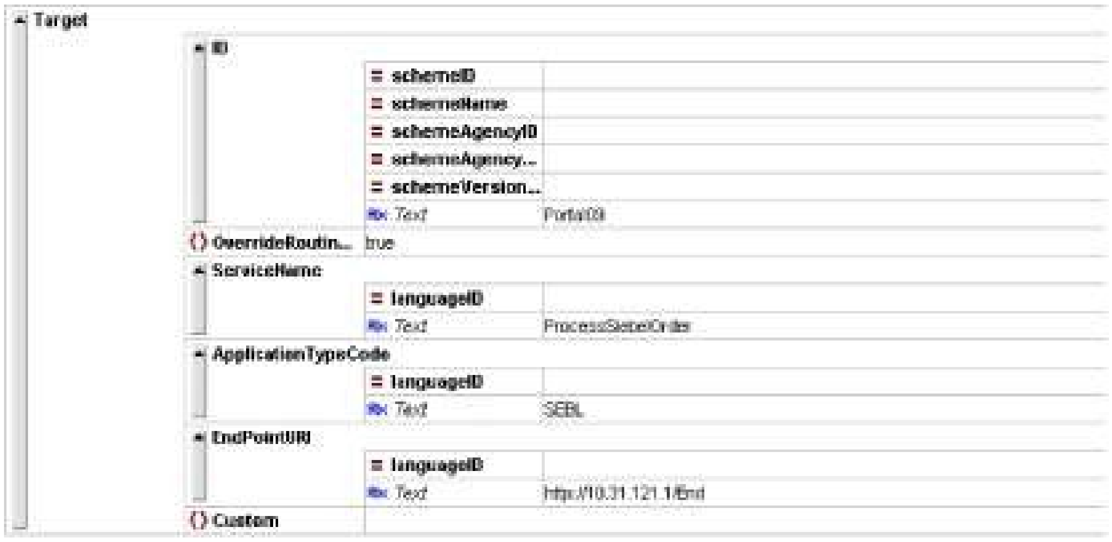
This field should be populated at the same time as the Target/ID field is populated in the EBS (or in an EBF), usually in the ABC service. The value of this field should come from the function aia:getSystemType(<ID>), where ID is the system ID value that is populated in Target/ID. Enterprise business service 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 customers can extend to add their own elements when needed.

Example

This diagram illustrates a sample Target element:



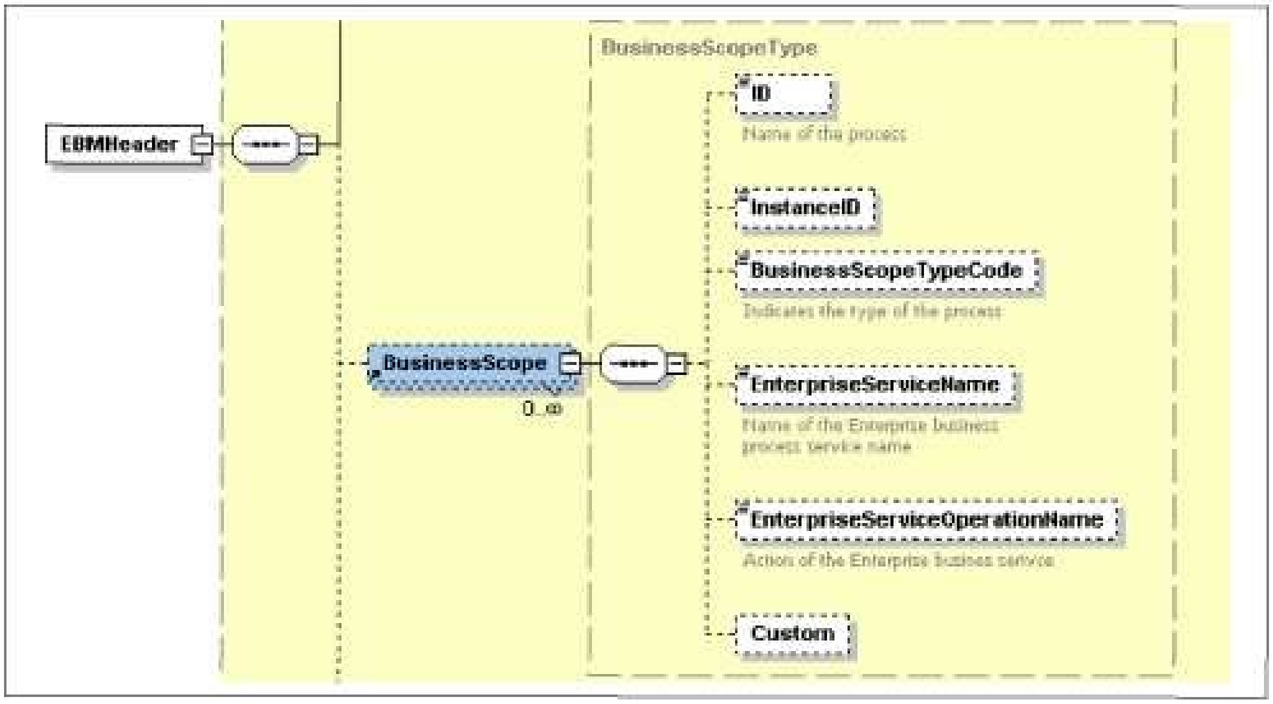
```
Target
            ID
                        schemeID
                        schemeName
                        schemeAgencyID
                        schemeAgency...
                        schemeVersion...
                        Text               Portal09
            OverrideRoutin...  true
            ServiceName
                        languageID
                        Text               ProcessSiebelOrder
            ApplicationTypeCode
                        languageID
                        Text               SEBL
            EndPointURI
                        languageID
                        Text               http://10.31.121.1/End
            Custom
```

Structure of the Target element

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 will have only 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.

This diagram describes how it works:


EBMHeader
BusinessScope
0..∞
BusinessScopeType
ID
Name of the process
InstanceID
BusinessScopeTypeCode
Indicates the type of the process
EnterpriseServiceName
Name of the Enterprise business process service name
EnterpriseServiceOperationName
Action of the Enterprise business service
Custom

Structure of the BusinessScope element

ID

An optional identifier that identifies the contract this instance relates to. ReqABCSImpl populates this value. 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.

InstanceID

A unique identifier that references the instance of the scope (for example, process execution instance of document instance). This is the name of the process or message given by the applications. The name used here will be used in the BSR error notification table process name.

BusinessScopeTypeCode

Indicates the kind of business scope. Values are BusinessScope (UMM), BusinessService (for ebXML), and Message. ReqABCSImpl populates this value.

EnterpriseServiceName

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

EnterpriseServiceOperationName

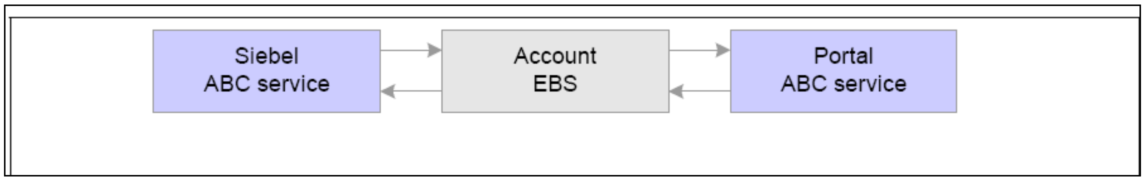
Name of the action of the EBS this message belongs. Known to the message creator, be it ABC service or EBS. ReqABCSImpl populates this value.

Custom

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

Use Case Example: Request/Response

GetAccountBalance - This will send a request message with account number as input and receive account balance as response.



Use Case Example: Request/Response

- Request EBM

  Two rows: One for the Process and one for the Request EBM Message involved in the process.

```
<BusinessScope>
<ID>GETACCTBAL/10 01</ID>
<InstanceID> Siebel- Portal-Get -Account-Balance</InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>GetAccountBalance</EnterpriseService
OperationName> </BusinessScope> <BusinessScope>
<ID>ACCTBALMSG/9001[the EBMID in the top element to be used
here]</ID>
<InstanceID> AccountBalanceReqMessage</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>GetAccountBalance</EnterpriseService
OperationName> </BusinessScope>
```

- Response EBM

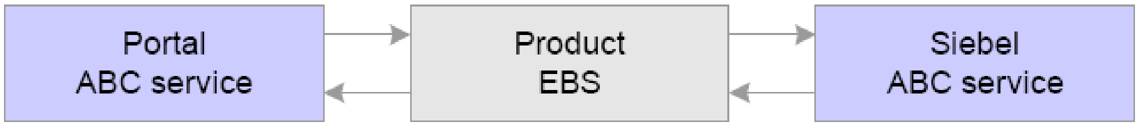
  Two rows: One for the process and one for the Response EBM Message involved in the process

```
<BusinessScope>
<ID>GETACCTBAL/10 01</ID>
<InstanceID> Siebel- Portal-Get -Account-Balance</InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>GetAccountBalance</EnterpriseService
OperationName> </BusinessScope>
<BusinessScope>
<ID>ACCTBALMSG/9002[the EBMID in the top elemt to be used here]</ID>
<InstanceID> AccountBalanceResMessage</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>GetAccountBalance</EnterpriseService
```

```
OperationName>
</BusinessScope>
```

Use Case Example: Asynchronous Process

SyncProduct - Portal will send an async message to Siebel to sync the product details.



Use Case Example: Asynchronous Process

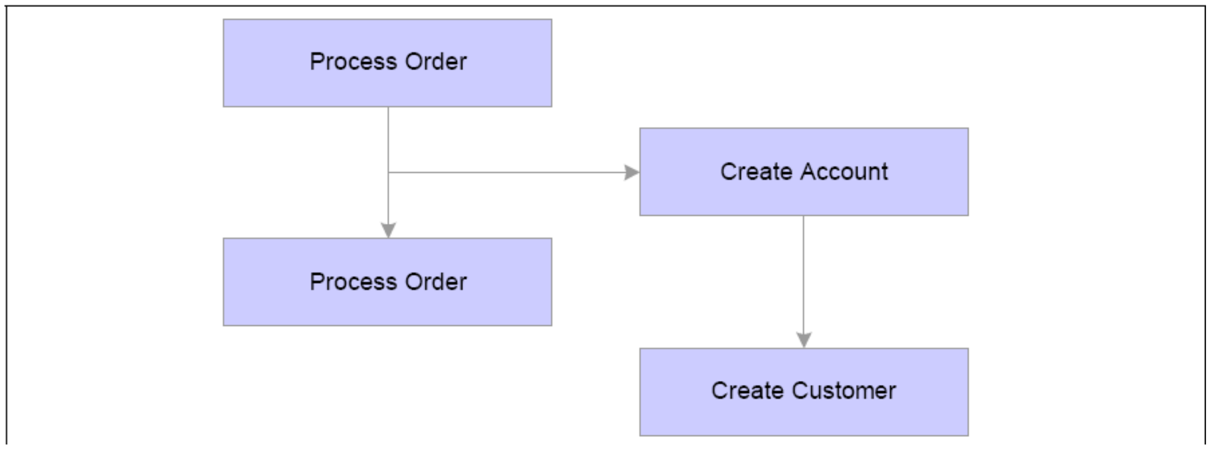
Request EBM

Two rows: One for the process and one for the Request EBM Message involved in the process.

```
<BusinessScope>
<ID>PRODSYNC/10 0 3</ID>
<InstanceID> Portal-Siebel-Product-Sync</InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>ProductEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>SyncProduct</EnterpriseServiceOperat
ionName>
</BusinessScope> <BusinessScope>
<ID>PRODSYNCREQ/90 03</ID>
<InstanceID> ProductSyncReqMessage</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>ProductEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>SyncProduct</EnterpriseServiceOperat
ionName>
</BusinessScope
```

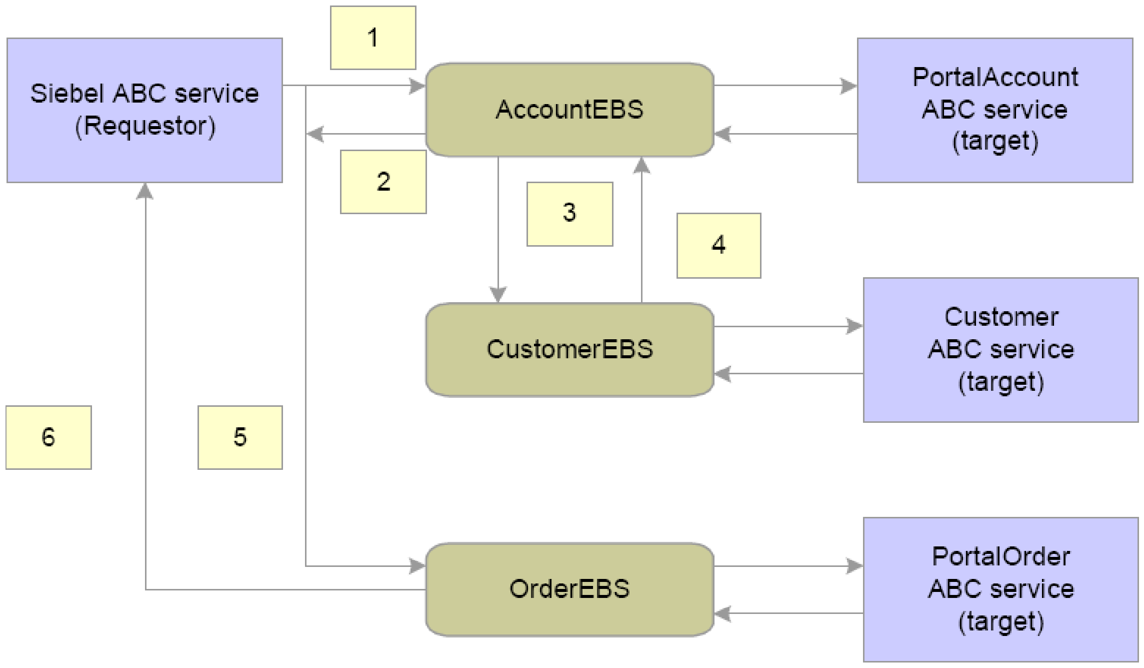
Use Case Example: Synchronous Process with Spawning Child Processes

ProcessOrder - Siebel will send an order. It will first spawn CreateAccount in Portal and then it will process the order in the portal.



Synchronous Process with Spawning Child Processes

This diagram illustrates the ProcessOrder flow:


Siebel ABC service (Requestor)
1
AccountEBS
PortalAccount ABC service (target)
2
3
4
CustomerEBS
Customer ABC service (target)
6
5
OrderEBS
PortalOrder ABC service (target)

Process Order flow

- ProcessOrder Request EBM (Message 5 in the diagram)

  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)

```
<BusinessScope>
<ID>ORDPROCESSING/10 04</ID>
<InstanceID>End-to-End-Order-Processing</InstanceID>
```

```
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>OrderEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>ProcessOrder</EnterpriseServiceOpera
tionName> </BusinessScope>
<BusinessScope>
<ID> PROCESSORDERREQMSG /9004</ID>
<InstanceID>Process-Order-Request-Message</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>OrderEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>ProcessOrder</EnterpriseServiceOpera
tionName> </BusinessScope>
<BusinessScope>
<ID> CREATEACCTREQMSG /9005</ID>
<InstanceID>Create-Account-Request-Message</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOper
ationName> </BusinessScope>
<BusinessScope>
<ID> CREATEACCTRESPMSG /9020</ID>
<InstanceID>Create-Account-Response-Message</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOper
ationName> </BusinessScope>
```

- Process Order Response EBM (Message 6 in the diagram)

  Two rows:

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

```
<BusinessScope>
<ID>ORDPROCESSING/10 04</ID>
<InstanceID>End-to-End-Order-Processing</InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>OrderEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>ProcessOrder</EnterpriseServiceOpera
tionName>
</BusinessScope> <BusinessScope>
<ID> PROCESSORDERRESPMSG /9019</ID>
<InstanceID>Process-Order-Response-Message</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>OrderEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>ProcessOrder</EnterpriseServiceOpera
tionName>
</BusinessScope>
```

- Create Account Request EBM (Message 1 in the diagram)

  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)

```
<BusinessScope>
<ID>CREATEACCT/10 0 8</ID>
<InstanceID>Portal-Account-Creation</InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOper
ationName> </BusinessScope>
<BusinessScope>
<ID>CREATEACCTREQMSG/9 0 0 8</ID>
<InstanceID>Create-Account-Request-Message</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOper
ationName> </BusinessScope>
<BusinessScope>
<ID>CREATECUSTREQMSG/9 0 0 9</ID>
<InstanceID>Create-Customer-Request-Message</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOpe
rationName> </BusinessScope>
<BusinessScope>
<ID>CREATECUSTRESPMSG/9021</ID>
<InstanceID>Create-Customer-Response-Message</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOpe
rationName>
</BusinessInstruction> >
```

- Create Account Response EBM (Message 2 in the diagram)

  Two rows:

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

```
<BusinessScope>
<ID>CREATEACCTRESP/10 0 8</ID>
<InstanceID>Portal-Account-Creation-Response</InstanceID>
<BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOper
ationName> </BusinessScope>
<BusinessScope>
<ID>CREATEACCTRESPMSG/9020</ID>
<InstanceID>Create-Account-Response-Message</InstanceID>
<BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
<EnterpriseServiceName>AccountEBS</EnterpriseServiceName>
<EnterpriseServiceOperationName>CreateAccount</EnterpriseServiceOper
ationName> </BusinessScope>
```

- Create Customer Request EBM (Message 3 in the diagram)

  Two rows:

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

  ```
  <BusinessScope>
  <ID>CREATECUST/10 09</ID>
  <InstanceID>Oracle-Customer-Create</InstanceID>
  <BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
  <EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
  <EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOpe
  rationName>
  </BusinessScope>
  <BusinessScope>
  <ID>CREATECUSTREQMSG/90 0 9</ID>
  <InstanceID>Create-Customer-Request-Message</InstanceID>
  <BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
  <EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
  <EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOpe
  rationName>
  </BusinessScope>
  ```

- Create Customer Response EBM (Message 4 in the diagram)

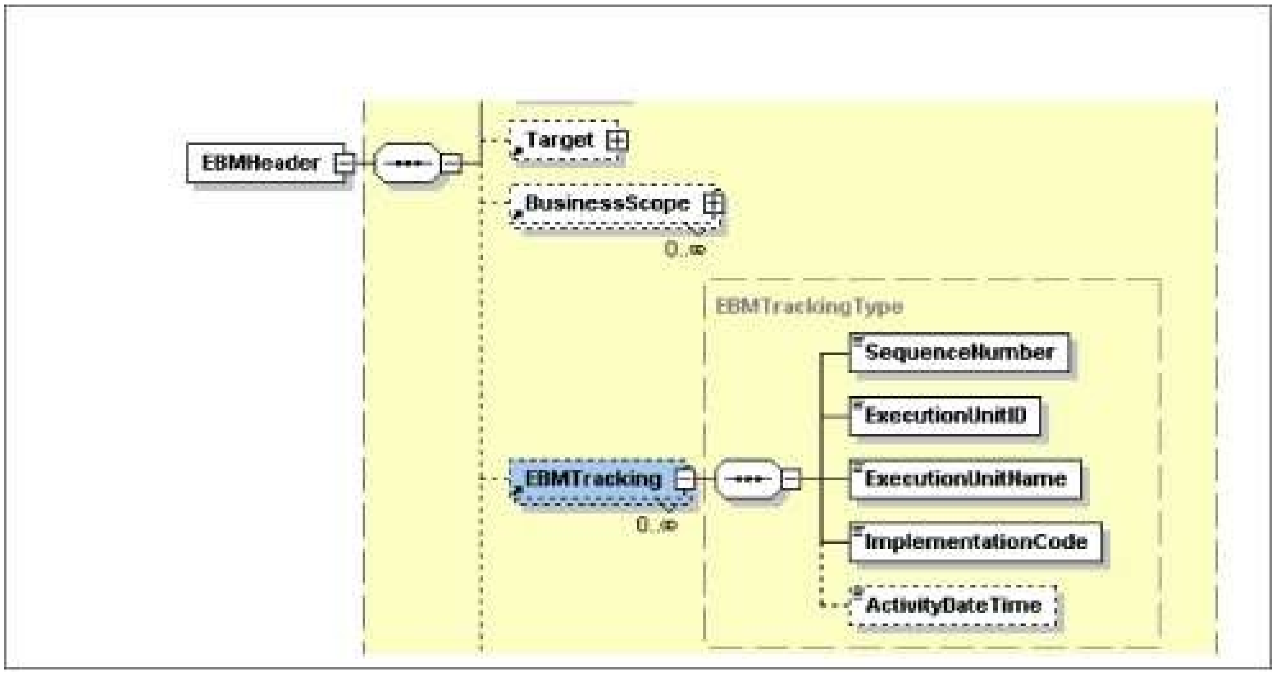
  Two rows:

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

  ```
  <BusinessScope>
  <ID>CREATECUSTRESP/10 09</ID>
  <InstanceID>Oracle-Customer-Create-Response</InstanceID>
  <BusinessScopeTypeCode>BusinessScope</BusinessScopeTypeCode>
  <EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
  <EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOpe
  rationName>
  </BusinessScope>
  <BusinessScope>
  <ID>CREATECUSTREQMSG/9021</ID>
  <InstanceID>Create-Customer-Response-Message</InstanceID>
  <BusinessScopeTypeCode>Message</BusinessScopeTypeCode>
  <EnterpriseServiceName>CustomerEBS</EnterpriseServiceName>
  <EnterpriseServiceOperationName>CreateCustomer</EnterpriseServiceOpe
  rationName>
  </BusinessScope>
  ```

EBMTracking

Contains tracking information about the each node that the EBM has been through. EBMTracking may appear multiple times, once for each execution unit it passes through.



Structure of the EBMTracking element

SequenceNumber

Contains the sequence number of the node the EBM has been through.

ExecutionUnitID

Contains the ID of the execution unit, node, or process ID.

ExecutionUnitName

Contains the fully qualified name of the execution unit, node, or process ID.

ImplementationCode

Contains the category of the execution unit, which indicates the type of the execution unit such as BPEL, ESB, 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)?
```

Example of EBMTracking:

```
<EBMHeader>
<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}QueryInvoicePortalProvABCSImpl</ExecutionUnitName>
<CategoryCode>BPEL</CategoryCode>
<Activity DateTime>2 001-12-17T09:30:47-05:00</ActivityDateTime>
</EBMTracking>
. . .
</EBMHeader>
```

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.

Example:

```
<EBMHeader>
  <Custom>
    . . .
</Custom>
    . . .
</EBMHeader>
```

Populating EBM Header Components

During the transformation phase in ESB, or during transformation or assignment in BPEL, you need to populate the appropriate EBM header fields as described here.

Populating 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 is a provider ABC implementation service.

This XSL snippet demonstrates populating the standard header elements.

```
<xsl:variable name="messageIDVariable" select="orcl:generate-
guid()"/>
<corecom:EBMID>
<xsl:value-of select="$messageIDVariable"/>
</corecom:EBMID>
<corecom:EBMName>
<xsl:text disable-output-
escaping="no">{http://xmlns.oracle.com/EnterpriseObjects/Industry/Te
lco/EBO/Item/V1}{SyncItemPublicationEBM}</xsl:text>
</corecom:EBMName >
<corecom:EBOName >
<xsl:text disable-output-
escaping="no">{http://xmlns.oracle.com/EnterpriseObj
ects/Industry/Telco/EBO/Item/V1}{Item}</xsl:text>
</corecom:EBOName >
<corecom:CreationDateTime>
<xsl:value-of select="xp20:current-dateTime()"/>
</corecom:CreationDateTime>
<corecom:VerbCode >
<xsl:text disable-output-escaping="no">Sync</xsl:text>
</corecom:VerbCode>
```

Populating Message Processing Instruction

Adds instructions to indicate how a message will be processed. This is normally used to tell the system to run the message in production mode or in simulation/testing mode. These fields are currently populated by the CAVS. Usually the fields are initially populated in the SOAP Header before initiating a testing request.

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

Populating 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 is a provider ABC implementation service.

This XPath function is provided to retrieve the Sender System information from Business Service Repository (BSR) based on the system ID/Code and then return the data as an XML Node-Set:

Namespace ebh = http://www.oracle.com/XSL/Transform/java /oracle.apps.aia.core.ebmheader.GetEBMHeaderDataFunction

```
ebh:getEBMHeaderSenderSystemNode(senderSystemCode as string,
senderSystemID as string,
) return senderSystem as node-set
```

Either the senderSystemCode or the senderSystemID should be passed. The function will locate the system information in BSR based on wither the code or the ID.

This is an XSL code snippet for populating the Send System information:

```
<corecom:Sender>
<corecom:ID>
<xsl:value-of select="$senderNodeID"/>
</corecom:ID>
<corecom:Description>
<xsl:value-of select="$senderNodeVariable/Description"/>
</corecom:Description>
<corecom:IPAddress>
<xsl:value-of select="$senderNodeVariable/IPAddress"/>
</corecom:IPAddress>
<corecom:CallingServiceName>
<xsl:text disable-output-escaping="no">
SyncPriceListLineListSiebelProvABCSImpl</xsl:text>
</corecom:CallingServiceName>
<corecom:Application>
<corecom:ID>
<xsl:value-of select="$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>
<xsl:call-template name="SenderType_ext">
<xsl:with-param name="currentNode" select="."/>
</xsl:call-template>
</corecom:Sender>
```

Adding Object Cross Reference information in the Sender System Information

Adds a cross-reference entry in the Sender System's ObjectCrossReference.

- EBOID
- SenderObjectIdentification
  - ID
  - NAME
  - ContextID

Populating Target System Information

Adds the target system information. This is usually populated to override the standard routing. EBS or provider ABC services can set these elements to indicate where the request should be routed.

```
<ns2:Target>
<ns2:ID>
<xsl:text disable-output-escaping="no">ORCL01</xsl:text>
</ns2:ID>
</ns2:Target>
```

Adding 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 will have 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.

This example 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 are not taken into account.

```
<corecom:BusinessScope>
<xsl:variable name="processIDVariable" select = "orcl:generate-guid
()"/>
<corecom:ID>
<xsl:value-of select='concat("SYNCITEM","/",$processIDVariable)'/>
</corecom:ID
<corecom:InstanceID>
<xsl:text disable-output-escaping="no">Portal-to-Siebel-Product -
Synchronization-Process</xsl:text>
</corecom:InstanceID>
<corecom:BusinessScopeTypeCode>
<xsl:text disable-output-escaping="no">BusinessProcess</xsl:text>
</corecom:BusinessScopeTypeCode>
<corecom:EnterpriseServiceName>
<xsl:text disable-output-escaping="no">ItemEBS</xsl:text>
</corecom:EnterpriseServiceName>
<corecom:EnterpriseServiceOperationName>
<xsl:text disable-output-escaping="no">Sync</xsl:text>
</corecom:EnterpriseServiceOperationName>
<xsl:call-template name="BusinessScopeType_ext">
<xsl:with-param name="currentNode" select="."/>
</xsl:call -template> </corecom:BusinessScope>
<corecom:BusinessScope>
<corecom:ID>
<xsl:value-of
select='concat("SYNCITEMREQMSG","/",$messageIDVariable)'
</corecom:
<corecom:InstanceID>
<xsl:text disable-output-escaping="no">Portal-to-Siebel-Product -
```

```
Synchronization-Request</xsl:text>
</corecom:InstanceID>
<corecom:BusinessScopeTypeCode>
<xsl:text disable-output-escaping="no">BusinessProcessMessage</xsl:
text>
</corecom:BusinessScopeTypeCode>
<corecom:EnterpriseServiceName>
<xsl:text disable-output-escaping="no">ItemEBS</xsl:text>
</corecom:EnterpriseServiceName>
<corecom:EnterpriseServiceOperationName>
<xsl:text disable-output-escaping="no">Sync</xsl:text>
</corecom:EnterpriseServiceOperationName>
</corecom:BusinessScope>
```

Populating EBM Tracking Information

Adds an EBMTracking entry 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:

This section will have one row each for requester ABCSImpl, EBS and Provider ABCSImpl. This transformation will be for requester ABCSImpl.

```
<ns2:EBMTracking>
<ns2:SequenceNumber>
<xsl:value-of select="position()"/>
</ns2:SequenceNumber>
<ns2:ExecutionUnitID>
<xsl:value-of select="ora:getInstanceId()"/>
</ns2:ExecutionUnitID>
<ns2:ExecutionUnitName>
<xsl:text disable-output-
escaping="no">ProcessInvoiceSiebelReqABCS.bpel
</xsl:text>
</ns2:ExecutionUnitName>
<ns2:ImplementationCode>
<xsl:text disable-output-escaping="no">BPEL</xsl:text>
</ns2: ImplementationCode>
<ns2:ActivityDateTime>
<xsl:value-of select="xp2 0:current-dateTime()"/>
</ns2:ActivityDateTime>
</ns2:EBMTracking>
For EBS add this row transformation (you have to map all other data
elements along with adding this section)
<ns2:EBMTracking>
<ns2:SequenceNumber>
<xsl:value-of select="position() + 1"/>
</ns2:SequenceNumber>
<ns2:ExecutionUnitID>
```

```
<xsl:value-of select="ora:getInstanceId()"/>
</ns2:ExecutionUnitID>
<ns2:ExecutionUnitName>
<xsl:text disable-output-escaping="no">AccountEBS.esb</xsl:text>
</ns2:ExecutionUnitName>
<ns2: ImplementationCode>
<xsl:text disable-output-escaping="no">ESBL</xsl:text>
</ns2: ImplementationCode>
<ns2:ActivityDateTime>
<xsl:value-of select="xp20:current-dateTime()"/>
</ns2:ActivityDateTime>
</ns2:EBMTracking> For provider ABCSImpl add this row in
transformation (if you are using BPEL just use assign for this, you
cannot add this row in ESB)
<ns2:EBMTracking>
<ns2:SequenceNumber>
<xsl:value-of select="position() + 1"/>
</ns2:SequenceNumber>
<ns2:ExecutionUnitID>
<xsl:value-of select="ora:getInstanceId()"/>
</ns2:ExecutionUnitID> <ns2:ExecutionUnitName>
<xsl:text disable-output-escaping="no">
ProcessInvoiceSiebelProvABCS.bpel</xsl:text>
</ns2:ExecutionUnitName>
<ns2: ImplementationCode>
<xsl:text disable-output-escaping="no">BPEL</xsl:text>
</ns2: ImplementationCode>
<ns2:ActivityDateTime>
<xsl:value-of select="xp20:current-dateTime()"/>
</ns2:ActivityDateTime>
</ns2:EBMTracking>
```

Chapter 10: Working with Message Routing

This chapter discusses:

- Routing at the enterprise business service (EBS).
- Routing at the application business connector (ABC) service implementation.
- Identifying the target system.

Routing at the EBS Service

Routing rules are specified for each operation defined on EBS services. The routing rules are used to decide where the incoming enterprise business message (EBM) should be routed to, either an enterprise business flow (EBF), EBS, ABC service, or the Composite Application Validation System (CAVS). The routing rules are specified as XPath expressions in the filter of the ESB routing rule.

Routing rules need to be mutually exclusive since all the rules will be evaluated and messages will be routed to an end point based on the rule evaluation.

Each of the operations of an EBS should have these rules at the minimum:

- One routing rule for CAVS enabling.

  This rule should check whether the EBM Header > MessageProcessingInstruction > EnvironmentCode is set to ***CAVS***.

  For more information, see Chapter 12: Developing Oracle AIA Services to be CAVS-Enabled and Provide Multi-Instance Support.

- One or more routing rules to connect to provider the ABC service or EBF.

  The filter expression specified in these routing rules has to ensure that the message is not a test message. For each ABC service or EBF, there will be one routing rule. The conditions can be one of these:

  - Target System ABC service populated in the EBM header

    In this case the filter will have an expression to check whether the target system ABC service in the EBM header has been pre-populated and the Override Routing Indicator is set to false.

    Example of a filter expression for a SalesOrderEBS Routing Rule for determining the target system ABC service is provided here.

/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 ABC service. The filter expression should ensure that the target system information has not been prepopulated in the EBM header.

  Example expression:

  starts-with(/sordebo: CreateSalesOrderEBM/sordebo: DataArea/sordebo: Create SalesOrder/sordebo: SalesOrderLine/sordebo:SalesOrderLineSchedule/ns5:ShipToPartyReference/ns5:LocationReference /ns5:Address/ns5:CountrySubDivisionCode,'9') and /sordebo:CreateSalesOrderEBM/ns5:EBMHeader /ns5:Target/ns5:ID = ""

- Enable error handling and logging

  EBS services should handle errors to allow clients or administrators to resubmit or re-trigger processes. This will be done through a central error handler.

For more information, see Chapter 13: Configuring Oracle AIA Processes for Error Handling and Trace Logging.

Routing at the Provider ABC Service Implementation

The Provider ABC service can be routed either to CAVS or to the target application. There are two situations:

- Dynamic SOAP PartnerLink and CAVS enablement within Provider ABC service Impl
- Dynamic Non-SOAP PartnerLink and CAVS enablement within Provider ABC service Impl

Dynamic SOAP PartnerLink and CAVS enablement Within Provider ABC Service Impl

Code in bold text should be replaced with code specific to your service.

Oracle AIA Configuration properties defined for the service:

```
<ServiceConfiguration
serviceName="{http://xmlns.oracle.com/ABCSImpl/Siebel
/Industry/Telco/QueryCustomerPartyListSiebelProvABCSImpl/V1}QueryCus
tomerPartyList
SiebelProvABCSImpl">
<Property
```

```
name="CMU_spcAccount_spcQuery.Routing.Target.Default.EndpointURI">ht
tp:
//64.181.170.13
5/eai_enu/start.swe?SWEExtSource=SecureWebService&amp;SWEExtCmd=
Execute&amp;UserName=j shen&amp;Password=jshen</Property>
<Property
name="CMU_spcAccount_spcQuery.Routing.RouteToCAVS">false</Property>
</ServiceConfiguration>
```

In the BPEL process file:

```
. . .
xmlns:wsa=http://schemas.xmlsoap.org/ws/20 03/03/addressing
. . .
<variables>
. . .
<variable name="TargetEndpointLocation" type="xsd:string"/>
<variable name="EndpointReference" element="wsa:EndpointReference"/>
. . .
</variables>
. . .
<sequence name="SetDynamicPartnerlink">
<bpelx:exec name="GetTargetEndpointLocation" language="java"
version="1.5">
<![CDATA[java.lang.String targetEndpointLocation = null;
// check configuration for routing to CAVS try { boolean routeToCAVS
=
java.lang.Boolean.parseBoolean(oracle.apps.aia.core.config.Configura
tion.getServiceProperty("{http://xmlns.oracle.com/ABCSImpl/Siebel
/Industry/Telco/QueryCustomerPartyListSiebelProvABCSImpl/V1}QueryCus
tomerPartyList SiebelProvABCSImpl",
"CMU_spcAccount_spcQuery.Routing.RouteToCAVS"));
if (routeToCAVS) {
targetEndpointLocation =
oracle.apps.aia.core.config.Configuration.get
SystemModuleProperty("CAVS",
"SyncResponseSimulator.Soap.EndpointURL");
//addAuditTrailEntry("Endpoint location = '" +
targetEndpointLocation + "' selected from
SyncResponseSimulator.Soap.EndpointURL configuration.");
}
}
catch (oracle.apps.aia.core.config.PropertyNotFoundException e) {
}
if (targetEndpointLocation==null || targetEndpointLocation=="") {
// try to get EndpointLocation from EBM Header
try {
oracle.xml.parser.v2.XMLElement targetEndpointLocationElement =
(oracle.xml.parser.v2.XMLElement)getVariableData("QueryCustomerParty
ListRequest Msg", "QueryCustomerPartyListEBM",
"/telcocustmr:QueryCustomerPartyListEBM
/corecom:EBMHeader/corecom:Target/corecom:EndPointURI[text() ! = '
'] ") ;
targetEndpointLocation = targetEndpointLocationElement.getText();
//addAuditTrailEntry("Endpoint location = '" +
targetEndpointLocation + "'
selected from EBM header.");
}
catch (com.oracle.bpel.client.BPELFault e) {
//addAuditTrailEntry("/telcocustmr:QueryCustomerPartyListEBM/corecom
:
EBMHeader/corecom:Target/corecom: EndPointURI not found.");
}
if (targetEndpointLocation==null || targetEndpointLocation=="") {
// try to get EndpointLocation from Configuration
try {
targetEndpointLocation =
oracle.apps.aia.core.config.Configuration.get
ServiceProperty("{http://xmlns.oracle.com/ABCSImpl/Siebel/Industry/T
elco/Query
CustomerPartyListSiebelProvABCSImpl/V1}QueryCustomerPartyListSiebelP
rovABCSImpl",
"CMU_spcAccount_spcQuery.Routing.Target.Default.EndpointURI");
//addAuditTrailEntry("Endpoint location = '" +
targetEndpointLocation + "' selected from configuration
properties.");
}
catch (oracle.apps.aia.core.config.PropertyNotFoundException e) {
//addAuditTrailEntry("Routing.Target.Default.EndpointLocation not
found.");
}
}
}
if (targetEndpointLocation!=null && targetEndpointLocation!="") {
setVariableData("TargetEndpointLocation",targetEndpointLocation);
//addAuditTrailEntry("TargetEndpointLocation = " +
targetEndpointLocation);
}]]>
</bpelx:exec>
<switch name="Switch_SetEndpoint">
<case condition="string-length(bpws:getVariableData('TargetEndpoint
Location'))>0">
<assign name="AssignPartnerlinkEndpointReference">
<copy>
<from>
<wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws
/20/03/03/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="CMU_spcAccount_spcQuery"/>
</copy>
</assign>
</case>
<otherwise>
```

```
<empty name="Empty_NoSetEndpoint"
</otherwise>
</switch>
</sequence>
. . .
```

Dynamic Non-SOAP PartnerLink and CAVS Enablement Within Provider ABC Service Impl

Code in bold should be replaced with code specific to your service.

Oracle AIA Configuration properties defined for the service:

```
<ServiceConfiguration
serviceName="{http://xmlns.oracle.com/ABCSImpl/Portal
/Industry/Telco/CreateCustomerPartyPortalProvABCSImpl
/V1}CreateCustomerPartyPortal
ProvABCSImpl">
<Property
name="BRMCUSTService.Routing.Target.Default.EndpointURI">eis/BRM<
/Property>
<Property name="BRMCUSTService.Routing.RouteToCAVS">false</Property>
</ServiceConfiguration>
```

Local PartnerLink WSDL file:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CUST"
targetNamespace="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes
[Target WSDL Namespace] "
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:brm="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes"
xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/j ca/"
xmlns:plt="http://schemas.xmlsoap.org/ws/2 003/05/partner-link/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
<import
namespace="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes"
location="http://ap6032 fems.us.oracle.com:7831 /AIAComponents
/ApplicationObjectLibrary/Portal/BRM7 3/wsdls/BRMCUSTService.wsdl"/>
<binding name="CAVSAnyBinding" type="brm:BRMCUSTService_ptt [Port
type from Target wsdl]">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap
/http"/>
<operation name="PCM_OP_CUST_SET_STATUS [operation from target
wsdl]">
<soap:operation style="document" soapAction="simulate"/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="PCM_OP_CUST_COMMIT_CUSTOMER" >
<soap:operation style="document" soapAction="simulate"/>
```

```
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="PCM_OP_CUST_DELETE_PAYINFO">
<soap:operation style="document" soapAction="simulate"/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="PCM_OP_CUST_UPDATE_SERVICES">
<soap:operation style="document" soapAction="simulate"/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="PCM_OP_CUST_MODIFY_CUSTOMER">
<soap:operation style="document" soapAction="simulate"/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="PCM_OP_CUST_UPDATE_CUSTOMER">
<soap:operation style="document" soapAction="simulate"/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>
<service name="CAVSAnyService">
<port name="CAVSAnyPort" binding="brm:CAVSAnyBinding"
<soap:address location="http://set.at.runtime"/>
</port>
</service>

</definitions>
```

In the BPEL process file:

```
. . .
xmlns:wsa=http://schemas.xmlsoap.org/ws/20 03/03/addressing
xmlns:brm=http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes
```

```
. . .
<variables>
. . .
<variable name="EndpointReference" element="wsa:EndpointReference"/>
<variable name="TargetEndpointLocation" type="xsd:string"/>
<variable name="QualifiedServiceName" type="xsd:string"/>
. . .
</variables>
. . .
<sequence name="SetDynamicPartnerlink">
<bpelx:exec name="GetTargetEndpointLocation" language="java"
version="1.5">
<![CDATA[java.lang.String targetEndpointLocation = null;
java.lang.String qualifiedServiceName = "brm:BRMCUSTService" ;
// check configuration for routing to CAVS
try {
boolean routeToCAVS =
java.lang.Boolean.parseBoolean(oracle.apps.aia.core.
config.Configuration.getServiceProperty("{http://xmlns.oracle.com/AB
CSImpl/
Portal
/Industry/Telco/CreateCustomerPartyPortalProvABCSImpl/V1}CreateCusto
merPartyPortal ProvABCSImpl",
"BRMCUSTService.Routing.RouteToCAVS"));
if (routeToCAVS) {
targetEndpointLocation =
oracle.apps.aia.core.config.Configuration.get
SystemModuleProperty("CAVS",
"SyncResponseSimulator.Soap.EndpointURL");
//addAuditTrailEntry("Endpoint location = '" +
targetEndpointLocation + "'
selected from SyncResponseSimulator.Soap.EndpointURL
configuration.");
if (targetEndpointLocation!=null && targetEndpointLocation!="")
qualifiedServiceName = "brm:CAVSAnyService";
}
}
catch (oracle.apps.aia.core.config.PropertyNotFoundException e)
{
if (targetEndpointLocation==null || targetEndpointLocation=="") {
// try to get EndpointLocation from EBM Header try
{
oracle.xml.parser.v2.XMLElement targetEndpointLocationElement =
(oracle.xml.parser.v2.XMLElement)getVariableData("CreateCustomerPart
yRequestMsg",
"CreateCustomerPartyEBM",
"/telcocustmr:CreateCustomerPartyEBM/corecom:
EBMHeader/corecom:Target/corecom:EndPointURI[text()!='']");
targetEndpointLocation = targetEndpointLocationElement.getText();
//addAuditTrailEntry("Endpoint location = '" +
targetEndpointLocation + "'selected from EBM header.");
}
catch (com.oracle.bpel.client.BPELFault e) {
//addAuditTrailEntry("/telcocustmr:CreateCustomerPartyEBM/corecom:
EBMHeader/corecom:Target/corecom: EndPointURI not found.");
}
if (targetEndpointLocation==null || targetEndpointLocation=="") {
```

```
// try to get EndpointLocation from Configuration
try {
targetEndpointLocation =
oracle.apps.aia.core.config.Configuration.get
ServiceProperty("{http://xmlns.oracle.com/ABCSImpl/Portal/Industry/T
elco/Create
CustomerPartyPortalProvABCSImpl/V1}CreateCustomerPartyPortalProvABCS
Impl", "BRMCUSTService.Routing.Target.Default.EndpointURI");
//addAuditTrailEntry("Endpoint location = '" +
targetEndpointLocation
+ "' selected from configuration properties.");
}
catch (oracle.apps.aia.core.config.PropertyNotFoundException e) {
//addAuditTrailEntry("Routing.Target.Default.EndpointLocation not
found.");
}
}
}
if (targetEndpointLocation!=null && targetEndpointLocation!="") {
setVariableData("TargetEndpointLocation",targetEndpointLocation);
setVariableData("QualifiedServiceName",qualifiedServiceName);
//addAuditTrailEntry("TargetEndpointLocation = " +
targetEndpointLocation)
}]]>
</bpelx:exec>
<switch name="Switch_SetEndpoint">
<case condition="string-length(bpws:getVariableData('TargetEndpoint
Location'))>0">
<assign name="AssignPartnerlinkEndpointReference">
<copy>
<from>
<wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws^ /20
03/03/addressing">
<wsa:Address/>
<wsa:ServiceName xmlns:brm="http://xmlns.oracle.com/BRM/schemas
/BusinessOpcodes"/>
</wsa:EndpointReference>
</from>
<to variable="EndpointReference"/>
</copy>
<copy>
<from variable="TargetEndpointLocation"/>
<to variable="EndpointReference"
query="/wsa:EndpointReference/wsa:Address"/> <
/copy>
<copy>
<from variable="QualifiedServiceName"/>
<to variable="EndpointReference"
query="/wsa:EndpointReference/wsa:ServiceName"/>
</copy>
<copy>
<from variable="EndpointReference"/>
<to partnerLink="BRMCUSTService"/>
</copy>
</assign>
</case>
<otherwise>
```

```
<empty name="Empty_NoSetEndpoint"/>
</otherwise>
</switch>
</sequence>
```

Identifying the Target System in the EBS

The EBS can route the request from the Requester ABC service/EBF or another EBS to one of the many Provider ABC services available. The target system needs to 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 the following steps leads to the correct ABC service:

1. Using content-based routing.

   The routing rule in the EBS is based on content of the message and this is used to decide the target ABC service. 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 ABC services.

2. Using the target system information in the EBM header.

   If the target system information is already set in the EBM, this is used for routing to the correct ABC service.

3. From the Xref for operations other than Create.

   For all other operations other than the Create, the target system has already been figured out and the cross reference IDs have been set. So in this case, the Xref function lookupPopulatedColumns is used to identify the systems in which the data synchronization of entity has been done.

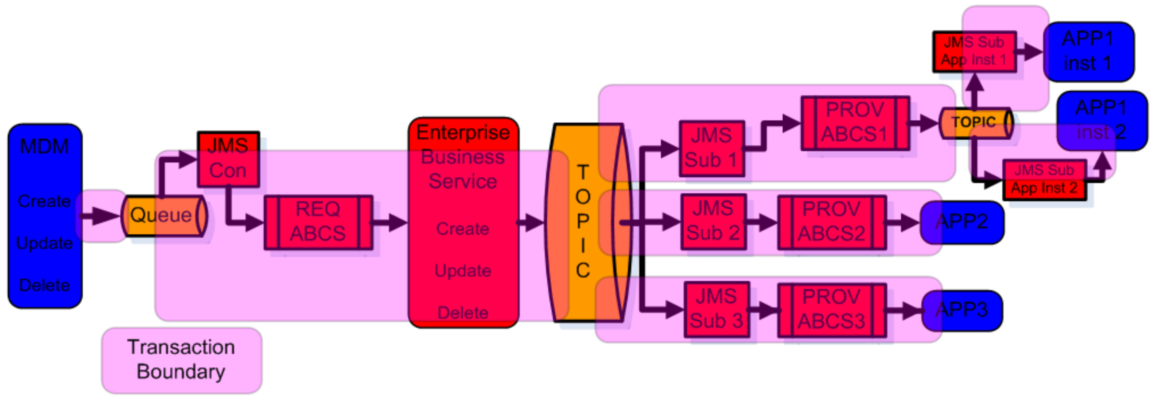
Chapter 11: Enabling Transactions for an Integration Scenario in Asynchronous Patterns

This chapter discusses how to:

- Describe transactions in asynchronous message exchange patterns.
- Configure global transactions across Oracle Application Integration Architecture (AIA) services in an integration scenario.

Describing Transactions in Asynchronous Message Exchange Patterns

Asynchronous message exchange patterns aid in processing messages in a store-and-forward manner. This aids in guaranteed delivery of messages and scalability. This diagram depicts a use case in which a Master Data Management (MDM) application publishes updates with recipients unknown.



Master Data Management application publishing updates with recipients unknown

- MDM, or the master application, pushes messages (lists of entities) to the Java Message Service (JMS) queue.
- The message in the queue triggers the JMS consumer service, a JMS adapter configured on an Enterprise Service Bus (ESB) service, which invokes the requestor application business connector (ABC) service.
- The requestor ABC service (BPEL implementation) splits the message, if needed, and invokes the enterprise business service (EBS).
- The EBS (ESB service) pushes message to the topic.
- Durable JMS subscribers, a JMS adapter configured on an ESB service, receives the

message copy and pushes it to the provider ABC service.

- Each of the provider ABC services (BPEL implementations) pushes the message to the participating application.
- If there is more than one participating application instance, as in the case of APP1 illustrated in the diagram, the message is pushed to another JMS topic.
- Durable JMS subscribers, a JMS adapter configured on an ESB service, receives the message copy and pushes it to individual APP1 instances.

As illustrated in the diagram, the message is processed using multiple transactions. This is a depiction of the store-and-forward mechanism. This ensures that the message is never lost.

Configuring Global Transactions Across Oracle AIA Services in an Integration Scenario

This section provides configuration details for these scenarios:

- The ESB service participates in a global transaction initiated by the JMS server.
- The ESB service invokes BPEL as a part of a global transaction.
- The BPEL service invokes an ESB service as a part of a global transaction.

ESB Service Participates in a Global Transaction Initiated by the JMS Server

A JMS adapter is configured on an ESB service when an application pushes a message to a JMS queue. In the case of Oracle E-Business Suite pushing messages to Oracle Advanced Queuing (AQ), an Oracle Application (OA) adapter is configured on an ESB service.

For the JMS adapter:

Follow the steps provided in Configuring JMS Adapter.

For the OA adapter:

- Create a separate XA data source for the inbound or outbound adapter’s connection factory.
- Add the properties for the oc4-ra.xml file as illustrated here:

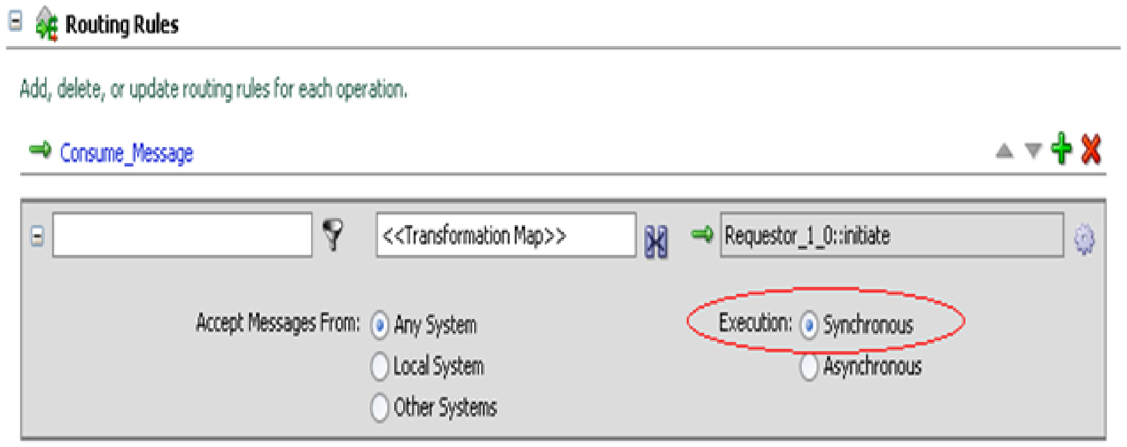
```
$ORACLE_HOME/product/10.1.3.1/OracleAS_1/j2ee/oc4j_soa/application-
deployments/default/AppsAdapter/oc4j-ra.xml
<connector-factory location="eis/Apps/apps1" connector-name="Oracle
Applications Adapter">
         <config-property name="xADataSourceName"
value="jdbc/Apps1DataSource"/>
         <config-property name="dataSourceName" value=""/>
         <connection-pooling use="none">
         </connection-pooling>
         <security-config use="none">
         </security-config>
</connector-factory>
```

- Add the properties for the data-sources.xml file as illustrated here:

```
$ORACLE_HOME /product/10.1.3.1/OracleAS_1/j2ee/oc4j_soa/config/data-
sources.xml
<connection-pool name='dbSample_CONNECTION_POOL'
inactivity-timeout='60'
validate-connection='false'
num-cached-statements='5'
time-to-live-timeout='600'>

<connection-factory factory-
class='oracle.jdbc.xa.client.OracleXADataSource'
user='scott'
password='tiger'
url="..."
</connection-factory>

</connection-pool>

<managed-data-source name='DBSampleNonEmulatedDS'
connection-pool-name='dbSample_CONNECTION_POOL'
jndi-name='jdbc/dbSample' tx-level="global" />
```

ESB Service Invokes a BPEL Service as a Part of a Global Transaction

To ensure that the ESB service invokes the BPEL as a part of the global transaction, the ESB routing rule execution should be set to ***Synchronous***, as depicted in this screenshot.



