Oracle® Application Server Integration InterConnect

User's Guide

10*g* Release 2 (10.1.2)

B14069-02

December 2005

ORACLE®

Oracle Application Server Integration InterConnect User's Guide, 10*g* Release 2 (10.1.2)

B14069-02

Primary Author: Rima Dave

Contributing Author: Vimmika Dinesh

Contributor: Rahul Pathak, Maneesh Joshi, Harish Sriramulu, Aska Onoda, Aruna Kasturi, Sandeep Jain

Contents

5 Using Procedures in iStudio

6 Enabling Infrastructure

7 Using Oracle Workflow

11 Using Enterprise Manager

12 Using InterConnect Manager

A InterConnect Interoperability Feature with Plug-in for BPEL

B Integration Scenario

C Using the Data Definition Description Language

D Transformations

List of Figures

Preface

This Preface contains these topics:

Audience

This guide is targeted at the following types of users:

- Business analysts and integration engineers, for iStudio.
- System Administrators, for the run-time component.

The audience should have the following prerequisites, which are discussed but not explained:

- Domain knowledge of the applications being integrated.
- Database concepts and working knowledge of SQL, PL/SQL, or SQL* Plus.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, refer to these Oracle resources:

- *Oracle Application Server Integration InterConnect Installation Guide*
- *Oracle Application Server Integration InterConnect Release Notes*

Conventions

The following text conventions are used in this document:

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

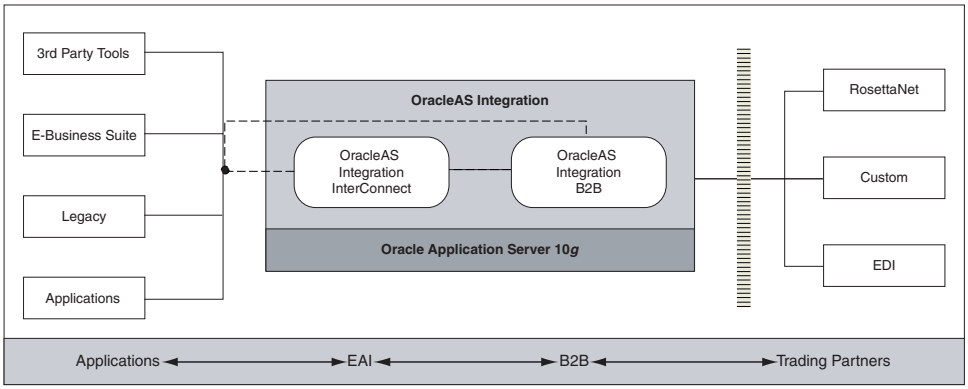
1

Getting Started with OracleAS Integration InterConnect

This chapter provides an overview of Oracle Application Server Integration InterConnect (OracleAS Integration InterConnect), its features, and components. It contains the following topics:

1.1 What is OracleAS Integration InterConnect?

OracleAS Integration InterConnect is an integral component of Oracle Application Server and provides a comprehensive application integration framework to enable integration of enterprise software. It is built on top of the Oracle Application Server integration platform and leverages its underlying services. It is designed to integrate heterogeneous systems, such as Oracle, non-Oracle, and legacy applications. Together with Oracle Application Server Integration B2B (OracleAS Integration B2B), it provides a complete end-to-end solution for integrating your enterprise and beyond. OracleAS Integration B2B provides extensive protocol support to enable the deployment of industry-recognized Business-to-Business (B2B) standards: RosettaNet, Electronic Data Interchange (EDI), Applicability Statement 2 (AS2), and custom configurations.



OracleAS Integration InterConnect provides the following benefits:

- Speed of Integration Development: Elevate the integration problem from a technical coding exercise to a functional modeling exercise, thereby reducing or eliminating the programming effort normally associated with integration. This ensures that the development time is reduced significantly.
- Speed of Run-Time Integration Execution: Minimize latency and maximize throughput for real-time cross-application integration. This ensures that performance is enhanced.
- Speed of Integration Evolution: Expose an integration methodology that promotes reuse of existing integration logic and minimizes change impact. As the integration scenario evolves over time (existing applications are upgraded, new applications are added, old applications are removed), the change impact is limited to just the application that is undergoing the change. The other applications are shielded from these changes. This reduces the complexity and management issues that arise over the integration lifecycle.

1.1.1 OracleAS Integration InterConnect Components

OracleAS Integration InterConnect has the following core components:

- OracleAS Integration InterConnect Hub
- OracleAS Integration InterConnect Adapters
- OracleAS Integration InterConnect Development Kit

1.1.1.1 OracleAS Integration InterConnect Hub

The hub consists of a middle-tier repository server program communicating with a database. The repository has the following functionality:

- At design time, all integration logic defined in iStudio is stored in tables in the repository as metadata.
- At run time, the repository provides access to this metadata for adapters to integrate applications.

The repository server is deployed as a standalone Java application running outside the database. The repository schema is a set of tables in the Oracle Application Server hub database. The adapters and iStudio connect to the repository server using Remote Method Invocation (RMI).

1.1.1.2 OracleAS Integration InterConnect Adapters

Adapters perform two major tasks:

- Provide connectivity between an application and the hub.
- Transform and route messages between the application and the hub.

Adapters are deployed as standalone Java applications running outside the database. Adapters can be deployed in the following configurations:

- Co-located with the OracleAS Integration InterConnect Hub
- Co-located with the application adapters are connecting to
- Located on a separate computer

See Also: "Using Adapters for Integration" on page 1-8

1.1.1.3 OracleAS Integration InterConnect Development Kit

iStudio is a design-time integration specification tool targeted at business analysts. This tool helps business analysts specify the integration logic at a functional level, instead of a technical coding level. iStudio exposes the integration methodology using wizards and reduces, or eliminates, the need for writing code to specify the integration logic. This reduces the total time required to complete an integration.

iStudio is a multiuser tool with fine-grained locking for all OracleAS Integration InterConnect first class objects. As a result, multiple users can work simultaneously on the same integration scenario without compromising the consistency of the metadata.

iStudio enables business analysts to complete the following tasks:

- Define data to be exchanged across applications
- Semantically map data across applications
- Define the business process collaboration across applications using Oracle Workflow and associate the semantic maps with business processes, if required
- Configure and deploy the integration

iStudio is deployed as a standalone Java application running outside the database. It can be deployed on any computer with access to the hub computer running Windows.

See Also: Chapter 2, "Using iStudio"

1.1.1.3.1 OracleAS Integration InterConnect SDKs OracleAS Integration InterConnect Software Development Kit (SDK) enables you to customize OracleAS Integration InterConnect to meet your integration needs.

iStudio SDK The iStudio SDK is a collection of Java `jar` and `Javadoc` files usually deployed on the same computer as iStudio. The iStudio SDK is available only on Windows. Using java and the iStudio SDK, you can build the following:

- New transformation functions
- New browsers to import application-native data structures and APIs into iStudio

Documentation and samples are provided with the iStudio SDK.

Adapter SDK The Adapter SDK is a collection of Java `jar` and `Javadoc` files that can be deployed on any computer. The Adapter SDK is available on all tier-one platforms. Using the Adapter SDK, users can write new adapters in Java for applications or protocols not currently supported by OracleAS Integration InterConnect. Specifically, only the bridge subcomponent must be written. The agent is a generic engine already written and is part of each adapter.

Oracle Workflow Oracle Workflow provides a comprehensive business process management system that enables traditional workflow applications and process collaboration in a single solution. Using Oracle Workflow Business Event System, OracleAS Integration InterConnect can model an integration solution on business processes. With OracleAS Integration InterConnect and Oracle Workflow, business collaborations across two or more applications can be defined to implement the organization's business processes.

1.2 Standard Messaging

OracleAS Integration InterConnect provides the following basic services expected of a messaging middleware platform:

- Guaranteed delivery of messages: All messages have guaranteed end-to-end delivery. A message is delivered exactly once and in the order sent.
- Scalability: Multiple adapters are instantiated to serve one application. The hub runs in an Oracle Real Application Clusters environment.
- Load Balancing: Messages can be partitioned based on the load between multiple adapters servicing one application. One or more adapters can serve all messages for one application. In addition, one or more adapters can be dedicated for each integration point in which the application participates.
- Run-Time Management: The Oracle InterConnect Manager helps to manage the integration scenario and components at run time. The Oracle Enterprise Manager 10*g* Application Server Control and IC Manager enable users to start and stop components, monitor message flow, detect problems, and manage errors.
- Deployment Support: The messaging hub consists of Advanced Queues that are configured for run time. You can configure the number of queues to create, name these queues, and match adapters with messages in a specific named queue.

The following supplementary features do not require any additional coding:

- Content-Based Routing: Route messages by building business rules based on message content. For example, a procurement system routes fulfillment requests to different fulfillment centers based on an originating location.
- Cross-Referencing: Correlate keys that uniquely identify the entities in one application with corresponding entities created in other applications. For example, a purchase order created in a procurement system has a native ID *X*. The purchase order is then routed to a fulfillment system, where it is created with native ID *Y*. As a result, *X* and *Y* must be cross-referenced for OracleAS Integration InterConnect to correlate communication about this same logical entity in two different systems without each system interpreting the native ID of the other system.
- Domain Value Mapping: Map code tables across systems. For example, a purchase order in a procurement system has a PO Status field with domain values, `Booked` and `Shipped`. The corresponding field in a fulfillment system has the domain values `1` and `2`. OracleAS Integration InterConnect enables the user to create the mappings `booked=1`, `shipped=2` so that it can correlate these values at run time without each system interpreting the domain value set of the other system.

1.2.1 Supported Messaging Paradigms

OracleAS Integration InterConnect supports the following messaging paradigms. These paradigms are defined in iStudio at design time. The definitions are used at run time to route the messages suitably:

- Publish/Subscribe Messaging: An application publishes a message if it sends data to the OracleAS Integration InterConnect hub without knowing the destination applications. In addition, data is not expected in return. An application subscribes to a message if it receives the data from the OracleAS Integration InterConnect hub regardless of the application that sent the data. Also, it does not send any data in return. Events in iStudio are used to model this paradigm.
- Request/Reply Messaging: An application publishes a message and expects a message in return as a reply. The application subscribing to the request sends a reply to the sender after processing the request. Procedures in iStudio are used to model this paradigm. Request/Reply has two types of messaging:

- Synchronous: The application making the request is blocked until it receives a reply.
- Asynchronous: The application makes the request and proceeds with processing. It does not wait for a response. A reply is delivered asynchronously and is consumed by the application.

- Point-to-Point Messaging: Both Publish/Subscribe and Request/Reply can acquire a point-to-point characteristic if the sending application explicitly specifies which application should receive the message. This can be modeled using content-based routing in iStudio.

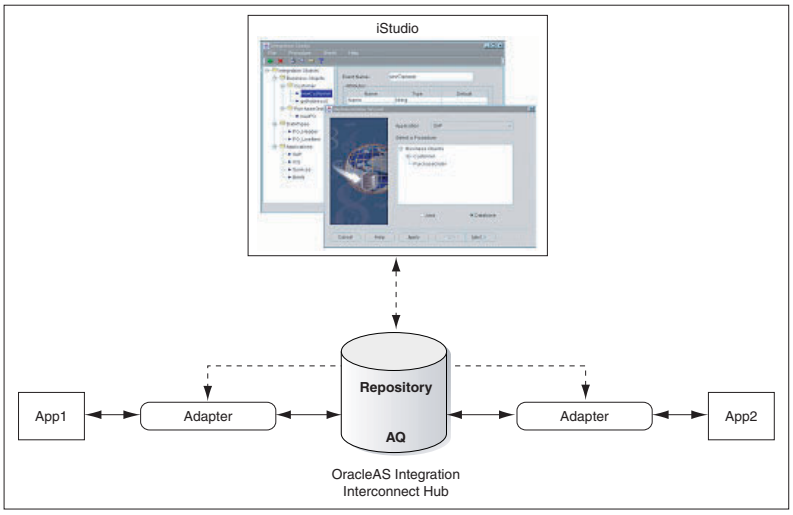
1.3 OracleAS Integration InterConnect Integration Process

Application integration using OracleAS Integration InterConnect involves the following two phases:

- Design Time
- Run Time

Figure 1–1 provides an overview of design-time and run-time phases in integration.

Figure 1–1 A Graphical Overview of Design-Time and Run-Time Phases in Integration



1.3.1 Design Time

During design time, a business analyst uses iStudio to define the integration objects, the applications that participate in the integration, and the specifications of the data exchanged between applications. All the specifications are stored as metadata in the OracleAS Integration InterConnect Repository. In addition, one or more OracleAS Integration InterConnect adapters are configured to service each application participating in the integration.

1.3.2 Run Time

At run time, when an application sends messages, the adapters attached to it retrieve the metadata from the repository, determine message format, perform transformations, and route the message to corresponding queues in the OracleAS Integration InterConnect hub.

For applications receiving messages, the adapters retrieve the metadata from the repository to receive messages from the OracleAS Integration InterConnect hub queues, determine message formats, perform transformations, and then deliver the messages to the application.

1.3.3 Separation of Integration Logic and Platform Functionality

Integration using OracleAS Integration InterConnect is a two-step process. During design time, integration logic is modeled in iStudio and captured in the repository as metadata. Metadata is created in the repository using iStudio during design time and is represented by application views, common views, and transformations. At run time, the underlying services treat this metadata as run-time instructions to enable the conversation among participating applications. Integration has two components:

- Integration logic: Consists of the business rules and transformation logic necessary to integrate heterogeneous systems. Using iStudio, this integration logic can be modeled and the results stored in the repository as metadata.
- Platform functionality: Consists of the integration infrastructure provided with OracleAS Integration InterConnect and the Oracle Database. In addition, OracleAS Integration InterConnect provides application and protocol adapters. The platform services provide the infrastructure necessary for integration.

1.3.4 Unique Integration Methodology

iStudio exposes an integration methodology that eliminates the complexities of point-to-point custom integration solutions. The integration methodology is based on a hub-and-spoke model.

1.3.4.1 How the Hub-and-Spoke Model Works

An integration point is the context, which ties in a particular message exchange, between two or more participating applications in the integration scenario. OracleAS Integration InterConnect supports two types of integration points:

- Events: This type of integration point is used to model the publish/subscribe messaging paradigm.

  For example, an integration scenario may require that customer information across two applications be synchronized in real time. Whenever a new customer is created in the application, `App1`, the customer should also be created in the application, `App2`. `Create_Customer` is an *event* that triggers communication between the two applications. `App1` produces the information, `and App2` consumes it. In this scenario, `create_customer` is an integration point.

- Procedures: This type of integration point is used to model the request/reply messaging paradigm.

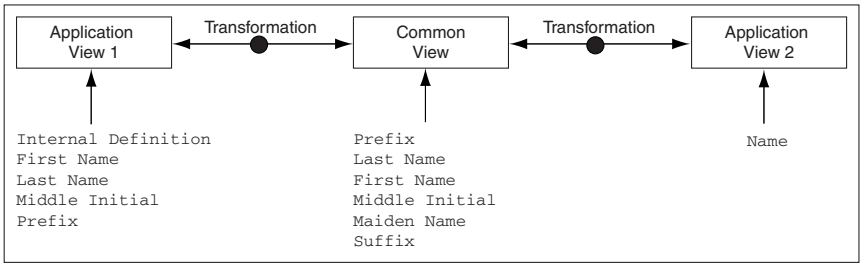
  For example, a user of an application `App1` may request information about an item stored in application `App1`. The information about the requested item might be segmented across the two applications `App1` and `App2`. To give a meaningful response to the user of `App1`, it is necessary to query `App2` for information about the item. `Get_Item_Info` is an integration point between the two applications because it triggers communication between the two applications. `App1` produces a query, and `App2` consumes it. `App2` produces the response, and `App1` consumes it.

The common view consists of a list of such integration points, each with its own associated data. Applications participate in the integration by binding to one or more of these common view integration points.

For each binding, applications have their own application view of data that needs to be exchanged. Each binding involves a mapping, or transformation, between the application view and the common view in the context of the integration point. In this model, the application views are the spokes and the common view is the hub.

Figure 1–2 illustrates an example of the hub-and-spoke model, where the common view is the hub and the application views are the spokes.

Figure 1–2 OracleAS Integration InterConnect Hub-and-Spoke Model



The application view for `App1` contains: `First Name`, `Last Name`, `Middle Initial`, and `Prefix`. Similarly, the application view for `App2` contains: `Name` (one field that describes `Last Name` and `First Name`).

In this example, `Create_Customer` is an integration point. If the information to exchange is only the new customer's name, the common view has all the information potentially captured in a name defined in an application-independent method. This information must be a superset of all the information that needs to be exchanged between `App1` and `App2`, for example, `Prefix`, `First Name`, `Last Name`, `Middle Initial`, `Maiden Name`, and `Suffix`.

When `App1` sends this information out or publishes an event, transformations are defined from its application view to the common view. When `App2` receives this information or subscribes to an event, transformations are defined from the common view to its application view.

The hub-and-spoke model has the following advantages:

- Loosely Coupled Integration: If applications are integrated directly with each other, then any change in one application will result in changes required for the other applications. In OracleAS Integration InterConnect, applications integrate to the common view and not directly with each other. This reduces the number of integration interfaces.
- Easy Customization: If an application is upgraded or changed, then only the corresponding application view needs to be remapped to the hub. The other spokes and their relationships with the hub remain unchanged. This localizes the change impact to the affected application.
- Easy Extensibility: If an application is added or removed from the integration scenario, then other integrated applications are not affected. For example, if a new application is added to the integration scenario, then it must define its spoke component (the application view) and map that component to the hub (common view) for each integration point. This does not affect other applications in the integration.
- Enhanced Reusability: If the common view of an application is already built, then this common view can be reused to integrate the application with any other application. For example, to integrate the Marketing CRM module to SAP, the integration would be from iMarketing to common view to SAP. If there is a

requirement to integrate iMarketing to Peoplesoft, then the iMarketing to common view integration can be reused. Only the common view to the Peoplesoft integration needs to be built.

1.3.5 Integration Lifecycle Management

Managing, customizing, and evolving an integration over time is as important as creating the integration in the first place. The hub-and-spoke integration model has advantages to help achieve this goal. In addition, the OracleAS Integration InterConnect repository, which contains all the integration logic, provides extensive services for managing changes over time. The repository provides fine-grained versioning of all OracleAS Integration InterConnect first class objects such as events, messages, and data types. Some important aspects of versioning to aid the lifecycle support include:

- Basic Versioning: New versions of first class objects, such as messages, can be created to address changing integration needs. Different versions of the same object can coexist in the repository. This approach has two advantages:
  - Eliminates the need for an expanded namespace to address modifications
  - Allows related entities to be grouped together for easy management
- Multiple Active Versions: Multiple versions of the same message can be active in the same integration scenario simultaneously. This helps transition and integration incrementally without requiring changes to existing messages. For example, if a purchase order definition for an application or the application view of the purchase order needs to change, then a new version of the message can be created and activated for that application. Once this metadata is created, the application can smoothly transition from sending and receiving messages based on the old definitions to the new one.
- Migration Support: Different versions of metadata can be migrated across repositories on a first class object basis. This feature enables fine-grained control of content in different repositories, such as a development repository and a production repository.
- Consistency Control: OracleAS Integration InterConnect detects and flags metadata conflicts. This helps prevent accidental overwriting of metadata and maintains consistency of metadata in the repository.

1.3.6 Using Adapters for Integration

Adapters are run-time components, which process integration logic captured in the repository as run-time instructions, to enable the integration. Prepackaged adapters help applications at run time to participate in the integration without any programming effort.

Adapters perform the following tasks:

- Application Connectivity: Adapters connect applications with OracleAS Integration InterConnect hub to transfer data between them. The logical subcomponent within an adapter that handles this responsibility is called a bridge. This is the protocol/application-specific piece of the adapter that communicates with the application.

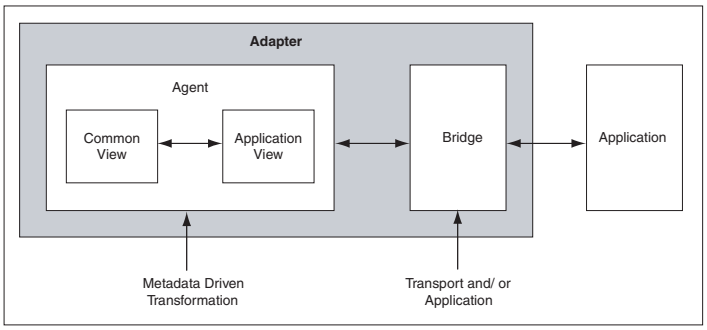
  For example, the database adapter can connect to an Oracle database using JDBC and run-time SQL APIs. The bridge subcomponent only knows how to call the correct APIs.

- Transformations: Transform data from the application view to common view and conversely as dictated by the repository metadata. In general, adapters are responsible for carrying out all the run-time instructions captured through iStudio as metadata in the repository. Transformations are an important subset of these instructions. The logical subcomponent within an adapter that handles the run-time instructions is called an agent. This is the generic run-time engine in the adapter that is independent of the application to which the adapter connects. It focuses on the integration scenario based on the integration metadata in the repository. There is no integration logic coded into the adapter itself. All integration logic is stored in the repository. The repository contains the metadata that drives this subcomponent.

  In the preceding database adapter example, the bridge subcomponent knows which SQL APIs to call but not how to call them. All adapters have the same agent code but the metadata is different. This difference in metadata controls and differentiates the behavior of each adapter.

The OracleAS Integration InterConnect Adapter Architecture is displayed in Figure 1–3.

Figure 1–3 OracleAS Integration InterConnect Adapter Architecture



See Also: *OracleAS Integration InterConnect Installation Guide* for a complete list of OracleAS Integration InterConnect Adapters.

1.4 What's New in This Release?

Oracle Application Server Integration InterConnect 10*g* Release 2 (10.1.2) introduces a number of new features to provide simplified configuration and management.

XML Schema Definitions Support

OracleAS Integration InterConnect 10*g* Release 2 (10.1.2) supports XML Schema Definitions (XSD). It enables you to import user-defined XML schemas for business objects. You can configure the run-time validation of XML message structures against the XSDs.

InterConnect Interoperability Feature

InterConnect Interoperability Feature contains a plug-in for BPEL. The BPEL plug-in enables interaction between OracleAS Integration InterConnect and Oracle BPEL Process Manager. It enables the use of BPEL processes defined in Oracle BPEL Process Manager for business process automation requirements in OracleAS Integration InterConnect.

Enterprise Manager

Oracle Enterprise Manager 10*g* Application Server Control Console is a GUI tool that enables you to manage the integration components at run time. You can configure hub, repository, and adapters using the Enterprise Manager. In addition, you can view the status and log files of integration components.

Enhancements in iStudio

OracleAS Integration InterConnect 10*g* Release 2 (10.1.2) provides enhanced iStudio UI capabilities that help you to use the iStudio in an effective way. The look and feel of icons in iStudio has changed.

JCA Adapter

A new adapter called the JCA adapter has been included in this release.

2

Using iStudio

This chapter describes iStudio and its concepts. It contains the following topics:

2.1 Overview of iStudio

The iStudio interface is a design-time integration specification tool used to help business analysts specify the integration logic at a functional level, instead of a technical coding level. iStudio exposes the integration methodology using simple wizards and reduces or eliminates the need for writing code to specify the integration logic. This reduces the total time required to complete an integration.

iStudio is a multiuser tool with fine-grained locking for all OracleAS Integration InterConnect first-class objects. This enables multiple users to work simultaneously on the same integration scenario without compromising the consistency of the metadata.

iStudio enables business analysts to perform the following tasks:

- Define the data that needs to be exchanged across applications.
- Semantically map the data across applications.
- Define the business process collaboration across applications and associate the semantic maps with business processes if required.
- Configure and deploy the integration.

iStudio is deployed as a standalone Java application running outside the database. iStudio runs only on Windows and can be deployed on any computer that has access to the computer where the hub is installed.

> **See Also:** *Oracle Application Server Integration InterConnect Installation Guide*

2.1.1 iStudio Concepts

The following concepts are described:

- Transformations or Mappings
- Metadata Versioning
- Tracking Fields
- Content-Based Routing
- Cross Reference Tables
- Domain Value Mapping
- Routing and the Message Capability Matrix

2.1.1.1 Applications

Each component integrated with OracleAS Integration InterConnect is referred to as an application. Each application expresses interest in specific messages, what its internal data type is, and how the message should be mapped to or from that internal type to the external world.

> **See Also:** Chapter 3, "Creating Applications, Common Views, and Business Objects"

2.1.1.2 Common Views and Business Objects

OracleAS Integration InterConnect follows a hub-and-spoke integration methodology. The common view is the *hub view* of the integration, in which each spoke is an application participating in the integration. The common view consists of the following elements:

- Business Objects: A collection of logically related integration points. For example, Create Customer, Update Customer, Delete Customer, and Get Customer Info are integration points that logically belong to a Customer business object.
- Events: An integration point used to model the publish/subscribe paradigm. An event has associated data, which is the common view of all the data to be exchanged through this event.
- Procedures: An integration point used to model the request/reply paradigm. This is a modeling paradigm only. No actual procedures are called. Like events, procedures have associated data that represents the common view of data exchanged through the procedure.
- Common Data Types: A data type used to define data for reuse. It is useful to define complex hierarchical data.

> **See Also:** Chapter 3, "Creating Applications, Common Views, and Business Objects"

2.1.1.2.1 Events An event is an integration point used to model the publish/subscribe paradigm. An event is associated with only one data set. An event has associated data that is the common view of all data to be exchanged through the event. The data associated with an event in the common view must be a superset of the data of participating applications. The publish/subscribe paradigm is used for asynchronous one-way communication. The sending application is said to publish the event. The receiving application subscribes to the event.

> **See Also:** Chapter 4, "Using Events in iStudio"

2.1.1.2.2 Procedures A procedure is an integration point used to model the request/reply paradigm. This is a modeling paradigm only. No actual procedures are called. The request/reply paradigm is used for two-way context-sensitive communication. This communication can be either synchronous or asynchronous. In synchronous communication, the requesting application is blocked until it receives a reply. In asynchronous communication, the requesting application gets the reply asynchronously, it does not block-wait for the response after sending the request. An application can either invoke a procedure to model sending a request and receiving a reply, or implement a procedure to model receiving a request and sending a reply. Similar to events, procedures have associated data. A procedure has two data sets, one for the request or `IN` data and one for the reply or `OUT` data.

See Also: Chapter 5, "Using Procedures in iStudio"

2.1.1.3 Transformations or Mappings

Transformations are used to map the application view of data to its corresponding common views and the common view to corresponding application view. This is used in the context of publishing or subscribing to an event or invoking or implementing a procedure. There are several built-in transformation routines provided with OracleAS Integration InterConnect that are used to build complex mappings. In addition, using the iStudio SDK enables new transformation routines to be created using Java. These transformations can be imported into iStudio and then used similar to a built-in routine.

For example, assume that there are two applications, `App1` and `App2`.

`App1` publishes the event, and its application view contains the following fields:

```
First Name

Last Name

Middle Initial
```

`App2` subscribes to the event, and its application view contains the following field:

`Name`: One field in the form of `LastName`, `FirstName`

When publishing or subscribing to the event, the application view for `App1` and `App2` must be mapped to the common view using transformations. Assume that the common view event contains the following fields:

```
Prefix

First Name

Last Name

Middle Initial

Suffix
```

So, there are two transformations performed. In the first, the data from `App1` is transformed to the common view, and in the second, the data from the common view is transformed to the application view of `App2`.

See Also:

- Chapter 5, "Using Procedures in iStudio"
- Appendix D, "Transformations"

2.1.1.4 Metadata Versioning

iStudio supports versioning for application and common data types, events, procedures, and messages.

An owner is the creator of the object and only the owner can modify the object. However, other users can create new versions or copy the original object under a new name. The owner is specified when the repository is installed.

The following functionality is available for versioning:

- Automatic Versioning: Whenever a new version is created, it automatically gets a version number. For example, assume that an event called `NewCustomerEvent` is created. When this object is created for the first time, the assigned owner is `OAI` and the version is `V1`. The event name is `NewCustomerEvent/OAI/V1`.
- Modify Object: The owner is the only user who can modify the contents of an event and the data associated with it. However, the owner cannot change the version number or the name of the event.
- Create New Version: If the owner wants to retain the original event but wants to create a new version of the information with modified data, then the owner can create a new version. For example, assume that `OAI` wants to retain the original version of the `NewCustomerEvent` object but wants a new version of the object with modified data. Using the Create New Version functionality will provide two objects, `NewCustomerEvent/OAI/V1` and `NewCustomerEvent/OAI/V2`.
- Load Version: Not all versions of objects are loaded into iStudio. To work with a specific version of an object, use the Load Version capability. When a new version is created, it becomes the current version.
- Copy Object: To create a new object that has many elements identical to an already existing object, first load the existing object and copy it in iStudio. Copying the object allows only modifications to the data and modifications to the name of the event. Once the name has been modified, the existing object and the object with the new name will coexist in the repository.

Note: Names of events must be unique.

See Also: Chapter 4, "Using Events in iStudio"

2.1.1.5 Tracking Fields

Tracking fields are one or more application view fields in the context of a particular message. If specified in iStudio, then tracking fields can be used to track messages at run time using the Oracle InterConnect Manager. Tracking is performed only from the perspective of the sending application.

For example, if `App1` publishes a new purchase order and specifies the `PO_order number` field as the tracking field, then you can log in to the run-time console and specify the message to track, or `New Purchase Order` in this case. You are then prompted to enter the purchase order number to display the corresponding tracking information.

2.1.1.6 Content-Based Routing

Content-based routing enables you to define rules to route messages based on message content. For example, a sales lead generation system can route leads to different sales force automation systems based on the location of the potential customer.

See Also: Chapter 6, "Enabling Infrastructure"

2.1.1.7 Cross Reference Tables

Keys for corresponding entities created in different applications can be correlated through cross-referencing in iStudio.

See Also: Chapter 6, "Enabling Infrastructure"

2.1.1.8 Domain Value Mapping

Code tables can be mapped across systems using domain value mapping in iStudio.

See Also: Chapter 6, "Enabling Infrastructure"

2.1.1.9 Routing and the Message Capability Matrix

In the OracleAS Integration InterConnect hub, advanced queues in the database are used to store, route, and forward messages from the sending application adapters to the receiving application adapters. The sending adapters evaluate the recipients based on the metadata. The following method is used to route messages.

- Every adapter has one or more queues where it receives messages.
- The Message Capability Matrix allows queues to be specified for receiving messages on a per-message per-receiving application basis.

Note: By default, OracleAS Integration InterConnect comes preconfigured with one queue named the `oai_hub_queue`. This queue is used for all messages and applications. This queue does not need to be changed unless the single-queue implementation causes a performance bottleneck.

2.2 Starting iStudio

The database and repository must be running before logging in to iStudio. To log in to iStudio, perform the following steps:

1. From the Windows Start menu, select **OracleAS Integration InterConnect**.
2. Select **iStudio**.
3. When iStudio starts, the last opened project is automatically loaded into the default workspace.

See Also: "Creating a New Project" on page 2-11

2.3 Parts of the iStudio Window

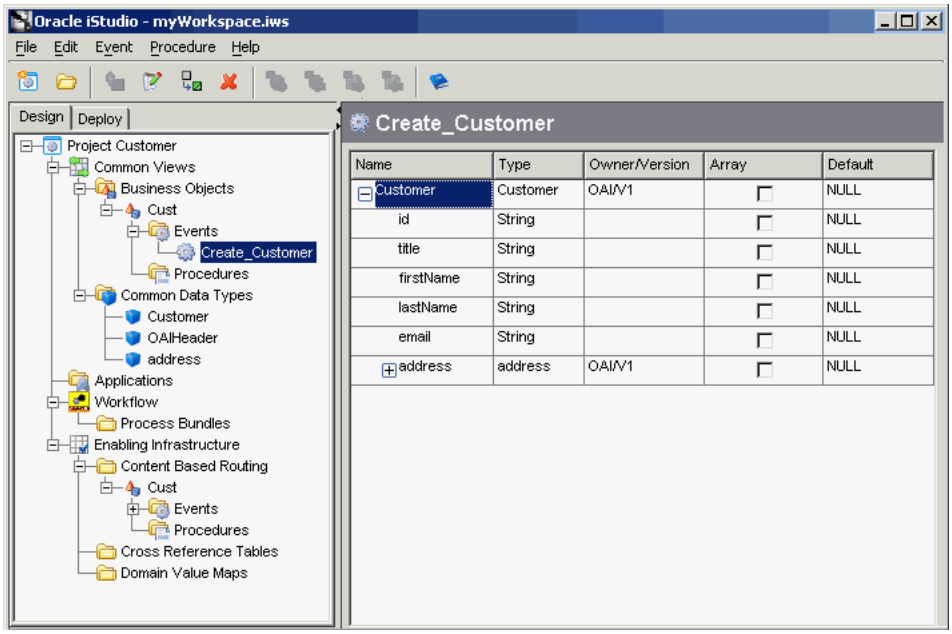
The main iStudio window has the following parts:

- Menu Bar
- Toolbar
- Design Navigation List
- Deploy Navigation List
- Context Menus

- Detail View

When iStudio is started, the main window is displayed as shown in Figure 2–1.

Figure 2–1 OracleAS Integration InterConnect iStudio



2.3.1 Menu Bar

The menu bar provides access to all commands. Click each menu to display its commands. Click a command to run it. There are five menus:

- File Menu
- Edit Menu
- Procedure Menu
- Event Menu
- Help Menu

2.3.1.1 File Menu

Use the File menu to create new projects and workspaces, open existing projects and workspaces, or reload existing projects. You can also create objects such as events, procedures, and common data types from the File menu. Commands include:

- New Project: Creates a new project.
- Open Project: Opens an existing project. In the Open dialog box, select the directory and project, then click **Open**.
- New Workspace: Creates a new workspace.
- Open Workspace: Opens an existing workspace. In the Open dialog box, select the directory and workspace, then click **Open**.
- Reload Project: Reloads a project. When Reload Project is selected, a list of current projects is displayed. Select the project to reload from the list.

- Migrate: Migrates objects from one repository to another. This functionality can be used to migrate metadata from a development to a production integration system.
- New: Creates a new object in iStudio. When New is selected, a list of available objects is displayed. Select the object to create.
- Sync Adapters: Pushes metadata to adapters. Adapters can be configured to cache metadata locally so that they avoid any run-time performance penalties around repository access. If they are so configured, any changes made in iStudio will need to be explicitly pushed to the corresponding adapters to refresh their local cached repository metadata. This functionality provides that explicit mechanism to refresh adapter-cached metadata.

  Note: Before you attempt to sync adapters, you must ensure that there are no messages flowing between the relevant adapters.

- Exit: Exits iStudio.

2.3.1.2 Edit Menu

Use the Edit menu to edit, copy, or delete selected objects. If an object is selected and the Edit menu is not available, then that selected object cannot be edited. Commands include:

- Edit: Edits a selected object. The type of editing depends on the object selected.
- Copy: Copies a selected object.
- Delete: Deletes a selected object.
- Rename: Renames a selected application.
- Version: Creates a new version of or loads a selected object.
- Domain Value Map: Adds or removes applications from a domain value map.
- Cross Reference Table: Adds or removes applications from a cross reference table.
- Workflow: Deploys events to Oracle Workflow or edits Oracle Workflow configuration information.

2.3.1.3 Procedure Menu

Use the Procedure menu to invoke or implement procedures. Commands include:

- Invoke: Invokes a selected procedure by launching the Invoke Wizard.
- Implement: Implements a selected procedure by launching the Implement Wizard.

2.3.1.4 Event Menu

Use the Event menu to publish or subscribe to events. Commands include:

- Publish: Publishes a selected event by launching the Publish Wizard. An event must be created.
- Subscribe: Subscribes to a selected event by launching the Subscribe Wizard. An event must be created.

2.3.1.5 Help Menu

The Help menu provides links to online help. Commands include:

- Contents: Opens the User's Guide.
- About: Displays version information for iStudio.

2.3.2 Toolbar

The toolbar consists of icons that represent frequently used commands. To display a caption describing the icon, place the cursor on the icon. The following functions are provided:

Function	Icon	Description
New Project		Creates a new project in iStudio.
Open Project		Opens an existing project in iStudio.
Create Integration Object		Creates a new integration object.
Create Like		Creates a new integration object similar to an existing object. This icon is enabled only when an object is selected in the Navigator.
Edit Integration Object		Edits a selected integration object.
Delete Integration Object		Deletes a selected integration object. This icon is enabled only when an integration object is selected in the Navigator.
Publish Event		Publishes a selected event. This icon is enabled only when an integration object is selected in the Navigator.
Subscribe Event		Subscribes to a selected event. This icon is enabled only when an integration object is selected in the Navigator.
Invoke Procedure		Invokes a selected procedure. This icon is enabled only when an integration object is selected in the Navigator.
Implement Procedure		Implements a selected procedure. This icon is enabled only when an integration object is selected in the Navigator.

Function	Icon	Description
Help		Displays the help file.

2.3.3 Design Navigation List

The Design Navigation list displays the hierarchical layout of all objects used in the design phase of an opened project. Each object type in the Deploy Navigation list is identified by an icon and a name. A container is represented by a folder icon and is a logical grouping of a specific type of object, such as a business object and an application data type.

The objects are grouped as follows:

- Common Views
- Applications
- Workflow
- Enabling Infrastructure

2.3.4 Deploy Navigation List

The Deploy Navigation list displays the hierarchical structure of all objects used in the deployment phase of an opened project. Each object type in the Deploy Navigation list is identified by an icon and a name. A container is represented by a folder icon and is a logical grouping of a specific type of object, such as process bundles.

The objects are grouped as follows:

- Applications
- Workflow

2.3.5 Context Menus

You can right-click an object to display a context menu, that is, a shortcut menu related to the object.

Navigation List	Selected Item	Context Menu Options
Design	Object, such as Common View, Application, Business Objects, and Common Data Types	New, Edit, Copy, Delete
	Container object, such as an existing event or procedure	New, Edit, Copy, Delete, Load Version, New Version
	Workflow object	New, Edit, Copy, Delete, Launch WF Builder, Launch WF Home Page
Deploy	Object such an Application	New, Edit, Copy, Delete, Deploy, Export PL/SQL
	Workflow object	New, Edit, Copy, Delete, Deploy, Edit Configuration, Launch WF Home Page, Export
	Container object, such as an existing routing object	New, Edit, Copy, Delete, Create Partition

2.3.6 Detail View

To the right of the Navigation list is the Detail View, composed of one or more property sheets displaying information about the object selected. Often, these property sheets may be edited.

2.4 Using Workspaces in iStudio

A workspace stores user settings and preferences, such as application login credentials and information about the project last opened. Inside a workspace, users can work on multiple projects.

2.4.1 Creating a New Workspace

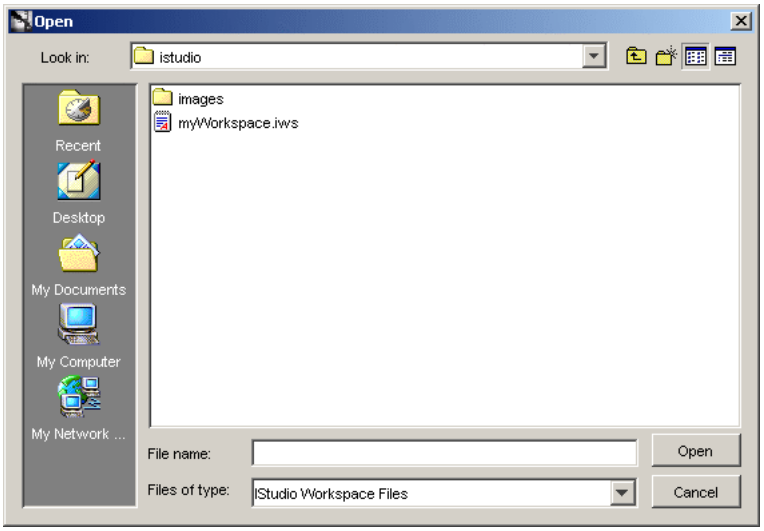
To create a new workspace:

1. From the File menu, select **New Workspace**. The New Workspace dialog box is displayed.
2. Enter a name for the workspace in the **Workspace Name** field.
3. Click **OK**.

2.4.2 Opening an Existing Workspace

To open an existing workspace:

1. From the File menu, select **Open Workspace**. The Open dialog box is displayed.
2. Enter the name and path to an existing workspace, or select the workspace to open.



3. Click **Open**. The selected workspace is displayed in iStudio.

2.5 Using Projects in iStudio

A project in iStudio captures all the integration logic for one integration scenario. An integration scenario is defined as a set of two or more applications integrated with each other using OracleAS Integration InterConnect. One project corresponds to one repository. For example, a user may have a development integration environment and

a production integration environment. These are two separate projects and must be self-contained in their own separate repositories.

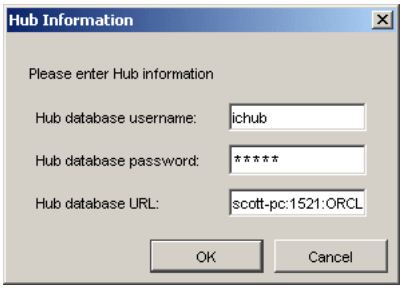
As iStudio is a multiuser tool, multiple users can work on the same project, simultaneously, without jeopardizing the integrity of the metadata.

Note: To create a project in iStudio, the repository must be running.

2.5.1 Creating a New Project

The repository must be running in order to create a project in iStudio. To create a new project in iStudio:

1. From the File menu, select **New Project**. The New Project dialog box is displayed.
2. Enter the project name and click **OK**. The Hub Information dialog box is displayed.



3. Enter information in the following fields:
   - **Hub database username**: The name of the hub database user. The default user name is `ichub`.
   - **Hub database password**: The password associated with the hub database user. The default password is set when OracleAS Integration InterConnect is installed.
   - **Hub database URL**: Information of the following format:

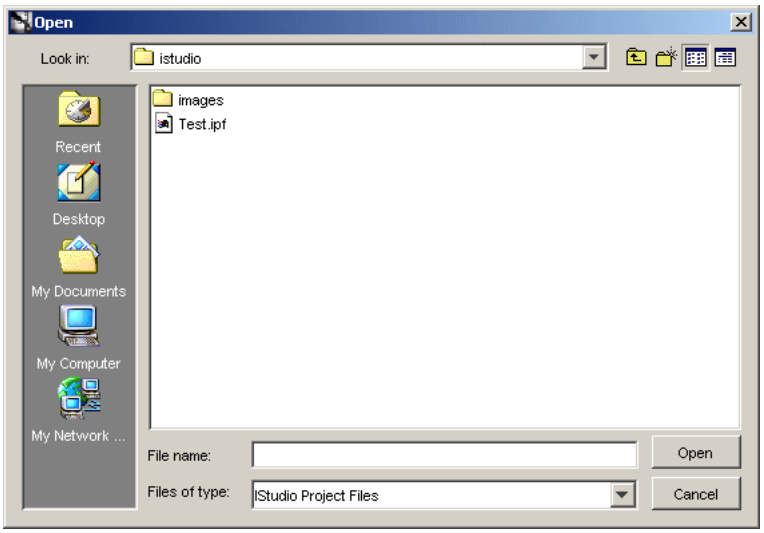
     ```
     machine name:port number:database sid
     ```

4. Click **OK**.

2.5.2 Opening an Existing Project

To open an existing project:

1. From the File menu, select **Open Project**. The Open dialog box is displayed.



1. Enter the name and path to an existing project or select the workspace to open.
2. Click **Open**. The selected project opens in iStudio.

3

Creating Applications, Common Views, and Business Objects

This chapter describes how to create and manage applications, common views, and business objects using iStudio. It contains the following topics:

- Overview of Applications
- Overview of Common Views and Business Objects

3.1 Overview of Applications

Each component integrated with OracleAS Integration InterConnect is referred to as an application. Each application expresses interest in specific messages, what its internal data type is, and how the message should be mapped to or from that internal type to the external world.

3.1.1 Application View

Each application has its internal data types, formats, and structures that it exposes to the external world. This is the application's public interface. This application interface is called the application view of data. Transformations are used to bridge the gap between application views and common views of data.

Once an application is created in iStudio, it can start participating in the integration scenario by plugging into events and procedures available in the common view.

> **See Also:** Chapter 4, "Using Events in iStudio" and Chapter 5, "Using Procedures in iStudio"

3.1.2 Application Data Types

Application Data Types are useful for reusing structure definitions for application views. You can define an application data type once and then use it in multiple application views. They are especially useful for defining complex hierarchical data. Application data types have the same function as common data types but relate to a particular application.

> **See Also:** Section 3.2.1.2, "Creating Common Data Types"

3.1.3 Creating an Application

To create an application:

1. From the File menu, select **New**, then select **Application**. The Create Application dialog box is displayed.
2. Enter a name for the application in the **Application Name** field.
3. Click **OK**.

   The application created is displayed in the Design Navigation list under the Applications node.

3.2 Overview of Common Views and Business Objects

The common view is the *hub view* of the integration, where each spoke is an application wanting to participate in the integration. After defining a common view by creating a business object and common data types, existing events can be published or subscribed to, and procedures can be invoked or implemented.

See Also: Chapter 2, "Using iStudio"

3.2.1 Defining Common Views

When defining a common view, you must create business objects and common data types.

3.2.1.1 Creating Business Objects

To create a new business object:

1. From the **File** menu, select **New**, then select **Business Object**. The Create Business Object dialog box is displayed.
2. Enter a name for the business object in the **Business Object Name** field.
3. Click **OK**. The business object is displayed in the Design Navigation list under the Common View node.