<<Transformation Map>>
Requestor_1_0::initiate
Accept Messages From: [✓ Any System] Local System Other Systems
Execution: [✓ Synchronous] Asynchronous

Set the ESB routing rule execution to ***Synchronous***

Additionally, these process-level configuration properties are required in the target BPEL process' bpel.xml file:

```
<configurations>
   <property name="transaction">participate</property>
   <property name="deliveryPersistPolicy">off.immediate</property>
```

```
      <property name="handleTopLevelFault">false</property>
    </configurations>
```

BPEL Service Invokes an ESB Service as a Part of a Global Transaction

These partnerLink properties need to be added to the “bpel.xml” for all partnerLinks to be included in the global transaction:

```
   <partnerLinkBinding name="PartnerLink_1">

      <property name="wsdlLocation">your wsdl location</property>
      <property name="transaction">participate</property>
   </partnerLinkBinding
```

Chapter 12: Developing Oracle AIA Services to be CAVS-Enabled and Provide Multi-Instance Support

This chapter provides instructions on how to develop Oracle Application Integration Architecture (AIA) services that are ready to work with the Composite Application Validation System (CAVS) and provide multi-instance support. Specifically, this chapter discusses how to:

- Develop provider application business connector (ABC) services to be CAVS-enabled and provide multi-instance support.
- Develop requester ABC services to be CAVS-enabled and provide multi-instance support.
- Develop enterprise business flows (EBFs) to be CAVS-enabled.

Note: 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.

This chapter also provides these supporting discussions:

- Describing CAVSEndpointURL value designation.
- Purging CAVS-related cross-reference entries to enable rerunning of test scenarios.

For more information about using the CAVS, see Oracle Application Integration Architecture Core Infrastructure Components Guide, "Working with the CAVS."

Developing Provider ABC Services to be CAVS-Enabled and Provide Multi-Instance Support

The following steps describe how to code a provider ABC service using for dynamic PartnerLinks to invoke target participating application web services and provide CAVS enablement and multi-instance support.

1. The provider ABC service must have these service-level configuration properties defined:

```
<Property name="Default.SystemID">[DefaultTargetSystemID]</Property>

<!-- Partnerlink for which SOAP invocation is OK -->
<Property
name="Routing.[PartnerlinkName].RouteToCAVS">[true|false]</Property>
<Property
name="Routing.[PartnerlinkName].[TargetSystemID].EndpointURI">[SoapE
ndpointURL]</Property>
<Property
```

```
name="Routing.[PartnerlinkName].CAVS.EndpointURI">[CAVSSoapEndpointU
RL]</Property>

<!-- Partnerlink for which SOAP invocation is NOT OK, and ESB
binding must be used for Tx support -->
<Property
name="Routing.[PartnerlinkName].RouteToCAVS">[true|false]</Property>
<Property
name="Routing.[PartnerlinkName].[TargetSystemID].ServiceName">[{Serv
iceNamespace}ServiceName*]</Property>
<Property
name="Routing.[PartnerlinkName].CAVS.EndpointURI">[CAVSSoapEndpointU
RL]</Property>
<Property
name="Routing.[PartnerlinkName].CAVS.ServiceName">[{ServiceNamespace
}ServiceName*]</Property>
```

Note. The ServiceName here is not the logical Oracle AIA service name. It must reflect the @name attribute of the <service> element found in the service's runtime WSDL. See step 7 below.

2. Ensure that you have the following namespace prefixes defined in your BPEL process:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
xmlns:corecom=http://xmlns.oracle.com/EnterpriseObjects/Core/Common/
V2
```

3. Add this variable to your global variables section:

```
<variable name="SystemID" type="xsd:string"/>
```

4. Add the AddTargetSystemID.xsl file to your BPEL project in the 'bpel' directory. Be sure to replace the `[ABCServiceNamespace]` and `[ABCServiceName]` tokens appropriately inside the file.

5. Add the following assignment as the first step in the BPEL process. Be sure to replace the tokens appropriately:

```
<assign name="GetTargetSystemID">
    <copy>
        <from expression="ora:processXSLT('AddTargetSystemID.xsl',
bpws:getVariableData('[RequestEBMVariableName]',
'[RequestEBMPartName]'))"/>
        <to variable="[RequestEBMVariableName]"
            part="[RequestEBMPartName]"/>
    </copy>
    <copy>
        <from variable="[RequestEBMVariableName]"
              part="[RequestEBMPartName]"

query="/[NamespacePrefixedEBMName]/corecom:EBMHeader/corecom:Target/
corecom:ID"/>
        <to variable="SystemID"/>
    </copy>
</assign>
```

6. Add the following <scope> once for each PartnerLink prior to its invoke activity.
   - Replace the tokens in the 'AssignDynamicPartnerlinkVariables' assignment activity to set the variables appropriately.
   - Also update the PartnerLink name token in the 'AssignPartnerlinkEndpointReference' assignment activity at the bottom.
   - There should not be any need to update the embedded Java code.

```
<scope name="SetDynamicPartnerlinkScope">
    <variables>
        <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 cavsServiceNamePropertyName =
"Routing."+partnerLinkName+".CAVS.ServiceName";
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 routeToCavsPropertyName =
"Routing."+partnerLinkName+".RouteToCAVS";
java.lang.String defaultSystemIdPropertyName = "Default.SystemID";
java.lang.String targetEndpointLocation = null;
java.lang.String targetServiceName = null;
java.lang.String targetServiceNamespace = null;
java.util.regex.Pattern serviceQNamePattern =
Pattern.compile("^\s*(\{(.*?)\})?(.*)");
java.lang.String targetID = null;
boolean addAudits = false;   //set this to true only for debugging

if (addAudits) addAuditTrailEntry("Partnerlink = " + partnerLinkName);

// check configuration for CAVS routing flag
boolean routeToCAVS = false;
try {
    routeToCAVS =
java.lang.Boolean.parseBoolean(oracle.apps.aia.core.config.Configuration.get
ServiceProperty(serviceName, routeToCavsPropertyName));
    if (addAudits) addAuditTrailEntry("RouteToCAVS = " + routeToCAVS);
} catch (java.lang.Exception e) {
    if (addAudits) addAuditTrailEntry("Configuration property " +
routeToCavsPropertyName + " not found!");
}

// get CAVS endpoint and/or service name
if (routeToCAVS) {
    try {
        targetEndpointLocation =
oracle.apps.aia.core.config.Configuration.getServiceProperty(serviceName,
cavsEndpointPropertyName);
        if (addAudits) addAuditTrailEntry("Endpoint = '" +
targetEndpointLocation + "' selected from configuration property " +
cavsEndpointPropertyName);
    } catch (java.lang.Exception e) {
        if (addAudits) addAuditTrailEntry("Configuration property " +
cavsEndpointPropertyName + " not found!");
        targetEndpointLocation = null;
    }
    try {
        targetServiceName =
oracle.apps.aia.core.config.Configuration.getServiceProperty(serviceName,
cavsServiceNamePropertyName);
        if (addAudits) addAuditTrailEntry("ServiceName = '" +
targetServiceName + "' selected from configuration property " +
cavsServiceNamePropertyName);
        // parse the QName for namespace and service name
        java.util.regex.Matcher m =
serviceQNamePattern.matcher(targetServiceName);
        m.matches();
        targetServiceNamespace = m.group(2)!=null ? m.group(2).trim() :
null;
        targetServiceName = m.group(3)!=null ? m.group(3).trim() : null;
    } catch (java.lang.Exception e) {
        if (addAudits) addAuditTrailEntry("Configuration property " +
cavsServiceNamePropertyName + " not found!");
        targetServiceName = null;
    }
}

else {  // NOT CAVS

    // check bpel variable for already retrieved target system Id
    try {
        targetID =
(java.lang.String)getVariableData(systemIdBpelVariableName);
```

```
        if (addAudits && targetID!=null) addAuditTrailEntry("Using
previously stored Target System ID = '" + targetID + "'");
    } catch (com.oracle.bpel.client.BPELFault e) {}

    // if not found, try to get it from the EBM header
    if (targetID==null || targetID=="") {
        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");
        }

        // Set the SystemID variable
        try {
            if (targetID!=null && targetID!="")
setVariableData(systemIdBpelVariableName, targetID);
        } catch (com.oracle.bpel.client.BPELFault e) {}
    }


    // if still not found, try to get it from configuration properties
    if (targetID==null || targetID=="") {
        try {
            targetID =
oracle.apps.aia.core.config.Configuration.getServiceProperty(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!");
        }

        // Set the SystemID variable
        try {
            if (targetID!=null && targetID!="")
setVariableData(systemIdBpelVariableName, targetID);
        }
        catch (com.oracle.bpel.client.BPELFault e) {
        }
    }

    // if we have target ID...
    if (targetID!=null && targetID!="") {
        // get EndpointLocation and/or ServiceName from configuration
properties
        java.lang.String endpointPropertyName =
"Routing."+partnerLinkName+"."+targetID+".EndpointURI";
        java.lang.String serviceNamePropertyName =
"Routing."+partnerLinkName+"."+targetID+".ServiceName";
        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!");
            targetEndpointLocation = null;
        }
        try {
            targetServiceName =
oracle.apps.aia.core.config.Configuration.getServiceProperty(serviceName,
serviceNamePropertyName);
            if (addAudits) addAuditTrailEntry("ServiceName = '" +
targetServiceName + "' selected from configuration property " +
serviceNamePropertyName);
            // parse the QName for namespace and service name
            java.util.regex.Matcher m =
serviceQNamePattern.matcher(targetServiceName);
            m.matches();
            targetServiceNamespace = m.group(2)!=null ? m.group(2).trim() :
null;
            targetServiceName = m.group(3)!=null ? m.group(3).trim() : null;
        } catch (oracle.apps.aia.core.config.PropertyNotFoundException e) {
            if (addAudits) addAuditTrailEntry("Configuration property " +
serviceNamePropertyName + " not found!");
            targetServiceName = null;
        }

    }
}
// assign the EndpointReference
org.w3c.dom.Document xmlDoc = new oracle.xml.parser.v2.XMLDocument();
org.w3c.dom.DocumentFragment endpointRef = xmlDoc.createDocumentFragment();
org.w3c.dom.Element endpointRefElem =
xmlDoc.createElementNS("http://schemas.xmlsoap.org/ws/2003/03/addressing",
"wsa:EndpointReference");
if (targetEndpointLocation!=null) {
    org.w3c.dom.Element addressElem =
xmlDoc.createElementNS("http://schemas.xmlsoap.org/ws/2003/03/addressing",
"wsa:Address");
    org.w3c.dom.Text addressText =
xmlDoc.createTextNode(targetEndpointLocation);
    addressElem.appendChild(addressText);
    endpointRefElem.appendChild(addressElem);
}
if (targetServiceName!=null) {
    org.w3c.dom.Element serviceNameElem =
xmlDoc.createElementNS("http://schemas.xmlsoap.org/ws/2003/03/addressing",
"wsa:ServiceName");
    org.w3c.dom.Text serviceNameText = xmlDoc.createTextNode("ns1:" +
targetServiceName);
    serviceNameElem.setAttributeNS("http://www.w3.org/2000/xmlns/",
"xmlns:ns1", targetServiceNamespace);
    serviceNameElem.appendChild(serviceNameText);
    endpointRefElem.appendChild(serviceNameElem);
}
endpointRef.appendChild(endpointRefElem);
try {
    setVariableData("EndpointReference", endpointRef);
} catch (com.oracle.bpel.client.BPELFault e) {}
]]>
        </bpelx:exec>
        <assign name="AssignPartnerlinkEndpointReference">
            <copy>
```

```
                <from variable="EndpointReference"/>
                <to partnerLink="[PartnerlinkName]"/>
            </copy>
        </assign>
    </sequence>
</scope>
```

7. If the PartnerLink is an Enterprise Service Bus (ESB) service (such as an adapter service) and you need to use the ESB binding rather than SOAP, then you must have defined ServiceName rather than EndpointURI configuration properties in step 1.

   The Java code snippet above will automatically handle setting ServiceName rather than EndpointURI at runtime for the dynamic PartnerLink. However, for this to work, the runtime WSDL for the PartnerLink must have all possible services defined, including one for CAVS. Therefore, you need to create a new Ref WSDL in your project to provide this, and set it as the PartnerLink's runtime WSDL in the bpel.xml file.

   Example code is provided below with inline comments for explanation.

   Example Ref.wsdl

```
<?xml version="1.0" encoding="utf-8"?>
<definitions
targetNamespace="http://xmlns.oracle.com/ABCS/Ebiz/Core/ProcessSales
OrderEbizAdapter/V1"
              xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:tns="http://xmlns.oracle.com/ABCS/Ebiz/Core/ProcessSalesOrderE
bizAdapter/V1"
              xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
              xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              xmlns:esb="http://www.oracle.com/esb/">

    <!-- NOTE: the targetNamespace must match the adapter service's
targetNamespace -->


    <!-- Import the concrete wsdl for the adapter service -->
    <import
namespace="http://xmlns.oracle.com/ABCS/Ebiz/Core/ProcessSalesOrderE
bizAdapter/V1"

location="http://<server>:<port>/esb/wsil/AIASystem/Ebiz/ABCS/Proces
sSalesOrderEbizAdapter?wsdl"/>


    <!-- If a customer were to implement an additional application
instance, they would create an additional adapter service for that
instance,
           and would add the additional WSDL import here for that
adapter.  Generally OOTB there is only a single instance adapter, so
this import would
           not be included in the OOTB code.  -->
    <import
```

```
namespace="http://xmlns.oracle.com/ABCS/Ebiz/Core/ProcessSalesOrderE
bizAdapter/V1"

location="http://<server>:<port>/esb/wsil/AIASystem/Ebiz/ABCS/Proces
sSalesOrderEbizAdapter_Instance02?wsdl"/>


    <!-- To maintain CAVS support, you need to define the CAVS
service and SOAP binding… -->

    <!-- IMPORTANT NOTE: the portType named here in the @type
attribute must be the same as that defined in the imported adapter
service wsdl -->
    <binding name="CAVSService_SOAPBinding"
             type="tns:ProcessSalesOrderEbizAdapter_ptt">
        <soap:binding style="document"

transport="http://schemas.xmlsoap.org/soap/http"/>
        <!-- IMPORTANT NOTE: the operation name must be the same as
that defined in the imported adapter service wsdl.  If there are
more than one
                 Operation, then this should be repeated for each
one  -->
        <operation name="ProcessSalesOrder">
            <soap:operation style="document" soapAction="simulate"/>
<!-- soap action should be 'simulate' -->
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>

    <service name="CAVSService">
        <port name="CAVSService_SOAPPort"
              binding="tns:CAVSService_SOAPBinding">
            <soap:address location="http://set.at.runtime"/>
        </port>
    </service>

</definitions>
```

Example bpel.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<BPELSuitcase>
  <BPELProcess src="CreateSalesOrderEbizProvABCSImpl.bpel"
id="CreateSalesOrderEbizProvABCSImpl">
    <partnerLinkBindings>
      …
      <partnerLinkBinding name="ProcessSalesOrderEbizAdapter">
        <property
name="wsdlLocation">http://<server>:<port>/esb/wsil/AIASystem/Ebiz/A
BCS/ProcessSalesOrderEbizAdapter?wsdl</property>
```

```
        <!-- Set the new Ref wsdl as the runtime wsdl for the
partnerlink -->
        <property
name="wsdlRuntimeLocation">ProcessSalesOrderEbizAdapterRef.wsdl</pro
perty>
        <property name="transaction">participate</property>
        <!-- Note: Do NOT set the preferredPort property -->
      </partnerLinkBinding>
      …
    </BPELProcess>
</BPELSuitcase>
```

Note. Provider ABC services that are asynchronous and invoke a callback to a response EBS service must also CAVS-enable this invocation. The steps for CAVS-enabling this invocation are the same as the first four steps described below for CAVS-enabling a requester ABC service.

Developing Requester ABC Services to be CAVS-Enabled and Provide Multi-Instance Support

The following steps describe how to code a requester ABC service for CAVS-enabled invocation of an enterprise business service that is implemented as an ESB routing service.

1. The requester ABC service must have the following service-level configuration properties defined:

```
<Property name="Default.SystemID">[DefaultSenderSystemID]</Property>
<Property
name="Routing.[EBSPartnerlinkName].[EBSOperationName].RouteToCAVS">[
true|false]</Property>
<Property
name="Routing.[EBSPartnerlinkName].[EBSOperationName].CAVS.EndpointU
RI">[CAVSEndpointURL] </Property>
```

2. Include the following XSLT snippet in your application business message (ABM)-to-enterprise business message (EBM) transformation to properly set the MessageProcessingInstruction section of the resulting EBM.

```
<corecom:MessageProcessingInstruction>
    <xsl:variable name="ConfigServiceName"
select="'{[ABCServiceNamespace]}[ABCServiceName]'"/>
    <xsl:variable name="RouteToCAVS"
select="aia:getServiceProperty($ConfigServiceName,
'Routing.[EBSPartnerlinkName].[EBSOperationName].RouteToCAVS',
false())='true'"/>
    <corecom:EnvironmentCode>
        <xsl:choose>
            <xsl:when test="$RouteToCAVS">
                <xsl:text>CAVS</xsl:text>
            </xsl:when>
            <xsl:otherwise>
                <xsl:variable name="EnvCode"
select="aia:getServiceProperty($ConfigServiceName,
'Routing.[EBSPartnerlinkName].[EBSOperationName].MessageProcessingIn
struction.EnvironmentCode', false())"/>
                <xsl:choose>
                    <xsl:when test="$EnvCode!=''">
                        <xsl:value-of select="$EnvCode"/>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:text>PRODUCTION</xsl:text>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:otherwise>
        </xsl:choose>
    </corecom:EnvironmentCode>
    <xsl:if test="$RouteToCAVS">
        <corecom:DefinitionID>
            <xsl:value-of
select="aia:getServiceProperty($ConfigServiceName,
'Routing.[EBSPartnerlinkName].[EBSOperationName].CAVS.EndpointURI',
false())"/>
        </corecom:DefinitionID>
    </xsl:if>
    <xsl:call-template name="MessageProcessingInstructionType_ext"/>
</corecom:MessageProcessingInstruction>
```

3. Add a routing rule to the target enterprise business service (EBS) operation to route to CAVS. The XPath filter for the CAVS routing rule should be equivalent to this:

```
/*/corecom:EBMHeader/corecom:MessageProcessingInstruction/corecom:En
vironmentCode = 'CAVS'
```

4. Add the SetCAVSEndpoint.xsl file to the target EBS ESB project, and associate the transformation to the CAVS routing rule.

Some requester ABC services will need to communicate back directly with the calling participating application. For this type of PartnerLink, the requester ABC service is acting like a provider ABC service in that it is invoking a participating application web service.

Perform steps 1, 2, 3, and 6 in Developing Provider ABC Services to be CAVS-Enabled and Provide Multi-Instance Support to set up the dynamic CAVS-enabled PartnerLink for this invocation.

Developing EBFs to be CAVS-Enabled

Enterprise business flows (EBFs) are similar to requester ABC services in that they invoke EBSs. To CAVS-enable each of these EBS invocations and EBS callbacks, perform steps 1, 2, 3, and 4 in Developing Requester ABC Services to be CAVS-Enabled and Provide Multi-Instance Support.

Describing CAVSEndpointURL Value Designation

The CAVSEndPointURL value will be set at design time as:

- If the ABC service or EBF is invoking a synchronous service, then the CAVSEndPointURL value in the AIAConfigurationProperties.xml file will be set to a default value of:

http://<host>:<port>/AIAValidationSystemServlet/syncresponsesimulator

- If the ABC service/EBF is invoking a Asynchronous one-way service, then the CAVSEndPointURL value in the AIAConfigurationProperties.xml file will be set to a default value of: http://<host>:<port>/AIAValidationSystemServlet/asyncrequestrecipient
- If the ABC service/EBF is invoking a Asynchronous Request-delayed response service, then the CAVSEndPointURL value in the AIAConfigurationProperties.xml file will be set to a default value of: http://<host>:<port>/AIAValidationSystemServlet/asyncresponsesimulator
- If the ABC service/EBF is invoking a Response EBS, then the CAVSEndPointURL value in the AIAConfigurationProperties.xml file will be set to a default value of: http://<host>:<port>/AIAValidationSystemServlet/asyncresponserecipient

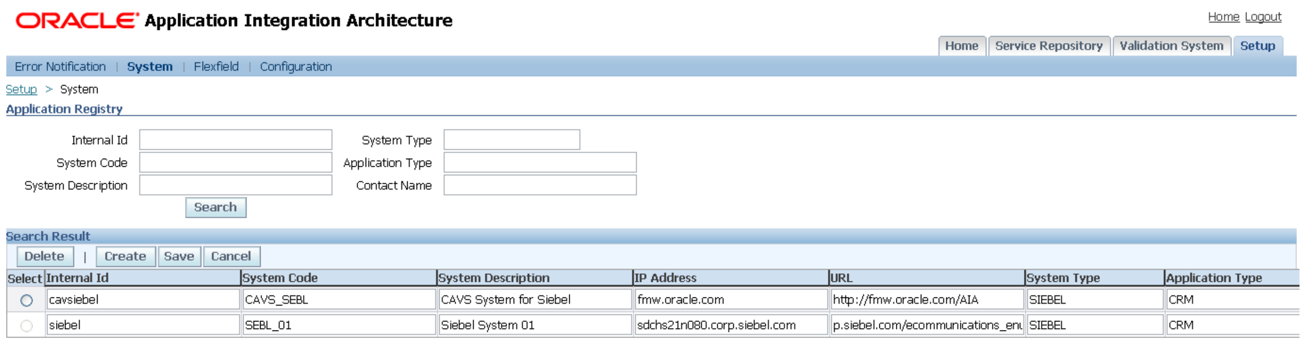
Purging CAVS-Related Cross Reference Entries to Enable Rerunning of Test Scenarios

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 once the test scenario has been executed. This enables rerunning of the test scenario.

This purge is accomplished as:

1. Process integration packs (PIPs) that are delivered to work with Oracle AIA Foundation Packs are delivered with cross-reference systems in place. They are named *CAVS_<XYZ>,* where *<XYZ>* is the participating application system. For example, for systems EBIZ and SEBL, the PIP is delivered with cross-reference systems *CAVS_EBIZ* and *CAVS_SEBL.*
2. For every system type defined on the 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.



Internal Id	
System Code	
System Description	
System Type	
Application Type	
Contact Name	

Select	Internal Id	System Code	System Description	IP Address	URL	System Type	Application Type
○	cavsiebel	CAVS_SEBL	CAVS System for Siebel	fmw.oracle.com	http://fmw.oracle.com/AIA	SIEBEL	CRM
○	siebel	SEBL_01	Siebel System 01	sdchs21n080.corp.siebel.com	p.siebel.com/ecommunications_enu	SIEBEL	CRM

System and CAVS-related system entries on the System – Application Registry page

For more information about the System – Application Registry page, see *Oracle Application Integration Architecture Foundation Pack Core Infrastructure Components Guide,* “Managing the Oracle AIA Application Registry.”

3. When testing a provider ABC service in isolation, the EBM will be passed from the CAVS to the provider ABC service with the NamespacePrefixedEBMName/EBMHeader/Target/ID element set as *CAVS_<XYZ>*.
4. When testing a requester ABC service in isolation, the element in the application business message (ABM) that normally contains the Internal ID value will now contain the CAVS-specific Internal ID value set for the system on the System – Application Registry page.
5. When testing an entire flow (requester ABC service-to-EBS-to-provider ABC service), you must set the Default.SystemID property of the provider ABC service to *CAVS_<XYZ>,* where *<XYZ>* is the system.
    a. To do this, edit the Default.SystemID property value in the AIAConfigurationProperties.xml file in the AIA_HOME/config directory.
    b. Access the Configuration page and click Reload to load the edit.
    c. You can now commence testing the entire flow.

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

Chapter 13: Configuring Oracle AIA Processes for Error Handling and Trace Logging

This chapter discusses how to:

- Configure BPEL processes to adhere to error handling requirements.
- Configure Enterprise Service Bus (ESB) processes to adhere to error handling requirements.
- Perform common configurations for BPEL and ESB processes to adhere to error handling requirements.
- Describe the Oracle AIA fault message schema.
- Describe the Oracle AIA ESB fault policy schema.
- Extend fault messages.
- Extend error handling.
- Configure Oracle AIA processes for trace logging.

Configuring BPEL Processes to Adhere to Error Handling Requirements

In this section, we provide an overview of BPEL error handling and discuss how to:

- Implement error handling for partner link errors.
- Implement error handling for non-partner link errors.

Understanding BPEL Error Handling

The Oracle Application Integration Architecture (AIA) Error Handling Framework groups BPEL process errors into two categories:

- Partner link errors.
- Non-partner link errors.

Partner Link Errors

BPEL fault policies kick-in when there is a partner link error. A partner link error can occur in one of two scenarios:

- The partner link invocation fails.
- The partner link receives a fault.

BPEL fault policies can be specified at the domain level, process level, or partner link level.

If there is no fault policy defined at the partner link level or process level, the domain level fault policy becomes active.

A default domain-wide fault policy is provided as a part of the Error Handling Framework deployment. This domain-level policy calls the BPELJavaAction, oracle.apps.aia.core.eh.BPELJavaAction, to perform error notifications and error logging. Once the BPELJavaAction completes, the domain-level policy rethrows the fault so that execution control returns to the catch/catch-all block defined in the BPEL process.

Implementing Error Handling for Partner Link Errors

To implement error handling for partner link errors:

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 (ABC) service, 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 will be able to access and use the EBM header details.

2. **Define a fault policy for the BPEL process and bind the process with this policy in bpel.xml.**

   Always use the BPELJavaAction, oracle.apps.aia.core.eh.BPELJavaAction, as specified in the default policy. Including this BPELJavaAction, accounts for error notifications and error logging.

3. **Define the catch blocks.**

   The default behavior of the fault policy after the BPELJavaAction is to do a rethrow. This returns the execution control to the catch or catch-all block specified in the BPEL process. Note that you must catch the fault using the same method you would use if the fault policy did not exist. In other words, you must catch the exact fault, such as a binding or remote fault that the partner link raised. Define a catch block for each partner link 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 executing the process.

Guidelines for BPEL Fault Policies

Each BPEL process should have a specific fault policy defined. The fault policy should intercept Oracle AIA faults and invoke the ora-rethrow-fault action to avoid duplicate notifications. For all other faults, it should invoke the aia-ora-java action.

To define a fault policy to intercept Oracle AIA faults:

1. Examine your partner link WSDL and check to see if 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 illustrated here:

```
<Conditions>
     <faultName xmlns:telcocustmrebs=" <partnerlink namespace>"
      name="telcocustmrebs: <name of the fault in the partner link
      wsdl>">
          <condition>
               <action ref="ora-rethrow-fault"/>
          </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. For example:

```
<Conditions>
     <faultName xmlns:telcocustmrebs="http://xmlns.oracle.com/
      EnterpriseServices /Industry/Telco/CustomerParty/V1"
      name="telcocustmrebs:fault">
          <condition>
               <action ref="ora-rethrow-fault"/>
          </condition>
     </faultName>
     <faultName>
          <condition>
               <action ref="aia-ora-java"/>
          </condition>
     </faultName>
</Conditions>
```

Guidelines for BPEL Catch and Catch-All Blocks

Each BPEL process should have explicit catch blocks for remote faults, binding faults, Oracle AIA faults, and any other fault expected on a partner link at design time. Use these guidelines for defining these catch blocks:

- In the case of an asynchronous process, construct an Oracle AIA fault message and reply.
- Do a rethrow of the fault that has been caught. This rethrow enables the process to appear as faulted in the Oracle BPEL Console.

Each BPEL process should have a catch-all block.

To define this catch-all block:

1. Construct an Oracle AIA fault message.
2. Invoke the AIAAsyncErrorHandlingBPELProcess with this Oracle AIA fault message as input.
3. If a reply is needed, send this Oracle AIA fault message as the reply.
4. Do a rethrow of the fault that has been caught.

   This rethrow will enable the process to appear as faulted in the Oracle BPEL Console.

Unless otherwise required, these catch and catch-all blocks can be defined at the top-level scope and do not need to be defined at the scope for each partner link.

For more information, see Fault Management Framework in *Oracle SOA Suite 10.1.3.3.0 New Features TechNote:*

Implementing Error Handling for Non-Partner Link Errors

BPEL fault policies are not invoked for non-partner link errors.

To handle non-partner link errors:

1. Construct an Oracle AIA fault message for these errors.
2. Throw this fault as a named fault and catch it in a catch block.
3. Invoke the AIAAsyncErrorHandlingBPELProcess with this Oracle AIA fault message as input.
4. If a reply is needed, send this Oracle AIA fault message as the reply.
5. Do a rethrow of the fault that has been caught.

   This rethrow will enable the process to appear as faulted in the Oracle BPEL Console.

For more information, see *Oracle BPEL Process Manager Developer's Guide.*

Configuring ESB Processes to Adhere to Error Handling Requirements

In this section, we provide an overview of ESB error handling and discuss how to configure ESB fault policies.

Understanding ESB Error Handling

The Error Handling Framework groups ESB process errors into two categories:

- Retryable.
- Non-retryable.

Retryable Errors

Retryable errors are resubmitted programmatically using the retry action set for the failed ESB service in the esb-fault-policy.xml file.

In the esb-fault-policy.xml file, each ESB process can have a set of conditions defined, to which you can associate retry actions. The fault policy is used to derive the retry action for the failed condition of the process. After resubmitting the error instance for the specified retry count value, appropriate action is taken based on the success or failure of the retry.

In the case of a successful retry, a trace message is logged to the Oracle AIA trace logger. In the case of a failed retry, the Error Handling Framework performs appropriate error notifications and error logging.

Non-Retryable Errors

If an EBS error is non-retryable, the Error Handling Framework performs appropriate error notifications and error logging and no further configurations are required. The Oracle Fusion Middleware defines the concept of a retryable or non-retryable ESB error.

For more information, see *Oracle Enterprise Service Bus Developer's Guide.*

Configuring ESB Fault Policies

ESB fault policies are configured in the esb-fault-policy.xml file located in <aia.home>/Config/. Use this file to set the retry options for a service. If the service-level retry options are not set, the default retry options are used. If a service has an entry in this policy file and no condition is satisfied for that service, then no retry action is taken.

Note: To avoid unnecessary processing, ensure that you specify retry options only when explicitly required. In addition, if the ESB service does not need to explicitly retry an error, there is no need to specify the service in the esb-fault-policy.xml file.

Default retry options are set as:

1. Default Retry Count: 0

   Indicates the number of times a failed service will be retried.

2. Default Retry Interval: 0

   Indicates the wait interval time.

3. Exponential Back-Off: 0

   Indicates if the process will be tried with an exponential wait factor.

Service-level retry options are:

1. Process Name: <ESB Service Qualified Name (QName)>

   Uses the format SystemID.ServiceName.

2. Test Failure Condition: <Any XPath function that can be evaluated on the error text>

   For example, contains ("Error occurred in transformation") or contains ("null pointer").

3. Retry Action: ora-retry

   This is the only action available for ESB process errors.

4. Retry count: <integer>

   Indicates the number of times a failed transaction will be retried.

5. Retry Interval: <integer>

   Indicates the wait interval time.

6. Exponential Back-Off: 0 or 1

   Indicates if the process will be tried with an exponential wait factor.

Performing Common Configurations for Both BPEL and ESB Processes to Adhere to Error Handling Requirements

This section discusses the standard configuration steps that need to be performed whether you are handling a BPEL or ESB error.

In this section, we discuss how to:

- Define notification roles.
- Define corrective action codes.
- Define error message codes.

Defining Notification Roles for Each Service

You can define Actor and FYI notification roles for each service in the BSR ERROR NOTIFICATIONS table on the Error Notifications page.

For more information, see *Oracle Application Integration Architecture Core Infrastructure Components Guide,* "Setting Up Error Notifications and Logging," Setting Up Error Notifications for Oracle AIA Processes.

This table contains these columns of particular interest:

- SYSTEM_ID

  The system ID value is obtained from the EBM header.

- ERROR CODE

  In the case of BPEL, the error code is the fault code. In the case of ESB, the error code is ESBError string, as ESB does not have the concept of error codes.

- SERVICE NAME

  The service name is the name of the BPEL process or ESB service QName that encountered an error.

- PROCESS NAME

  The process name value is obtained from the EBM header.

- NOTIFICATION ROLE
- FYI NOTIFICATION ROLE

The default values for Actor and FYI roles are specified in the AIAConfigurationProperties.xml file located here: <aia.home>/config/.

```
<Property
name="EH.DEFAULT.ACTOR.ROLE">AIAIntegrationAdmin</Property>
<Property name="EH.DEFAULT.FYI.ROLE"></Property>
```

Note: For a given process, if no entry is found in the BSR_ERROR_NOTIFICATIONS table, the default roles specified in AIAConfigurationProperties.xml are used. Therefore, you are not required to populate the BSR_ERROR_NOTIFICATIONS table unless there is an explicit need.

The logic used to determine the appropriate notification roles and corrective action for an error is provided here.

- If these four input values; SYSTEM_ID, ERROR CODE, SERVICE NAME, and PROCESS NAME; are available and map to a specific entry in the BSR_ERROR_NOTIFICATIONS table, use it.
- If ERROR_CODE, SERVICE_NAME, and PROCESS_NAME are available and map to an entry in the table, use it.

- If SERVICE_NAME and PROCESS_NAME are available and map to an entry in the table, use it.
- If SERVICE_NAME is available and maps to an entry in the table, use it.
- If none of these values are available, the default values are fetched from the AIAConfigurationProperties.xml file.

Defining Corrective Action Codes

For non-partner link errors, define corrective action codes and details in the oracle.apps.aia.core.eh.i18n.EHCorrectiveActionRB class. This is a ListResourceBundle and contains the key-value pairs.

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

  This is a concatenated string of language code, country code, and variant. For example, en-US-WIN.

- Delimiter

  The delimiter used in Locale parameter, such as -.

Defining Error Message Codes

For non-partner link errors, define error message codes and details in the oracle.apps.aia.core.eh.i18n.EHResourceBundle class. This is a ListResourceBundle and contains the key-value pairs.

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

  This is a concatenated string of language code, country code, and variant. For example, en-US-WIN.

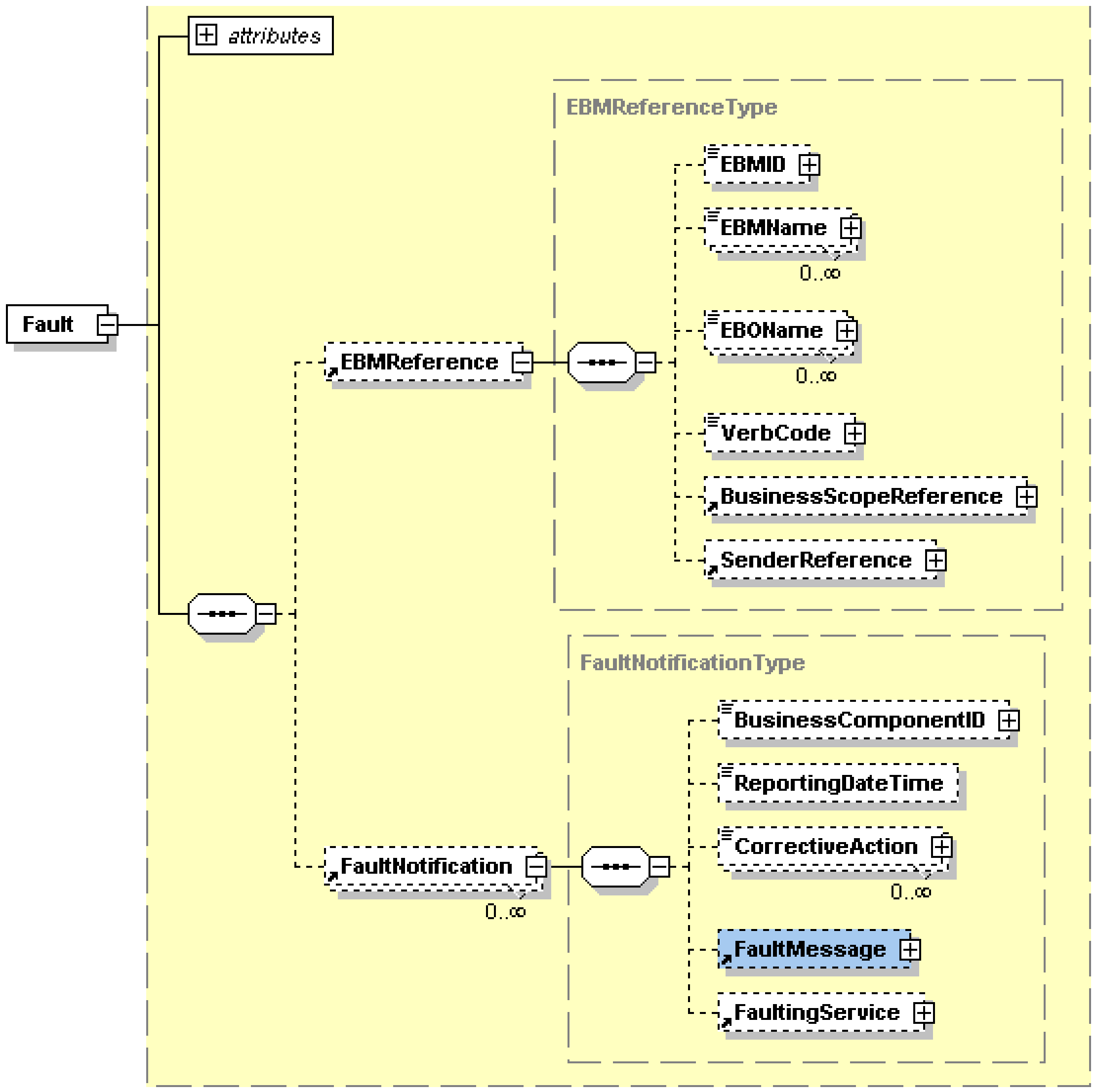
- Delimiter

  The delimiter used in Locale parameter, such as -.

Describing the Oracle AIA Fault Message Schema

This section provides details about the Oracle AIA fault message schema.

The top-level element of this schema is Fault. It has two elements, EBMReference and FaultNotification.



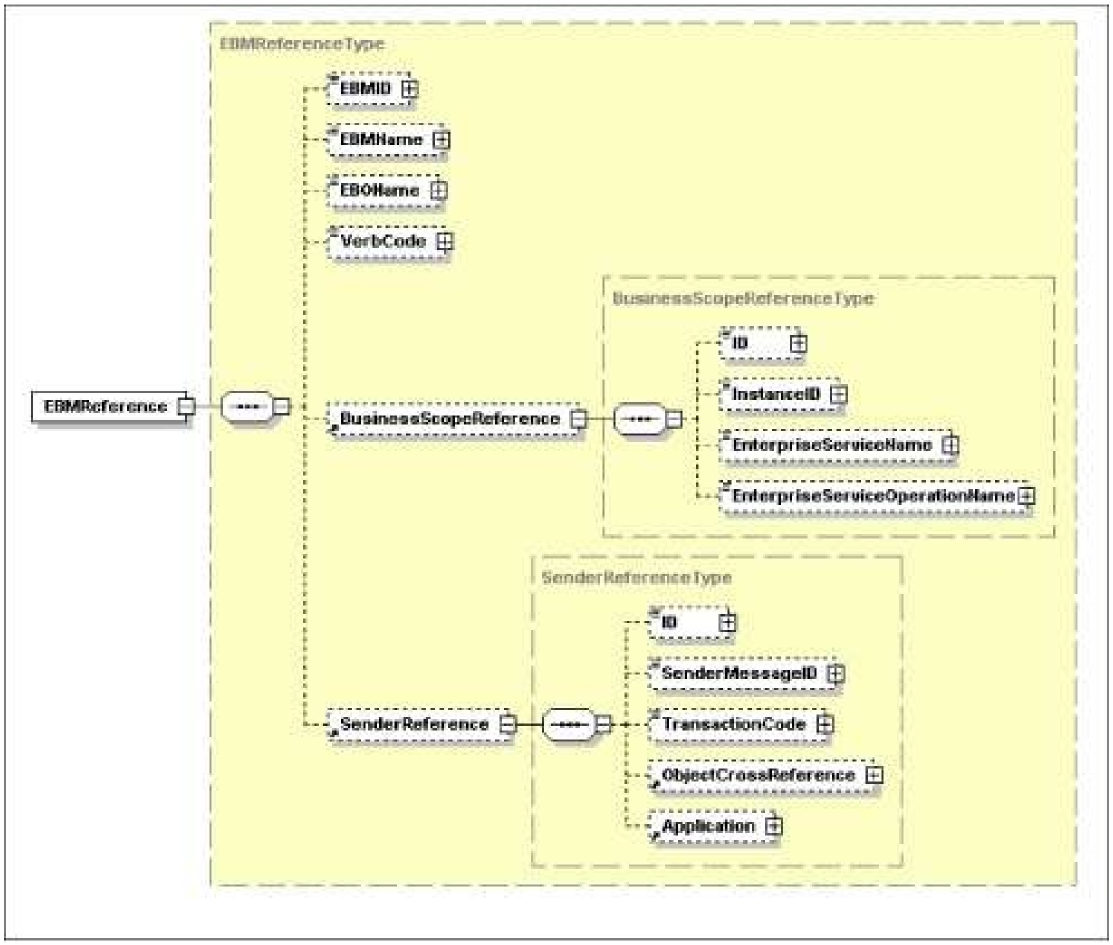
Fault element and its child elements

Name	Purpose	Details
EBMReference	Provides contextual information about the fault instance. All values are taken from the EBM header of the EBM in a	**For more information**, see Describing the EBMReference

Name	Purpose	Details
	faulted service instance.	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.



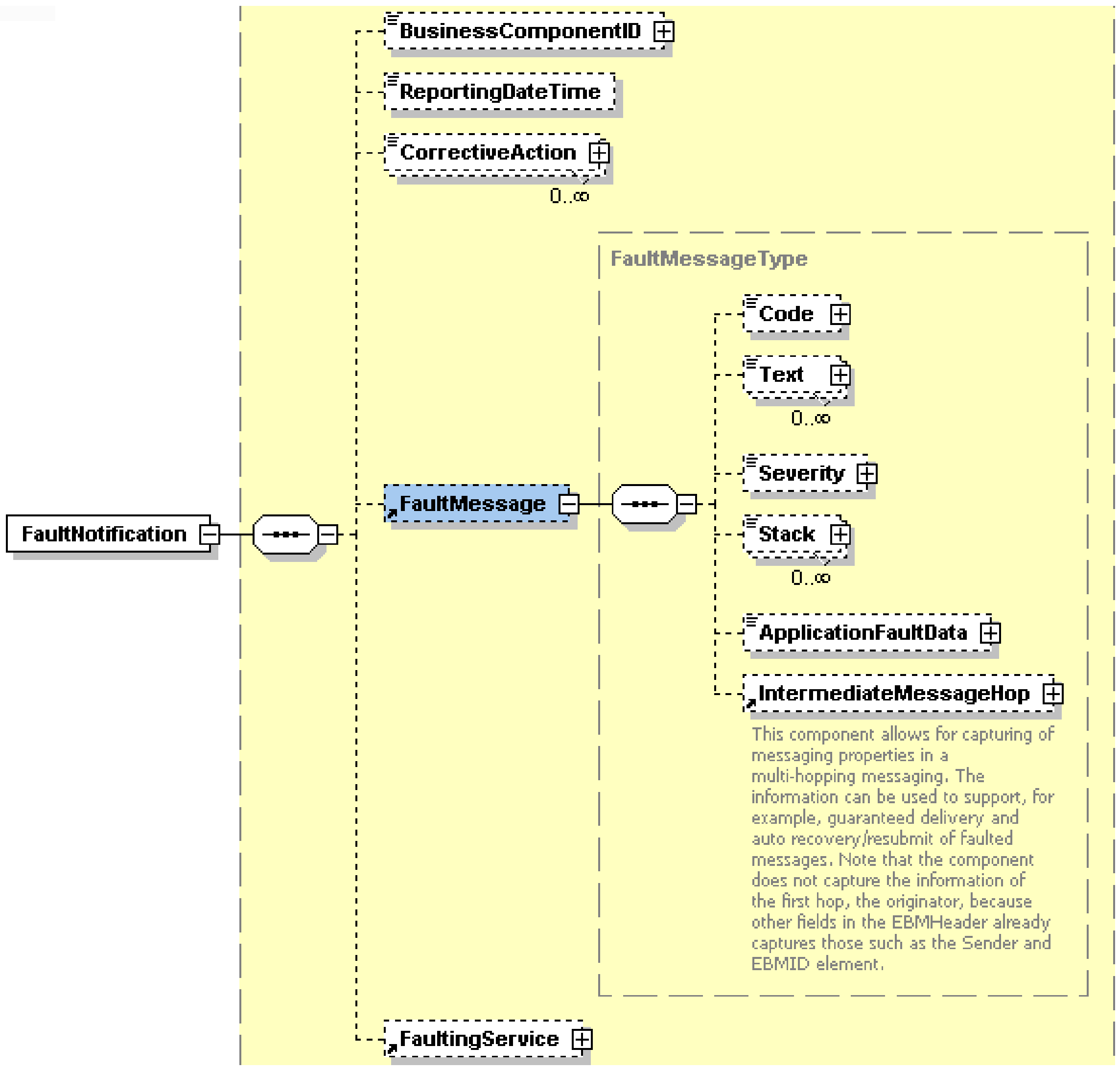
EBMReference element and its child elements

Name	Purpose	Details
EBMID	Provides the EBMID in the message.	**For more information**, see Chapter 9: Working with EBM Headers.

Name	Purpose	Details
EBMName	Provides the EBMName in the message.	**For more information**, see Chapter 9: Working with EBM Headers.
EBOName	Provides the EBOName in the message.	**For more information**, see Chapter 9: Working with EBM Headers.
VerbCode	Provides the VerbCode in the message.	**For more information**, see Chapter 9: Working with EBM Headers.
BusinessScopeReference	Provides the BusinessScopeReference in the message. Provides details about the end-to-end scenario in which the faulted service instance was participating. **Note:** This is the instance of the BusinessScopeReference where BusinessScopeTypeCode is equal to BusinessProcess.	**For more information**, see Chapter 9: Working with EBM Headers.
SenderReference	Provides the SenderReference in the message.	**For more information**, see Chapter 9: Working with EBM Headers.

Describing the FaultNotification Element

This section provides details about the FaultNotification element in the Oracle AIA fault message schema.

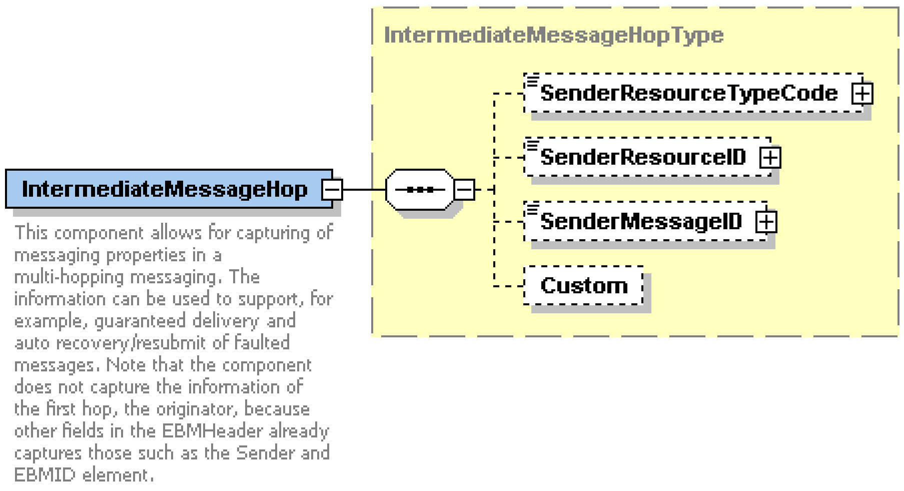

BusinessComponentID
ReportingDateTime
CorrectiveAction
0..∞
FaultMessageType
Code
Text
0..∞
Severity
Stack
0..∞
ApplicationFaultData
IntermediateMessageHop
This component allows for capturing of messaging properties in a multi-hopping messaging. The information can be used to support, for example, guaranteed delivery and auto recovery/resubmit of faulted messages. Note that the component does not capture the information of the first hop, the originator, because other fields in the EBMHeader already captures those such as the Sender and EBMID element.
FaultMessage
FaultNotification
FaultingService

FaultNotification element and its child elements

Name	Purpose	Details
BusinessComponentID	Unique key for the application.	Provides an agnostic representation of the object instance.
ReportingDateTime	Provides the date and time at which the service faulted.	The date and time at which the service faulted.
CorrectiveAction	Provides the possible corrective action for the fault.	The corrective action for the fault.
FaultMessage	Provides details of the actual fault message	See the FaultMessage Element section.
FaultingService	Provides details of the faulting service.	See the FaultingService Element section.

FaultMessage Element

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 Chapter 14: Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery.


IntermediateMessageHopType
SenderResourceTypeCode
SenderResourceID
SenderMessageID
Custom
IntermediateMessageHop
This component allows for capturing of messaging properties in a multi-hopping messaging. The information can be used to support, for example, guaranteed delivery and auto recovery/resubmit of faulted messages. Note that the component does not capture the information of the first hop, the originator, because other fields in the EBMHeader already captures those such as the Sender and EBMID element.

IntermediateMessageHop element

IntermediateMessageHop Element

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.

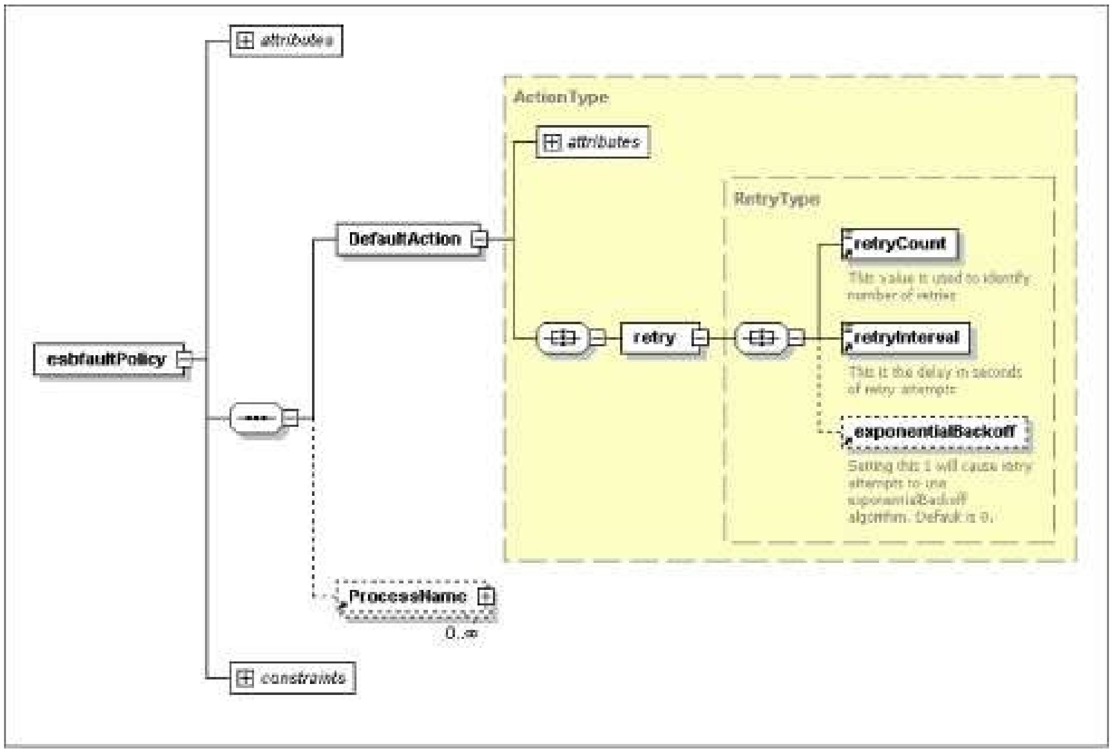
FaultingService Element

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, ESB, 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. IF the service is an ESB service, this is the ESB flow ID.

Describing the Oracle AIA ESB Fault Policy Schema

This section provides details about the Oracle AIA ESB fault policy schema.

The Oracle AIA ESB fault policy schema has a default section and one section for each service. The top-level element is esbfaultPolicy. Only retry actions are supported.

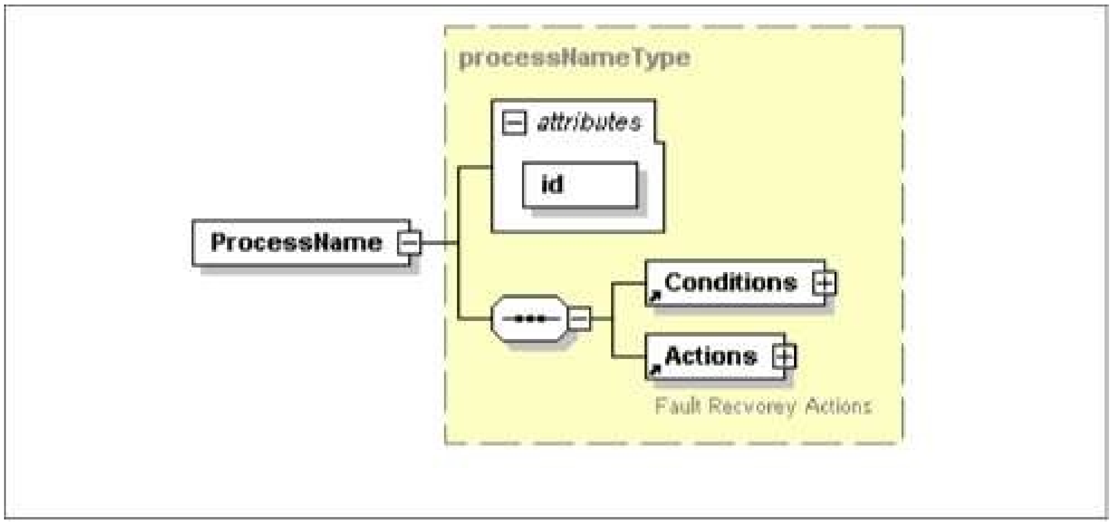

attributes
ActionType
attributes
RetryType
DefaultAction
retryCount
This value is used to identify number of retries
esbfaultPolicy
retry
retryInterval
This is the delay in seconds of retry attempts
exponentialBackoff
Setting this 1 will cause retry attempts to use exponentialBackoff algorithm. Default is 0.
ProcessName
0..∞
constraints

esbfaultPolicy element and its child elements

Name	Purpose	Details
DefaultAction	If the service does not have its own entry, provides the default options to be used.	The default retryCount, retryInterval, and exponentialBackOffoptions.
ProcessName	Provides service-specific options.	**For more information**, see Describing the ProcessName Element.

Describing the ProcessName Element

This section provides details about the ProcessName element in the Oracle AIA ESB fault policy.

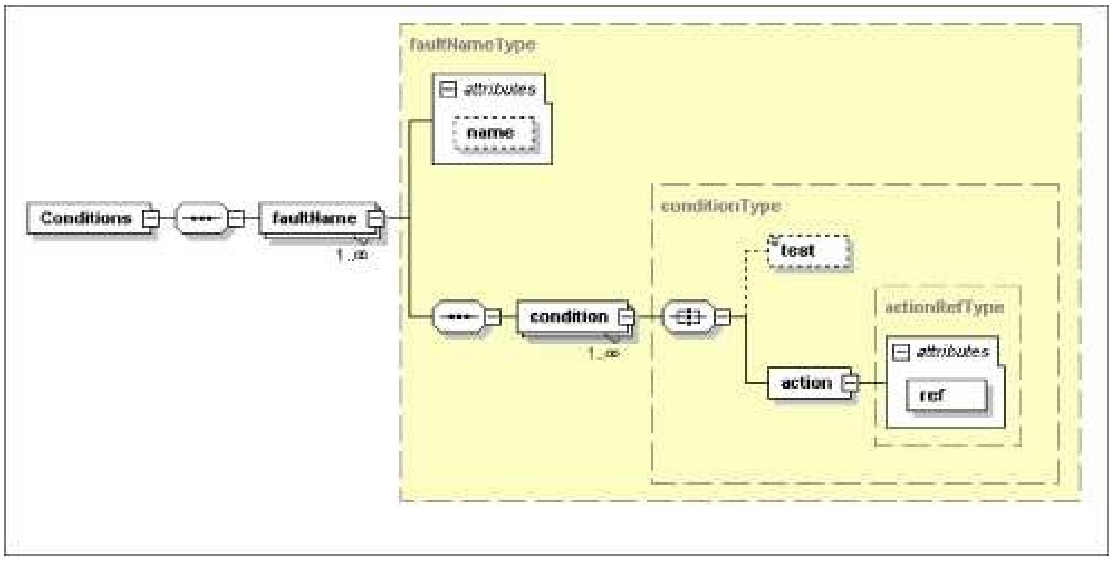


ProcessName element in the Oracle AIA ESB fault policy

Name	Purpose	Details
Id	Provides the actual ESB service name.	This should be unique in the policy file and identifies the faulted service.
Conditions	Provides the conditions to match the fault instance.	
Actions	Provides the retry actions for the service.	

Conditions Element

This section provides details about the Conditions element in the Oracle AIA ESB fault policy.

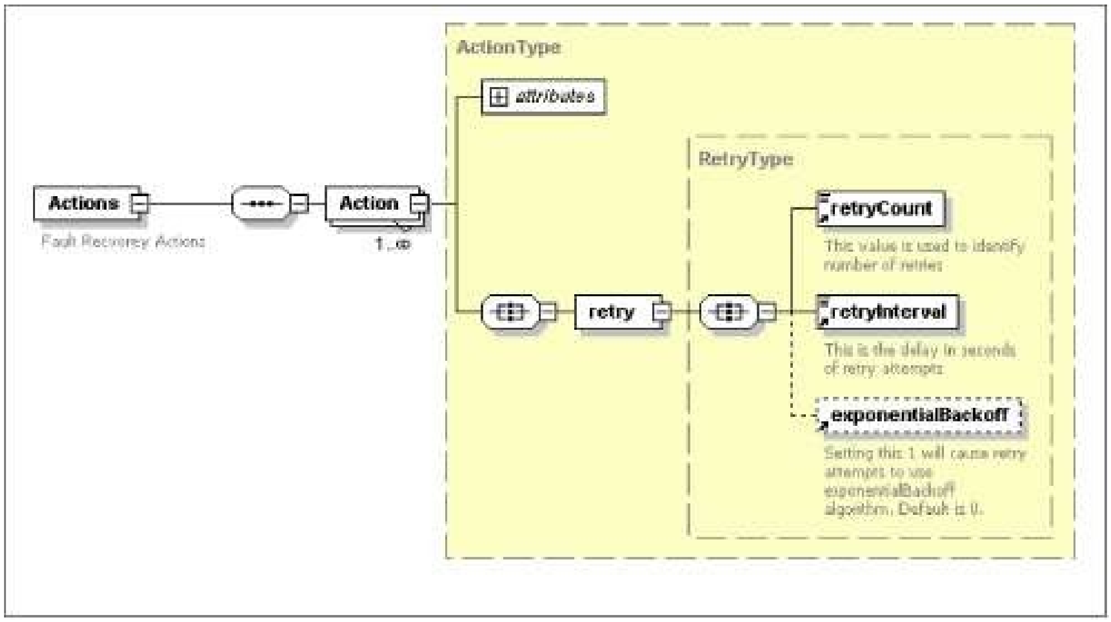


Conditions element in the Oracle AIA ESB fault policy

Name	Purpose	Details
FaultName	Provides the name of the fault.	The name attribute should always be env:Server. All ESB faults are identified as env:server. It can have one or more condition elements.
Condition	Provides maps of a certain fault condition to an action reference.	This has an optional test element and an action reference. If no test element is specified, the action reference will be used as default for the specific process.
Test	Provides an XPath expression to evaluate the fault condition.	This could contain any XPath expression that will be evaluated on the fault message created by the ESB in the case of an error.
Action	Provides a reference to an action for this process.	The ref attribute should match the ID of an action specified for this process.

Actions Element

This section provides details about the Actions element in the Oracle AIA ESB fault policy.


ActionType
attributes
RetryType
Actions
Fault Recovery Actions
Action
1..∞
retryCount
This value is used to identify number of retries
retry
retryInterval
This is the delay in seconds of retry attempts
exponentialBackoff
Setting this 1 will cause retry attempts to use exponentialBackoff algorithm. Default is 0.

Actions element in the Oracle AIA ESB fault policy

The Actions element can have one or more Action elements. The ID attribute of an action uniquely identifies the Action. The retry element stores the retry count, retry interval, and exponential back-off settings.

Extending Fault Messages

This section provides an overview of fault message extension and discusses how to extend fault messages.

Overview

When an error occurs within an integration flow, within an ESB service or BPEL process, the Error Handling framework captures the error within a fault message. This fault message is made available in the error details within the Error Console.

For more information about the Error Console, see *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* "Using the Error Console."

Fault message content is defined by the FaultType message schema definition in Meta.xsd, which is located in \EnterpriseObjects\Release2\Core\Common\V2\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.

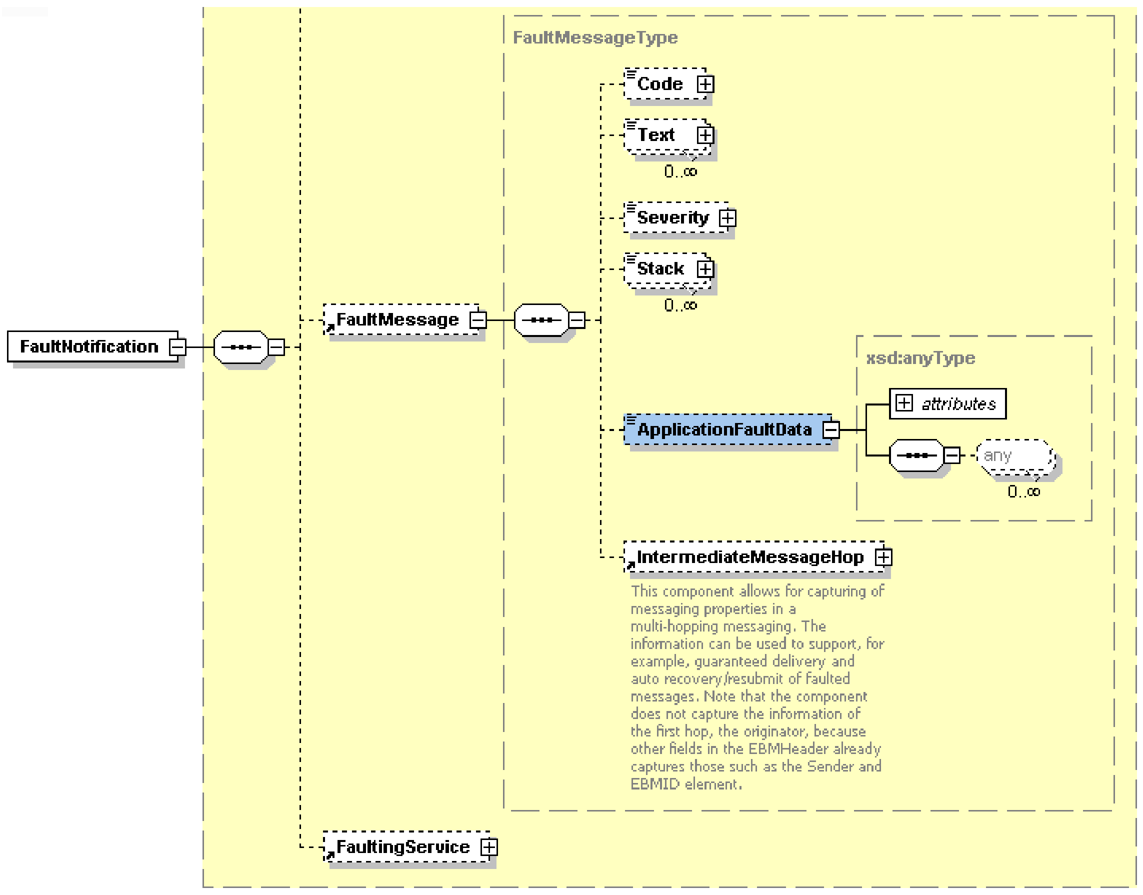
Extending fault details can add functionally rich information to the fault message to help the integration flow consumer better understand the context of the fault, leading to more effective error resolution. These additional fault details can be used to enable extended error handling functionality as well.

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

For example, you may 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 utilizes the ApplicationFaultData element of type ***xsd:anyType*** in the FaultType message schema definition in Meta.xsd.


FaultMessageType
Code
Text
0..∞
Severity
Stack
0..∞
FaultNotification
FaultMessage
xsd:anyType
attributes
ApplicationFaultData
any
0..∞
IntermediateMessageHop
This component allows for capturing of messaging properties in a multi-hopping messaging. The information can be used to support, for example, guaranteed delivery and auto recovery/resubmit of faulted messages. Note that the component does not capture the information of the first hop, the originator, because other fields in the EBMHeader already captures those such as the Sender and EBMID element.
FaultingService

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 PartnerLink, BPEL custom, and ESB errors.

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 Ext 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.

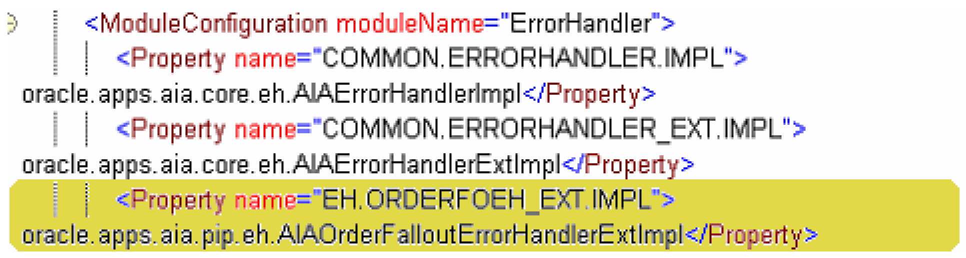
   Based on the combination of error code, system code, service name, and process name parameters, the Error Handling framework checks to see if the error extension handler has a non-default parameter defined.

   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.

   It is within this error extension handler that 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 *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* "Setting Up Error Notifications for Oracle AIA Processes," Setting Up Error Notification Details for Oracle AIA Processes.

3. Access the AIAConfigurationProperties.xml file in <AIA.HOME>/CONFIG/. Define a property name that matches the error extension handler name you defined in step 2. The value for this property is the fully qualified class path of the handler.



```
<ModuleConfiguration moduleName="ErrorHandler">
    <Property name="COMMON.ERRORHANDLER.IMPL">
oracle.apps.aia.core.eh.AIAErrorHandlerImpl</Property>
    <Property name="COMMON.ERRORHANDLER_EXT.IMPL">
oracle.apps.aia.core.eh.AIAErrorHandlerExtImpl</Property>
    <Property name="EH.ORDERFOEH_EXT.IMPL">
oracle.apps.aia.pip.eh.AIAOrderFalloutErrorHandlerExtImpl</Property>
```

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:

   - handleBPELSystemError for BPEL PartnerLink errors
   - handleBusinessError for BPEL custom errors
   - handleESBSystemError for ESB errors

Following is the interface structure:

```
package oracle.apps.aia.core.eh;

import com.oracle.bpel.client.config.faultpolicy.IFaultRecoveryContext;
import org.w3c.dom.Element;

public interface IAIAErrorHandlerExtension
{
    /**
     *
     * @param iFaultRecoveryContext
     * @param faultMessageConstructed
     * @param inputPayLoad - this will be populated provided the variable name is "INPUT_PAYLOAD" in the developed
process.
     * @return
     */
    public String handleBPELSystemError( IFaultRecoveryContext iFaultRecoveryContext, String faultMessageConstructed,
Element inputPayLoad );

    /**
     *
     * @param inputPayLoad
     * @param faultMessageConstructed
     * @return
     */
    public String handleESBSystemError( String inputPayLoad, String faultMessageConstructed );

    /**
     *
     * @param faultMessageConstructed
     * @return
     */
    public String handleBusinessError( String faultMessageConstructed );
}
```

Interface structure to be implemented by the error extension handler

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

For more information about viewing error logs in Oracle Enterprise Manager, see *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* "Enabling and Viewing Trace and Error Logs."

Extending Error Handling

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

Overview

The default error handling behavior for BPEL PartnerLink errors, BPEL business errors, and ESB 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 Error Console.

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, you may also need to 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 a value against Error Type for an error code, system code, process name, and service name value combination.

   For more information about the Error Notifications page, see *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* “Setting Up Error Notifications for Oracle AIA Processes,” Setting Up Error Notification Details for Oracle AIA Processes.

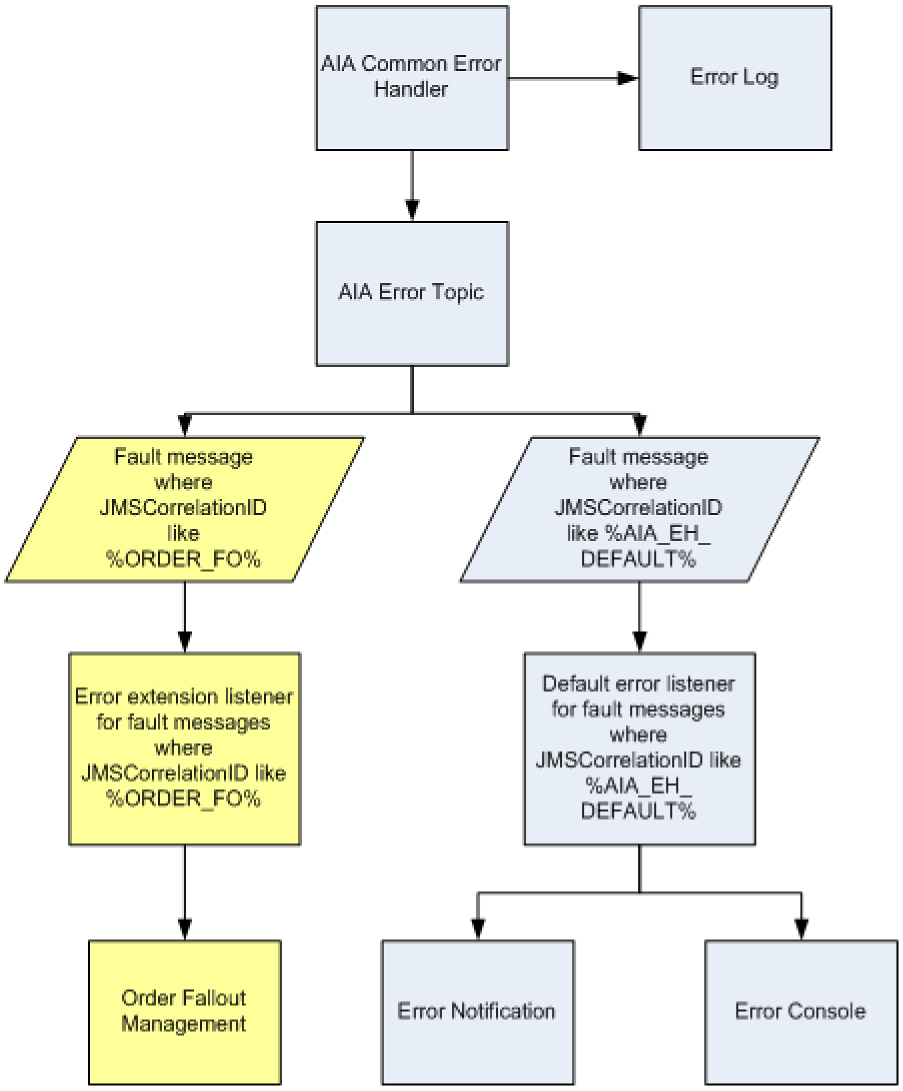
   The Error Handling Framework uses the Error Type value to stamp the JMSCorrelationID JMS header. The JMSCorrelationID is used by the custom error listener to identify fault messages that require its custom error handling.

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 will pick up fault message with the JMSCorrelationID value of ***ORDER_FO***. ***ORDER_FO*** is the Error Type value you defined on the Error Notifications page in step 1.

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.


AIA Common Error Handler
Error Log
AIA Error Topic
Fault message where JMSCorrelationID like %ORDER_FO%
Fault message where JMSCorrelationID like %AIA_EH_ DEFAULT%
Error extension listener for fault messages where JMSCorrelationID like %ORDER_FO%
Default error listener for fault messages where JMSCorrelationID like %AIA_EH_ DEFAULT%
Order Fallout Management
Error Notification
Error Console

Sample extended error handling flow alongside a default error handling flow

Configuring Oracle AIA Processes for Trace Logging

The custom XPath trace logging functions are available to BPEL and ESB services operating in an Oracle AIA ecosystem.

- Determines whether trace logging is enabled for the service or at the overall system level:

```
aia:isTraceLoggingEnabled(String logLevel, String processName)
```

- Generates the actual trace log:

  ```
  aia:logTraceMessage(String  level, Element ebmHeader, String
  message)
  ```

When developing a BPEL or ESB 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.

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 will be 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.

Chapter 14: Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery

This chapter provides an overview of implementing error handling and recovery for the asynchronous message exchange pattern (MEP) to ensure guaranteed message delivery and discusses how to:

- Configure milestones.
- Configure services between milestones.
- Use the Message Resubmission Utility API.

Overview

In the context of the Oracle Application Integration Architecture (AIA), guaranteed message delivery for the asynchronous MEP means that the message initiated from a sender is persisted until it is successfully delivered to and acknowledged by the receiver, if acknowledgement is expected.

The sender and receiver are not necessarily the participating applications. Rather, they are logical milestones in an Oracle AIA integration scenario. There could be multiple milestones 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.

Once 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 *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* “Using the Message Resubmission Utility.”

These points summarize primary aspects of an implementation of the guaranteed message delivery programming model for the asynchronous message exchange pattern.

- 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 target. The source and target may be applications. Each persistence store represents a milestone and may be a database, file system, JMS queue, JMS topic, or Oracle Enterprise Service Bus (ESB) Resequencer store.

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 of the message from the source.
  - Processing of the message by one or more services.
  - Delivery of 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, once the exception condition has been removed, the faulted messages in the source must be reset. This enables them to be resubmitted. In case of business errors, the faulted messages in the source must be removed and sent to fallout management for further action.

The Message Resubmission Utility can be used to resubmit the messages for reprocessing by the correct source.

Error handling and recovery for the asynchronous MEP to ensure guaranteed message delivery is implemented as:

1. Ensure that each message has a unique message identifier.
2. Populate the enterprise business message (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. Resolve the JMS destination (or source application) specific to the source milestone identifier in the AIA Configuration Properties file.
5. 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 *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* "Using the Message Resubmission Utility."
For more information about the Message Resubmission Utility API, see Using the Message Resubmission Utility API.

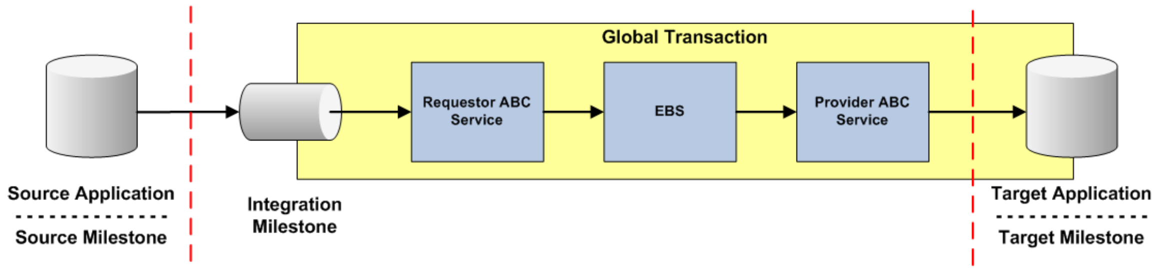
Configuring Milestones

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.

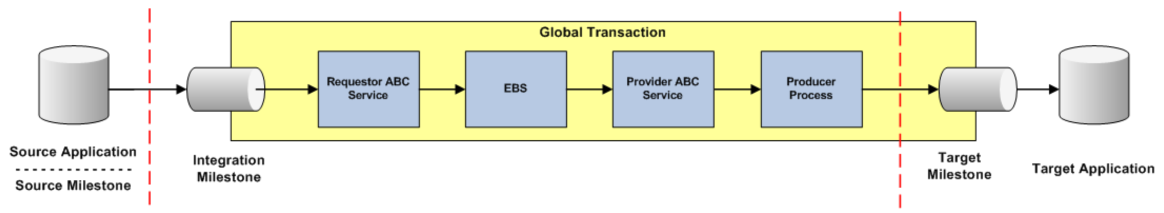
A milestone can be a JMS queue, a JMS topic, or an Oracle ESB Resequencer store.

For more information about configuring a JMS queue, see Chapter 8: Designing and Constructing JMS Adapter Services.
For more information about configuring a JMS topic, see Chapter 19: Describing the Publish-and-Subscribe Programming Model.
For more information about configuring the Oracle ESB Resequencer, see Oracle Enterprise Service Bus Resequencer.

These diagrams illustrate a few possible milestone locations across an integration scenario.



Integration scenario in which the receiver target milestone is the target participating application



Integration scenario in which the receiver target milestone is within the global transaction space

To configure a milestone:

1. Set the source milestone identifier-to-source milestone mapping in the Services Configuration section of the AIA Configuration Properties file.

   <Property name="**[Source Milestone Identifier]**"> **[Source Milestone Name]**</Property>
   </ServiceConfiguration>

2. Ensure that there is a unique MSGID value for each message in the source milestone. To do this, set the JMS header with the unique MSGID.

Configuring Services Between Milestones

This section discusses how to:

- Populate message resubmission values.
- Configure all services to participate in a single global transaction.
- Configure fault policies to not issue rollback messages.

Completing these activities will ensure 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 or the Oracle ESB Resequencer service transformation:

- SenderResourceTypeCode
  Indicates the type of resource or system where the rolled back message is stored, whether Queue, Topic or Resequence Store
- SenderResourceID
  Identification of the resource/system of type SenderResourceTypeCode
- SenderMessageID
  Identification of the message persisted in the resource/system associated with type SenderResourceTypeCode

For more information about resubmitting messages, see *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* “Using the Message Resubmission Utility.”

Configuring All Services to Participate in a Single Global Transaction

Configure a single global transaction as:

- Ensure that there are no commit points between two milestones.
- Ensure that the work done between two milestones is one logical unit of work.

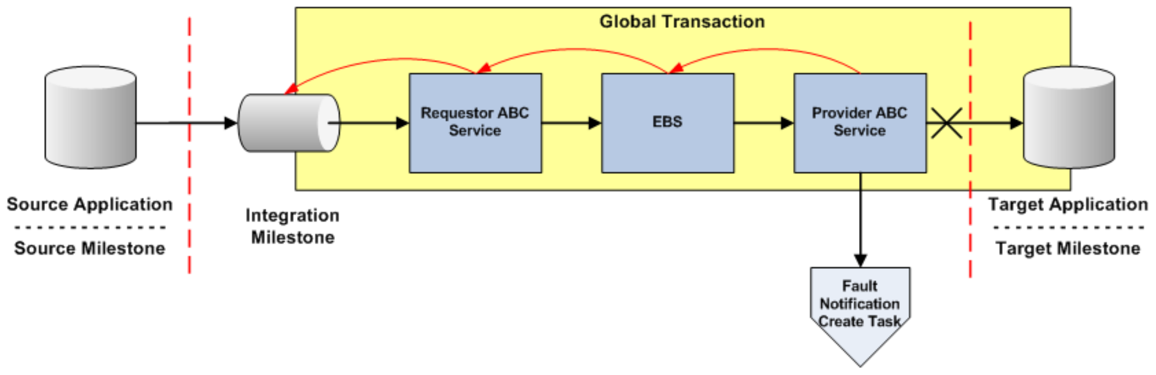
For more information about configuring the global transaction, see Chapter 3: Designing and Developing EBSs, Chapter 4: Designing and Constructing ABC Services, and Chapter 5: Designing and Constructing EBFs.

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 Resequencer Store. It is here that the message will be persisted until it can be resubmitted for delivery to the service or component.

The ESB and BPEL processes along the transaction rollback path will 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 will issue unnecessary notifications. For example, in the transaction rollback flow illustrated in the diagram, redundant rollback notifications would be sent out by the enterprise business service (EBS) and the Requestor application business connector (ABC) service, in addition to the one sent out by the Provider ABC service, which is the only one that should be issued.



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.

  ```
  <bpelx:exec name="Java_Embedding_1" language="java" version="1.5">
     <![CDATA[oracle.apps.aia.core.eh.IAIAErrorHandler instance =
     oracle.apps.aia.core.eh.AIAErrorHandlerFactory.getErrorHandler();
     instance.sendNotification((oracle.xml.parser.v2.XMLElement)getVar
     iableData("AIAFaultMessage","AIAFault","/corecom:Fault"));]]>
  </bpelx:exec>
  ```

- Add an empty "no-op" action to the fault policies of ESB and BPEL processes along the transaction rollback flow. This empty "no-op" action is ***aia-no-action***.

When an ESB or BPEL process receives a rollback message, the control will be directed to the class ***oracle.apps.aia.core.eh.BPELJavaNoAction***, which is implemented against the ***aia-no-action*** action. The ***oracle.apps.aia.core.eh.BPELJavaNoAction*** class is an empty operation, meaning it does nothing, and thus suppresses further notifications in the rollback flow.

These sample BPEL and ESB 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.

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicy version="2.0.1"
id="SamplesCreateCustomerPartyPortalProvABCSImplFaultPolicy"
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>
         <faultName
xmlns:plmfault="http://xmlns.oracle.com/EnterpriseServices/Core/Cust
omerParty/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.BPELJavaNoAction"
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.BPELJavaAction"
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>
```

Sample ESB Fault Policy File

```
<?xml version="1.0" encoding="UTF-8"?>
<esbfaultPolicy id="aia"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.oracle.com/aia/esbfaultpolicy
esb-fault-policy.xsd"
xmlns="http://schemas.oracle.com/aia/esbfaultpolicy">
   <DefaultAction id="default">
         <retry>
               <retryCount>0</retryCount>
               <retryInterval>1</retryInterval>
               <exponentialBackoff/>
         </retry>
   </DefaultAction>
   <ProcessName id="testProcess2">
         <Conditions>
               <faultName name="env:Server">
                     <condition>
                           <test>$fault.summary/summary[contains(.,
"ORABPEL-02180")]</test>
                           <action ref="aia-do-nothing"/>
                     </condition>
                     <condition>
                           <test>contains(Fault/faultstring,"ESB
Fault")</test>
                           <action ref="ora-retry"/>
                     </condition>
                     <condition>
                           <test>$fault.code/code="2"</test>
                           <action ref="ora-retry-crm-endpoint"/>
                     </condition>
               </faultName>
         </Conditions>
         <Actions>
               <Action id="aia-do-nothing">
                     <retry>
                           <retryCount>0</retryCount>
                           <retryInterval>0</retryInterval>

   <exponentialBackoff>0</exponentialBackoff>
                     </retry>
               </Action>
               <Action id="ora-retry">
                     <retry>
                           <retryCount>3</retryCount>
                           <retryInterval>3</retryInterval>

   <exponentialBackoff>1</exponentialBackoff>
```

```
                    </retry>
                </Action>
                <Action id="ora-retry-crm-endpoint">
                    <retry>
                        <retryCount>5</retryCount>
                        <retryInterval>5</retryInterval>

        <exponentialBackoff>1</exponentialBackoff>
                    </retry>
                </Action>
            </Actions>
        </ProcessName>
    </esbfaultPolicy>
```

Using the Message Resubmission Utility API

The Message Resubmission Utility API enables external programs to utilize the functionality provided by the Message Resubmit Utility.

For more information about the Resubmit Utility, see *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* "Using the Message Resubmission Utility."

To utilize the Resubmit Utility API, invoke the ***oracle.apps.aia.core.eh.resubmit.util.AIAEHResubmissionUtil*** class.

There are three methods that can be called, depending on the type of resource in which the faulted message is persisted.

Method signatures include:

- For persistence in a topic:

  ```
  public boolean resubmitFailedMessageTopic( String topicName, String
  topicTableName, String messageID );
  ```

- For persistence in a queue:

  ```
  public boolean resubmitFailedMessageQueue( String queueName, String
  queueTableName, String messageID );
  ```

- For persistence in a Resequencer store:

  ```
  public boolean resubmitFailedMessageResequencer( String
  routingServiceName, String groupID );
  ```

For more information about message persistence, see Configuring Milestones.

Chapter 15: Extensibility for Oracle AIA Artifacts

This chapter discusses:

- Extending enterprise business objects (EBOs).
- Extending enterprise business flows (EBFs).
- Extending application business connector (ABC) services.
- Extending enterprise service bus (ESB) routing rules.
- Extending enterprise business messages (EBMs).
- Extending enterprise business services (EBSs).

For more information about extensibility, see to the *Oracle Application Integration Architecture - Foundation Pack: Concepts and Technologies Guide.*

Extending EBOs

This section provides overviews of EBO extensibility, EBO structure, and enterprise business message (EBM) structure and discusses how to:

- Extend an EBO.
- Create complex content extensions.

Understanding EBO Extensions

Business application integrations require the sharing of information across heterogeneous applications. Most often, the intended functions of these applications are different. As such, each application's view of a given object and its data may be different. Oracle EBOs address these different views of data across the Oracle portfolio of business applications. However, the implementation of these business applications within the given context of a given customer often introduces the need for additional information within the EBO. For this reason, EBOs embrace the ability to extend each object and component within the EBOs.

Simply making changes throughout the EBOs would likely result in chaos and most certainly could not be supported. Instead, EBO extensions must be made using the orderly and prescribed approach described in this chapter. By making use of the Custom element and the predefined type for the given Custom element(s) being extended and by placing the extended content in the appropriate Custom file, it is possible to add content to the EBO and Component. This EBO extension methodology is applicable to Oracle EBOs whenever additional content is required by an implementation.

In order to allow the customers to extend the EBOs in a non-intrusive manner, every business and common component (complex types) present in each of the EBOs will have an element called ‘Custom’ at the end. The data type for this custom element is defined in the schema modules allocated for holding customer extensions.

Extending an EBO

Much of the data needed for typical business transactions is captured in EBOs and the resulting EBMs. General business EBOs and EBMs are in the Core library. Additionally, individual vertical EBOs and EBMs are provided within the Industry library, which contains a library for each supported industry.

While every organization's transactional needs differ, a large portion of their needs is the same. Consider, for example, how many different ways can there be to do a purchase order? Where possible, the industry-specific EBOs utilize the core EBOs and provide the additional context of the given industry.

While a wealth of information is provided in the EBOs and EBMs, there will almost always be a need to extend the EBMs to meet the requirements of an implementation. However, wherever content is provided, we recommend that it be used. This may mean mapping to different terminology to communicate a common understanding. New content should be added only when EBOs do not contain all of the information that needs to be communicated.

To add new content, you must identify which business processes, objects, and components require the new content, as well as when and how the new content will be used. Once you have this information, you are ready to create the extension.

This section discusses how to:

- Identify extensions.
- Identify extension types.
- Create the extension.
- Create complex content extensions.

Identifying the Extensions

To help illustrate the process of extending an EBO, we will use a sample scenario. We need to extend a SalesOrder being communicated between peer applications that are sharing information about a SalesOrder that is being placed to be fulfilled. The peer applications are a Sales Automation application and an Order Management system. The Sales Automation application sends an order to the Order Management system, and the information being communicated uses a ProcessSalesOrderRequestEBM. The ProcessSalesOrderRequestEBM comes from the Sales Automation application, where an order has been entered and is sent to the Order Management system to fulfill. As a part of the process, the ProcessSalesOrderResponseEBM is used to confirm or reject the order.

In our sample scenario, we will extend both an EBO and a Common Component. This will enable us to illustrate how these extensions are made in different files.

For the EBO extension, we want to add a GrandTotalAmount for the full order in the SalesOrderEBO itself. For the Common Component extension, we want to add the DockNumber to the address in the ShipToParty reference.

Identifying the Extension Type

Now that we have identified the EBO and Common Component to be extended, we need to identify the Custom element types defined within the SalesOrderEBO and the Address Common Component.

There are two ways to accomplish this.

Oracle EBOs follow a consistent naming convention with rules that specifically state how elements must be named, how the Custom elements must be defined, and where the resulting types must be defined.

Based on these rules, we can derive that the complexType for the SalesOrder Custom element is defined in the Core\Custom\EBO\SalesOrder directory in a file named CustomSalesOrderEBO. Furthermore, the name of the type for the Custom element on SalesOrder is CustomSalesOrderType. Because GrandTotalAmount is an attribute, it should be added to Base. So the type to extend should be CustomSalesOrderBaseType.

Likewise, the complexType for the Address Custom element is defined in the Core\Custom\Common\V1 directory in a file named CustomCommonComponents. Furthermore, the name of the type for the Custom element on Address is named CustomAddressType.

The second way to derive this information would be to simply look at the elements. The Custom element on the SalesOrder is defined as:

```
<xsd:element name="Custom"
type="coresordcust:CustomSalesOrderBaseType" minOccurs=
" 0"
 />
```

The Custom element on the Address is defined as:

```
<xsd:element name="Custom" type="corecomcust:CustomAddressType"
minOccurs="0"/>
```

Note: Notice that our sample scenario extends the Core SalesOrder and the Core Address. It is also possible to extend a given industry's Common Components or EBOs by simply working within the specific industry directory. In our example, we are working within the Core directory.

You should extend only within the directory that you want to be affected by the extension. A given industry extension should be done within the given industry directory so that it does not affect other industries. Extensions that are meant to affect more than one industry should be made within the Core directory.

For more information about naming standards, see Appendix A: Oracle AIA Naming Standards.

Creating the EBO Extension

According to our assessment of the sample scenario, we know that we must edit two files to make our extension. The order in which we perform these edits is a matter of personal preference, however a rule of thumb for working with XML schemas is to start from the bottom and work up to the top.

Making the Common Component Extension

To extend the address:

1. Open the Core\Custom\V1 CustomCommonComponents schema module.
2. Find the CustomAddressType complexType.
3. Add an XML Schema sequence model.
4. To the sequence model add a DockNumber element and define it to use the corecom:CodeType type.

   The resulting XML schema will look like:

```
<xsd:complexType name="CustomAddressType">
  <xsd:sequence>
<xsd:element name="DockNumber" type="corecom:CodeType"/>
</xsd:sequence> </xsd:complexType>
```

   Now that this edit has been made, the DockNumber can be used to communicate the dock number for Addresses throughout the EBOs that use the Address type.

   In this example, the DockNumber element does not have a minOccurs or maxOccurs attribute, which means that the DockNumber must be populated if the Custom element occurs. To make the DockNumber optional, set the minOccurs to 0, as illustrated here:

```
<xsd:complexType name="CustomAddressType">
<xsd:sequence>
<xsd:element name="DockNumber" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
```

Making the EBO Extension

To extend the SalesOrderEBO:

1. Open the Core\Custom\EBO\SalesOrder\CustomSalesOrderEBO schema module.
2. Find the CustomSalesOrderBType complexType.
3. Add an XML Schema sequence model.
4. To the sequence model, add a GrandTotalAmount element and define it to use the corecom:AmountType type.
5. Set the minOccurs attribute to 0 to make the GrandTotalAmount optional.

   The resulting XML schema will look like:

```
<xs:complexType name="CustomSalesOrderType">
<xs:sequence>
<xs:element name="GrandTotalAmount" type="corecom:AmountType"
minOccurs="0"/>
</xs:sequence>
```

```
</xs:complexType>
```

All elements should be added as optional to support Query.

Creating Complex Content Extensions

The same method demonstrated to make simple extensions could be used to make extensions that include complex structures. However, keep in mind that while it is possible to add complex content, your user extensions to the applications must also process your complex structures.

To add complex content extensions, you can either use pre-defined common components types or define your own in custom files. For example, if you want to add an OEMPartyReference to the SalesOrder, you can add the OEMPartyReference element to the SalesOrderEBO by adding the element to the CustomSalesOrderType:

```
<xsd:complexType name="CustomSalesOrderType">
<xsd:sequence>
<xsd:element name="GrandTotalAmount" type="corecommon:AmountType"
minOccurs="0"
/>
<xsd:element name="OEMPartyReference"
type="corecommon:PartyReferenceType" min
Occurs="0"/>
</xsd:sequence>
</xsd:complexType>
```

Extending EBFs

An EBF can also invoke custom code during its execution. These will serve as extensibility points. Customers 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 EBF. Each extension point will allow one hook – hence, only a single customer extension can be plugged in. The mechanism is similar to the mechanism for creating ABC services with extensible services, which allows customers to have service implementations that require no modifications to the delivered ABC services.

Designing Extensions-Aware EBF

Each of the extensibility point is modeled as a service operation with a well-defined interface. EBF authors define these interfaces. The EBF implementers at the customer site will have the option to implement the interfaces to instill the specialized behavior. The extensibility interfaces consist of service operations that the EBF invokes to execute the custom message enrichment or transformation or validations specific code implemented by the customer.

The extension points to be provided in an EBF should be arrived at after discussions with product management based on the business requirements. In order for the WSDL to be created for the customer extension EBF, the architecture team will provide the WSDL template. The extension service specific annotation capabilities are not available at this time.

The out-of-the-box implementation of these service operations for all of the EBF 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 EBF is developed to invoke the appropriate service operation at each of the extensibility points. To minimize the overheads, a check is made to ensure that the customer has implemented the service for the relevant extensibility interface. Oracle Application Integration Architecture (AIA) configuration properties have one property for each of the extensibility points. Setting the property to ***Yes*** indicates that there is a custom implementation for the extensibility point. The default value for these properties is ***No*** – hence, the EBF will never invoke the meaningless out-of-the-box implementations of these extensions.

The EBF should assume that the customer extension services will be co-located – hence, appropriate properties need to be set to indicate that. Similarly, the EBF should set the transaction property to ***participate*** to enlist the extension service as part of its own transaction.

Configuration Parameters

The 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 EBF requires these parameters to be specified.

The parameters are required to be configured with a value of ***true*** for a service invocation. When these parameters are not specified in the configuration section for an EBF, the value considered by default is ***false***.

The EBF requires these 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. A suitable name should be selected. This parameter is to be checked 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 sub-sequence of process activities is embedded in the Switch activity.

3. 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.

4. Invoke

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

5. Assign

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

Defining Service Partner Link 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 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.

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 also includes bindings to protocols, concrete address locations, and service elements.

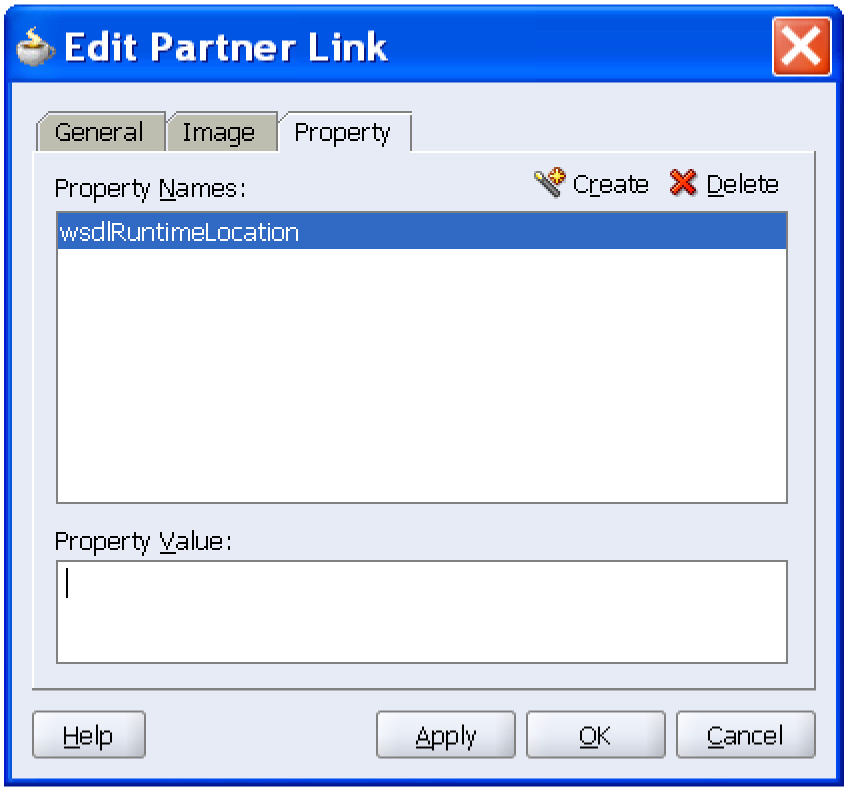
Defining Partner Link Using Abstract WSDL

An abstract WSDL is selected to define the partner link of the service. This WSDL provides interfaces to the operations to be invoked at the extension points. It should also include/import the schemas for the EBO. However, it is devoid of communication transport protocols, or network address locations,, and so on

Specifying a Concrete WSDL at Runtime

A concrete WSDL is a copy of the abstract WSDL and also concrete data types associated with that concrete implementation. 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.

A concrete WSDL is specified using partner link properties. The partner link properties are simple name-value pair properties that can be accessed at runtime by the BPEL process. Using the Property tab of the partner link, the deployment descriptor properties specific to this partner link are defined in the bpel.xml file (This file defines the locations of the WSDL files for services to be called by this BPEL process flow). The location of the concrete WSDL is specified using the property WSDLRuntimeLocation.



Property Names:

wsdlRuntimeLocation

Property Value:

Partner Link Property tab

Testing Extensibility with a Servlet as a Sample Extension Service

The sample extension service used in the document is a java servlet that just will mirror the input payload back. The Servlet as the endpoint has the advantage that it can be used as partner-link service to test any extension. This is useful when the development teams have to test *any* extension-aware EBF that has PartnerLink services defined using Abstract WSDL.

The concrete WSDL whose location is specified as the value of the WSDLRuntimeLocation, points to the servlet as the endpoint. This WSDL has the servlet URL for the service location.

The snippet from the WSDL is illustrated here.

```
<service name="mirrorAnyService">
           <port name="[Name of EBF Extension Service]"
binding="tns: [Name of EBF Extension Service]_Binding">
               <soap:address location="http://ple-
jgau.us.oracle.com:7777/MirrorServlet/mirror"/>
           </port>
    </service>
```

The SOAP request is treated as the servlet payload and the servlet outputs back the entire payload. The SOAPAction element too is rendered as 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:7777/MirrorServlet/mirrorOpen and Closed Issues

Extending ABC Services

An ABC service, regardless of whether it is requestor or provider specific, has the ability to invoke custom code either two or four times during its execution. These will serve as extensibility points.

The ABC service supporting request/response pattern in either synchronous or asynchronous mode has four extensibility points. An ABC service supporting fire-and-forget patterns has two extensibility points. Customers 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 ABC service. Each extension point will allow one hook – hence, only a single customer extension can be plugged in.

This section describes the mechanism for creating ABC services with extensible services, which allows customers to have service implementations that require no modifications to the delivered ABC services.

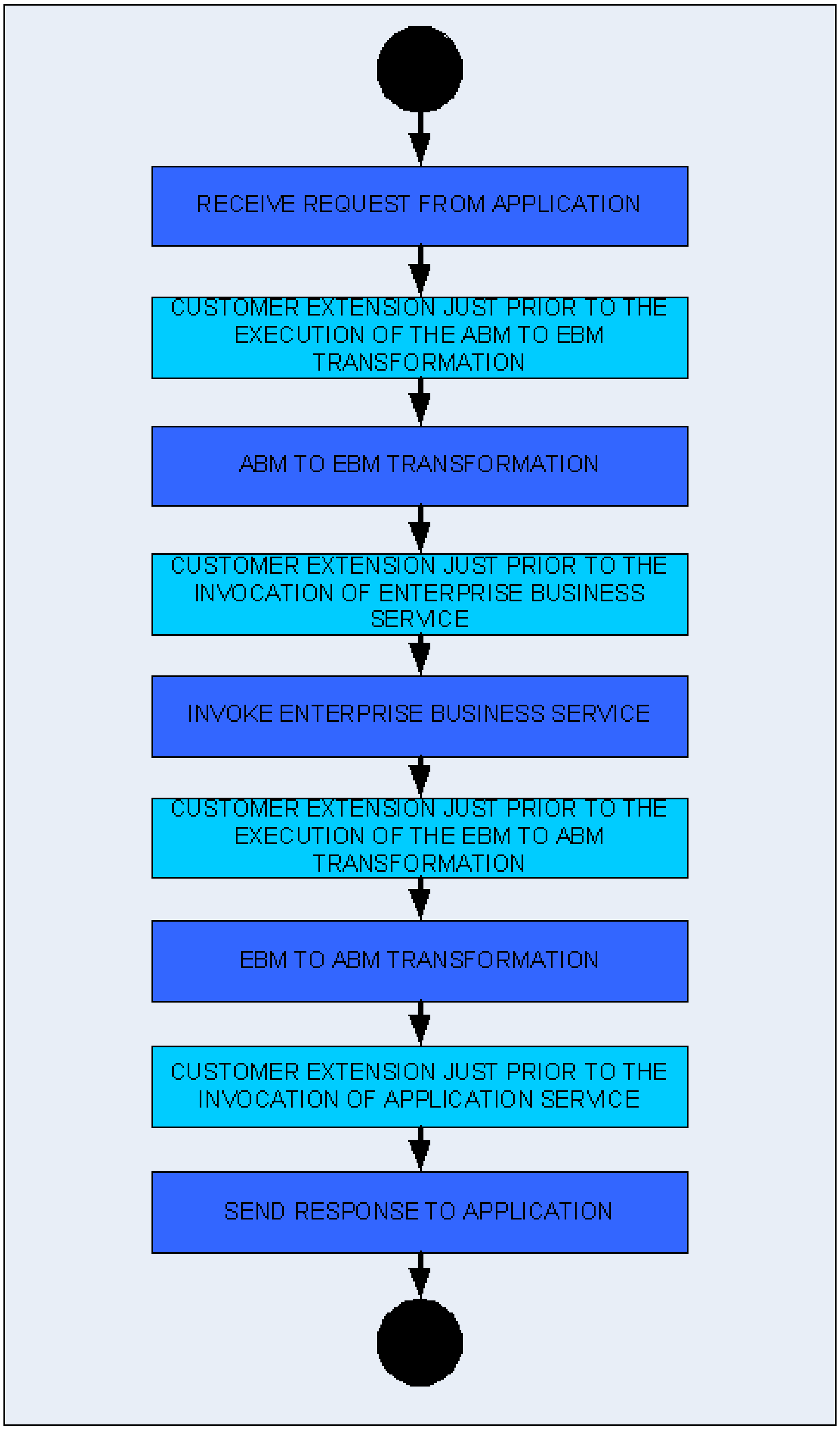
Requestor ABC Service Extensibility Points

Requestor ABC services provide these four extensibility points to allow the customers to hook their custom code to them.