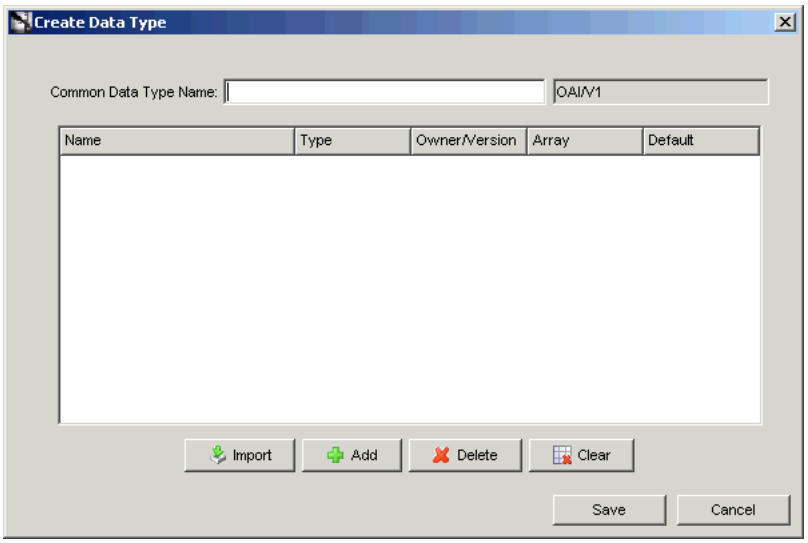
3.2.1.2 Creating Common Data Types

When creating the data associated with an event or a procedure, it is possible to define the data once and reuse it for different integration points. Common data types are used to define such data for reuse and are especially useful for defining complex hierarchical data.

For example, a purchase order contains a header object and an array of line item objects. In addition, the header object contains two address objects: `Bill_To` and `Ship_To`. As a result, the purchase order can be defined once and used for other purchase order-related integration points, such as `Create_Purchase_Order`, `Update_Purchase_Order`, and `Get_Purchase_Order`. Moreover, `Address` can be defined once and used in the `Bill_To` and `Ship_To` addresses.

To create a common data type:

1. From the **File** menu, select **New**, then select **Common Data Type**. The Create Data Type dialog box is displayed.



Enter a name for the common data type in the **Common Data Type Name** field.

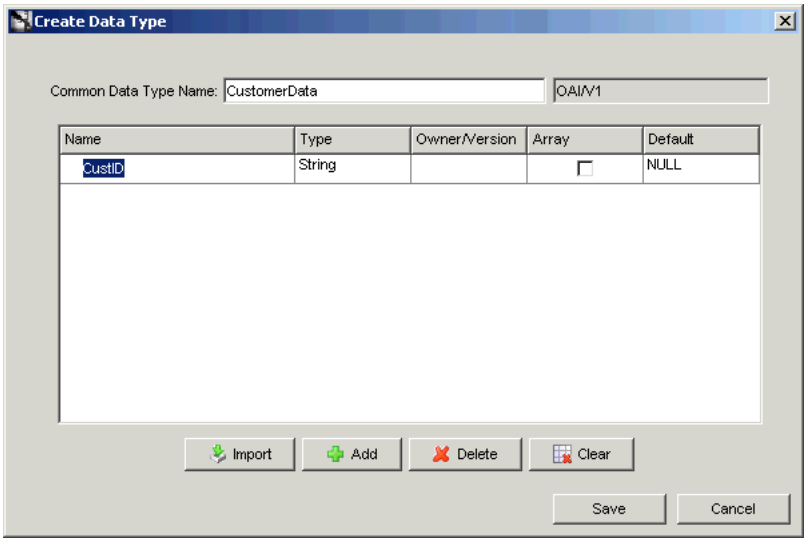
The owner and version number of the common data type are displayed next to the common data type name. This field cannot be edited.

2. Specify the attributes for this common data type using one of the following methods:
   - Add attributes individually.
   - Import attributes from already existing application data types or APIs.

3.2.1.3 Adding Attributes

To add attributes:

1. In the Create Data Type dialog box, click **Add**. A new entry is displayed in the attribute list.



2. Specify the following information by editing the fields.

   Name: The name of the attribute.

Type: The type of the attribute. Select the type by selecting the Type column in the attribute entry. A list is displayed. The attribute can be of primitive type such as string, integer, float, double, date, or another common data type used to build hierarchical data types.

Array: Select this check box if the attribute is a collection instead of a single attribute.

Default: The default value of the field if it is not populated at run time.

3. Click **Save**. Repeat the preceding steps to add other attributes.

3.2.1.4 Importing Attributes

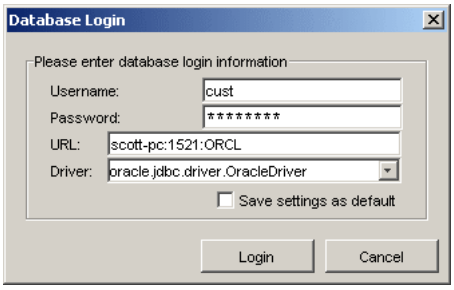
To import attributes:

1. In the Create Data Type dialog box, click **Import**. Attributes can be imported from various sources.

   The following steps describe the database import facility.

   See Also: Appendix C, "Using the Data Definition Description Language"

2. Click **Database**. The Database Login dialog box is displayed.



3. Enter information in the following fields:

   User Name: The database login name

   Password: The database login password

   URL: The computer name: port number: database `SID`

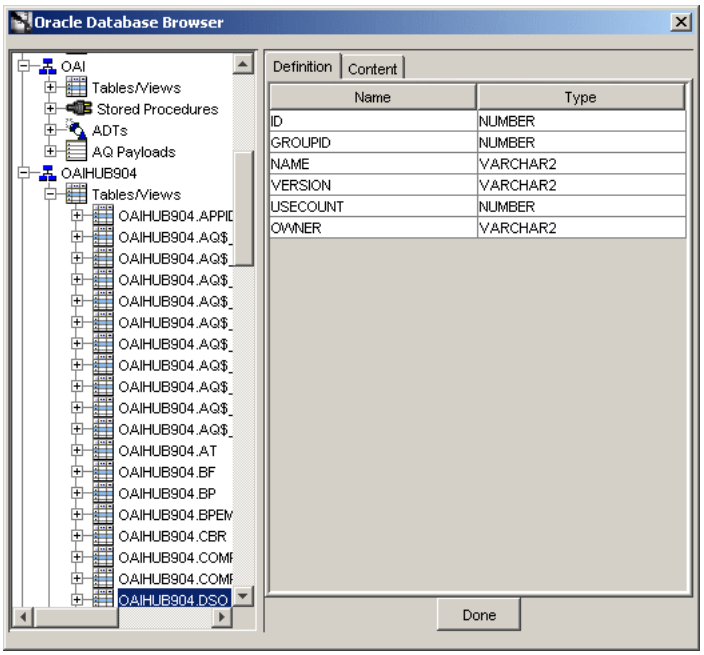
   Driver: The JDBC driver used to connect to the database

   Save settings as default: Select this check box to save the settings for the workspace.

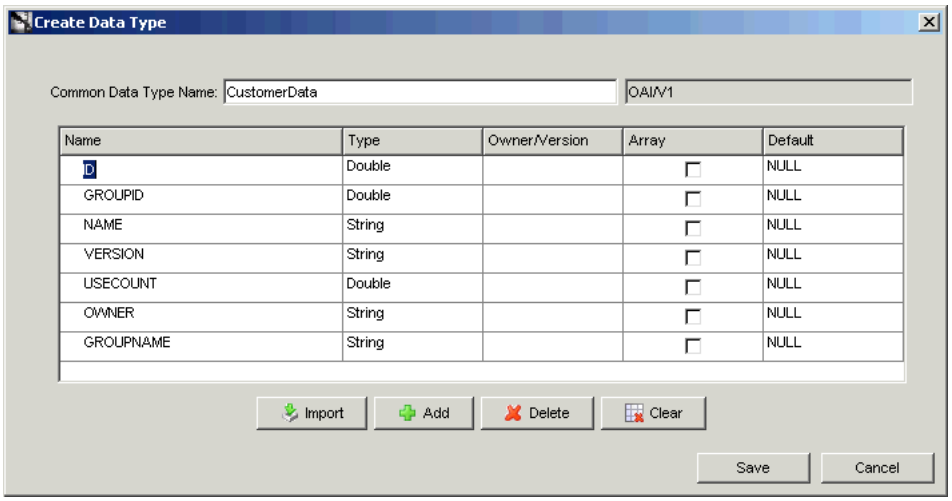
4. Click **Login**.

   After logging in, the database tables and arguments are displayed in the Database Browser window.

   Select the fields to add. To select a range of fields, press **Shift** when clicking the mouse button. To select multiple items, press **Ctrl** while clicking the mouse button.



5. Click **Done** to import the attributes into the common data type. The selected attributes are displayed in the Create Data Type dialog box.



3.2.1.5 Deleting and Clearing Attributes

To delete a selected attribute:

- In the Create Data Type dialog box, select the attribute to be deleted, and click **Delete**.

To clear all attributes:

- In the Create Data Type dialog box, click **Clear**.

4

Using Events in iStudio

This chapter describes how to use iStudio to create, publish, and subscribe to events. It contains the following topics:

4.1 Overview of Events

An event is an integration point used to model the publish/subscribe paradigm. An event has associated data that is the common view of all the data to be exchanged through this event.

The data structure that should be used for defining the common view is entirely dependent on the integration scenario. The only condition that the data structure should satisfy is that it should be a superset of all the application views for applications that will publish or subscribe to this event. The choices for common view data include:

- Corporate internal standards enforced by the organization where OracleAS Integration InterConnect is being implemented.
- Industry standard definitions, such as Open Applications Group (OAG) and Business Object Definitions (BOD).
- One of the application views can be used as the common view, if the scenario has one application that is driving the integration.

For example, `App1` and `App2` publish customer names, and `App3` subscribes to it. If `App1` publishes `First Name`, `Last Name`, and `Middle Initial`, and `App2` publishes `First Name`, `Last Name`, `Prefix`, and `Suffix`, then the event could be defined as follows:

```
New Customer Event
Prefix
First Name
Last Name
Middle Initial
Suffix
```

See Also: "How the Hub-and-Spoke Model Works" on page 1-6

4.1.1 Event Maps

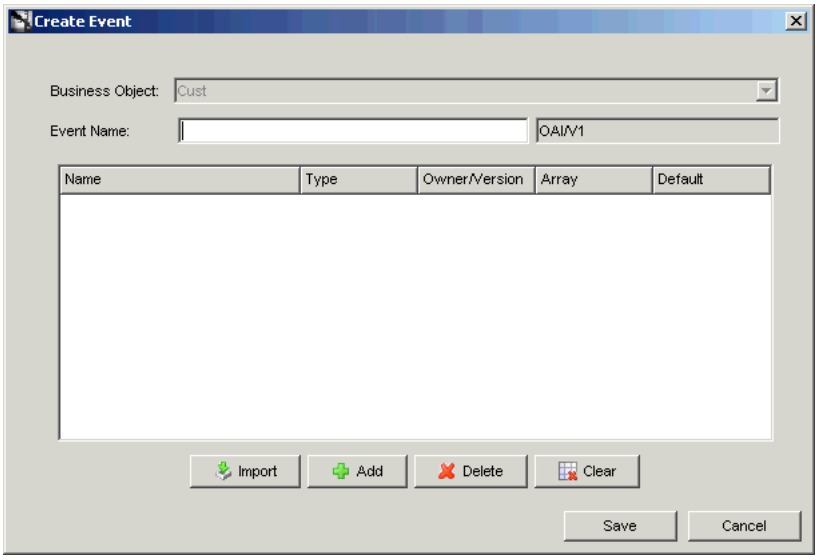
If an application publishes exactly the same data structure for two or more events, then event maps allow OracleAS Integration InterConnect to identify which message corresponds to which event. For example, an application publishes the same Customer Application data type whether or not it is a `Create_Customer` event or an `Update_Customer` event. Through event maps, OracleAS Integration InterConnect can determine which messages correspond to `Create_Customer` and `Update_Customer`.

Note: Event maps need to be used only if two or more events published by a particular application have the same application view structure.

4.2 Creating Events

To create an event:

1. From the **File** menu, click **New**, then select **Event**. The Create Event dialog box is displayed.



2. Enter the information in the following fields:
   - **Business Object**: The name of the category to which the event belongs. Select a category from the list.
   - **Event Name**: The name of the event. Only alphanumeric characters can be used.
   - **OAI/V1**: The owner and version number of the business object. This field cannot be edited.
3. Add or import attributes to this event.
4. Click **Save**.

See Also:

- "Adding Attributes" on page 3-3
- Appendix C, "Using the Data Definition Description Language"
- "Deleting and Clearing Attributes" on page 3-5

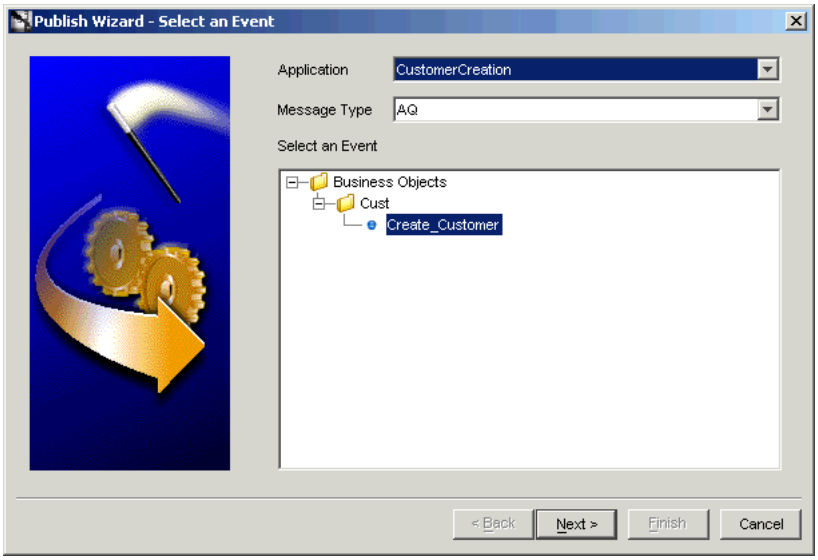
4.3 Publishing and Subscribing to an Event

The publish/subscribe paradigm is used for asynchronous one-way communication. The sending application is said to publish the event. The receiving application subscribes to the event.

4.3.1 Publishing an Event

Publishing an event in an application in iStudio involves the Publish Wizard. To start the Publish Wizard:

1. In the Design Navigation list, expand the **Application** node to display the Published Events node.
2. Right-click **Published Events** and select **Publish**. The Publish Wizard is displayed.



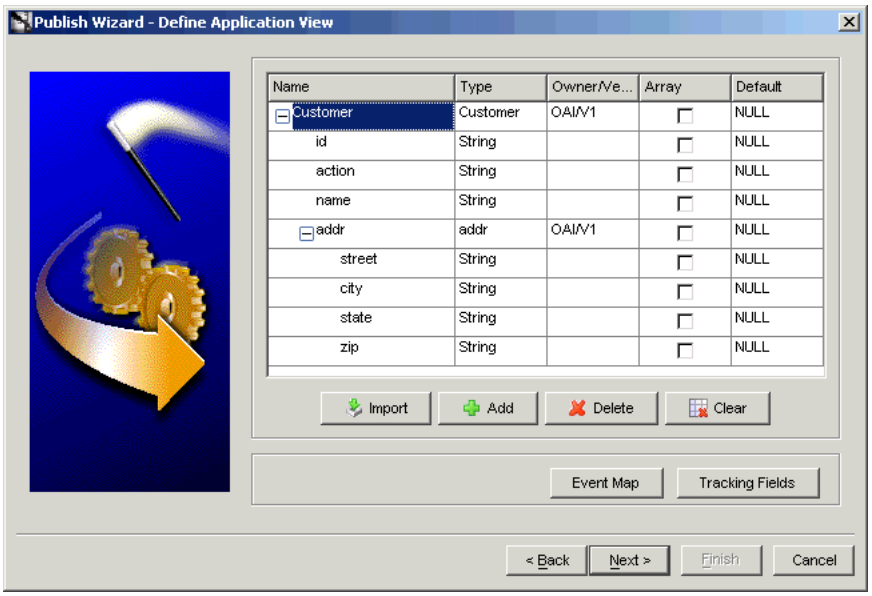
3. Specify the information for the following fields:
   - Application: The name of the application is selected by default.
   - Message Type: The mode of communication between OracleAS Integration InterConnect and the application. Select from the following message types:
     - Database: OracleAS Integration InterConnect communicates with the application using the database.
     - Generic: OracleAS Integration InterConnect communicates with the application using a user-defined bridge.
     - XML: OracleAS Integration InterConnect communicates with the application using XML data described through a data type definition (DTD) or XML Schema Definition (XSD) by using the FTP, SMTP, HTTP, and MQSeries adapters.

- AQ: OracleAS Integration InterConnect communicates with the application through Oracle Advanced Queues using the Advanced Queue adapter. The payload can be Oracle Objects, where fields may be XML or RAW XML.
- D3L: OracleAS Integration InterConnect communicates with the application using non-XML data formats described through D3L by using the FTP, SMTP, HTTP, and MQSeries adapters.
- JCA: OracleAS Integration InterConnect communicates with the application using the J2CA adapter.

Note: Other choices will be visible if you have purchased and installed additional adapters for Oracle e-business Suite, SAP, Peoplesoft, and Siebel.

- Select an Event: Select the event name.

4. Click **Next**. The Define Application View dialog box is displayed.



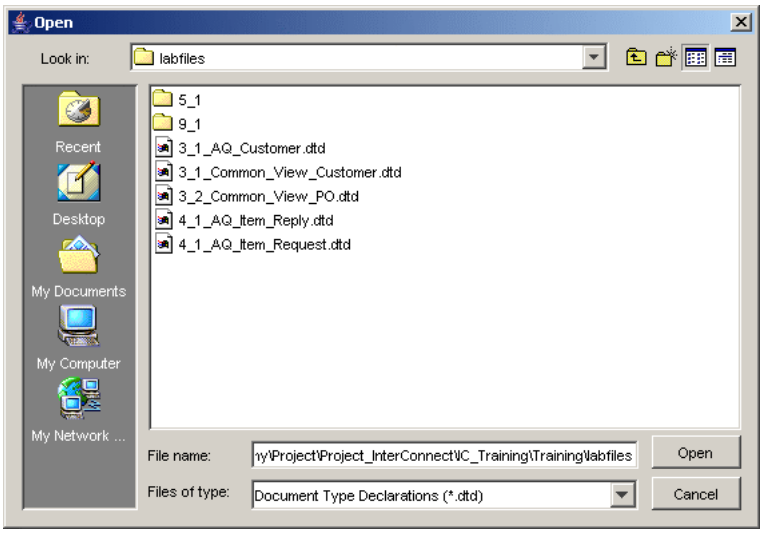
5. Once an event is selected to publish, the application view is defined. The application view page is initially an empty table. Define the attributes by using Add, or import the definitions from a database or an API Repository by using Import.

   a. Click **Add** or **Import** to add or import attributes.

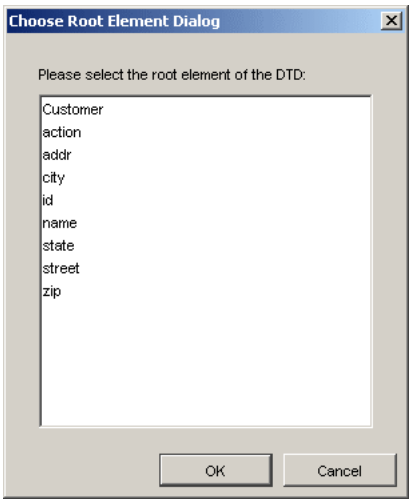
   See Also:

   - "Adding Attributes" on page 3-3
   - "Importing Attributes" on page 3-4
   - "Deleting and Clearing Attributes" on page 3-5
   - Appendix C, "Using the Data Definition Description Language"

   b. To import an XML DTD, click **Import** and select **XML**.



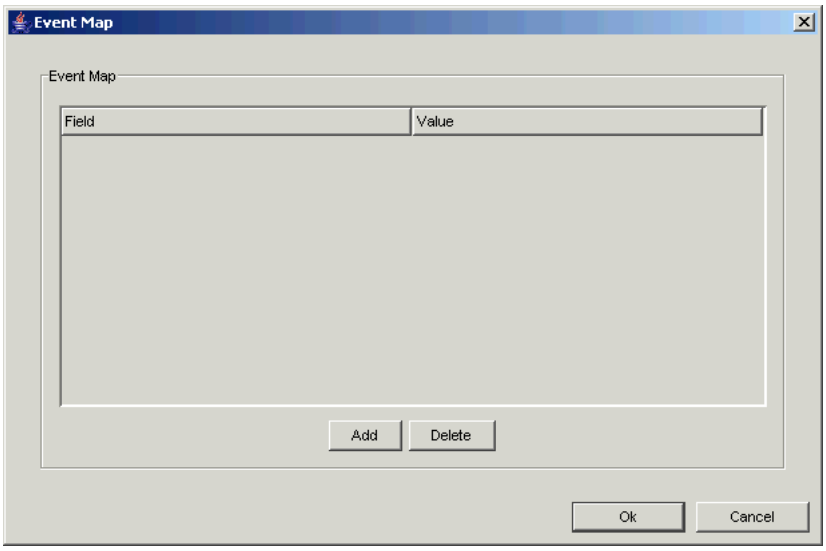
c. Select a DTD file and click **Open**. The Choose Root Element dialog box is displayed.



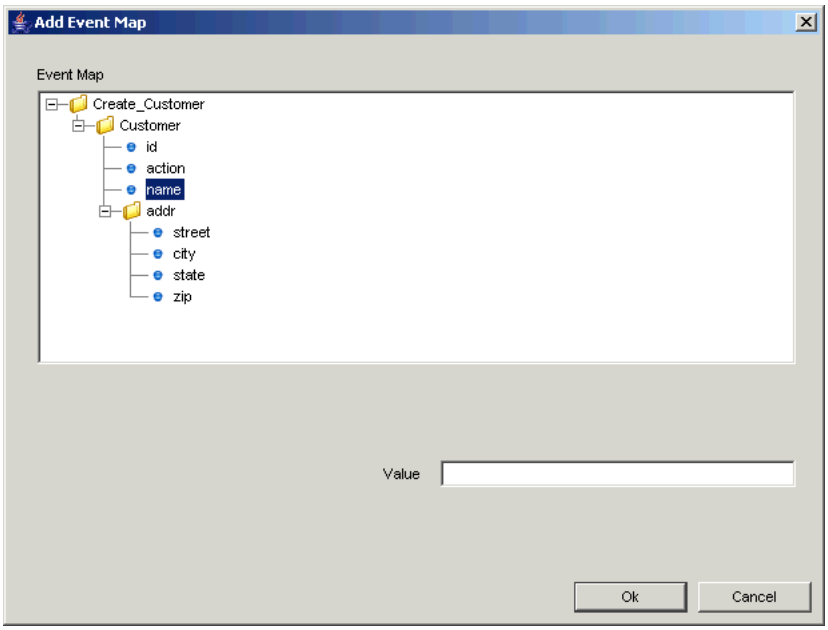
d. Select a root DTD element and click **OK**.

e. If this is an XML-type message, then the Event Map button is enabled. To define the event map, click **Event Map**. The Event Map dialog box is displayed.

Note: Event maps must be used *only* if two or more events published by a particular application have the same application view structure.



f. Click **Add** to add an event map attribute. The New Event Map dialog box is displayed.



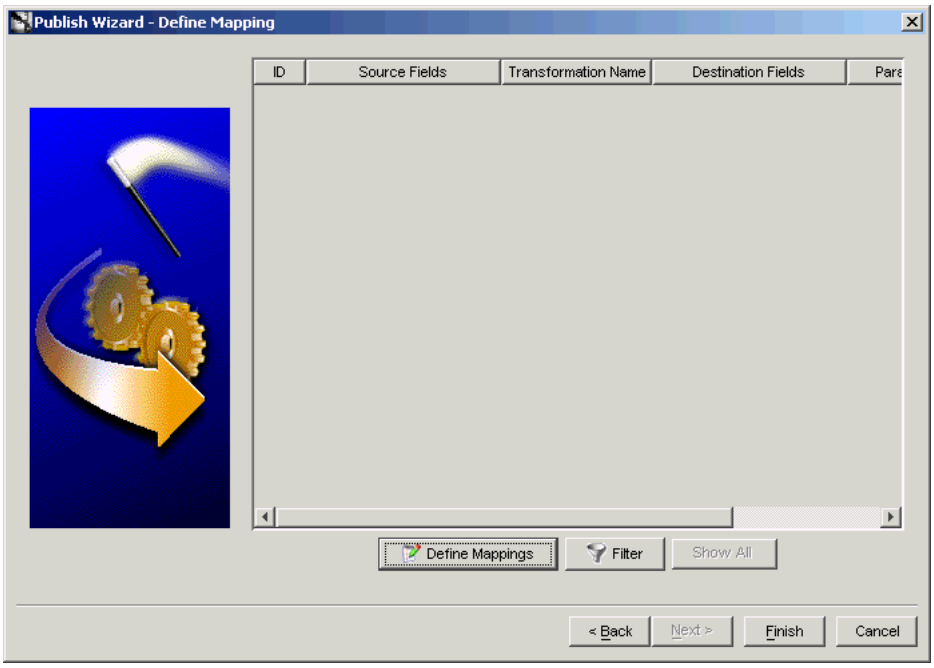
g. Expand the list, select an attribute, and enter a value in the **Value** field.

h. Click **OK** on the Add Event Map dialog box to return to the Event Map dialog box.

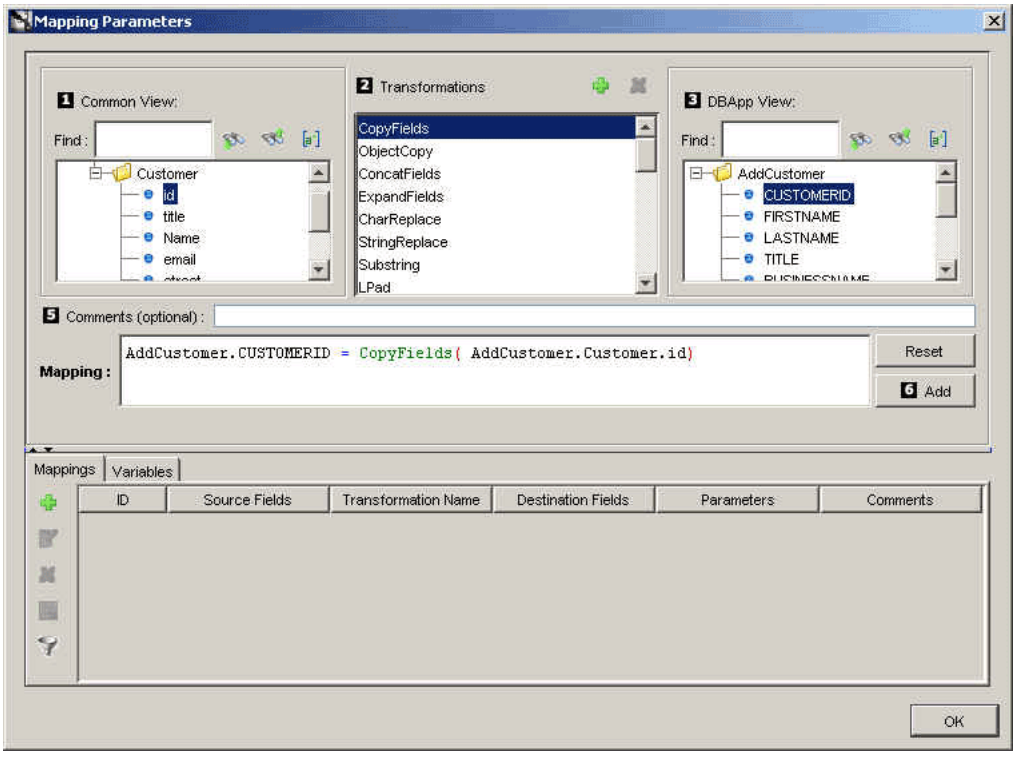
i. Click **OK** to return to the Publish Event Wizard.

j. Click **Next**.

6. Click **Next** on the Define Application View dialog box. The Define Mapping dialog box is displayed. Mapping involves copying the individual fields or simple shape-change transformations.



a. Click **DefineMappings** to define new mappings. The Mapping Parameters dialog box is displayed.



Use a transformation to map fields in the application view to fields in the common view. For example, to map the `FirstName` and `LastName` fields in the common view to the `Name` field in the application view, use the `ConcatFields` transformation.

See Also: "Invoking a Procedure" on page 5-3

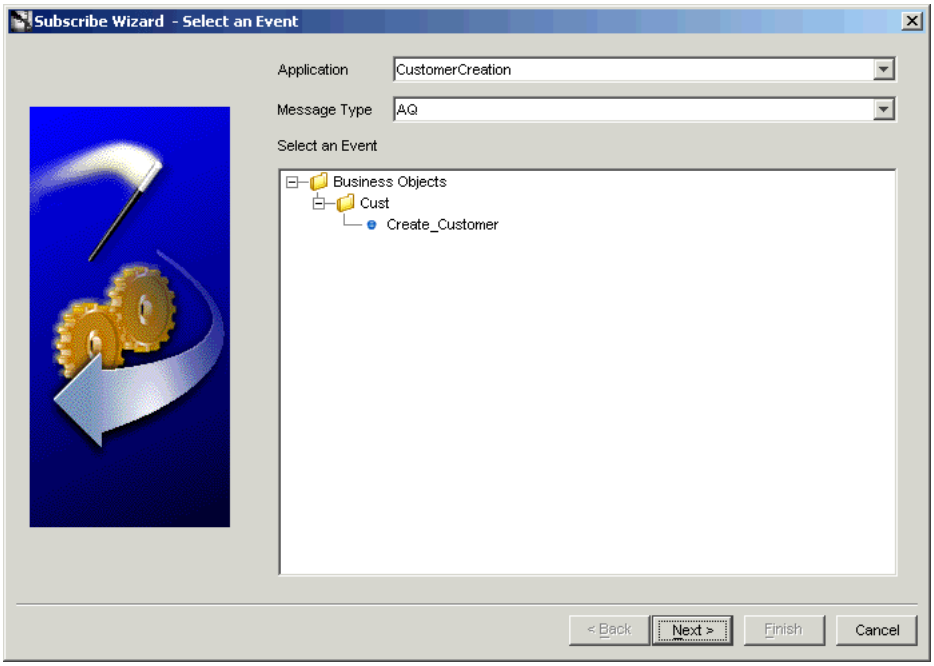
b. Click **OK** to return to the Publish Event Wizard.

7. Click **Finish**.

4.3.2 Subscribing to an Event

Subscribing to an application event in iStudio involves using the Subscribe Wizard. To subscribe to an event in an application:

1. In the Design Navigation list, expand the **Application** node.
2. Right-click **Subscribed Events** and select **New**. The Subscribe Wizard is displayed.



a. Specify the information for the following fields:

Application: The name of the application selected in the navigation list, which invokes the procedure, appears selected by default.

Message Type: The message type that specifies the mode of communication between OracleAS Integration InterConnect and the application. Select from the following message types:

- Database: OracleAS Integration InterConnect communicates with the application using the database.
- Generic: OracleAS Integration InterConnect communicates with the application using a user-defined bridge.
- XML: OracleAS Integration InterConnect communicates with the application using XML data described through a DTD or XSD by using the FTP, SMTP, HTTP, and MQSeries adapters.
- AQ: OracleAS Integration InterConnect communicates with the application through Oracle Advanced Queues using the Advanced Queue adapter. The payload can be Oracle Objects where fields may be XML or RAW XML.

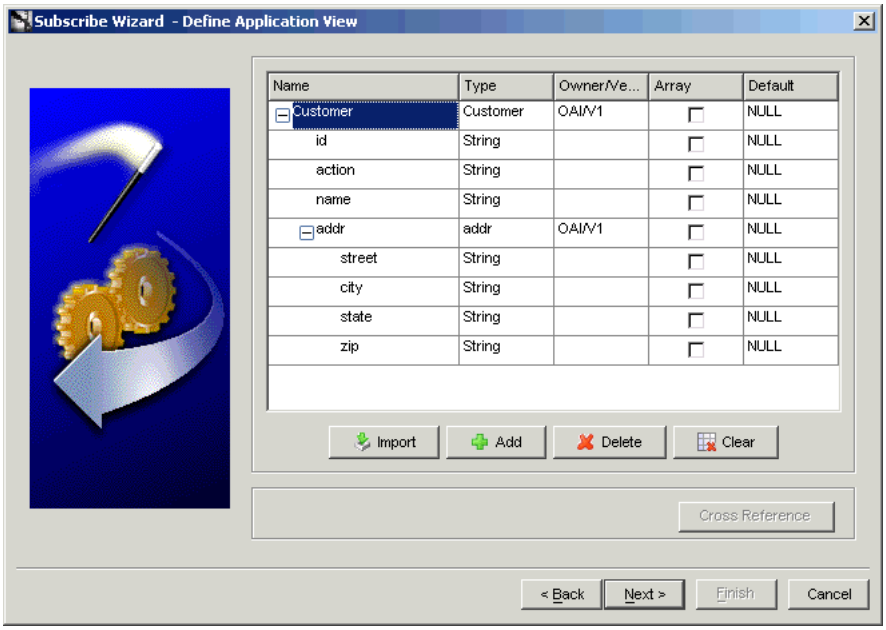
- D3L: OracleAS Integration InterConnect communicates with the application using non-XML data formats described through D3L using the FTP, SMTP, HTTP, and MQSeries adapters.
- JCA: OracleAS Integration InterConnect communicates with the application using the J2CA adapter.
- Oracle Applications: OracleAS Integration InterConnect communicates with Oracle Applications using this message type.

Note: Other choices will be visible if you have purchased and installed additional adapters for SAP, Peoplesoft, and Siebel.

---

b. Select the event name.

c. Click **Next**. The Define Application View page is displayed.



3. Once an event is selected to subscribe to, the application view is defined. The application view page is initially an empty table. Define the attributes by using Add, or import the definitions from a database or an API Repository by using Import.

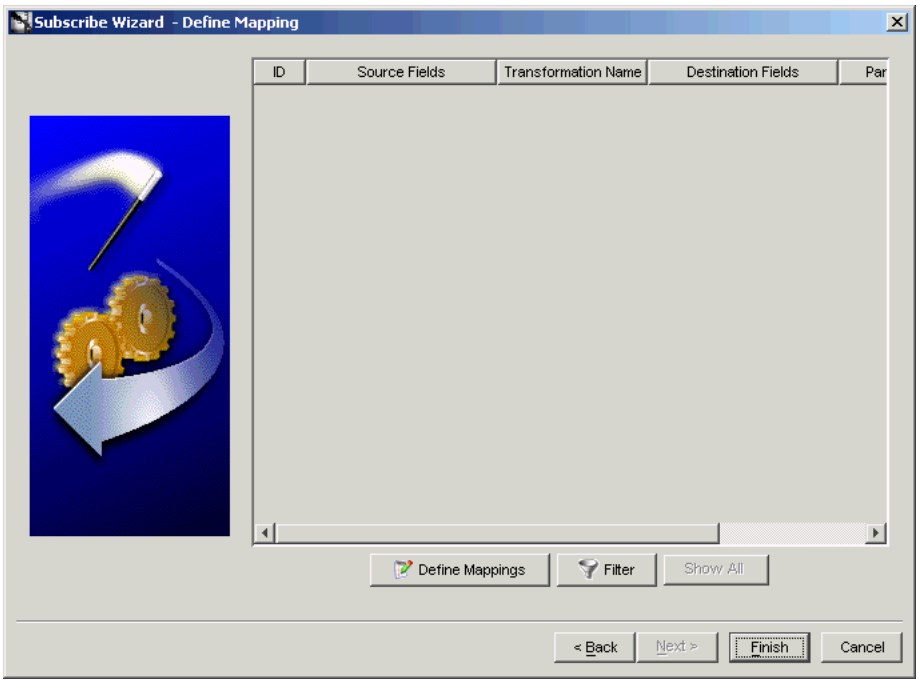
a. Click **Add** or **Import** to add or import attributes.

See Also:

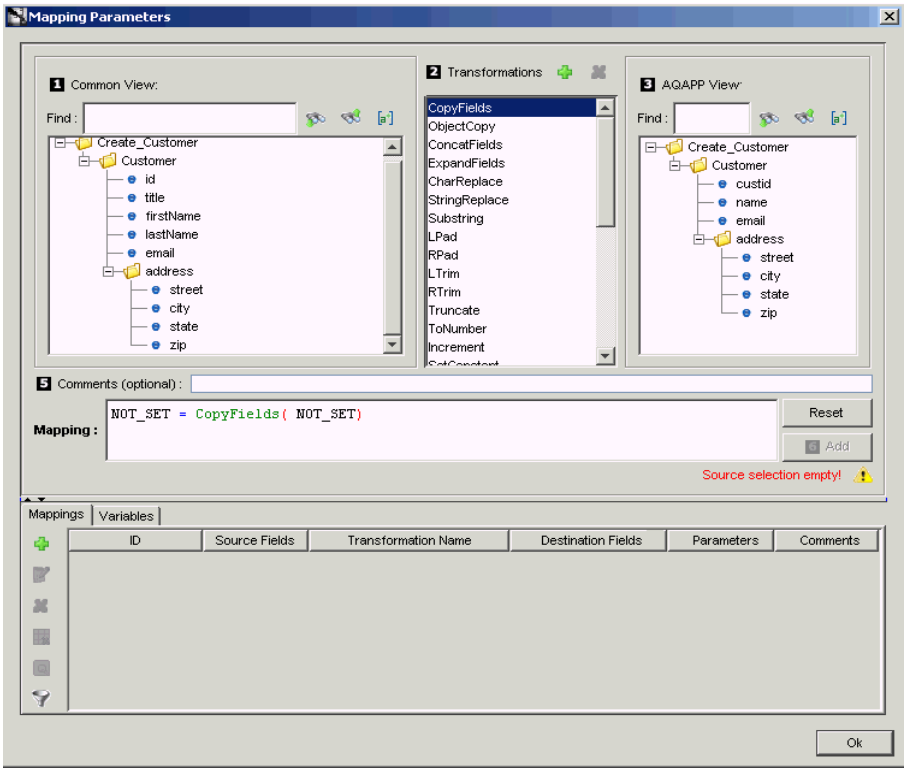
- "Adding Attributes" on page 3-3
- "Importing Attributes" on page 3-4
- "Deleting and Clearing Attributes" on page 3-5
- Appendix C, "Using the Data Definition Description Language"

b. Populate and look up cross-reference tables by clicking **Cross Reference**. The Cross Reference dialog box is displayed. Click **OK** to return to the Subscribe Wizard.

4. Click **Next** on the Define Application View page. The Define Mapping dialog box is displayed. Mapping can involve either copying the individual fields or simple shape-change transformations.



   a. Click **Define Mappings** to define mappings. The Mapping Parameters dialog box is displayed.



   Use a transformation to map fields in the common view to fields in the application view. For example, to map the `FirstName` and `LastName` fields

in the common view to the `Name` field in the application view, use the `ConcatFields` transformation.

See Also: "Invoking a Procedure" on page 5-3

b. Click **OK** to return to the Subscribe Event Wizard.

5. Click **Finish**.

5

Using Procedures in iStudio

This chapter describes using iStudio to create, invoke, and implement procedures. It contains the following topics:

5.1 Using Procedures

A procedure is an integration point used to model the request/reply paradigm. The request/reply paradigm is used for two-way context-sensitive communication. This communication can be either synchronous or asynchronous. In synchronous communication, the requesting application is blocked until it receives a reply. In asynchronous communication, the requesting application gets the reply asynchronously, it does not block-wait for the response after sending the request. This is a only modeling paradigm. No actual procedures are called. An application can either invoke a procedure to model sending a request and receiving a reply, or implement a procedure to model receiving a request and sending a reply. Similar to events, procedures have associated data. While an event is only associated with one data set, a procedure has two data sets, one for the request, `IN` data, and one for the reply, `OUT` data.

> **Note:** Synchronous request/reply can be used only if an application supports an outbound synchronous interface. Currently, only the OracleAS Integration InterConnect Database adapter qualifies for such support. For all other adapters, only asynchronous request/reply is available. This is a limitation of the protocols exposed by systems to communicate.

For example, if a `Get_Address` procedure is defined so that the request contains the social security number, `SSN`, for a person and the reply contains the address in four fields: `Street`, `City`, `Zip`, and `State`, then the procedure is defined as follows:

```
get Address Procedure
SSN IN
Street OUT
City OUT
Zip OUT
State OUT
```

The data structure that should be used for defining the common view is entirely dependent on the integration scenario, and the choice is left to the implementer. The

only condition that the data structure should satisfy is that it should be a superset of all the application views for applications that will be publishing or subscribing to this event. The choices for common view data include:

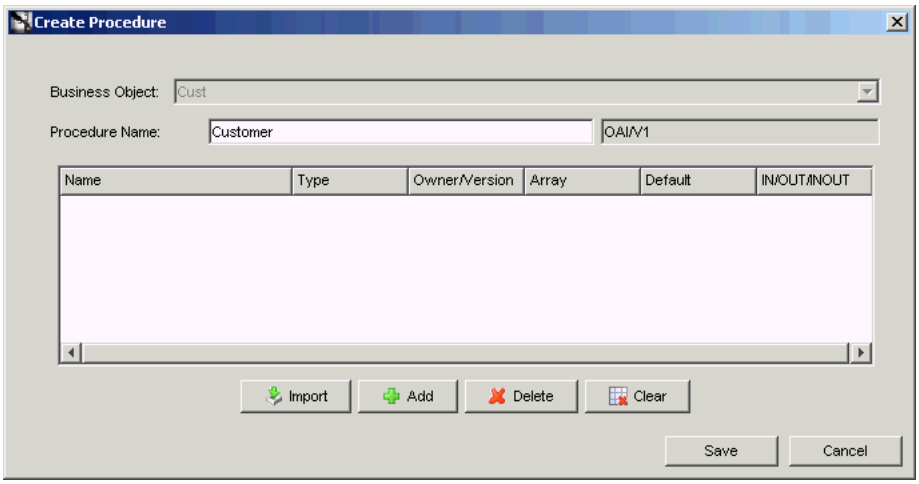
- Corporate internal standards enforced by the organization where OracleAS Integration InterConnect is being implemented.
- Industry standard definitions, such as Open Applications Group (OAG) and Business Object Definitions (BOD).
- One of the application views can be used as the common view, if the scenario has one application that is driving the integration.

See Also: "How the Hub-and-Spoke Model Works" on page 1-6

5.1.1 Creating a Procedure

To create a procedure:

1. From the File menu, select **New**, and then select **Procedure**. The Create Procedure dialog box is displayed.



2. Enter the required information in the following fields:
   - Business Object Name: The name of the category to which the procedure belongs. Select BO name from the list.
   - Procedure Name: The name of the procedure. Only alphanumeric characters can be used.
   - OAI/V1: The owner and version number of the procedure. This field cannot be edited.
3. Add or import attributes to the procedure.
4. Click **Save**.

See Also:

- "Adding Attributes" on page 3-3
- "Importing Attributes" on page 3-4
- "Deleting and Clearing Attributes" on page 3-5
- Appendix C, "Using the Data Definition Description Language"

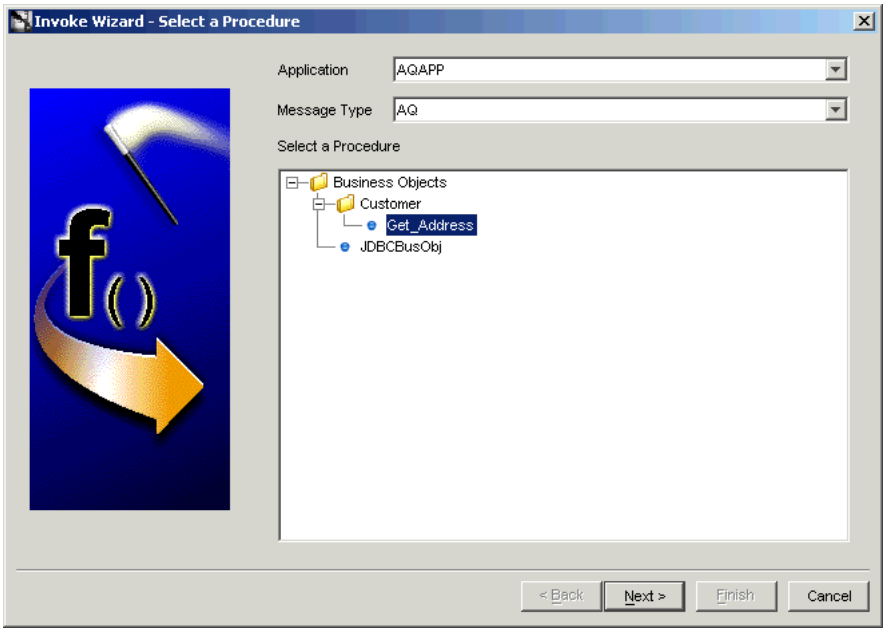
5.2 Invoking and Implementing a Procedure

Procedures are used to model the request/reply messaging paradigm. The requesting application invokes the procedure. The replying application implements the procedure.

5.2.1 Invoking a Procedure

Use the Invoke Wizard to invoke a procedure in iStudio. To start the Invoke Wizard:

1. In the Design Navigation list, select expand the **Application** node.
2. Right-click **Invoke Procedures**, and select **New**. The Invoke Wizard is displayed.