- Just prior to the execution of transformation of application business message (ABM) to EBM
- Just prior to the invocation of the enterprise business service (EBS)
- Just prior to the execution of transformation of EBM to ABM
- Just prior to the invocation of callback service or response return

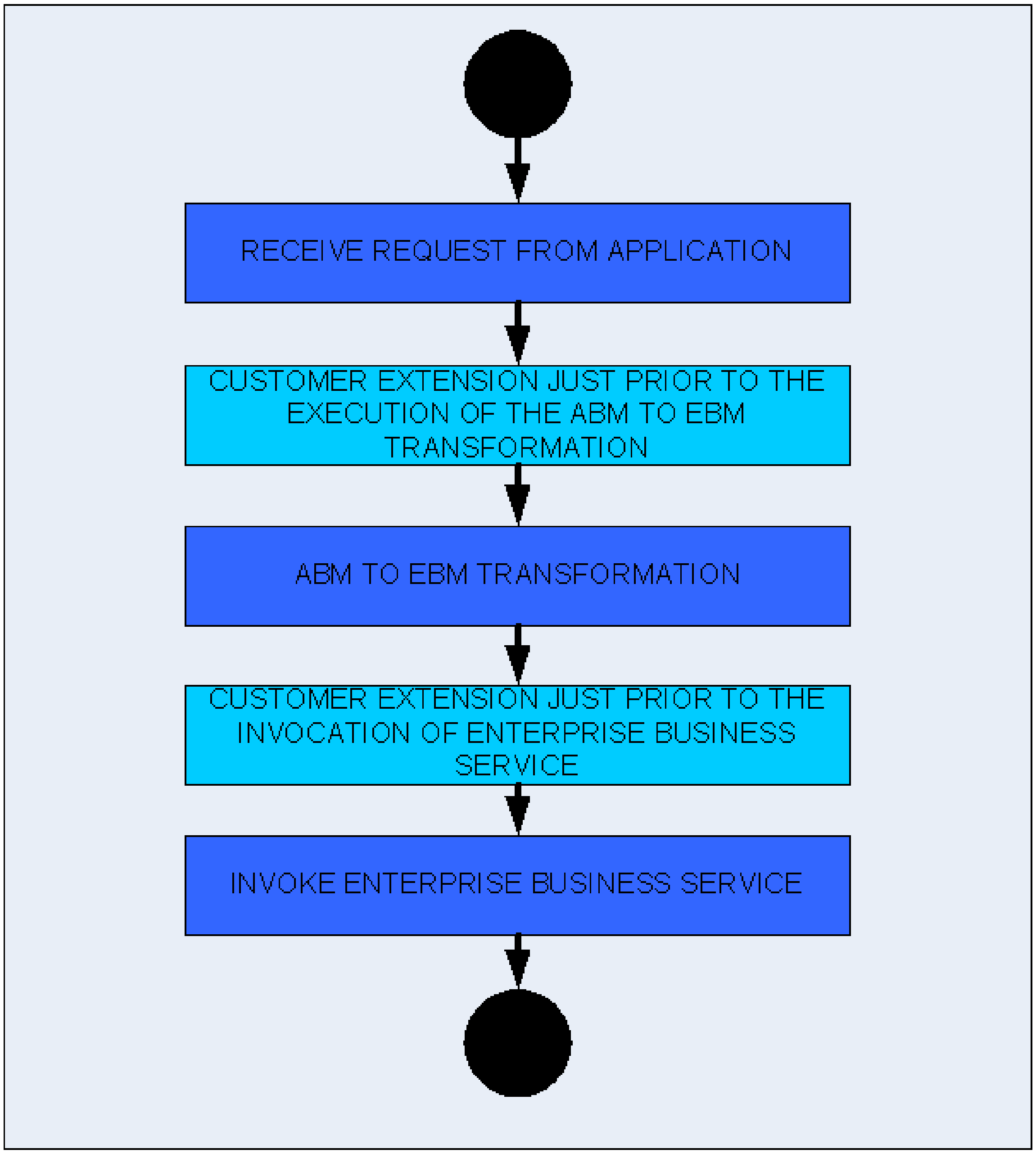
The third and fourth extension points are available only in ABC services implementing request/response pattern.

This activity diagram depicts the high-level flow of activities in a requestor-specific ABC service. The diagram makes an assumption that the EBS with which it is interacting employs a request/response interaction style. Note that the steps for executing the customer extension to do additional tasks are optional.


RECEIVE REQUEST FROM APPLICATION
CUSTOMER EXTENSION JUST PRIOR TO THE EXECUTION OF THE ABM TO EBM TRANSFORMATION
ABM TO EBM TRANSFORMATION
CUSTOMER EXTENSION JUST PRIOR TO THE INVOCATION OF ENTERPRISE BUSINESS SERVICE
INVOKE ENTERPRISE BUSINESS SERVICE
CUSTOMER EXTENSION JUST PRIOR TO THE EXECUTION OF THE EBM TO ABM TRANSFORMATION
EBM TO ABM TRANSFORMATION
CUSTOMER EXTENSION JUST PRIOR TO THE INVOCATION OF APPLICATION SERVICE
SEND RESPONSE TO APPLICATION

Requestor-specific ABC service using request/response interaction style

This activity diagram depicts the high-level flow of activities in a requestor-specific ABC service. The diagram makes an assumption that the EBS with which it is interacting employs a fire-and-forget interaction style. Note that the steps for executing the customer extension to do additional tasks are optional.


RECEIVE REQUEST FROM APPLICATION
CUSTOMER EXTENSION JUST PRIOR TO THE EXECUTION OF THE ABM TO EBM TRANSFORMATION
ABM TO EBM TRANSFORMATION
CUSTOMER EXTENSION JUST PRIOR TO THE INVOCATION OF ENTERPRISE BUSINESS SERVICE
INVOKE ENTERPRISE BUSINESS SERVICE

Requestor-specific ABC service using fire and forget interaction style

The first extensibility point made available to the implementers of the Requestor ABC service 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 will have access to ABM that is about to be transformed. The custom code 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 will have access to EBM that is about to be used for invocation of EBS. The custom code 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 will have 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 will have 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.

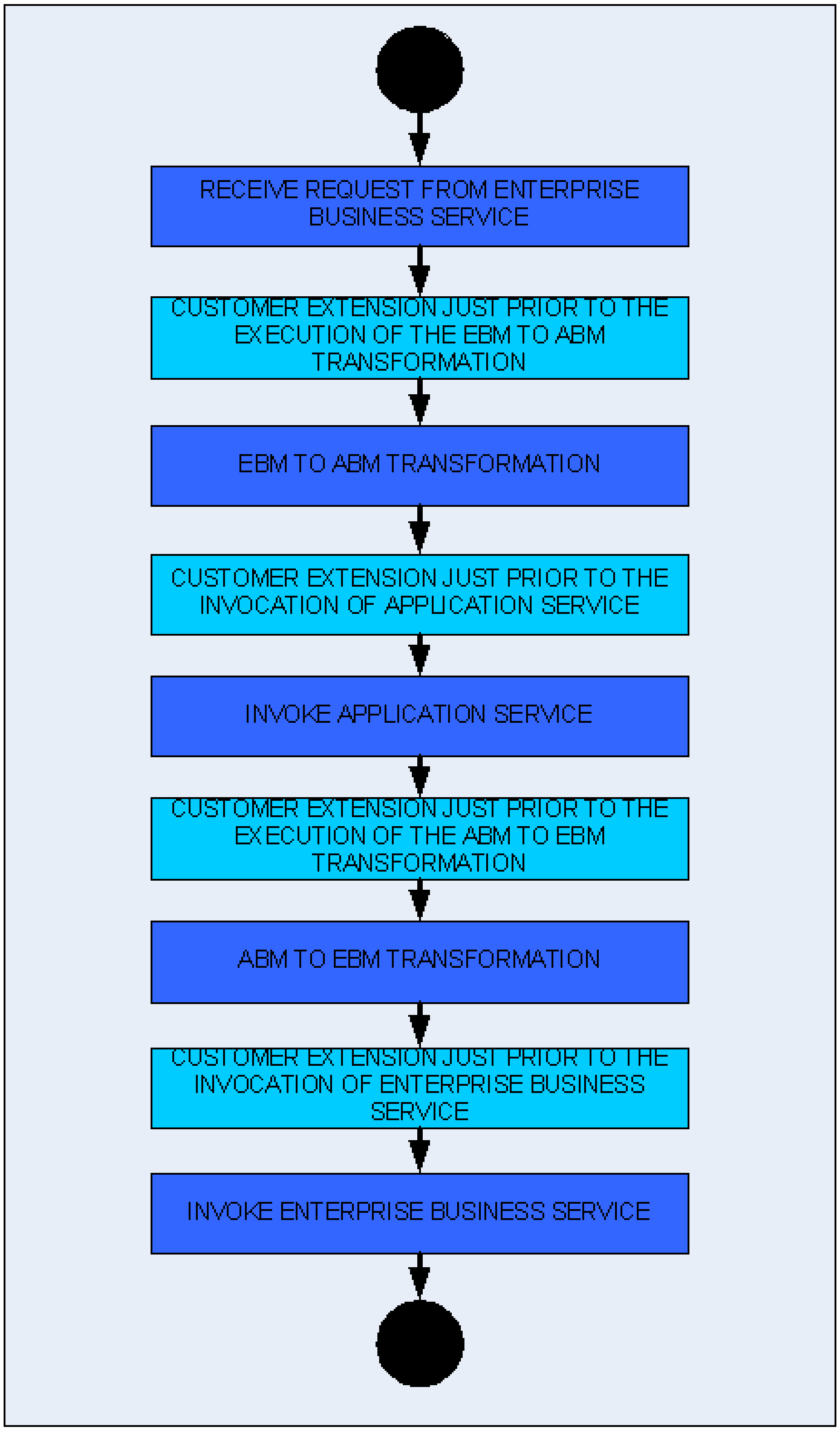
Provider ABC Service Extensibility Points

Provider ABC services provide these four extensibility points to allow customers to hook their custom code to them.

- Just prior to the execution of transformation of EBM to ABM
- Just prior to the invocation of Application Service
- Just prior to the execution of transformation of ABM to EBM
- Just prior to the invocation of callback EBS or return of response message

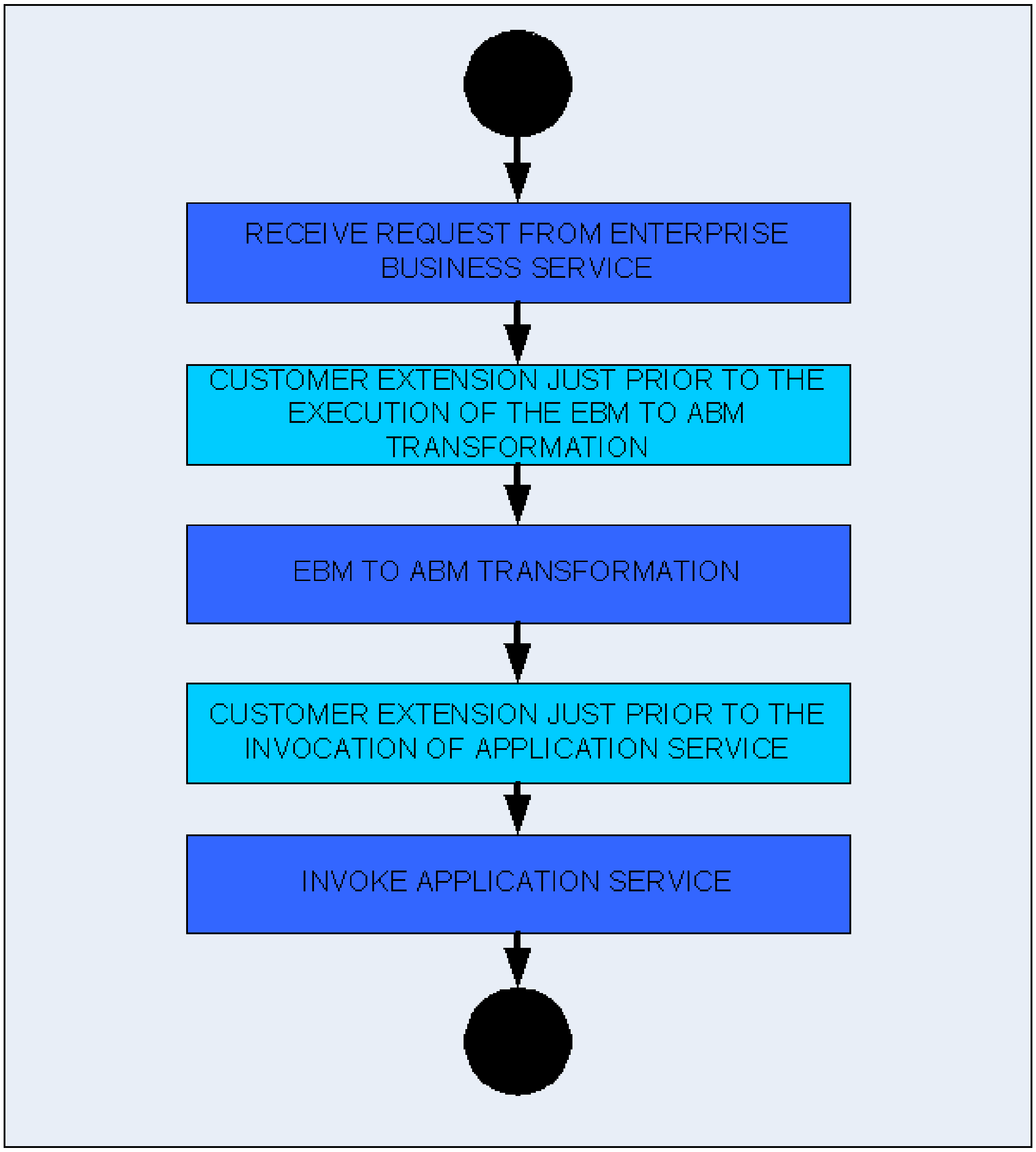
The third and fourth extension points will be available only in ABC services implementing request/response pattern.

This activity diagram depicts the high-level flow of activities in a provider-specific ABC service. The diagram makes an assumption that the EBS with which it is interacting employs a request/response interaction style.


RECEIVE REQUEST FROM ENTERPRISE BUSINESS SERVICE
CUSTOMER EXTENSION JUST PRIOR TO THE EXECUTION OF THE EBM TO ABM TRANSFORMATION
EBM TO ABM TRANSFORMATION
CUSTOMER EXTENSION JUST PRIOR TO THE INVOCATION OF APPLICATION SERVICE
INVOKE APPLICATION SERVICE
CUSTOMER EXTENSION JUST PRIOR TO THE EXECUTION OF THE ABM TO EBM TRANSFORMATION
ABM TO EBM TRANSFORMATION
CUSTOMER EXTENSION JUST PRIOR TO THE INVOCATION OF ENTERPRISE BUSINESS SERVICE
INVOKE ENTERPRISE BUSINESS SERVICE

Provider-specific ABC service using request/response interaction style

This activity diagram depicts the high-level flow of activities in a provider-specific ABC service. The diagram makes an assumption that the EBS with which it is interacting employs a fire-and-forget interaction style.


RECEIVE REQUEST FROM ENTERPRISE BUSINESS SERVICE
CUSTOMER EXTENSION JUST PRIOR TO THE EXECUTION OF THE EBM TO ABM TRANSFORMATION
EBM TO ABM TRANSFORMATION
CUSTOMER EXTENSION JUST PRIOR TO THE INVOCATION OF APPLICATION SERVICE
INVOKE APPLICATION SERVICE

Provider-specific ABC service using fire-and-forget interaction style

The first extensibility point made available to the implementers of the Provider ABC service can be used to inject code to perform tasks such as custom validation, message alteration, message filtering, and so on. The custom code will have 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 will have 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 will have 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 will have 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.

Designing Extensions-Aware ABC Services

Each of the extensibility points is modeled as a service operation having a well-defined interface. ABC service authors define these interfaces. The ABC service implementers at the customer site will have the option to implement the interfaces to instill the specialized behavior. The extensibility interfaces consist of service operations that the ABC service invokes to execute the custom message enrichment or transformation or validations specific code implemented by the customer.

Each of the ABC services will be accompanied by a corresponding customer extension service. In a request/response ABC service, there will be 4 extensibility points – hence, the ABC service extension service will have 4 service operations. For a fire-and-forget ABC service, the corresponding ABC service extension service will have 2 service operations. In order for the WSDL to be created for the customer extension ABC service, the architecture team will be providing the WSDL template. The extension service specific annotation capabilities will not available at this time.

The out-of-the-box implementation of these service operations for all of the ABC service extension services will be to invoke the same piece of code that would always return the same message. This piece of code has been implemented as a servlet.

The ABC service will be developed to invoke the appropriate service operation at each of the extensibility points. To minimize the overheads, a check will be made to ensure that the customer has implemented the service for the relevant extensibility interface. Oracle AIA Configuration properties will have one property for each of the extensibility points. Setting the property to ‘Yes’ indicate that there is a custom implementation for the extensibility point. The default value for these properties will be ‘No’ – hence, the ABC service will never be invoking the meaningless out-of-the-box implementations of these extensions.

This table lists the service operations for the requestor ABC service specific extensibility points.

Extensibility Point	Service Operation Name
Just prior to the execution of transformation of ABM to EBM	Pre-ProcessABM
Just prior to the invocation of the EBS	Pre-ProcessEBM
Just prior to the execution of transformation of EBM to ABM	Post-ProcessEBM

Extensibility Point	Service Operation Name
Just prior to the invocation of callback service or response return	Post-ProcessEBM

This table lists the service operations for the provider ABC service specific extensibility points.

Extensibility Point	Service Operation Name
Just prior to the execution of transformation of EBM to ABM	Pre-ProcessEBM
Just prior to the invocation of Application Service	Pre-ProcessABM
Just prior to the execution of transformation of ABM to EBM	Post-ProcessABM
Just prior to the invocation of callback EBS or response return	Post-ProcessEBM

The ABC service should assume that the customer extension services will be co-located – hence, appropriate properties need to be set to indicate that. Similarly, the ABC service should set the transaction property to ‘participate’ to enlist the extension service as part of its own transaction.

Configuration Parameters

The 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 ABC service requires these parameters to be specified. The parameters for each of the four extension points are:

- ABCSExtension.PreProcessABM
- ABCSExtension.PreProcessEBM
- ABCSExtension.PostProcessEBM
- ABCSExtension.PostProcessABM

These parameters are required to be configured with value ‘true’ for a service invocation. When these parameters are not specified in the configuration section for an ABC service, the value considered by default is ‘false’.

Designing Extension Points

This section details the steps to be carried in order to provide extension points in a requestor ABC service with ‘one-way’ invocation call.

In a requestor ABC service with a request-only call to a service, there are two possible extension points – first extension point is just prior to the transformation from ABM to the EBM and second extension point just prior to 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 merely returns the payload.

Setting up the Extension Point Pre-ProcessABM

The requestor ABC service requires these 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 '. This parameter is to be checked for it value of 'true'. This is achieved by using an appropriate 'AIAXPathFunction' in the switch expression.

2. Assign-Invoke-Assign

   The Assign-Invoke-Assign sub-sequence of process activities is embedded in the Switch activity.

3. 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.

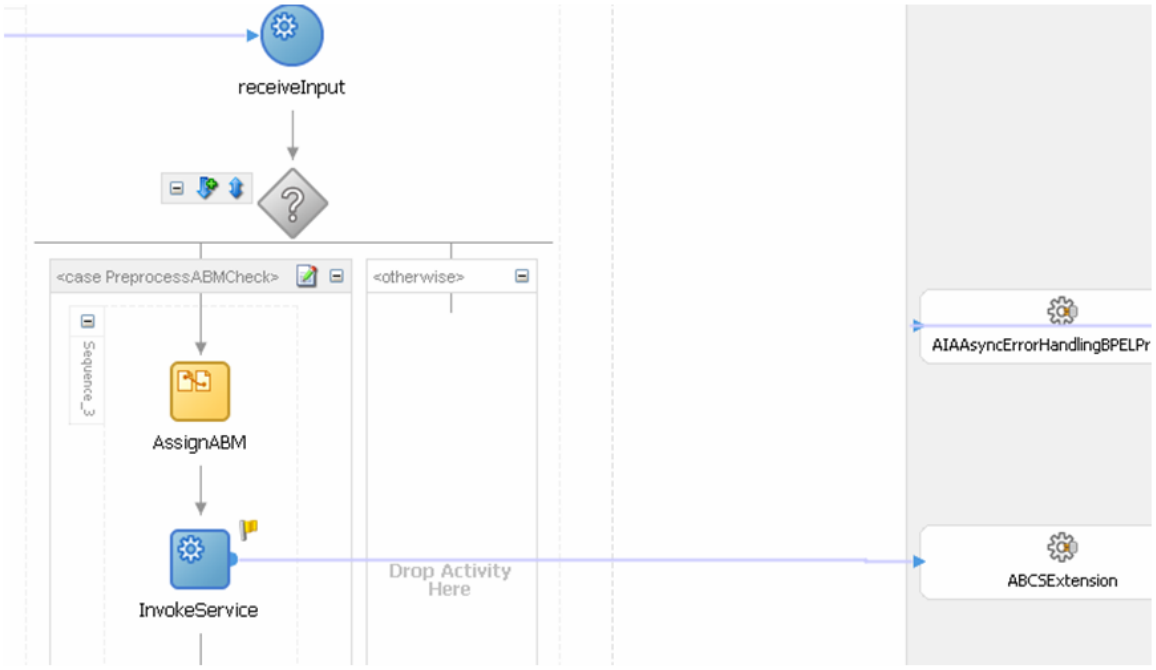
4. Invoke

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

5. Assign

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

This diagram illustrates the sequence of Setting up the Extension Point Pre-ProcessABM.



Setting up the extension point Pre-ProcessABM

Setting up the Extension Point Pre-ProcessEBM

The requestor ABC service requires these 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‘. This parameter is to be checked for it value of ‘true’. This is achieved by using an appropriate ‘AIAXPathFunction’ in the switch expression.

2. Assign-Invoke-Assign

   The Assign-Invoke-Assign sub-sequence of process activities is embedded in the Switch activity.

3. Assign

   The Assign activity assigns the value of EBM to the input variable of the ‘invoke’ step that follows.

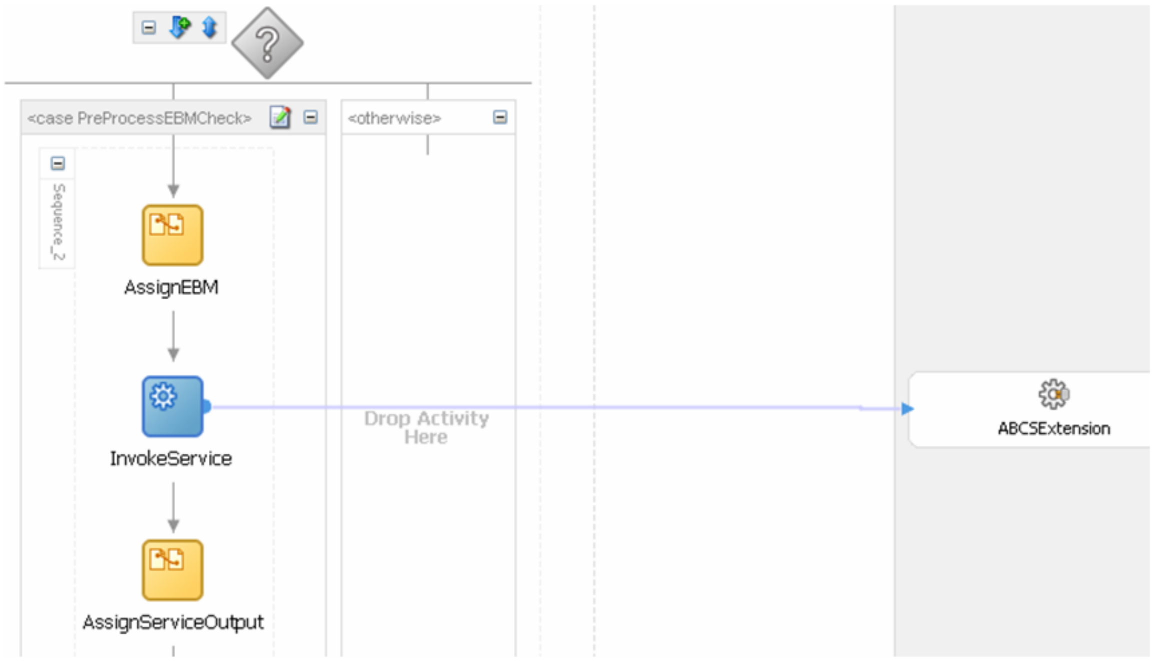
4. Invoke

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

5. Assign

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

This diagram illustrates the sequence.



Setting up the extension point pre-process EBM

Defining Service Partner Link 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.

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 also includes bindings to protocols, concrete address locations, and service elements.

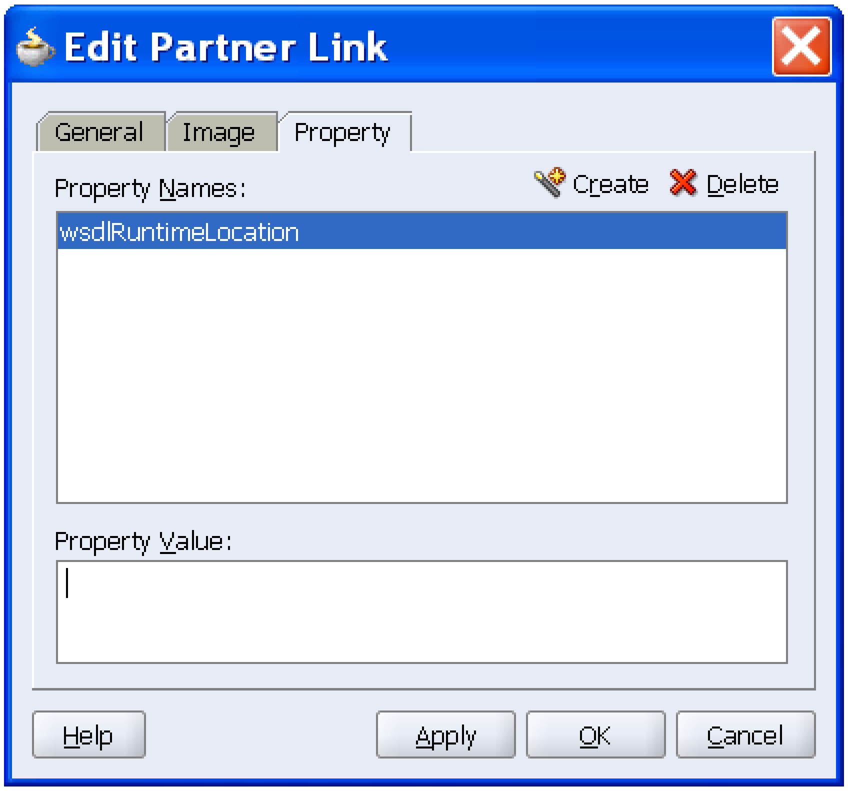
Defining Partner Link Using Abstract WSDL

An abstract WSDL is selected to define the partner link of the service. This WSDL provides interfaces to the operations to be invoked at the extension points. It should also include/import the schemas for the application business object (ABO) and EBO. However, it is devoid of communication transport protocols, or network address locations, and so on.

Specifying a Concrete WSDL at Runtime

A concrete WSDL is a copy of the Abstract WSDL and also concrete data types associated with that concrete implementation. 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.

A concrete WSDL is specified using partner link properties. The partner link properties are simple name-value pair properties that can be accessed at runtime by the BPEL process. Using the property tab of the partner link, the deployment descriptor properties specific to this partner link are defined in the bpel.xml file (This file defines the locations of the WSDL files for services to be called by this BPEL process flow). The location of the concrete WSDL is specified using the property WSDLRuntimeLocation.



Property Names:
wsdlRuntimeLocation

Property Value:

PartnerLink Property tab

Testing the Extensibility with Servlet as Sample Extension Service

The sample extension service used in the document is a java servlet that just will mirror the input payload back. The Servlet as the endpoint has the advantage that it can be used as partner-link service to test any extension. This is useful when the development teams have to test any extension-aware ABC service that have PartnerLink services defined using Abstract WSDL.

The concrete WSDL whose location is specified as the value of the WSDLRuntimeLocation, points to the servlet as the endpoint. This WSDL has the servlet URL for the service location.

The snippet from the WSDL is illustrated here:

```
<service name="mirrorAnyService">
           <port name="CreateCustomerSiebelReqABCSImplExt"
binding="tns:CreateCustomerSiebelReqABCSImplExt_Binding">
```

```
                <soap:address location="http://ple-
jgau.us.oracle.com:7777/MirrorServlet/mirror"/>
            </port>
    </service>
```

The implementations for the methods PreProcessABM are not required - the soap request is treated as the servlet payload and the servlet outputs back the entire payload. The SOAPAction element too is 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:7777/MirrorServlet/mirror

Extending ESB Routing Rules

The task is to move new routing rules added to the version 'x' ESB Routing Services by customers to the version 'x+1' ESB Routing Services delivered by ABS as part of upgrade.

These are the steps:

1. In the version 'x' ESB Routing Services, open the <System Name>_<ESB Routing Service name>.esbsvc file in edit mode.
2. Copy all the <routingrule> ... </routingrule> sections which have been added as a customization to the version 'x' ESB Routing Services.
3. In the version 'x+1' ESB Routing Services, open the <System Name>_<ESB Routing Service name>.esbsvc file in edit mode.
4. Paste all the <routingrule> ... </routingrule> sections copied in point 2 to the <routingrules> ... </routingrules> section after the existing routing rules.
5. From the <targetoperation> of each of the additional routing rules, identify the <serviceQName> and copy the <serviceQName>.esbvc to the 'x+1' ESB Routing Service folder.
6. Open the 'x+1' ESB Routing Service in JDeveloper and check that new routing rules are added and that the target service is set.
7. Register the service and test.

Extending EBMs

An EBM is an assembly of schema components. Therefore, extensibility means a new EBM. The solution is to create a new EBM XSD containing the required EBM definitions.

For example, let’s say there is an ItemEBM.xsd, which contains CreateItemEBM, UpdateItemEBM, and QueryItemEBM, but there is a need for DeleteItemEBM and ValidateItemEBM as well.

To extend ItemEBM.xsd:

1. Create CustomItemEBM.xsd.

2. In CustomItemEBM.xsd, include the definitions for DeleteItemEBM and ValidateItemEBM.

Extending EBSs

Extending an EBS means adding new operations. This can be accomplished by creating an extension EBS (a new ESB routing service) for holding the customer-specific operations.

This newly created ESB routing service can either use the same EBM or the newly created EBM used to house the EBMs for these new operations. Because these are newly created entities, they are totally upgrade-safe. ItemEBS continues to support the standard CRUD operations and is based on the shipped ItemEBS.wsdl.

To support custom operations:

1. Create CustomItemEBS.wsdl, a new WSDL file that defines the interface/contract that includes the custom operations.
2. Create a new ESB project containing a CustomItemEBS routing service pointing to your new WSDL.
3. Deploy the CustomItemEBS.

Chapter 16: Understanding Security

This chapter discusses:

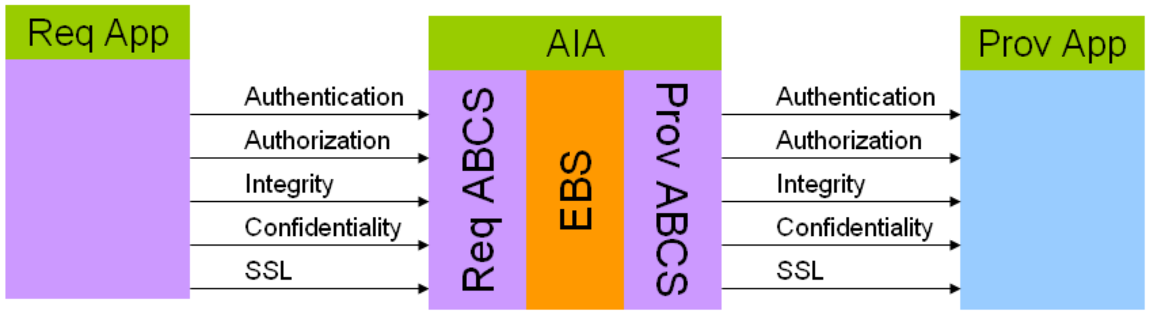
- Secure remote connections.
- Application security context.

Securing Remote Connections

This section provides an overview of securing remote connections including:

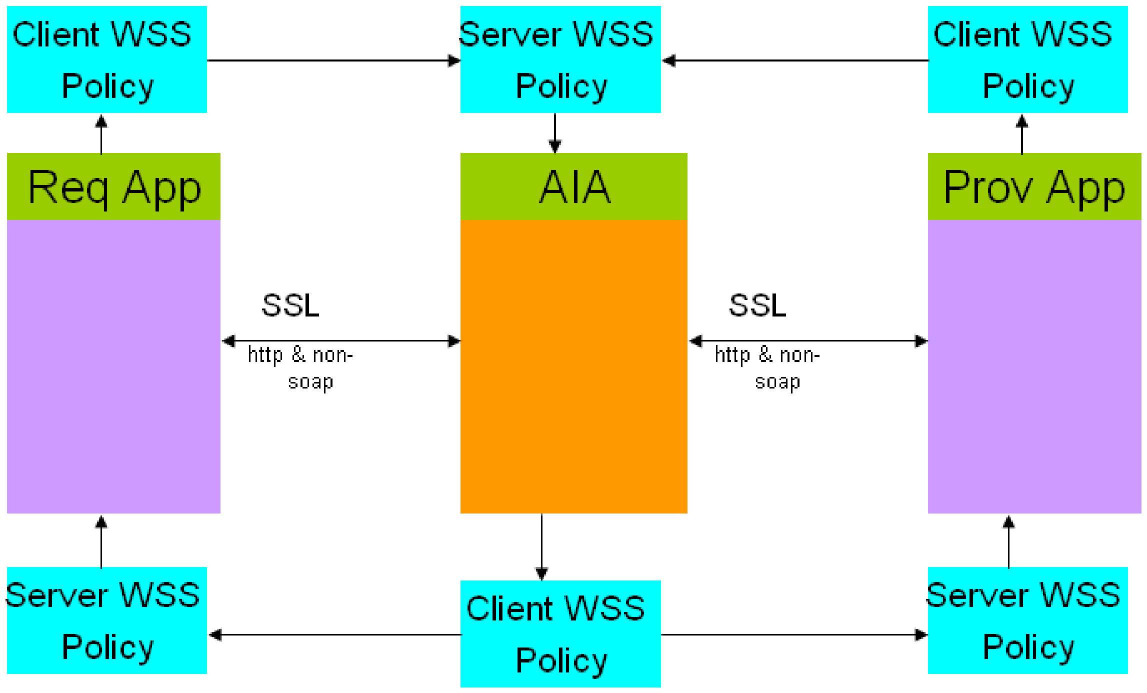
- Identifying clients through authentication
- Securing messages through encryption
- Avoiding message tampering with digital signatures
- Encrypting the channel through SSL

Oracle AIA recommends decoupling security logic from service development by configuring Webservice Security declaratively using Oracle Web Services Manager (OWSM) during deployment. Developers and Customers should use Webservices security unless there is a compelling reason such as participating applications do not support.



High-level security architecture diagram

Oracle AIA recommends using OWSM to configure WS-Security in Oracle AIA. A client side policy needs to be configured on participating application and server side policy on Oracle AIA layer when the message communication is from participating application to Oracle AIA. A client side policy needs to be configured on Oracle AIA side and server side policy on participating application when the message communication is from Oracle AIA layer to participating application.



Oracle AIA security with policies and SSL

For detailed steps on how to use OWSM to configure polices, refer to WS-Basic Authenticate Sample.

Application Security Context

This section provides an overview of application security and discusses:

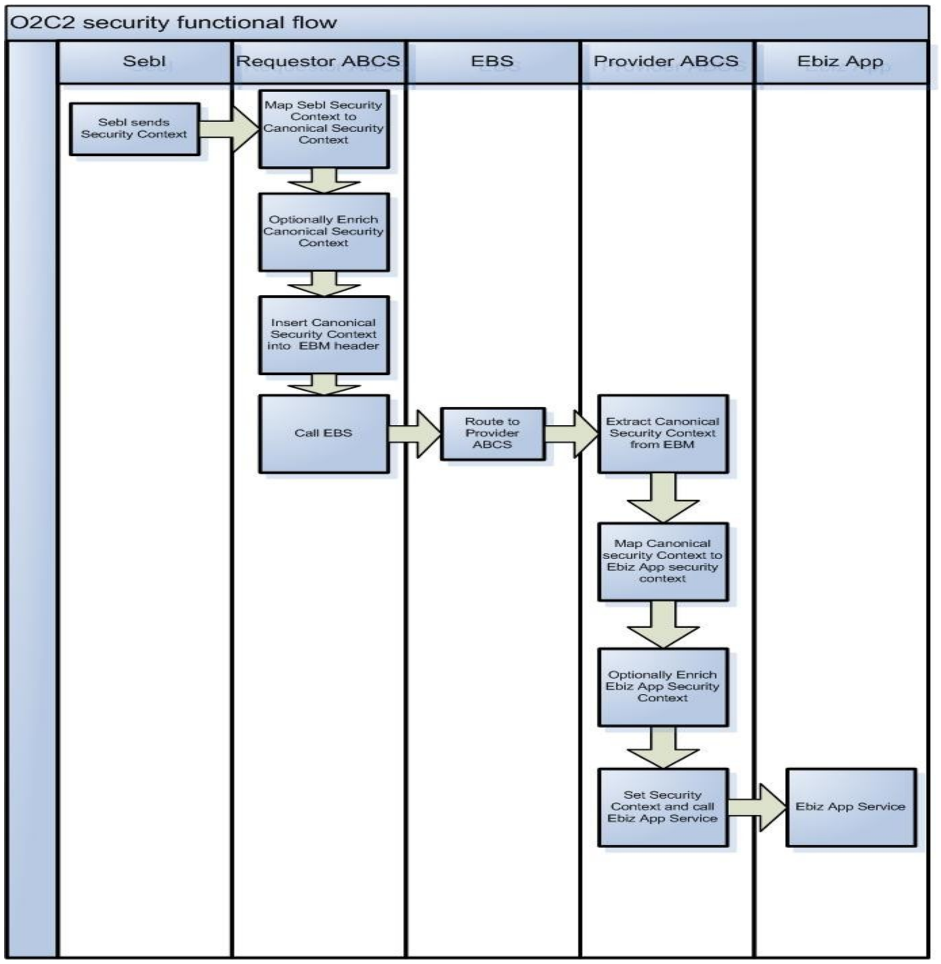
- Exchange of security context between participating application and application business connector (ABC) service
- Mapping application security context in the ABC service to and from standard security context
- AppContext mapping service or XPath function
- Structure for security context
- Attribute names
- Propagating standard security context through enterprise business service (EBS) and enterprise business flow (EBF)

The Oracle AIA application security model allows Oracle AIA to integrate participating applications with different security representations in a standard way by eliminating point-to-point security.

The chapter discusses the exchange of security information between participating applications and ABC services, transforming of application security context into standard format in ABC services, propagating the standard security context from ABC service to ABC service through EBS and EBF, and transforming standard security context to application security context.

The participating applications are developed during different times with different concepts and implementation of authentication and authorization. When applications are integrated, there is a need to pass authentication and authorization information between applications. Oracle 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.

This diagram illustrates the high level functional flow:


O2C2 security functional flow
Sebl
Requestor ABCS
EBS
Provider ABCS
Ebiz App
Sebl sends Security Context
Map Sebl Security Context to Canonical Security Context
Optionally Enrich Canonical Security Context
Insert Canonical Security Context into EBM header
Call EBS
Route to Provider ABCS
Extract Canonical Security Context from EBM
Map Canonical security Context to Ebiz App security context
Optionally Enrich Ebiz App Security Context
Set Security Context and call Ebiz App Service
Ebiz App Service

Security functional flow

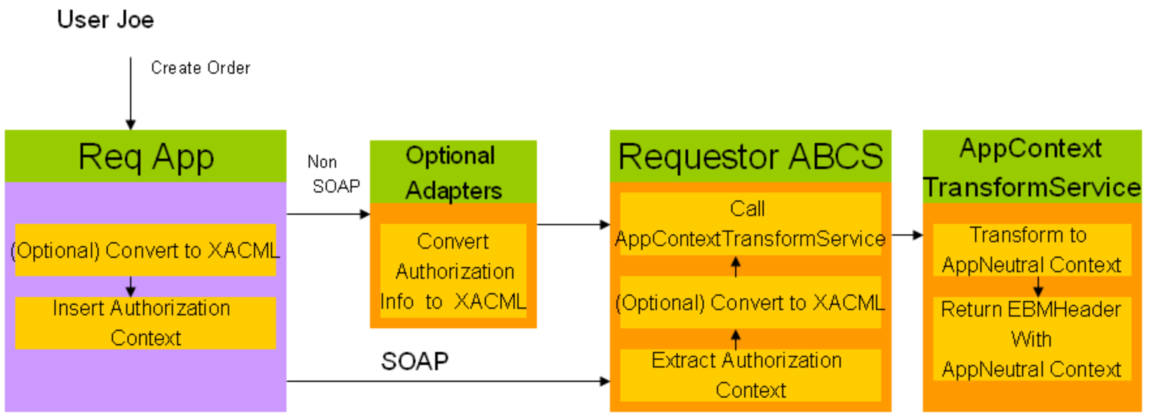
Exchange of Security Context between Participating Applications and ABC Services

AppContext is any information that needs to be sent to the provider application to process the message sent from requestor application or vice versa. This will include but is not limited to authentication and authorization information. In this release, we address the exchange of authorization information in app context, but the design will support adding other context information in future releases.

Oracle AIA evaluated several standards to represent authorization information and determined XACML Context Request is the best standard to use. XACML is an OASIS standard for managing access control policy. Released in 2003 and based on XML, 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 Oracle AIA for exchanging authorization info.

Requestor Applications

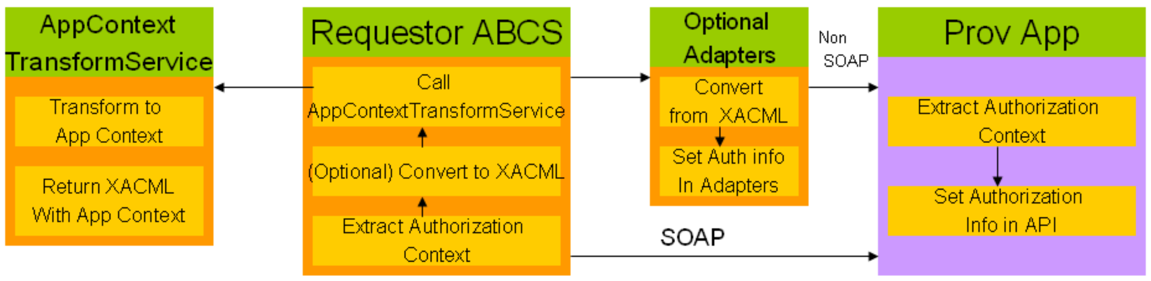
The preferred approach is to let the requestor application send app context information as an XACML request to the Requestor ABC service. If the applications are not capable of formulating context information in XACML request, then the participating application will send app context information in a SOAP header or as part of business message content. Oracle 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 app context in a custom way and prepares the participating application specific XACML request and sends it to the ABC service.


User Joe
Create Order
Req App
(Optional) Convert to XACML
Insert Authorization Context
Non SOAP
SOAP
Optional Adapters
Convert Authorization Info to XACML
Requestor ABCS
Call AppContextTransformService
(Optional) Convert to XACML
Extract Authorization Context
AppContext TransformService
Transform to AppNeutral Context
Return EBMHeader With AppNeutral Context

Requestor application flow

Provider Applications

The preferred approach is to let the provider ABC service send the app 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 ABC service will send the app security context in 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 ABC service sends app context in XACML request format to the adapter service that sets the appropriate security context needed for the security mechanism in use.


AppContext TransformService
Transform to App Context
Return XACML With App Context
Requestor ABCS
Call AppContextTransformService
(Optional) Convert to XACML
Extract Authorization Context
Optional Adapters
Convert from XACML
Set Auth info In Adapters
Non SOAP
SOAP
Prov App
Extract Authorization Context
Set Authorization Info in API

Provider application flow

Mapping Application Security Context in ABC Services To and From Standard Security Context

The requestor ABC service will either receive the application security context in XACML format or convert into XACML format. The requestor ABC service calls an external service to map application security context to standard security context. The ABC service passes the application security context in XACML format and receives application neutral security context in XACML format.

AppContext Mapping Service

Oracle AIA recommends using one external service per application. This service is also responsible for populating additional values needed in 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)

```
<definitions
targetNamespace="http://www.oracle.com/AIA/AppContextTransformServic
e"
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/AppContextTransformServic
e" 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/EnterpriseObjectLib
rary/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/EnterpriseObjectLib
rary/Release2/Core/Common/V2/Meta.xsd" />
</xsd:schema>
   </types>
   <message name="Request">
     <part name="Request" element="xacml-context:Request"/>
   </message>
   <message name="EBMHeader">
     <part name="EBMHeader" element="corecom:EBMHeader"/>
   </message>
   <portType name="TransformAppContext">
     <operation name="TransformToAppContext">
```

```
      <input message="EBMHeader"              name="EBMHeader"/>
      <output message="Request"/>
    </operation>
    <operation name="TransformToAppNeutralContext">
      <input message="Request"              name="Request"/>
      <output message="Request"/>
    </operation>
  </portType>
</definitions>
```

This service is implemented for the participating application and meets any integration scenario using that application.

Oracle AIA recommends using BPEL with co-location to implement this service. ABC services should call this service using a dynamic partner link so that customers can plug in other implementations of this service.

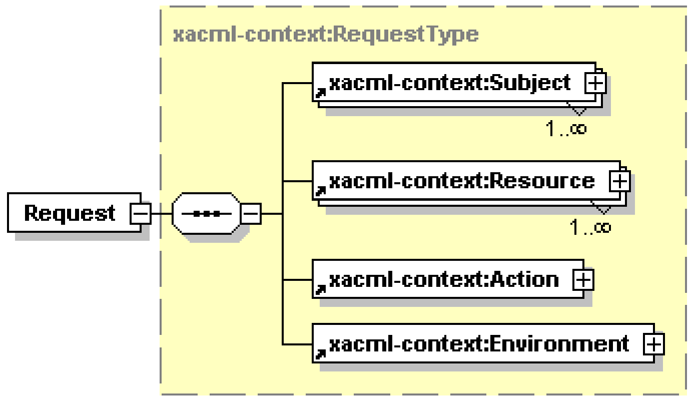
The property to load the service implementation is loaded from AIAConfig property file. The name of the property is TransformAppContextService. By default this property is not configured and the default implementation is used.

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 does not need to be changed.

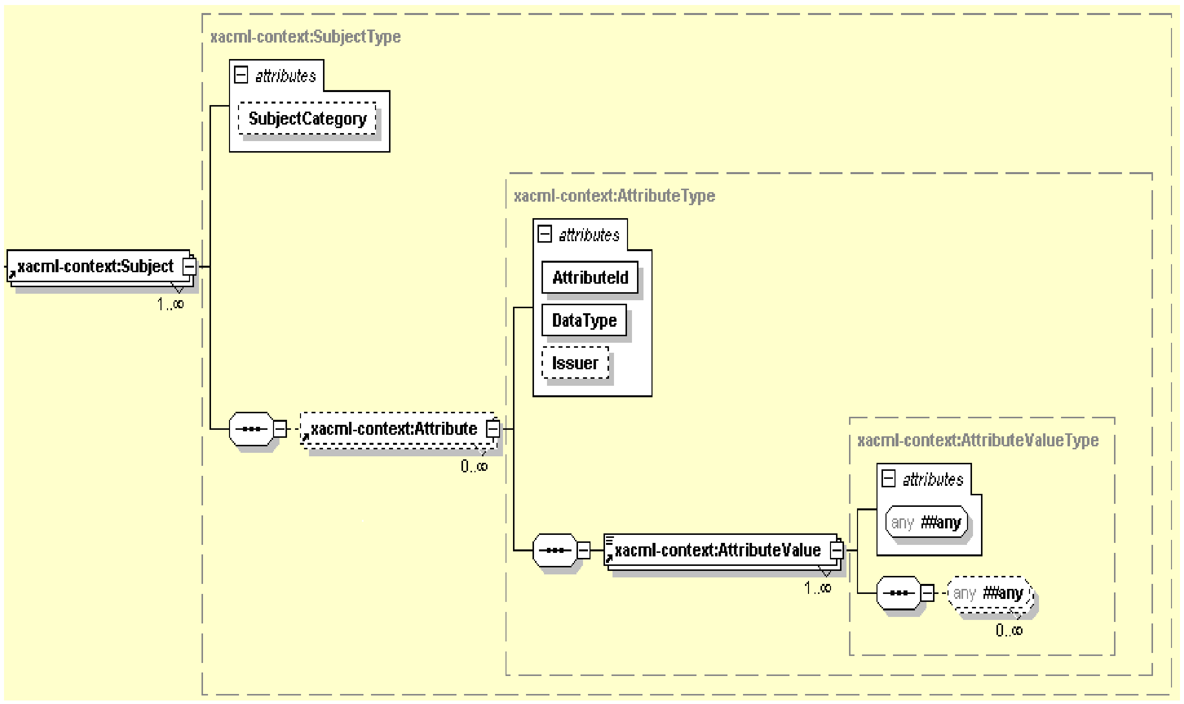
Structure for Security Context

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.