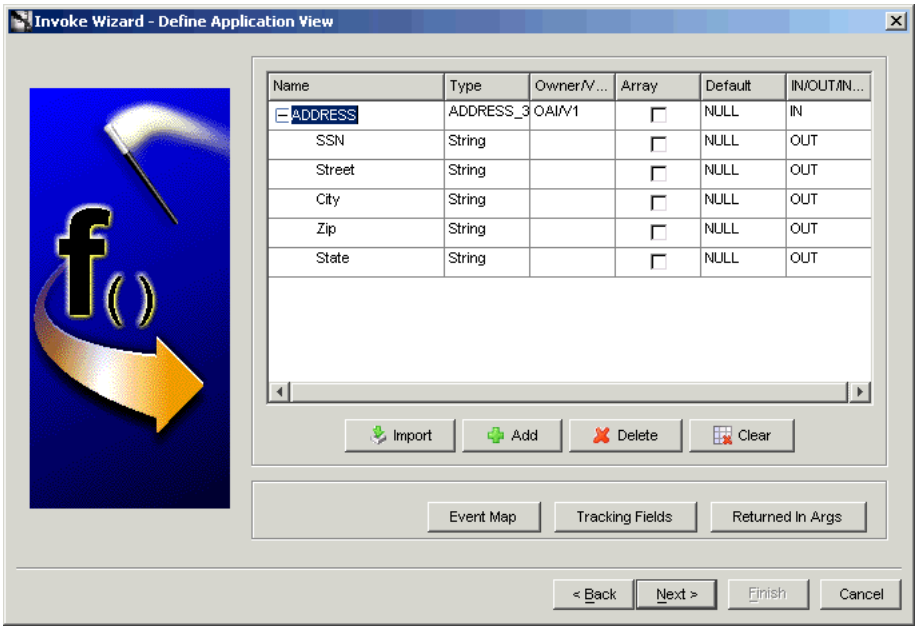
3. Specify the information for the following fields:
   - Application: The name of the source application is selected by default.
   - Message Type: The mode of communication between OracleAS Integration InterConnect and the application. Select one of the following message types:
     - Database: OracleAS Integration InterConnect communicates with the application using the database.
     - Generic: OracleAS Integration InterConnect communicates with the application using a user-defined bridge.
     - XML: OracleAS Integration InterConnect communicates with the application using XML data described through a data type definition (DTD) or XML Schema Definitions (XSD) by using the FTP, SMTP, HTTP, and MQSeries adapters.
     - D3L: OracleAS Integration InterConnect communicates with the application using non-XML data formats described through D3L using the FTP, SMTP, HTTP, and MQSeries adapters.
     - AQ: OracleAS Integration InterConnect communicates with the application through Oracle Advanced Queues by using the Advanced Queue adapter. The payload can be RAW XML or Oracle Objects where fields may be XML.

– JCA: OracleAS Integration InterConnect communicates with the application using the J2CA adapter.

Note: Other choices will be visible if you have purchased and installed additional adapters for Oracle Applications, SAP, Peoplesoft, and Siebel.

---

4. Select the procedure to invoke in the Select a Procedure dialog box.
5. Click **Next** in the Select a Procedure dialog box. The Define Application View dialog box is displayed.



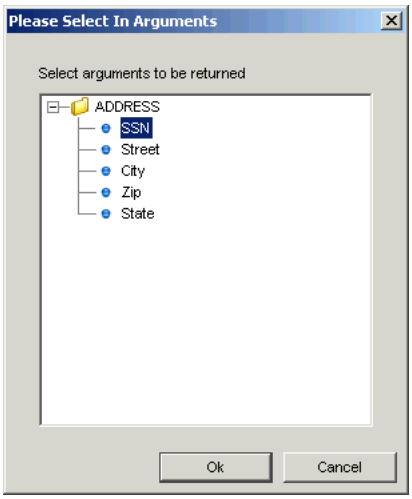
Once a procedure is selected to invoke, the application view is defined. The application view dialog box is initially an empty table.

6. Define the attributes by using the **Add** button, or import the definitions from a database or an API Repository by using the **Import** button.

See Also:

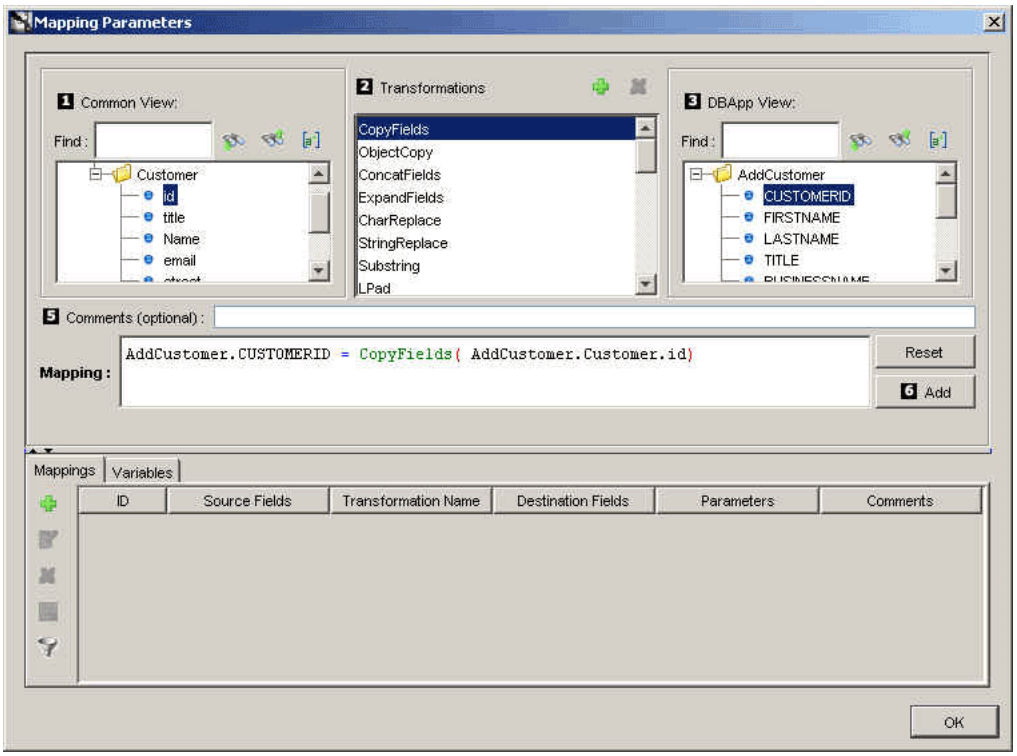
- "Adding Attributes" on page 3-3
- "Importing Attributes" on page 3-4
- "Deleting and Clearing Attributes" on page 3-5
- Appendix C, "Using the Data Definition Description Language"

7. Click **Returned In Args** to specify IN arguments to be returned. The Please Select In Arguments dialog box is displayed.



8. Select the input and output arguments to be returned. Use the left mouse button to select multiple arguments. The input arguments which are not user-defined are displayed for selection.
9. Click **OK** to return to the Define Application View dialog box.
10. Click **Next**. The Define Mapping IN Arguments dialog box is displayed.

    Mapping arguments involves copying the individual fields or simple shape-change transformations.
11. Click **New** to define mappings. The Mapping Parameters dialog box is displayed.



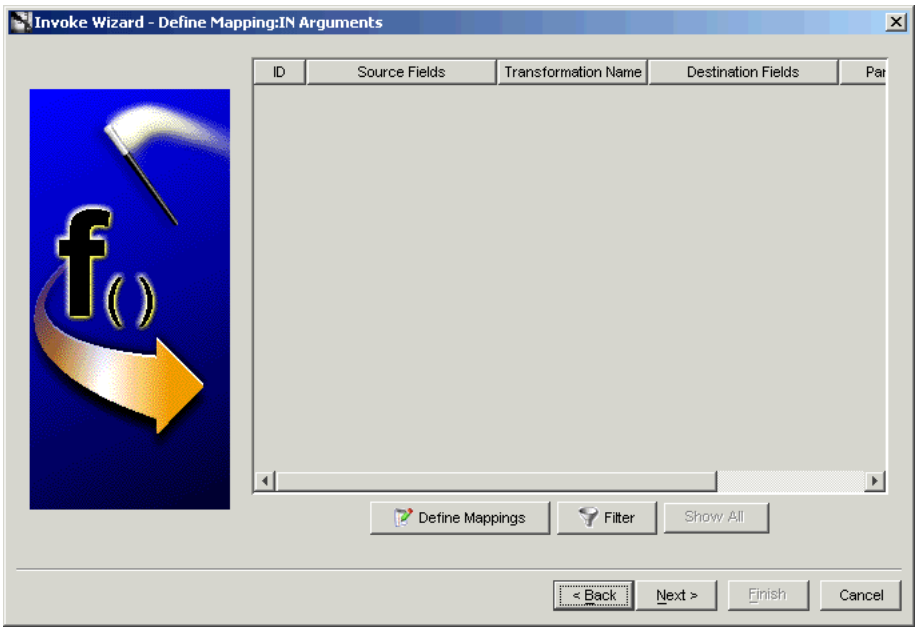
To map fields in the application view to fields in the common view, use a transform. To use a transform to map fields,

1. Select fields to map from in the application view. Use the left mouse button to select multiple fields in a view.
2. Select the transformation, for example, `ConcatFields`.
3. Select the fields to map to in the common view. Use the left mouse button to select multiple fields in a view.
4. Click **Apply** to confirm selection and continue specifying additional mappings.
5. When all mappings have been made, click **OK**.

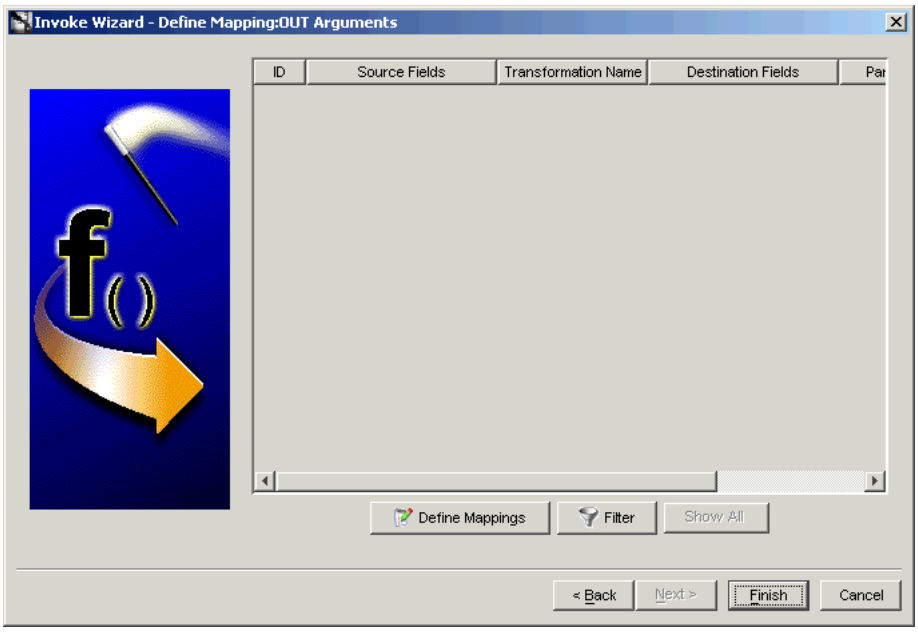
See Also: "Invoking a Procedure" on page 5-3

12. Click **Next**. The Define Mapping IN Arguments dialog box is displayed.

    Mapping arguments involves copying the individual fields or simple shape-change transformations. Use this dialog box to map the common view return arguments to the application view return arguments.



13. Click **Next**. The Define Mapping OUT Arguments dialog box is displayed.



14. Click **New** to define mappings.

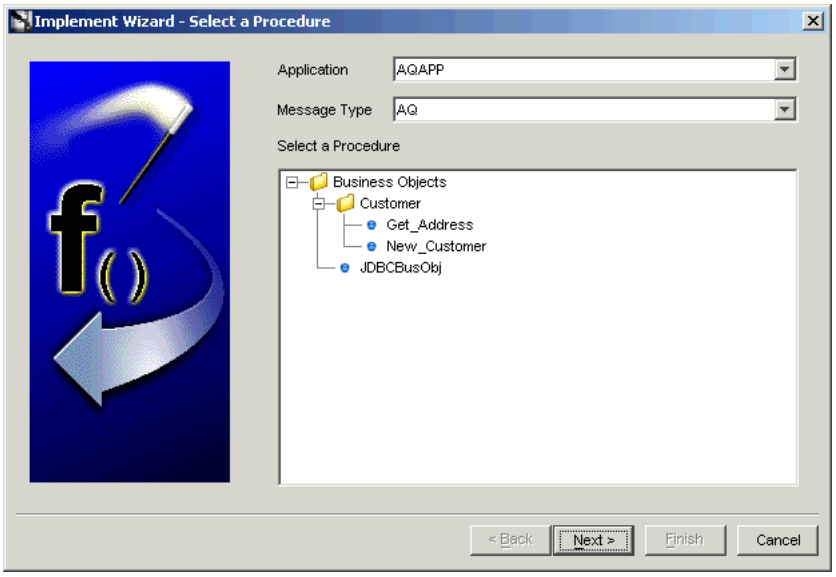
    See Also: Step 5 on page 5-4

15. Click **Finish**.

5.2.2 Implementing a Procedure

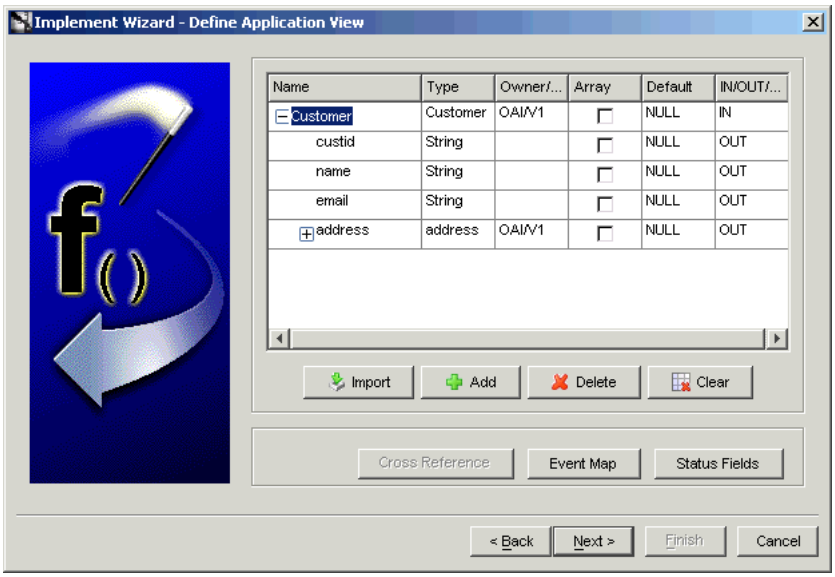
Implement a procedure in iStudio using the Implement Wizard. To start the Implement Wizard:

1. In the Design navigation list, expand the **Application** node.
2. Right-click **Implemented Procedures**, and select **New**. The Implement Wizard is displayed.



Use this dialog box to select a procedure to implement.

3. Specify information for the following fields:
   - Application: The name of the application selected in the navigation list, which invokes the procedure, appears selected by default. Select an application from the list.
   - Message Type: This field specifies the mode of communication between OracleAS Integration InterConnect and the application. Select from the following message types:
     - Database: OracleAS Integration InterConnect communicates with the application by using the database.
     - Generic: OracleAS Integration InterConnect communicates with the application by using a user-defined bridge.
     - XML: OracleAS Integration InterConnect communicates with the application using XML data described through a DTD or XSD by using the FTP, SMTP, HTTP, and MQSeries adapters.
     - AQ: OracleAS Integration InterConnect communicates with the application through Oracle Advanced Queues by using the Advanced Queue adapter. The payload can be Oracle Objects where fields may be XML or RAW XML.
     - D3L: The adapter communicates with the application by using D3L.
     - JCA: OracleAS Integration InterConnect communicates with the application by using the J2CA adapter.
4. Select the procedure to invoke.
5. Click **Next**. The Define Application View dialog box is displayed.



Initially, this dialog box is an empty table. Attributes can be defined by using Add. Attribute definitions can be imported from a database or an API Repository by using Import.

See Also:

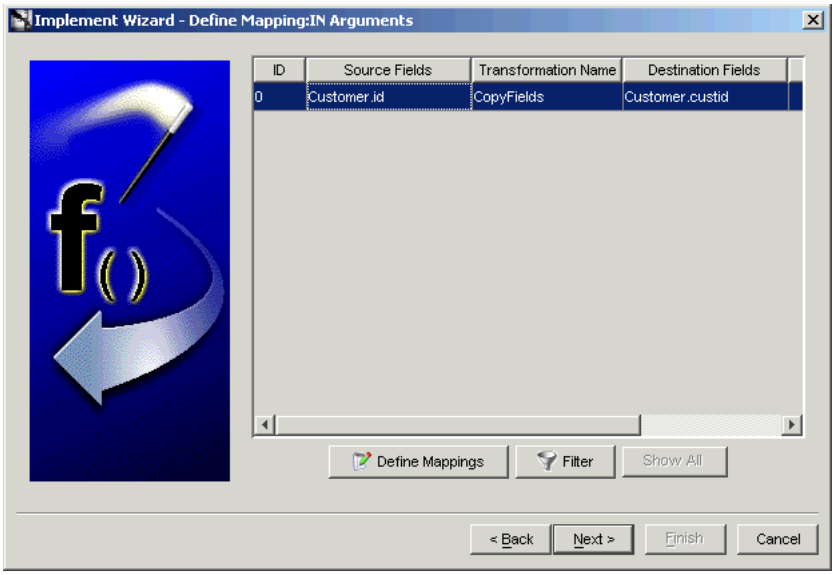
- "Adding Attributes" on page 3-3
- "Importing Attributes" on page 3-4
- "Deleting and Clearing Attributes" on page 3-5
- Appendix C, "Using the Data Definition Description Language"

6. Click **Cross Reference** to populate cross-reference tables.

   See Also: "Populating Cross-reference Tables" on page 6-9

7. Click **Next**. The Define Mapping IN Arguments dialog box is displayed.

   Mapping may involve copying individual fields, or simple shape-change transformations.



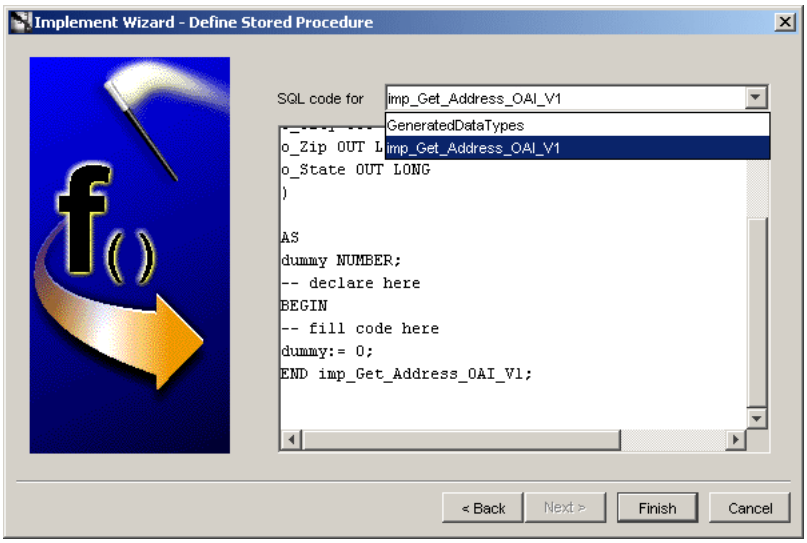
8. Click **New** to define IN mappings.

   See Also: Step 5 on page 5-4

9. Click **Next**. The Define Mapping OUT Arguments dialog box is displayed.
10. Click **New** to define OUT mappings.

    See Also: Step 5 on page 5-4

11. Click **Next**. The Define Stored Procedure dialog box is displayed.



If the message type selected was database, then the data is received by a stored procedure. In this stored procedure, the action performed when the values are returned to the application can be specified. The adapter invokes the stored procedure at run time with the corresponding data.

The following arguments will be returned:

- All `OUT` arguments.
- All `IN` arguments specified to be returned as part of the reply.

12. Select a generated procedure from the SQL Code For list.
13. Click **Finish**.

6

Enabling Infrastructure

This chapter describes the infrastructure tasks in iStudio. It contains the following topics:

6.1 Enabling Infrastructure

Enabling Infrastructure provides additional important features that are critical to enable an end-to-end integration. These features include:

- Content Based Routing. Route messages by building business rules based on message content. For example, a procurement system routes fulfillment requests to different fulfillment centers based on an originating location.

  Content based routing can be done for both events and procedures. The adapter that publishes an event or invokes a procedure, evaluates the content based routing rules to determine the recipient list. This recipient list is then added to the JMS header for the message before it is sent to the OracleAS Integration InterConnect hub. In the hub, AQ looks up the recipient list and wakes up the relevant target adapters.

  > **Note:** When defining content based routing rules, make sure that all routing cases are covered for all possible values of the fields used in the rule. Once you define even one content based routing rule, the default event based routing rules are no longer in effect.

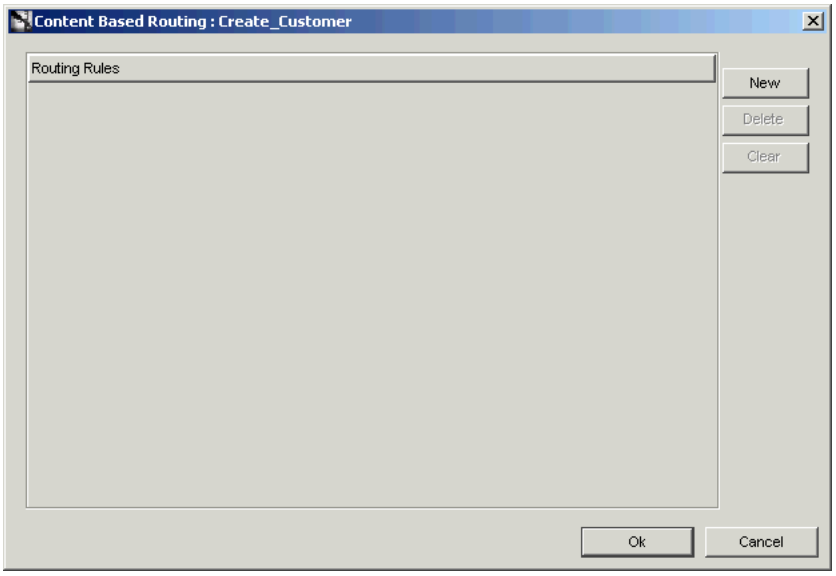
- Domain Value Mapping. Map code tables across systems. For example, a purchase order in a procurement system has a `PO Status` field with domain values `Booked` and `Shipped`. The corresponding field in a fulfillment system has the domain values `1` and `2`. OracleAS Integration InterConnect enables the user to create mappings such as `booked=1`, `shipped=2` so that it can correlate these values at run time without each system understanding the domain value set of the other system.
- Cross Referencing. Correlate keys that uniquely identify the entities in one application with corresponding entities created in other applications. For example, a purchase order created in a procurement system has a native ID `X`. The purchase order is then routed to a fulfillment system, where it is created with native ID `Y`.

As a result, X and Y must be cross-referenced for OracleAS Integration InterConnect to correlate communication about this same logical entity in two different systems without each system understanding the native ID of the other system.

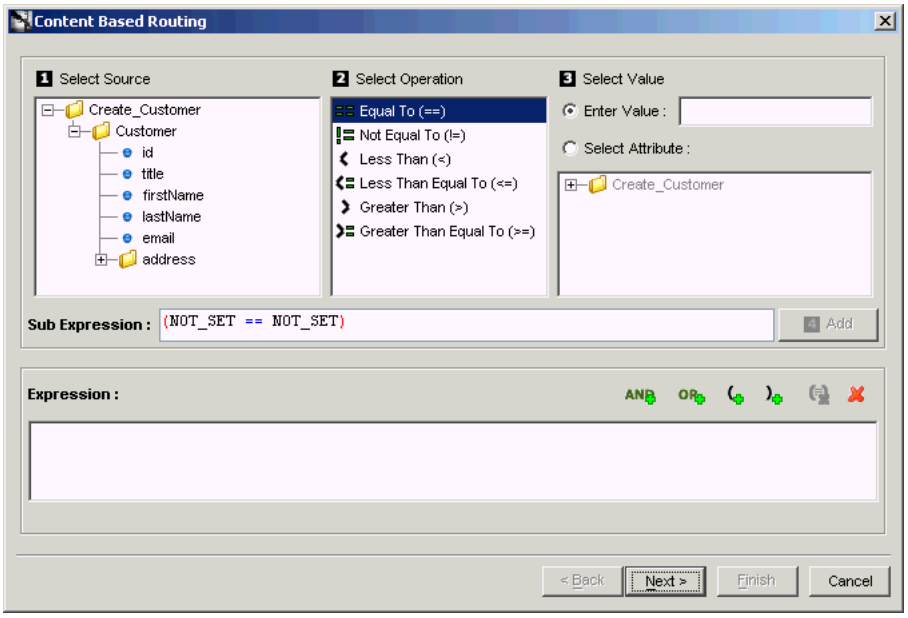
6.2 Working with Content Based Routing

To modify content-based routing for an event or procedure:

1. Right-click the event or procedure under the **Content Based Routing** node in the Design Navigation list, and then click **Edit**. The Content Based Routing Rules dialog box is displayed.



2. Click **New**. The following Content Based Routing dialog box is displayed.



3. Perform the following:

a. Select the source event attribute to be used for building rules from the Select Source list.

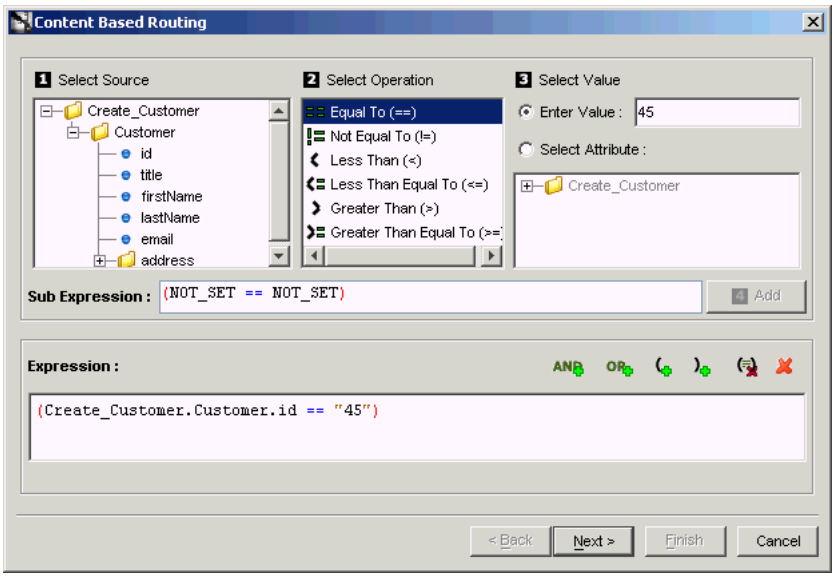
b. Select an operator from the Select Operation list.

c. For literal values, enter a value in the **Enter Value** field. For attributes, select an attribute from the Select Attribute list. Use Select Attribute to compare one value in an attribute to another. An attribute can be selected either from the message payload (common view data) or the message header that accompanies this payload.

d. Click **Add**.

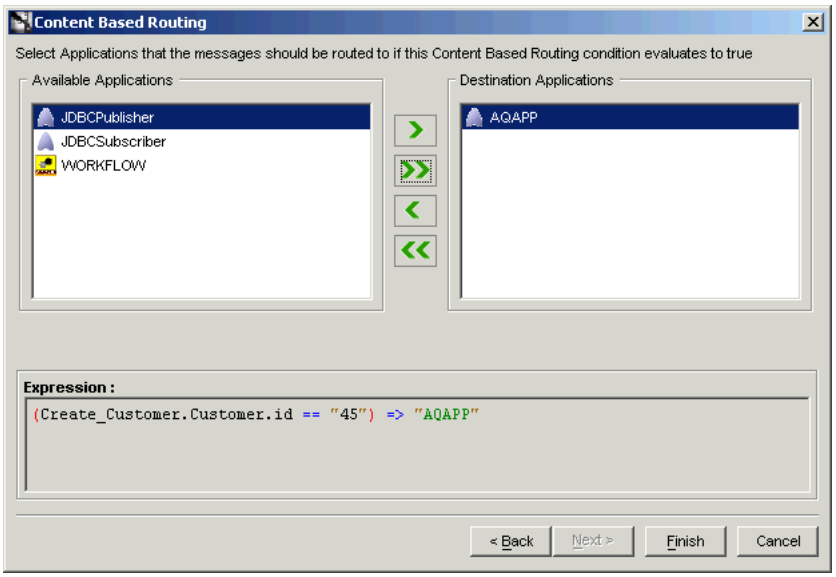
e. You can add to the condition by using the AND and OR operators. To add to a rule, click **Append "(" to expression** button. This can be used to build complex routing rules such as `Age<50 AND Salary>100000 OR AGE >=50`.

Figure 6–1 Creating Content Based Routing

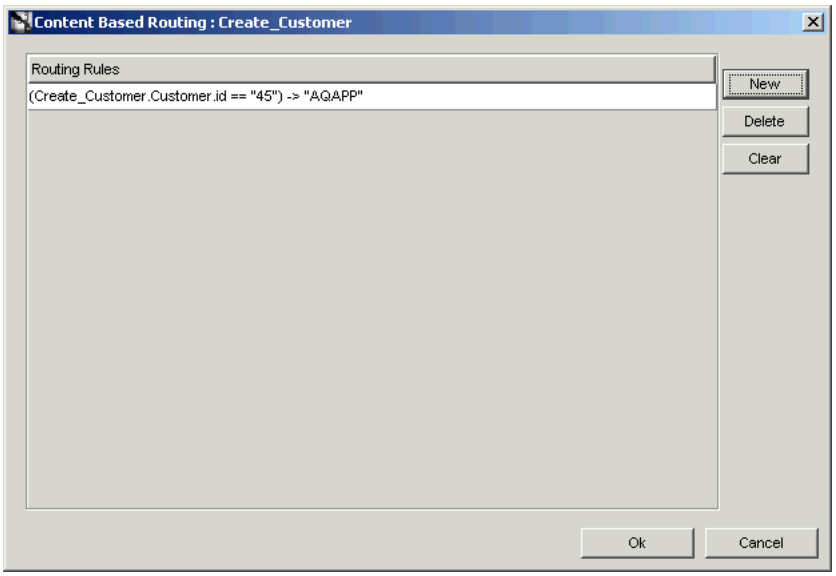


4. Click **Next**. The Content Based Routing dialog box is displayed.

5. Select one or more applications from the Available Applications list and click the right arrow button.



6. Click **Finish**. The content based routing Rule is created, and it is displayed in the Content Based Routing dialog box.

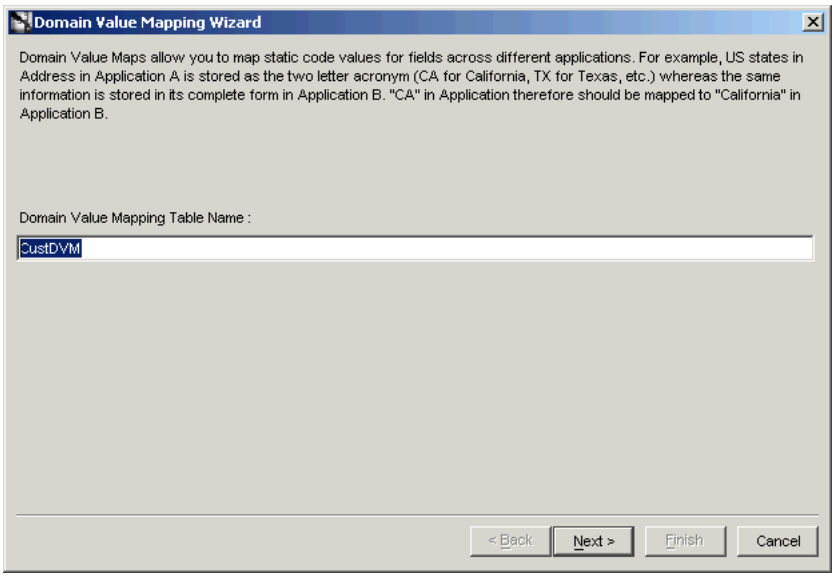


7. Click **New** to add another rule or click **Ok** to finish.

6.3 Working with Domain Value Mappings

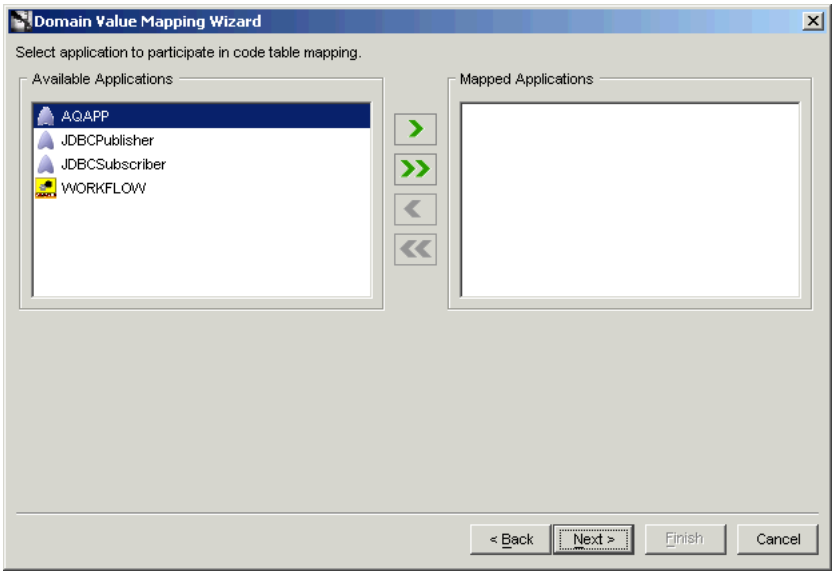
To create a domain value mappings table:

1. In iStudio, right-click **Domain Value Maps** under Enabling Infrastructure.
2. Select **New**. The Create Domain Value Mapping Wizard is displayed.



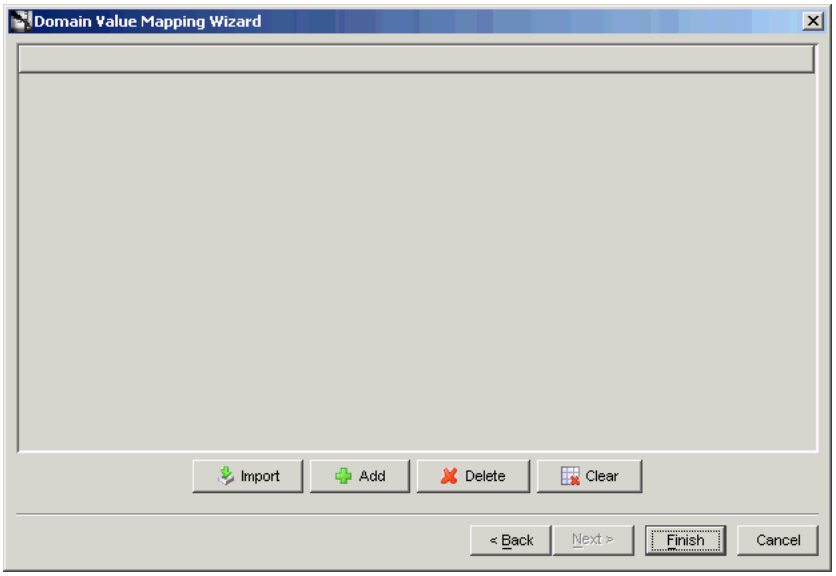
3. Enter a name for the domain value map in the **Domain Value Map Table Name** field.
4. Click **Next**.
5. Select an application name from the **Available Applications** list, as shown in Figure 6–2.

Figure 6–2 Adding Applications to a Domain Value Mapping



6. Click the right arrow button to add the selected application to the Mapped Applications list.
7. Click **Next**.
8. Click **Add** to add mappings, or **Import** to import mappings, as shown in Figure 6–3.

Figure 6–3 Adding or Importing Mappings to a Domain Value Mapping



9. Click **Finish**.

6.3.1 Adding Applications to Domain Value Mappings

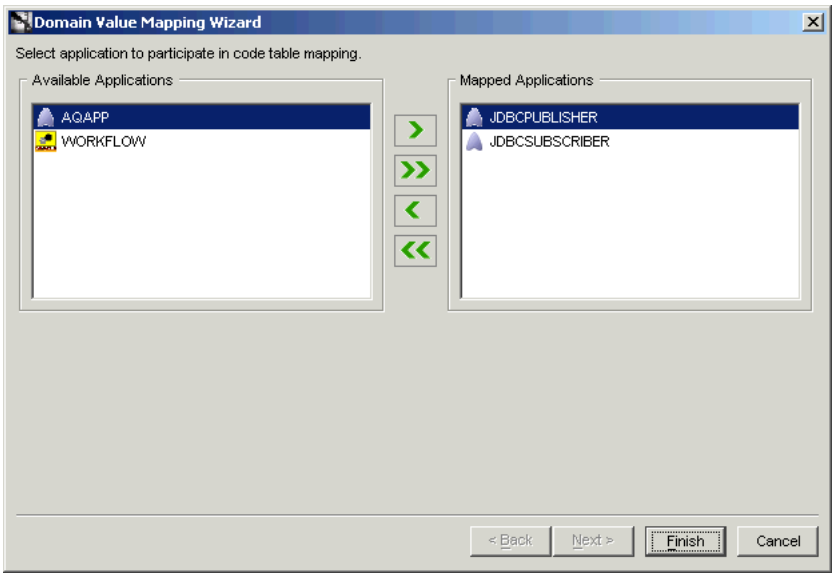
To add applications to domain value mappings:

1. In the Design Navigation list, right-click a domain value mapping.
2. From the context menu, select **Add Application**. The Domain Value Mapping Wizard is displayed as shown in Figure 6–2.
3. Select an application name from the **Available Applications** list.
4. Click the right arrow button to add the selected application to the Mapped Applications list.
5. Click **Finish**.

6.3.2 Removing Applications from Domain Value Mappings

To remove applications from the domain value mappings:

1. In the Design Navigation list, right-click a domain value mapping.
2. From the context menu, select **Remove Application**. The Domain Value Mapping Wizard is displayed.

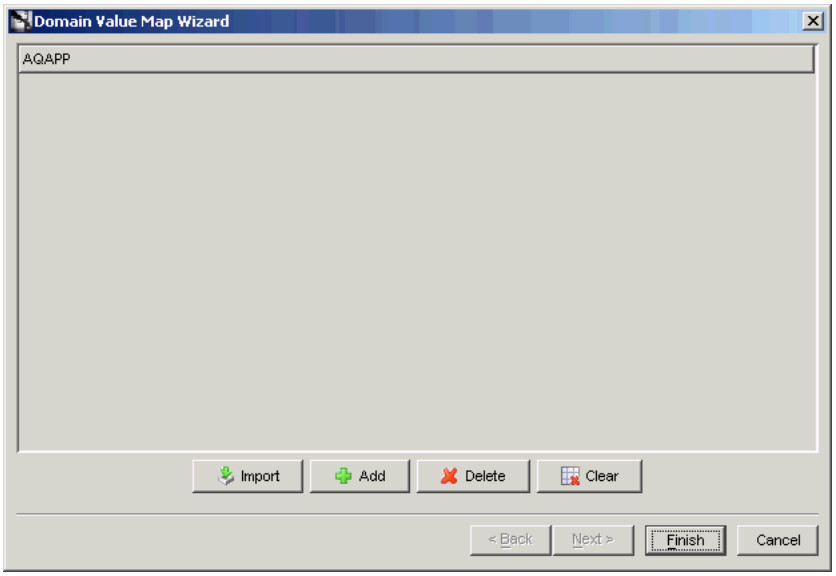


3. Select the application name to remove from the **Mapped Application** list.
4. Click the left arrow button.
5. Click **Finish**.

6.3.3 Modifying Domain Value Mappings

To modify data domain value mappings:

1. In the Design Navigation list, right-click a domain value mapping.
2. From the context menu, select **Edit Values**. The Domain Value Map Wizard is displayed.



3. Click **Add** to add mappings, or **Import** to import mappings.
4. Click **Finish**.

6.3.4 Deleting Domain Value Mappings

To delete a selected domain value mapping:

1. Select the domain value mapping to be deleted.
2. Click **Delete**.

6.3.5 Deleting Domain Value Mapping Tables

To delete a domain value mapping table:

1. Select the domain value mapping table to be deleted and right-click.
2. From the context menu, select **Delete**.
3. Click **Yes** in the Confirm Delete dialog box.

6.3.6 Modifying Attribute Mappings

To modify a selected attribute mapping, use the Define Mapping dialog box on the Publish Wizard:

See Also: "Publishing an Event" on page 4-3

1. Select a mapping, and click **Edit Mapping**.
2. Edit the fields of the selected mapping.
3. Click **OK**.

6.3.7 Removing Attribute Mappings

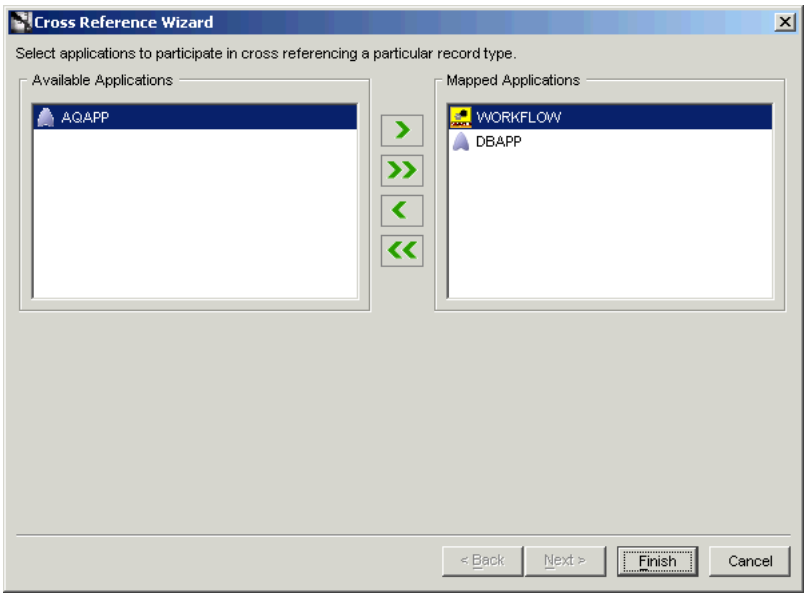
To remove attribute mappings, use the Define Mapping dialog box in the Publish Wizard.

- To remove a mapping, delete the attribute and click **Remove**.
- To remove all mappings, click **Clear**.

6.4 Working with Cross Referencing

Creating a cross-reference in iStudio creates a table in the repository schema. To create a cross-reference table:

1. In iStudio, right-click **Cross Reference Tables** under Enabling Infrastructure.
2. Select **New**. The Cross Reference Wizard is displayed.
3. Enter a name for the cross reference table in the **Cross Reference Table Name** field.
4. Click **Next**.
5. Select an application name from the **Available Applications** list.
6. Click the right arrow button to add the selected application to the Mapped Application list.



7. Click **Finish**.

6.4.1 Adding Applications to Cross-reference Tables

To add applications to a cross-reference table:

1. In the Design Navigation list, select a cross-reference table and right-click.
2. From the context menu, select **Add Application**. The Cross Reference Wizard is displayed.
3. Select an application name from the **Available Applications** list.
4. Click the right arrow button to add the selected application to the Mapped Applications list.
5. Click **Finish**.

6.4.2 Removing Applications from Cross-reference Tables

To remove applications from a cross-reference table:

1. In the Design Navigation list, select a cross-reference table and right-click.
2. From the context menu, select **Remove Application**. The Cross Reference wizard is displayed.
3. Select the application name to be removed from the **Mapped Applications** list.
4. Click the left arrow button.
5. Click **Finish**.

6.4.3 Populating Cross-reference Tables

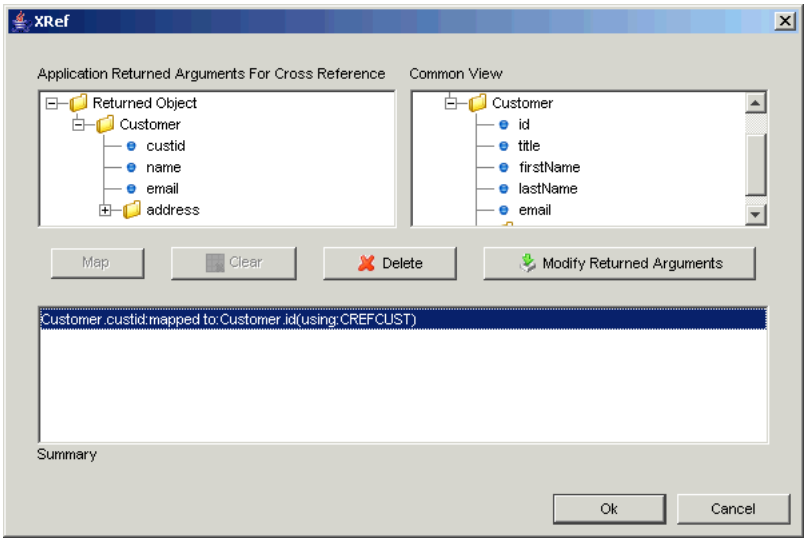
To populate cross-reference tables, returned arguments must first be defined.

Use the Subscribe Wizard to access the correct dialog box for populating cross-reference tables.

See Also: "Subscribing to an Event" on page 4-8

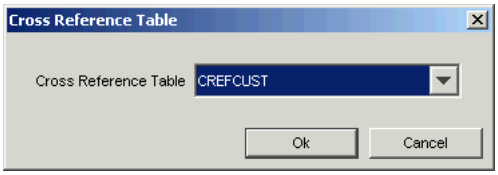
To populate cross-reference tables:

1. Click **Cross Reference** in the Define Application View dialog box. The XRef dialog box is displayed.



   The Application Returned Arguments box displays the returned arguments. This information is initially populated with the `OUT` arguments from the application view.

2. Click **Modify Returned Arguments** to modify the returned arguments list.
3. Select corresponding attributes in the Application Returned Arguments For XRef and Common View boxes, then click **Map**. The Cross-reference Table dialog box is displayed.



4. Specify the cross reference table name to be populated using these attributes values.
5. Click **OK**.

7

Using Oracle Workflow

This chapter discusses using Oracle Workflow to apply business logic to an integration. It contains the following topics:

- Oracle Workflow Overview
- OracleAS Integration InterConnect Integration with Oracle Workflow
- Using Oracle Workflow with OracleAS Integration InterConnect
- Model Business Process

7.1 Oracle Workflow Overview

Oracle Workflow is integrated with OracleAS Integration InterConnect. In the context of OracleAS Integration InterConnect, Oracle Workflow is used for business process collaborations across two or more applications. A business process collaboration is defined as the conversation between two or more applications in the context of a business process.

OracleAS Integration InterConnect leverages the robust design-time and run-time Oracle Workflow business process definition and execution support to make these business processes explicit and manageable.

> **Note:** Knowledge of Oracle Workflow, its tools, and its Business Event System is required to use OracleAS Integration InterConnect with Oracle Workflow for business process collaboration. For more information about Oracle Workflow, refer to *Oracle Workflow Administrator's Guide*.

7.1.1 Solving Common Business Problems with Oracle Workflow

The following are some of the common business problems that can be solved using Oracle Workflow.

7.1.1.1 Error Management and Compensating Transactions

If there is an interaction problem between two or more applications, then the errors arising from the problem can be centrally managed, and suitable remedial actions can be defined and performed.

Example 7–1 Oracle Workflow, OracleAS Integration InterConnect, and Error Management

Consider a situation where it may be required to keep data of an order entry system synchronized with a backend Enterprise Resource Planning (ERP) system. Assume that a new purchase order is created in the order entry system and an attempt is made to create a corresponding new purchase order through messaging by using OracleAS Integration InterConnect in the backend ERP system. The attempt fails. To deal with this scenario, the integrator can use Oracle Workflow to automatically send a compensating message to the order entry system to undo the creation of the purchase order and notify the user who created the order.

In this example, OracleAS Integration InterConnect and Oracle Workflow can be used to model the following for every purchase order that is over $50,000:

- Send a notification to a named approver and wait for approval.
- If approved, send the message to the ERP system. Otherwise, send a message to the order entry system to roll back the order creation.

7.1.1.2 Human Interaction

OracleAS Integration InterConnect adds human interaction to better capture business processes. In the preceding example, OracleAS Integration InterConnect and Oracle Workflow can be used to model the following:

For every purchase order that is over $50,000, send a notification to a named approver, and wait for approval. If approved, then send the message to the ERP system; otherwise send a message to the order entry system to roll back the order creation.

7.1.1.3 Message Junctions

Fan-in and fan-out of messages can be effectively modeled using OracleAS Integration InterConnect and Oracle Workflow. Fan-in messages involve combining two or more messages into one message. Fan-out messages involve splitting one message into two or more messages.

Example 7–2 Fan-in and Fan-out of Messages

A global organization has a centralized Human Resources ERP application in the United States. Each country has one or more local systems that capture local employee information. If a new employee joins the Japanese branch of this organization, data is entered into a local human resources application and local benefits application. Each entry submits a message for adding this information to the centralized system.

The centralized system needs data from both systems combined and will only commit the data if it was entered successfully in both of the local systems connecting to OracleAS Integration InterConnect. Using Oracle Workflow, this process can be modeled so that OracleAS Integration InterConnect routes messages from both local systems to Oracle Workflow. Oracle Workflow waits until it receives both messages, combines the data, and launches a single message to be delivered by OracleAS Integration InterConnect to the centralized human resources system.

7.1.1.4 Stateful Routing

OracleAS Integration InterConnect provides extensive support for stateless routing through event-based and content-based routing features. Using Oracle Workflow, stateful routing can be accomplished. The decision to route can be based on the event or the content of the message.

7.1.1.5 Composite Services

An internal (organization focused) or external (customer/partner focused) service can be built through a well-defined set of business processes involving communication between two or more applications. For example, a brick-and-mortar retail company wants to provide an online procurement service to their customers. Behind the user interface are several business processes controlling communication across several internal applications to deliver a robust, high-performance service to the customer.

Note: The ability to define explicit business process collaborations is a feature, not a requirement for completing integrations. It is not necessary to use Oracle Workflow for integration if the business process definition is simple enough to be implicitly captured in the messaging through the core functionality in iStudio.

7.2 OracleAS Integration InterConnect Integration with Oracle Workflow

This section describes how OracleAS Integration InterConnect and Oracle Workflow are integrated. It includes the following sections:

- Design-Time Tools
- Run Time

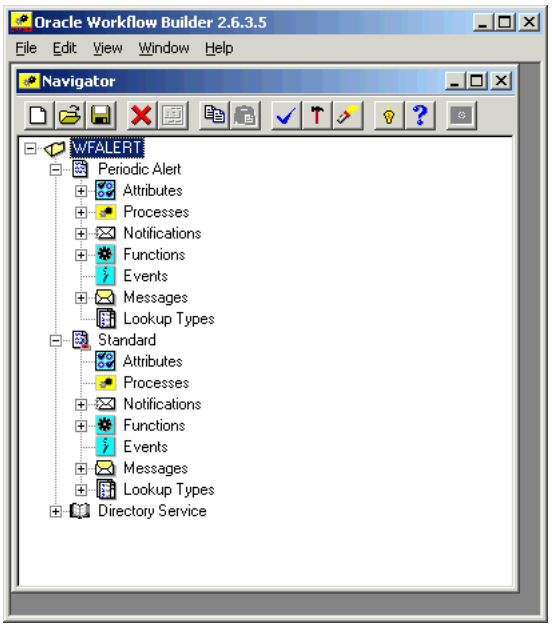
7.2.1 Design-Time Tools

During design time, business process and event definitions in iStudio can be deployed to Oracle Workflow. Consequently, Oracle Workflow tools can be launched from within iStudio to graphically create process diagrams in the context of enterprise integration through OracleAS Integration InterConnect.

Using iStudio, the following Oracle Workflow tools can be used:

- Oracle Workflow Builder: Use this tool to complete business process definitions defined and deployed through iStudio. Oracle Workflow Builder creates the process diagrams as shown in Figure 7–1.

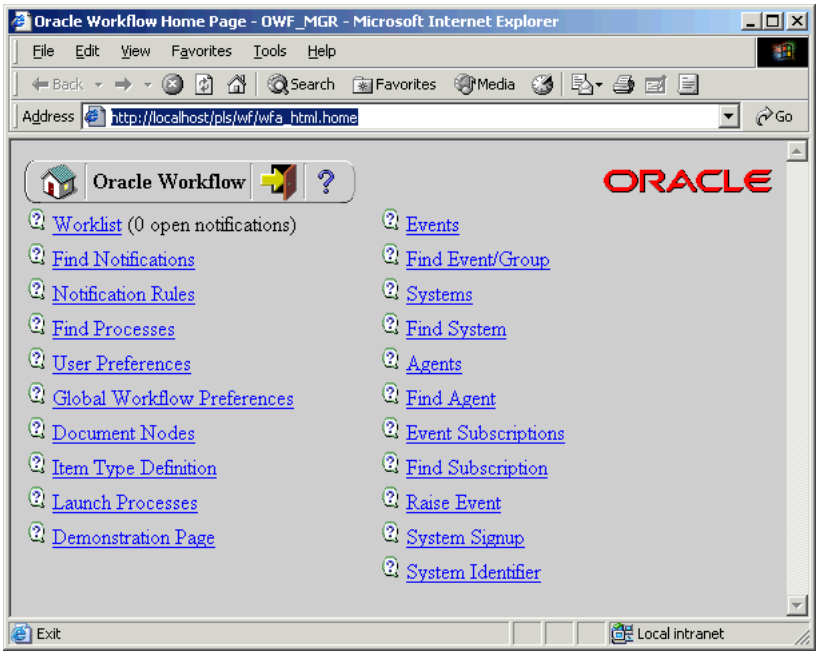
Figure 7–1 Oracle Workflow Builder



- Oracle Workflow Home Page: Use this tool for centralized access to the Web-based features of Oracle Workflow. The Business Event System management and administration is a key feature on this page as shown in Figure 7–2.

  See Also: For more information about the Business Event System, refer to *Oracle Workflow Administrator's Guide*

Figure 7–2 Oracle Workflow Home Page



7.2.2 Run Time

OracleAS Integration InterConnect integrates with the Business Event System of Oracle Workflow. The Business Event System is an application service that uses the Advanced Queuing infrastructure to communicate business events between systems. OracleAS Integration InterConnect registers itself as an external system in Business Event System so the following conditions exist:

- Messages can flow from applications through OracleAS Integration InterConnect, in the common view format, to the Business Event System. The messages will either trigger an event or continue Oracle Workflow business processes, as defined by iStudio processes and described in Oracle Workflow Builder diagrams.
- Messages can flow from the Business Event System to OracleAS Integration InterConnect in the common view format to applications to either continue or end Oracle Workflow business processes.

At run time, Oracle Workflow is integrated with OracleAS Integration InterConnect at the hub. Messages are passed between OracleAS Integration InterConnect and the Business Event System of Oracle Workflow by using Advanced Queues. The OracleAS Integration InterConnect Oracle Workflow Communication Infrastructure facilitates this communication.

At design time, to keep the integration methodology consistent, iStudio reuses the messaging paradigms of publish/subscribe and request/reply to specify communication between OracleAS Integration InterConnect and Oracle Workflow. For inbound Oracle Workflow messages, an iStudio user can specify using a business process, which events Oracle Workflow should subscribe to and which procedures Oracle Workflow should implement. For outbound messages, events Oracle Workflow can publish and procedures it can invoke can be specified.

See Also: "Using Oracle Workflow with OracleAS Integration InterConnect" on page 7-5

7.3 Using Oracle Workflow with OracleAS Integration InterConnect

OracleAS Integration InterConnect can be used with Oracle Workflow. The following steps describe, in general terms, how to apply business logic:

1. Model Business Process. The user designs the business process using iStudio, and then deploys the process bundles from iStudio to a `WFT` file. Next, the user will complete the process diagrams in Oracle Workflow Builder using the WFT file.
2. Deploy Business Processes for Run Time. The user will deploy the events to the Business Event System by using iStudio. Next, the user will deploy the process diagram to the database by using Oracle Workflow Builder.

7.3.1 Model Business Process

To model the business process:

- Design process bundles by using iStudio.
- Deploy process bundles from iStudio to a `WFT` file.
- Complete process diagrams in Oracle Workflow Builder by launching Oracle Workflow Builder from iStudio and by using the deployed `WFT` file.

7.3.2 Deploy Business Processes for Run Time

To deploy business processes for run time:

- Deploy events to the Business Event System from iStudio.
- Deploy a process diagram from a file to the database using Oracle Workflow Builder.

7.4 Model Business Process

This section describes how iStudio and Oracle Workflow work together in OracleAS Integration InterConnect. It also has instructions about how to use iStudio and Oracle Workflow during design time for business process collaborations across applications. This section includes the following topics:

- Process Bundle
- Business Process
- Activity

7.4.1 Process Bundle

A process bundle is a set of logically related business processes. The process bundles map one-to-one with an Oracle Workflow item.

See Also: "Business Process" on page 7-6

7.4.2 Business Process

A business process contains a set of OracleAS Integration InterConnect common view events or procedures that must be routed through Oracle Workflow in one Oracle Workflow business process. These events and procedures manifest themselves as Oracle Workflow business events and can be used to define a process diagram in Oracle Workflow Builder. It is a one-to-one mapping between OracleAS Integration InterConnect and Oracle Workflow Builder.

7.4.3 Activity

Activities in iStudio enable the user to define the common view events and procedures that must be a part of an Oracle Workflow business process. The following are types of activities in iStudio:

- Publish Event: Oracle Workflow publishes an OracleAS Integration InterConnect common view event. At deployment time, a business event corresponding to the common view event is created in the Business Event System.
- Subscribe Event: Oracle Workflow subscribes to an OracleAS Integration InterConnect common view event. At deployment time, a business event corresponding to the common view event is created in the Business Event System.
- Invoke Procedure: Oracle Workflow invokes an OracleAS Integration InterConnect common view procedure. At deployment time, two business events corresponding to the common view procedure are created in the Business Event system. One event is for sending the request, and the other is for receiving the reply.
- Implement Procedure: Oracle Workflow implements an OracleAS Integration InterConnect common view procedure. At deployment time, two business events corresponding to the common view procedure are created in the Business Event

System. One event is for receiving the request, and the other is for sending the reply.

The following table describes how iStudio and Oracle Workflow concepts are mapped.

iStudio Concept	Oracle Workflow Concept	Mapping
Process Bundle	Item	One-to-one
Business Process	Business Process	One-to-one
Common View Event	Business Event	One-to-one[1]
Common View Procedure	Business Event	Two business events for each procedure
Publish Activity	Send Event Activity	One-to-one
Subscribe Activity	Receive Event Activity	One-to-one
Invoke Activity	Send Event Activity (for the request)	
	Receive Event Activity (for the reply)	
Implement Activity	Send Event Activity (for the reply)	
	Receive Event Activity (for the request)	

[1] Only for events that are part of a business process in iStudio. Events that are part of the common view but not part of a business process are not instantiated as Oracle Workflow business events. All common view events need not be part of business processes. Depending on the integration, some common view events could be exchanged directly between applications without involving Oracle Workflow. These events use the core functionality of OracleAS Integration InterConnect. Other events may need to be part of an explicit business process.This set of events become business events in Oracle Workflow. The same is true for common view procedures.

7.4.4 Creating a Process Bundle

To create a process bundle using iStudio:

1. From the project list, click **Workflow**, and expand the subtree.
2. Right-click **Process Bundles**, and select **New**. The Create Process Bundle dialog box is displayed.
3. Enter the name of the process bundle in the **Process Bundle Name** field.
4. Click **OK**.

7.4.5 Creating a Business Process

To create a business process:

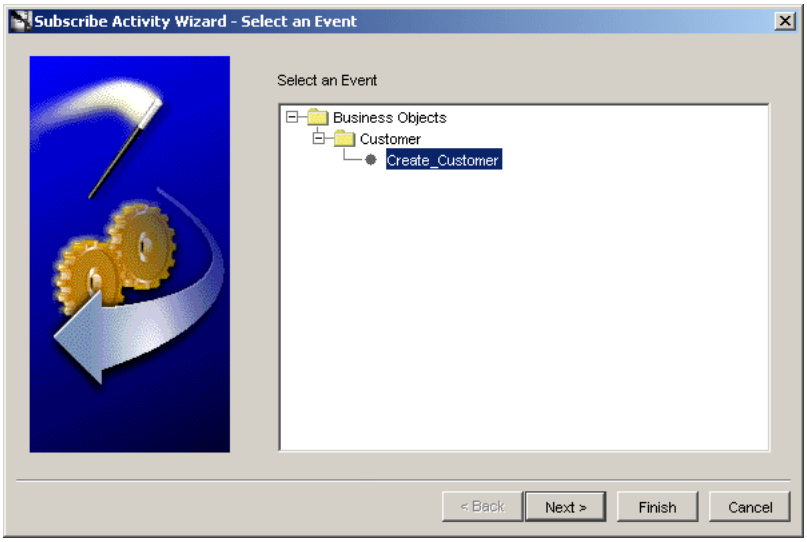
1. From the project list, expand the process bundle for the business process to be created.
2. Right-click **Business Processes**, and select **New**. The Create Business Process dialog box is displayed.
3. Enter a name for the business process in the **Business Process Name** field.
4. Click **OK**.

7.4.6 Populating a Business Process with Activities

To populate a business process with activities:

1. From the project list, select a business process.
2. Right-click the business process and in the context menu, select the activity to be part of the business process. Select from the following activities:
   - Publish Activity: Oracle Workflow sends a message to OracleAS Integration InterConnect in the context of the business process.
   - Subscribe Activity: Oracle Workflow receives a message from OracleAS Integration InterConnect.
   - Invoke Activity: Oracle Workflow sends a request message to OracleAS Integration InterConnect and receives a reply.
   - Implement Activity: Oracle Workflow receives a request from OracleAS Integration InterConnect and sends a reply.

   For example, select the Subscribe Activity. The Subscribe Activity Wizard is displayed.
3. Select an event for the activity.



4. Click **Finish**.

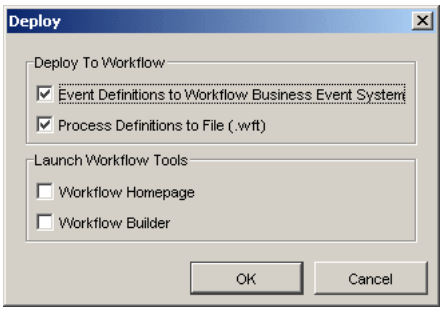
   Repeat these steps for adding other activities to the process.

> **Note:** When you create multiple activities under a business process, the list of activities is unordered because the order in which the activities are added is not important. The order can be defined in Oracle Workflow Builder through a process diagram.

7.4.7 Deploying to Oracle Workflow

After populating business processes with activities, the information must be deployed to Oracle Workflow to graphically model a business process. To deploy this information to Oracle Workflow:

1. In the Deploy tab in iStudio, right-click **Workflow**, and select **Deploy To Workflow**. The Deploy dialog box is displayed.



2. There are two sets of information that need to be deployed, either independently or together:

   - Oracle Workflow Business Events: Business Events need to be created in the Business Event System. This is a requirement for run time only. You can deploy these after all design-time work, including modeling the process, is complete.

     Note: iStudio checks if an event is already deployed before deploying it. You can redeploy all events at any time. If you deploy an event after all design-time work, then you don't have to redeploy the event.

     To check if events have been deployed, launch the Oracle Workflow Home page.

     See Also: "Launching the Oracle Workflow Home Page" on page 7-10

   - Oracle Workflow Process Definitions through `WFT` file generation: Information about business processes captured in iStudio provides a foundation for building process diagrams in Oracle Workflow Builder. Deploying process definitions is required for design time.

     Note: When deploying process definitions, iStudio prompts for a filename. If an existing file is specified, then iStudio will overwrite the file. As a result, if there are existing process definitions in a file modified using Oracle Workflow Builder, then do not select that file name as the target; otherwise, all modifications made will be lost.

   By default, both choices are selected. The dialog box also enables the following to be automatically launched:

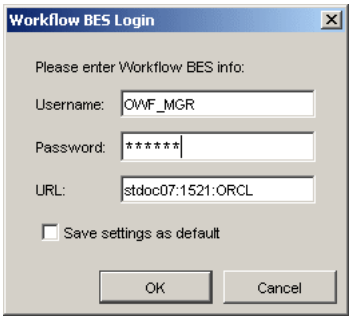
   - Oracle Workflow Builder: Defines business process diagrams.
   - Oracle Workflow Home Page: Verifies Business Event deployment.

   By default, these choices are unselected. Choose to launch these tools with deployment, or complete this task at a later time by using the Design tab.

   See Also: "Launching Oracle Workflow Tools" on page 7-10

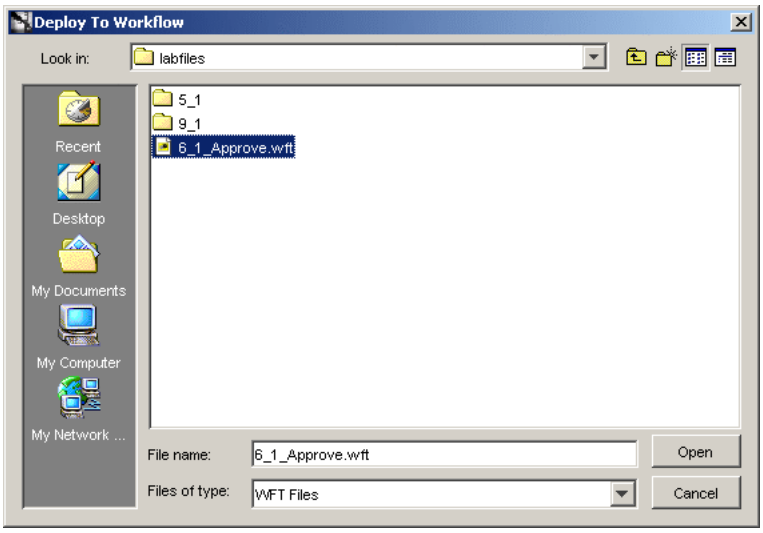
3. Select a choices, and click **OK**.
4. If deploying event definitions to the Oracle Workflow Business Event system is selected, the following dialog box is displayed.

   Enter the required information based on the selections made during Oracle Workflow installation, and click **OK**.



5. If Deploying Process Definitions to a `WFT` file was selected, then a Deploy To Workflow dialog box is displayed.

   Enter the name and location of the file to create, and click **OK**.



Note: When deploying process definitions, iStudio prompts for a file name. If an existing file is specified, then iStudio will overwrite the file. If there are existing process definitions in a file modified using Oracle Workflow Builder, then do not select that file name as the target; otherwise all modifications made will be lost.

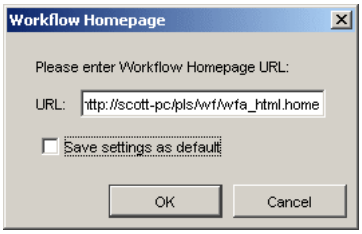
7.4.8 Launching Oracle Workflow Tools

Oracle Workflow tools can be directly accessed through iStudio. You do not need to start Workflow independently. The following sections discuss how to launch Oracle Workflow tools in iStudio.

7.4.8.1 Launching the Oracle Workflow Home Page

To launch the Oracle Workflow Home page:

1. In the Design tab in Studio, right-click **Workflow**.
2. Select **Launch Workflow Homepage**. The Workflow Homepage dialog box is displayed.

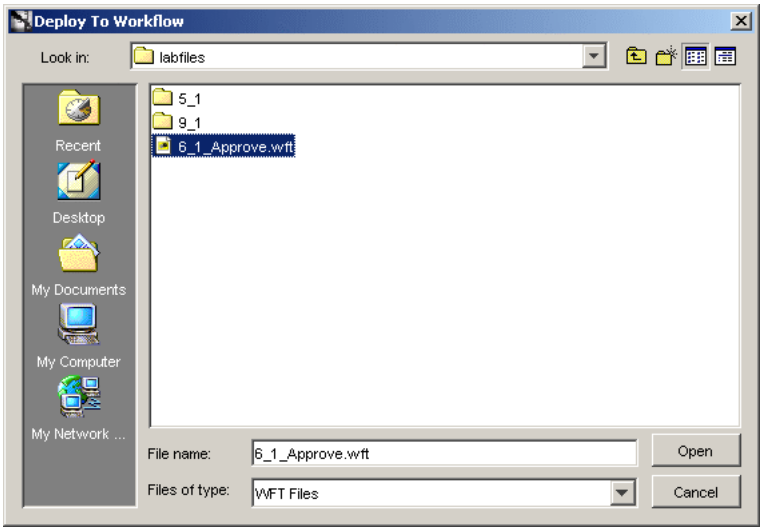


3. Ensure that the URL is correct, and click **OK**. The Username and Password Required dialog box is displayed.
4. Enter the login information for the Oracle Workflow home page, and click **OK**. The Oracle Workflow home page is launched using the default browser.

7.4.8.2 Launching Oracle Workflow Builder

To launch Oracle Workflow Builder:

1. In the Design tab in iStudio, right-click **Process Bundle** to view in Oracle Workflow Builder.
2. Select **Launch Workflow Builder**. The Deploy To Workflow dialog box is displayed.



3. Select an existing `WFT` file name to load into Oracle Workflow Builder. The assumption is that a process definition has already been deployed to a file.

   Oracle Workflow Builder is launched depending on which process definition file is selected.

> **Note:** To launch Oracle Workflow Builder outside of a specific OracleAS Integration InterConnect process bundle, right-click Workflow and select Launch Workflow Builder.

7.4.9 Modifying Existing Oracle Workflow Processes

When modifying existing Oracle Workflow processes, do not add, modify, or remove OracleAS Integration InterConnect event activities directly in Oracle Workflow Builder. Always make event-related process changes in iStudio, redeploy to the file, and import in Oracle Workflow Builder.

For example, the following steps must be carried out to create an integration-related Oracle Workflow business process:

1. Create a process bundle in iStudio and create business processes with some activities.
2. Deploy to the `my_process_bundle.wft` file.
3. Import the file into Oracle Workflow Builder.
4. Make non-event modifications to the process in Oracle Workflow Builder, such as adding notifications or decision functions to complete the business process.
5. Save the modified process to `my_process_bundle.wft`.

If two new events need to be added to the business process, then use the following guidelines:

1. Using iStudio, make the additions to the particular business process.
2. Deploy to a different file such as `changes_to_my_process_bundle.wft`. Do not deploy to `my_process_bundle.wft` because any non-event modifications made through Oracle Workflow Builder will be lost.
3. Launch Oracle Workflow Builder and import both `my_process_bundle.wft` and `changes_to_my_process_bundle.wft`.
4. Move the required modifications from the process representing `changes_to_my_process_bundle.wft` to the process representing `my_process_bundle.wft`.
5. Save the modified process to `my_process_bundle.wft`.

   The `my_process_bundle.wft` file now contains the updated process definition.

8

Deployment

This chapter describes the deployment tasks in iStudio. It contains the following topics:

- Deploying PL/SQL Stored Procedures
- Specifying Application Queue Names for Advanced Queuing Adapter
- Deploying Workflow Events and Process Definitions
- Sync Adapters from iStudio

8.1 Deploying PL/SQL Stored Procedures

PL/SQL stored procedures are generated in iStudio if the database adapter or the Oracle Applications adapter (only for tables, views, or PL/SQL APIs as interfaces) is used to connect to an application. These stored procedures enable an application to interface with OracleAS Integration InterConnect through the Oracle database. Refer to the respective adapter guides for more information about the content of these stored procedures. This code is generated regardless of the integration point used, which is the event for publish/subscribe or procedure for request/reply, and must be deployed in the application schema to be performed at run time. There are two ways to deploy this generated PL/SQL code:

- Manual Deployment
- Auto Deployment

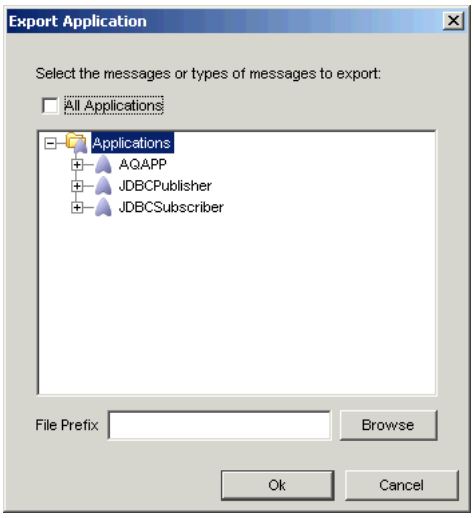
8.1.1 Manual Deployment

This model requires two steps to deploy your code:

- Export the generated code to a file
- Load the exported file manually to the target application schema

This model enables you to use the development environment to modify (if application is receiving data) or build upon (if application is sending data) the generated PL/SQL code. The file export mechanism is also useful for storing the final PL/SQL code in source control systems. To export stored procedures:

1. In iStudio, click the **Deploy** tab.
2. Right-click **Applications**, and select **Export PL/SQL**. The Export Application dialog box is displayed.



3. Select the messages to export. Messages can be filtered as follows:
   - Export all messages: Select Applications at the top of the directory.
   - Export all messages of a certain type for all applications: Check All Applications, then select one or more types of messages to export.
   - Export all messages for a specific application: Select the application name.
   - Export all messages of a certain type for a specific application: Select the type under the application name in the directory.
   - To export specific messages: Select the messages by name. To select more than one message or class of messages, click the application.
4. Enter the name of the file to contain the exported stored procedures in the File Prefix field. The name generates multiple files.

   To view the directory page, click **Browse**.
5. Click **OK**. The stored procedure is now exported to a text file, which is stored in the user-specified directory (iStudio directory by default), on your computer.
6. Load the exported file into the target schema. The exported PL/SQL file is deployed for the selected application.

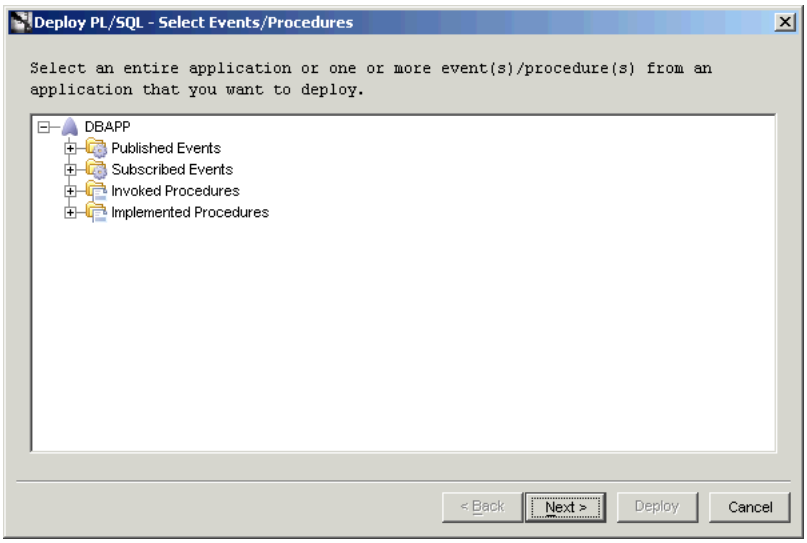
8.1.2 Auto Deployment

This option enables you to deploy the PL/SQL from iStudio, using the Deploy PL/SQL Wizard.

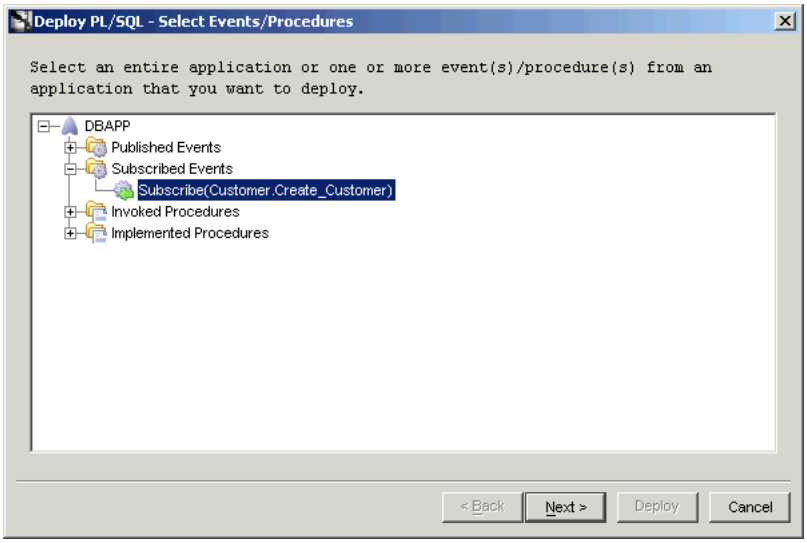
To deploy PL/SQL stored procedures from iStudio:

1. Click the **Deploy** tab in the iStudio window.
2. Right-click a database application and select **Deploy PL/SQL**. The Deploy PL/SQL - Select Events/Procedures dialog box is displayed.

   The application list displays the published/subscribed events and invoked/implemented procedures.

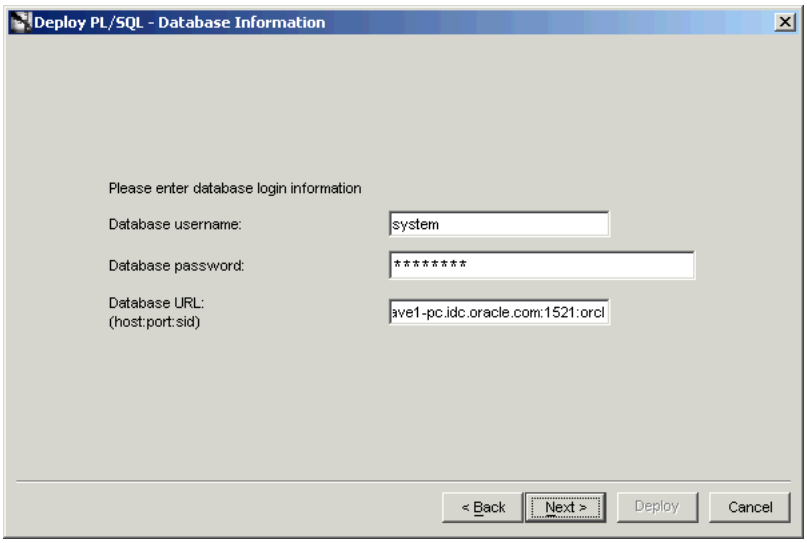


3. Select the application, event, or procedure to deploy the corresponding PL/SQL.

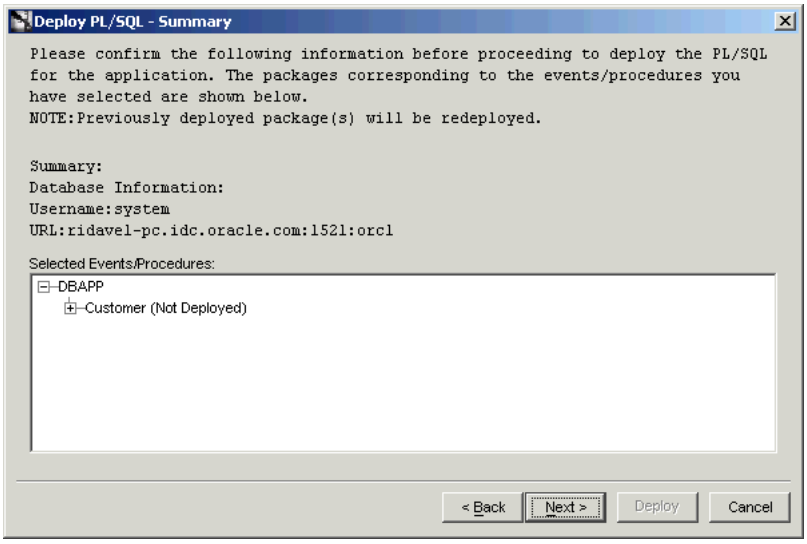


> **Note:** You can deploy PL/SQL code either for one application at a time or at a message level.

4. Click **Next**. The Deploy PL/SQL - Database Information dialog box is displayed. This page enables you to specify the database connection information for deploying the PL/SQL code.



5. Enter information in the following fields:
   - Database username: The database user name required for connecting to the database.
   - Database password: The password required for connecting to the database.
   - Database URL: The URL of the database required for connecting to the database. The URL should be in the form: `host:port:SID`.
6. Click **Next**. The Deploy PL/SQL- Summary dialog box is displayed, which displays a summary of the database connectivity information entered in the previous dialog box.



7. The Deploy PL/SQL - Summary dialog box displays the following:
   - Database Information
   - Selected Events/Procedures

     This page displays a list of selected packages and the corresponding procedures contained in those packages that you have selected for

deployment. The status of each package is displayed in parenthesis next to the package name.

Note: The status can be any of the following:

Not deployed

Package Status Unknown

Previously Deployed

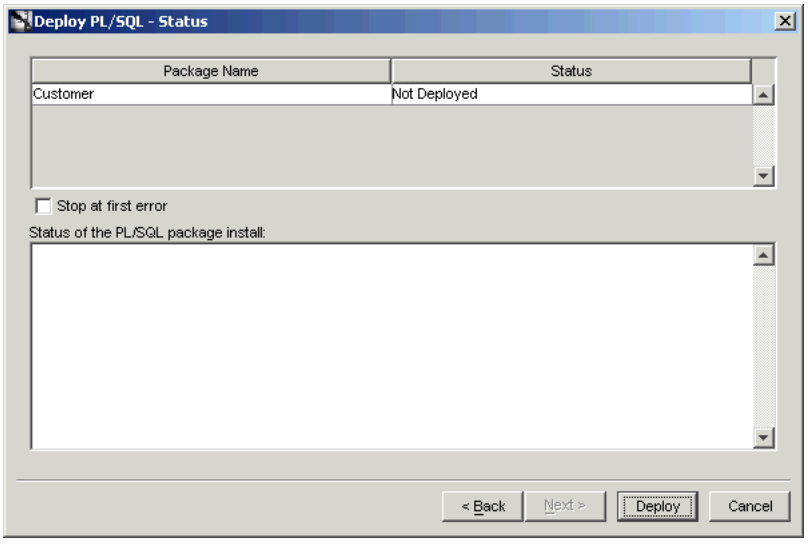
Previously Deployed, but package specification invalid

Previously Deployed, but package body invalid

Previously Deployed, but package specification and body invalid

---

8. Click **Next**. The Deploy PL/SQL - Status dialog box is displayed.



The Status dialog box displays:

- A non-editable table: A table that displays the packages of the corresponding application, event, or procedure selected for deployment, and the current status of the package.
- A Stop At First Error checkbox: If this option is checked and multiple packages are being deployed, then the deployment process will stop after encountering the first error. You will be prompted whether you want to continue with the deployment of the remaining packages or not.

9. Click **Deploy**. The generated PL/SQL is deployed for the selected application, event, or procedure.

   After the deployment is complete, the status of the PL/SQL deployment process is displayed. If an exception is thrown during the deployment process, then the exception is displayed. After deploying each package, the Status column is updated with success or error messages. The messages are:

   - Deployed successfully: The PL/SQL package was deployed successfully.
   - Deployed, but package specification invalid: The PL/SQL package was deployed. However, the status of the package specification in the database is invalid.

- Deployed, but package body invalid: The PL/SQL package was deployed. However, the status of the package body in the database is invalid.
- Deployed, but package specification and body invalid: The PL/SQL package was deployed. However, the status of both, the package specification and body in the database, is invalid.
- Failed to Deploy: The PL/SQL package could not be deployed in the database.
- Package status unknown: An error occurred while deploying the PL/SQL package in the database.

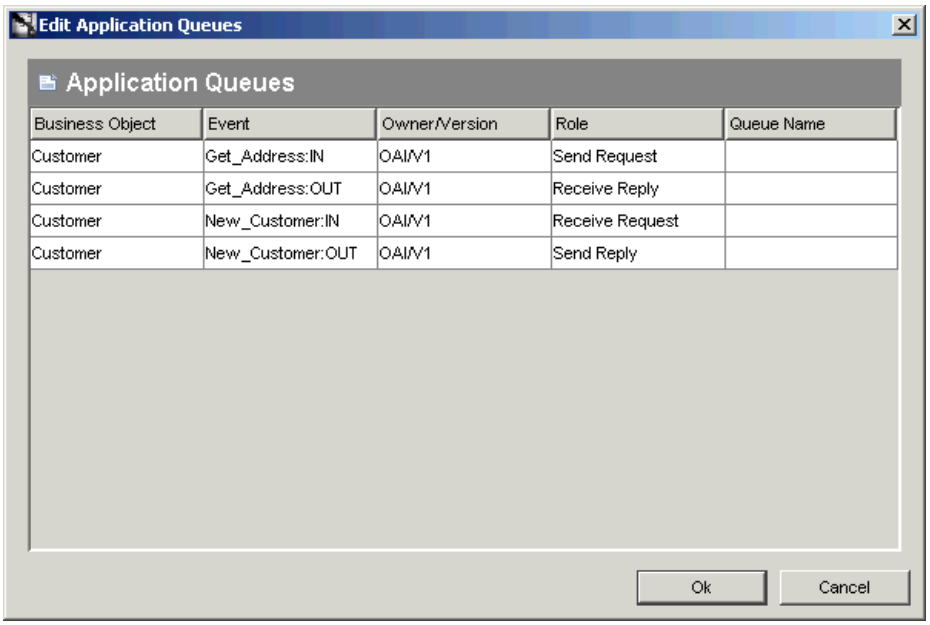
8.2 Specifying Application Queue Names for Advanced Queuing Adapter

If an application is configured to use the Advanced Queuing adapter (AQ adapter), then the user must specify which queues will be used to send and receive data to/from the application. These queues are called application queues.

Note: Application queues are not the same as the hub queues. The hub queue names come preconfigured, and all adapters use the hub queues to communicate with each other. Application queues, on the other hand, are used only by applications that expose data to the adapter using AQ. Application queues are peculiar to an AQ adapter enabled spoke system, and the queue names must be specified at deployment time so that the adapter can communicate to the application.

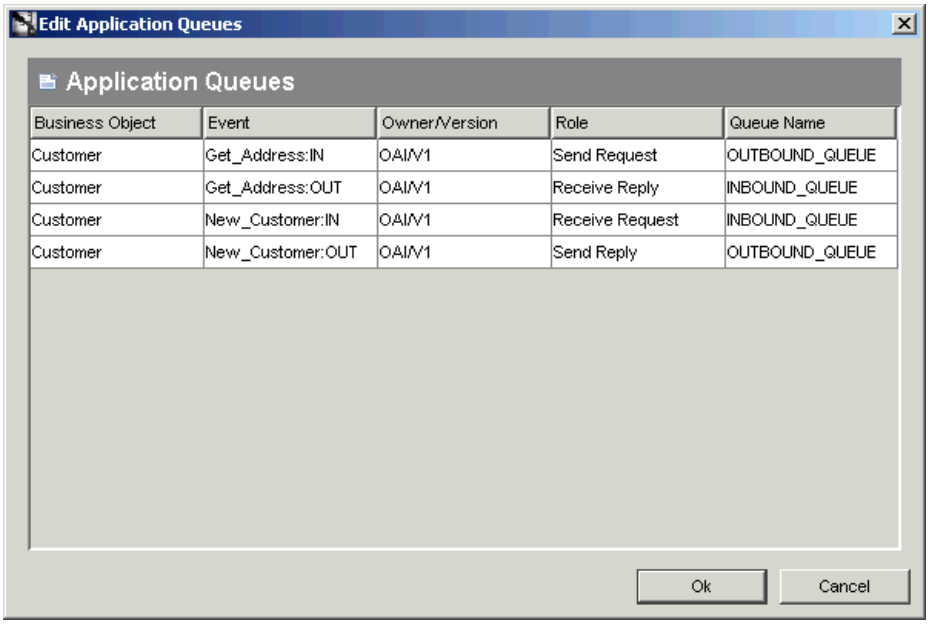
The following steps describe this task for an application with name `AQAPP`.

1. On the Deploy tab in iStudio, expand the **Applications** list and navigate to AQAPP application.
2. Expand the **AQAPP** node and navigate to the Routing node.
3. Expand the **Routing** node and select **Application Queues**. The Application Queues property sheet is displayed on the right side of the iStudio window.
4. Select **Edit** from the Edit menu. The Edit Application Queues dialog box is displayed.



5. Add the application Queue name to each event.

Queue Name	Event
INBOUND_QUEUE	PO_Cancel
OUTBOUND_QUEUE	PO_Insert, PO_Update, and PO_Delete



6. Click **OK**.

8.3 Deploying Workflow Events and Process Definitions

After business processes have been populated with activities, you must deploy the information to Oracle Workflow and then graphically model a business process.

See Also: Section 7.4.7, "Deploying to Oracle Workflow"

8.4 Sync Adapters from iStudio

Adapters can be configured to cache metadata locally to minimize communication with the repository at run time. If so configured, these adapters are not aware of changes made through iStudio after they have cached the metadata. You can refresh the adapter local cache with the new repository metadata by using the **Sync Adapters** option. The following steps describe this task:

- Select **File** from the menu bar, then **Sync Adapters**. The Sync Adapters dialog box is displayed.
- Select the applications and click **OK**.

9

Run-Time System Concepts and Components

This chapter describes the run-time concepts of OracleAS Integration InterConnect. It contains the following topics:

- Integration Architecture
- Components
- Run-Time System Features
- Real Application Clusters Configuration

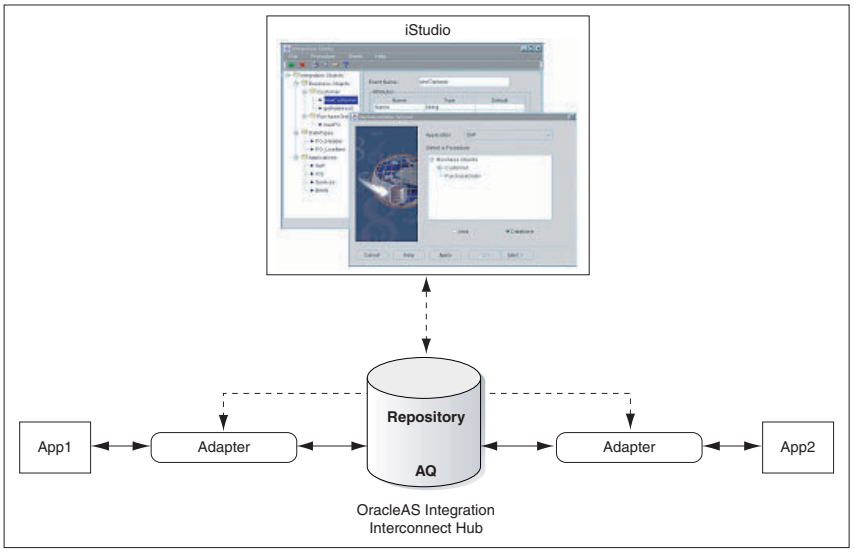
9.1 Integration Architecture

OracleAS Integration InterConnect run-time system is an event-based distributed messaging system. An event is any action that starts the communication through messaging between two or more applications integrated through OracleAS Integration InterConnect. The messaging system can be deployed both within an enterprise or across enterprise boundaries.

The run-time system enables inter-application communication through hub-and-spoke integration. This methodology keeps the applications decoupled from each other by integrating them to a central hub rather than to each other directly. The applications are at the spokes of this arrangement and are unaware of the other applications they are integrating with. To them, the target of a message (or the source) is the hub. As each application integrates with the hub, transformation of data between the application and hub (in either direction) is sufficient to integrate two or more applications.

Figure 9–1 provides an overview of design-time and run-time phases in integration.

Figure 9–1 A Graphical Overview of Design-Time and Run-Time Phases in Integration



9.2 Components

The following are the main components in the run-time system:

- Adapters
- Repository
- Advanced Queues
- Oracle Workflow

9.2.1 Adapters

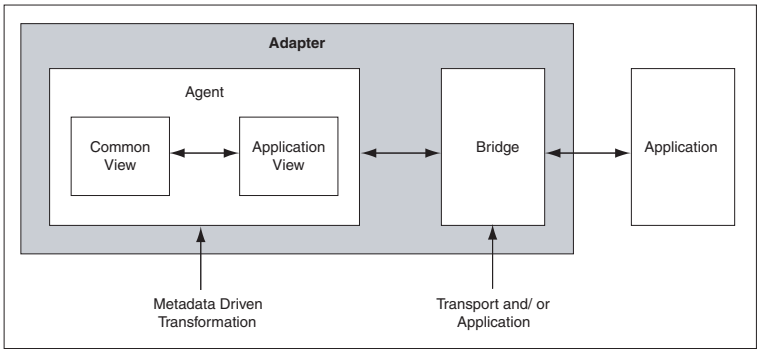
Prepackaged adapters help applications at run time to participate in the integration without any programming effort.