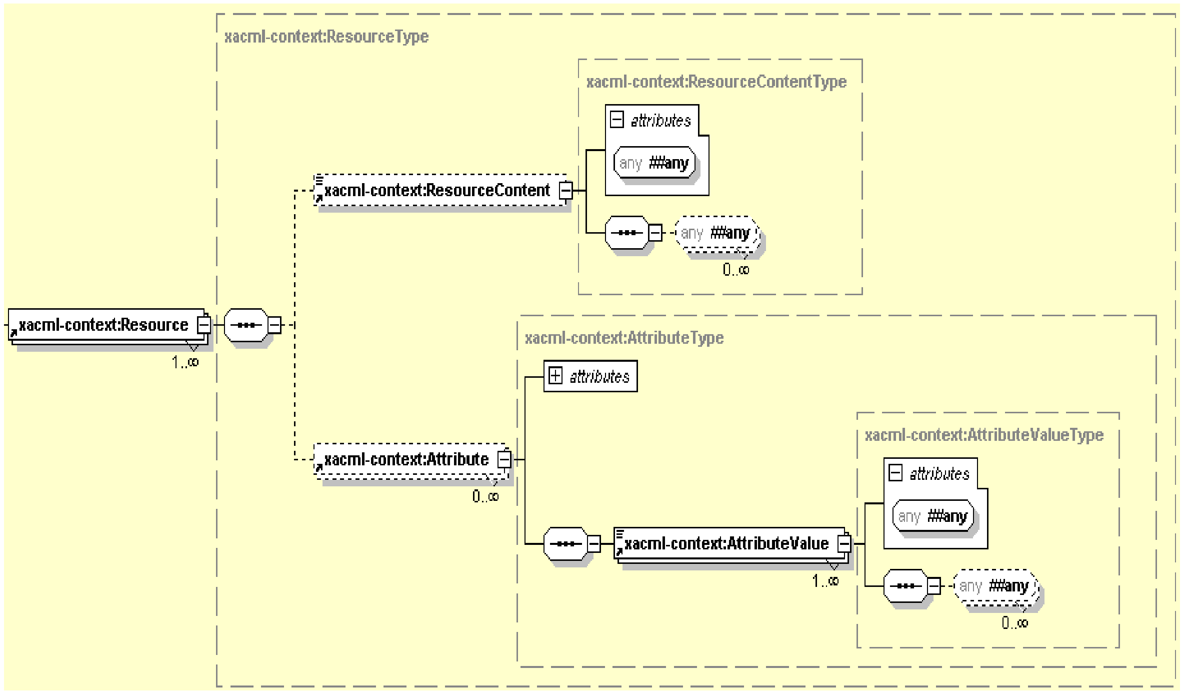
These diagrams illustrate the request element in XACML context:



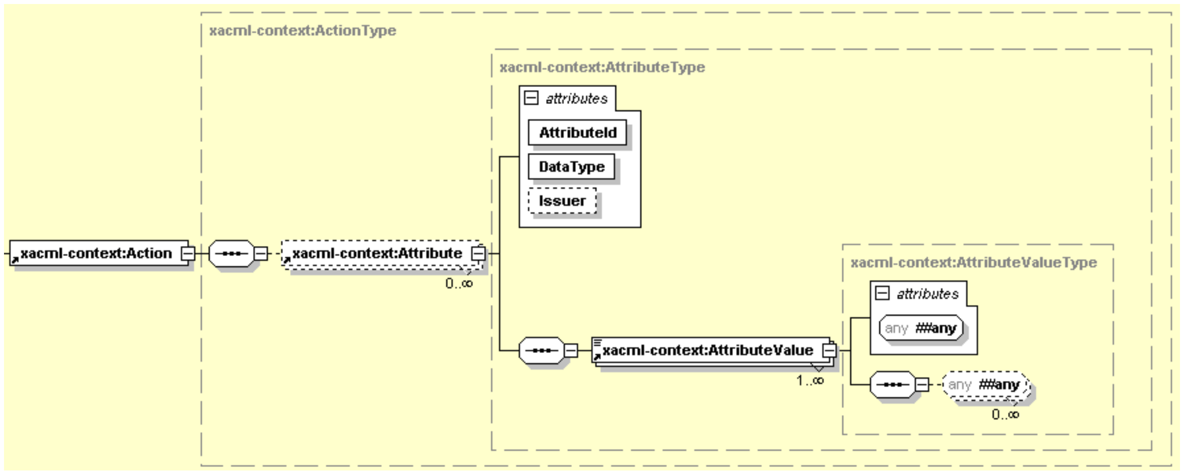
Structure of XACML Request



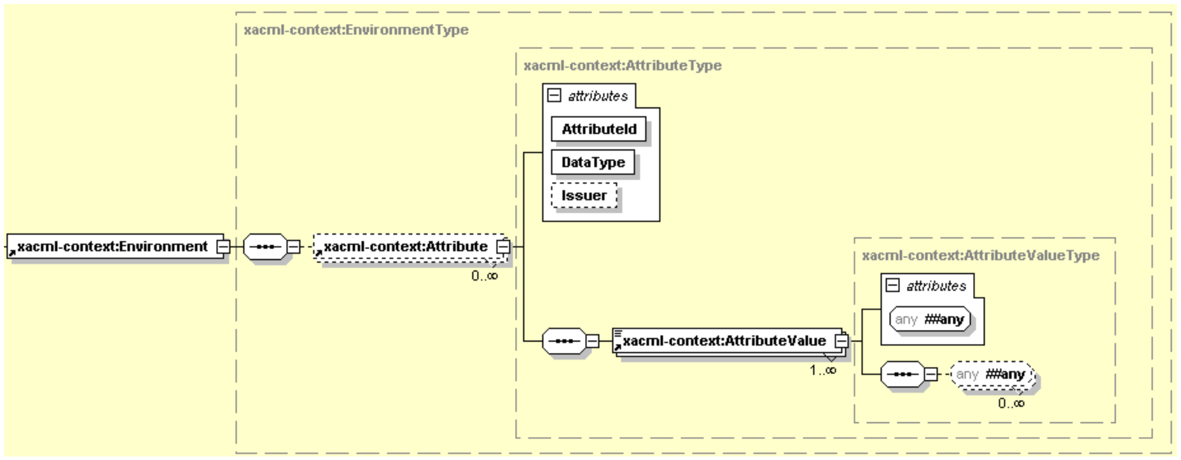
Structure of XACML Subject


xacml-context:ResourceType
xacml-context:ResourceContentType
attributes
any ##any
xacml-context:ResourceContent
any ##any
0..∞
xacml-context:Resource
1..∞
xacml-context:AttributeType
attributes
xacml-context:AttributeValueType
attributes
any ##any
xacml-context:Attribute
0..∞
xacml-context:AttributeValue
1..∞
any ##any
0..∞

Structure of XACML Resource



Structure of XACML Action


xacml-context:EnvironmentType
xacml-context:AttributeType
attributes
AttributeId
DataType
Issuer
xacml-context:Environment
xacml-context:Attribute
0..∞
xacml-context:AttributeValueType
attributes
any ##any
xacml-context:AttributeValue
1..∞
any ##any
0..∞

Structure of XACML Environment

This example shows the SEBL AppContext information that is 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>
```

Attribute Names

1. Service information attributes
   - AIA:Service:Name – Name of the service calling the transform service
2. 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
3. Application attributes
   - Oracle AIA recommends using this convention for naming the attributes for all the applications: Application name: attribute name.
4. Application neutral attributes

   Oracle AIA recommends using AIA as prefix for all the application neutral attributes. These are the application neutral attributes are 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 ABC service, security context is propagated along with the EBM to the target ABC service where it is used to propagate to the target application

Part 2: Data Integration

Chapter 17: Oracle AIA Components for Data Integration

This chapter discusses how to:

- Describe data integration patterns.
- Build Oracle Data Integrator (ODI) projects.
- Use the XREF Knowledge Module.
- Handle update – delete in source interfaces.
- Work with domain value maps (DVMs).

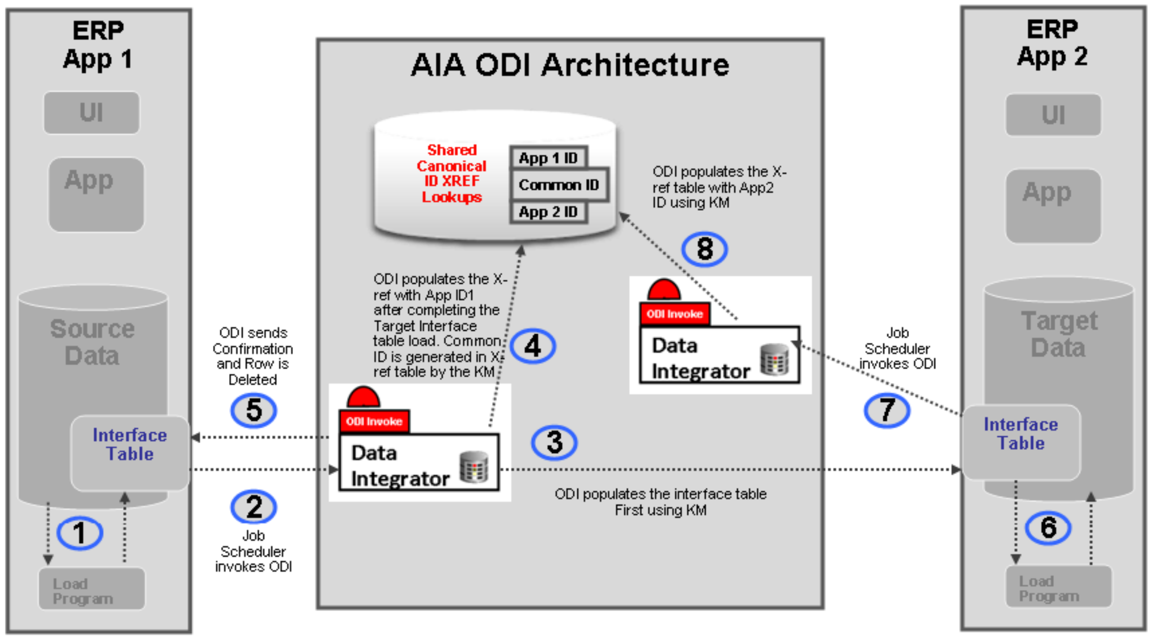
Describing Data Integration Patterns

Oracle Application Integration Architecture (AIA) supports these data integration patterns:

- Initial data loads
- High-volume transactions with a cross reference (XREF) table
- Intermittent high-volume transactions
- High-volume transactions without an XREF table

Initial Data Loads and High-Volume Transactions with an XREF Table

When implementing any integration, there may be some records that need to be synchronized. Using Oracle Data Integrator (ODI) allows us to handle a potentially large dataset and maintain XREFs, if necessary.


ERP App 1
UI
App
Source Data
Interface Table
Load Program
1
2
Job Scheduler invokes ODI
5
ODI sends Confirmation and Row is Deleted
AIA ODI Architecture
Shared Canonical ID XREF Lookups
App 1 ID
Common ID
App 2 ID
ODI populates the X-ref with App ID1 after completing the Target Interface table load. Common ID is generated in X-ref table by the KM
4
ODI Invoke
Data Integrator
3
ODI populates the interface table First using KM
ODI populates the X-ref table with App2 ID using KM
8
ODI Invoke
Data Integrator
Job Scheduler invokes ODI
7
ERP App 2
UI
App
Target Data
Interface Table
6
Load Program

Data load with XREF table

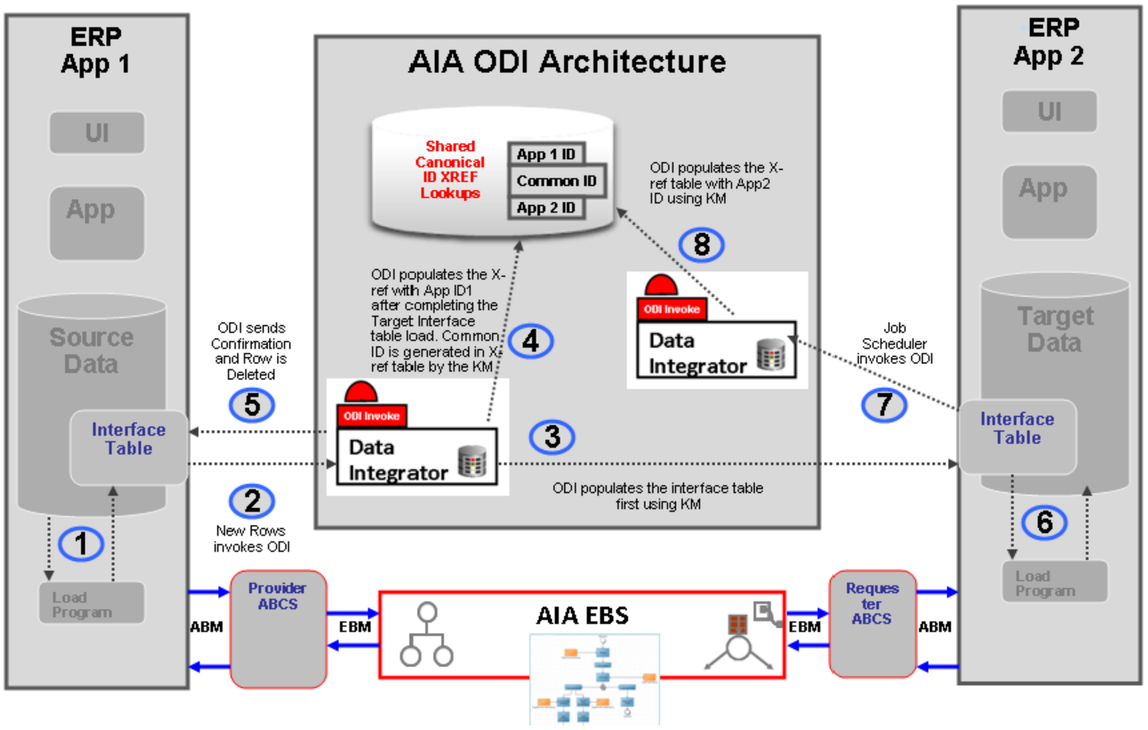
The Oracle AIA Order-to-Cash process integration pack uses this pattern to synchronize accounts in Oracle E-Business Suite with customers in Siebel CRM. The loosely coupled Oracle AIA enterprise business service (EBS) could be used to achieve this integration, but ODI is used given the potential of an ultra-high volume of records.

Since there can be upwards of millions of accounts in Oracle E-Business Suite, the risk of incurring an unacceptable installation time is mitigated by leveraging the point-to-point data integration with ODI. After the integration is implemented, further account creation or updates in Oracle E-Business Suite are synchronized in Siebel CRM using Oracle AIA EBS, and ODI is no longer used.

This same pattern is used in ongoing high-volume transactions, typically in a batch process. For example, in a retail environment, store locations may send order information in a batch to a central office. This is an ongoing batch where a high volume of orders may occur.

Intermittent High-Volume Transactions

The intermittent high-volume transactions pattern covers the scenario in which a data integration is managed both by ODI and by Oracle AIA EBS. Normal (Instancebased) synchronization is done through Oracle AIA EBS, but if there is the need to run transactions in batch, then ODI can be used to perform the synchronization. ODI can also be used for intermittent transactions that may contain a very large object, such as an order with an exceptional number of lines.

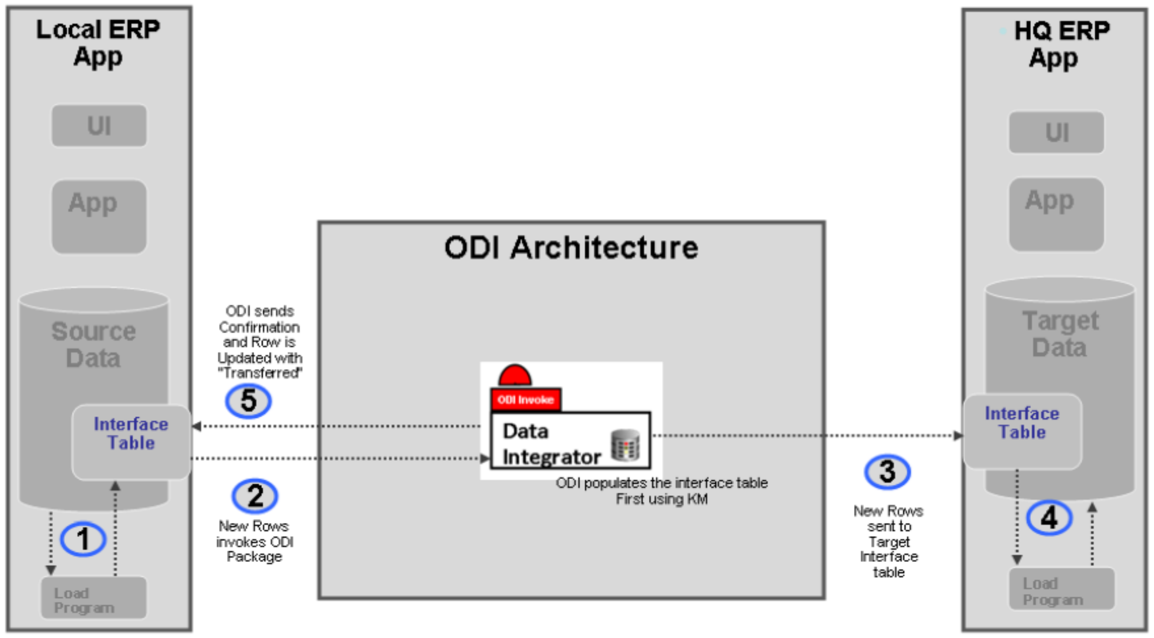

ERP App 1
UI
App
Source Data
Interface Table
Load Program
AIA ODI Architecture
Shared Canonical ID XREF Lookups
App 1 ID
Common ID
App 2 ID
ODI populates the X-ref table with App2 ID using KM
8
ODI populates the X-ref with App ID1 after completing the Target Interface table load. Common ID is generated in X-ref table by the KM
4
ODI sends Confirmation and Row is Deleted
5
ODI Invoke
Data Integrator
3
ODI populates the interface table first using KM
2
New Rows invokes ODI
1
Job Scheduler invokes ODI
7
ERP App 2
UI
App
Target Data
Interface Table
6
Load Program
Provider ABCS
ABM
EBM
AIA EBS
Reques ter ABCS

Intermittent High Volume Transactions

When using this pattern, there is typically some switching logic employed to decide which route to take. If a system were receiving incoming orders entered through a CRM system, for example, it may use the Oracle AIA EBS route, since these orders are singular in nature, and may also require some level of feedback to a customer service representative. If this same system were to also receive batches of orders through an EDI gateway, then it may place those orders in an interface table to be picked up by ODI and processed in batch.

High-Volume Transactions without an XREF Table

If no further data manipulation language (DML) operations need to occur with synchronized records, maintaining an XREF record would not be necessary. An example of this may be the retail environment where store locations transmit their daily sales transactions to the headquarters (HQ) location.


Local ERP App
UI
App
Source Data
Interface Table
1
Load Program
ODI sends Confirmation and Row is Updated with "Transferred"
5
2
New Rows invokes ODI Package
ODI Architecture
ODI Invoke
Data Integrator
ODI populates the interface table First using KM
3
New Rows sent to Target Interface table
HQ ERP App
UI
App
Target Data
Interface Table
4
Load Program

High-volume transactions without an XREF table

If the HQ location does not need to perform DML operations, such as the case of a data warehouse, then maintaining the XREF is not necessary.

Building Oracle Data Integrator Projects

The bulk data processing strategy for Oracle AIA using ODI is about building a point-to-point solution, taking into account the need to set up data in the Xref using DVM, and ensuring that the data processed can participate in Oracle AIA services at runtime.

To develop ODI packages for Oracle AIA:

1. Define data servers.

   The source and target data servers defined are logical entities referring to the physical database schema chosen for bulk data processing.

   Each of the data servers defined should be linked to the physical database schemas.

   For more information, see *Oracle Data Integrator Reference Guide.*

2. Reverse engineer data servers.

   Reverse engineer the data server to generate the various models along with the data stores.

3. Define interfaces.

   For each data store, create interfaces as required.

In the process of creating interfaces, the mapping between the source and target fields will be specified.

4. Define packages.

   The packages are the steps in which the created interfaces are executed in addition to intermediate steps involving the Xref IKM and DVM.

   The package also includes steps to clean up the source tables.

   These are described in detail in these sections.

Using the XREF Knowledge Module

In ODI, the creation of XREF data is a two-step process. Each step is an interface.

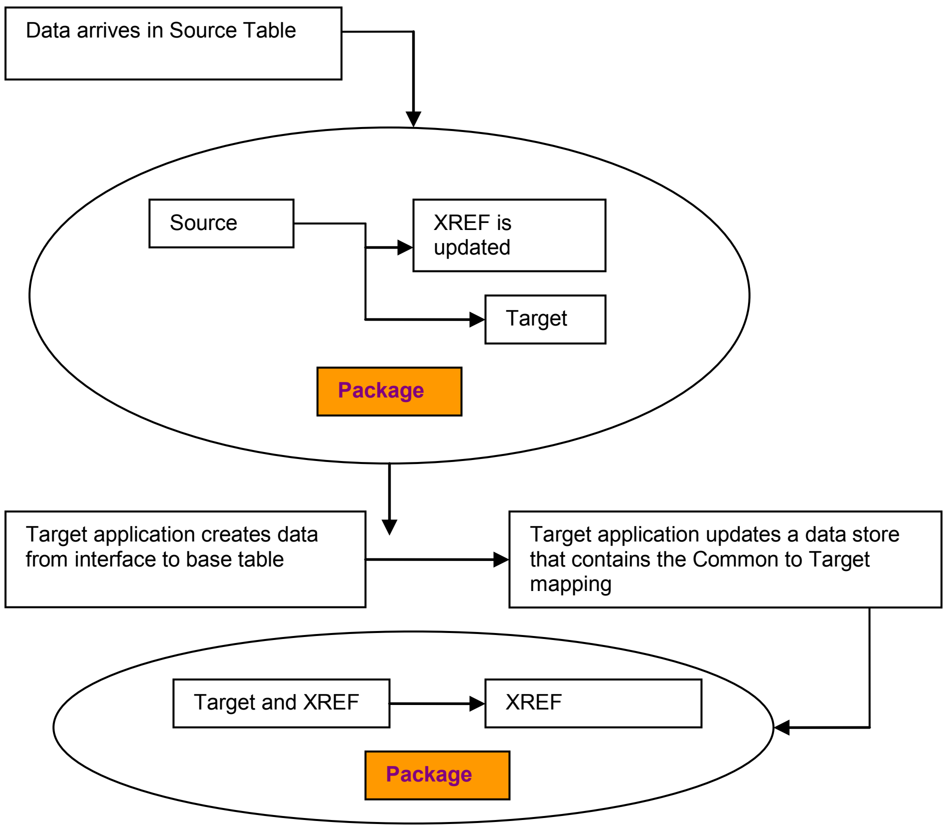
1. In the first interface, the user's source table is the source in ODI and the user's target table is the target in ODI.

   While transporting data from source to target table you 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 ODI 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.

2. 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 execution are available.

   A second interface must be executed where this data store is the source in ODI and XREF table is the target. This creates the appropriate target columns in the XREF table.


Data arrives in Source Table
Source
XREF is updated
Target
Package
Target application creates data from interface to base table
Target application updates a data store that contains the Common to Target mapping
Target and XREF
XREF
Package

XREF data creation flow

For more information, see *Oracle Data Integrator Knowledge Module Developer's Guide.*

Prerequisites

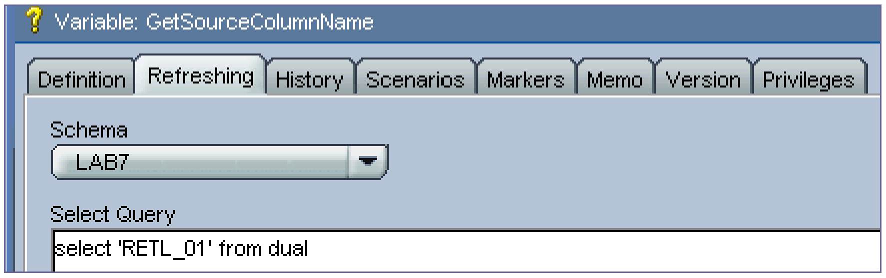
- Define the master and work repository.
- Define all topology parameters
  - Physical architecture (for source, target and XREF_DATA)
  - Logical architecture
  - Define a logical repository with the name "ESB_XREF" that points to the XREF database
  - Contexts
- Define your data models
- Create a project
- Import these Knowledge Modules into your project:
  - Loading Knowledge Module (LKM) SQL to SQL - Enterprise Service Bus (ESB) XREF

- LKM MSSQL to SQL (ESB XREF)
- Check Knowledge Module (CKM) Oracle (Shipped out of the box)
- (Integration Knowledge Module (IKM) SQL Control Append (ESB XREF)
- IKM Oracle Incremental Update (shipped out of the box)

Define Variables (Source and Target Column Names)

Since XREF column names cannot be hard coded, two variables must be defined to hold the source and target column names. Normally these column names will be derived from AIAConfiguration file. This document describes how to refresh this from a SQL “select” statement.

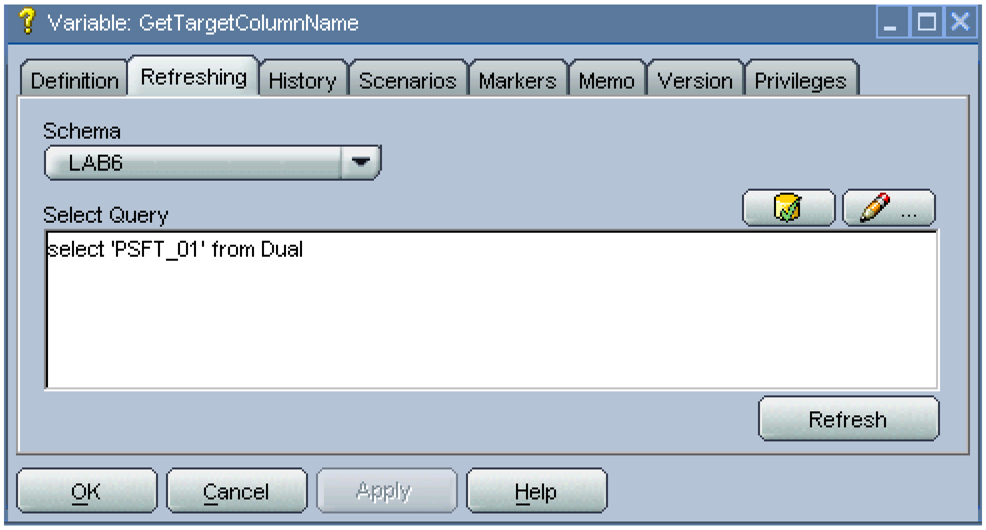
- Create a variable GetSourceColumnName: This variable will be used to determine the column name of the XREF_DATA table. The refreshing tab will have the appropriate SQL to get the value from the source depending on the implementation



Schema: LAB7

Select Query: select 'RETL_01' from dual

- Create a variable GetTargetColumnName: This variable will be used to determine the column name of the XREF_DATA table. The refreshing tab will have the appropriate SQL to get the value from the source depending on the implementation



Schema: LAB6

Select Query: select 'PSFT_01' from Dual

Creating the First Interface (Source to Target)

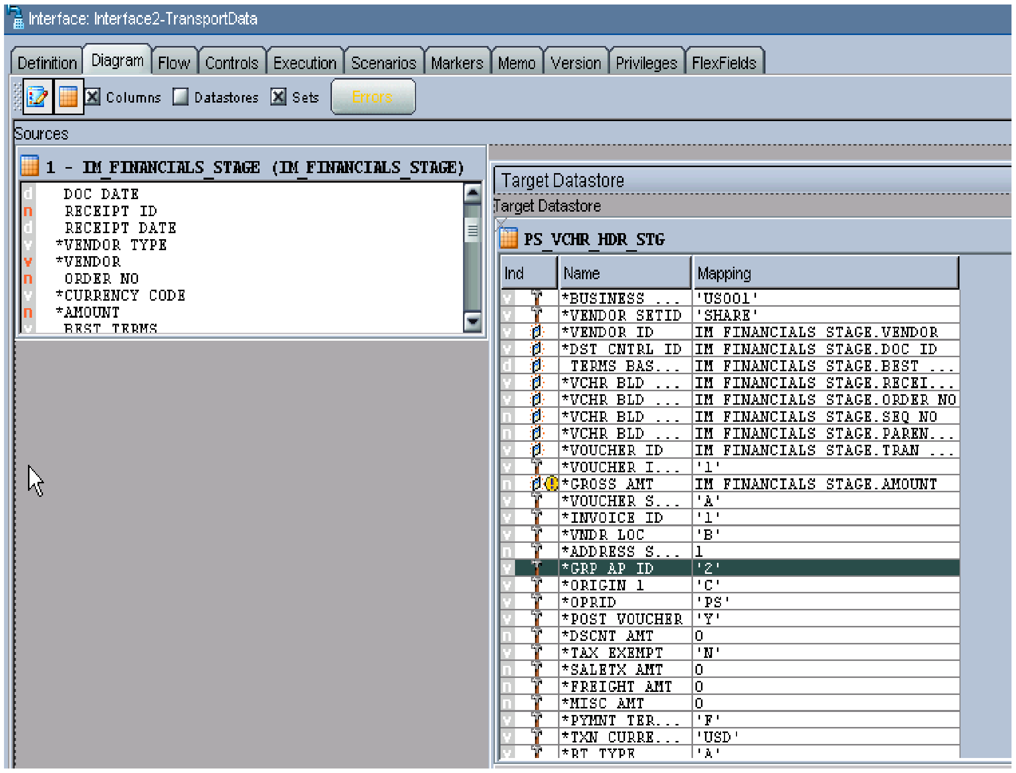
To create the first interface:

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) is in the target data store

   c. Provide mapping for each field.



Interface: Interface2-TransportData

Definition | Diagram | Flow | Controls | Execution | Scenarios | Markers | Memo | Version | Privileges | FlexFields

[✓ Columns] Datastores [✓ Sets] Errors

Sources

1 - IM_FINANCIALS_STAGE (IM_FINANCIALS_STAGE)

d	DOC DATE
n	RECEIPT ID
d	RECEIPT DATE
v	*VENDOR TYPE
v	*VENDOR
n	ORDER NO
v	*CURRENCY CODE
n	*AMOUNT
v	BEST TERMS

Target Datastore

Target Datastore

PS_VCHR_HDR_STG

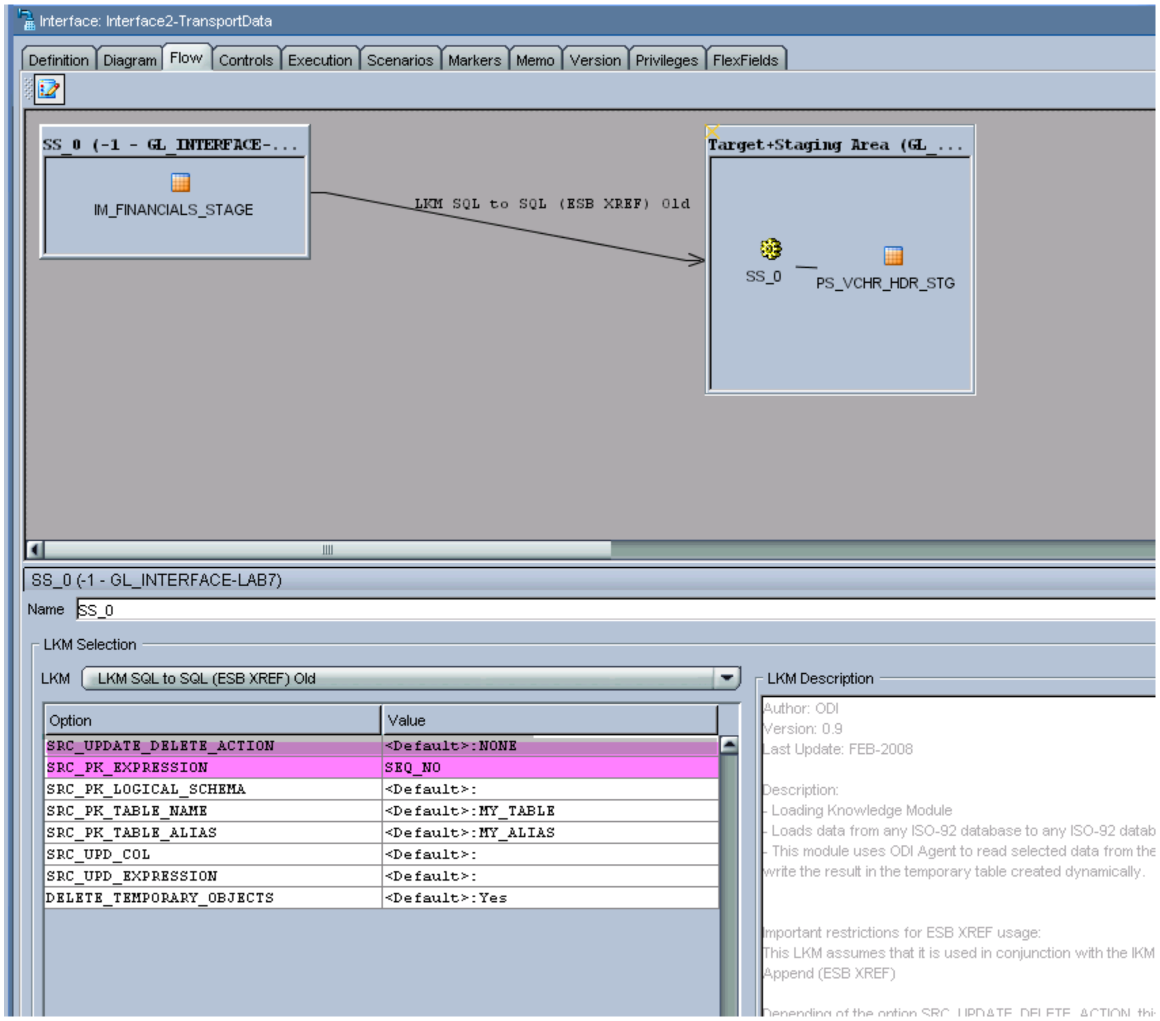
Ind	Name	Mapping
v	*BUSINESS ...	'US001'
v	*VENDOR SETID	'SHARE'
v	*VENDOR ID	IM FINANCIALS STAGE.VENDOR
v	*DST CNTRL ID	IM FINANCIALS STAGE.DOC ID
d	TERMS BAS...	IM FINANCIALS STAGE.BEST ...
v	*VCHR BLD ...	IM FINANCIALS STAGE.RECEI...
v	*VCHR BLD ...	IM FINANCIALS STAGE.ORDER NO
n	*VCHR BLD ...	IM FINANCIALS STAGE.SEQ NO
n	*VCHR BLD ...	IM FINANCIALS STAGE.PAREN...
v	*VOUCHER ID	IM FINANCIALS STAGE.TRAN ...
v	*VOUCHER I...	'1'
n	*GROSS AMT	IM FINANCIALS STAGE.AMOUNT
v	*VOUCHER S...	'A'
v	*INVOICE ID	'1'
v	*VNDR LOC	'B'
n	*ADDRESS S...	1
v	*GRP AP ID	'Z'
v	*ORIGIN 1	'C'
v	*OPRID	'PS'
v	*POST VOUCHER	'Y'
n	*DSCNT AMT	0
v	*TAX EXEMPT	'N'
n	*SALETX AMT	0
n	*FREIGHT AMT	0
n	*MISC AMT	0
v	*PYMNT TER...	'F'
v	*TXN CURRE...	'USD'
v	*RT TYPE	'A'

2. Go to the Flow tab.

3. Select LKM as the customized “LKM SQL to SQL (ESB XREF)” if you are using DB2 or Oracle as your source database.

   If your source database is MSSQL then use the customized knowledge module “LKM MSSQL to SQL (ESB XREF)”. This KM has an option called SOURCE_PK_EXPRESSION. You need to pass the expression that will represent the source key value that you want to store 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 then use the expression to be used to derive the key value.

For example if there are two key columns in the table and you want to store the concatenated value of those columns as your source value in XREF table then 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 then set the option SRC_UPDATE_DELETE_ACTION as "None".



SS_0 (-1 - GL_INTERFACE-LAB7)

Name: SS_0

LKM Selection

LKM: LKM SQL to SQL (ESB XREF) Old

Option	Value
SRC_UPDATE_DELETE_ACTION	<Default>:NONE
SRC_PK_EXPRESSION	SEQ_NO
SRC_PK_LOGICAL_SCHEMA	<Default>:
SRC_PK_TABLE_NAME	<Default>:MY_TABLE
SRC_PK_TABLE_ALIAS	<Default>:MY_ALIAS
SRC_UPD_COL	<Default>:
SRC_UPD_EXPRESSION	<Default>:
DELETE_TEMPORARY_OBJECTS	<Default>:Yes

LKM Description

Author: ODI
Version: 0.9
Last Update: FEB-2008

Description:
- Loading Knowledge Module
- Loads data from any ISO-92 database to any ISO-92 datab
- This module uses ODI Agent to read selected data from the
write the result in the temporary table created dynamically.

Important restrictions for ESB XREF usage:
This LKM assumes that it is used in conjunction with the IKM
Append (ESB XREF)

Depending of the option SRC_UPDATE_DELETE_ACTION thi

4. Choose CKM Oracle as you check Knowledge Module.

5. In the IKM choose the customized knowledge module IKM SQL Control Append XREF (Merge and Reuse).

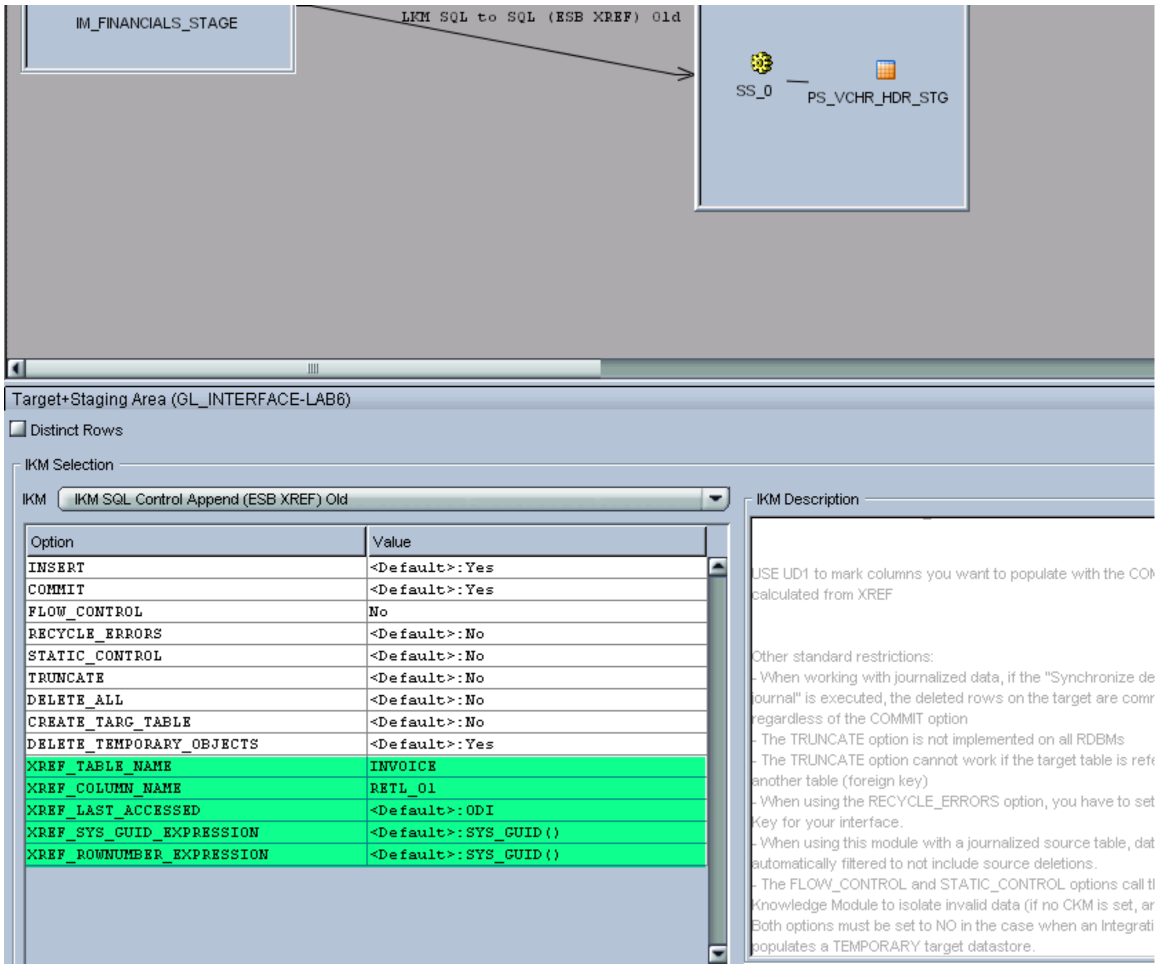
In this module define these options:

a. XREF_TABLE_NAME – Name of your XREF table

b. XREF_COLUMN_NAME – This is the name of the source column in XREF table. Enter the variable name you defined earlier (#GetSourceColumnName) in this option

c. XREF_SYS_GUID_EXPRESSION – Whether you want to use GUID or a sequence for the common identifier. For GUID use SYS_GUID for sequence use the sequence name in this option value

d. XREF_ROWNUMBER_EXPRESSION – This is the value that goes into the ROWNUMBER column of the XREF_DATA table. Use the default value of GUID unless you have the need to change it to a sequence.

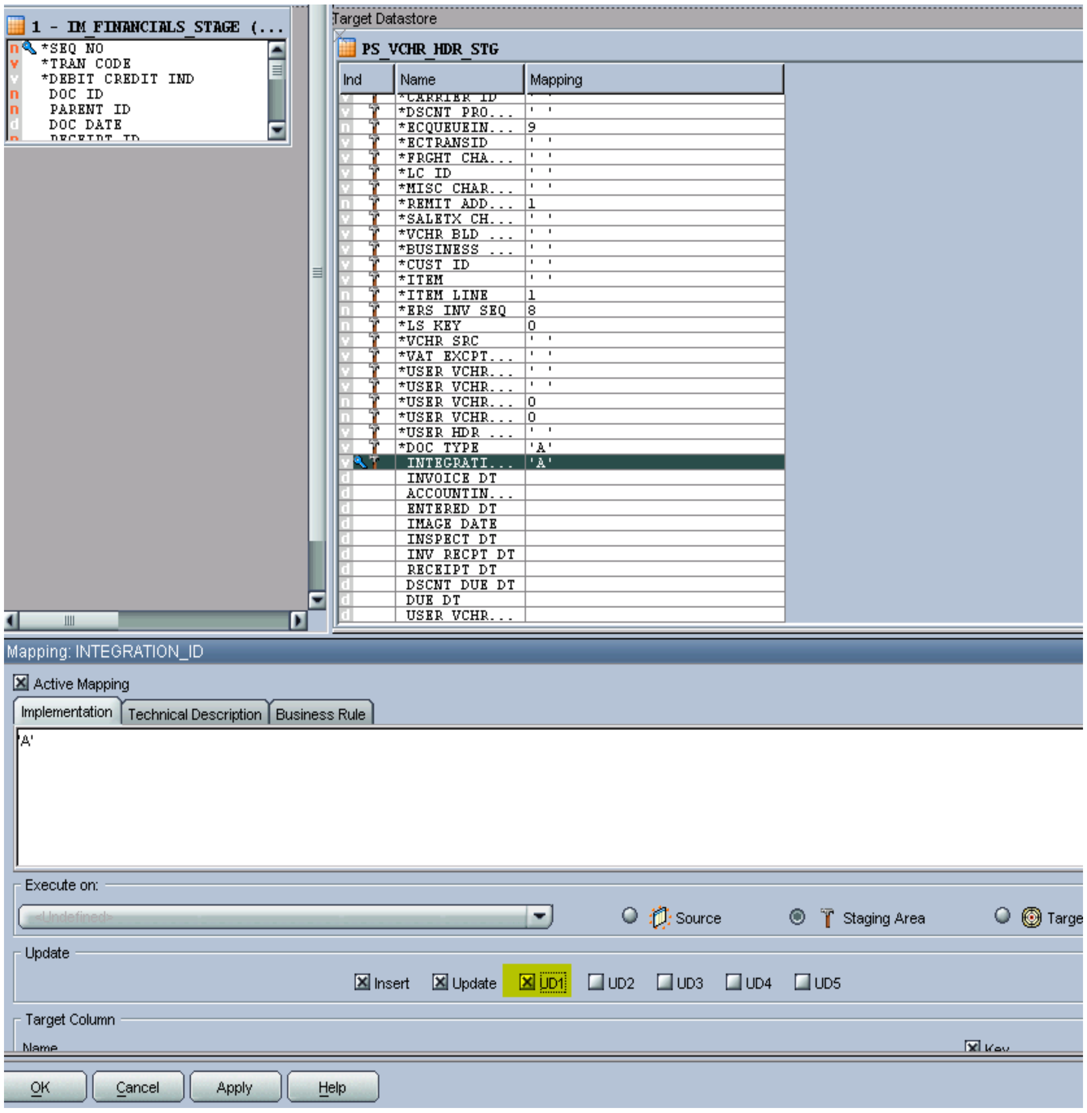

IM_FINANCIALS_STAGE
LKM SQL to SQL (ESB XREF) Old
SS_0
PS_VCHR_HDR_STG
Target+Staging Area (GL_INTERFACE-LAB6)
Distinct Rows
IKM Selection
IKM
IKM SQL Control Append (ESB XREF) Old
IKM Description

Option	Value
INSERT	<Default>:Yes
COMMIT	<Default>:Yes
FLOW_CONTROL	No
RECYCLE_ERRORS	<Default>:No
STATIC_CONTROL	<Default>:No
TRUNCATE	<Default>:No
DELETE_ALL	<Default>:No
CREATE_TARG_TABLE	<Default>:No
DELETE_TEMPORARY_OBJECTS	<Default>:Yes
XREF_TABLE_NAME	INVOICE
XREF_COLUMN_NAME	RETL_01
XREF_LAST_ACCESSED	<Default>:ODI
XREF_SYS_GUID_EXPRESSION	<Default>:SYS_GUID()
XREF_ROWNUMBER_EXPRESSION	<Default>:SYS_GUID()

USE UD1 to mark columns you want to populate with the CO
calculated from XREF

Other standard restrictions:
- When working with journalized data, if the "Synchronize de
journal" is executed, the deleted rows on the target are com
regardless of the COMMIT option
- The TRUNCATE option is not implemented on all RDBMs
- The TRUNCATE option cannot work if the target table is ref
another table (foreign key)
- When using the RECYCLE_ERRORS option, you have to set
Key for your interface.
- When using this module with a journalized source table, da
automatically filtered to not include source deletions.
- The FLOW_CONTROL and STATIC_CONTROL options call t
Knowledge Module to isolate invalid data (if no CKM is set, a
Both options must be set to NO in the case when an Integrat
populates a TEMPORARY target datastore.

Note: The option XREF_LAST_ACCESSED will be remove. Please disregard it.
Note: If the target table does not have any placeholder for the common identifier and you are planning to populate the **source** identifier in one of the columns, then you must use the standard mapping rules of ODI to indicate what source identifier to populate in which column.
Note: If the target column that you want to hold the common identifier is a unique key of the target table, then you must put a dummy mapping on that column. This is due to an ODI limitation otherwise the key will not be shown next time you open the interface. At runtime this dummy mapping will be overwritten with the generated common identifier by the integration knowledge module. Mark the UD1 column to indicate which target column where the column value will go.



1 - IM_FINANCIALS_STAGE (...

- n *SEQ NO
- v *TRAN CODE
- v *DEBIT CREDIT IND
- n DOC ID
- n PARENT ID
- d DOC DATE
- RECEIPT ID

Target Datastore

PS_VCHR_HDR_STG

Ind	Name	Mapping
v	*CARRIER ID	' '
v	*DSCNT PRO...	' '
n	*ECQUEUEIN...	9
v	*ECTRANSID	' '
v	*FRGHT CHA...	' '
v	*LC ID	' '
v	*MISC CHAR...	' '
n	*REMIT ADD...	1
v	*SALETX CH...	' '
v	*VCHR BLD ...	' '
v	*BUSINESS ...	' '
v	*CUST ID	' '
v	*ITEM	' '
n	*ITEM LINE	1
n	*ERS INV SEQ	8
n	*LS KEY	0
v	*VCHR SRC	' '
v	*VAT EXCPT...	' '
v	*USER VCHR...	' '
v	*USER VCHR...	' '
n	*USER VCHR...	0
n	*USER VCHR...	0
v	*USER HDR ...	' '
v	*DOC TYPE	'A'
v	INTEGRATI...	'A'
d	INVOICE DT	
d	ACCOUNTIN...	
d	ENTERED DT	
d	IMAGE DATE	
d	INSPECT DT	
d	INV RECPT DT	
d	RECEIPT DT	
d	DSCNT DUE DT	
d	DUE DT	
d	USER VCHR...	

Mapping: INTEGRATION_ID

☒ Active Mapping

Implementation | Technical Description | Business Rule

```
'A'
```

Execute on: <Undefined> ○ Source [● Staging Area] ○ Targe

Update: ☒ Insert ☒ Update ☒ UD1 ☐ UD2 ☐ UD3 ☐ UD4 ☐ UD5

Target Column

Name ☒ Key

If you have no need to send the common value to the target table then you can ignore this step.

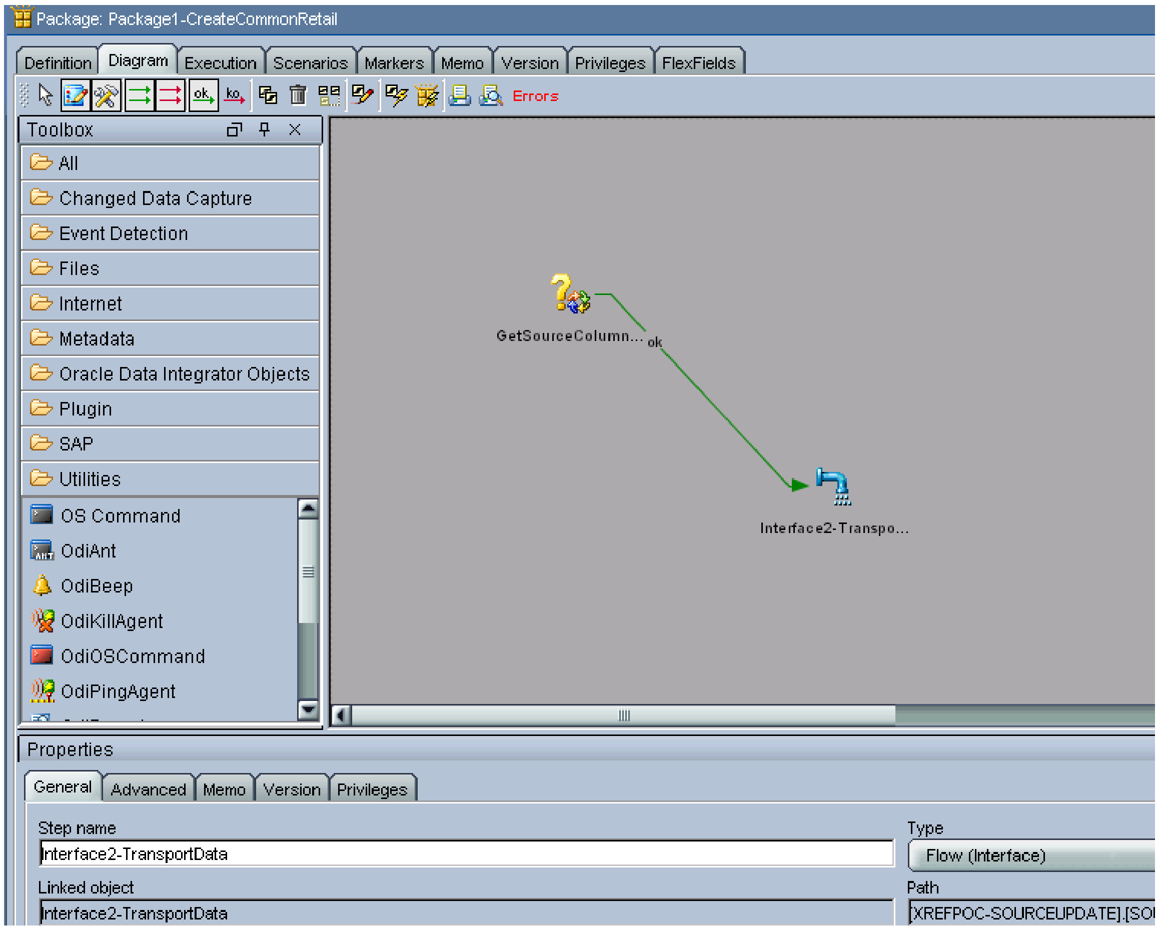
6. Validate and save the interface.

Creating the Package

To create a package to execute the interfaces:

This should contain at least two steps:

1. Refresh the variable that holds the source column name.
2. Execute the interface.



Step name: Interface2-TransportData

Type: Flow (Interface)

Linked object: Interface2-TransportData

Path: [XREFPOC-SOURCEUPDATE].[SO

Note: Every implementation will add its own error handling steps in the package.

3. Validate and save the package.

4. Execute the package.

   Normally, this package will be executed as soon as the data arrives in the source table. This can be achieved by using the ODI changed data capture. These sections describe how to execute a package when the data arrives in a source table.

Defining the XREF View in SOA

Create an XREF View in the XREF database:

```
  CREATE OR REPLACE FORCE VIEW "ORAESB"."INVOICE_XREF_VW"
("ROW_NUMBER", "XREF_TABLE_NAME", "RETL_01", "COMMON", "PSFT_01") AS
  select row_number, XREF_TABLE_NAME,
  max(decode(XREF_COLUMN_NAME, 'RETL_01', VALUE,null)) RETL_01,
  max(decode(XREF_COLUMN_NAME, 'COMMON', VALUE,null)) COMMON,
  max(decode(XREF_COLUMN_NAME, 'PSFT_01', VALUE,null)) PSFT_01
  from XREF_DATA
GROUP BY row_number, XREF_TABLE_NAME;
```

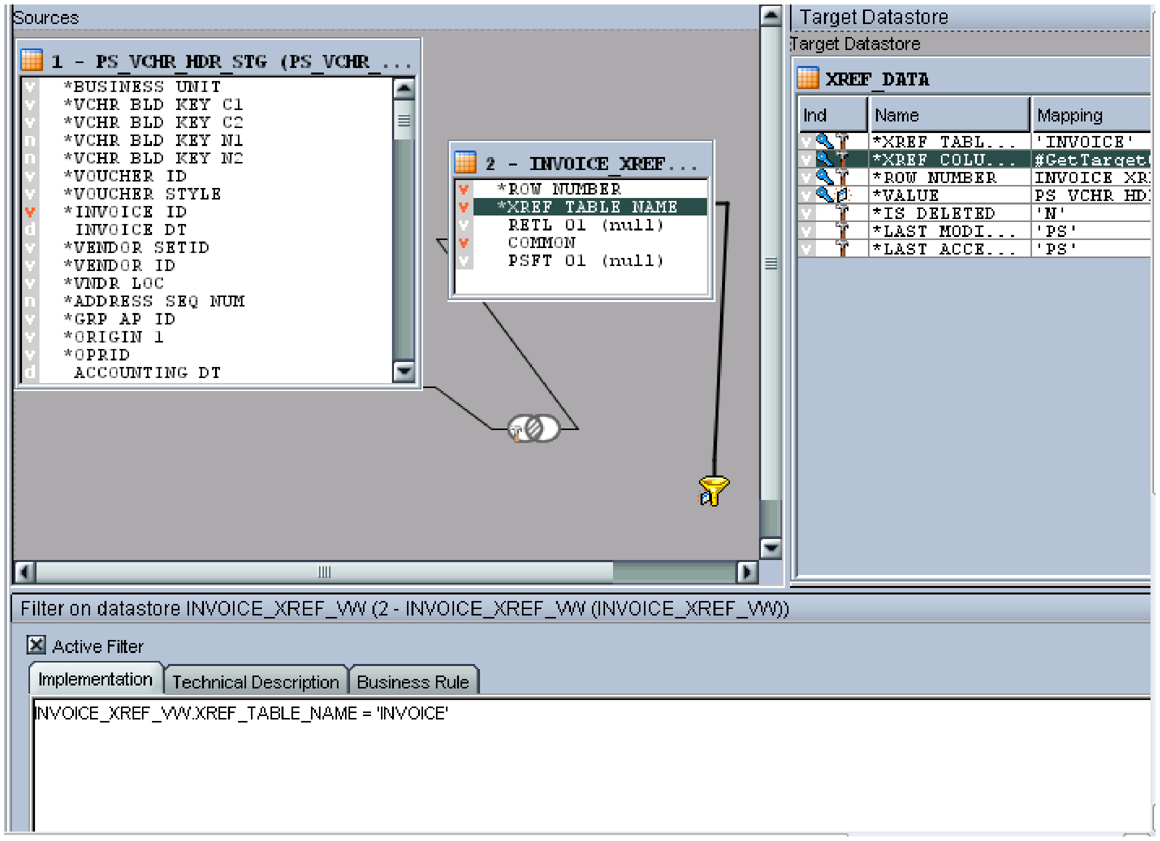
Note: Construct this view for each implementation.

Creating the Second Interface (Update Target Identifier in XREF)

Once the data is moved to target base tables and the target identifier is created, the identifier needs to get back to 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 this may come in a separate table. This mapping data store will be used in this interface in the source. This interface will be packaged and finally a separate process from target system will execute that package.

- Create an interface for data transport. In the sources section drop 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.

- Apply a filter for XREF_VW with a where clause to filter data from your table name only, for example, "XREF_VW.XREF_TABLE_NAME='INVOICE'" if you are using this for INVOICE



1 - PS VCHR HDR STG (PS VCHR ...

```
*BUSINESS UNIT
*VCHR BLD KEY C1
*VCHR BLD KEY C2
*VCHR BLD KEY N1
*VCHR BLD KEY N2
*VOUCHER ID
*VOUCHER STYLE
*INVOICE ID
 INVOICE DT
*VENDOR SETID
*VENDOR ID
*VNDR LOC
*ADDRESS SEQ NUM
*GRP AP ID
*ORIGIN 1
*OPRID
 ACCOUNTING DT
```

2 - INVOICE XREF...

```
*ROW NUMBER
*XREF TABLE NAME
 RETL 01 (null)
 COMMON
 PSFT 01 (null)
```

XREF DATA

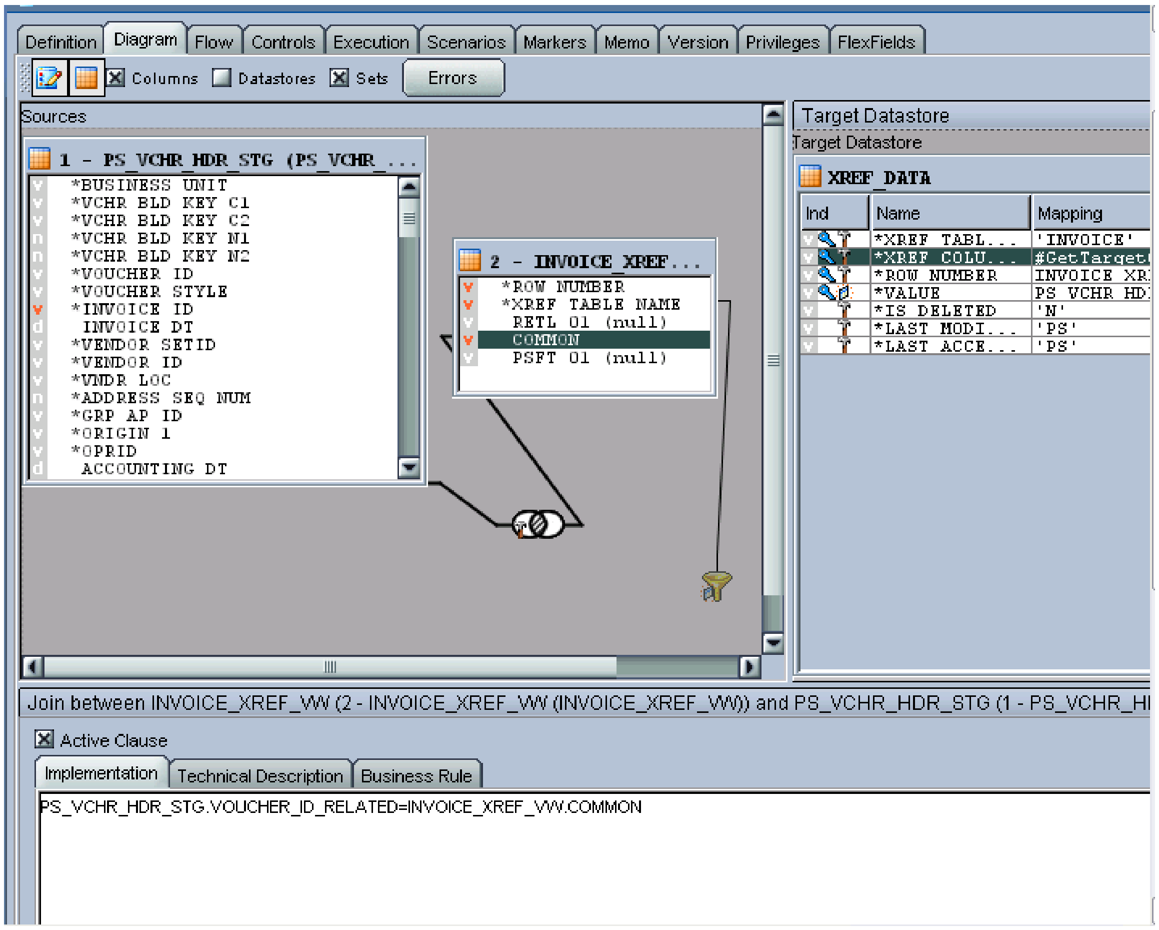
Ind	Name	Mapping
	*XREF TABL...	'INVOICE'
	*XREF COLU...	#GetTarget
	*ROW NUMBER	INVOICE XR
	*VALUE	PS VCHR HD
	*IS DELETED	'N'
	*LAST MODI...	'PS'
	*LAST ACCE...	'PS'

Filter on datastore INVOICE_XREF_VW (2 - INVOICE_XREF_VW (INVOICE_XREF_VW))

[✓] Active Filter

```
INVOICE_XREF_VW.XREF_TABLE_NAME = 'INVOICE'
```

- Join mapping data store and XREF_VW with the columns that store common (or source) identifier. In this example the column of the PeopleSoft interface table that stores common data is VOUCHER_ID_RELATED.



Ind	Name	Mapping
	*XREF TABL...	'INVOICE'
	*XREF COLU...	#GetTarget
	*ROW NUMBER	INVOICE XR
	*VALUE	PS VCHR HD
	*IS DELETED	'N'
	*LAST MODI...	'PS'
	*LAST ACCE...	'PS'

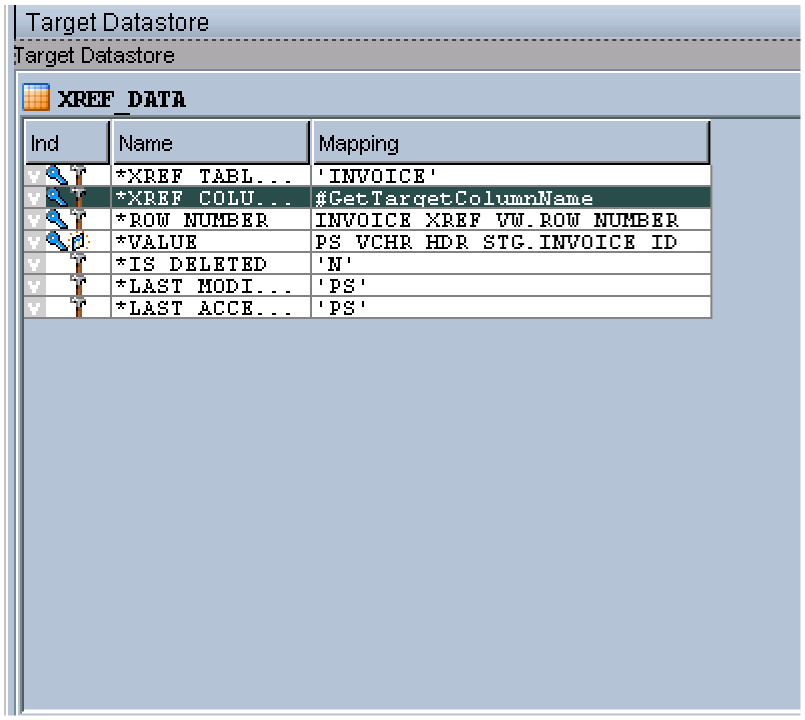
Join between INVOICE_XREF_VW (2 - INVOICE_XREF_VW (INVOICE_XREF_VW)) and PS_VCHR_HDR_STG (1 - PS_VCHR_H

☒ Active Clause

Implementation | Technical Description | Business Rule

```
PS_VCHR_HDR_STG.VOUCHER_ID_RELATED=INVOICE_XREF_VW.COMMON
```

- The XREF_TABLE_NAME map should be the XREF_TABLE name of the implementation.
- XREF_COLUMN_NAME map should be “#GetTargetColumnName” (pointing to the variable that was created earlier).
- Map ROW_NUMBER with the ROW_NUMBER of the XREF_VW.
- The map for VALUE field will be the column that stores the target identifier in the mapping data store (in this example INVOICE_ID column of the PS_VCHR_HDR_STG).
- Map for IS_DELETED is set to ‘N’.
- Map for LAST_MODIFIED and LAST_ACCESSED will be different for each implementation.
- Mark XREF_TABLE_NAME, XREF_COLUMN_NAME and VALUE as Key.



Ind	Name	Mapping
	*XREF_TABL...	'INVOICE'
	*XREF_COLU...	#GetTargetColumnName
	*ROW_NUMBER	INVOICE_XREF_VW.ROW_NUMBER
	*VALUE	PS_VCHR_HDR_STG.INVOICE_ID
	*IS_DELETED	'N'
	*LAST_MODI...	'PS'
	*LAST_ACCE...	'PS'

- In the flow tab use load knowledge module as “LKM SQL to Oracle”.
- Use integration knowledge module as “IKM Oracle incremental update”.
- In the controls tab use check knowledge module as “CKM Oracle”.
- Validate and save the interface.

Handling Update – Delete in Source Interfaces

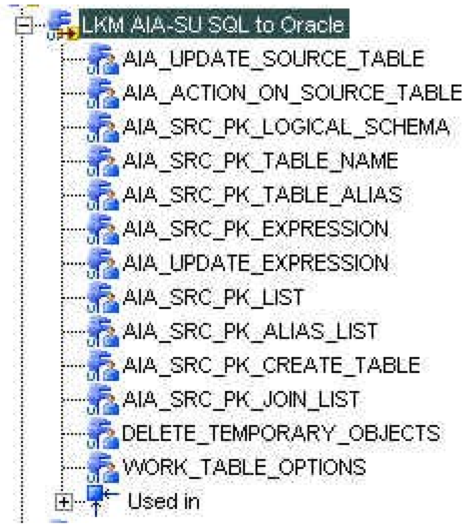
The existing LKM needs to be modified. LKM first loads the data from the source table to the staging tables on the target system side. Then target table will be loaded from the staging area. The idea here is to create a temporary table on the source side with columns being the PK of the source table. Then update source table data using this temporary table. These are the steps to be incorporated into the original LKM.

1. Create a temporary relation on source schema with columns being PK of source table
2. Populate this temp relation first before the actual Load data step
3. (Load data step populates the staging area on target system from source table)
4. Modify Load data step to load only those tables which were inserted into temp table
5. Delete the rows from source table. (Post integration)
6. Delete temp relation. (Post integration)

ODI Step Details

The LKM being customized for Oracle AIA is “**LKM SQL to Oracle**”

These are the options that were created under this LKM to generalize the solution.



LKM AIA-SU SQL to Oracle
AIA_UPDATE_SOURCE_TABLE
AIA_ACTION_ON_SOURCE_TABLE
AIA_SRC_PK_LOGICAL_SCHEMA
AIA_SRC_PK_TABLE_NAME
AIA_SRC_PK_TABLE_ALIAS
AIA_SRC_PK_EXPRESSION
AIA_UPDATE_EXPRESSION
AIA_SRC_PK_LIST
AIA_SRC_PK_ALIAS_LIST
AIA_SRC_PK_CREATE_TABLE
AIA_SRC_PK_JOIN_LIST
DELETE_TEMPORARY_OBJECTS
WORK_TABLE_OPTIONS
Used in

Option	Description
AIA_UPDATE_SOURCE_TABLE	Set this option to "Yes" if you wish to update/delete rows of a source table. When this option is set to "No" then this LKM behaves in the original way as proposed by ODI product. So when you do not want to update/delete source table data, all you need to do when using this LKM is to simply set this option value to “No”.
AIA_ACTION_ON_SOURCE_TABLE	Indicate the action to be performed on the source table. Possible values are either "Delete" or "Update". Default value is "Delete". If the option value is "Update" then the other option AIA_UPDATE_EXPRESSION must be set to appropriate value.
AIA_SRC_PK_LOGICAL_SCHEMA	Indicate the source table's logical schema. The source table is the one from which we want to delete records after processing them. This logical schema is used to resolve the actual physical schema at runtime depending on the Context.
AIA_SRC_PK_TABLE_NAME	Indicate the source table name of which we want to delete records after processing them
AIA_SRC_PK_TABLE_ALIAS	Indicate the source table's alias within this interface. The source table is the one from which we want to delete records after processing them.
AIA_SRC_PK_EXPRESSION	Expressions that concatenate values from the PK to have them fit in a single large varchar column. Example for the source Orderline Table (aliased OLINE in the interface) you can use expression: TO_CHAR(OLINE.ORDER_ID) || '-' || TO_CHAR(OLINE.LINE_ID)

Option	Description

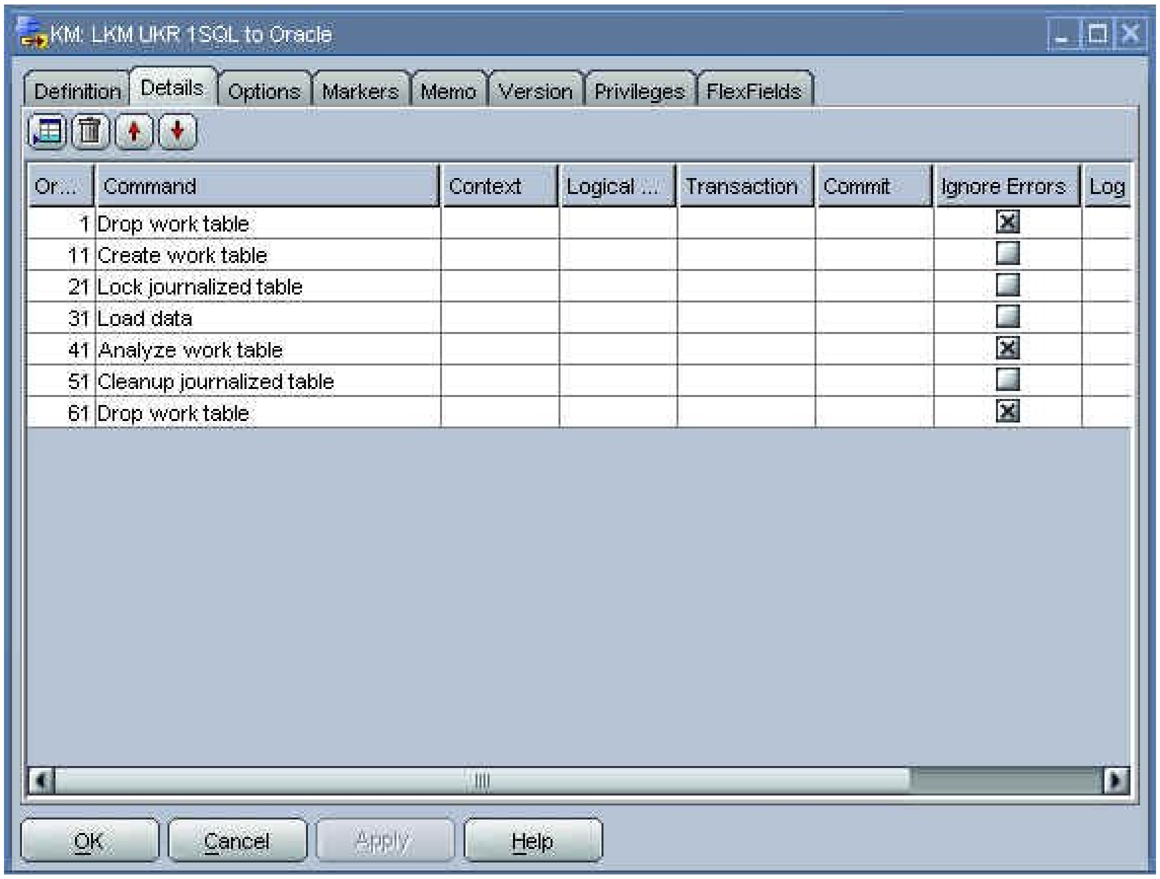
AIA_UPDATE_EXPRESSION	This indicates the expression for the update statement's SET value.
	
	Example:
	
	UPDATE EMPLOYEE E1
	
	SET E1.column1 = value1,
	
	    E1.column2 = value2
	
	WHERE E1.ENO = 101
	
	The value of this option could be
	
	E1.column1 = value1,
	
	E1.column2 = valule2
	
	In the update statement

AIA_SRC_PK_LIST	Mention PK columns separated by comma Syntax.: column1, column2
AIA_SRC_PK_ALIAS_LIST	Mention PK columns prefixed with table alias name separated by comma. Syntax: alias.column1, alias.column2
AIA_SRC_PK_CREATE_TABLE	Mention the PK columns with datatypes and size separated by comma as it occurs in create table SQL statement Syntax: column1 datatype(size) not null, column2 datatype(size) not null
AIA_SRC_PK_JOIN_LIST	Mention the join condition for the PK columns. Choose the same alias that you have chosen for the AIA_SRC_PK_TABLE_ALIAS. On the other side always choose STT as the alias name. Syntax: alias.column1 = stt.column1 and alias.column2 = stt.column2

LKM Steps

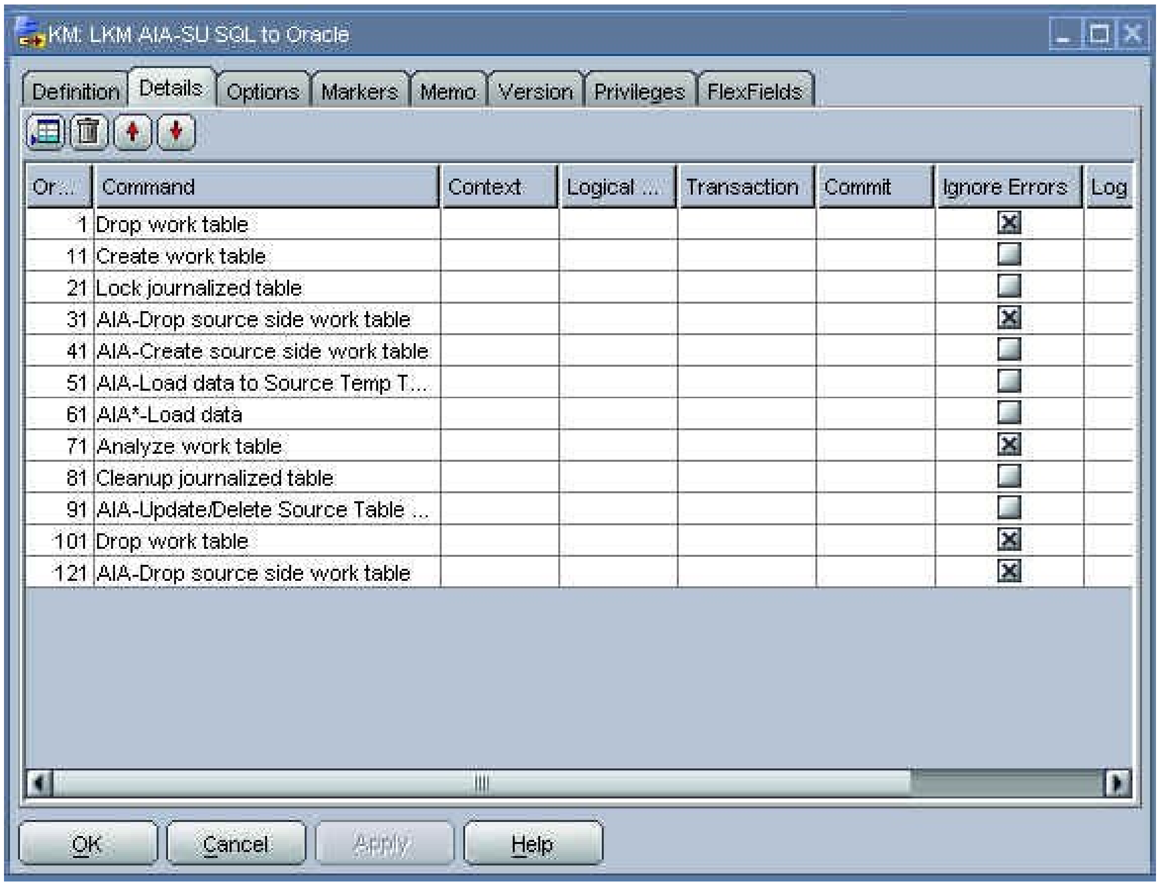
Some steps were added to the LKM in order to update the source table data.

The original LKM steps were:


KM: LKM UKR 1SQL to Oracle

Or...	Command	Context	Logical ...	Transaction	Commit	Ignore Errors	Log
1	Drop work table					☒	
11	Create work table					☐	
21	Lock journalized table					☐	
31	Load data					☐	
41	Analyze work table					☒	
51	Cleanup journalized table					☐	
61	Drop work table					☒	

The customized LKM steps for Oracle AIA are:



Or...	Command	Context	Logical ...	Transaction	Commit	Ignore Errors	Log
1	Drop work table					☒	
11	Create work table					☐	
21	Lock journalized table					☐	
31	AIA-Drop source side work table					☒	
41	AIA-Create source side work table					☐	
51	AIA-Load data to Source Temp T...					☐	
61	AIA*-Load data					☐	
71	Analyze work table					☒	
81	Cleanup journalized table					☐	
91	AIA-Update/Delete Source Table ...					☐	
101	Drop work table					☒	
121	AIA-Drop source side work table					☒	

Changes to the LKM are described in this table:

Step Name	Launching	New/Original
Drop work table	Pre-integration (IKM)	Original
Create work table	Pre-integration (IKM)	Original
Lock journalized table	Pre-integration (IKM)	Original
AIA-Drop source side work table	Pre-integration (IKM)	New
AIA-Create source side work table	Pre-integration (IKM)	New
AIA-Load data to Source Temp Table	Pre-integration (IKM)	New
AIA*-Load data	Pre-integration (IKM)	Modified
Analyze work table	Pre-integration (IKM)	Original
Cleanup journalized table	Post-integration (IKM)	Original
AIA-Update/Delete Source Table Data	Post-integration (IKM)	New
Drop work table	Post-integration (IKM)	Original
AIA-Drop source side work table	Post-integration (IKM)	New

Examples

This section provides examples.

Problem 1

Source table: Semp2

Seno_part1	Seno_part2	Sename	Saddress	Sflag
101	1	Uday	Hyderabad	NEW
101	2	Kiran	Miyapur	NEW
101	3	Reddy	Nizampet	NEW

Primary Key: (Seno_part1, Seno_part2)

Target table: Temp2

Teno_part1	Teno_part2	Tename	Taddress

The mapping is straight mapping. Once the data is loaded from source table to target table, **we would like to delete the rows** from the source table.

Solution

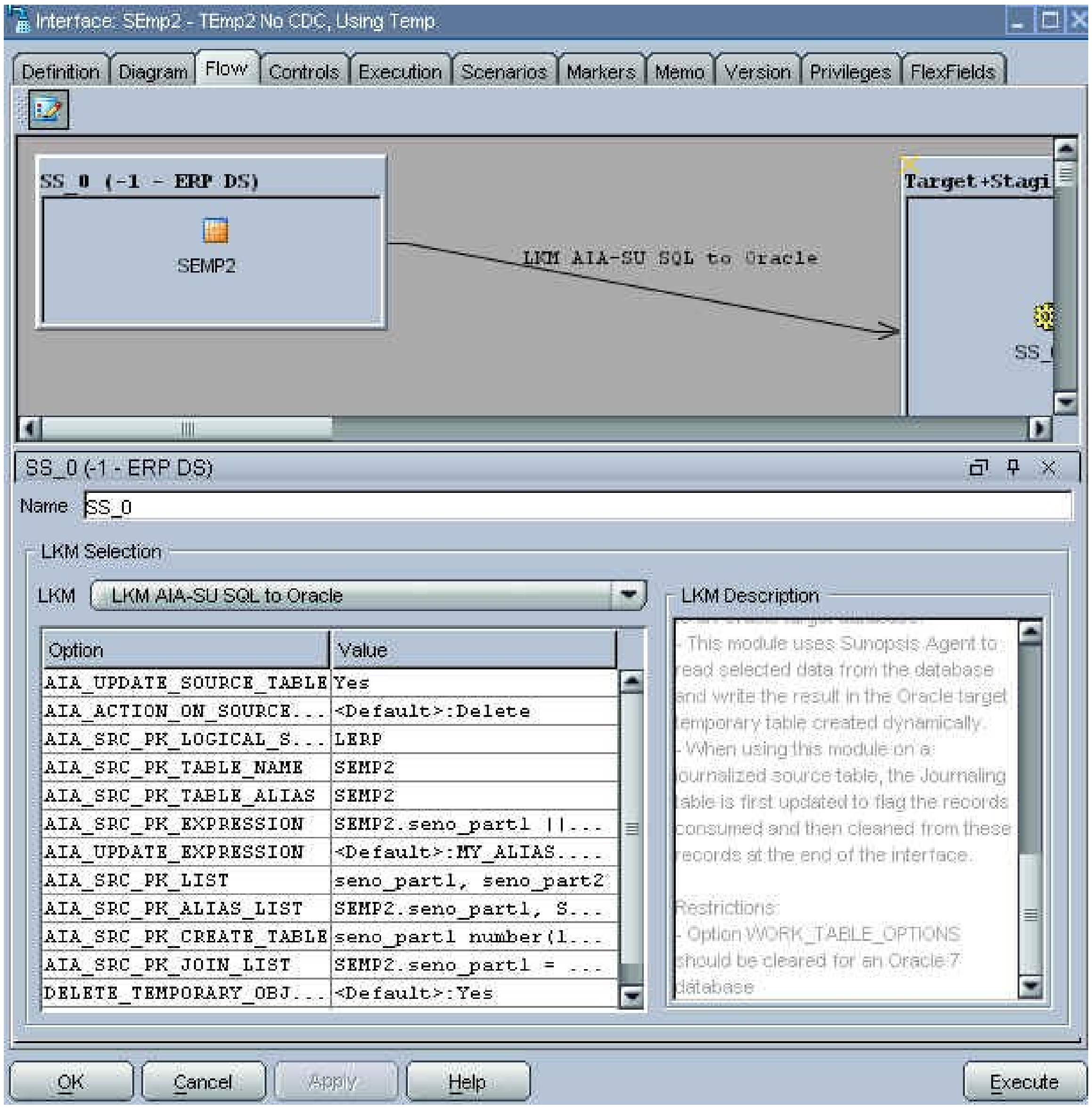
The mapping is a straight mapping.

Use the modified LKM: **LKM AIA-SU SQL to Oracle**

Set the options values for this LKM as:

Option	Value
AIA_UPDATE_SOURCE_TABLE	Yes
AIA_ACTION_ON_SOURCE_TABLE	Delete
AIA_SRC_PK_LOGICAL_SCHEMA	LERP
AIA_SRC_PK_TABLE_NAME	SEMP2
AIA_SRC_PK_TABLE_ALIAS	SEMP2
AIA_SRC_PK_EXPRESSION	SEMP2.SENO_PART1 || '-' || SEMP2.SENO_PART2
AIA_UPDATE_EXPRESSION	Not Applicable
AIA_SRC_PK_LIST	seno_part1, seno_part2
AIA_SRC_PK_ALIAS_LIST	SEMP2.seno_part1, SEMP2.seno_part2
AIA_SRC_PK_CREATE_TABLE	seno_part1 number(12) not null, seno_part2 number(12) not null

Option	Value
AIA_SRC_PK_JOIN_LIST	SEMP2.seno_part1 = stt.seno_part1 And SEMP2.seno_part2 = stt.seno_part2



Name: SS_0

LKM Selection

LKM: LKM AIA-SU SQL to Oracle

Option	Value
AIA_UPDATE_SOURCE_TABLE	Yes
AIA_ACTION_ON_SOURCE...	<Default>:Delete
AIA_SRC_PK_LOGICAL_S...	LERP
AIA_SRC_PK_TABLE_NAME	SEMP2
AIA_SRC_PK_TABLE_ALIAS	SEMP2
AIA_SRC_PK_EXPRESSION	SEMP2.seno_part1 ||...
AIA_UPDATE_EXPRESSION	<Default>:MY_ALIAS....
AIA_SRC_PK_LIST	seno_part1, seno_part2
AIA_SRC_PK_ALIAS_LIST	SEMP2.seno_part1, S...
AIA_SRC_PK_CREATE_TABLE	seno_part1 number(1...
AIA_SRC_PK_JOIN_LIST	SEMP2.seno_part1 = ...
DELETE_TEMPORARY_OBJ...	<Default>:Yes

LKM Description

- This module uses Sunopsis Agent to read selected data from the database and write the result in the Oracle target temporary table created dynamically.
- When using this module on a journalized source table, the Journaling table is first updated to flag the records consumed and then cleaned from these records at the end of the interface.

Restrictions:
- Option WORK_TABLE_OPTIONS should be cleared for an Oracle 7 database

Result

Source table: Semp2

Seno_part1	Seno_part2	Sename	Saddress	Sflag

Target table: Temp2

Teno_part1	Teno_part2	Tename	Taddress
101	1	Uday	Hyderabad
101	2	Kiran	Miyapur
101	3	Reddy	Nizampet

Source table data (rows) are deleted.

Problem 2

Source table: Semp2

Seno_part1	Seno_part2	Sename	Saddress	Sflag
101	1	Uday	Hyderabad	NEW
101	2	Kiran	Miyapur	NEW
101	3	Reddy	Nizampet	NEW

Primary Key: (Seno_part1, Seno_part2)

Target table: Temp2

Teno_part1	Teno_part2	Tename	Taddress

The mapping is straight mapping. Once the data is loaded from source table to target table, **we would like to update the rows** from the source table as: Sflag column value should be “Loaded” and Saddress column should be appended with “India”.

Solution

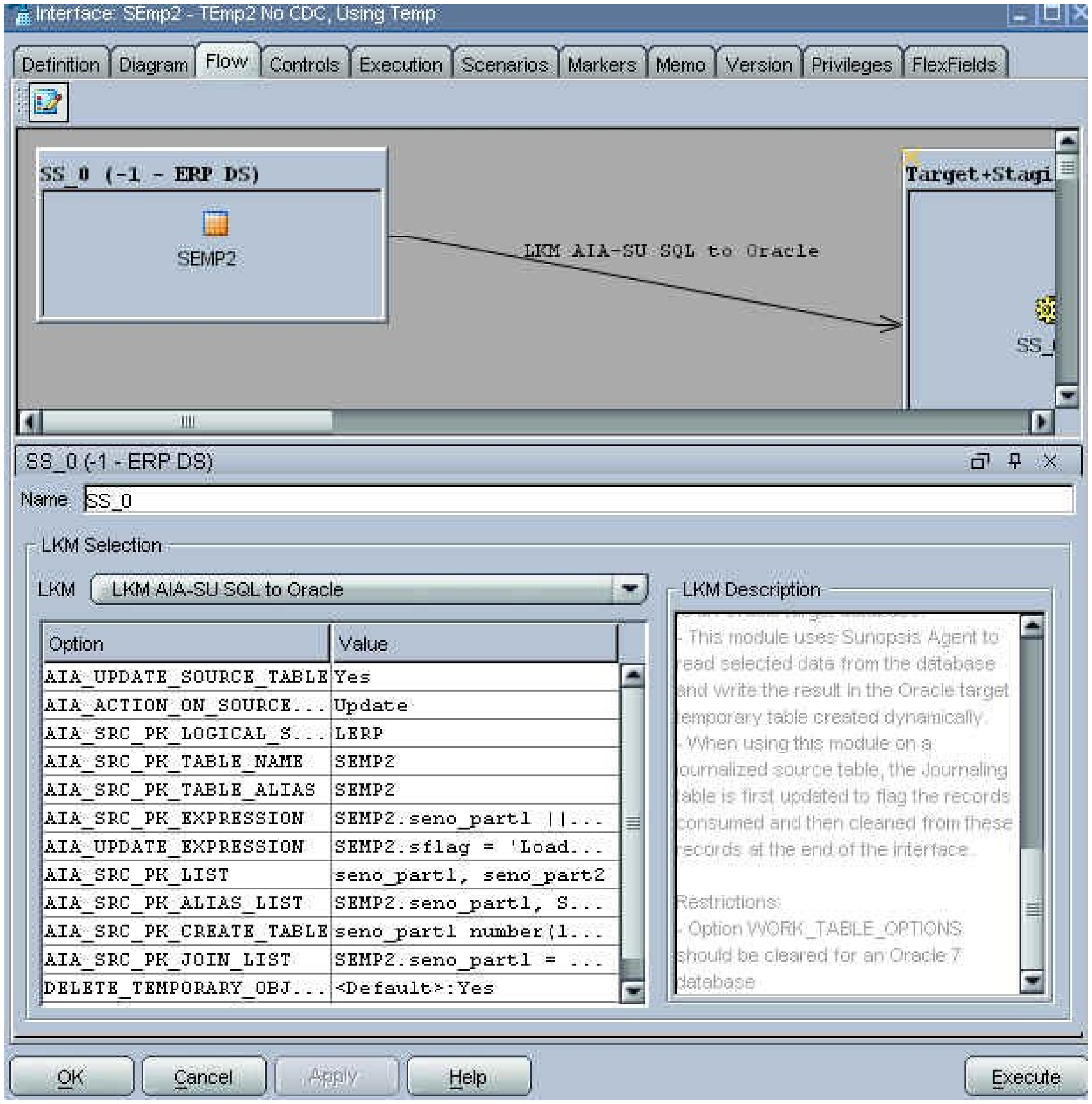
The mapping is a straight mapping.

Use the same modified LKM: **LKM AIA-SU SQL to Oracle**

Set the options values for this LKM as:

Option	Value
AIA_UPDATE_SOURCE_TABLE	Yes
AIA_ACTION_ON_SOURCE_TABLE	Update
AIA_SRC_PK_LOGICAL_SCHEMA	LERP
AIA_SRC_PK_TABLE_NAME	SEMP2
AIA_SRC_PK_TABLE_ALIAS	SEMP2
AIA_SRC_PK_EXPRESSION	SEMP2.SENO_PART1 || '-' || SEMP2.SENO_PART2

Option	Value
AIA_UPDATE_EXPRESSION	SEMP2.sflag = 'Loaded', SEMP2.saddress = SEMP2.saddress || ', India'
AIA_SRC_PK_LIST	seno_part1, seno_part2
AIA_SRC_PK_ALIAS_LIST	SEMP2.seno_part1, SEMP2.seno_part2
AIA_SRC_PK_CREATE_TABLE	seno_part1 number(12) not null, seno_part2 number(12) not null
AIA_SRC_PK_JOIN_LIST	SEMP2.seno_part1 = stt.seno_part1 And SEMP2.seno_part2 = stt.seno_part2



Name: SS_0

LKM Selection

LKM: LKM AIA-SU SQL to Oracle

Option	Value
AIA_UPDATE_SOURCE_TABLE	Yes
AIA_ACTION_ON_SOURCE...	Update
AIA_SRC_PK_LOGICAL_S...	LERP
AIA_SRC_PK_TABLE_NAME	SEMP2
AIA_SRC_PK_TABLE_ALIAS	SEMP2
AIA_SRC_PK_EXPRESSION	SEMP2.seno_part1 ||...
AIA_UPDATE_EXPRESSION	SEMP2.sflag = 'Load...
AIA_SRC_PK_LIST	seno_part1, seno_part2
AIA_SRC_PK_ALIAS_LIST	SEMP2.seno_part1, S...
AIA_SRC_PK_CREATE_TABLE	seno_part1 number(1...
AIA_SRC_PK_JOIN_LIST	SEMP2.seno_part1 = ...
DELETE_TEMPORARY_OBJ...	<Default>:Yes

LKM Description

- This module uses Sunopsis Agent to read selected data from the database and write the result in the Oracle target temporary table created dynamically.
- When using this module on a journalized source table, the Journaling table is first updated to flag the records consumed and then cleaned from these records at the end of the interface.

Restrictions:
- Option WORK_TABLE_OPTIONS should be cleared for an Oracle 7 database

Result

Source table: Semp2

Seno_part1	Seno_part2	Sename	Saddress	Sflag
101	1	Uday	Hyderabad, India	Loaded
101	2	Kiran	Miyapur, India	Loaded
101	3	Reddy	Nizampet, India	Loaded

Target table: Temp2

Teno_part1	Teno_part2	Tename	Taddress
101	1	Uday	Hyderabad
101	2	Kiran	Miyapur
101	3	Reddy	Nizampet

Source table data (rows) are updated. Two columns are updated.

Problem 3

Source table: Semp2

Seno_part1	Seno_part2	Sename	Saddress	Sflag
101	1	Uday	Hyderabad	NEW
101	2	Kiran	Miyapur	NEW
101	3	Reddy	Nizampet	NEW

Primary Key: (Seno_part1, Seno_part2)

Target table: Temp2

Teno_part1	Teno_part2	Tename	Taddress

The mapping is straight mapping. Once the data is loaded from source table to target table, **we would not like to touch the source table.** That is, it should behave in the same manner as the original LKM.

Solution

The mapping is a straight mapping.

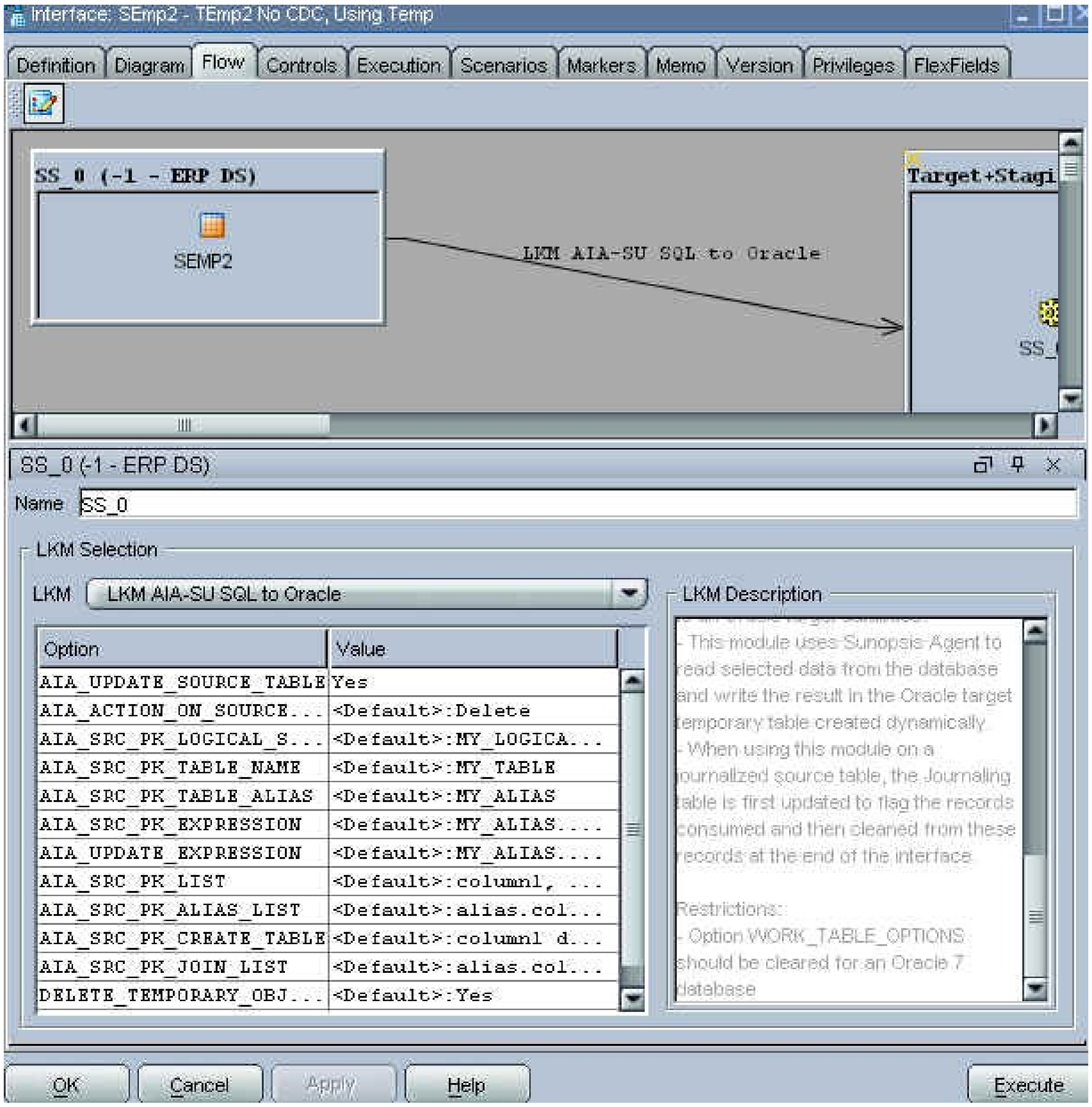
Original LKM (**LKM SQL to Oracle**) can be used for this.

We can achieve this even with the modified LKM: **LKM AIA-SU SQL to Oracle**

This solution is how to do that using the modified LKM.

Set the options values for this LKM as:

Option	Value
AIA_UPDATE_SOURCE_TABLE	No
AIA_ACTION_ON_SOURCE_TABLE	Not applicable
AIA_SRC_PK_LOGICAL_SCHEMA	Not applicable
AIA_SRC_PK_TABLE_NAME	Not applicable
AIA_SRC_PK_TABLE_ALIAS	Not applicable
AIA_SRC_PK_EXPRESSION	Not applicable
AIA_UPDATE_EXPRESSION	Not applicable
AIA_SRC_PK_LIST	seno_part1, seno_part2
AIA_SRC_PK_ALIAS_LIST	SEMP2.seno_part1, SEMP2.seno_part2
AIA_SRC_PK_CREATE_TABLE	seno_part1 number(12) not null, seno_part2 number(12) not null
AIA_SRC_PK_JOIN_LIST	SEMP2.seno_part1 = stt.seno_part1 And SEMP2.seno_part2 = stt.seno_part2



SS_0 (-1 - ERP DS)

Name: SS_0

LKM Selection

LKM: LKM AIA-SU SQL to Oracle

Option	Value
AIA_UPDATE_SOURCE_TABLE	Yes
AIA_ACTION_ON_SOURCE...	<Default>:Delete
AIA_SRC_PK_LOGICAL_S...	<Default>:MY_LOGICA...
AIA_SRC_PK_TABLE_NAME	<Default>:MY_TABLE
AIA_SRC_PK_TABLE_ALIAS	<Default>:MY_ALIAS
AIA_SRC_PK_EXPRESSION	<Default>:MY_ALIAS....
AIA_UPDATE_EXPRESSION	<Default>:MY_ALIAS....
AIA_SRC_PK_LIST	<Default>:column1, ...
AIA_SRC_PK_ALIAS_LIST	<Default>:alias.col...
AIA_SRC_PK_CREATE_TABLE	<Default>:column1 d...
AIA_SRC_PK_JOIN_LIST	<Default>:alias.col...
DELETE_TEMPORARY_OBJ...	<Default>:Yes

LKM Description

- This module uses Sunopsis Agent to read selected data from the database and write the result in the Oracle target temporary table created dynamically.
- When using this module on a journalized source table, the Journaling table is first updated to flag the records consumed and then cleaned from these records at the end of the interface.

Restrictions:
- Option WORK_TABLE_OPTIONS should be cleared for an Oracle 7 database

Result

Source table: Semp2

Seno_part1	Seno_part2	Sename	Saddress	Sflag
101	1	Uday	Hyderabad	NEW
101	2	Kiran	Miyapur	NEW
101	3	Reddy	Nizampet	NEW

Target table: Temp2

Teno_part1	Teno_part2	Tename	Taddress
101	1	Uday	Hyderabad

Teno_part1	Teno_part2	Tename	Taddress
101	2	Kiran	Miyapur
101	3	Reddy	Nizampet

Source table data is not updated/deleted. (This is the same as original LKM behavior).

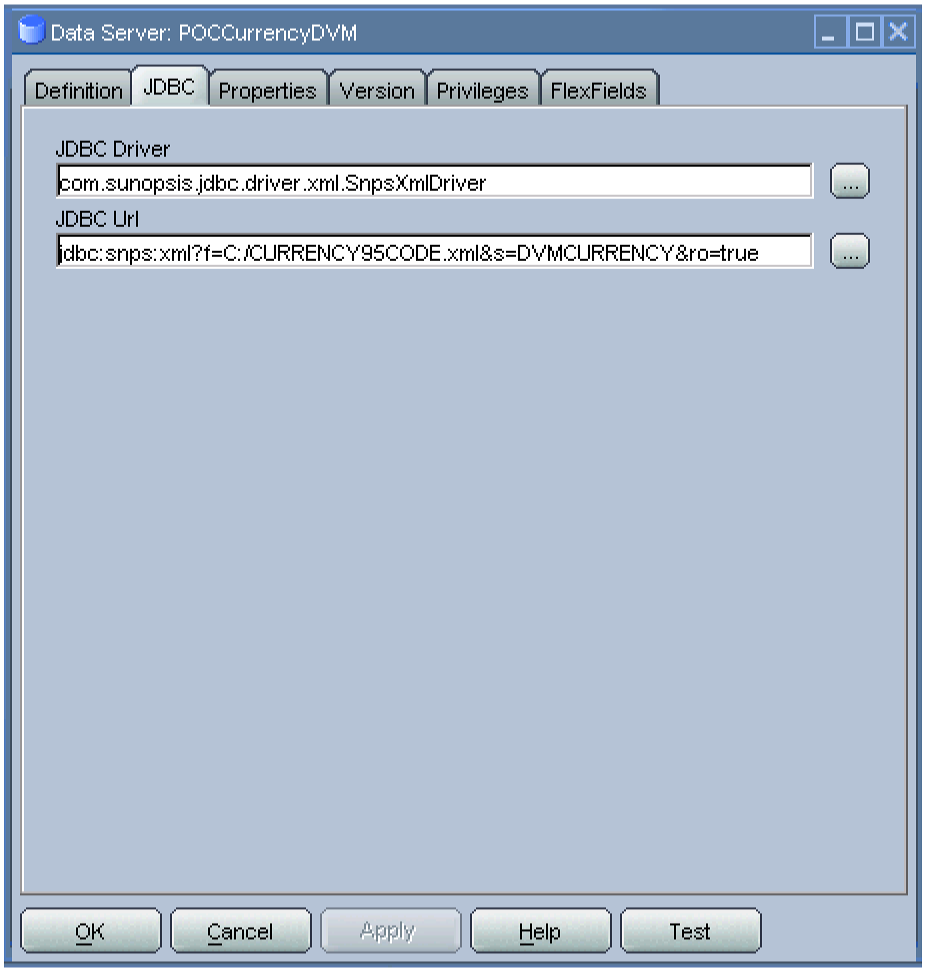
Working with DVM

The DVM maps are available as xml files and can be used as delivered. The steps are:

To use DVM maps:

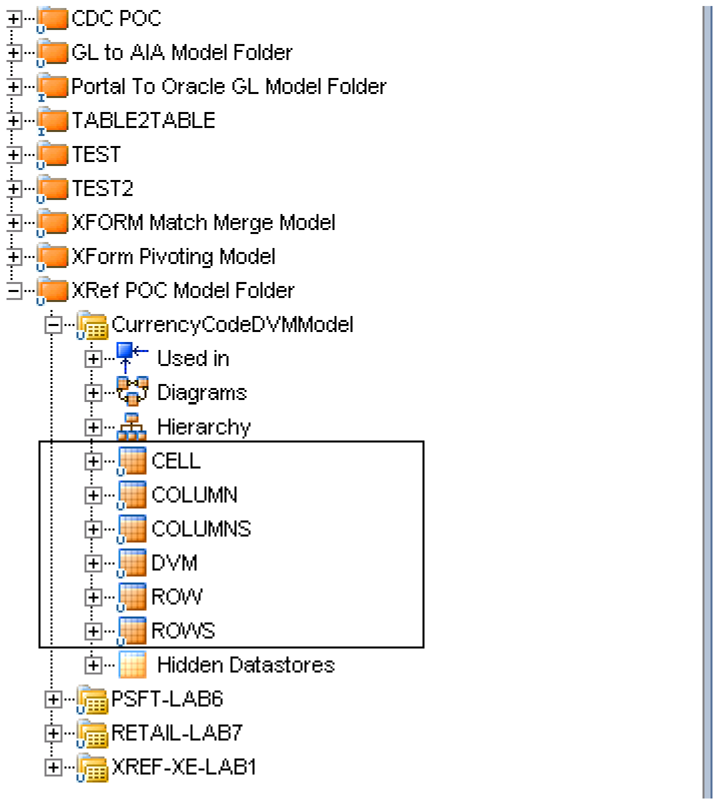
1. Reverse - engineer the DVM and its results into multiple relational tables.

   This 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.



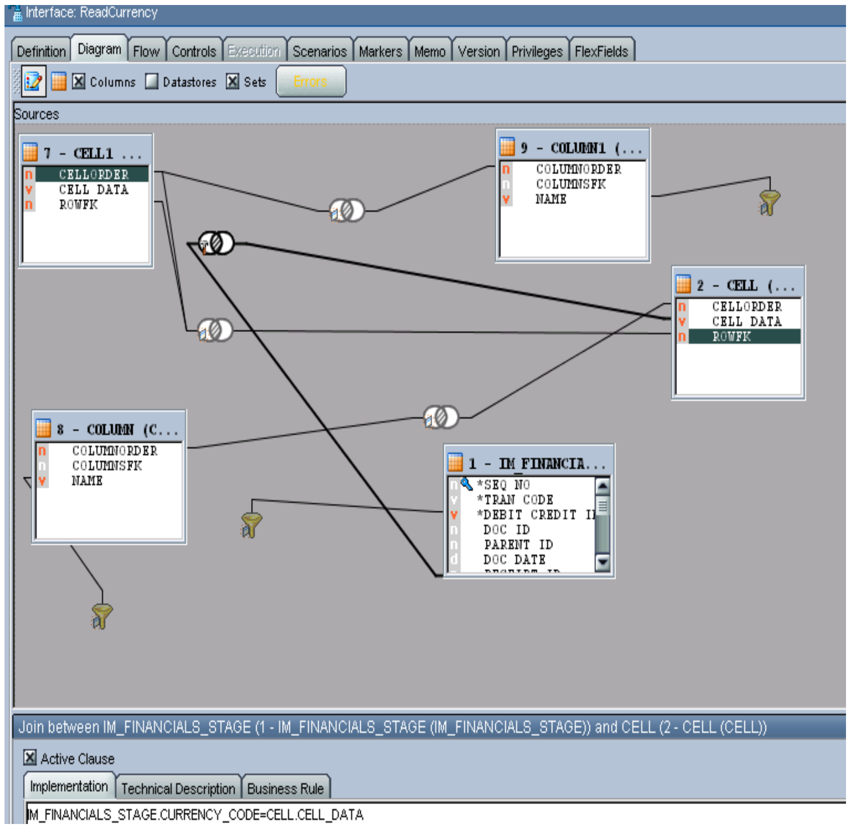
JDBC Driver
com.sunopsis.jdbc.driver.xml.SnpsXmlDriver
JDBC Url
jdbc:snps:xml?f=C:/CURRENCY95CODE.xml&s=DVMCURRENCY&ro=true

It turns into six tables after the reverse engineering as illustrated here:



2. Join the multiple tables to derive the corresponding mapping in the interface.

   Join IM_FINANCIALS_STGAE with CELL table. Use the column in your main source table that you want for DVM in the join.

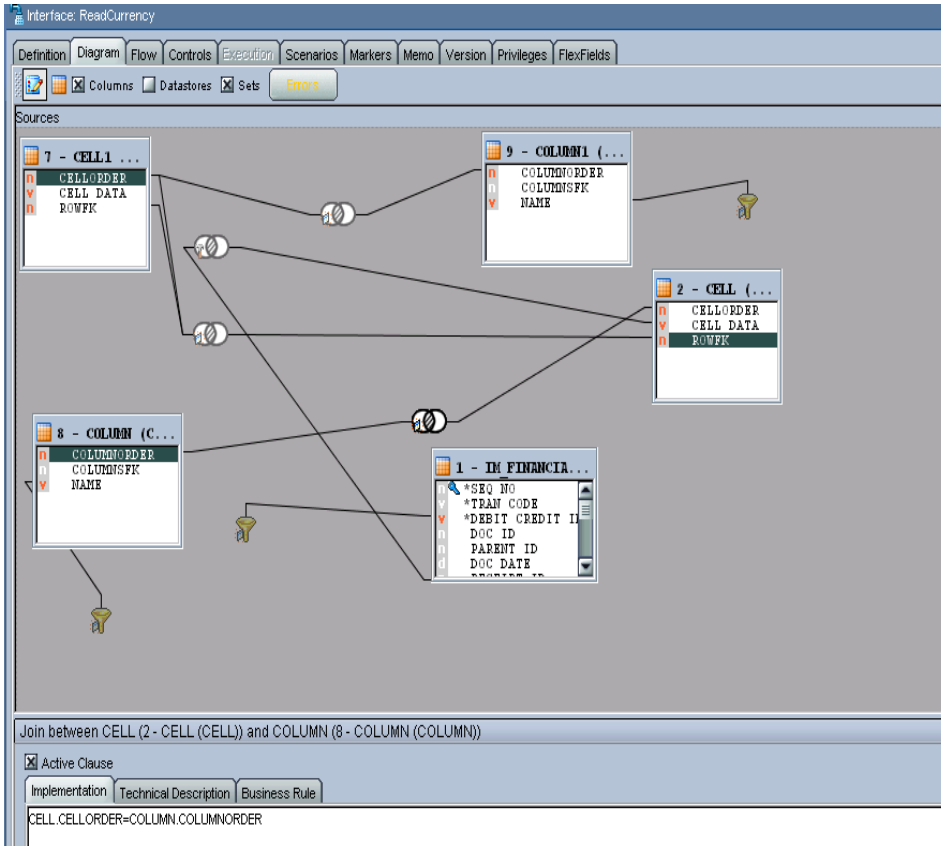


Join between IM_FINANCIALS_STAGE (1 - IM_FINANCIALS_STAGE (IM_FINANCIALS_STAGE)) and CELL (2 - CELL (CELL))

[✓ Active Clause]

IM_FINANCIALS_STAGE.CURRENCY_CODE=CELL.CELL_DATA

Join CELL table with COLUMN table.

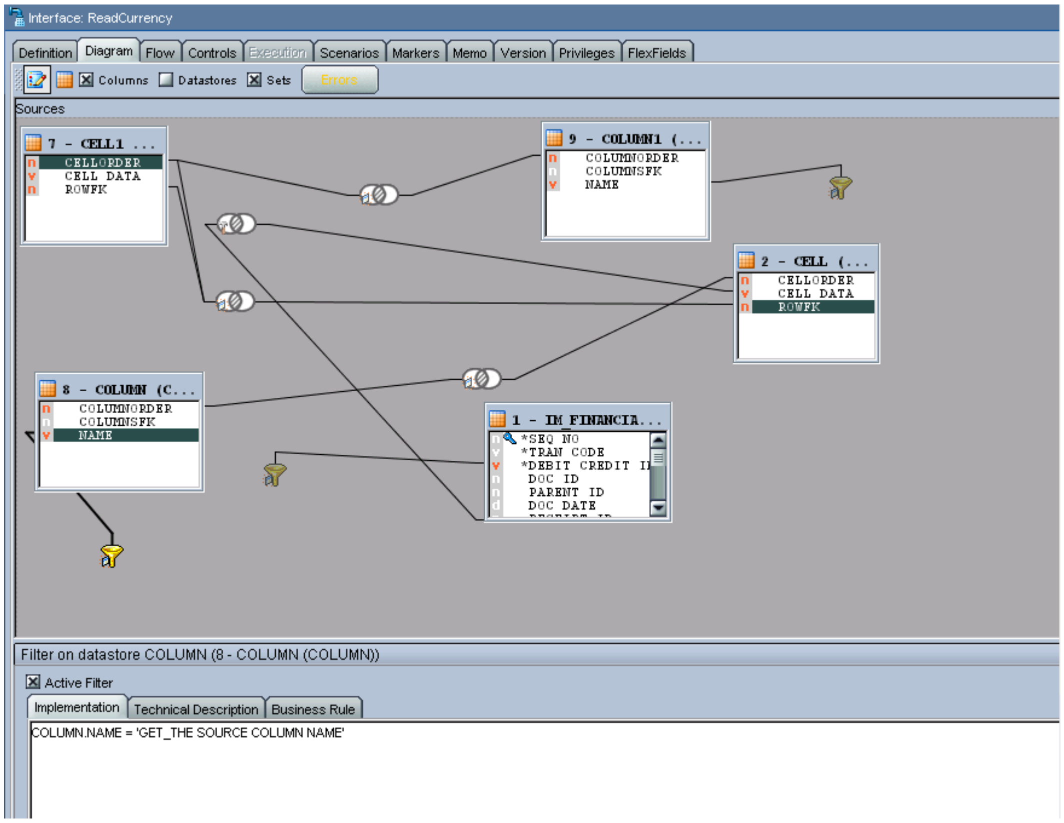


Join between CELL (2 - CELL (CELL)) and COLUMN (8 - COLUMN (COLUMN))