9.2.1.1 Agent and Bridge Combination

Adapters are the run-time component for OracleAS Integration InterConnect. Adapters have the following features:

- Application Connectivity: Connect to applications to transfer data between the application and OracleAS Integration InterConnect. The logical subcomponent within an adapter that handles this connectivity is called a bridge. This protocol/application-specific subcomponent of the adapter knows how to communicate with the application. For example, the database adapter is capable of connecting to an Oracle database using JDBC and calling SQL APIs. This subcomponent does not know which APIs to call but only how to call them.
- Transformations: Transform data to and from the application view to common view as dictated by the repository metadata. In general, adapters are responsible for carrying out all the run-time instructions captured through iStudio as metadata in the repository. Transformations are an important subset of these instructions. The logical subcomponent within an adapter that handles the run-time instructions is called an agent. This is the generic run-time engine in the adapter that is independent of the application to which the adapter connects. It focuses on the integration scenario based on the integration metadata in the repository. There

is no integration logic coded into the adapter itself. All integration logic is stored in the repository. The repository contains the metadata that drives this subcomponent. For example, in a database adapter, the agent subcomponent knows which SQL APIs to call but not how to call them. All adapters have the same agent code. It is the difference in metadata that each adapter receives from the repository that controls and differentiates each adapter.



Adapters can be configured to cache the metadata at run time to address performance needs. There are three settings for caching metadata:

- No Caching: For each message, the adapter will query the repository for metadata. This setting is recommended for an early or unstable integration development environment.
- Demand Caching: The adapter will query the repository only once for each message type and then cache that information. For subsequent messages of the same type, it will use the information from the cache. This setting is recommended for a stable integration development environment.
- Full Caching: At startup time, the adapter will cache all its relevant metadata. This setting is recommended for a production environment.

> **Note:** For more information about the adapters provided by OracleAS Integration InterConnect, refer to *Oracle Application Server InterConnect Installation Guide*.

Adapters are stateless by default. As a result, in case an adapter is not working, the message is either with the application or in the OracleAS Integration InterConnect Hub.

9.2.2 Repository

The repository consists of two components:

- Repository Server: A Java application that runs outside the database. It provides RMI services to create, modify, or delete metadata at design time using iStudio and query during run time using adapters. Both adapters and iStudio act as RMI clients to communicate with the repository server.
- Repository Database: The repository server stores metadata in database tables. The server communicates to the database using JDBC.

Adapters have the ability to cache metadata. If the repository metadata is modified after adapters have cached metadata, then the relevant adapters can be notified through iStudio's Sync Adapters functionality.

Note: You need to specify a port number for the repository in the `repo_admin_port` parameter of the `repository.ini` file located at `$ORACLE_HOME/integration/interconnect/repository`. You also need to specify the port number in the `agent_admin_port` parameter of the `adapter.ini` file of an adapter located at `$ORACLE_HOME/integration/interconnect/adapters/ADAPTERNAME`. The port number in the `repository.ini` file and the `adapter.ini` file should be the same.

In addition, you need to open the seven ports on the firewall starting from the port number specified in `repo_admin_port` parameter of the `repository.ini` file.

See Also: Section 8.4, "Sync Adapters from iStudio"

9.2.3 Advanced Queues

Advanced Queues provide the messaging infrastructure for OracleAS Integration InterConnect in the hub. In addition to being the store and forward unit, Advanced Queues provide message retention, auditing, tracking, and guaranteed delivery of messages.

See Also: *Oracle Database Application Developer's Guide* for information about Advanced Queues

9.2.4 Oracle Workflow

Oracle Workflow facilitates integration at the business process level through its Business Event System. OracleAS Integration InterConnect and Oracle Workflow are integrated to leverage this facility for business process collaborations across applications.

9.3 Run-Time System Features

The OracleAS Integration InterConnect run-time features are as follows:

- Messaging Paradigms
- Message Delivery
- Message Retention
- Routing Support
- Load Balancing
- High Availability
- Backup and Recovery

9.3.1 Messaging Paradigms

The OracleAS Integration InterConnect run time supports three major messaging paradigms:

- Publish/Subscribe
- Request/Reply (synchronous and asynchronous)

- Point-to-Point

Point-to-Point messaging can be achieved both in the context of Publish/Subscribe and Request/Reply by using content-based routing.

Applications can be configured (for each integration point) to support any of these paradigms.

> **See Also:** Chapter 1, "Getting Started with OracleAS Integration InterConnect"

9.3.2 Message Delivery

The following are the features of message delivery:

- Guaranteed Delivery: All messages are guaranteed to be delivered from the source applications to the destination applications.
- Exactly Once Delivery: The destination applications will receive each sent message exactly once. The messages are never lost or duplicated.
- In Order Delivery: The messages are delivered in the exact same order as they were sent. This is applicable only when there is one instance of the adapter running for each serviced application.

9.3.3 Message Retention

Messages remain in the run-time system until they are delivered. Advanced Queues in the hub provide message retention. Messages are deleted when each application that is scheduled to receive a specific message has received that message. For auditing purposes, you can configure the system to retain all successfully delivered messages.

9.3.4 Routing Support

Routing is a function of the Advanced Queues in the hub. By default, `oai_hub_queue` is the only multiconsumer Advanced Queue configured as the persistent store for all messages for all applications. This queue will handle all standard as well as content-based routing needs. The queue is created automatically when you install the repository in the hub. The only reason to change this configuration is if Advanced Queues becomes a performance bottleneck. This is unlikely because most of the message processing is done in the adapters, not in the hub.

> **See Also:** "Load Balancing" on page 9-5

9.3.4.1 Content-Based Routing

Content-based routing enables you to route messages to specific destination applications based on message content. For example, an electronic funds transaction settlement application is designed to transmit bank transactions with a specific bank code to identify the destination bank system. When the electronic funds transfer application publishes a message at run time, the OracleAS Integration InterConnect run-time component determines the bank code value based on metadata stored in the repository, and routes the message to the corresponding recipient system.

9.3.5 Load Balancing

OracleAS Integration InterConnect provides the following two methods to manage load balancing across different instances of the same adapter:

- Partitions
- Instances

OracleAS Integration InterConnect addresses adapter scalability through a well-defined methodology.

9.3.5.1 Partitions

OracleAS Integration InterConnect uses partitioning to manage load balancing across different instances of the same adapter. At run time, it is possible that the adapter attached to a particular application becomes a performance bottleneck. You can detect this by monitoring the message throughput information using the InterConnect Manager.

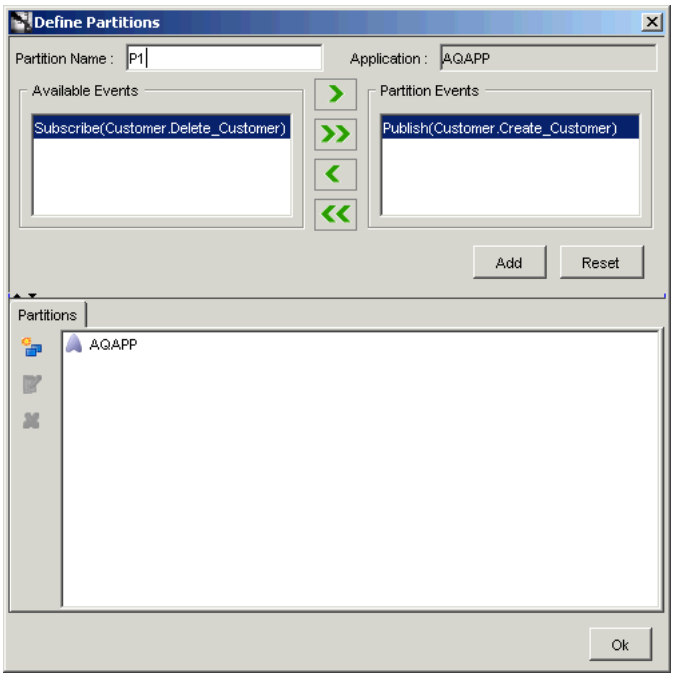
You can create multiple partitions of an application to enable an adapter to share the message load at run time. Partitions are always bound to specific events or procedures. We can create partitions on one or more events and procedures.

For example, application `AQAPP` is publishing the `Create_Customer` event and subscribing to the `Delete_Customer` event. You can create two partitions `P1` and `P2` on `AQAPP`. You can specify that partition `P1` handles all the messages from the `Create_Customer` event and partition `P2` handles all the messages from the `Delete_Customer` event. In partitioning, the messages are processed in same order in which the publishing application sends the messages. If there is only one instance of each partition, then the order of the messages is maintained on per-event or per-procedure basis. Perform the following steps to create multiple partitions of an application:

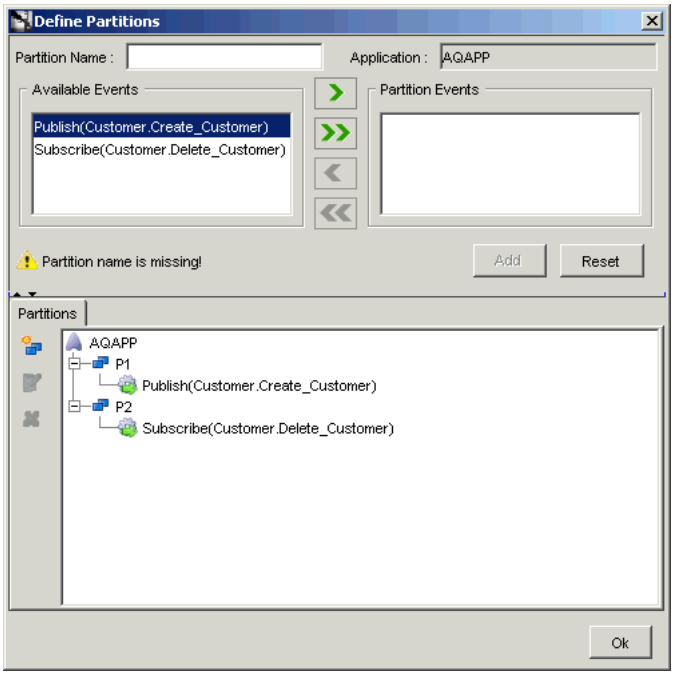
Design Time:

Perform the following design time steps in iStudio to create a partition:

1. Click the **Deploy** tab.
2. Click **Applications**, *`ApplicationName`*, and then click **Routing**.
3. Right-click **Message Capability Matrix** and select **Create Partition**.
4. Enter the name of the partition in **Partition Name** field.



5. In the **Available Events** list, select each of the events that you want to include.



6. Click the right-arrow button and then click **Add**.

Run Time

Perform the following run-time steps to create a partition:

1. Use the `copyAdapter` script to make a copy of the existing adapter in the same Oracle home. Enter the following command:

```
c:\> cd ORACLE_HOME\integration\interconnect\bin
c:\> copyAdapter AQAPP AQAPPTEMP
```

2. Edit the adapter.ini file of each adapter located in *ORACLE_HOME*/integration/interconnect/adapters/*ADAPTERNAME* directory.
   - Change the application parameter value to the old application name. For example, AQAPPTEMP to AQAPP.
   - Delete the comment tags before the partition parameter and specify the partition name.
   - Specify the application name followed by the partition name in the agent_subscriber_name parameter. For example, AQAPPP1.
   - Specify the application name followed by the partition name in the agent_message_selecter parameter.

   Note: The name of the application in the adapter.ini file and iStudio should be same.

9.3.5.2 Instances

You can create multiple instances of an adapter to share the message load. Instances are always bound to an application. If an adapter is receiving a large number of messages and is not able to process these messages efficiently, then it is recommended that you create multiple instances of the adapter. Messages are shared between the multiple instances. However, the messages are not processed in the same order in which publishing application sends the messages. Therefore, the in-order message delivery is lost.

In partitioning, the adapter subscribes to the hub queue with application_name+partition_name whereas in instances, the adapter subscribes to the hub queue with only application_name. Therefore, all the instances of an adapter that correspond to an application can listen to all the messages for that application. AQ handles delivering the message to only one of them. For example, DB1 and DB2 are two instances of the Database adapter and both instances are bound to an application DBAPP. When publishing a message, both instances publish the messages with the publisher name DBAPP. The adapter instance that reads the message first will publish the message and delete it from the OAI schema. This ensures that the duplicate messages are not published. Similarly, when subscribing to a message, both adapter instances subscribe to the hub queue with the same application name. Although both can listen to all the messages meant for DBAPP, a message will be delivered to either DB1 or DB2.

Perform the following steps to create multiple instances of an adapter:

1. Use the copyAdapter script to make multiple copies of the existing adapter in the same Oracle home.

```
c:\> cd ORACLE_HOME\integration\interconnect\bin
c:\> copyAdapter copyAdapter AQAPP AQAPPTEMP
```

2. Edit the adapter.ini file of the each newly created adapter. The adapter.ini file is located in the *ORACLE_HOME*/integration/interconnect/adapters/*ADAPTERNAME* directory.
   a. Change the value of the application parameter to the old application name, for example AQAPPTEMP to AQAPP.
   b. Delete the comment tags before the instance_number parameter.

c. Specify the instance number in the `instance_number` parameter.

9.3.5.3 Load Balancing with Multiple Adapters

Multiple adapters can be attached to one application to share the message load. This can be done in several ways depending upon the needs of your integration environment. For example, Application A publishes three different kinds of events: EventA, EventB, and EventC. Three potential scenarios should be examined to determine how one or more adapters could be attached to the application to meet performance objectives.

Scenario 1

The order in which the messages are sent by application A must be strictly adhered to for the life of the messages. Messages sent by application A must be received by the subscribing applications in the same order across the different event types.

Recommendation In this case, you cannot add more than one instance of the same adapter to Application A for load balancing.

Scenario 2

The order in which messages are sent by Application A must be adhered to but not across different event types. Application A publishes the following messages in order: `M1_EventA`, `M2_EventB`, `M3_EventA`. `M1_EventA`, and `M3_EventA` must be ordered with respect to each other because they correspond to the same event type. `M2_EventB` has no ordering restrictions with respect to `M1_EventA` and `M3_EventA`.

Recommendation In this case, you can leverage the partitioning feature enabled through iStudio's Deploy tab. This feature enables you to allocate specific adapters for specific message types, thereby segmenting the run time load processing. For this scenario, you can create two partitions: `Partition1` corresponds to `EventA` and `Partition2` corresponds to `EventB`. Dedicate one adapter to each partition (specified at adapter install time or through modification of `adapter.ini` after install). The end result: The order of messages is maintained and the processing power has doubled because of two adapter servicing the messages instead of just one. This kind of partitioning is called Message-based partitioning.

Scenario 3

There is no message order dependency, even within the same event type.

Recommendation Two approaches for load balancing are available:

1. One or more adapters are added using the entire message capability matrix. This means that at run time, any one of the adapters would be available to receive any message, though only one of them would actually receive the message. The adapter that is first to request the next message for processing will determine the adapter that will receive the message. This is called Pure Load Balancing partitioning.
2. Message-based partitions are created based on projections of the number of messages for a particular event type. For example, if there will be three times as many `EventA` messages than `EventB` or `EventC` messages, then you could create two partitions, one for handling `EventA` messages, and the other for handling the other two event types. Now, you can dedicate several adapters to handle the EventA message load only. Fewer adapters can be dedicated to the other two event types.

9.3.6 High Availability

Enterprise applications need high availability (HA) because they cannot afford downtime. OracleAS Integration InterConnect uses Oracle Process Manager and Notification (OPMN), Oracle Database Server, and Oracle Real Application Clusters to enable high availability for its components.

See Also: *Oracle Application Server High Availability Guide*

9.3.7 Backup and Recovery

The OracleAS Backup and Recovery feature can be used to back up the critical configuration files for any OracleAS Integration InterConnect 10*g* Release 2 (10.1.2) installation. You can use the `config_misc_files.inp` file provided by the OracleAS Backup and Recovery tool to back up InterConnect configuration files. The `config_misc_files.inp` file is located in the following directory:

```
$ORACLE_HOME/backup_restore/config
```

See Also: *Oracle Application Server Administrator's Guide*

The following files should be backed up from the OracleAS Integration InterConnect install along with other Application Server component files.

[Hub Component]

```
$ORACLE_HOME/integration/interconnect/hub/hub.ini
$ORACLE_HOME/integration/interconnect/repository/repository.ini
$ORACLE_HOME/integration/interconnect/security/cwallet.sso
$ORACLE_HOME/integration/interconnect/security/ewallet.p12
$ORACLE_HOME/integration/interconnect/adapters/workflow/adapter.ini
$ORACLE_HOME/integration/interconnect/adapters/workflow/ErrorManagement.xml [if
file exists]
```

[Adapter Component]

```
$ORACLE_HOME/integration/interconnect/adapters/<adaptername>/adapter.ini
$ORACLE_HOME/integration/interconnect/adapters/<adaptername>/ErrorManagement.xml
[if file exists]
$ORACLE_HOME/integration/interconnect/security/cwallet.sso [if adapter not
installed in the same midtier as hub]
$ORACLE_HOME/integration/interconnect/security/ewallet.p12 [if adapter not
installed in the same midtier as hub]
```

You can append the previously mentioned Hub Components and Adapter Components OracleAS Integration InterConnect configuration file names to the `config_misc_files.inp` file with the same file name format.

If all files in a directory have to be backed up, then you can specify only the directory names or use wildcards. You can also exclude certain files from the backup by specifying those file names in the `config_exclude_files.inp` file. However, you cannot specify directories or use wildcards in the `config_exclude_files.inp` file; only single entries are allowed.

9.4 Real Application Clusters Configuration

In Real Application Clusters environment, all active instances can concurrently perform transactions against a shared database. Real Application Clusters coordinates each instance's access to the shared data to provide data consistency and data integrity.

It features balanced workloads among the nodes by controlling multiple server connections during periods of heavy use and provide persistent, fault-tolerant connections between clients and the Real Application Clusters database.

> **See Also:** *Oracle Application Server Concepts* Guide for information on Real Application Clusters.

9.4.1 OracleAS Integration InterConnect Adapters Supporting Real Application Clusters

OracleAS Integration InterConnect adapters leverage Real Application Clusters technology, provide consistent and uninterrupted service without having to restart the adapters, if an instance fails, and provide guaranteed message delivery. OracleAS Integration InterConnect adapters connect to the first of the listed available nodes. Nodes are defined in the `adapter.ini` and `hub.ini` files.

> **See Also:** OracleAS Integration InterConnect adapters installation documentation for information about `adapter.ini` and `hub.ini` files associated with specific adapters

If one node fails, then the database connection is established with the next available node in the `adapter.ini` or `hub.ini` file recursively until a successful connection is established. Failover is transparent to the user.

The hub connections for all adapters and the spoke connections for Database and Advance Queuing adapters are RAC enabled. The adapter process is also RAC enabled.

9.4.1.1 Adapter Failover Mechanism

In the earlier OracleAS Integration InterConnect releases, the adapters failed over to the next node in the Real Application Clusters environment for any exception. This release changes the adapter failover mechanism. The adapters are designed to fail over only when the corresponding node fails. This means that a normal exception will not cause a failover to be triggered. Instead, the adapter will fail over only when the node itself fails.

9.4.2 Configuration

The `adapter.ini` and `hub.ini` files must be populated with the information about the host, port, and instance for all the nodes. Additional sets of parameters which specify the number of nodes are also required to be populated. All existing entries remain the same, except that a new entry for each node is added. Table 9–1 describes the additional sets of parameters which specify the number of nodes required to be populated.

Table 9–1 Additional Parameters for RAC Configuration

File Name	Parameter
hub.ini	■ host_num_nodes: The number of nodes in a cluster. For example: hub_num_nodes=2 ■ hub_hostx: The host where the Real Application Clusters database is installed. For example: hub_host2=dscott13 ■ hub_portx: The port where the TNS listener is listening.For example: hub_port2=1521 ■ hub_instancex: The instance on the respective node where x varies between 2 and the number of nodes. For example: hub_instance2=orcl2
adapter.ini for the Advanced Queuing adapter	ab_bridge_num_nodes aq_bridge_host aq_bridge_port aq_bridge_instance
adapter.ini for the Database adapter	db_bridge_num_nodes db_bridge_schema1_hostx db_bridge_schema1_portx db_bridge_schema1_instancex: where x is a value between 2 and the number of nodes.

Sample hub.ini File

The following is a sample hub.ini file.

```
hub_username=ichub
encrypted_hub_password=<encrypted_password> use $ORACLE_
HOME/integration/<version>/bin/encrypt for encryption
hub_use_thin_jdbc=true
hub_host=dlsun1312
hub_instance=iasdb
hub_port=1521
hub_num_nodes=2
hub_host2=vindaloo
hub_instance2=orcl
hub_port2=1521
```

9.4.3 Sample Database Adapter adapter.ini File with the Spoke Database Entry

The following is a sample adapter.ini file for the Database adapter that shows the spoke database entry.

```
db_bridge_schema1_host=dlsun1312
db_bridge_schema1_port=1521
db_bridge_schema1_instance=iasdb
db_bridge_num_nodes=2
db_bridge_schema1_host2=vindaloo
```

```
db_bridge_schema1_port2=1521
db_bridge_schema1_instance2=orcl
```

10

XML Schema Definition

This chapter provides an overview of XML Schema Definition (XSD) and describes how InterConnect supports XSD. It contains the following topics:

10.1 Overview of XSD

An XSD defines the structure of an XML document. It specifies the elements and attributes that can appear in an XML document and the type of data these elements and attributes can contain. This information is used to verify that each element or attribute in an XML document adheres to its description.

10.1.1 Advantages of XSD over DTD

Following are the advantages of XSD over Document Type Definition (DTD):

- XSD is extensible. You can derive new elements from the existing elements. DTD is not extensible.
- XSD is defined in XML. It does not require intermediate processing by a parser. DTD is not defined in XML. You need separate parsers for DTD and XML.
- XSD supports data types. You can restrict the content of an element. DTD does not support data types. Therefore, you cannot restrict the content of an element.
- XSD supports default values. You can specify default values of the elements. You cannot specify the default values of elements in DTD.
- XSD supports references to external XML schemas. You can include or import more than one XML schema within an XML schema. You cannot include other DTDs within a DTD.

10.2 XSD Elements and Attributes

An XML schema defines elements and their structures. It also defines the attributes and their data types. The elements' structures can be of `simpleType` or `complexType`, depending on whether the element is a leaf element or a parent element.

> **See Also:** "XSD Type Definitions" for more information on `simpleType` and `complexType`

10.2.1 XSD Elements

XSD elements can be of type `simpleType`, `complexType`, or `anyType`. An element of type `simpleType` contains only text. It cannot have attributes and elements. An element of type `complexType` can contain text, elements, and attributes. An element of type `complexType` is parent to all the elements and attributes contained within it. An `any` element in an XSD specifies that any well-formed XML is allowed in its place in XML instance.

InterConnect stores the type `simpleType` as a member of the object created for the parent element. The data type of this member is the InterConnect data type corresponding to the XSD data type. For example, `Address1` is a `complexType` that contains two `simpleType` elements, `City` and `State`. `myAddress` is an element of `Address1 complexType`. InterConnect stores `Address1` as an object, and `City` and `State` as InterConnect data type String. `myaddress` is stored as InterConnect data type `Reference` that contains reference to object of `complexType`, `Address1`. InterConnect stores the name of `any` element in the `_any_` format. At run time, InterConnect treats an `any` element as string.

Elements can be local or global. An elements declared as the direct child of an XML schema is called a global element and can be used throughout the schema. Elements declared within a complex type definition are local elements. So, you cannot use these elements anywhere else in the schema. When you import an XSD, InterConnect displays all the global elements of this XML schema as well as all the imported or included XML schemas. These global elements are grouped together based on their namespaces. You can browse through the namespaces and select a global element. This global element acts as root of the message when defining an application view or common view. It also acts as root of the type generated when defining application data types or common data types.

When you select an element as the root element, all the elements starting from the structure of the selected element untill the end are stored in the repository. An element can have constraints such as `minOccurs` and `maxOccurs`. If the value of the `maxOccurs` constraint of an element is greater than one, then the element is stored as an array.

10.2.2 XSD Attributes

XSD attributes are always of type `simpleType`. The syntax for defining an attribute is:

```
<xs:attribute name="myattribute" type="string"/>
```

In this example, `myattribute` is an attribute of data type `string`. An attribute can have a fixed or default value. InterConnect stores the fixed or default value of an attribute. Attributes are stored as member of the object created for the parent element. The data type of this attribute is the InterConnect data type corresponding to the XSD base data type.

An attribute group defines an association between a name and a set of attribute declarations. You can reuse the attribute groups in `complexType` definitions. The syntax for defining an attribute group is:

```
<xsd:attributeGroup name="Address">
<xsd:attribute name="Street1" type="xs:string"/>
<xsd:attribute name="Street2" type="xs:string"/>
<xsd:attribute name="City" type="xs:string"/>
<xsd:attribute name="State" type="xs:string"/>
</xsd:attributeGroup>
```

In this example, `Address` is the name of the attribute group that contains attributes such as `Street1`, `Street2`, `City`, and `State`. InterConnect resolves the attribute group references by copying the attributes definition to the referencing `complexType`.

anyAttribute

Like the `any` element, `anyAttribute` specifies that any attribute can be used in its place in the XML instance. InterConnect stores `anyAttribute` as an attribute with its name in the format `_anyAttribute_`. The type of this attribute is stored as String.

10.2.3 Nillable

Sometimes, you need to represent unknown information, or inapplicable information, explicitly with an element. For example, you might need to represent a null value being sent to or from a relational database with an element. In such cases, you can declare the `nillable` attribute of an element as `true` in its XML schema definition. InterConnect stores the `nillable` attribute value in the repository while importing the XSD.

10.3 XSD Type Definitions

XSD Type definitions are used to create new `simpleType` data type or `complexType` data type. A type definition that is used as a base for creating new definitions is known as the base type definition. A `simpleType` or `complexType` type can be either named or anonymous. A named `simpleType` or `complexType` is always defined globally. You can define a named `simpleType` or `complexType` and refer it in schema document as type of an element. An anonymous `simpleType` or `complexType` type does not have a name. So, an anonymous `simpleType` or `complexType` type cannot be referenced.

10.3.1 simpleType

XML schema has several built-in data types. A `simpleType` type is derived from an XML schema built-in data type. For example, to create a new `simpleType` type called `myInt` that has range of values between 10000 and 99999, you can base the definition of `myInt` on the built-in data type `Integer`.

Each XML schema built-in data type is mapped to an InterConnect data type. When you import an XSD that has an element of type `simpleType`, InterConnect identifies to which InterConnect data type this element should be mapped. InterConnect also stores the original XSD data type of an element. For example, if you import an XSD that contains an element of XSD data type `dateTime`, then it will be stored as InterConnect data type `Date`. In addition, InterConnect also stores the `dateTime` data type, which is the XSD data type of the element, and its format in the repository. This pattern is used at run time for converting the `Date` object to date strings.

Table 10–1 shows how various XSD data types are mapped to InterConnect data types in InterConnect.

Table 10–1 Mapping Between XSD Data Types and InterConnect Data Types

XSD Data Type	InterConnect Data Type
xsd:integer, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, xsd:positiveInteger	Integer
xsd:double, xsd:decimal	Double
xsd:float	Float
xsd:string, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:ID, xsd:IDREF, xsd:ENTITY, xsd:anyURI, xsd:QName, xsd:NOTATION, xsd:NMTOKENS, xsd:IDREFS, xsd:ENTITIES, xsd:duration, xsd:time, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth	String
xsd:binary and xsd:base64binary	Binary
xsd:date, xsd:datetime	Date
xsd:boolean	Boolean

10.3.2 complexType

A `complexType` type definition contains a set of element declarations, element references, and attribute declarations.You first define a `complexType` type in schema and then, you define the elements and attributes of this `complexType` type.

InterConnect stores a `complexType` type as an object. A `complexType` type can be either named or anonymous. When an imported XML schema has anonymous `complexType` type, then InterConnect generates its object name based on the name of the element. The type of this object is `anonymous complexType`. For example, when you import an XSD that contains an element `MyElement` of type `anonymous complexType`, InterConnect generates and stores the object for the element with a typename such as `MyElement_CT`.

10.3.3 anyType

All the `complexTypes` and `simpleTypes` data types are derived from `anyType`. So, `anyType` is the base type of all `simpleTypes` and `complexTypes` data types. You can create an element of type `anyType` by not specifying its data type in the

definition. You cannot restrict the content of an anyType element. When you import an XSD that contains an element of type anyType, then InterConnect stores this anyType as String. An element of type anyType can be mapped only to an element of type anyType while doing transformations.

10.3.4 anySimpleType

All the simpleTypes data types of XML schema are derived from anySimpleType. So, anySimpleType is the base type of all simpleTypes data types. While importing an XSD, if an attribute does not have its type defined, then InterConnect treats the attribute type as anySimpleType. At run time, an anySimpleType element is treated as String. An anySimpleType element is mapped only to an element or attribute of type anySimpleType during transformation.

10.4 XSD Content Models

You can define and name a group of elements in XSD. You can use XSD content models to define the order of elements in a group. When you import a content model, InterConnect automatically detects and resolves the redundancy of the model groups. For example, if a sequence model group with maxOccurs=1 directly contains another sequence model group with maxOccurs=1, then InterConnect automatically detects that the second model group is redundant.

10.4.1 Choice Group

The choice group describes a choice between several possible elements of group. The choice group specifies that only one element can appear in an XML instance. For example, Address1 is a complexType data type that contains two elements, OfficeAddress and ResidenceAddress. You can create a choice group if you want either OfficeAddress or ResidenceAddress to appear in the XML document.

A choice group can contain other groups as well as elements. InterConnect creates an element for a choice group. This element contains all the members of the choice group and is stored as an object in InterConnect having the name _Choice_. If the value of maxOccurs for a choice group is greater than one, then the object of this choice group is created as an array in InterConnect.

10.4.2 Sequence Group

The sequence group specifies that all the elements defined in this group must appear once in the XML document. These elements must appear in a specific order. For example, Address1 is a complexType type that contains two elements OfficeAddress and ResidenceAddress. You can create a sequence group if you want OfficeAddress to appear before ResidenceAddress in XML document.

A sequence group can contain other groups as well as elements. InterConnect creates an element for a sequence group. This element contains all the members of the sequence group and is stored as as object with its name in the format _Sequence_. If the value of maxOccurs for a sequence group is greater than one, then the object of this sequence group is created as an array in InterConnect.

10.4.3 All group

The All group specifies that all the elements defined in this group must appear once in the XML document. These elements can appear in any order in the XML document.

The All group cannot contain other groups. InterConnect creates a element for the All group. This element contains all the members of the All group and is stored as an object in InterConnect having the name `_All_`. If the value of `maxOccurs` is greater than one, then the object of this All group is created as an array in InterConnect.

10.5 XSD Namespace

XSD namespace is a collection of element and attribute names and their types. The namespace is identified by a unique name, which is a Uniform Resource Identifier (URI). Namespaces are associated with a prefix when they are declared, and this prefix is used along with a local name to represent an element in an XML document. For example, `Book:Title` where `Book` is the namespace prefix and `Title` is name of the element. XSD namespace provides a way to distinguish between duplicate elements, attribute names and their types. When you import an XSD, InterConnect displays all elements grouped by namespace names.

Target namespace defined in a schema specifies the namespace where all the global elements and attributes defined in schema reside. When an element in XSD does not have a named namespace, it means that this element belongs to the `NoNamespace` namespace.

10.5.1 Include and Import Schema

You can divide an XSD into parts so that it is easier to maintain and read. You can use the `xsd:include` and `xsd:import` statements to use the content of one XSD file in another. The `include` statement is for schemas that exist in the same target namespace or do not have a defined target namespace. The `import` statement is for schemas that exist in another target namespace.

When you import an XSD, InterConnect automatically takes care of all the included or imported XSDs. InterConnect first resolves the schema that is being imported. All references to the types and elements of the schema being imported or included are replaced by their definitions.

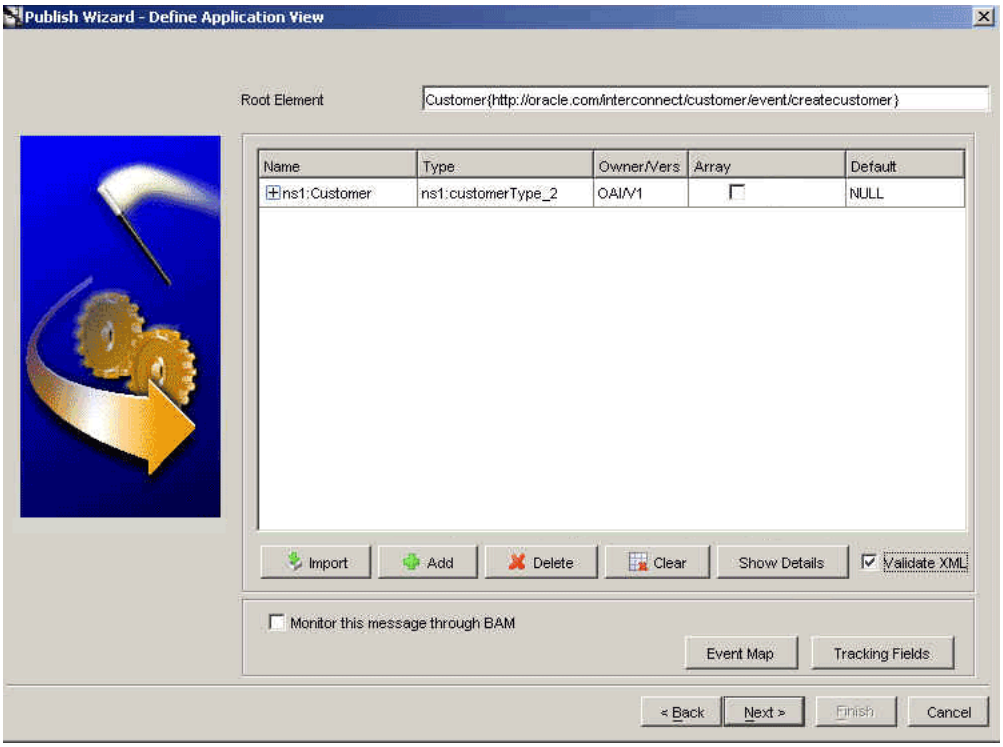
10.5.2 InterConnect Validation

InterConnect stores the selected XSD and all included and imported XSD as Binary Large Objects (BLOB) in the InterConnect repository. This is done to validate XML instances against the actual XSDs at run time. Following are the three points at which a message is validated during its flow in InterConnect:

1. When InterConnect receives a message from an application:

   This validation is performed against the publishing application view XSD. Figure 10–1 displays the **Validate XML** option as selected. This indicates that the XML instance will be validated against application view XSD.

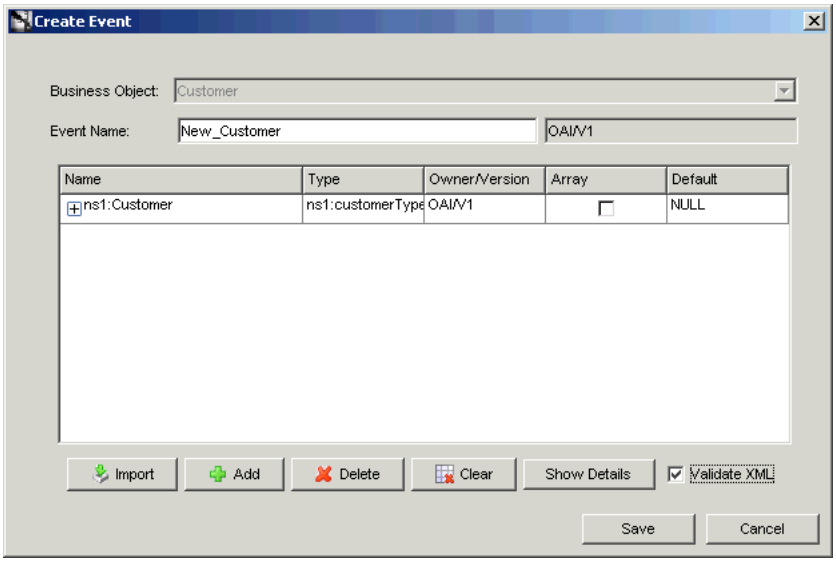
Figure 10–1 Enabling Validation for an Application View



2. After InterConnect transforms a message from application view to common view:

   This validation is performed against the common view XSD. Figure 10–2 displays the **Validate XML** option as selected. This indicates that the XMl instance will be validated against common view XSD.

Figure 10–2 Enabling Validation for a Common View



3. After InterConnect transforms a message from common view to application view: This validation is performed against the subscribing application view XSD.

Note: XML instances are validated against the actual XSDs only when you select the validate option. You can activate or turn off these points of validation during design time in iStudio.

10.6 XSD Type Derivations

You can derive new `complexTypes` data types by extending or restricting existing `simpleTypes` or `complexType` data types. When you import an XSD, InterConnect creates elements for the base and derived types.

10.6.1 Deriving complexTypes by Extension

Extension means taking an existing type as the base and creating a new type by adding to its content. Definition of the base `complexType` data type is included in the definition of new `complexType` data type.You can add new elements and attributes to new `complexType` data type. For example, `complexType Employee` contains two `simpleType` elements: `FirstName` and `LastName`. You can use `complexType Employee` to create two new `complexTypes`, `Permanent` and `Contract`. In addition to the `FirstName` and `LastName` elements, `Permanent complexType` can contain elements such as `EmpID` and `DateOfJoining`, whereas `Contract complexType` can contain elements such as `ContractStartDate` and `ContractEndDate`.

When you import an XSD in InterConnect, the schema parser parses the schema and gives the resolved schema model. The resolved schema model resolves the type definitions by copying the definition of derived types to deriving type and adding the new elements and attributes to it.

10.6.2 Deriving complexTypes by Restriction

You can derive a `complexType` data type by restricting a base `complexType` data type. Restriction is type definition whose declarations are in a one-to-one relation with another specified type definition.

A `complexType` type derived by restriction is very similar to its base type, except that its declarations are more restricted than the corresponding declarations in the base type. All the components of the base type that are to be included in the derived type must be repeated in the derived type definition. For example `complexType Employee` contains eight `simpleType` elements: `EmployeeID`, `FirstName` , `LastName`, `DateOfBirth`, `DateOfJoining`, `DepartmentNumber`, `DepartmentName`, and `DepartmentLocation`. You can derive a new `ComplexType EmpDept` that contains department-related information of an employee such as `EmployeeID`, `DepartmentNumber`, `DepartmentName`, and `DepartmentLocation`.

When you derive a new type by restriction, the definition of derived types is not copied to the deriving type. InterConnect stores the definition which is being used by the elements in the hierarchy of the selected root element.

10.7 Unsupported Features and Limitations of XSD

There are certain features or constructs that are not supported by OracleAS Integration InterConnect. Also, certain features of XSD have some limitations. This section describes the following topics:

- Unsupported Features

10.7.1 Unsupported Features

Following are the unsupported features or constructs in XSD:

- Mixed Content Types

  If you try to import an XSD having mixed content types, then the InterConnect iStudio throws an error.

- Support for `xsi:type` in XML Instance

  In an XML instance, if you specify the `xsi:type` attribute, then the derived types by extension or restriction appear at the locations where the base type element is expected. InterConnet does not support the `xsi:type` attribute and assumes that the element is of the original type.

- The `any` element

  In a model group definition, if an `any` element is not located at the end of a sequence, then InterConnect throws an error.

10.7.2 Limitations of XSD

Any `ComplexType` element having only one single element, which is an `any` element, is assumed as a `String` by InterConnect. An element and an attribute contained in the an element cannot have the same name.

XSD validation (the Validate XML option) cannot be used if the payload has a complex structure(XSD based field with additional fields).

11

Using Enterprise Manager

This chapter describes how to manage OracleAS Integration InterConnect using the Oracle Enterprise Manager 10*g* Application Server Control Console. It contains the following topics:

- Overview of OracleAS Integration InterConnect Management using Oracle Enterprise Manager 10g Application Server Control Console
- Monitoring and Configuring OracleAS Integration InterConnect from Oracle Enterprise Manager 10g Application Server Control Console

11.1 Overview of OracleAS Integration InterConnect Management using Oracle Enterprise Manager 10*g* Application Server Control Console

The Oracle Enterprise Manager 10*g* Application Server Control Console provides a comprehensive system management platform to perform monitoring and administration tasks on an OracleAS Integration InterConnect instance. It provides real-time system monitoring details such as up and down status, current CPU and memory consumption, and current performance status of various adapters.

11.1.1 OracleAS Integration InterConnect Management Concepts and Components

The following OracleAS Integration InterConnect components can be managed by using the Oracle Enterprise Manager 10*g* Application Server Control Console:

- Repository
- Hub Queue
- Adapters

11.1.1.1 Repository

Repository is used to store metadata in the hub database. At run time, the repository provides access to this metadata to adapters. You can use the Oracle Enterprise Manager 10*g* Application Server Control Console to monitor and configure the repository.

11.1.1.2 Hub Queue

Oracle Advanced Queuing (AQ) is the underlying messaging infrastructure of OracleAS Integration InterConnect. The `OAI_HUB_QUEUE` advanced queue in OracleAS Integration InterConnect is responsible for routing and delivering the messages. You can use the Oracle Enterprise Manager Database Control to manage

hub queues and to view and manage the errors that may have occurred during message delivery.

11.1.1.3 Adapters

Adapters are the run-time engines that carry out the integration based on the metadata modeled during design time. Adapters transform and route messages between the application and the hub. You can use the Oracle Enterprise Manager 10*g* Application Server Control Console to configure and monitor adapters.

11.1.2 Oracle Enterprise Manager 10*g* Application Server Control Console Tasks

You can perform the following monitoring and administration tasks by using the Oracle Enterprise Manager 10*g* Application Server Control Console:

- Start and stop the repository, adapters, and hub queue.
- View the current running status (started or stopped) and memory and CPU usage of the repository, adapters, and hub queue.
- Configure the repository and adapters.
- View the OracleAS Integration InterConnect log files.

11.2 Monitoring and Configuring OracleAS Integration InterConnect from Oracle Enterprise Manager 10*g* Application Server Control Console

This section provides information about how to monitor and configure various OracleAS Integration InterConnect components. It contains the following topics:

- Accessing OracleAS Integration InterConnect
- Viewing Status and System Metrics
- Starting and Stopping OracleAS Integration InterConnect Components
- Monitoring Adapter Performance
- Configuring OracleAS Integration InterConnect Components
- Locating and Viewing Log Files

11.2.1 Accessing OracleAS Integration InterConnect

Perform the following steps to access OracleAS Integration InterConnect from the Oracle Enterprise Manager 10*g* Application Server Control Console:

1. Go to the following URL:

   `http://`*`hostname`*`:`*`port`*`/`

   where:

   - *`hostname`* is the name of the host on which the OracleAS Integration InterConnect middle-tier instance is installed
   - *`port`* is the number assigned during OracleAS Integration InterConnect installation. The following sources also identify the port number used to access the Oracle Enterprise Manager 10*g* Application Server Control Console:
     - The `ipsetupinfo.txt` file in the `install` subdirectory of your Oracle home directory

– The End of Installation page that is displayed during installation

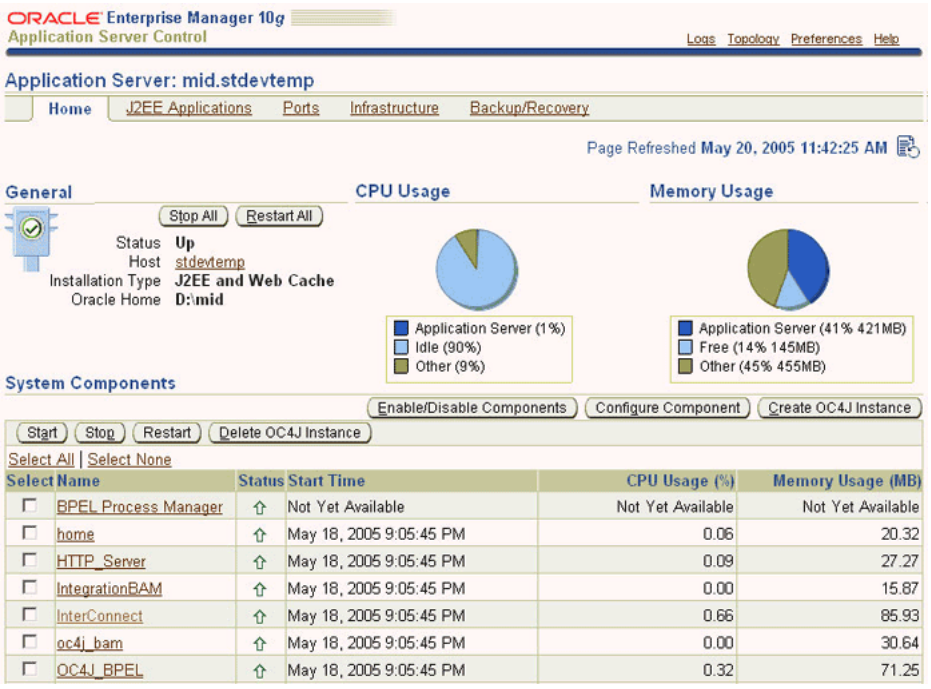
2. Enter the following login details when prompted:

Field	Value
Username	Enter `ias_admin`
Password	Enter the password you specified during J2EE and Web Cache installation and OracleAS Integration InterConnect installation for the `ias_admin` user.

Note: If you have multiple Oracle Application Server middle-tier instances installed on a single host, then you need to specify different Oracle Enterprise Manager 10*g* ports for each instance. See the `portlist.ini` file in the *`$ORACLE_HOME`*`/install` directory for a list of ports currently in use.

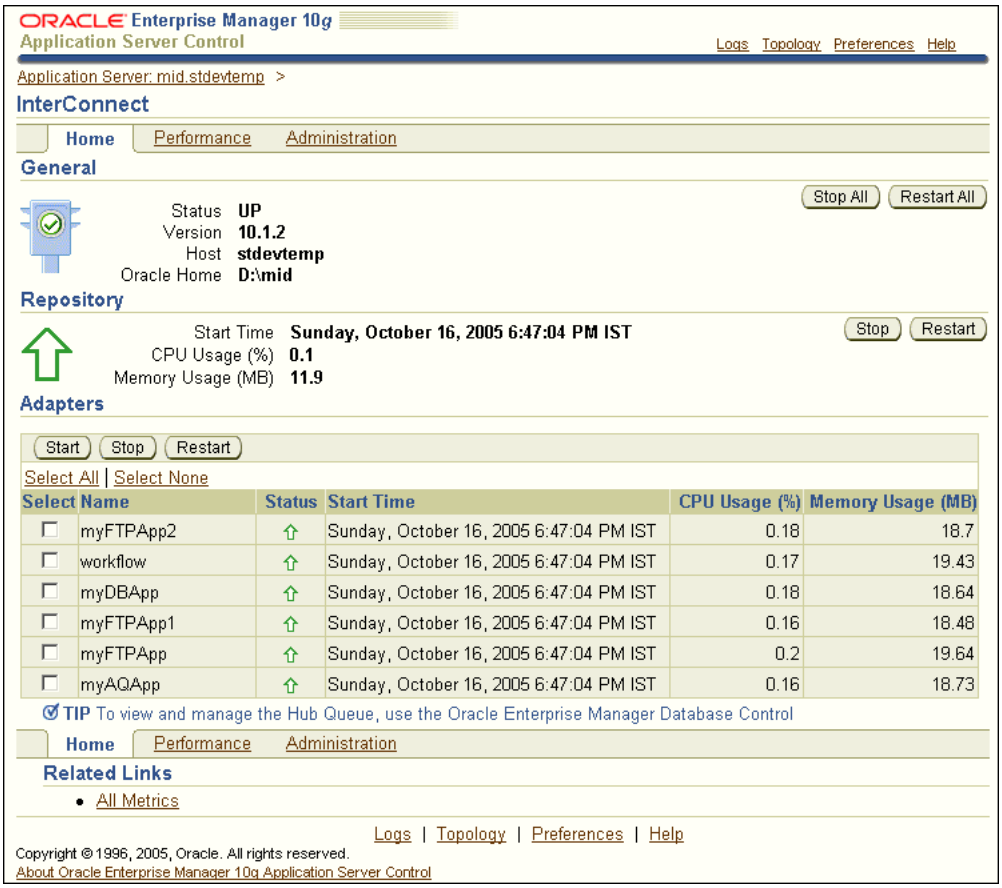
The Oracle Enterprise Manager 10*g* Application Server Control Console is displayed, as shown in Figure 11–1:

Figure 11–1 Oracle Enterprise Manager 10g Application Server Control Console



3. The System Components section provides basic information about various component installed on Oracle Application Server. Click **InterConnect**. The OracleAS Integration InterConnect middle-tier instance page is displayed, as shown in Figure 11–2:

Figure 11–2 OracleAS Integration InterConnect Middle-Tier Instance Page



11.2.2 Viewing Status and System Metrics

You can view the status and metrics of an OracleAS Integration InterConnect instance by using the following sections on the OracleAS Integration InterConnect middle-tier instance page:

- General Section
- Repository Section
- Adapters Section

11.2.2.1 General Section

The General section of the OracleAS Integration InterConnect middle-tier instance page, as shown in Figure 11–2, displays the current status of the OracleAS Integration InterConnect instance. It provides the following information:

- Status: The status of the OracleAS Integration InterConnect instance. The overall status of the instance depends upon the status of the repository and adapters. If the repository and all the adapters are running, then the status is shown as UP. However, if any one of these components is not running, then the status is shown as Down.
- Version: The version of OracleAS Integration InterConnect.
- Host: The Application Server middle-tier host computer on which OracleAS Integration InterConnect is installed.

- Oracle Home: The location of the Oracle home directory on the host computer.

11.2.2.2 Repository Section

The Repository section of the OracleAS Integration InterConnect middle-tier instance page, as shown in Figure 11–2, displays the current status of the repository. It provides the following information:

- Start Time: The time when the repository was started.
- CPU Usage: The amount of CPU being used by the repository.
- Memory Usage: The amount of memory being used by the repository.

11.2.2.3 Adapters Section

The Adapters section of the OracleAS Integration InterConnect middle-tier instance page, as shown in Figure 11–2, displays the current status of all adapters. It provides the following information about each adapter:

- Status: The status of an adapter.
- Start Time: The time when an adapter was started.
- CPU Usage: The amount of CPU being used by an adapter.
- Memory Usage: The amount of memory being used by an adapter.

11.2.3 Starting and Stopping OracleAS Integration InterConnect Components

This section describes how to start and stop various components of OracleAS Integration InterConnect by using the Oracle Enterprise Manager 10*g* Application Server Control Console. It contains the following topics:

- Starting and Stopping the Repository
- Starting and Stopping the Adapters
- Starting and Stopping the Hub Queue

11.2.3.1 Starting and Stopping the Repository

To start, stop, or restart the repository, use either of the following methods:

- Click **Start**, **Stop**, or **Restart** in the Repository section of the OracleAS Integration InterConnect middle-tier instance page, as shown in Figure 11–2.
- Click **Start All**, **Stop All**, or **Restart All** in the General section of OracleAS Integration InterConnect middle-tier instance page, as shown in Figure 11–2.

> **Note:** The **Start All**, **Stop All**, or **Restart All** buttons in the General section start, stop, or restart all the adapters and the repository. However, the hub queue is not started.

11.2.3.2 Starting and Stopping the Adapters

To start, stop, and restart adapters, use either of the following methods:

- Select one or more adapters in the Adapters section of the OracleAS Integration InterConnect middle-tier instance page. Click **Start**, **Stop**, or **Restart**, as shown in Figure 11–2.

- Click **Start All**, **Stop All**, and **Restart All** in the General section of OracleAS Integration InterConnect middle-tier instance page, as shown in Figure 11–2.

11.2.3.3 Starting and Stopping the Hub Queue

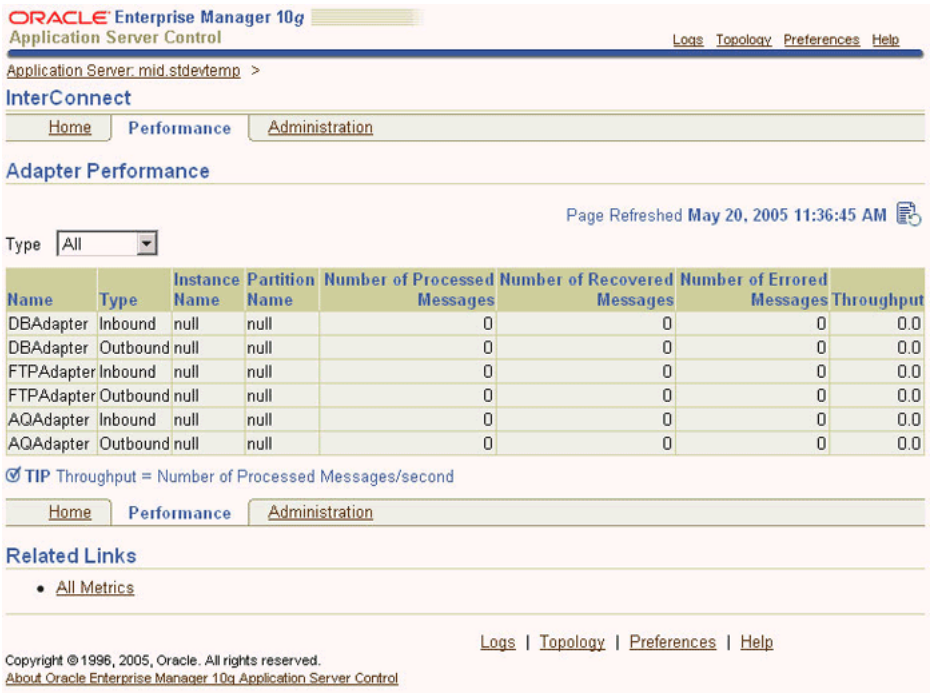
You cannot start or stop the hub queue from the Oracle Enterprise Manager 10*g* Application Server Control Console. You can start and stop the hub queue by using the Oracle Enterprise Manager Database Control Console.

> **See Also:** For information about Oracle Enterprise Manager Database Control, refer <Guide name to be added>.

11.2.4 Monitoring Adapter Performance

You can view the performance metrics of all adapters by clicking the **Performance** tab on the OracleAS Integration InterConnect middle-tier instance page. The Adapter Performance page is displayed, as shown in Figure 11–3. You can use the **Type** list to view only inbound messages, only outbound messages, or all messages.

Figure 11–3 OracleAS Integration InterConnect Middle-Tier Instance Adapters Performance Page



The Adapter Performance page provides the following information for each adapter:

- Name: The name of the adapter.
- Type: The type of the adapter, Inbound or Outbound.
- Instance Name: The name of the adapter instance.
- Partition Name: The name of the adapter partition.
- Number of Processed Messages: The number of messages successfully processed.
- Number of Recovered Messages: The number of messages that have been recovered. When a message could not be delivered to the target application by the adapter, it is moved to the `oai_agent_error` table. If the adapter is down, then

the message is persisted in the queue until the adapter is up and running. The message is then delivered to the application.

- Number of Errored Messages: The number of messages that are not successfully processed and delivered to the target application.
- Throughput: The number of messages processed in a second.

11.2.5 Configuring OracleAS Integration InterConnect Components

You can configure the OracleAS Integration InterConnect components by clicking the **Administration** tab on the OracleAS Integration InterConnect middle-tier instance page. Figure 11–4 displays the page that appears when you click the Administration link.

Figure 11–4 OracleAS Integration InterConnect Middle-Tier Instance Administration Page

You can use the Administration page to configure the following OracleAS Integration InterConnect components:

- Repository
- Adapters
- Hub

11.2.5.1 Configuring the Repository

To configure the repository:

1. Click **Repository Properties**, shown in Figure 11–4. The Repository Properties page is displayed, as shown in Figure 11–5.

Figure 11–5 OracleAS Integration InterConnect Middle-Tier Instance Repository Properties Page

2. Modify any of the properties listed in Table 11–1 and click **Apply**.

Table 11–1 Repository Properties

Property	Description
Repository Owner	The name of the repository owner.
Debug	Enable or Disable debugging of messages in the logs.
Service Path	The value of the environment variable PATH. This parameter is mandatory when Oracle Application Server middle-tier is installed on Microsoft Windows.
Service Classpath	The class path used by the adapter Java Virtual Machine. This parameter is mandatory when Oracle Application Server middle-tier is installed on Microsoft Windows.
Service Class	The Entry Class for the Windows Operating System Service. This parameter is mandatory when Oracle Application Server middle-tier is installed on Microsoft Windows.
Maximum Java Stack Size	The maximum size up to which the Java Virtual Machine Stack can grow. This parameter is required for Microsoft Windows only.
Maximum Native Stack Size	The maximum size to which the Java Virtual Machine Native Stack can grow. This parameter is required for Microsoft Windows only.
Minimum Heap Size	The minimum heap size for the Adapter Java Virtual Machine. This parameter is required for Microsoft Windows only.

Table 11–1 (Cont.) Repository Properties

Property	Description
Maximum Heap Size	The maximum heap size for the Adapter Java Virtual Machine. This parameter is required for Microsoft Windows only.
JDK Version	The JDK Version.
JDK DLL Name	The name of the JDK DLL.

11.2.5.2 Configuring the Adapters

To configure the adapters:

1. Click **Adapter Properties**, shown in Figure 11–4. The Adapter Properties page is displayed, as shown in Figure 11–6.

Figure 11–6 OracleAS Integration InterConnect Middle-Tier Instance Adapters Properties Page

2. Select an adapter from the Adapter Name list. The properties for the selected adapter are displayed.

Note: The adapter properties are different for different adapters. For example, the properties that are displayed when you select a database adapter are different from the properties of the FTP adapter.

---

3. Modify the adapter properties and click **Apply**.

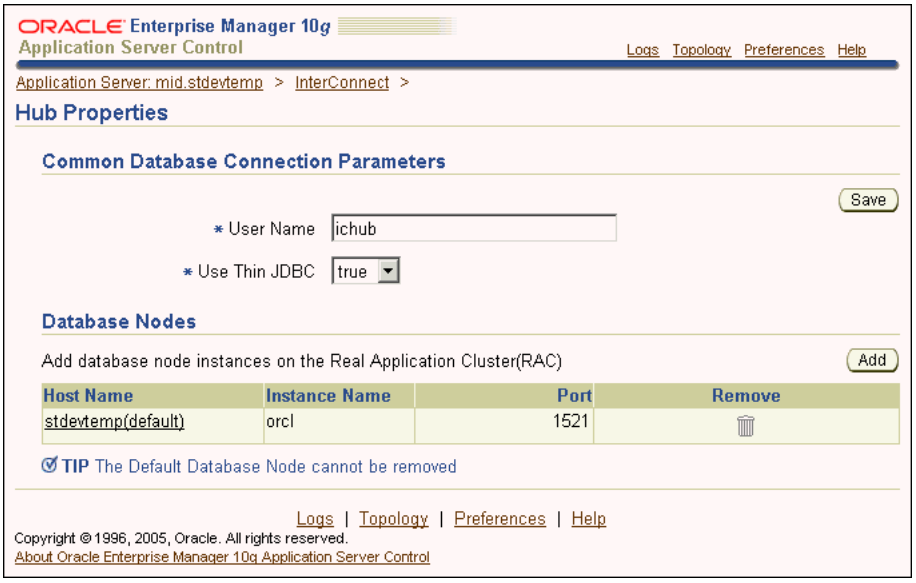
   See Also: For detailed information about properties of an adapter, refer to the following guides:

   - *Oracle Application Server Integration InterConnect Adapter for MQSeries Installation and User's Guide*
   - *Oracle Application Server Integration InterConnect Adapter for FTP Installation and User's Guide*
   - *Oracle Application Server Integration InterConnect Adapter for HTTP Installation and User's Guide*
   - *Oracle Application Server Integration InterConnect Adapter for SMTP Installation and User's Guide*
   - *Oracle Application Server Integration InterConnect Adapter for DB Installation and User's Guide*
   - *Oracle Application Server Integration InterConnect Adapter for AQ Installation and User's Guide*
   - *Oracle Application Server Integration InterConnect Adapter for Oracle Applications Installation and User's Guide*
   - *Oracle Application Server Integration InterConnect Adapter for JCA Installation and User's Guide*

11.2.5.3 Configuring the Hub Queue

To configure the Hub properties, click **Hub Properties**, shown in Figure 11–4. The Hub Properties page is displayed, as shown in Figure 11–7.

Figure 11–7 OracleAS Integration InterConnect Middle-Tier Instance Hub Properties Page



You can use the Hub Properties page to:

- Modify the Common Database Connection Parameters
- Add a Database Node
- Modify a Database Node
- Delete a Database Node

Modify the Common Database Connection Parameters

To modify the common database connection parameters, update any of the following fields and click **Save**:

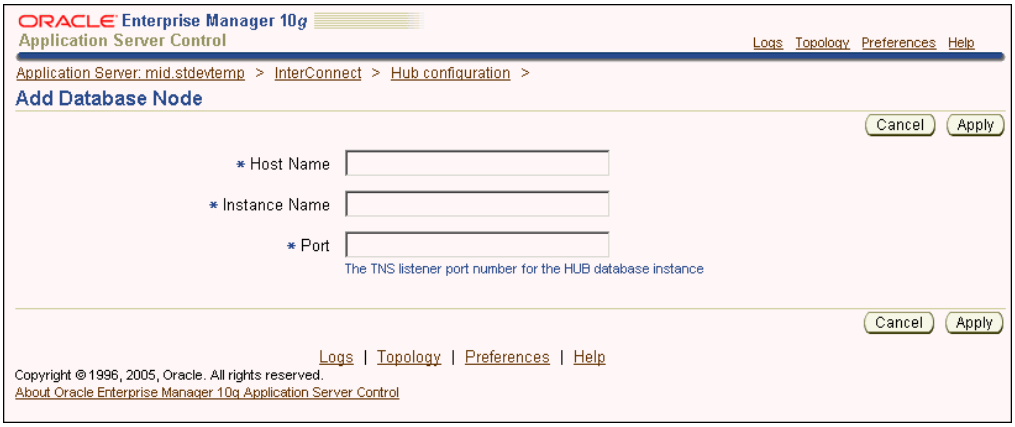
- User Name: The user name for connecting to Real Application Cluster.
- Use Thin JDBC: Specify whether to use a thin JDBC driver or thick JDBC driver.

Add a Database Node

To add a database node instance on the Real Application Cluster:

1. Click **Add**. The Add Database Node page is displayed, as shown in Figure 11–8.

Figure 11–8 OracleAS Integration InterConnect Middle-Tier Instance Add Database Node Page



2. Enter the host name of the database node.
3. Enter the instance name of the database node.
4. Enter the TNS listener port number for the hub database instance of a node.
5. Click **Apply**.

Modify a Database Node

To modify a database node:

1. Click the node name. The Edit Database Node page is displayed.
2. Enter the host name of the database node.
3. Enter the instance name of the database node.
4. Enter the TNS listener port number for the hub database instance of a node.
5. Click **Apply**.

Delete a Database Node

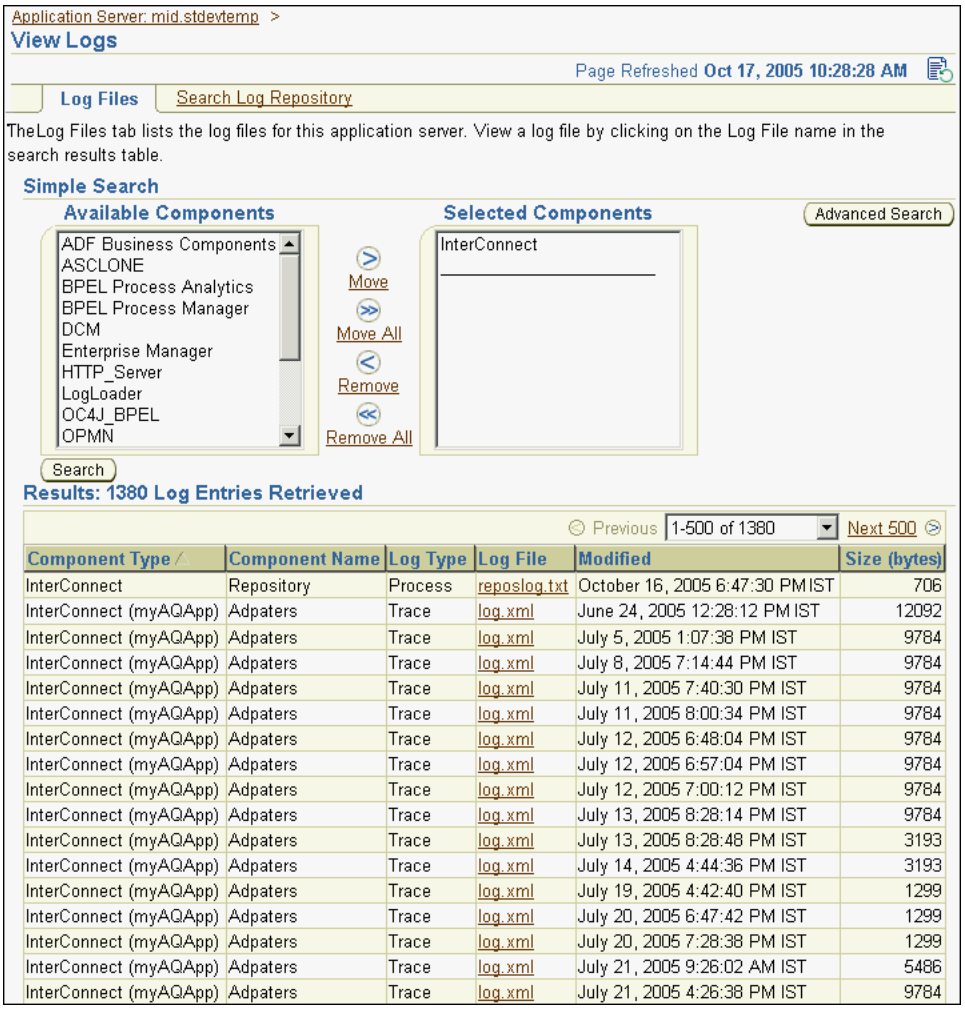
To delete a database node, click **Remove**.

11.2.6 Locating and Viewing Log Files

To view the InterConnect log files:

1. Click **Log** on the OracleAS Integration InterConnect middle-tier instance page. The View Logs page is displayed, as shown in Figure 11–9.

Figure 11–9 OracleAS Integration InterConnect Middle-Tier Instance View Logs Page



2. Select **InterConnect** from the Available Components list and click **Move**.
3. Click **Search**. All InterConnect log files are displayed in the Results section.
4. Select the file that you want to view.

12

Using InterConnect Manager

InterConnect Manager is a utility that takes care of both the run-time management and error handling requirements of OracleAS Integration InterConnect. This chapter describes the functionality of the utility in detail, in the following topics:

12.1 Overview of InterConnect Manager

InterConnect Manager is a command-line based, menu-driven utility that enables you to:

- List messages present in the hub queue and hub error table.
- View details and content of specific messages.
- Export messages from queues to files.
- Import messages to queues and send edited messages again.
- Track messages.
- Start and stop adapters.
- Install and uninstall adapters.
- Create and drop the hub schema in the hub database.
- Import/export the integration metadata repository to and from a file.
- Perform Adapter Error Management.

12.2 Starting InterConnect Manager

InterConnect Manager can run in two modes:

- Command-line, menu-driven mode
- Silent mode

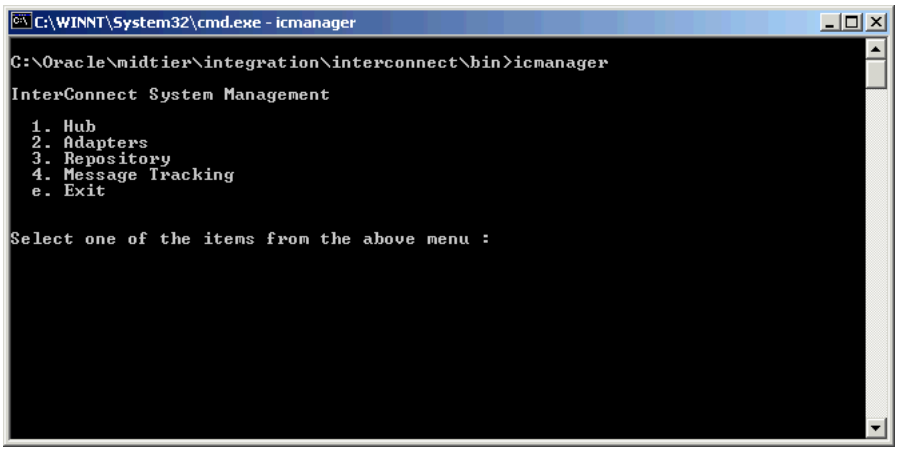
The silent mode is provided for calling the utility from another script.

To start InterConnect Manager in the command-line mode, enter the following command at the prompt:

```
ICManager [-properties hub.ini]
```

InterConnect Manager gets all its information from `OAIHOME/hub/hub.ini`. If you want it to take information from another location instead, then use the properties parameter and provide the absolute path of the `hub.ini` file as argument. This brings up the main menu of InterConnect Manager as shown in Figure 12–1. From here, you can manage the hub, adapters, and repository. You can also track messages across the OracleAS Integration InterConnect system, from one application to the other.

Figure 12–1 InterConnect Manager Main Menu



12.3 Using InterConnect Manager

The main menu of InterConnect Manager consists of the following options:

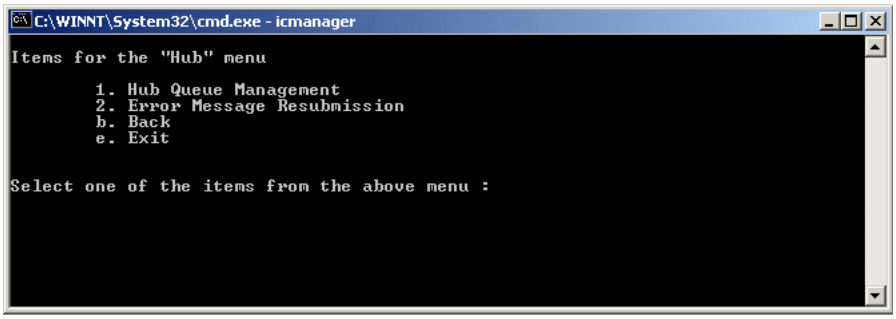
- Hub
- Adapters
- Repository
- Message Tracking

To select any option, enter its number at the prompt. Each option has further menu options under it.

12.3.1 Hub

When you select Hub in the main menu, the menu shown in Figure 12–2 is displayed.

Figure 12–2 Hub Menu



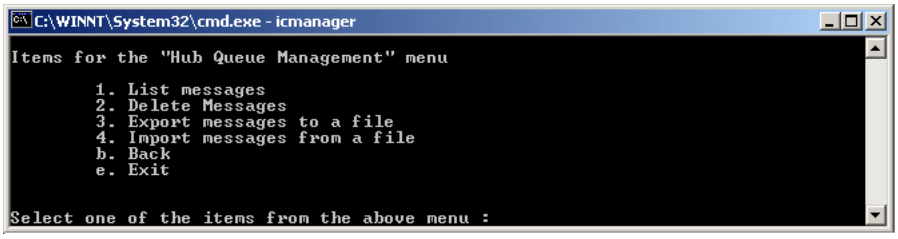
Use the Hub menu to manage the hub queues, and to view and manage the errors that may have occurred during message delivery. Key menu options include:

- Hub Queue Management
- Error Message Resubmission

12.3.1.1 Hub Queue Management

When you select Hub Queue Management in the Hub menu, the menu shown in Figure 12–3 is displayed.

Figure 12–3 Hub Queue Management Menu

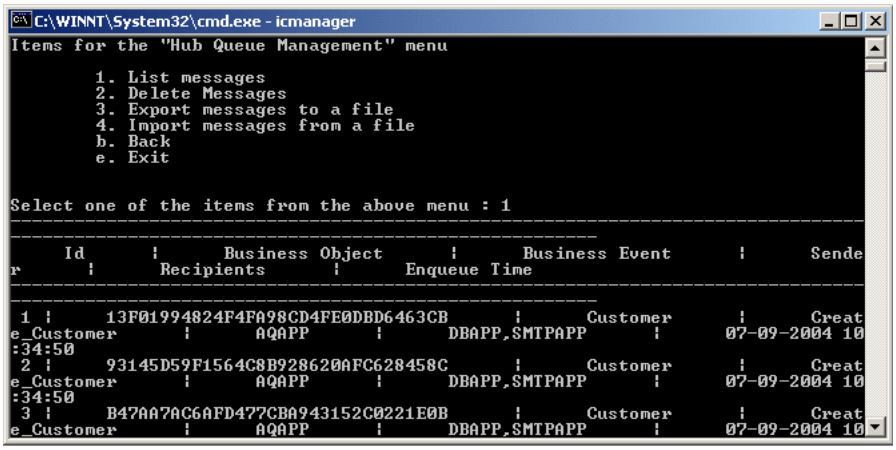


Hub Queue Management enables you to view and manage messages present in the hub queue. When messages arrive from adapters, they are placed in the hub queue. The hub processes each message in the queue on a first come-first serve basis, applying various routing rules and sending the messages to various adapters. You can also export one or more messages to a file, edit, and import them. Key menu options include:

- List Messages
- Delete Messages
- Export Messages to a File
- Import Messages from a File

12.3.1.1.1 List Messages When you select the List Messages option in the Hub Queue Management menu, details of all the messages present in the hub queue are displayed as shown in Figure 12–4.

Figure 12–4 List Messages

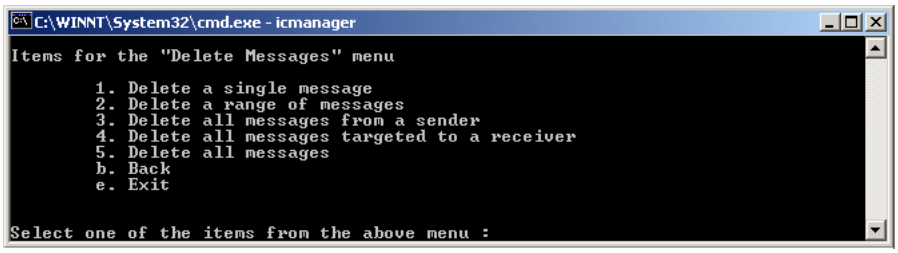


The message details include:

- Id: Id acts a unique identifier for each message in the system. No two messages will have the same Id.
- Enqueue Time: Enqueue Time is the time when the hub receives the message from the adapter.
- Business Object: The Business Object field contains the name of the business object corresponding to the message.
- Event: Each business object consists of one or more events. The Event field contains the name of the Business Event that triggered the message.
- Sender: The Sender field contains the name of the application that sent the message.
- Recipients: The Recipients field contains the names of the applications that will receive the message.

12.3.1.1.2 Delete Messages When you select the Delete Messages option in the Hub Queue Management menu, the menu in Figure 12–5 is displayed.

Figure 12–5 Delete Messages



The key menu options are:

When an option is chosen, InterConnect Manager displays a list of messages present in the queue, and prompts the user about which messages should be deleted from the queue.

- Delete a single message
- Delete a range of messages
- Delete all messages from a sender
- Delete all messages targeted to a receiver
- Delete all Messages

Delete a single message

InterConnect Manager requests the index number of the message to be deleted and then removes the message from the queue.

Delete a range of messages

InterConnect Manager requests the low range value and the high range value. It then removes all messages from the hub queue with Ids in the specified range.

Delete all messages from a sender

InterConnect Manager requests the name of the sender, and removes all the messages present in the hub queue.

Delete all messages targeted to a receiver

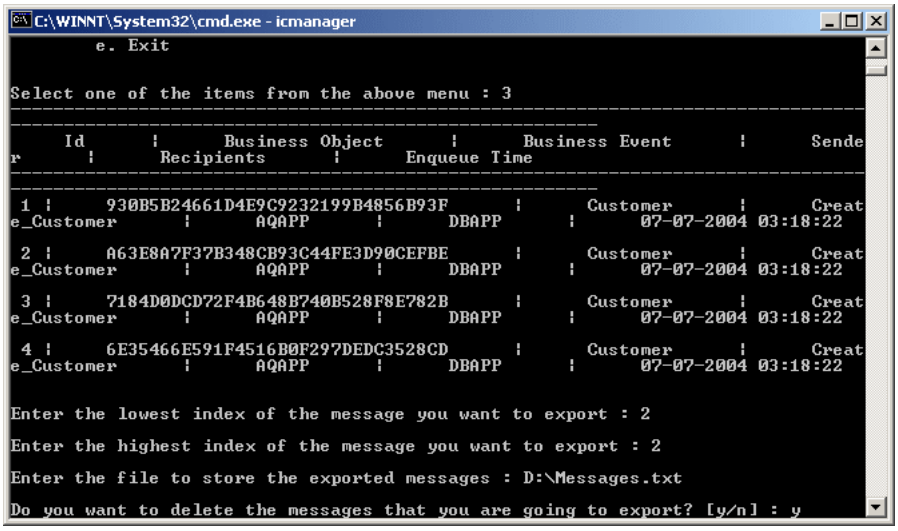
InterConnect Manager requests the name of the receiver, and removes all its messages present in the hub queue. If a message has more than one recipient and one of them is targeted from removal, only the name of the targeted recipient is removed from the message. For example, if you decide to delete all messages targeted to application A, and a particular message in the queue has recipients A and B, the message will not be dropped. Instead, A will be removed from the recipient list.

Delete all Messages

This option removes all messages present in the hub queue.

12.3.1.1.3 Export Messages to a File This option enables you to export a message, or a range of messages to a file. When you select this option, the menu shown in Figure 12–6 is displayed.

Figure 12–6 Export Messages to a File



InterConnect Manager displays the list of messages present in the hub queue. It then requests for the following information for the export process:

1. Enter the lowest index of the message you want to export:

   Enter the lower bound of the range of messages to be exported.

2. Enter the highest index of the message you want to export:

   Enter the upper bound of the range of messages to be exported.

   If you wish to export only one message, then enter the same message Id for both the lower and upper bounds.

3. Enter the file to store the exported messages:

   Enter the full path of the text file that stores the exported message. If the file already exists, then it will be overwritten.

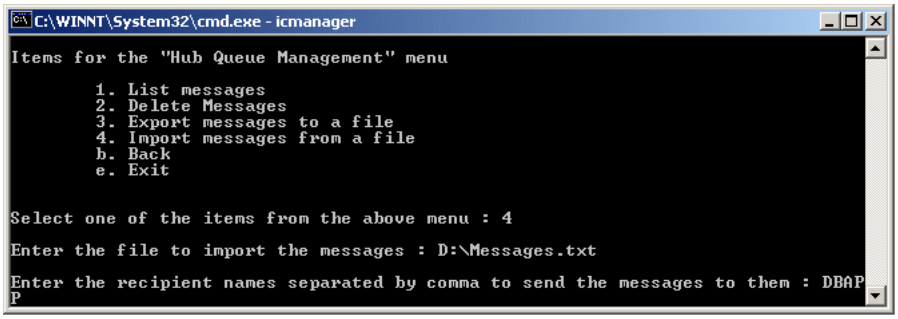
4. Do you want to delete the message that you are going to export?

   If you wish to drop the messages from the queue, then enter `y`. Once the messages have been exported to the target file, InterConnect Manager displays a confirmation. You can open the file in any text editor, view the contents, and change them to suit your needs. You can then import the messages back into the

hub queue by choosing the Import Messages From a File option in the Hub Queue Management menu.

12.3.1.1.4 Import Messages from a File The Import Messages From a File option enables you to import a message or a range of messages from a file into the hub queue. When you select this option, the menu shown in Figure 12–7 is displayed.

Figure 12–7 Import Messages from a File



InterConnect Manager requests following information for the import process:

1. Enter the file to import the messages:

   The messages that you want to import must all be present in a single XML file. Enter the full path of the file that contains the messages to be imported.

2. Enter the recipient name separated by comma to send the message to them:

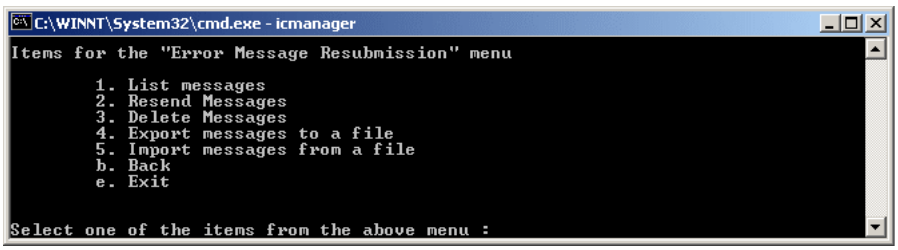
   Enter names of the applications that will receive the imported messages. Separate the names using commas.

12.3.1.2 Error Message Resubmission

If a message cannot be delivered, for some reason, to the target application by the adapter, then it is placed in the Error table of the hub. The Error Message Resubmission option provides you with the tools to manage the Error table of the hub, and enables you to carry out various actions on the messages in the queue.

When you select Error Message Resubmission in the Hub menu, the menu shown in Figure 12–8 is displayed.

Figure 12–8 Error Message Resubmission Menu



Key menu options include:

- List Messages
- Resend Messages
- Delete Messages

- Export Messages to a File
- Import Messages from a File

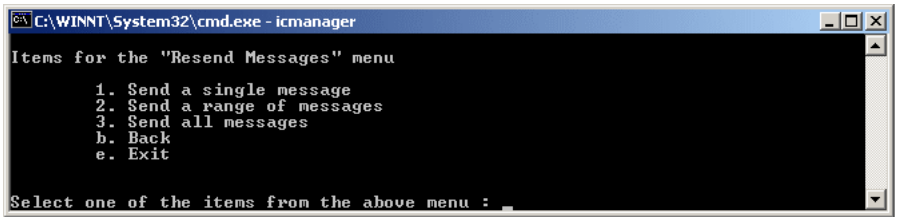
12.3.1.2.1 List Messages The List Messages option provides the details of all the messages present in the error queue. The message details include:

- Id: Id acts as a unique identifier for each message in the system. No two messages will ever have the same Id.
- Enqueue Time: Enqueue Time is the time when the hub receives the message from the adapter.
- Sender: The Sender field contains the name of the application that sent the message.
- Recipient: The Recipient field contains the name of the application that was supposed to receive the message.
- LoggingComponent: The LoggingComponent field contains the name of the component that logged the error. This helps to identify the exact point at which the error occurred.
- Error Description: The Error Description field gives a brief description of the error and the action taken.

12.3.1.2.2 Resend Messages The Resend Messages option of the Error Management menu enables you to resend messages that have been added into the Error table. For example, if the message could not be delivered to the target application by the adapter, then the message is moved to the `oai_agent_error` table. But if the adapter is down, then the message will persist in the queue, until the adapter is up and running.

When you select the Resend Messages option in the Error Management menu, the menu shown in Figure 12–9 is displayed.

Figure 12–9 Resend Messages



InterConnect Manager displays another menu where you can select:

- Send a Single Message
- Send a Range of Messages
- Send All Messages

InterConnect Manager then requests the following information related to the resend operation:

1. Enter the index of the error message you want to resend:

   Enter the index number of the message to be resent.

2. Enter the recipient name separated by comma to send the message to them:

   Enter the names of the application that will receive the message. Separate the names using commas.

3. Enter the priority for the imported messages:

   The priority level decides how quickly the messages will be sent to the recipients. You can select from level 0-9, with 0 being the lowest and 9 being the highest priority.

4. Do you want to delete the selected error message from the error table?

   If you wish to drop the messages from the Error table, then enter y, else enter n.

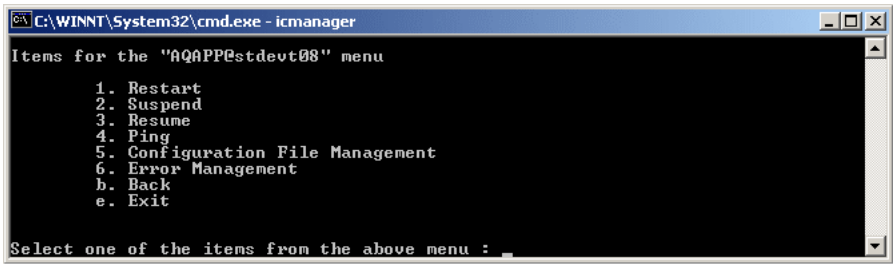
The functionality of the Delete Messages, Export Messages To a File, and Import Messages From a File options are explained in earlier sections.

> **See Also:** Delete Messages on page 12-4, Export Messages to a File on page 12-5, and Import Messages from a File on page 12-6

12.3.2 Adapters

When you select Adapters in the main menu, the menu in Figure 12–10 is displayed.

Figure 12–10 Adapters Menu



The complete list of adapters that are present is displayed. When you select any adapter, InterConnect Manager displays the Adapters menu. Use the Adapters menu to manage the various adapters that form the spokes in the OracleAS Integration InterConnect hub and spoke paradigm. An identical menu is presented for each adapter. Key menu options include:

- Restart: Restarts the adapter.
- Suspend: Temporarily suspends all activity on the adapter. The adapter no longer sends or accepts messages.
- Resume: Enables the adapter to start its normal activities again. This is a counterpart to the Suspend command.
- Ping: Checks to see if the adapter is up and active.
- Configuration File Management
- Error Management

The Configuration File Management and Error Management options are described in detail in the following section.

12.3.2.1 Configuration File Management

The Configuration File Management option of the Adapter Management menu enables you to manage configuration files for adapters. To change the behaviour of an adapter, you must edit its `adapter.ini` file. InterConnect Manager enables you to remotely read and edit the configuration file.

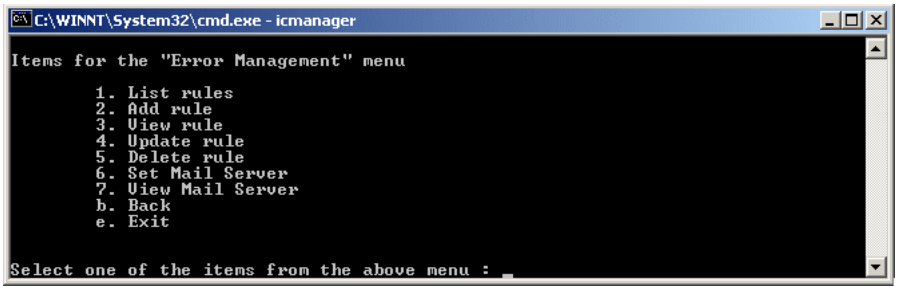
Key menu options in the Configuration File Management menu include:

- View Config File: Displays the contents of the config file (`adapter.ini` file).
- Edit Config File: Reads the adapter's config file (`adapter.ini` file) from the remote machine and saves it to the local machine.
- Update Config File: Writes the adapter's config file (`adapter.ini` file) to the remote machine where the adapter is installed. You can stop and restart the adapter to reread the config file.

12.3.2.2 Error Management

The Error Management option of the Adapter Management menu enables you to create rules for errors that occur during adapter operation. For example, if an adapter receives messages with an invalid format, the messages are processed in accordance with the rules specified here. Figure 12–11 displays the Error Management Menu.

Figure 12–11 Error Management Menu



Key menu options in the Error Management menu include:

- List Rules
- Add Rule
- View Rule
- Update Rule
- Delete Rule
- Set Mail Server
- View Mail Server

12.3.2.2.1 List Rules The List Rules option displays the list of rules that are currently set for the adapter.

12.3.2.2.2 Add Rule The Add Rule option enables you to add new rules for the adapter error messages.

To add a new rule, enter 2 in the Error Management menu. InterConnect Manager asks a series of questions that help build the new rule for the error messages. The questions are as follows:

1. Enter the name for this rule:

   Enter a name for the rule you are about to create. Use alphanumeric characters only and do not use spaces in the name.

2. Enter the error codes separated by comma:

   The rule will apply to all error codes listed here.

3. Do you want the adapter to retry the message in case of above errors?

   If you enter `y`, then the adapter will retry sending the message. If you enter `n`, then skip to Step 7.

4. How many times would you like to retry the message?

   Enter the number of times the adapter must retry sending the message before giving up. If the retries fail, then a message is deleted from the queue.

5. What is the interval for each retry in milliseconds?

   Enter the time interval between each retry.

6. Do you want to perform more actions if retry fails?

   If you select `y`, then InterConnect Manager continues with more questions. If you select `n`, then the questions stop at this point.

7. Do you want to send mail notification?

   If you select `y`,then InterConnect Manager continues with more questions. If you select `n`, then the questions stop at this point.

8. Enter the From address:

9. Enter the To addresses separated by comma:

10. Enter the subject [$$ERROR_CODE and $$ERROR_MESSAGE$$ can be used as part of the subject]:

    You can enter any text in the subject field, which will be used as the subject of the mails sent in case of an error. If you have used the variables $$ERROR_CODE$$ and $$ERROR_MESSAGE$$ in the message, then they are dynamically replaced with the appropriate error code and error message before the mail is sent.

11. Enter the message body [$$ERROR_CODE and $$ERROR_MESSAGE$$ can be used as part of the subject]:

    You can enter any text in the message body, which will be used as the message sent in case of an error. If you have used the variables $$ERROR_CODE$$ and $$ERROR_MESSAGE$$ in the message, then they are dynamically replaced with the appropriate error code and error message before the mail is sent.

12.3.2.2.3 View Rule The View Rule option enables you to view the existing rules.

12.3.2.2.4 Update Rule The Update Rule option enables you to edit the existing rules.

12.3.2.2.5 Delete Rule The Delete Rule option enables you to delete an existing rule.

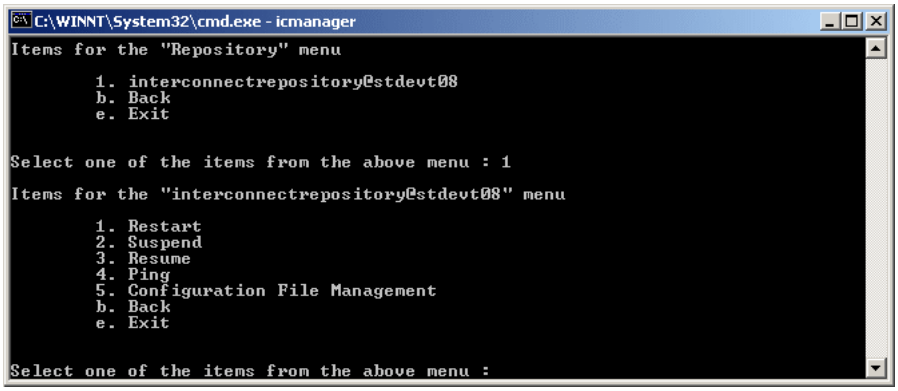
12.3.2.2.6 Set Mail Server The Set Mail Server option enables you to set the SMTP mail server that will be used to mail updates and errors.

12.3.2.2.7 View Mail Server The View Mail Server option enables you to view the current SMTP mail server.

12.3.3 Repository

When you select Repository in the main menu, the menu shown in Figure 12–12 is displayed.

Figure 12–12 Repository Menu



Use the Repository menu to manage the hub repository.

Key menu options include:

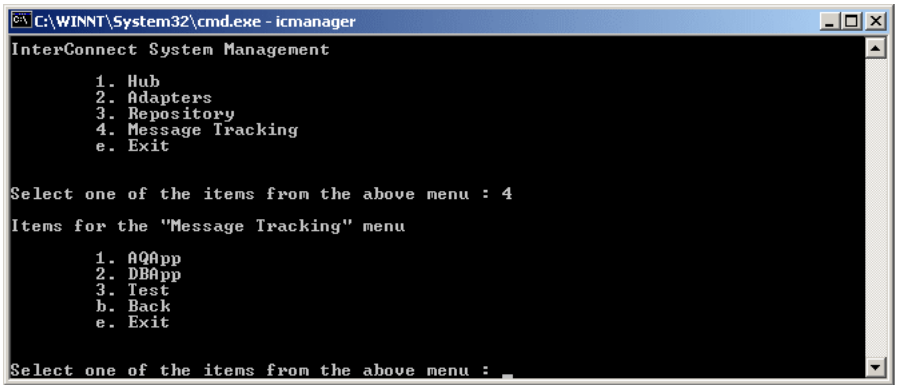
- Restart
- Suspend
- Resume
- Ping
- Configuration File Management

You can carry out all major administrative tasks on the repository from InterConnect Manager. In the case of Configuration File Management option, the functionality is identical to that described on page 12-8, except that the config file read from or written to is the `repository.ini` file.