```
CELL.CELLORDER=COLUMN.COLUMNORDER
```

3. Add a filter for COLUMN table (to find out the source column name).

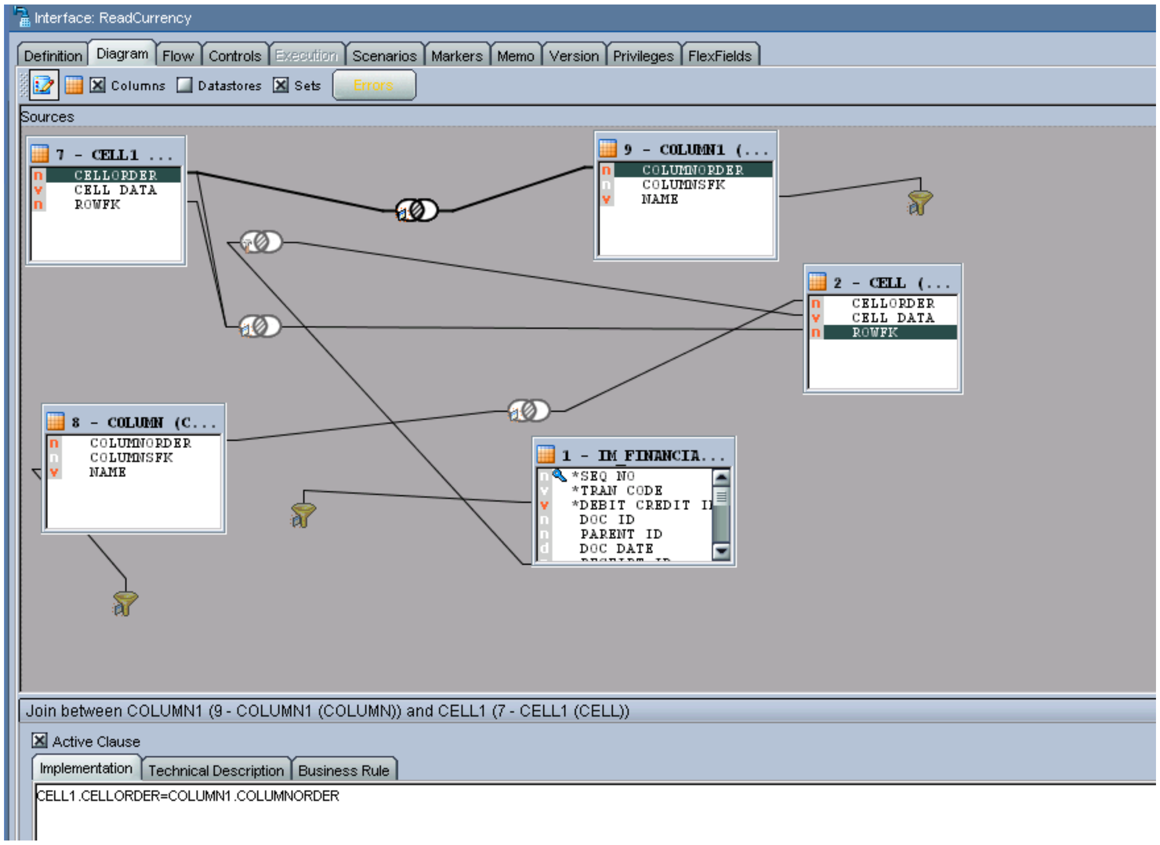
This filter can use a variable that holds the source column name.



```
COLUMN.NAME = 'GET_THE SOURCE COLUMN NAME'
```

Drop 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 will fetch the target values.

Join CELL1 with COLUMN1 table.

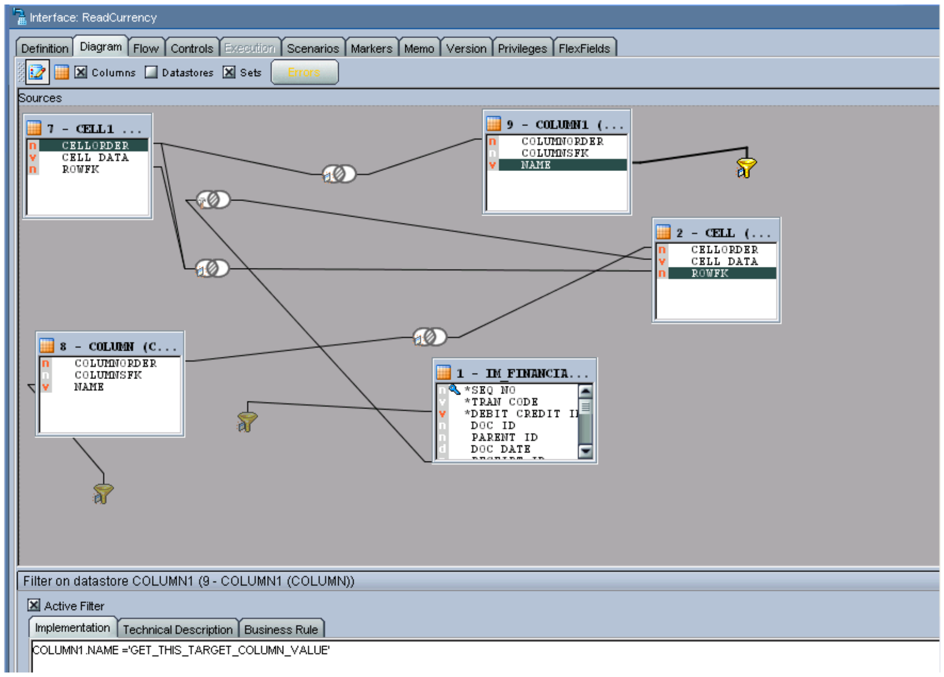


Join between COLUMN1 (9 - COLUMN1 (COLUMN)) and CELL1 (7 - CELL1 (CELL))

```
CELL1.CELLORDER=COLUMN1.COLUMNORDER
```

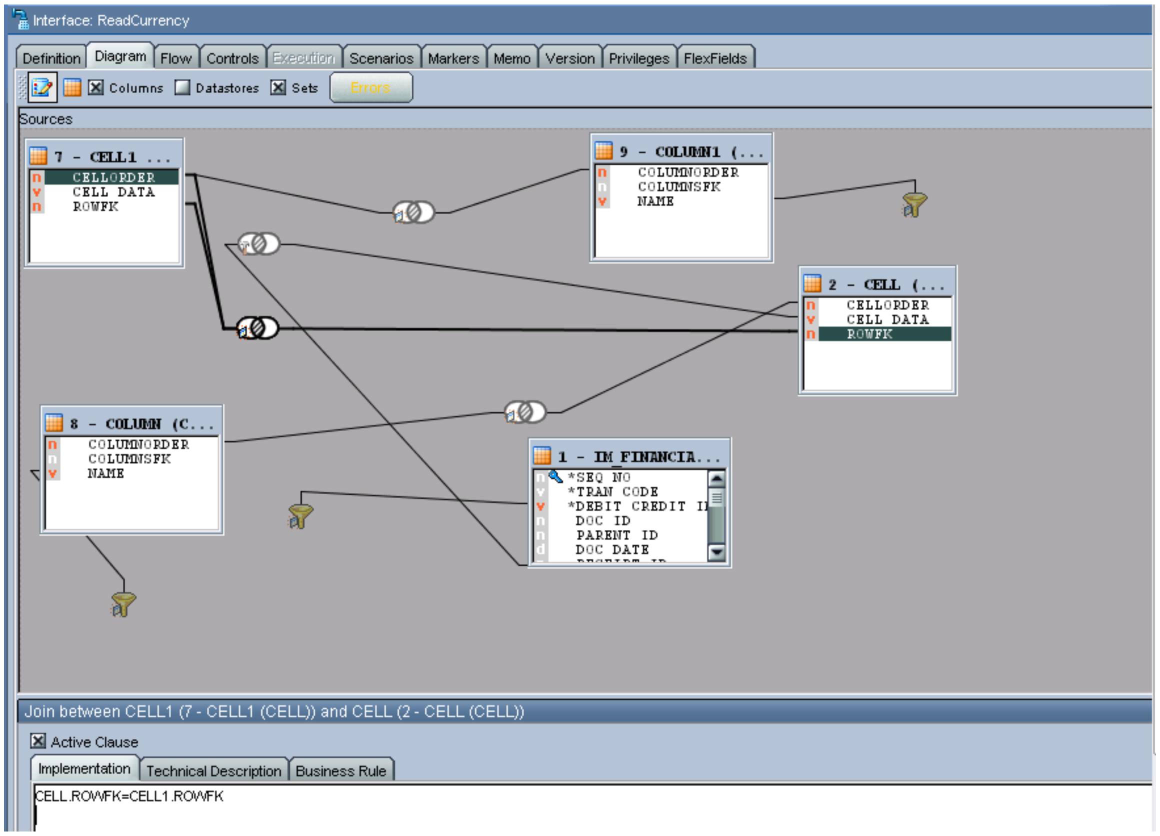
4. Add a filter on COLUMN1 table (to find out the target column name).

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



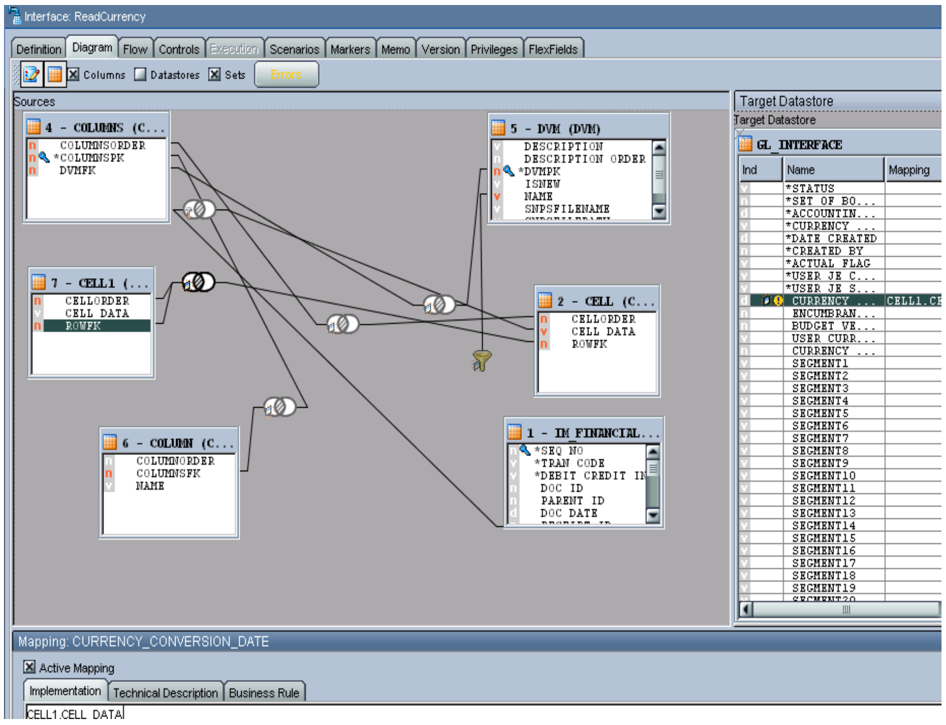
COLUMN1.NAME ='GET_THIS_TARGET_COLUMN_VALUE'

Finally self join CELL table with another CELL (CELL1) table.



CELL.ROWFK=CELL1.ROWFK

In the target interface mapping use this second CELL data (CEL1 table's data).



Ind	Name	Mapping
v	*STATUS	
n	*SET OF BO...	
d	*ACCOUNTIN...	
v	*CURRENCY ...	
d	*DATE CREATED	
n	*CREATED BY	
v	*ACTUAL FLAG	
v	*USER JE C...	
v	*USER JE S...	
d	CURRENCY ...	CELL1.CE
n	ENCUMBRAN...	
n	BUDGET VE...	
v	USER CURR...	
n	CURRENCY ...	
v	SEGMENT1	
v	SEGMENT2	
v	SEGMENT3	
v	SEGMENT4	
v	SEGMENT5	
v	SEGMENT6	
v	SEGMENT7	
v	SEGMENT8	
v	SEGMENT9	
v	SEGMENT10	
v	SEGMENT11	
v	SEGMENT12	
v	SEGMENT13	
v	SEGMENT14	
v	SEGMENT15	
v	SEGMENT16	
v	SEGMENT17	
v	SEGMENT18	
v	SEGMENT19	

Mapping: CURRENCY_CONVERSION_DATE

☒ Active Mapping

Implementation | Technical Description | Business Rule

CELL1.CELL_DATA

Remember to repeat all these joins for each DVM centric column.

For more information, see the *Oracle Data Integrator Reference Guide* and *Oracle Data Integrator Knowledge Module Developer's Guide.*

Part 3: Programming Models

Chapter 18: Describing the Event Aggregation Programming Model

This chapter provides an overview of the Event Aggregation programming model and discusses how to implement the Event Aggregation programming model.

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.

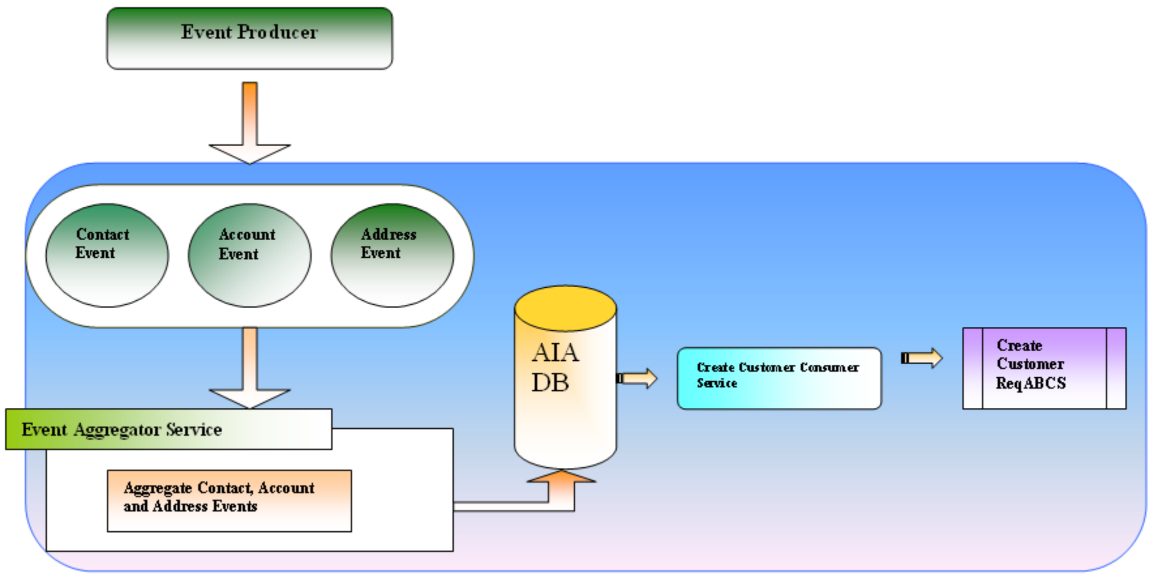
The Event Aggregation programming model can be used for relationship- and time-based aggregation.

The Event Aggregation programming model provides these:

- 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.

This diagram 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.



The Event Aggregation service raises 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.

We recommend 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 there is a need to gracefully handle them by notifying the Event Producer. BPEL provides fault and exception handling functionality that makes it a good choice for this scenario.

We recommended 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.

Once 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, as well as 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 multi-level relationships between entities.

The stored procedures help to 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: Master Data Management to Siebel CRM

This implementation discussion is based on this use case.

In the Master Data Management (MDM) Customer project, 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, there is a need to 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.

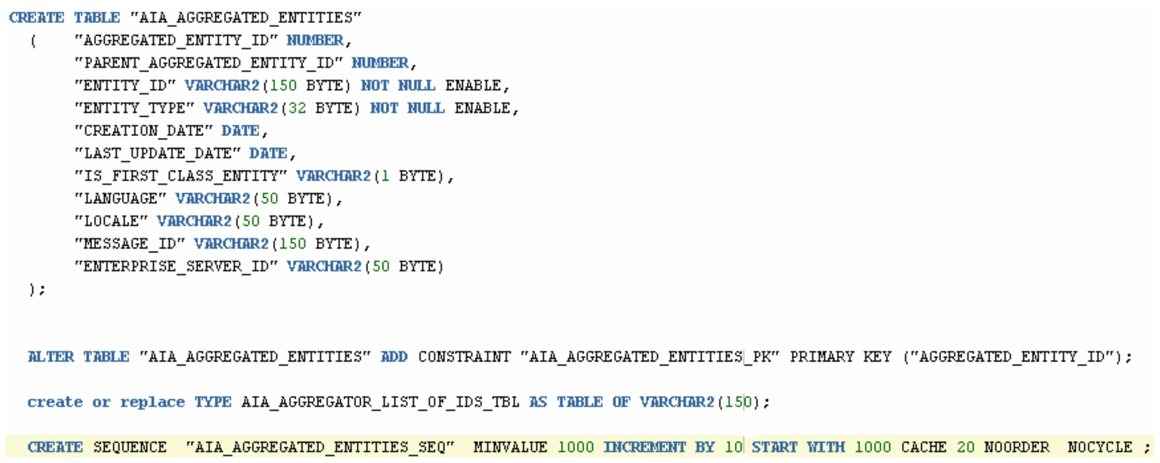
Creating the Event Aggregation Service

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

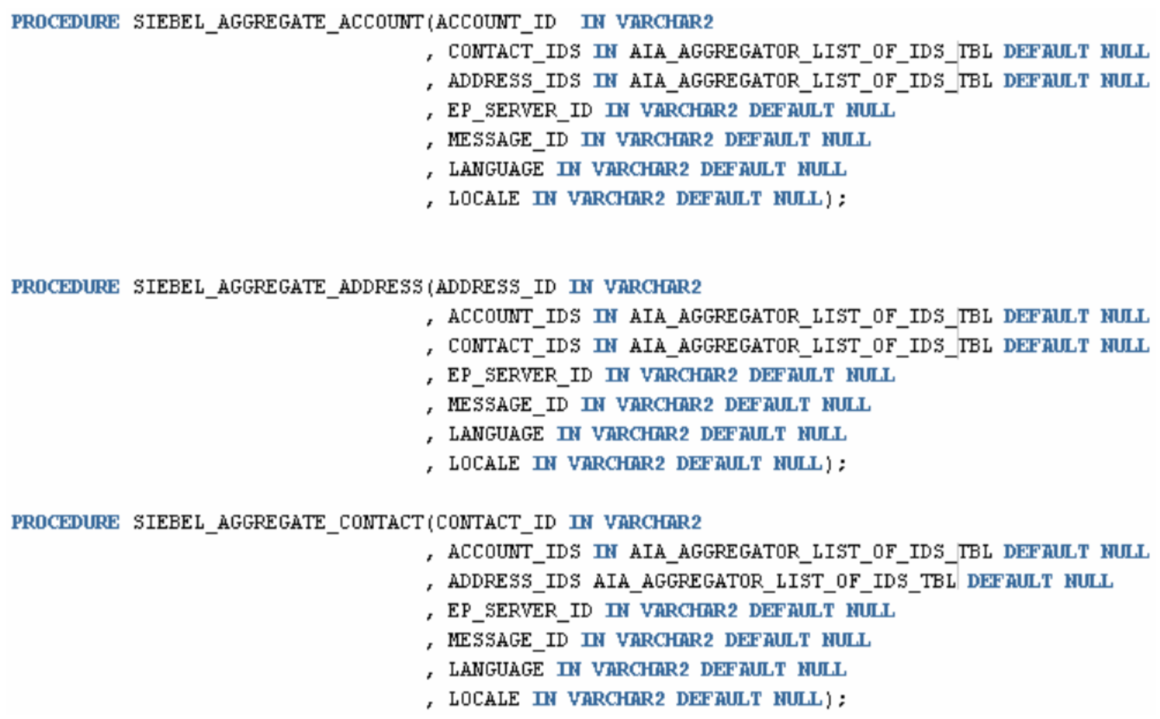
To create PL/SQL objects:

1. Create a table object “AIA_AGGREGATED_ENTITIES” in the database.



```
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 ;
```

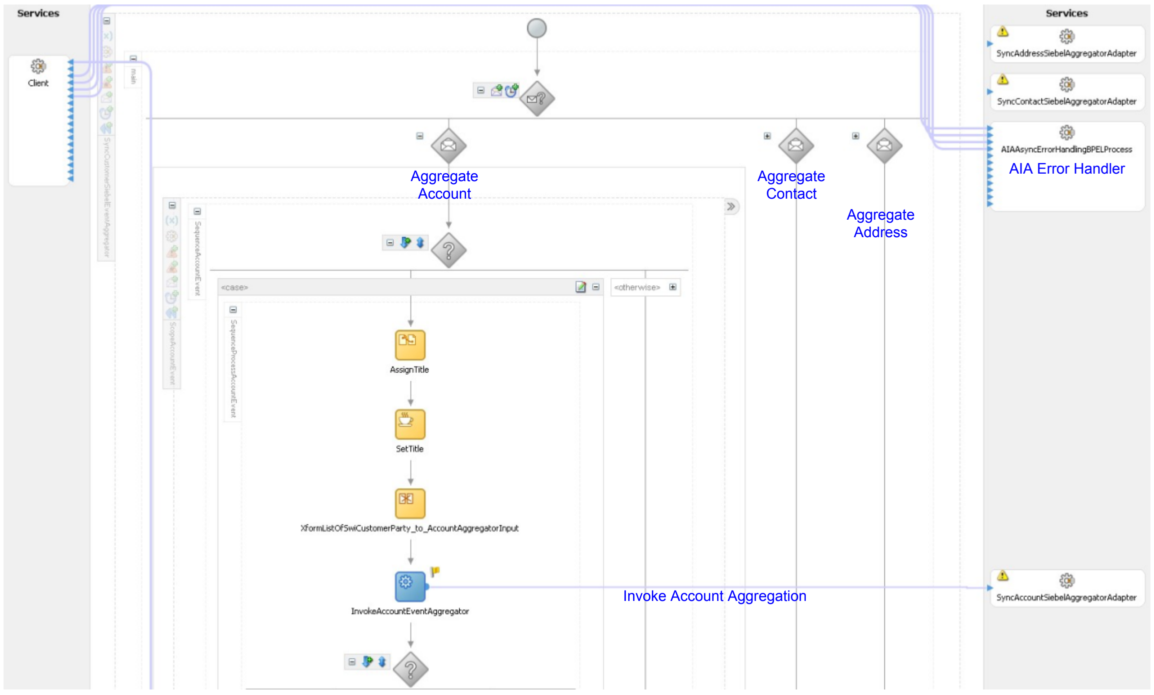
2. Create a stored procedure object “AIA_AGGREGATOR_PUB,” which contains the programming logic to maintain the relationships between the Contact, Account, and Address events in the table object created in step 1.



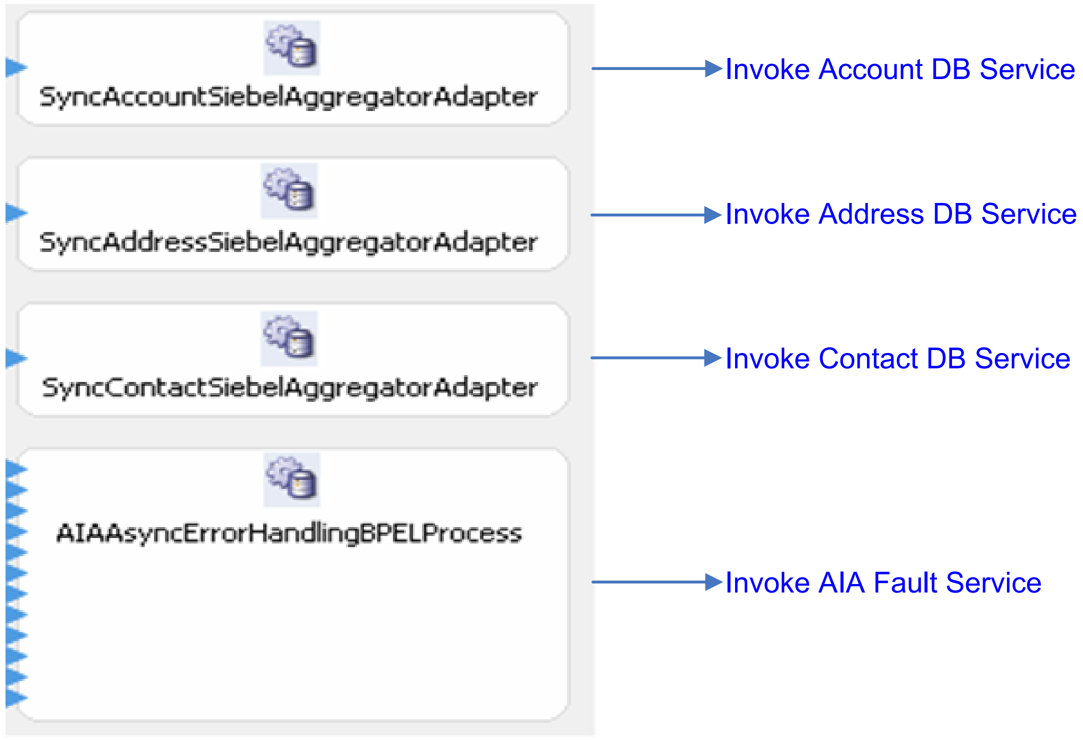
```
PROCEDURE SIEBEL_AGGREGATE_ACCOUNT(ACCOUNT_ID  IN VARCHAR2
                                  , CONTACT_IDS IN AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                  , ADDRESS_IDS IN AIA_AGGREGATOR_LIST_OF_IDS_TBL DEFAULT NULL
                                  , EP_SERVER_ID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

To create the database services and aggregate service:

1. Create a BPEL project to invoke the database services created in the previous procedure.


Services
Client
Aggregate Account
Aggregate Contact
Aggregate Address
<case>
<otherwise>
AssignTitle
SetTitle
XformListOfSwiCustomerParty_to_AccountAggregatorInput
InvokeAccountEventAggregator
Invoke Account Aggregation
Services
SyncAddressSiebelAggregatorAdapter
SyncContactSiebelAggregatorAdapter
AIAAsyncErrorHandlingBPELProcess
AIA Error Handler
SyncAccountSiebelAggregatorAdapter

2. Create PartnerLinks for the database services with the help of database adapters for the Contact, Account, and Address events.


SyncAccountSiebelAggregatorAdapter
Invoke Account DB Service
SyncAddressSiebelAggregatorAdapter
Invoke Address DB Service
SyncContactSiebelAggregatorAdapter
Invoke Contact DB Service
AIAAsyncErrorHandlingBPELProcess
Invoke AIA Fault Service

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 will invoke this BPEL service for Event Aggregation.

We recommend 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 the Consumer Service

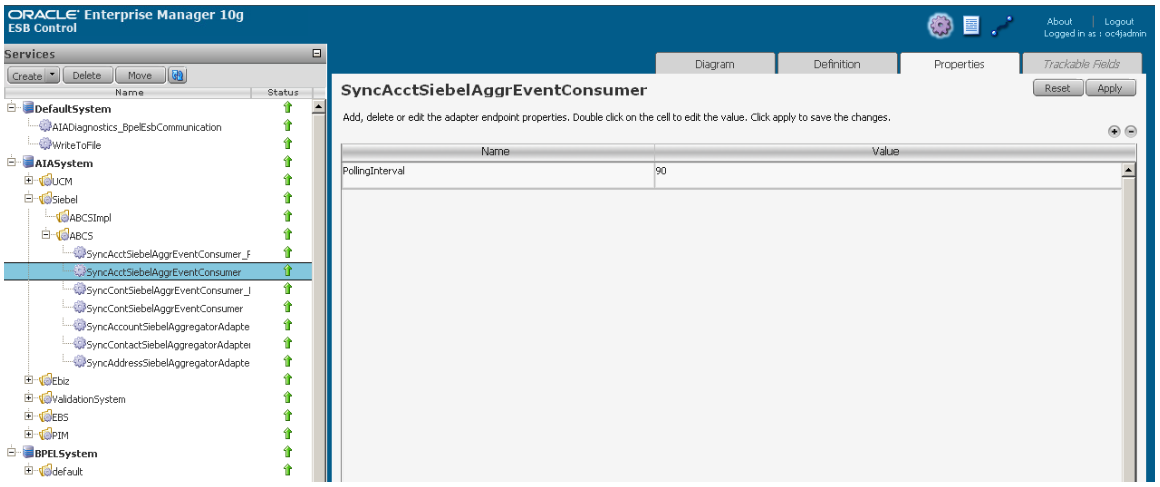
To create the Consumer Service:

1. Create a Consumer Service with the help of database adapters using Oracle Enterprise Service Bus (ESB).

   We recommend that you implement the Consumer Service using ESB, unless you need to perform data enrichment. Use database adapter functionality to purge records from the database upon successful processing.

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.

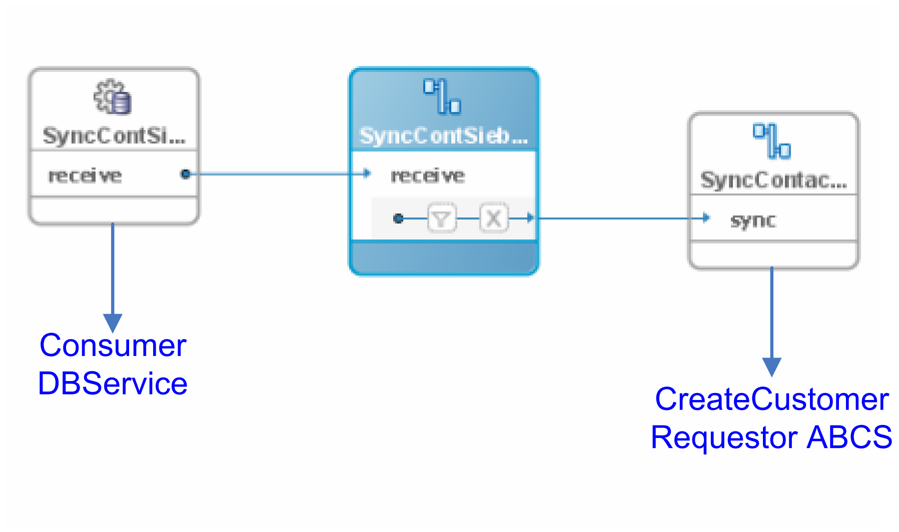

ORACLE Enterprise Manager 10g
ESB Control
About
Logout
Logged in as : oc4jadmin
Services
Create
Delete
Move
Name
Status
DefaultSystem
AIADiagnostics_BpelEsbCommunication
WriteToFile
AIASystem
UCM
Siebel
ABCSImpl
ABCS
SyncAcctSiebelAggrEventConsumer_F
SyncAcctSiebelAggrEventConsumer
SyncContSiebelAggrEventConsumer_I
SyncContSiebelAggrEventConsumer
SyncAccountSiebelAggregatorAdapte
SyncContactSiebelAggregatorAdapte
SyncAddressSiebelAggregatorAdapte
Ebiz
ValidationSystem
EBS
PIM
BPELSystem
default
Diagram
Definition
Properties
Trackable Fields
Reset
Apply

SyncAcctSiebelAggrEventConsumer

Add, delete or edit the adapter endpoint properties. Double click on the cell to edit the value. Click apply to save the changes.

Name	Value
PollingInterval	90

3. Create a routing service to invoke the Create Customer Requester ABC service.



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 *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* “Setting Up and Using Error Handling and Logging.”

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.

We recommend using BPEL for the implementation of the Event Aggregator Service layer. Using BPEL will help to ensure 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.

Chapter 19: Describing the Publish-and-Subscribe Programming Model

This chapter provides an overview of the Publish-and-Subscribe programming model and describes:

- Publish-and-subscribe programming model use cases.
- Publish-and-subscribe programming model solution scenarios.
- Publish-and-subscribe programming model development activity details.

Overview

There are multiple integration scenarios in which participating applications publish events and messages that are subscribed to by multiple participating applications. This pattern is transactional in the sense that changes are made to the object instances in the participating applications. These scenarios require an asynchronous and durable implementation model.

Multiple variations of these scenarios are described in these uses cases.

Describing Publish-and-Subscribe Programming Model Use Cases

This section discusses these use cases for the Publish-and-Subscribe programming model:

- One publisher, multiple subscriber
- Multiple publisher, multiple subscriber

Describing the One Publisher, Multiple Subscriber Use Case

In this case, a Master Data Management (MDM) application is the master application or a participating application is the master application. For example, Siebel Universal Customer Master (UCM), an MDM application, is the Customer master or the Siebel CRM participating application is the Customer master.

Customer information can be created, updated, and deleted in multiple participating applications and sent to the master application or MDM for updating, cleaning and validation.

Subsequently, the master application or MDM, as the single source of truth, publishes the customer information for multiple subscribing participating applications to consume.

These different message types require different solution scenarios:

- Master application or MDM publishes a message with multiple object instances.

The message published by the master application has a list of object instances, for example, a list of customers or a list of products.

All object instances in the list do not need to be updated to all participating applications. Therefore, Oracle Application Integration Architecture (AIA) services need to provide features that enable appropriate filtering and routing to the needed applications.

- MDM publishes a message with multiple object instances with recipient details

  The message published by MDM contains a list of object instances and also the recipient (subscribing application) details for each of the object instances. MDM registers all of the participating applications and also maintains cross reference information. Therefore, in a case in which Siebel UCM is publishing master data updates, Oracle AIA services should use these recipient details to route the object instances to appropriate subscribing applications.

Describing the Multiple Publisher, Multiple Subscriber Scenario Use Case

In this case, there is a bi-directional synchronization of object instances with no application designated as a master. Multiple applications participating in the integration scenario can create, update, and delete the same entity and publish those changes. These changes are picked up by subscribing applications. Oracle AIA services should provide features that enable filtering and routing to subscribing applications.

Describing Use Case Characteristics

These uses cases share these characteristics.

Asynchronous Publication of Discrete Messages

Messages are pushed to the Java Message Service (JMS) queue by the publishing application. The JMS consumer receives the messages and triggers the requestor Application Business Connector (ABC) service.

Message Splitting

Messages may consist of a list of object instances. In this case, the requestor ABC service splits the messages. If the list of recipients is known, the messages are split into messages for each recipient.

Recipient Information May Be Available in Published Messages

MDM systems, such as Siebel UCM and Oracle Customer Data Hub (CDH), register participating applications and also store cross reference information. Messages published by these applications, when handling object instances that are lists, contain information about the list of recipients for each of the object instances in the message.

Guarantee Receipt of Messages by Recipients

The applications publish messages asynchronously, so Oracle AIA services must guarantee the delivery of messages to recipients.

Sequence Published Messages

There could be situations in which updates to the same entity are published in quick succession. To ensure that messages are received and processed by consumers in the correct order, messages must be sequenced.

Avoid Echoing of Messages

When updates to object instances in one participating application are updated in other applications, they may raise update events that leading to the publication of other messages. This message echo effect must be avoided

Ensure Conflict Detection and Conflict Avoidance

There may be concurrent updates to the same entity in different participating applications. When these applications publish updates, both applications may receive each other's updates. This would lead to a data integrity issue. This conflict must be detected and avoided.

Describing Publish-and-Subscribe Programming Model Solution Scenarios

This section provides solution options for these Publish-and-Subscribe programming model scenarios:

- Participating applications publish updates to MDM or a master application.
- MDM publishes updates to known recipients (subscribing applications).
- Participating applications publish updates to other participating applications.

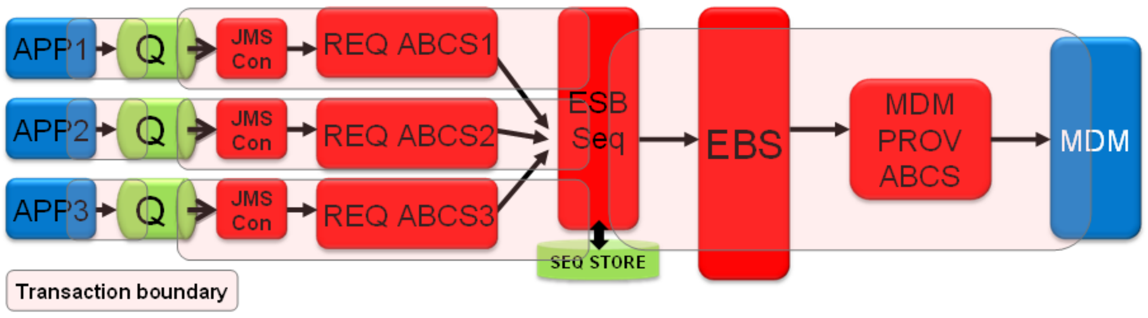
  This is a bi-directional synchronization with no MDM or master application in place.
- MDM publishes updates to participating applications.

Participating Applications Publish Updates to MDM or Master Application

This section provides two solution options for this scenario.

Participating Applications Publish Updates to MDM or Master Application: Solution Option 1

This flowchart illustrates this solution option:



Participating applications publish updates to MDM or master application: solution option 1

Solution Flow

Transaction 1:

Participating applications push messages to the queue.

Transaction 2:

Messages in the queue trigger the JMS consumer.

The JMS consumer triggers the requestor ABC service.

The requestor ABC service transforms the messages into an enterprise business message (EBM) and invokes the Oracle Enterprise Service Bus (ESB) Resequencer service.

Transaction 3

The Oracle ESB Resequencer sequences the EBMs, group-wise, and invokes the enterprise business service (EBS) routing service.

The EBS routing service routes the EBM to the MDM provider ABC service.

The provider ABC service pushes the EBM to MDM.

Pros and Cons

- EBMs are resequenced.
- If one of the ABC services fails, the messages in the corresponding queue may not be part of the messages processed in the sequence.

Development Activities

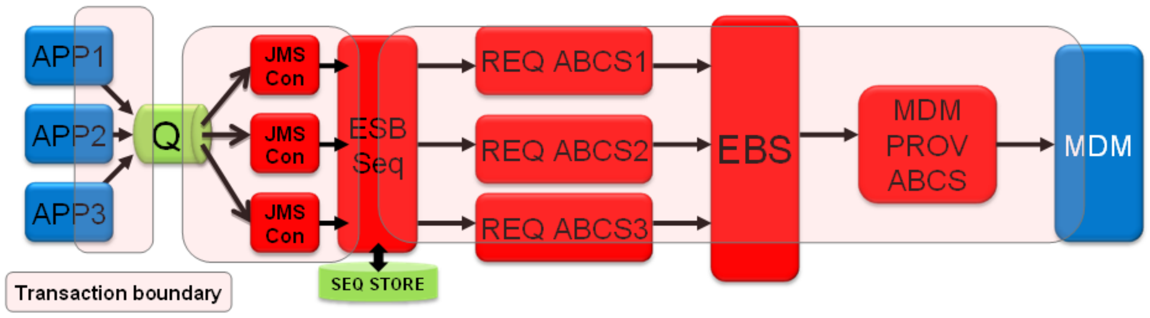
- Configure the JMS queue as an XA resource.
- Configure the JMS client in the publishing application to push messages to the JMS queue.
- Configure the JMS consumer to propagate transactions.
- Configure the ESB Resequencer service to initiate a transaction.

- Ensure that the requestor ABC service participates in the global transaction.
- Ensure that the requestor ABC service invokes the ESB Resequencer service with the global transaction in place.
- Ensure that the EBS routing service propagates the transaction.
- Ensure that the provider ABC service participates in the transaction.
- Ensure that the provider ABC service invokes MDM with the global transaction in place.

For more information about how to perform these development activities, see Describing Publish-and-Subscribe Programming Model Development Activity Details.

Participating Applications Publish Updates to MDM or Master Application: Solution Option 2

This flowchart illustrates this solution option:



Participating applications publish updates to MDM or master application: solution option 2

Solution Flow

Transaction 1:

All participating applications push messages to a single queue.

Transaction 2:

Messages in the queue trigger JMS consumers based on the schema format.

The JMS consumer applies a wrapper schema and sets the group ID and sequence ID to the root of the wrapper schema.

The JMS consumer invokes the Oracle ESB Resequencer service.

Transaction 3:

The Oracle ESB Resequencer sequences the messages, group-wise, and invokes the requestor ABC service.

The requestor ABC service transforms the messages into an EBM and invokes the EBS Routing Service

The EBS routing service routes the EBM to the MDM provider ABC service.

The provider ABC service pushes the transformed ABM to MDM.

Pros and Cons

- All messages from all applications will be processed sequentially.
- Application messages must be placed in a wrapper schema and then sent to the sequencing service.

Development Activities

- Configure the JMS queue as an XA (using the XAConnectionFactory) resource.
- Configure the JMS client in the publishing application to push messages to the JMS queue.
- In the JMS consumer, transform the message to place it in a wrapper schema
- Configure the JMS consumer to propagate the transactions.
- Configure the ESB Resequencer service to initiate a transaction.
- Ensure that the ESB Resequencer invokes the requestor ABC service with the global transaction in place.
- Ensure that the requestor ABC service participates in the global transaction.
- Ensure that the EBS routing service propagates the transaction.
- Ensure that the provider ABC service participates in the transaction.
- Ensure that the provider ABC service invokes MDM with the global transaction in place.

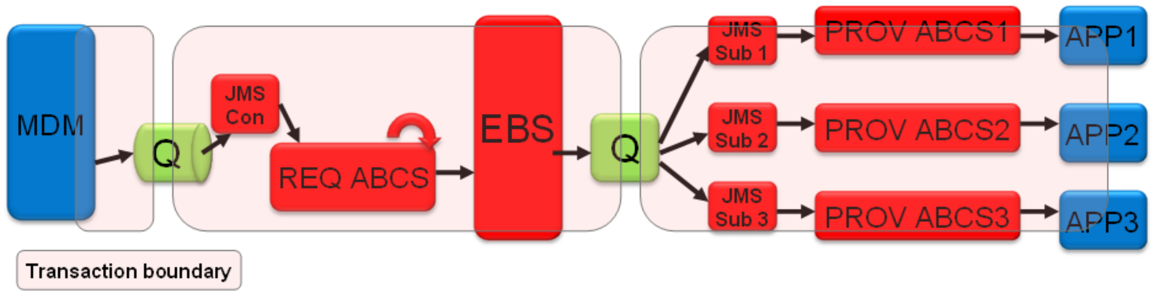
For more information about how to perform these development activities, see Describing Publish-and-Subscribe Programming Model Development Activity Details.

MDM Publishes Updates to Known Recipients (Subscribing Applications)

This section provides two solution options for this scenario.

MDM Publishes Updates to Known Recipients: Solution Option 1

This flowchart illustrates this solution option:



MDM publishes updates to known recipients: solution option 1

Solution Flow

Transaction 1

MDM pushes messages, the list of object instances, to the queue.

Transaction 2

Messages in the queue trigger the JMS consumer.

The JMS consumer triggers the requestor ABC service.

The requestor ABC service transforms and splits the message into an EBM and sets the target recipient information to the EBM header and invokes the EBS routing service

The EBS routing service sets the JMS header with the string matching the name of the JMS subscriber that corresponds to the target recipient information. The EBS routing service obtains this value by retrieving it from the AIA configuration properties file. It then pushes the message to the JMS queue.

Transaction 3

Multiple JMS subscribers corresponding to each of the provider ABC services are configured on the JMS queue.

Each JMS subscriber has “Message Selector” criteria set to match the JMS header property that matches the name of the JMS subscriber.

The JMS subscriber invokes the corresponding provider ABC service.

The provider ABC service pushes the transformed ABM to the participating application.

Pros and Cons

- There is a single JMS queue and all JMS subscribers pick up messages from it.
- A message selector must be added to the JMS consumer. This allows the JMS consumer to filter messages and pick only the messages that are meant for the corresponding ABC service. This is not visible to the user.
- Although the JMS queue is pushing the messages, semantically, the JMS consumer is deciding which messages the queue need. Therefore, the JMS queue, is in a sense, is pulling messages.

- When a new application is added, a new JMS consumer is created. No intrusion into existing code is necessary.

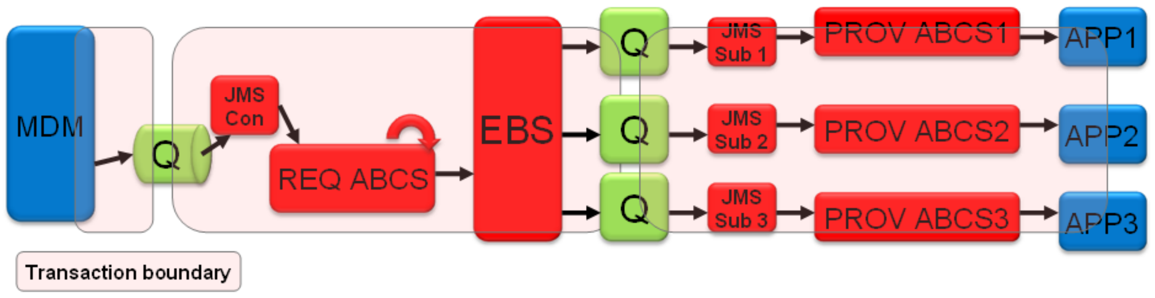
Development Activities

- Configure the JMS queue as an XA resource.
- Configure the JMS client in the publishing application to push messages to the JMS queue.
- Configure the JMS consumer to propagate transactions.
- Ensure that the requestor ABC service participates in the global transaction.
- Ensure that the EBS routing service propagates the transaction.
- Configure the EBS Routing Service to publish the message into JMS Queue after setting the JMS Header attribute with the subscriber information
- JMS Subscriber has a 'Message Selector' criteria set to match the JMS header property matching with the name of the JMS Subscriber
- Configure JMS Subscriber to propagate transaction
- Ensure that the provider ABC service participates in the transaction.
- Ensure that the provider ABC service invokes MDM with the global transaction in place.

For more information about how to perform these development activities, see Describing Publish-and-Subscribe Programming Model Development Activity Details.

MDM Publishes Updates to Known Recipients: Solution Option 2

This flowchart illustrates this solution option:



MDM publishes updates to known recipients: solution option 2

Solution Flow

Transaction 1:

MDM pushes messages, the list of object instances, to the queue.

Transaction 2:

Messages in the queue trigger the JMS consumer.

The JMS consumer triggers the requestor ABC service.

The requestor ABC service transforms and splits the message into an EBM and sets the target recipient information to the EBM header and invokes the EBS routing service.

The EBS routing service uses the information in the EBM Header to route the message to the correct JMS queue.

Transaction 3:

The JMS subscriber invokes the corresponding provider ABC service.

The provider ABC service pushes the transformed ABM to participating application.

Pros and Cons

- The EBS will have multiple routing rules to push to dedicated queues for each provider ABC service. The rules are visible using the Oracle ESB Console.
- There is no requirement to put a message selector on the JMS consumer.
- When a new application is added, a new routing rule and JMS queue must be added.

Development Activities

- Configure the JMS queue as a XA resource.
- Configure the JMS client in the publishing application to push messages to the JMS queue.
- Configure the JMS consumer to propagate transactions.
- Ensure that the requestor ABC service participates in the global transaction.
- Ensure that the EBS routing service propagates the transaction.
- Configure the EBS Routing Service to publish the message into JMS Queue.
- Configure JMS Subscriber to propagate transaction.
- Ensure that the provider ABC service participates in the transaction.
- Ensure that the provider ABC service invokes MDM with the global transaction in place.

For more information about how to perform these development activities, see Describing Publish-and-Subscribe Programming Model Development Activity Details.

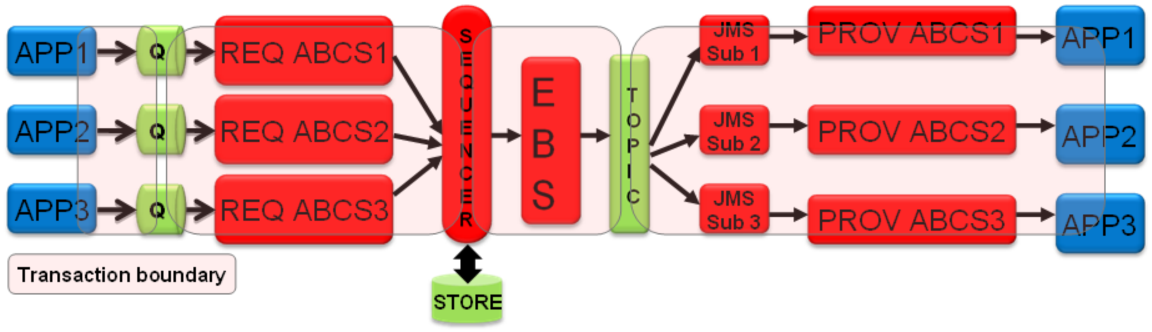
Participating Applications Publish Updates to Other Participating Applications

This is a bi-directional synchronization with no MDM or master application in place.

This section provides two solution options for this scenario.

Participating Applications Publish Updates to Other Participating Applications: Solution Option 1

This flowchart illustrates this solution option:



Participating applications publish updates to other participating applications: solution option 1

Solution Flow

Transaction 1:

The participating applications push messages to the queue.

Transaction 2:

Messages in the queue trigger the JMS consumer.

The JMS consumer triggers the requestor ABC service.

The requestor ABC service transforms the messages into an EBM and invokes the Oracle ESB Resequencer service.

Transaction 3:

The Oracle ESB Resequencer sequences the EBMs, group-wise, and invokes the EBS routing service.

The EBS routing service routes it to the JMS topic.

Transaction 4:

Multiple durable JMS subscribers corresponding to each of the provider ABC service are configured on the JMS topic.

Each durable subscriber receives a message copy.

The JMS subscriber invokes the corresponding provider ABC service.

The provider ABC service pushes it to the participating application.

Development Activities

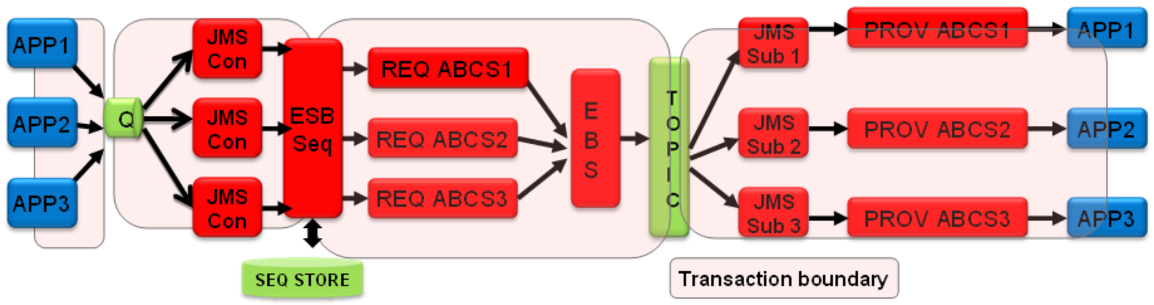
- Configure the JMS queue as an XA resource.
- Configure the JMS client in the publishing application to push messages to the JMS queue.
- Configure the JMS consumer to propagate transactions.

- Ensure that the requestor ABC service participates in the global transaction.
- Ensure that the requestor ABC service invokes the ESB Resequencer service with the global transaction in place.
- Configure the ESB Resequencer service to initiate a transaction.
- Ensure that the EBS routing service propagates the transaction.
- Configure a JMS topic.
- Configure durable subscribers.
- Ensure that the provider ABC service participates in the transaction.
- Ensure that the provider ABC service invokes MDM with the global transaction in place.

For more information about how to perform these development activities, see Describing Publish-and-Subscribe Programming Model Development Activity Details.

Participating Applications Publish Updates to Other Participating Applications: Solution Option 2

This flowchart illustrates this solution option:



Participating applications publish updates to other participating applications: solution option 2

Solution Flow

Transaction 1:

All participating applications push messages to a single queue.

Transaction 2:

Messages in the queue trigger JMS consumers based on schema format.

The JMS consumer applies a wrapper schema and sets the group ID and sequence ID to the root of the wrapper schema.

The JMS consumer invokes the Oracle ESB Resequencer service.

Transaction 3:

The Oracle ESB Resequencer sequences the messages, group-wise, and invokes the requestor ABC service.

The requestor ABC service transforms the messages into an EBM and invokes the EBS routing service.

The EBS routing service routes it to the JMS topic.

Transaction 4:

Multiple durable JMS subscribers corresponding to each of the provider ABC services are configured on the JMS topic.

Each durable subscriber receives a message copy.

The JMS subscriber invokes the corresponding provider ABC service.

The provider ABC service pushes it to the participating application.

Development Activities

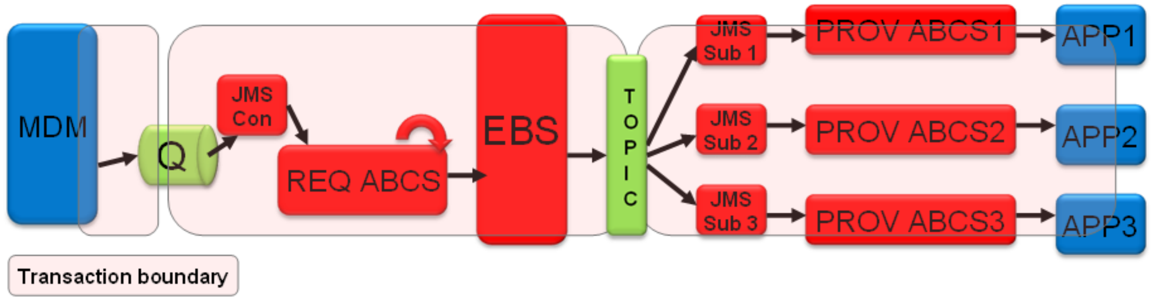
- Configure the JMS queue as an XA resource.
- Configure the JMS client in the publishing application to push messages to the JMS queue.
- In the JMS consumer, transform the message to place it in a wrapper schema.
- Configure the JMS consumer to propagate transactions.
- Configure the ESB Resequencer service to initiate a transaction.
- Ensure that the ESB Resequencer invokes the requestor ABC service with the global transaction in place.
- Ensure that the requestor ABC service participates in the global transaction.
- Ensure that the EBS routing service propagates the transaction.
- Configure a JMS topic.
- Configure durable subscribers.
- Ensure that the provider ABC service participates in the transaction.
- Ensure that the provider ABC service invokes the participating applications with the global transaction in place.

For more information about how to perform these development activities, see Describing Publish-and-Subscribe Programming Model Development Activity Details.

MDM Publishes Updates to Participating Applications

This section provides a solution option for this scenario.

This flowchart illustrates this solution option:



MDM publishes updates to participating applications

Solution Flow

Transaction 1:

MDM pushes messages, a list of object instances, to the queue.

Transaction 2:

Messages in the queue trigger the JMS consumer.

The JMS consumer triggers the requestor ABC service.

The requestor ABC service transforms and splits the message into an EBM and invokes the routing service.

The EBS routing service pushes the message to the JMS topic.

Transaction 3:

Multiple durable JMS subscribers corresponding to each of the provider ABC services are configured on the JMS topic.

Each durable subscriber receives a message copy.

The JMS subscriber invokes the corresponding provider ABC service.

The provider ABC service pushes it to the participating application.

Pros and Cons

- Each application receives updates independently and in a guaranteed manner.

Development Activities

- Configure the JMS queue as an XA resource.
- Configure the JMS client in the publishing application to push messages to the JMS queue.
- Configure the JMS consumer to propagate transactions.
- Ensure that the requestor ABC service participates in the global transaction.
- Ensure that the EBS routing service propagates the transaction.
- Configure a JMS topic.

- Configure durable subscribers.
- Ensure that the provider ABC service participates in the transaction.
- Ensure that the provider ABC service invokes the participating application with the global transaction in place.

For more information about how to perform these development activities, see Describing Publish-and-Subscribe Programming Model Development Activity Details.

Describing Publish-and-Subscribe Programming Model Development Activity Details

This section provides details about the development activities associated with Publish-and-Subscribe programming model solution scenarios.

For more information about Publish-and-Subscribe programming model solution scenarios, see Describing Publish-and-Subscribe Programming Model Solution Scenarios.

Namely, this section discusses how to:

- Configure a JMS topic.
- Configure an ESB service as a durable subscriber of a JMS topic.
- Configure the ESB Resequencer group ID and sequence ID for EBMs with different root elements.
- Use the ESB service to set the JMS header.
- Configure the JMS queue as an XA resource.
- Configure the JMS client in the publishing application to push messages to the JMS queue.
- Configure the JMS consumer to propagate transactions.
- Ensure that the requestor ABC service participates in the global transaction.
- Ensure that the requestor ABC service invokes the ESB Resequencer service with the global transaction in place.
- Configure the ESB Resequencer service to initiate a transaction.
- Ensure that the EBS routing service propagates the transaction.
- Ensure that the provider ABC service participates in the transaction.
- Ensure that the provider ABC service invokes MDM with the global transaction in place.
- Transform the message in the JMS consumer to place it in a wrapper schema.

Note: These discussions are not exhaustive, but rather are meant to be representative in nature.

Configuring a JMS Topic

To define a JMS Topic on OC4J:

1. Use JMS topic naming standards.

   - **Database User/Schema:** JMSUSER
   - **Table Name:** AIA_<EBO>JMSTTABV[Version Number]
   - **Topic Names:** AIA_<EBO>JMSTCV[Version Number]

2. Create the topic.

```
Begin
   DBMS_AQADM.CREATE_QUEUE_TABLE(Queue_table =>'
   AIA_<EBO>JMSTTABV[Version Number]',
      Queue_payload_type => 'SYS.AQ$_JMS_MESSAGE',
      sort_list => 'PRIORITY,ENQ_TIME',
      multiple_consumers=>true,
      compatible =>'8.1.5');
   DBMS_AQADM.CREATE_QUEUE(Queue_name=>'AIA_<EBO>JMSTV[Version
   Number]', Queue_table=>'AIA_<EBO>JMSTTABV[Version Number]');
   DBMS_AQADM.START_QUEUE(queue_name=>'AIA_<EBO>JMSTV[Version
   Number]');
End;
```

3. Create an entry in <SOA Home>\j2ee\home\applicationdeployments\default\JmsAdapter\oc4j-ra.xml.

   Resource provider information can be found in %ORACLE_HOME%/j2ee/home/config/application.xml.

```
<connector-factory location="eis/Jms/DBConnect" connector-name="Jms
Adapter">
  <config-property name="connectionFactoryLocation"
   value="java:comp/resource/DBConnect
  /XATopicConnectionFactories/'AIA_<EBO>JMSTOPICV[Version Number]"
/>
  <config-property name="factoryProperties" value="" />
  <config-property name="acknowledgeMode" value="CLIENT_ACKNOWLEDGE"
/>
  <config-property name="isTopic" value="true" />
  <config-property name="isTransacted" value="false" />
  <config-property name="username" value=" jmsuser" />
  <config-property name="password" value="jmsuser" />
  <connection-pooling use="none" />
  <security-config use="none" />
</connector-factory>
```

Two points to be noted in these entries:

- `<config-property name="isTopic" value="true" />`
- Use of “**XATopicConnectionFactories”**

Configure the Oracle ESB Service as a Durable Subscriber of the JMS Topic

To configure the Oracle ESB service as a durable subscriber of the JMS topic:

1. The JMS consumer must be configured as a durable subscriber. This will ensure that there only a single delivery of the message. This is done during JMS adapter configuration by setting the Durable Subscriber ID using this naming convention.
   - **JMS Consumer:** <Optional Verb><EBO>JMSConsumerV[Version Number]
   - **Durable Subscriber ID:** <Optional Verb><EBO>JMSConsumerV[Version Number]
2. Add an endpoint property “InstanceIdPrefix” to the JMS consumer so that each of the JMS consumer instances has a different ID. The property “InstanceIdPrefix” is not available in the drop-down list box in JDeveloper and cannot be added. Modify the .esbsvc file directly to add the property as:

```
<endpointProperties>
   <property name="InstanceIdPrefix" value="100"/>
</endpointProperties>
```

Configuring the Oracle ESB Resequencer Group ID and Sequence ID for EBMs with Different Root Elements

When there is an Oracle ESB Resequencer service between the requestor ABC service and the EBS routing service, the Resequencer receives EBMs from the requestor ABC service and sequences them based on their group ID and sequence ID. The Resequencer service is invoked directly by the requestor ABC service. Depending on the operation, the EBM root element may be different.

To set the group ID and sequence ID:

1. Set these values in the Endpointproperty of the ESB Routing Service:
   - **Name:** ResequencerGroupXPath
   - **Value:**

```
{//*/corecom:EBMHeader/corecom:Sender/
corecom:ObjectCrossReference[1]/corecom:EBOID};{namespace corecom=
“http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2”}
```

2. Set these values in the Endpointproperty of the ESB Routing Service:
   - **Name:** ResequencerSequenceIdXPath

- **Value:**

```
{//*/corecom:EBMHeader/corecom:CreationDateTime};{namespace corecom=
“http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V2”}
```

Using the ESB Service to Set the JMS Header

In the solution scenarios provided in this chapter, when the split message with recipient information is enqueued to a single JMS queue, the JMS header is set to the JMS consumer name that corresponds to the recipient. This information is retrieved and set from the AIA Configuration Properties file.

To use the ESB service to set the JMS header:

1. Create an entry in the AIA Configuration file for an appropriate entity related services:

```
<ServiceConfiguration serviceName="<Name of JMS Producer Service>">
<Property name="<Target System Id>"><Name of JMS
Consumer></Property>
</ServiceConfiguration>
```

2. In the JMS producer service XSL, make this entry:

```
<xsl:variable name=" JMSConSerName " select="
aiacfg:getServiceProperty(<EBOName>, <Name of JMS Producer Service>,
"/corecom:Target/ID", false)"/>

<xsl:variable name="JMSConName"
select="ehdr:setOutboundHeader('/jhdr:JMSOutboundHeadersAndPropertie
s/jhdr:JMSOutboundProperties/jhdr:Property[position()=1]/@name','JMS
ConName','jhdr=http://xmlns.oracle.com/pcbpel/adapter/jms/;')"/>

<xsl:variable name="JMSConNameValue"
select="ehdr:setOutboundHeader('/jhdr:JMSOutboundHeadersAndPropertie
s/jhdr:JMSOutboundProperties/jhdr:Property[position()=1]/@value',$JM
SConSerName,'jhdr=http://xmlns.oracle.com/pcbpel/adapter/jms/;')"/>
```

Configuring the JMS Queue as an XA Resource

For more information, see Chapter 8: Designing and Constructing JMS Adapter Services.

Configuring the JMS Client in the Publishing Application to Push Messages to the JMS Queue

For more information about configuring JMS clients in publishing applications, see relevant publishing application guides.

Configuring the JMS Consumer to Propagate Transactions

For more information, see Chapter 8: Designing and Constructing JMS Adapter Services.

Ensuring that the Requestor ABC Service Participates in the Global Transaction

For more information, see Chapter 4: Designing and Constructing ABC Services.

Ensuring that the Requestor ABC Service Invokes the ESB Resequencer Service with the Global Transaction in Place

For more information, see Chapter 4: Designing and Constructing ABC Services.

Configuring the ESB Resequencer Service to Initiate a Transaction

For more information, see Oracle Enterprise Service Bus Resequencer.

Ensuring that the EBS Routing Service Propagates the Transaction

For more information, see Chapter 3: Designing and Developing EBSs.

Ensuring that the Provider ABC Service Participates in the Transaction

For more information, see Chapter 4: Designing and Constructing ABC Services.

Ensuring that the Provider ABC Service Invokes MDM with the Global Transaction in Place

For more information, see Chapter 4: Designing and Constructing ABC Services.

Transform the Message in the JMS Consumer to Place it in a Wrapper Schema

In Participating Applications Publish Updates to MDM or Master Application: Solution Option 2 and Participating Applications Publish Updates to Other Participating Applications: Solution Option 2, participating applications push messages to a single queue and therefore, they must be sequenced.

To sequence them, the messages must be transformed into a common wrapper schema. Once they have been transformed, the Oracle ESB Resequencer service can be invoked.

A common schema needs to be designed for this purpose. At this point, Oracle AIA does not provide a schema for this purpose.

Appendix A: Oracle AIA Naming Standards

This appendix describes the naming standards and packaging structure to be used when developing integration scenarios using Oracle Application Integration Architecture (AIA). The appendix focuses on delivered components that comprise integration scenarios built on the platform.

The appendix covers the naming and packaging standards for enterprise business services (EBSs) and application business connector (ABC) services whether written in Enterprise Service Bus (ESB) or BPEL, as well as any other supplementary components used to achieve a complete integration scenario.

This appendix provides general guidelines and discusses naming standards for:

- XML.
- EBSs.
- Enterprise business flows (EBFs).
- ABC services.
- Common types.
- EBS.
- Domain value mapping (DVM) and cross references.
- BPEL.
- Custom Java classes.
- Package structure.
- Naming standard summary sheets.

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. These general naming standards should be applied whenever applicable:

- Avoid abbreviations

  Abbreviations may be ambiguous. The names used need to 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. This will make 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

These 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 make it easier to 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 these sections, especially naming rules for EBS and ABC services.

- All namespaces must start with "http://xmlns.oracle.com/".
- Namespaces used by EBOs and EBMs will 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 ABC services 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 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 this 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.

Prefixes	Namespace Pattern	Files
corecomcust	http://xmlns.oracle.com/EnterpriseObjects/Core/Custom/Common/V1	CustomCommonComponents, CustomReferenceComponents
coreinvcust	http://xmlns.oracle.com/EnterpriseObjects/Core/Custom/EBO/Invoice/V1	CustomInvoiceEBO
Corecom	http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V1	CommonComponents. Reference Components
Coreinv	http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V1	InvoiceEBO
Coreinv	http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V1	InvoiceEBM
telcocomcust	http://xmlns.oracle.com/EnterprisObjects/Industry/Telco/Custom/Common/V1	CustomCommonComponents, CustomReferenceComponents
telcoinvcust	http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/Custom/EBO/Invoice/V1	CustomInvoiceEBO
Telcocom	http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/Common/V1	CommonComponents. Reference Components
Telcoinv	http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/EBO/Invoice/V1	InvoiceEBO
Telcoinv	http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/EBO/Invoice/V1	InvoiceEBM

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.

Application	Short Name	Abbreviation
Oracle E-Business Suite	Ebiz	EBIZ
Oracle Siebel	Siebel	SEBL
Oracle PeopleSoft	PeopleSoft	PSFT
Oracle JD Edwards Enterprise One	JDEOne	JDE1
Oracle JD Edwards World	JDEWorld	JDEW
Oracle Transportation Management Suite	Logistics	LOGIS
Oracle Telephony @Work	Telephony	TELE
Oracle Demantra	Demantra	DMTR

Application	Short Name	Abbreviation
Oracle Communications Billing and Revenue Management	Portal	PORTAL
Oracle Retail Applications	Retek	RETEK

EBSs

EBSs are typically routing ESB services with routing rules to invoke the appropriate ABC services and initiate cross-functional BPEL flows.

EBS Namespace

EBSs will be published with this namespace pattern:

- http://xmlns.oracle.com/EnterpriseServices/Core/[EBO Name]/V[VersionNumber]
- http://xmlns.oracle.com/EnterpriseServices/Industry/[Industry Name]/{EBO Name}/V{VersionNumber}

Examples:

- http://xmlns.oracle.com/EnterpriseServices/Core/Invoice/V 1
- http://xmlns.oracle.com/EnterpriseServices/Industry/Telco/Invoice/V 1

EBS Name

Typically there is one EBS per enterprise business object (EBO) type. The service must have an operation per verb defined for the EBO as. The operations will be paired with the enterprise business messages (EBMs) for the EBO. The EBS service does not have any kind of processing beyond delegation to the ABC implementation services. The EBS service is typically implemented as an ESB routing service.

The enterprise business service local name should be the noun name with 'EBS' suffix:

- For Core: {EBO Name}EBS
- For Industry: {Industry Name} {EBO Name}EBS

  Example: OrderEBS, InvoiceEBS, AccountEBS, TelcoInvoiceEBS

  The version number will be appended to the name as:

- For Core: {EBO Name}EBSV{Version Number}
- For Industry: {Industry Name} {EBO Name}EBS V{Version Number}

  Example: OrderEBSV2, InvoiceEBSV2, AccountEBSV2, TelcoInvoiceEBSV2

EBS WSDL

Follow these naming standards for EBS WSDL.

WSDL File Name

The WSDL file name must be named exactly after the service name, that is the EBS name.

Example: AccountEBS.wsdl, OrderEBS.wsdl, TelcoInvoiceEBS.wsdl

The version number will be appended to the name as:

Example: AccountEBSV2.wsdl, OrderEBSV2.wsdl, TelcoInvoiceEBSV2.wsdl

Namespaces

Namespaces are defined in accordance to the EBS namespace standard. The namespace prefixes must follow the prefixes naming standard.

Types

Types must not be defined inside the WSDL file.

All used types need to be included or imported from externally defined schema URIs

Message

Follow these guidelines:

- The message name must be the operation name with a 'ReqMsg' suffix for input messages and a 'RespMsg' suffix for output messages.

  Example: CreateInvoiceReqMsg

- Request messages will be: {Operation Name}ReqMsg

  Example: for CreateOrderEBM: CreateOrderReqMsg

- Response messages will be: {Operation Name}RespMsg.

  Example: For CreateOrderEBM: CreateOrderRespMsg

- For patterns other than Request - Response, prefix 'Req' or 'Res' is not required.

Port Type and Operations

Port Type name is used to derive the Service Name as it appears in the concrete WSDL. However, ESB uses a different naming convention than BPEL. ESB adds 'Service' suffix to the port type to get the service name, while BPEL uses the port type as the service name.

In order to make sure that Service Names are consistent, this naming standard should be followed:

- ESB:

  Port Type = {Service Name}

- BPEL:

  Port Type = {Service Name}Service Example: EBS are ESB routing services, so the port type name is the same as the service name: OrderEBS, AccountEBS.

The operation name must map exactly to the corresponding EBM name without the 'EBM' suffix. For AcknowledgeInvoiceEBM the operation name will be 'AcknowledgeInvoice' which is basically the Verb + Noun Name.

Operation input will use the request message for the EBM in question, and operation output (if exists) will use the response message.

EBS WSDL Template and Sample

Please refer to the *TemplateEBS.wsdl and SampleEBS.wsdl* available in the Foundation Pack build.

EBF

These are the BPEL orchestration flows that are built as part of the integration.

Flow Namespace

EBFs are published with this namespace pattern:

- http://xmlns.oracle.com/EnterpriseFlows/Core/[flow name]/V[version number]
- http://xmlns.oracle.com/ EnterpriseFlows /Industry/[industry name]/[flow name]N[version number]

Examples:

- http://xmlns.oracle.com/EnterpriseFlows /Core/ProcessOrder/V1
- http://xmlns.oracle.com/ EnterpriseFlows /Industry/Telco/TelcoProcessOrder/V1

Flow Name

The EBF will be fired by certain verbs in an EBS such as Process in an OrderEBS. These BPEL flows will be named as:

- For Core - [Verb][EBO Name]EBF where EBF stands for: Enterprise Business Flow.
- For Industry - [Industry Name] [Verb] [EBO Name]EBF where EBF stands for: Enterprise

Business Flow.

Example: ProcessOrderEBF, VerifyCustomerEBF, TelcoBillingOrderEBF

The version number will be appended to the name as:

- For Core - [Verb] [EBO Name]EBFV[Version Number]
- For Industry - [Industry Name] [Verb] [EBO Name]EBF[Version Number]

Example: ProcessOrderEBFV2, VerifyCustomerEBFV2, TelcoBillingOrderEBFV2

BPEL Process WSDL

The WSDL of the EBF BPEL process is a subset of the EBS WSDL as it should contain an entry point operation matching an operation on the EBS WSDL.

The BPEL Process WSDL file name must match the process name.

Example: CustomerVerificationEBF.wsdl. or TelcoProcessSalesOrderEBF.wsdl

The version number will be appended to the name as: CustomerVerificationEBFV2.wsdl or TelcoProcessSalesOrderEBFV2.wsdl

ABC Service

ABC services are of two categories: requestor and provider. The requestor ABC service is a service which participating applications call to request or process data. The provider ABC service is a service called by the EBS to call out to participating application to provide certain functionality.

The per operation ABC Implementation services are front-ended by an Interface façade service that provides a single view to the implementation services which makes it easier for customer to reconfigure end points. It also provides protocol portability by allowing customers to insert protocol adapters between the interface services and the participating application.

Typically there is one Requester ABC Interface ESB Routing Service per noun per participating application version

Requestor ABC Service

Requester ABC services are the services that serve requests coming from client applications and process these requests and delegate them to the EBSs. The Requester ABC service consists of one Interface ESB routing service that routes the application calls into the appropriate ABC Implementation service. There is one ABC Implementation service per verb per noun per application. The implementation services can be implemented using ESB Routing services, and for complex scenarios it can be implemented as a BPEL process.

Requestor ABC Implementation Services Naming Standards

Follow these guidelines for the requester ABC Implementation services:

Namespace

ABC Implementation services will be published with this namespace pattern:

http://xmlns.oracle.com/ABCSImpl/{Participating Application Name}/{Core/ or Industry/[Industry Name]}/{Service Name}/V{version}

The version portion of the namespace indicates the version of the service API that needs to be in-sync with the application API. It is common that applications public APIs exposed as services may evolve with versions independent of the application version. For example in E-Business Suite R12 you may have two versions for the same public service, while some other Service has the same version both in R12 and R11. The mapping between the ABC service version and the actual application API version can be tracked as annotations on the ABC service WSDL.

Examples:

- http://xmlns.oracle.com/ABCSImpl/Siebel/Core/CreateInvoice/V1
- http://xmlns.oracle.com/ABCSImpl/Portal/Industry/Telco/CreateInvoice/V1

Local Name

The Requester ABC Implementation service local name should be operation name followed by the application functional group name followed by the participating application name, followed by the 'ReqABCSImpl' suffix.

- For Core - [Verb] [Entity Name] [Short Application Name]ReqABCSImpl
- For Industry - [Verb][Entity Name][Short Application Name][Industry Name}ReqABCSImpl

Examples:

- ProcessOrderSiebelReqTelcoABCSImpl
- UpdateCustomerPortalReqABCSImpl

The version number will be added as a suffix as:

- For Core - [Verb] [Entity Name] [Short Application Name]ReqABCSImplV[Version Number]
- For Industry - [Verb] [Entity Name][Short Application Name][Industry Name}ReqABCSImpl V[Version Number]

Examples:

- ProcessOrderSiebelReqTelcoABCSImplV2
- UpdateCustomerPortalReqABCSImplV2

Requestor ABC Implementation Services WSDLs

The requester ABC Implementation services WSDLs is application specific. The starting point for crafting the WSDL is usually the application business message ABM type.

The WSDL file name must be named exactly after the interface or implementation service names.

Example:

Requester ABC Implementation service: CreateOrderSiebelReqTelcoABCSImpl.wsdl

The version number will be added as a suffix.

Examples: ProcessOrderSiebelReqTelcoABCSImplV2.wsdl, UpdateCustomerPortalReqABCSImplV2.wsdl

Requester ABC Implementation WSDL Template

Please refer to the *TemplateAppRequesterABCSImpl.wsdl* and the *SampleAppRequesterABCSImpl.wsdl*.

Provider ABC Services

Provider ABC services are the services that EBSs call to request/post information to participating applications. Provider ABC service consists of one Provider Application Interface ESB routing service that front-ends all ABC Implementation services. There is one provider application interface service per noun per participating application and one ABC Implementation service per verb per noun per participating application. The implementation services can be implemented using ESB Routing services, and for complex scenarios it can be implemented as a BPEL process.

Provider ABC Services Naming Standards

Follow these guidelines for Provider ABC services:

Namespace

ABC Implementation services will be published with this namespace pattern:

http://xmlns.oracle.com/ABCSImpl/ApplicationName/{Core/ or Industry/[Industry Name]}/{service name}/V{VersionNumber}

The version portion of the namespace indicates the version of the service API that needs to be in-sync with the application API. It is common that applications public APIs exposed as services may evolve with versions independent of the application version. For example in E-Business Suite R12 you may have two versions for the same public service, while some other Service has the same version both in R12 and R11. The mapping between the ABC service version and the actual application API version can be tracked as annotations on the ABC service WSDL.

Examples:

- http://xmlns.oracle.com/ABCSImpl/Siebel/Core/Invoice/V1
- http://xmlns.oracle.com/ABCSImpl/Portal/Telco/Invoice/V1

Local Name

The Provider ABC Implementation service local name should be the EBO verb name followed by the noun name followed by the participating application name, followed by the 'ProvABCSImpl' suffix.

- For Core - {Verb} {Functional Group} {Application Short Name}ProvABCSImpl
- For Industry - {Verb} {Functional Group} {Application Short Name} {Industry}ProvABCSImpl

Examples:

- ProcessOrderSiebelProvTelcoABCSImpl
- UpdateCustomerPortalProvABCSImpl

The version number will be appended as suffix:

- For Core - {Verb} {Functional Group} {Application Short Name}ProvABCSImplV[Version Number]
- For Industry - {Verb} {Functional Group} {Application Short Name} {Industry}ProvABCSImpl V[Version Number]

Examples:

- ProcessOrderSiebelProvTelcoABCSImplV2
- UpdateCustomerPortalProvABCSImplV2

Provider ABC Interface and Implementation Services WSDLs

The provider ABC Interface service WSDL should match or be derived from the EBS WSDL because the types and operations should match with what is provided by the EBS WSDL.

The provider ABS Implementation service WSDL should be named the same as the service: CreateOrderSiebelProviderABCSImpl.wsdl, while the names of the WSDL components will follow whatever specified by the application.

The version number will be appended as suffix: Example:

CreateOrderSiebelProviderABCSImplV2.wsdl

Provider ABC Interface and Implementation WSDL Template

Please refer to the *TemplateAppProviderABCSImpl.wsdl* and the *SampleAppProviderABCSImpl.wsdl*.

Common Types

Developers may need to create types that are can be used in EBS or ABC services. These types may be used for Error Handling, Logging, Security,, and so on. These types are not publicly available and are just used by the implementation of the services in ESB or BPEL.

These types must conform to these naming standards:

Namespace

The types will be published with this namespace pattern:
http://xmlns.oracle.com/ABS/common/{type name}/V{version}

Examples:

- http://xmlns.oracle.com/ABS/common/ErrorHeader/V1
- http://xmlns.oracle.com/ABS/common/LoggingContent/V1

Local Name

{Type name}Type

Examples:

- ErrorHeaderType
- LoggingContentType

ESB

System and Group are units for organizing the ESB services. We recommend that the systems and groups reflect the functional classification of the services. Initially there are two Systems to represent horizontal and vertical services named Core and Industry. All horizontal services go under the Core system. Several sub-groups are created under the Industry group to represent the different industries such as: Telco, Retail, and LifeSciences.

Routing Services

The service is named as:

- EBS: {EBO Name}EBS

  Example: OrderEBS

- Requester ABC service implementation: [Verb][App Entity Name] [Short Application Name]ReqABCSImpl

  Example: CreateOrderSiebelReqABCSImpl

- Provider ABC service verb Implementation: [Verb][App Entity Name] [Short Application Name] [Optional
  Industry]ProvABCSImpl

  Example: CreateOrderSiebelProvABCSImpl

JMS Adapter Producer/Consumer Services

This table illustrates the naming standards for the JMS Adapter Producer/Consumer Services:

Artifact	Name	Example
Database User/Schema	JMSUSER	
Table Name	AIA_<EBO/ABO>JMSQTAB	AIA_CreateCustomerJMSQTAB
Queue Names	AIA_<EBO/ABO>JMSQueue	AIA Customer JMS Queue
Correlation ID	<Verb><EBO/ABO>	CreateCustomer
JMS Producer adapter enqueues message to AQ	<Verb><EBO /ABO><App>JMSProducer	CreateCustomerSiebelJMSProducer
JMS Consumer adapter service picks up message and routes to ABC service	<Verb><EBO /ABO><App>JMSConsumer	CreateCustomerSiebelJMSConsumer

DVMs and Cross References

When creating DVMs, these naming standards should be followed:

DVMs

When creating DVMs, these naming standards should be followed:

Map Name

Map names:

- Must start with the object name.

  This will allow 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, this naming standard should be followed:

Table Name

Table names:

- Must not exceed 48 characters.
- Must start with the object name.

  This will allow us 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

Follow these naming standards for BPEL activities:

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, ABC service 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, ABC service Impl, or Adapter Service.
  - The namespace must reflect the taxonomy of the process.
  - The namespace must include the major version number where appropriate.

Partner Link

Follow these guidelines:

- The name should follow the general standard naming standards.
- A partner link name is visible internally within the BPEL process.

- The partner link name must have the same name as the service it is linking to.

Pattern: <Service Name>

Example: CustomerService

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.
- Starts with the 'Compensate' prefix.
- Followed by the name of the scope encapsulating the tasks to be compensated.

Pattern: Compensate<scope name>

Example: CompensateProcessCreditCheckMilestone, Compensate TranseferFundsScope

Decide

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with the 'Decide' prefix.
- Followed by the name of the decision.

Pattern: Decide<name of decision>

Example: DecideRequiresManualApproval, DecideOverrideCreditLimit

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 'Invoice' prefix.
- Followed by the partner link to be invoked.
- Followed by 'Call' if synchronous invocation or 'Start' is 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

Reply

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with the 'Receive' prefix.
- Contains the name of the response message that is relevant to the message used in the corresponding
  Receive activity if applicable.

Pattern: Receive<Response Message Name>

Example: UpdateInvoiceResponseEBM

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 not intended to be presented in the end-user monitor, use 'Scope' as the suffix.

Pattern: <Name describing the Scoped Tasks>{ Scope |Milestone}

Examples: GetCreditRating Scope, 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: GetCustomerInfoSquence

Switch

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with the 'Switch' prefix.
- Followed by the name of 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.
- Starts with the 'Case' prefix.
- Followed by the name of 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: Terminate Timeout, 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

Sensors

Follow these guidelines:

- The name should follow the general standard naming standards.
- Starts with a name describing the nature of the sensor.
- Ends with 'Sensor' suffix.

Pattern: <Name describing the sensor>Sensor Example: CreditRatingSensor

Sensor Actions

Follow these guidelines:

The name should follow the general standard naming standards.

- Starts with the sensor name with the 'Sensor' suffix.
- Followed by the name describing the sensor action.
- Ends with 'Action' suffix.

Pattern: <Sensor Name>Sensor<Name describing the sensor action>Action

Example: CreditRatingSensorBAMFeedAction

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 class must follow the standard Java coding practices. The Java code components names types must conform to the Oracle Corporate Java Coding Standards: http://bali.us.oracle.com/bali/ojcs/front.html.

All of ABS 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 inline with our namespace naming standards. This will also allow us 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.

Note: 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

ESB and BPEL JDeveloper projects will be source controlled directly into ClearCase. The complete ABS source control packaging structure is described in the Infrastructure Release Installation and Packaging FDD. These projects may be source controlled under different hierarchy root directories to group them under core, industry, and projects within each industry.

Naming Standard Summary Sheets

Note: Only the first version of any service will not have a version number as the suffix - subsequent versions will have version number as the suffix.

EBS

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	OrderEBS.wsdl, CustomerEBS.wsdl, TelcoInvoiceEBS.wsdl
WSDL Message	Request message: [Verb][EBO Name]ReqMsg Response message: [Verb][EBO Name]RespMsg	CreateOrderReqMsg CreateOrderRespMsg
WSDL Port Type	Request – Response EBS or Request Only EBS : [ServiceName] Response EBS in one-way call: [Service Name]Response	OrderEBS, OrderEBSResponse, TelcoInvoiceEBS, TelcoInvoiceEBSResponse
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
ESB Routing Service Name	[Service Name]V[Version Number]	OrderEBS, OrderEBSV2, CustomerEBSV2,

Artifact	Name	Example
		TelcoInvoiceEBSV2

EBF

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

Requester ABC Service Implementation

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, …

Artifact	Name	Example
WSDL	[Service Name].wsdl	CreateOrderSiebelReqABCSImpl.wsdl UpdateOrderSiebelTelcoReqABCSImpl.wsdl CreateCustomerPortalTelcoReqABCSImpl.wsdl
WSDL Message	Request message: Verb[ABO Name]ReqMsg Response message: Verb[ABO Name]RespMsg	CreateOrderReqMsg, CreateOrderRespMsg UpdateOrderLinesReqMsg, UpdateOrderLinesRespMsg
WSDL Port Type	BPEL: [Service Name]Service ESB: [Service Name]	CreateOrderSiebelReqABCSImplService UpdateOrderSiebelReqABCSImpl
WSDL Port Type Operations	Verb[ABO Name] Should be self-describing to reflect the functionality needed by the application.	UpdateOrder, GetOrderLines, …
BPEL Flow Name/ESB Routing Service Name	[Service Name]V[Version Number]	CreateOrderSiebelReqABCSImplV2 UpdateOrderSiebelTelcoReqABCSImpl CreateCustomerPortalTelcoReqABCSImpl

Provider ABC Service Implementation

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 CreateCustomerPartyTelcoPortalP

Artifact	Name	Example
		rovABCSImpl.wsdl
WSDL Message	Request message: [Verb][ABO Name]ReqMsg Response message: [Verb][ABO Name]RespMsg	CreateOrderReqMsg, CreateOrderRespMsg
WSDL Port Type	BPEL: [Service Name]Service ESB: [Service Name]	CreateOrderSiebelProvABCSImpl Service UpdateOrderSiebelProvABCSImpl
WSDL Port Type Operations	[Verb][ABO Has one operation which should match the operations exposed by the corresponding EBS.	UpdateOrder, getOrderLines, …
BPEL Flow Name/ESB Routing Service Name	[Service Name]V[Version Number]	CreateOrderSiebelProvABCSImpl V2 UpdateOrderSiebelTelcoProvABC SImpl CreateCustomerPortalTelcoProvA BCSImpl

JMS Adapter Producer/Consumer Service for Requestors

Artifact	Name	Example
Database User/Schema	JMSUSER	
Table Name	AIA_<App><ABO>JMSQTABV[Version Number]	AIA_SiebelCustomerJMSQTAB AIA_EbizOrderJMSQTABV2
Queue Names	AIA_<App><ABO>JMSQueue V[Version Number]	AIA_SiebelCustomerJMSQueue AIA_EbizOrderJMSQueueV2
Correlation ID	<Verb><App><ABO> V[Version Number]	CreateSiebelCustomer CreateEbizOrderV2
JMS Producer adapter enqueues message to AQ	<Optional Verb><ABO><App>JMSProducer V[Version Number]	CreateCustomerSiebelJMSProducer OrderSiebelJMSProducerV2
JMS Consumer adapter service picks up message and routes to the ABC service	< Optional Verb><ABO><App>JMSConsumer V[Version Number]	CreateCustomerSiebelJMSConsume r OrderSiebelJMSConsumerV2

JMS Adapter Producer/Consumer Service for Providers

Artifact	Name	Example
Database User/Schema	JMSUSER	
Table Name	AIA_<App><ABO>ProvJMSQTABV[Version Number]	AIA_SiebelCustomerProvJMSQTAB AIA_EbizOrderProvJMSQTABV2
Queue Names	AIA_<App><ABO>ProvJMSQueue V[Version Number]	AIA_SiebelCustomerProvJMSQueue AIA_EbizOrderProvJMSQueueV2
Correlation ID	< Optional Verb><App><ABO>Prov V[Version Number]	CreateSiebelCustomerProv EbizOrderProvV2
JMS Producer adapter enqueues message to AQ	< Optional Verb><ABO><App>ProvJMSProducer V[Version Number]	CreateCustomerSiebelProvJMSProducer OrderSiebelJMSProvProducerV2
JMS Consumer adapter service picks up message and routes to the ABC service	< Optional Verb><ABO><App>ProvJMSConsumer V[Version Number]	CreateCustomerSiebelProvJMSConsumer OrderSiebelProvJMSConsumerV2

JMS Adapter Producer/Consumer Service for EBM

Artifact	Name	Example
Database User/Schema	JMSUSER	
Table Name	AIA_<EBO>JMSQTABV[Version Number]	AIA_CustomerJMSQTAB AIA_OrderJMSQTABV2
Queue Names	AIA_<EBO>JMSQueue V[Version Number]	AIA_CustomerJMSQueue AIA_OrderJMSQueueV2
Correlation ID	< Optional Verb><EBO> V[Version Number]	CreateCustomer OrderV2
JMS Producer adapter enqueues message to AQ	< Optional Verb><EBO>JMSProducer V[Version Number]	CreateCustomerJMSProducer OrderJMSProducerV2
JMS Consumer adapter service picks up message and routes to the ABC service	< Optional Verb><EBO>JMSConsumer V[Version Number]	CreateCustomerJMSConsumer OrderJMSConsumerV2

Participating Application Service

Artifact	Name	Example
Name	Registered in ESB with the same name as exposed by the application.	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	

Appendix B: XSL for Use in 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.

AddTargetSystemID.xsl

This section provides XSL that should be used in developing provider application business connector (ABC) services to be CAVS-enabled.

For more information about how to use this XSL, see Chapter 12: Developing Oracle AIA Services to be CAVS-Enabled and Provide Multi-Instance Support.

```
<?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.s
ervices.
  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/xpa
th"

xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.s
ervices.
  functions.ExtFunc"

xmlns:corecom="http://xmlns.oracle.com/EnterpriseObjects/Core/Common
/V2"

xmlns:aia=http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.c
ore.
```

```
  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

This section provides XSL that should be used in developing requester ABC services to be CAVS-enabled

For more information about how to use this XSL, see Developing Requester ABC Services to be CAVS-Enabled and Provide Multi-Instance Support.

```
<?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:Mes
sage
```

```
     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>
```

Appendix C: Delivered Oracle AIA XPath Functions

This appendix provides details about these XPath functions that are delivered with the Oracle Application Integration Architecture (AIA) Foundation Pack for use in Oracle AIA process integration packs (PIPs):

- aia:getSystemProperty()
- aia:getSystemModuleProperty()
- aia:getServiceProperty()
- aia:getEBMHeaderSenderSystemNode()
- aia:getSystemType()
- aia:getErrorMessage()
- aia:getCorrectiveAction()
- aia:isTraceLoggingEnabled()
- aia:logErrorMessage()
- aia:logTraceMessage()
- aia:getNotificationRoles()
- aiautil:getAIALocalizedString()

These functions can be called from BPEL and XSLT, as well as Enterprise Service Bus (ESB) 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 will be 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',tru
e())"/>
```

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 will be returned. If the system property with the same name is not found, then either an exception will be thrown or an empty string is returned, depending on the value of the `needAnException` parameter.

Usage

XSLT example:

```
<xsl:variable name="errHdlrImpl" select="aia:get
SystemModuleProperty('ErrorHandler','COMMON.ERRORHANDLER.
IMPL',true())"/>
```

BPEL example:

```
<assign name="AssignVar">
    <copy>
        <from expression="aia:getSystemModuleProperty('
         ErrorHandler','COMMON.ERROR
         HANDLER.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 will be returned. If the system property with the same name is not found, then either an exception will be 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/UpdateCustomerPartySiebelReqABCSImpl
/V2}UpdateCustomerPartySiebelReqABCSImplV2','Default.Syst
emID',true())"/>
```

BPEL example:

```
<assign name="AssignVar">
    <copy>
        <from expression= "aia:getServiceProperty
         ('{http://xmlns.oracle.com/ABCSImpl/
         Siebel/Core/UpdateCustomerPartySiebelReq
```

```
                    ABCSImpl/V2}UpdateCustomerPartySiebelReq
                    ABCSImplV2','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

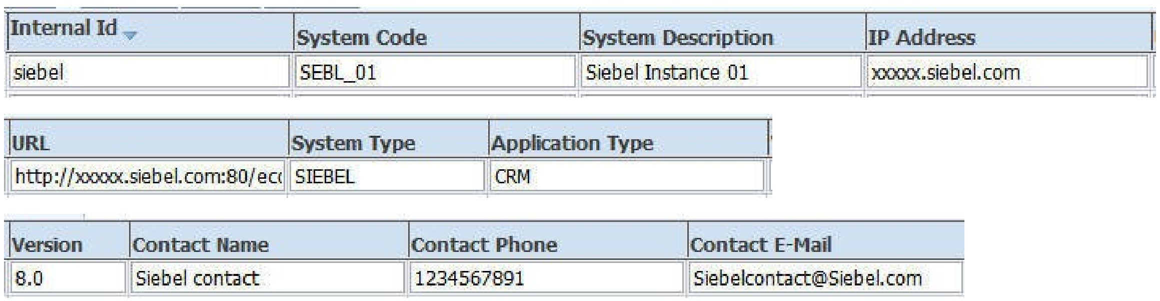
`senderSystemCode`	System Code as defined in BSR Systems.
`senderSystemID`	System Internal ID as defined in BSR Systems.

Returns

Returns a node-set of system information from the Business Service Repository (BSR) for the given System Code or Internal ID.

Usage

Given this set up in the BSR on the Application Registry page:



Internal Id	System Code	System Description	IP Address
siebel	SEBL_01	Siebel Instance 01	xxxxx.siebel.com

URL	System Type	Application Type
http://xxxxx.siebel.com:80/ec	SIEBEL	CRM

Version	Contact Name	Contact Phone	Contact E-Mail
8.0	Siebel contact	1234567891	Siebelcontact@Siebel.com

Settings on the Application Registry page

For more information about the Application Registry page, see *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* "Using the BSR," Managing the Oracle AIA Application Registry.

Both `aia:getEBMHeaderSenderSystemNode('SEBL_01', '')` and `aia:getEBMHeaderSenderSystemNode('', 'siebel')` would return this node-set:

```
<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>
```

XSLT example:

Note: This snippet requires `<xsl:stylesheet version="2.0" ...>`

```
<xsl:variable name="senderNodeVariable">
    <xsl:copy-of select="aia:getEBMHeaderSender
     SystemNode($RequestTargetSystemID,'')"/>
</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

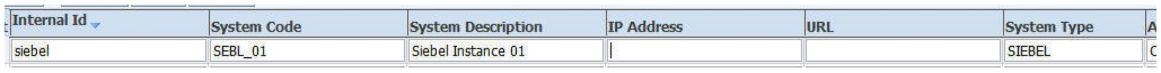
systemCode	System code as registered in the BSR database.

Returns

Returns the System Type associated with a System Code as registered in the BSR database.

Usage

Given this set up in the BSR on the Application Registry page:



Internal Id	System Code	System Description	IP Address	URL	System Type	A
siebel	SEBL_01	Siebel Instance 01	|		SIEBEL	C

Settings on the Application Registry page

For more information about the Application Registry page, see *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* "Using the BSR," Managing the Oracle AIA Application Registry.

The result of `aia:getSystemType('SEBL_01')` would be the string '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`	The error code.
`localeString`	Delimited locale string.
`Delimiter`	Delimiter used in the `localeString`.

Returns

This function will lookup 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.

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:Fault
         Notification/corecom:FaultMessage/
         corecom:Code"/>
    </copy>
    <copy>
        <from expression="aia:getErrorMessage('
         AIA_ERR_AIAO2C2_1007','','')"/>
        <to variable="AIAFaultMsg" part="AIAFault"
         query="/corecom:Fault/corecom:Fault
         Notification/corecom:FaultMessage/
         corecom:Text"/>
    </copy>
</assign>
```

aia:getCorrectiveAction()

```
namespace aia="g aia:getCorrectiveAction (string correctiveActionCode,
string localeString, string delimiter)
```

Parameters

`correctiveActionCode`	The corrective action code.
`localeString`	Delimited locale string.
`Delimiter`	Delimiter used in the `localeString`.

Returns

This function will lookup the Corrective Action Code resource bundle and return 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.

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 or not the specified `logLevel` is enabled for the specified process.

This function will do two things:

1. Check whether logging is enabled or disabled for the input process name from the AIAConfigurationProperties.xml file.
2. Check if the trace logger log level is set to log the input log level.

This function will return a true only when both these conditions return true, otherwise it will return 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.

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.

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. Level should be one of 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.

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`	The error code.
`serviceName`	Name of the errored service.

`processName`	Name of end-to-end process.

Returns

This function will query the BSR with input values and return 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 the BSR, this function returns the default roles specified in the AIAConfigurationProperties.xml file.

aia:getAIALocalizedString()

```
namespace
aia="http://www.oracle.com/XSL/Transform/java/oracle.apps.aia.core.x
path.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. Currently accepted values are: "Telco/SalesOrder", "Telco/CustomerParty", "Telco/BillingManagement", or "Telco/ProductLifeCycle".
`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. This mapping is used at runtime to determine which resource bundle class to use for looking up the localized string.

Example of the module-level 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

BPEL example:

```
<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;localizedString
           Params&quot;))"/>
          <to variable="internationalizedstring"/>
     </copy>
</assign>
```

Index