12.3.4 Message Tracking

When you select Message Tracking in the main menu, the menu shown in Figure 12–13 is displayed.

Figure 12–13 Message Tracking



The Message Tracking option first lists the applications participating in the integration. When you select an application, InterConnect Manager lists the Business Objects associated with the application.

When you select a Business Object, InterConnect Manager lists the Business Events associated with the Business Object.

When you select a Business Event, InterConnect Manager lists the Message Tracking menu.

Key menu options for the Message Tracking menu include:

- Consolidated Information
- Detailed Information
- Detailed Information For a Tracking ID

When you select the Consolidated Information option, InterConnect Manager asks you a set of questions that help identify the exact set of messages whose consolidated information needs to be collected and then displays the messages in a tabular format.

When you select the Detailed Information option, InterConnect Manager directly displays the list of all the messages associated with the particular Business Event.

When you select the Detailed Information For a Tracking ID option, InterConnect Manager asks for the tracking ID details and then displays the messages in a tabular format.

12.4 Using InterConnect Manager in Silent Mode

InterConnect Manager also has a silent mode of operation, where no menu is displayed and all commands are given directly on the command line. InterConnect Manager commands in the silent mode have the following form:

```
ICManager -component componenttype -name componentname -action actiontype
```

Here, `componenttype` is the type of the component, `componentname` is the name of the component instance that InterConnect Manager should act on, and `actiontype` specifies the action to carry out on the component.

You can use InterConnect Manager to carry out the following activities:

Restart a Component

To restart a component, provide the component type, component instance name and restart as the argument for the action parameter. For example, to restart an adapter called DBApp, the command is:

```
ICManager -component adapter -name DBApp -action restart
```

Similarly, to restart a repository called ICrepo, the command is:

```
ICManager -component repository -name ICrepo -action restart
```

Suspend a Component

To suspend a component, provide the component type, component instance name and suspend as the argument for the action parameter. For example, to suspend the DBApp adapter, the command is:

```
ICManager -component adapter -name DBApp -action suspend
```

Similarly, to suspend the ICrepo repository, the command is:

```
ICManager -component repository -name ICrepo -action suspend
```

Resume a Component

To bring a suspended component into running mode, provide the component type, component instance name and resume as the argument for the action parameter. For example, to resume the DBApp adapter, the command is:

```
ICManager -component adapter -name DBApp -action resume
```

Similarly, to resume the ICrepo repository, the command is:

```
ICManager -component repository -name ICrepo -action resume
```

Check a Component's Availability

To view if a component is running, provide the component type, component instance name and ping as the argument for the action parameter. For example, to view if the DBApp adapter instance is running, the command is:

```
ICManager -component adapter -name DBApp -action ping
```

Similarly, for the ICrepo repository, the command is:

```
ICManager -component repository -name ICrepo -action ping
```

List the Contents of Queues

To view the contents of either the errortable or the hub queue, provide hub as the component type, errortable or queue as the component name and list as the action. For example, to list the contents of the errortable queue, the command is:

```
ICManager -component hub -name errortable -action list
```

Similarly, to list the contents of the hub queue, the command is:

```
ICManager -component hub -name queue -action list
```

Delete the Contents of Queues

To delete the contents of either the errortable or the hub queue, provide hub as the component type, errortable or queue as the component name and delete as the action. For example, to list the contents of the errortable queue, the command is:

```
ICManager -component hub -name errortable -action delete
```

Similarly, to delete the contents of the errortable queue, the command is:

```
ICManager -component hub -name errortable -action delete
```

A

InterConnect Interoperability Feature with Plug-in for BPEL

This appendix provides an overview of the Interconnect Interoperability feature with plugin for BPEL (BPEL Plug-in) and how it enables interaction between Oracle BPEL Process Manager and OracleAS Integration InterConnect. It contains the following topics:

A.1 Introduction

The BPEL Plug-in bridges Oracle BPEL Process Manager and OracleAS Integration InterConnect. You can use the BPEL processes defined in Oracle BPEL Process Manager for business process automation requirements in OracleAS Integration InterConnect.

A.1.1 Benefits of the BPEL Plug-in

Following are the benefits of the BPEL Plug-in:

- BPEL processes defined in Oracle BPEL Process Manager can be used for business process automation requirements in OracleAS Integration InterConnect.
- Oracle BPEL Process Manager can be used with existing InterConnect based integration solutions.
- J2CA adapters plugged into Oracle BPEL Process Manager can be integrated with OracleAS Integration InterConnect.

A.2 BPEL Plug-in Concepts

This section contains the following topics:

A.2.1 BPEL Servers and InterConnect Hub Topologies

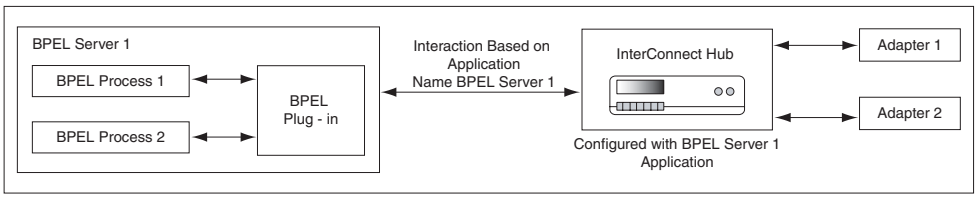
This section describes the various topologies in which Oracle BPEL Process Manager and InterConnect hubs can exist and how the BPEL Plug-in enables the interaction between multiple BPEL servers and InterConnect hubs. It contains following scenarios:

- Scenario 1: One BPEL Server Interacting with One InterConnect Hub
- Scenario 2: One BPEL Server Interacting with Two InterConnect Hubs
- Scenario 3: Two BPEL Servers Interacting with One InterConnect Hub
- Scenario 4: One BPEL Server in Load-Balancing Mode Interacting with One InterConnect Hub

A.2.1.1 Scenario 1: One BPEL Server Interacting with One InterConnect Hub

In this scenario, one instance of the BPEL Plug-in handles all interactions between the BPEL server and the InterConnect hub as shown in Figure A–1. To implement this scenario, create one application for the BPEL server in iStudio and define all interactions in this application. The BPEL Plug-in publishes and subscribes to the InterConnect hub queue with the application name representing the BPEL server.

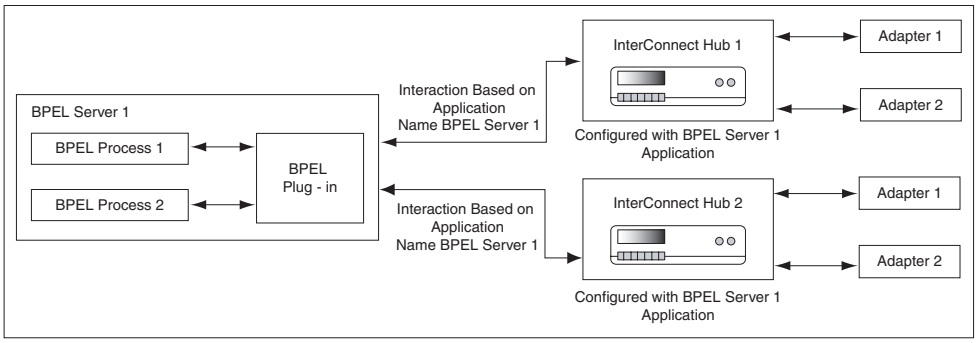
Figure A–1 One BPEL Server Interacting with One InterConnect Hub



A.2.1.2 Scenario 2: One BPEL Server Interacting with Two InterConnect Hubs

In this scenario, one instance of the BPEL Plug-in handles all interactions between the BPEL server and both the InterConnect hubs as shown in Figure A–2. To implement this scenario, create one application for the BPEL server in each InterConnect hub. The BPEL Plug-in publishes and subscribes to both the InterConnect hub queues using the corresponding application name.

Figure A–2 One BPEL Server Interacting with Two InterConnect Hubs

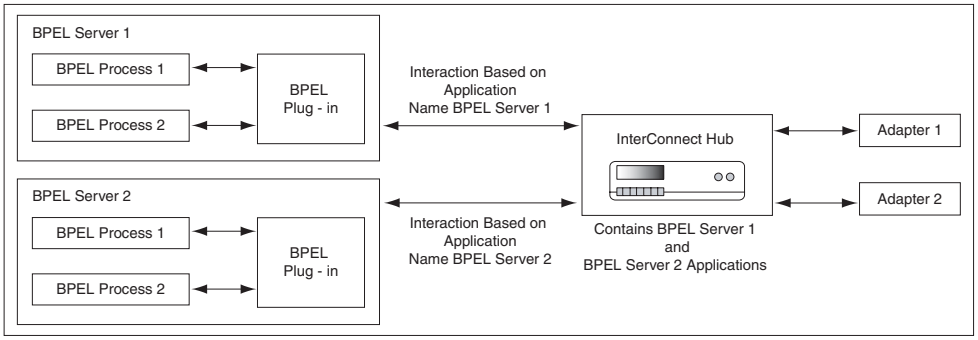


A.2.1.3 Scenario 3: Two BPEL Servers Interacting with One InterConnect Hub

In this scenario, there are two instances of the BPEL Plug-in as shown in Figure A–3. To implement this scenario, create two applications in iStudio. Each application

represents one BPEL server. The BPEL Plug-in publishes and subscribes to the InterConnect hub queue by using the corresponding application name.

Figure A–3 Two BPEL Servers Interacting with One InterConnect Hub

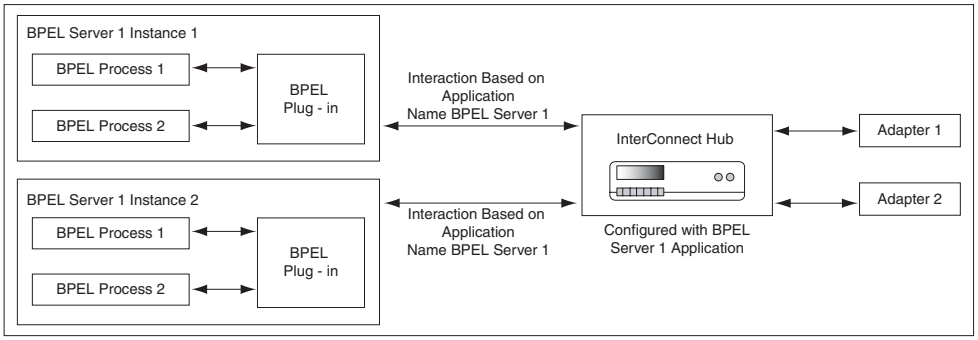


A.2.1.4 Scenario 4: One BPEL Server in Load-Balancing Mode Interacting with One InterConnect Hub

In this scenario, there are two instances of a BPEL server running in load-balancing mode as shown in Figure A–4. These instances are identical during run time. Thus, two instances of the Interconnect Interoperability feature with plugin for BPEL are created. Both instances cater to the same BPEL processes and publish or subscribe to the InterConnect hub queue with the same application name. Although both the instances subscribe to hub queue with the same application name but Advanced Queuing (AQ) ensures that the message is delivered to only one of the them.

If more than one subscriber have subscribed with the same name then AQ delivers the message to only one of them.

Figure A–4 One BPEL Server in Load-Balancing Mode Interacting with One InterConnect Hub



A.2.2 Local Cache of Adapter Metadata Concepts

When the BPEL Plug-in accesses the metadata for the first time, it is stored locally in cache. The BPEL Plug-in performs the following tasks to create and refresh the local cache of adapter metadata:

- When the connection is created for InterConnect hub, the BPEL Plug-in checks whether a local cache is available for that particular InterConnect hub. If a local cache is not available, then a local cache for that particular InterConnect hub is created. To create the local cache, the adapter connects to the InterConnect repository and retrieves all the relevant metadata.

- A queue is created during the creation of the InterConnect hub schema to hold the local cache refresh notifications for the BPEL Plug-in.
- The BPEL Plug-in creates a subscribing thread for this queue to receive the `Refresh local cache` message. On receiving the message, it refreshes the local cache with the new metadata from the InterConnect repository.

  Refresh Local Metadata is a new feature available with iStudio to refresh the metadata of BPEL Plug-in. You can refresh the metadata of the BPEL Plug-in by right clicking on a BPEL application in iStudio and selecting **Refresh Local Metadata** option. This triggers the BPEL Plug-in to get the current metadata from InterConnect and store it locally. This is similar to Sync adapters functionality available with InterConnect adapters.

A.2.3 BPEL Plug-in Interaction Concepts

This section describes the types of interaction between Oracle BPEL Process Manager and OracleAS Integration InterConnect. It contains the following topics:

- Oracle BPEL Process Manager and OracleAS Integration InterConnect Interaction Process
- Outbound Interaction Concepts
- Inbound Interaction Concepts
- Request/Reply Interaction Concept

A.2.3.1 Oracle BPEL Process Manager and OracleAS Integration InterConnect Interaction Process

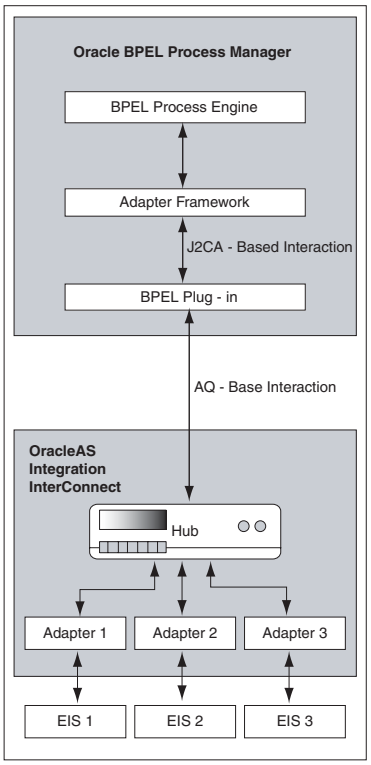
The BPEL Plug-in is deployed in the OC4J container of the Oracle BPEL Process Manager. Oracle BPEL Process Manager processes communicate with the BPEL Plug-in through adapter framework.

When the BPEL Plug-in receives a message from Oracle BPEL Process Manager, it converts the message into a format supported by Interconnect. The BPEL Plug-in identifies the recipient list by looking up the InterConnect metadata and queues the message to the InterConnect hub queue. The BPEL Plug-in directly interacts with the InterConnect Hub. However, when the BPEL Plug-in receives a message from the InterConnect hub, then it translates the received message into the XMLRecord object and delivers it to the BPEL process that subscribed to the message.

Figure A–5 displays the interaction process between Oracle BPEL Process Manager and OracleAS Integration InterConnect.

Note: The `XMLRecord` object contains two record elements: one for the header and one for the payload.

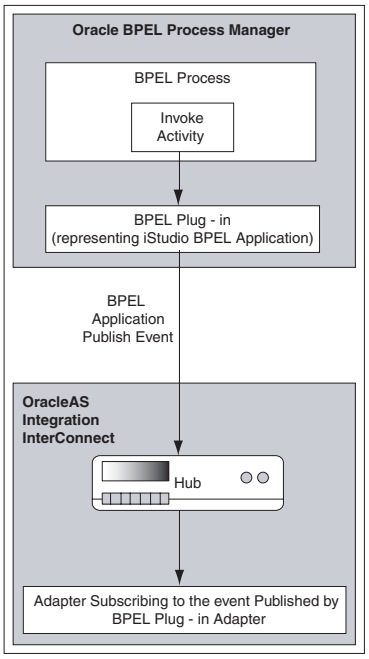
Figure A–5 Interaction Process between Oracle BPEL Process Manager and OracleAS Integration InterConnect



A.2.3.2 Outbound Interaction Concepts

In outbound interaction, Oracle BPEL Process Manager sends a message to OracleAS Integration InterConnect. This involves an invoke activity in Oracle BPEL Process Manager and a publish event in OracleAS Integration InterConnect as shown in Figure A–6.

Figure A–6 Outbound Interaction between Oracle BPEL Process Manager and OracleAS Integration InterConnect



An outbound interaction between Oracle BPEL Process Manager and OracleAS Integration InterConnect consists of the following steps:

1. The BPEL Plug-in receives a message from a BPEL process through invoke activity.
2. The BPEL Plug-in converts the message into an InterConnect message format so that it can be added to the hub queue.
3. The BPEL Plug-in adds appropriate header to the message, based on the type of the message.
4. Mapping between the Oracle BPEL Process Manager message and the InterConnect message is established based on the interaction specification parameters such as `BusinessObjectName`, `EventName`, and `EventVersion` defined at design time.

   Note: Mapping between the Oracle BPEL Process Manager message and the InterConnect message is established based on the values specified at design time. Thus, you should not change the data in generated WSDL files.

5. The BPEL Plug-in uses the local cache of adapter metadata to find a list of the recipients of the message and adds it to the recipient list of the message. It also evaluates the content based routing (CBR) conditions specified in the InterConnect repository before publishing the message to the hub queue.

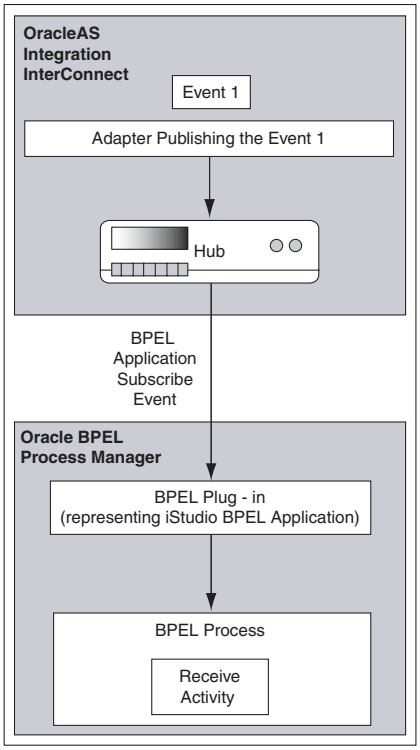
   See Also: "Local Cache of Adapter Metadata Concepts" on page A-3 for more information about how the BPEL Plug-in creates the local cache of adapter metadata.

6. The BPEL Plug-in queues the message to the InterConnect hub queue by using the Oracle AQ APIs.

A.2.3.3 Inbound Interaction Concepts

In inbound interaction, OracleAS Integration InterConnect sends a message to Oracle BPEL Process Manager. This involves a receive activity in Oracle BPEL Process Manager and a subscribe event in OracleAS Integration InterConnect as shown in Figure A–7.

Figure A–7 Inbound Interaction Between Oracle BPEL Process Manager and OracleAS Integration InterConnect



An inbound interaction between Oracle BPEL Process Manager and OracleAS Integration InterConnect consists of the following steps:

1. The BPEL Plug-in subscribes to the hub queue with the application name configured in the `oc4j-ra.xml` file. The task of subscribing to a hub queue is performed during the endpoint activation call to the adapter. The BPEL Plug-in keeps a track of all the subscribed hub queues so that a hub queue is not subscribed more than once for the same application.
2. The BPEL Plug-in receives all the events that the corresponding application in OracleAS Integration InterConnect is subscribing to.
3. The BPEL Plug-in translates the received events into the XMLRecord object. The received event is given to an endpoint by matching the value of `BusinessObjectName`, `Eventname`, and `EventVersion` parameters of the received message to values defined for endpoints.
4. The BPEL Plug-in uses the `onMessage` method to pass the received event to a BPEL process.

A.2.3.4 Request/Reply Interaction Concept

A Request/Reply interaction between Oracle BPEL Process Manager and OracleAS Integration InterConnect involves:

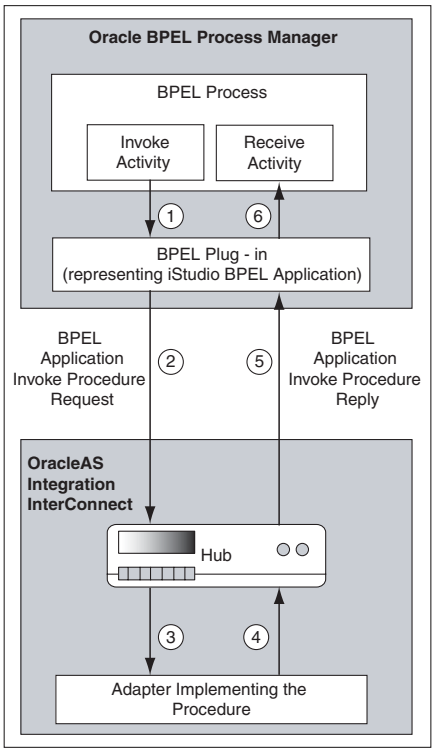
- invoke and receive activity in Oracle BPEL Process Manager (for asynchronous request/reply)
- receive and reply activity in Oracle BPEL Process Manager (for synchronous request/reply)
- invoke procedure and implement procedures in OracleAS Integration InterConnect

There can be two scenarios for request/reply interaction between Oracle BPEL Process Manager and OracleAS Integration InterConnect.

Scenario 1: BPEL Process Sending a Request and Receiving a Reply

In this scenario, a BPEL process sends a request to the InterConnect hub and receives a reply asynchronously. This interaction involves using invoke and receive activity in Oracle BPEL Process Manager and invoke procedure in OracleAS Integration InterConnect as shown in Figure A–8.

Figure A–8 BPEL Process Sending a Request and Receiving a Reply



When a BPEL process sends a request to InterConnect and receives a reply, the following steps are performed:

1. The BPEL Plug-in receives a message from a BPEL process. It checks the value of `ICMsgType` to find whether the message should be sent as a simple message or a request message.
2. The BPEL Plug-in converts the message into an InterConnect message object.
3. The BPEL Plug-in uses the local cache of adapter metadata to find a list of the recipients of the message and adds it to the recipient list of the message.
4. The BPEL Plug-in adds a request specific header to the message so that the receiving adapter can identify it is a request message. It also adds a `correlation`

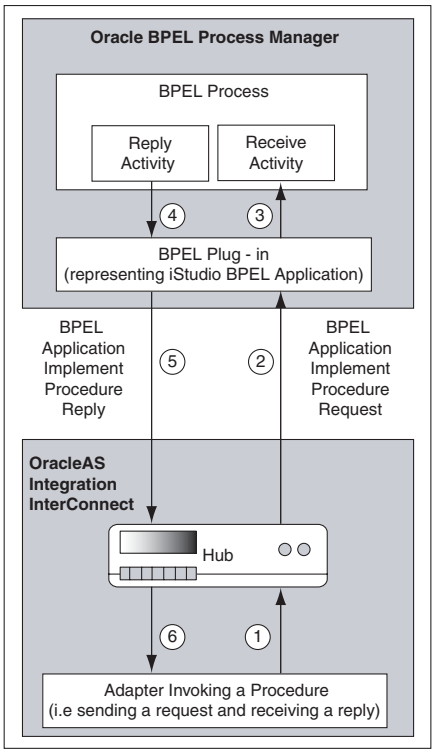
`id` element in the message header and this element is used to map a response to its request.

5. The BPEL Plug-in receives the response from InterConnect and translates it to the XMLRecord object. The response for a request contains the `correlation id` in its header. The BPEL Plug-in extracts the `correlation id` element and puts it in the inbound header of the message, which will be delivered to message endpoints.
6. The BPEL Plug-in invokes the `onMessage` method of an endpoint to deliver the received messages to a BPEL process.

Scenario 2: BPEL Process Receiving a Request and Sending a Reply

In this scenario, the BPEL process receives a request from InterConnect and sends a reply to InterConnect synchronously. This interaction involves the receive and reply activity in Oracle BPEL Process Manager and the implement procedure in OracleAS Integration InterConnect as shown in Figure A–9.

Figure A–9 BPEL Process Receiving a Request and Sending a Reply



When the BPEL process receives a request from OracleAS Integration InterConnect and sends a reply, the following steps are performed:

1. The BPEL Plug-in receives the request message from InterConnect.
2. The BPEL Plug-in converts the message into the XMLRecord object by Oracle BPEL Process Manager. The request message contains the `correlation id` in its header. The BPEL-IC adapter extracts the `correlation id` and puts it in the inbound header of the message, which will be delivered to the message endpoints.
3. The BPEL Plug-in invokes the `onMessage` method to deliver the received message to a BPEL process.
4. The BPEL Plug-in receives the response from BPEL process through WSIF handler using Java 2 Connector (J2C) extensions.

5. The BPEL Plug-in converts the received response to an InterConnect message object. It also identifies the list of recipients for this response message and adds them to the recipient list of the message.
6. The BPEL Plug-in adds a response specific header to the message so that the receiving adapter can identify it is a response message. It also adds a `correlation id` element in the message header.
7. The BPEL Plug-in adds the response message to the InterConnect hub queue using the Oracle JMS APIs.

A.2.4 Limitations of the BPEL Plug-in Feature

When deploying a BPEL process from standalone BPEL server to a middle-tier BPEL server on same host, you need to manually update the `port` parameter in the `bpel.xml` file. The reason being that standalone installation and the middle-tier insallation use different port numbers.

However, when deploying a BPEL process from standalone BPEL server to a middle-tier BPEL server on a different host, you need to manually update the `port` and the `host` parameters in the `bpel.xml` file. This is required because the host and port of the BPEL server on which the BPEL process was developed might be different from the one where it is getting moved.

A.3 Configuration

Perform the following one-time configuration tasks for the integration scenario between Oracle BPEL Process Manager and OracleAS Integration InterConnect:

- In Oracle BPEL Process Manager, add JNDI locations of connection factories for the various InterConnect hubs along with the connection information and application name in `oc4j-ra.xml` of the deployed BPEL Plug-in. Edit the `oc4j-ra.xml` file located in *`ORACLE_BPELHOME`*`/integration/bpelpm/orabpel/system/appserver/OC4J/J2EE/home/application-deployements/default/ICAdapter` for the following changes:
  - The location of the connection factory.

    ```
    <connector-factory location="eis/ICAdapter" connector-name="Interconnect Adapter">
    ```

  - The name of the application.

    ```
    <config-property name="applicationName" value="BPELServer"/>
    ```

  - The path of the repository.

    ```
    <config-property name="connectionString"
    value="jdbc:oracle:thin:@HostName:PortNo:SID"/>
    ```

  - The user name of the repository.

    ```
    <config-property name="userName" value="ichub904"/>
    ```

  - The password of the repository.

    ```
    <config-property name="password" value="ichub904"/>
    ```

  - The name of the repository.

    ```
    <config-property name="repoName" value="InterConnectRepository"/>
    ```

- In the `application.xml` file located in the *`BPEL_HOME`*`/integration/orabpel/system/appserver/oc4j/j2ee/home/config` directory, specify the path of the `oai.jar` file of the Interconnect to the BPEL server classpath. For example,

```
    <library
path="C:\oracle\ORACLEMIDDLETIER\integration\interconnect\lib\oai.jar"/>
```

- Specify whether your InterConnect repository is CORBA-based or RMI-based in the `pc.properties` file located in the *`OracleBPELPMHOME`*`\integration\orabpel\system\services\config\` directory.

  If your InterConnect repository is CORBA-based then specify the value of `oracle.tip.adapter.ic.repoConnMode` parameter in the `pc.properties` file as `CORBA` else specify the value as `RMI`. The default value of this parameter is `RMI`.

  Note: From InterConnect release 10.1.2.0.1, the repository is RMI-based. Prior to InterConnect release 10.1.2.0.1, the repository is CORBA-based.

  After configuration, you need to restart the InterConnect repository.

  Note: For using BPEL Plug-in with Oracle JDeveloper, the Oracle BPEL Process Manager should be up and running. This is because the servlet which serves the URLs in WSIL browser is deployed in the BPEL Server and will not work if the BPEL server is not running.

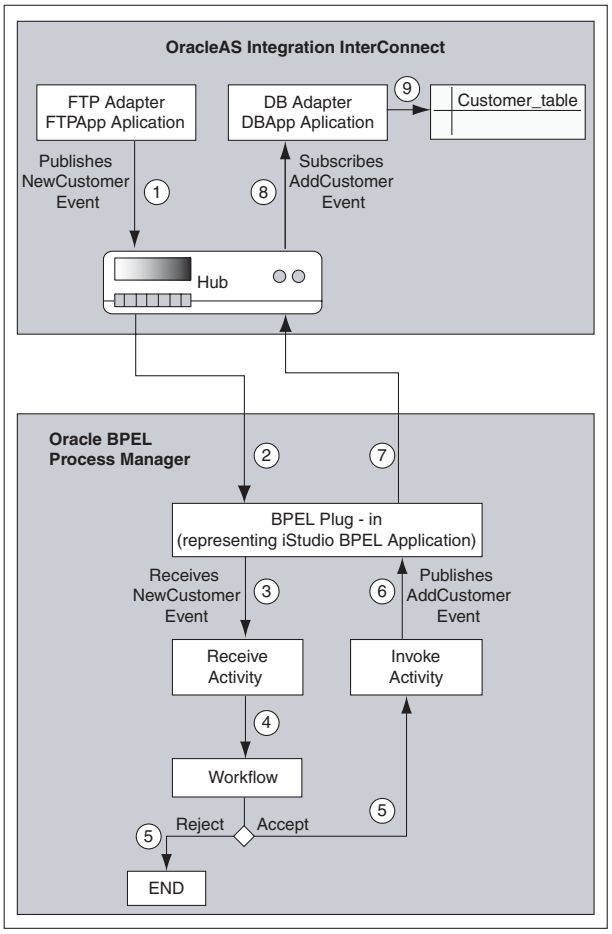
A.4 Use Case

This section describes how to design and execute the following integration scenario:

1. OracleAS Integration InterConnect FTP adapter publishes a `NewCustomer` event to the InterConnect hub queue.
2. A BPEL process subscribes to the `NewCustomer` event through BPEL Plug-in.
3. This event goes through a single-level human workflow approval.
4. After approval, the BPEL process publishes the `AddCustomer` event to the InterConnect hub queue.
5. OracleAS Integration InterConnect Database adapter subscribes to the `AddCustomer` event and inserts the approved customer data in the database.

The figure provides an overview of this scenario.

Figure A–10 Scenario Overview



This scenario consists of the following steps:

- Pre-Requisites
- Creating a Common View
- Creating a Publishing Event for FTP Application
- Creating a Subscribing Event for BPEL Server Application
- Creating a Publishing Event for BPEL Server Application
- Creating a Subscribing Event for DBAPP application
- Deploying the Stored Procedure Generated for Subscribing Event of DBAPP Application
- Starting Oracle BPEL Server and JDeveloper BPEL Designer
- Creating a Receive Activity to Receive NewCustomer Event
- Designing the Human Workflow System
- Creating a Invoke Activity to send the AddCustomer Event
- Run Time

A.4.1 Pre-Requisites

This example assumes that you are familiar with basic BPEL constructs, such as activities and partner links, and JDeveloper environment for creating and deploying BPEL Process.

Creating the XSD Files

You need to create the following files for this use case:

- `AddCustomerCommonView.xsd`: Use the following code to create the file

```
<?xml version="1.0"?>
  <schema targetNamespace=
   "http://oracle.com/interconnect/customer/event/addCustomer"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ns1="http://oracle.com/interconnect/customer/event/addCustomer">
         <element name="Customer" type="ns1:customerType"/>
         <complexType name="customerType">
            <sequence>
                 <element name="id" type="string"/>
                 <element name="title" type="string"/>
                 <element name="name" type="string"/>
                 <element name="email" type="string"/>
                 <element name="street" type="string"/>
                 <element name="city" type="string"/>
                 <element name="state" type="string"/>
                 <element name="zip" type="string"/>
            </sequence>
         </complexType>
  </schema>
```

- `NewCustomerCommonView.xsd`: Use the following code to create the file:

```
<?xml version="1.0"?>
  <schema targetNamespace=
   "http://oracle.com/interconnect/customer/event/newCustomer"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ns1="http://oracle.com/interconnect/customer/event/newCustomer">
         <element name="Customer" type="ns1:customerType"/>
         <complexType name="customerType">
            <sequence>
                 <element name="ID" type="string"/>
                 <element name="Title" type="string"/>
                 <element name="FirstName" type="string"/>
                 <element name="LastName" type="string"/>
                 <element name="Email" type="string"/>
                 <element name="Address" type="ns1:addressType"/>
            </sequence>
         </complexType>
         <complexType name="addressType">
            <sequence>
                 <element name="Street" type="string"/>
                 <element name="City" type="string"/>
                 <element name="State" type="string"/>
                 <element name="Zip" type="string"/>
            </sequence>
         </complexType>
  </schema>
```

- `NewCustomerFileAppView.xsd`:

```
<?xml version="1.0"?>
  <schema targetNamespace=
   "http://oracle.com/interconnect/customer/event/newCustomer"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ns1="http://oracle.com/interconnect/customer/event/newCustomer">
         <element name="Customer" type="ns1:customerType"/>
         <complexType name="customerType">
            <sequence>
                 <element name="ID" type="string"/>
                 <element name="Action" type="string"/>
                 <element name="Name" type="string"/>
                 <element name="Addr" type="ns1:addressType"/>
            </sequence>
         </complexType>
         <complexType name="addressType">
            <sequence>
                 <element name="Street" type="string"/>
                 <element name="City" type="string"/>
                 <element name="State" type="string"/>
                 <element name="Zip" type="string"/>
            </sequence>
         </complexType>
  </schema>
```

Creating the DBAPP Schema

In addition, run the following SQL script:

```
/* ***********************************
   Create DBAPP Schema
   *********************************** */
CREATE USER          DBAPP
PROFILE              DEFAULT
IDENTIFIED BY        WELCOME1
DEFAULT TABLESPACE   USERS
TEMPORARY TABLESPACE TEMP
ACCOUNT UNLOCK;
/* ***********************************
   Grant Privileges to DBAPP Schema
   *********************************** */
GRANT RESOURCE               TO DBAPP;
GRANT CONNECT                TO DBAPP;
GRANT EXECUTE ON  DBMS_AQ    TO DBAPP;
GRANT EXECUTE ON  UTL_RAW    TO DBAPP;
GRANT EXECUTE ON  DBMS_AQADM TO DBAPP;
GRANT EXECUTE ON  DBMS_AQIN  TO DBAPP;
GRANT EXECUTE ANY PROCEDURE  TO DBAPP;

/* ***********************************
   Create DBAPP.Customer Table
   *********************************** */
CREATE TABLE DBAPP.Customer_table
 (
   CustomerId     NUMBER NOT NULL,
   FirstName      VARCHAR2(30),
   LastName       VARCHAR2(30),
   Title          VARCHAR2(30),
   BusinessName   VARCHAR2(255),
   CustomerType   VARCHAR2(1),
   EmailAddress   VARCHAR2(255),
```

```
   Address        VARCHAR2(30),
   City           VARCHAR2(300),
   State          VARCHAR2(30),
   Zip            NUMBER
 )
/
```

In addition, your Oracle BPEL Process Manager should be configured for use with BPEL Plug-in. See Section A.3, "Configuration" for more information.

A.4.2 Creating a Common View

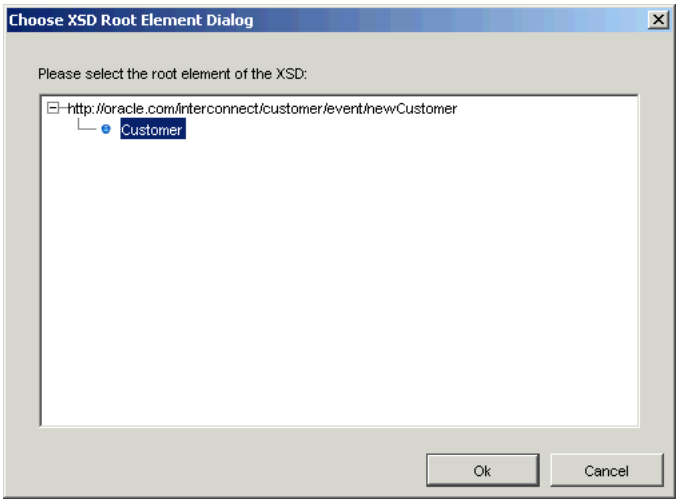
To create a common view in iStudio:

1. From the File menu, select New Project. The New Project dialog box is displayed.
2. Enter `ICInteroperability` in the **Project Name** field and click OK. The Hub Information dialog box is displayed.
3. Enter information in the following fields:
   - Hub database username: The name of the hub database user.
   - Hub database password: The password associated with the hub database user.
   - Hub database url: The URL of the database in the following format:

     ```
     machine name:port number:database sid
     ```

4. In iStudio, click **ICInteroperability**, **Common View**, and then click **Business Object**.
5. Right-click **Business Object** and select **New**. The Create Business Object dialog box is displayed.
6. Enter `Customer` in the **Business Object Name** field.
7. In iStudio, click **Customer** and then click **Event**.
8. Right-click **Event** and select **New**. The Create Event dialog box is displayed.
9. Perform the following steps in the Create Event dialog box:.
   a. Enter `NewCustomer` in the **Event Name** field.
   b. Click **Import**, **XML**, **XSD** and then **Local File System**.
   c. Select `NewCustomerCommonView.xsd` and click **OK**. The Choose Root Element dialog box is displayed.
   d. Select **Customer** as root element of the XSD as shown in Figure A–11 and then click **OK**.

Figure A–11 Choose Root Element Dialog box



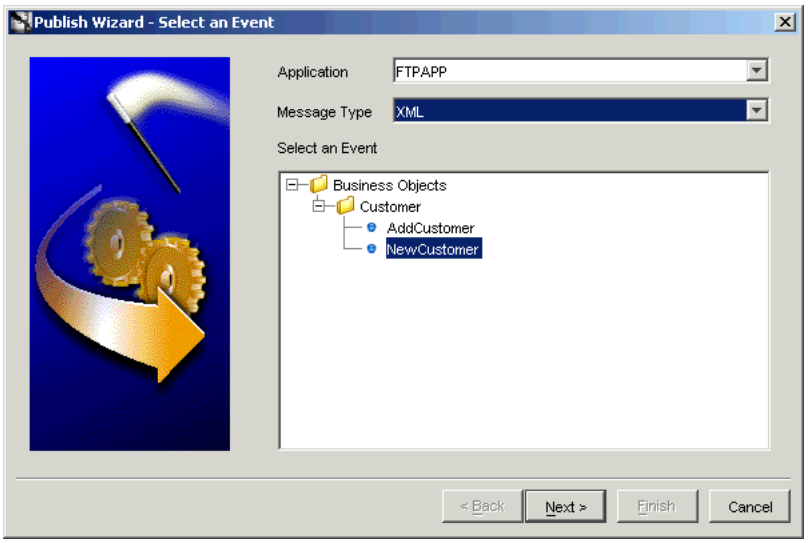
10. Click **Save** in the Create Event dialog box.
11. Repeat steps 7 to 10 to create another event `AddCustomer` by using the file `AddCustomerCommonView.xsd`.

A.4.3 Creating a Publishing Event for FTP Application

To create an FTP application that publishes the `NewCustomer` event:

1. In iStudio, right-click **Applications**, and select **New**. The Create Application dialog box is displayed.
2. Enter the `FTPAPP` in the **Application Name** field and click **OK**.
3. In iStudio, click **Applications**, **FTPAPP**, and then **Published Events**.
4. Right-click **Published Events** and select **New**. The Select an event dialog box is displayed.
5. Perform the following steps as shown in Figure A–12:
    1. Select **XML** from the **Message Type** list.
    2. Select **NewCustomer** from the **Select an Event** list, and then click **Next**. The Publish Wizard- Define Application View dialog box is displayed.

Figure A–12 Publish Wizard- Select an Event Dialog Box



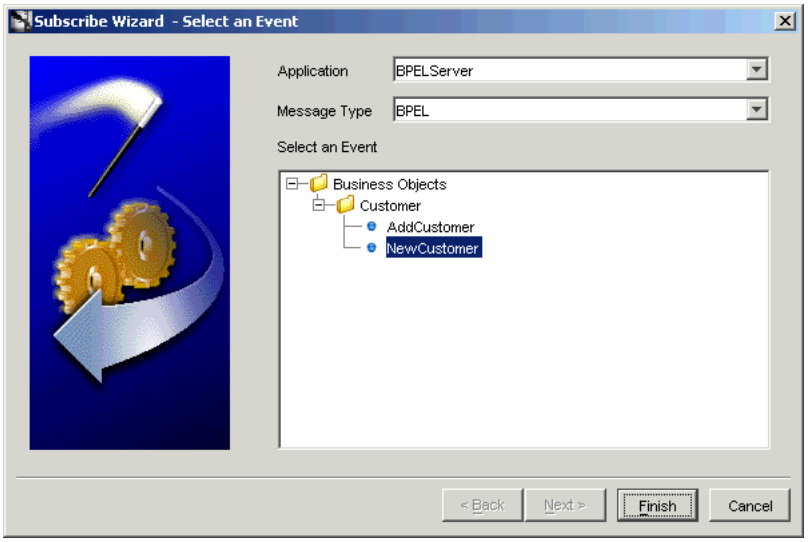
6. Click **Import**, **XML**, **XSD** and then **Local File System**.
7. Select `NewCustomerFileAppView.xsd` and click **OK**. The Choose XSD Root Element dialog box is displayed.
8. Select **Customer** as root element and click **OK**.
9. Click **Next**. The Define Mapping dialog box is displayed.
10. Click **Define Mapping**. The Mapping Parameters dialog box is displayed.
11. Perform the following:
    a. In FTPAPP View, click ***ns1:*NewCustomer**, **Customer** and then **id**.
    b. In Transformations, select **CopyFields**.
    c. In Common View, click ***ns1:*NewCustomer**, **Customer** and then **id**.
    d. Click **Add**.
    e. In FTPAPP View, click ***ns1:*NewCustomer**, **Customer** and then **addr**.
    f. In Transformations, select **CopyFields**.
    g. In Common View, click ***ns1:*NewCustomer**, **Customer** and then **address**.
    h. Click **Add**.
    i. In FTPAPP View, click ***ns1:*NewCustomer**, **Customer** and then **name**.
    j. In Transformations, select **ExpandFields**.
    k. In Common View, click ***ns1:*NewCustomer**, **Customer**.
    l. Press the SHIFT key and select **FirstName** and **LastName**.
    m. In the Additional Parameters section, enter comma "," in the **Value** field as shown in
    n. Click **Add**.
    o. Click **OK**.
12. Click **Finish** in the Define Mapping dialog box.

A.4.4 Creating a Subscribing Event for BPEL Server Application

To create a BPEL application that subscribes to the `NewCustomer` event:

1. In iStudio, right-click **Applications**, and select **New**. The Create Application dialog box is displayed.
2. Enter the `BPELServer` in the **Application Name** field and click **OK**.
3. Click **BPELServer**, and then click **Subscribed Events**.
4. Right-click **Subscribed Events** and select **New**. The Subscribe Wizard- Select an event dialog box is displayed.
5. Select **BPEL** from the **Message Type** list.
6. Select **NewCustomer** from the **Select an Event** list.

Figure A–13 Subscribe Wizard - Select an Event Dialog Box



7. Click **Next**.
8. Click **Finish**.

A.4.5 Creating a Publishing Event for BPEL Server Application

To create a BPEL application that publishes the `AddCustomer` event from Oracle BPEL Process Manager to OracleAS Integration InterConnect.

1. In iStudio, click **BPELServer**, and then **Published Events**.
2. Right-click **Published Events** and select **New**. The Publish Wizard- Select an event dialog box is displayed.
3. Select **BPEL** from the **Message Type** list.
4. Select **AddCustomer** from the **Select an Event** list.
5. Click **Finish**.

Note: You should create one application for each BPEL server instance in each InterConnect hub and group all the events for this instance in this application.

A.4.6 Creating a Subscribing Event for DBAPP application

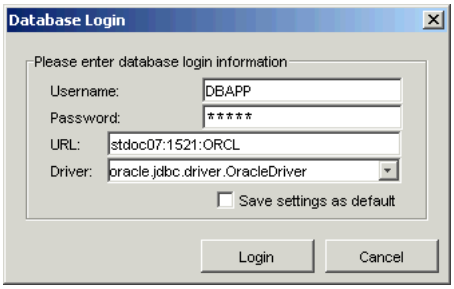
To create a database application that subscribes to the `AddCustomer` event:

1. In iStudio, right-click **Applications**, and **New**. The Create Application dialog box is displayed.
2. Enter `DBAPP` in the **Application Name** field and click **OK**.
3. Click **Applications**, **DBAPP**, and then right-click **Subscribed Events**.
4. Select **New**. The Select an Event dialog box is displayed.
5. Perform the following steps in the Select an Event dialog box:
   a. Select **Database** from the **Message Type** list.
   b. Select **AddCustomer** from the **Select an Event** list and click **Next**. The Subscribe Wizard- Define Application View dialog box is displayed.
6. Click **Import,** and click **Database**. The Database Login dialog box is displayed.
7. Perform the following steps in the Database Login dialog box:
   - Enter the name of the database user in **Username** field.
   - Enter the password associated with the database user in the Password field.
   - Enter the database URL in following format in the **URL** field:

     ```
     computer name:port number:database sid
     ```

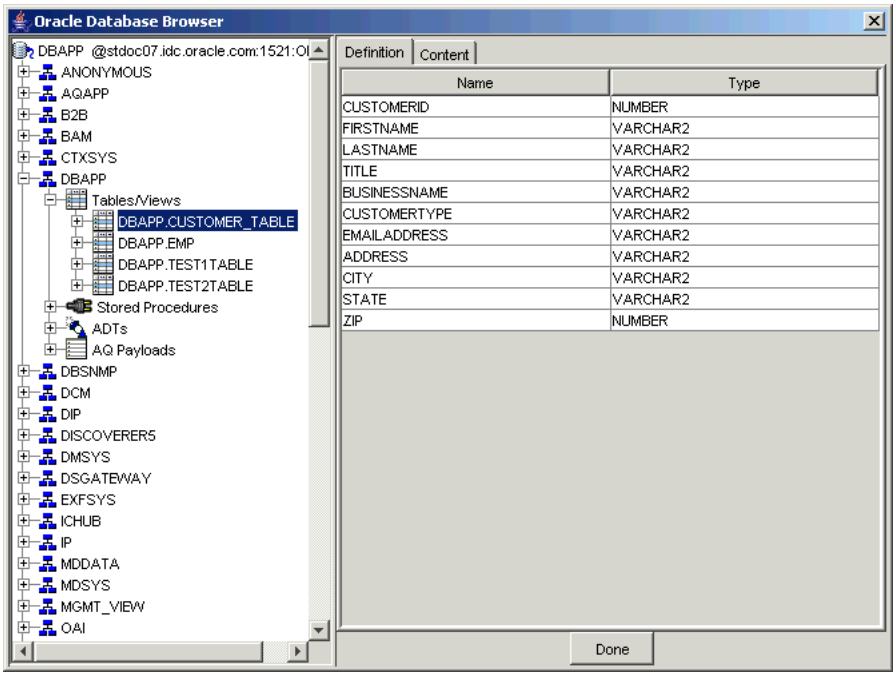
   - Select **oracle.jdbc.driver.OracleDriver** from the **Driver** list.
   - Click **Login**. The Define Application View dialog box is displayed.

Figure A–14 Database Login



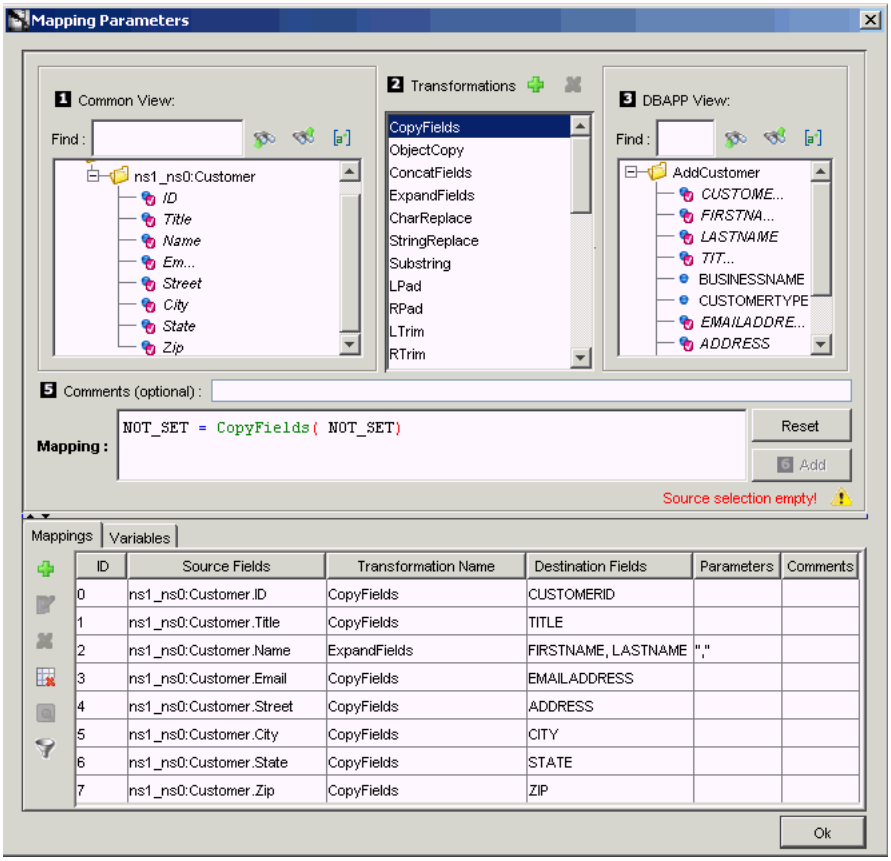
8. Click **Next**. The Oracle Database Browser dialog box is displayed.

Figure A–15 Database Browser



9. Click **DBAPP**, **Table/Views**, **DBAPP.CUSTOMER_TABLE** and then click **Done**. The Subscribe Wizard- Define Application View dialog box is displayed.
10. Click **Next**. The Define Mapping dialog box is displayed.
11. Click **Define Mapping**. The Mapping Parameters dialog box is displayed.

Figure A–16 Mapping Parameters



12. Perform the following steps in the Mapping Parameters dialog box:

a. In **Common View**, click **AddCustomer**, ***ns1_ns0*Customer**, and click **ID**.

b. In **Transformations**, select **CopyFields**.

c. In **DBAPP View**, click **AddCustomer**, and click **CUSTOMERID**.

d. Click **Add**.

e. Repeat steps 1 to 3 to map the following fields in the Common View to DBAPP View:

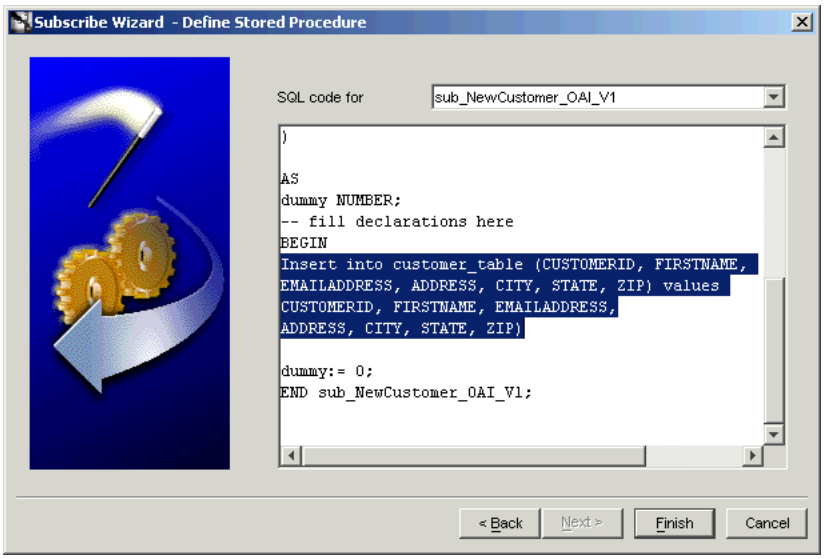
Common View Fields	DBAPP View Fields
Title	TITLE
Email	EMAIL
Street	ADDRESS
City	CITY
State	STATE
Zip	ZIP

f. Map the **Name** field of the Common View to the **First Name** and the **Last Name** field of the DBAPP View by selecting the **Expand** transformation. Specify comma " , " as the delimiter.

g. Click **OK**.

13. Click **Next** in the Define Mapping dialog box. The Define Stored Procedure dialog box is displayed.

Figure A–17 Subscribe Wizard- Define Stored Procedure



14. Perform the following steps in the Define Stored Procedure dialog box:

a. Select **sub_AddCustomer_OAI_V1** from the **SQL code for** list.

b. In the `BEGIN` section of the `sub_AddCustomer_OAI_V1` procedure, add the following code:

```
Insert into customer_table (CUSTOMERID, FIRSTNAME, EMAILADDRESS, ADDRESS,
CITY, STATE, ZIP) values (CUSTOMERID, FIRSTNAME, EMAILADDRESS,
ADDRESS, CITY, STATE, ZIP)
```

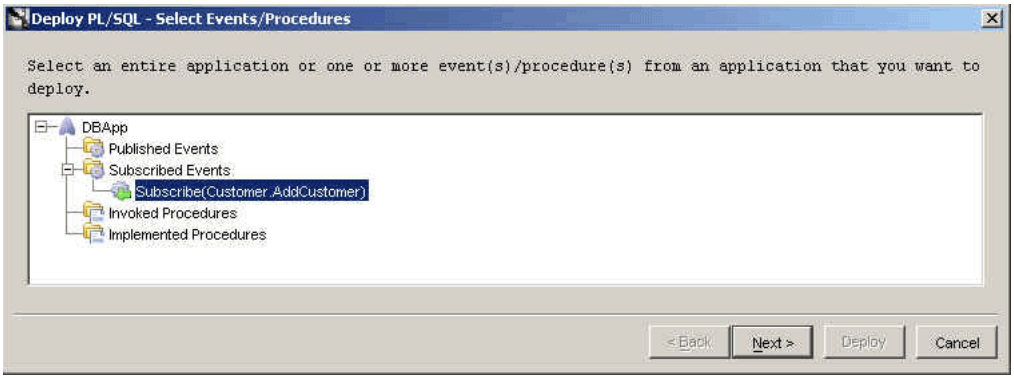
c. Click **Finish**.

A.4.7 Deploying the Stored Procedure Generated for Subscribing Event of DBAPP Application

To deploy the stored procedure generated for subscribing event of DBAPP Application:

1. In iStudio, click the **Deploy** tab.
2. Click **Application**, right-click **DBAPP**, and select **Deploy PL/SQL**. The Deploy PL/SQL- Select Events/Procedures dialog box is displayed.

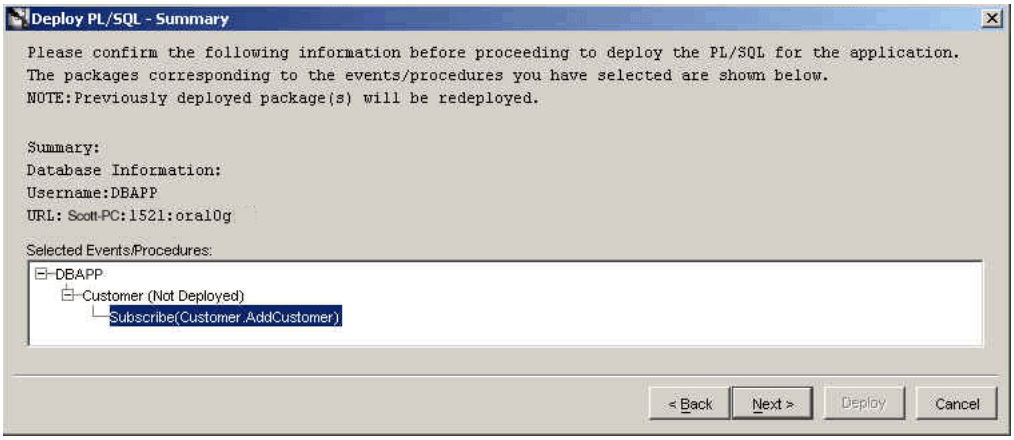
Figure A–18 Deploy PL/SQL- Select Events/ Procedures Dialog Box



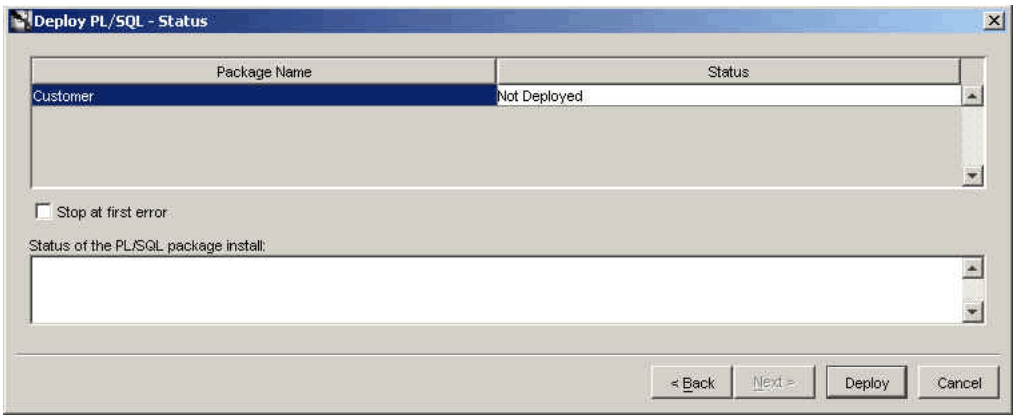
3. Click **DBAPP**, **Subscribed Events**, and then **Subscribe(Customer.AddCustomer)**.
4. Click **Next**. The Deploy PL/SQL- Database Information dialog box is displayed.
5. Perform the following:
   - Enter the name of the database user in the **Database username** field.
   - Enter the password associated with the database user in the **Database password** field.
   - Enter the database URL in following format in the **Database URL** field:

     ```
     computer name:port number:database sid
     ```

   - Click **Next**. The Deploy PL/SQL- Summary dialog box is displayed.



6. Click **DBAPP**, **Subscribed Events**, and **Subscribe(Customer.AddCustomer)**, and then click **Next**. The Deploy PL/SQL- Status dialog box is displayed.



7. Click **Customer** in the **Package Name** and click **Deploy**. The generated PL/SQL is deployed for the selected application, event or procedure.

A.4.8 Starting Oracle BPEL Server and JDeveloper BPEL Designer

Ensure that the BPEL Server and JDeveloper BPEL Designer are started. To start the BPEL PM Server, select **Start**, **All Programs**, **Oracle -** *Oracle_Home*, **Oracle BPEL Process Manager** *version_number*, **Start BPEL PM Server**.

To start the JDeveloper BPEL Designer, select **Start**, **All Programs**, **Oracle -** *Oracle_Home*, **Oracle BPEL Process Manager** *version_number*, **JDeveloper BPEL Designer**.

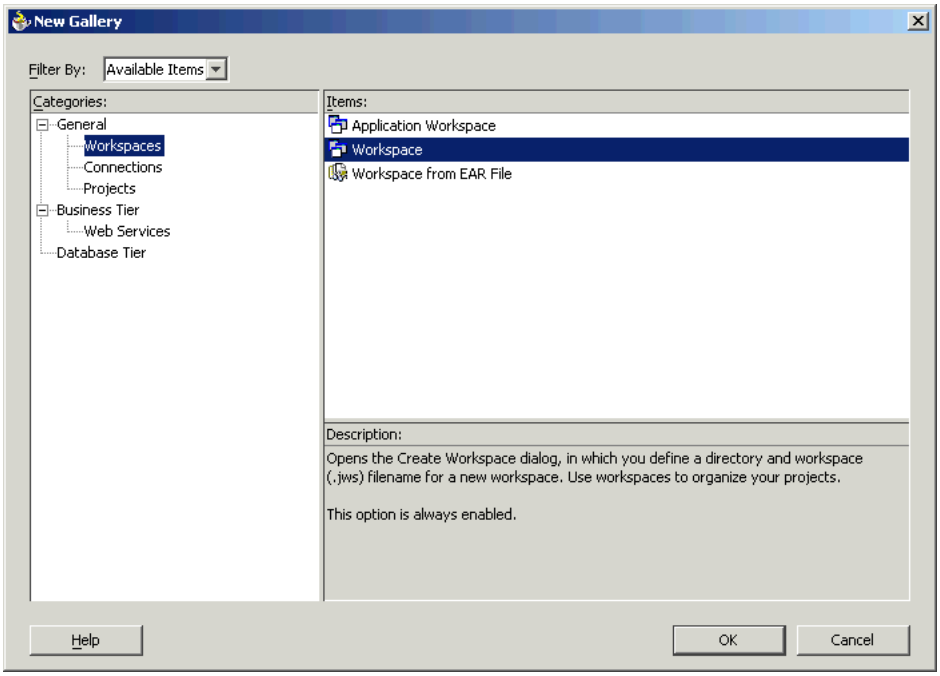
> **See Also:** Oracle BPEL Process Manager Developer's Guide for information on starting and stopping BPEL Server.

A.4.9 Creating a Receive Activity to Receive NewCustomer Event

To create a receive activity:

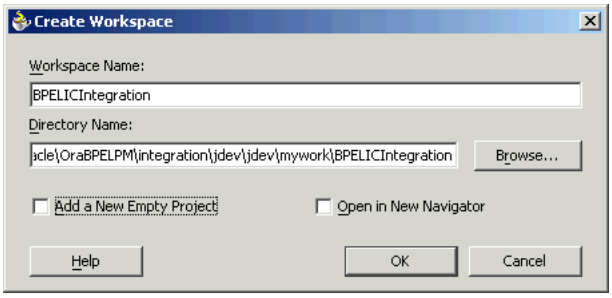
1. Right-click **Applications** and select **New**. The New Gallery dialog box is displayed.

Figure A–19 New Gallery Dialog Box



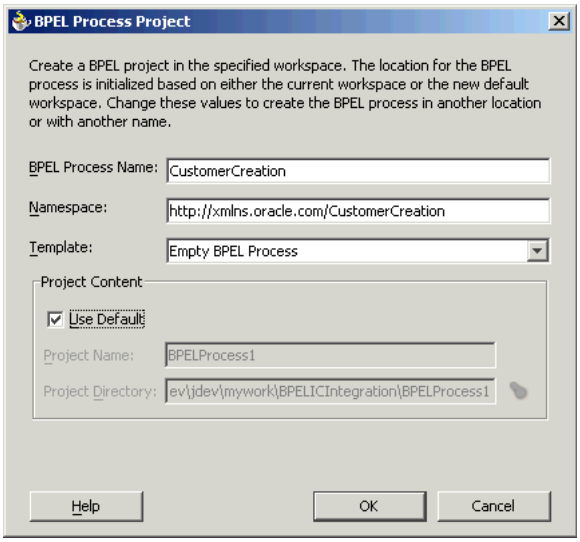
2. Double-click **Workspace** from Items and click **OK**. The Create Workspace dialog box is displayed.

Figure A–20 Create Workspace Dialog Box



3. Enter `BPELICIntegration` in the **Workspace Name** field and accept the default path in the **Directory Name** field.
4. Deselect **Add a New Empty Project**.
5. Click **OK**.
6. Right-click **BPELICIntegration** in the Applications Navigator section of the designer window. If the Applications Navigator section does not currently appear, select **Application Navigator** from the **View** main menu.
7. Select New **Project**.
8. Select **BPEL Process Project** from **Items** and click **OK**. The BPEL Process Project dialog box is displayed.
9. Enter `CustomerCreation` in the BPEL Process Name field.

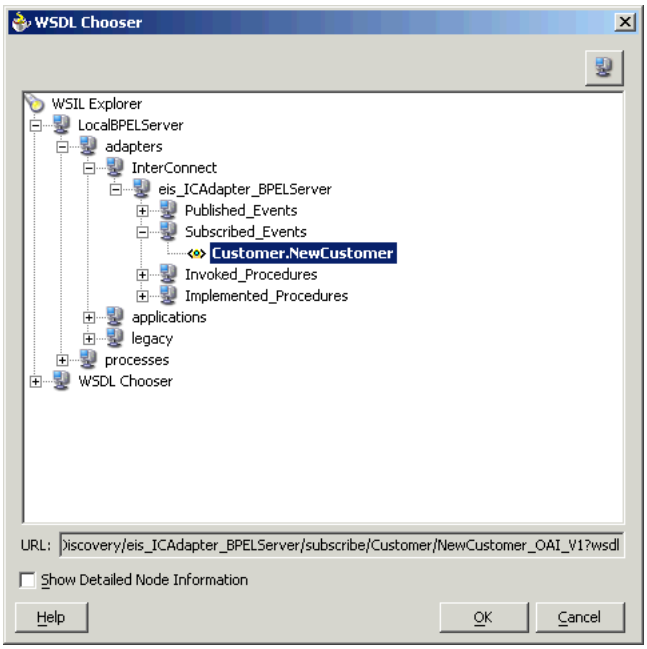
Figure A–21 BPEL Process Project Dialog Box



10. Select **Empty BPEL Process** from **Template** list.
11. Select the **Use Default** button.
12. Click **OK**.

    The `bpel.xml`, `CustomerCreation.bpel`, `CustomerCreation.wsdl`, and `build.xml` files are created.
13. Ensure that Process Activities is selected in the drop-down list of the Component Palette section in the upper right section of JDeveloper BPEL Designer.
14. Drag and drop a PartnerLink activity onto the right side of the designer window anywhere beneath the header Partner Links. Note that both sides of the designer window have sections in which partner links can be placed. The Create Partner dialog box is displayed.
15. Perform the following tasks to create a partner link as shown in Figure A–23:
    a. Enter `ReceiveFromIC` in the **Name** field.
    b. Click the flashlight (the second icon from the left named WSIL Browser) to access the WSDL Chooser dialog box.
    c. Select *`LocalBPELServer`*, **adapters**, **InterConnect**, **eis-ICadapter-BPELServer**, **Subscribed Events**, **Customer.NewCustomer** as shown in the Figure A–22 and click **OK**.

Figure A–22 WSDL Chooser

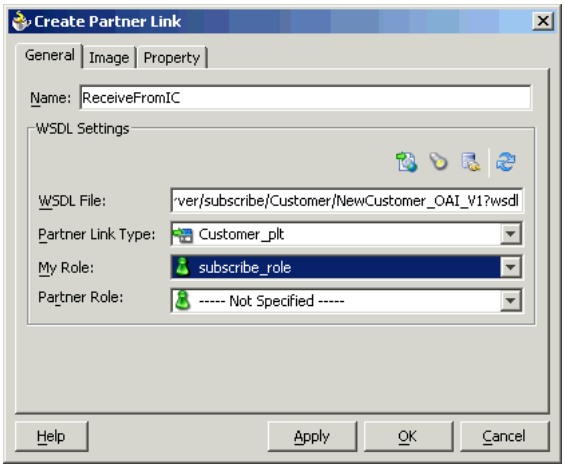


d. Select **subscribe_role** from the **My Role** list.

e. Click **OK**.

After performing these steps, the Create Partner Link dialog box would appear as shown in Figure A–23.

Figure A–23 Create Partner Link Dialog box



16. Drag and drop a **Receive** activity from the **Component Palette** to the **Drop Activity Here** section.
17. Double-click the **Receive** activity. The Receive dialog box is displayed.
18. Perform the following:

    a. Enter `ReceiveNewCustomer` in the **Name** field.

    b. Select `ReceivefromIC` in the **Partner Link** list.

c. Click the first icon to the right of the **Variable** field. This is the automatic variable creation icon.

d. Click **OK** on the Create Variable dialog box that appears. A variable named `ReceiveNewCustomer_subscribe_InputVariable` is automatically created in the **Input Variable** field.

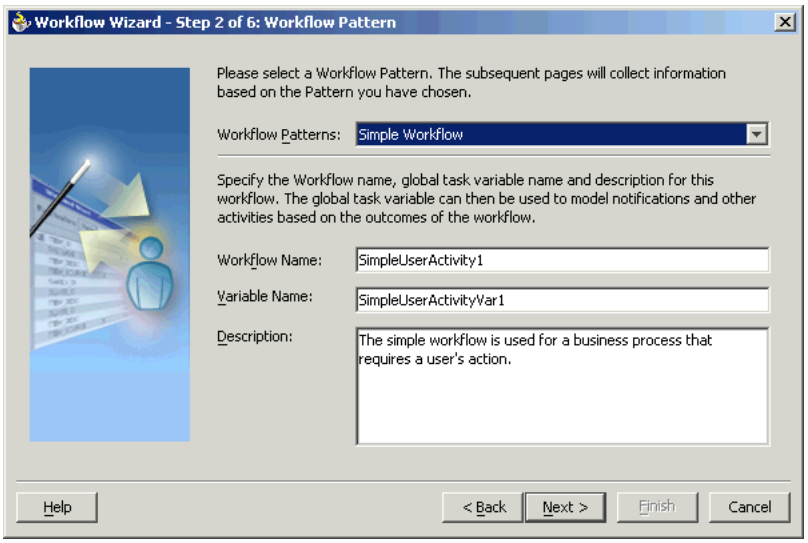
e. Select **Create Instance** and click **OK**.

A.4.10 Designing the Human Workflow System

To add user tasks to handle the manual approval or rejection of a create customer request:

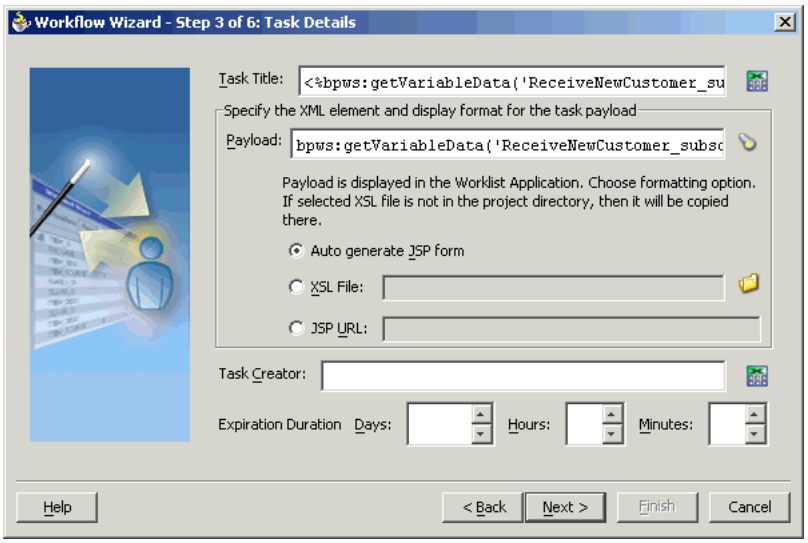
1. Drag and drop a User Task activity from the Component Palette section to below the **ReceiveNewCustomer** activity. The Workflow Wizard is displayed.
2. Leave the **Create New Workflow** check box selected and click **Next**. The Workflow Pattern window is displayed.
3. Select **Simple Workflow** from the **WorkFlow Patterns** as shown in Figure A–24.

Figure A–24 Workflow Wizard



4. Click **Next**. The Task Details window is displayed as shown in Figure A–25.

Figure A–25 Workflow Wizard: Task Details

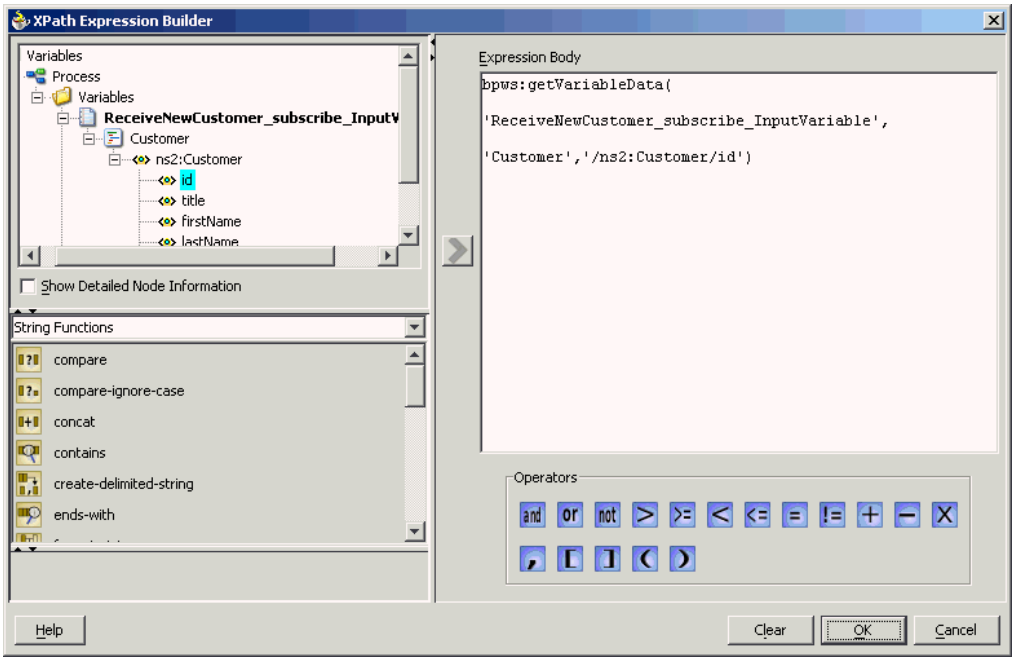


5. Perform the following:

   a. Click the icon to the right of the **Task Title** field shown in Figure A–25 to display the XPath Expression Builder window.

   b. Select **Variables**, **Process**, **Variables**, **ReceiveNewCustomer_subscribe_InputVariable**, **Customer**, **ns2:Customer**, **id** as shown in Figure A–26.

Figure A–26 XPath Expression Builder



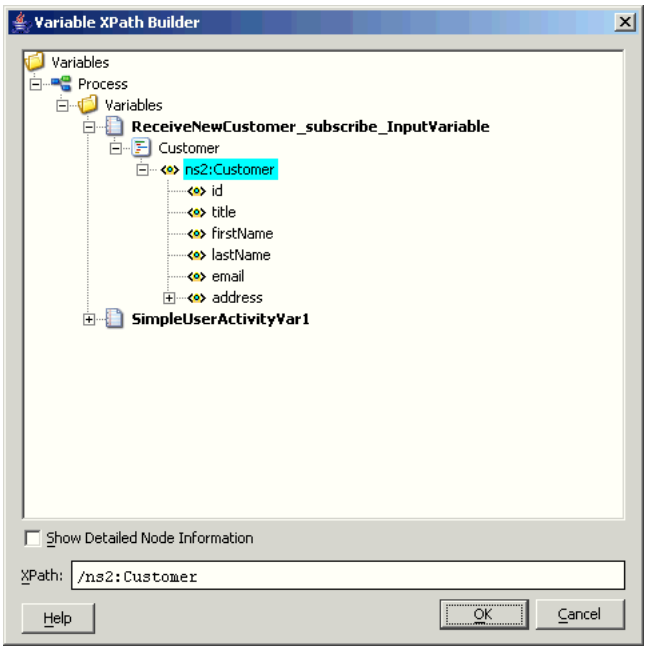
   c. Click the right arrow.

   d. Click OK.

6. In the Task Details window shown in Figure A–25, click the flashlight icon to the right of the **Payload** field to display the Variable XPath Builder window.

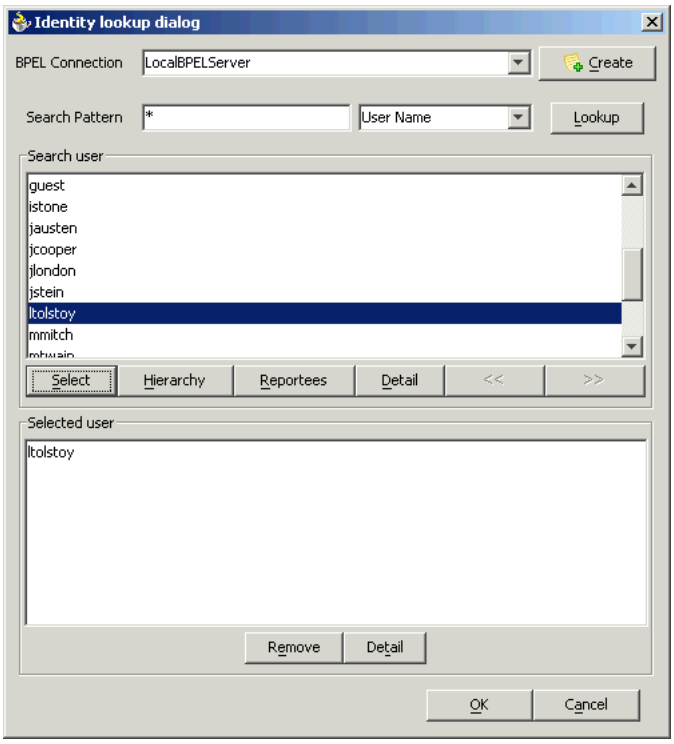
7. Select **Variables**, **Process**, **Variables**, **ReceiveNewCustomer_subscribe_InputVariable**, **Customer**, ***ns2***:**Customer** as shown in Figure A–27.

Figure A–27 Variable XPath Builder



8. Click **OK**.
9. Leave the Task Creator field empty.
10. Click **Next** in the Task Details window. The Task Outcomes dialog box is displayed.
11. Leave the two default outcomes as **ACCEPT** and **REJECT**.
12. Click **Next**. The Task Notification dialog box is displayed.
13. Accept the default settings and click **Next**. The Assignees dialog box is displayed.
14. Click the flashlight icon to the right of the **Users** field. The identity lookup dialog box is displayed.
15. Click **LookUp**.
16. Select your user name in the search user and click **Select**.

Figure A–28 Identity Lookup Dialog



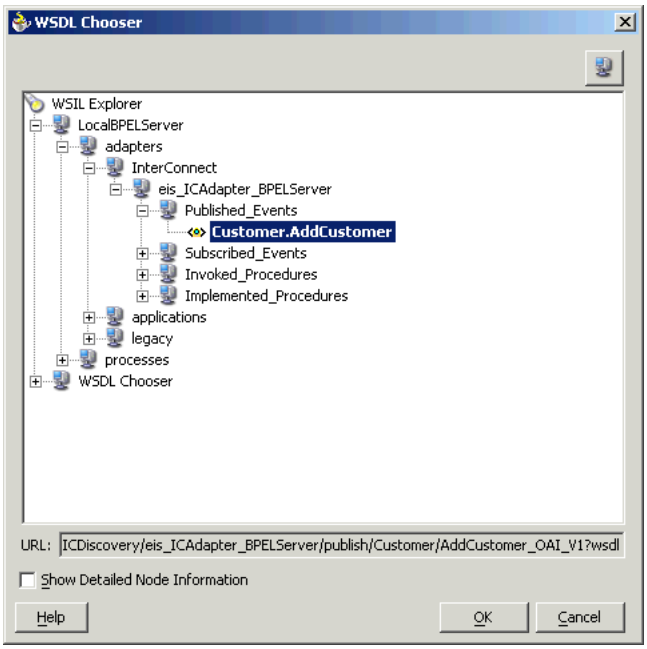
17. Click **OK**.
18. Click **Next**.
19. Click **Finish**.

A.4.11 Creating a Invoke Activity to send the AddCustomer Event

To create a invoke activity:

1. Drag and drop a PartnerLink activity onto the right side of the designer window anywhere beneath the header Partner Links. The Create Partner dialog box is displayed.
2. Perform the following:
    1. Enter `SendToIC` in the **Name** field.
    2. Click the flashlight (the second icon from the left named WSIL Browser) to access the WSDL Chooser dialog box.
    3. Select *`LocalBPELServer`*, **adapters**, **InterConnect**, **eis-ICadapter-BPELServer**, **Published Events**, **Customer.AddCustomer** as shown in the Figure A–30.

Figure A–29 WSDL Chooser

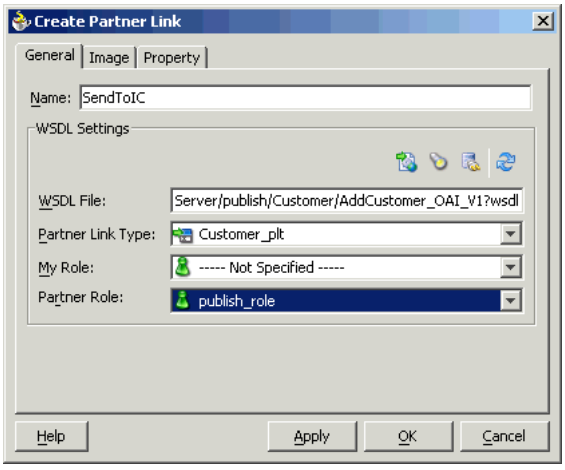


4. Click **OK**.
5. Select **publish_role** from the **Partner Role** list.
6. Click **OK**.

3. Drag and drop a **Invoke** activity from the **Component Palette** to the **<case Task Outcome is Accept>** section.
4. Double-click the **Invoke** activity. The Invoke dialog box is displayed.
5. Perform the following:
   a. Enter `SendApprovedCustomer` in the **Name** field.
   b. Select `SendToIC` in the **Partner Link** list.
   c. Click the first icon to the right of the **Input Variable** field. This is the automatic variable creation icon. The Create Variable dialog box is displayed.
   d. Click **OK**. A variable named `SendApprovedCustomer_publish_InputVariable` is automatically created in the **Input Variable** field.

   After performing these steps, the Create Partner Link dialog box would appear as shown in Figure A–30.

Figure A–30 Create Partner Link



6. Drag and Drop a Transform activity from BPEL Component Pallette before the Invoke step created in Step 3. Map the input variable corresponding to Receive activity created in Section A.4.9, "Creating a Receive Activity to Receive NewCustomer Event" to the input variable corresponding to the Invoke activity created in Step 3. Map all the fields of the customer payload to the target variable.

After designing the BPEL process, you need to compile and deploy the process to Oracle BPEL Server. If compilation and deployment are successful, you can run and manage the BPEL process from Oracle BPEL Console.

> **See Also:** *Oracle BPEL Process Manager Developer's Guide* for information on how to deploy a BPEL process.

A.4.12 Run Time

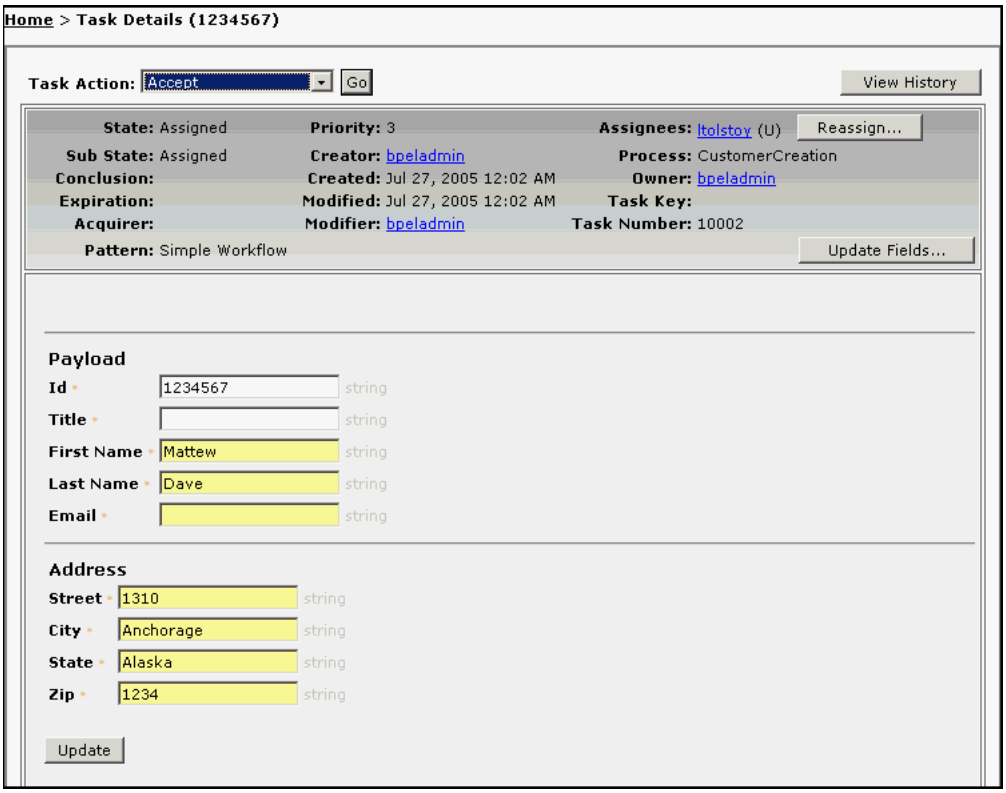
For run time, you need to configure the FTP adapter to poll an input directory where the incoming payload is to be copied. For information about how to configure the FTP adapter, refer to the *Oracle Application Server Integration InterConnect Adapter for FTP Installation and User's Guide*

During run time, copy an XML file containing information about the new customer, to the folder from which the InterConnect FTP adapter will read the file. Following example shows a sample input XML file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Customer xmlns="http://oracle.com/interconnect/customer/event/newCustomer">
  <ID xmlns="">1234</ID>
  <Action xmlns="">Action11</Action>
  <Name xmlns="">Name12</Name>
  <Addr xmlns="">
     <Street>Street13</Street>
     <City>City14</City>
     <State>State15</State>
     <Zip>4567</Zip>
  </Addr>
</Customer>
```

The FTP adapter sends the file to the OAI hub. From the OAI hub, this file is received by the `CustomerCreation` BPEL process. The process waits for the human approval as shown in Figure A–31.

Figure A–31 Oracle BPEL Worklist



You can accept or reject the request by using the BPEL Console. Once approved, the `CustomerCreation` BPEL process sends the `AddCustomer` message to the InterConnect hub queue. The InterConnect Database adapter receives the message and insert the new customer data into the `Customer_table` table.

> **See Also:** *Oracle BPEL Process Manager Developer's Guide* for information about how to use BPEL Console and BPEL Worklist.

B

Integration Scenario

This appendix provides an integration scenario and model based on a fictitious company, Acme, Inc. using OracleAS Integration InterConnect. It contains the following topics:

B.1 Integration Scenario Overview

Each division of Acme, Inc. has multiple Order Fulfillment Systems which are a legacy from various mergers and acquisitions. Maintaining the parts of these systems such as platforms, software, training, and so on is costly and time consuming for Acme. In addition, the lack of integration between the systems prevents business analysis at the enterprise level.

Acme has created a new centralized system, and the first phase of the integration project is to synchronize the purchase order information on one of the legacy systems with the new system.

B.1.1 The New Centralized System

The new order fulfillment system operates on an Oracle Database 10*g* and uses the OracleAS Integration InterConnect Database Adapter to communicate with the system.

B.1.2 The Legacy System

The legacy order fulfillment system operates on an Oracle8*i* database and uses the OracleAS Integration InterConnect Advanced Queuing Adapter to communicate with the system.

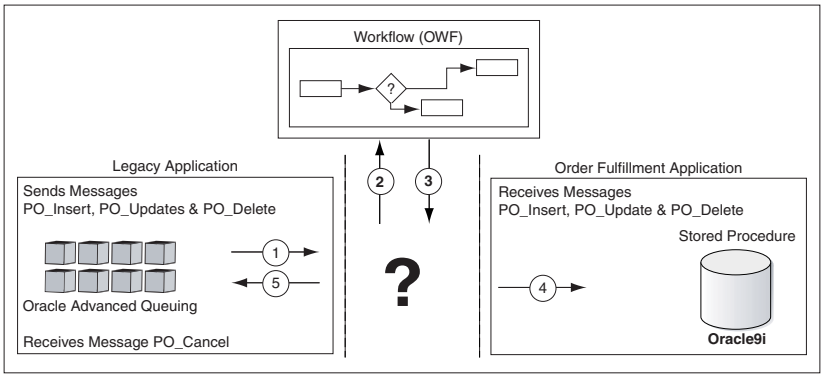
The Purchase Order table in this system has a database trigger to queue the changed records. OracleAS Integration InterConnect is configured to listen to that queue to accomplish the integration.

Note: There are many methods available to capture changes to a system. These methods include, but are not limited to, database triggers, interface tables, and database log files.

B.1.3 The Integration Scenario

Consider an organization that wishes to integrate its legacy system containing its purchase order tables with the new order fulfillment application running on Oracle Database 10*g*. Figure B–1 illustrates this integration scenario:

Figure B–1 Integration Scenario



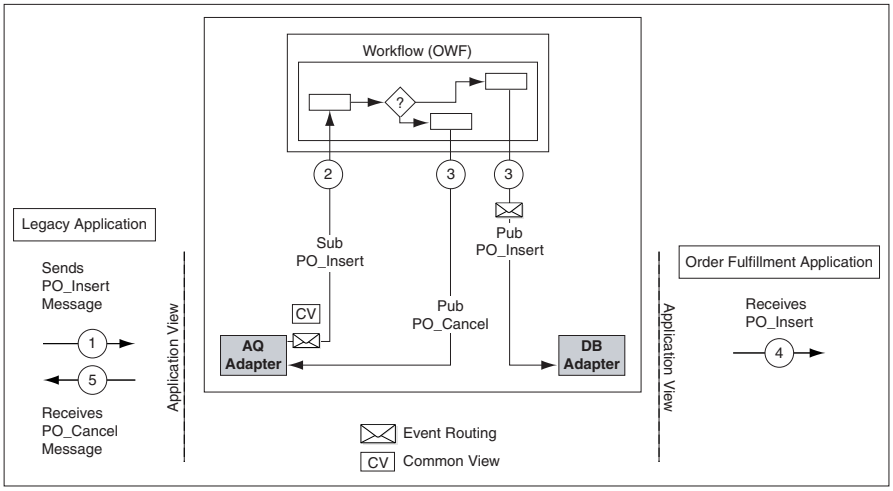
The first step in any integration scenario is to model the integration.

- Legacy System
  - Any change to the Purchase Order table in the legacy application is published using a database trigger. An administrator must approve all changes, such as insert, update, & delete before they are applied to the new order fulfillment System
- Order Fulfillment System
  - If a change is approved, then it is sent to the order fulfillment system. If a change is rejected, then a cancellation notification is sent to the legacy system.
- Additional Issues
  - The process must be non-intrusive. The user cannot alter the structure in either system.
  - Synchronization of the primary keys of each system must be maintained by the integration platform.
  - The integration must be scalable and support addition of systems.

B.2 Modeling the Integration

Figure B–2 illustrates how OracleAS Integration InterConnect integrates with the scenario in Figure B–1.

Figure B–2 Integration Modeling



How are we going to accomplish this task?

- Legacy Application publishes the insert, update, and delete messages to a queue. OracleAS Integration InterConnect Adapter for AQ (Advanced Queuing adapter) is used to send and receive messages to and from the queue.

  The Order Fulfillment Application uses a standard Oracle database and the OracleAS Integration InterConnect Adapter for DB (Database adapter).

- All messages are routed to Oracle Workflow to apply user-defined logic.

B.3 Implementing the Scenario

The following sections describe implementing the integration scenario using iStudio.

- Review Legacy System Database Trigger
- Create a Project
- Create the Common View Business Object
- Create Business Object Events
- Create Applications
- Create a Cross Reference Table
- Create Publish Events
- Subscribe to Events
- Create Content-based Routing
- Create an Oracle Workflow Process Bundle
- Deploy the Process Bundle to Oracle Workflow
- Creating Objects in Oracle Workflow for Modeling

B.3.1 Review Legacy System Database Trigger

The source system uses Oracle Database 10*g* Advanced Queuing to publish changes to the purchase order table. The user creates a database trigger on the purchase order table. When a record is updated, inserted, or deleted and then committed, the trigger

enqueues the appropriate payload. The OracleAS Integration InterConnect Advanced Queuing Adapter is configured to listen on this queue.

The following is an example of the code for the database trigger:

```
CREATE OR REPLACE TRIGGER AQAPP.ENQUEUE_PO
       AFTER INSERT OR DELETE OR UPDATE ON AQAPP.PURCHASE_ORDER FOR EACH ROW
DECLARE
       qname                   VARCHAR2(20)    := 'OUTBOUND_QUEUE';
       enqueue_options         DBMS_AQ.ENQUEUE_OPTIONS_T;
       message_properties      DBMS_AQ.MESSAGE_PROPERTIES_T;
       msgid                   RAW(16);
       recip_agent             SYS.AQ$_AGENT;
       raw_payload             RAW(32767);
       payload                 VARCHAR2(256);
BEGIN
  IF INSERTING THEN
    payload := '<?xml version="1.0" standalone="no"?>'        ||
      '<PO_Insert>' ||
      '<id>'        || :new.id           || '</id>'          ||
      '<action>'    || 'I'               || '</action>'      ||
      '<item>'      || :new.item         || '</item>'        ||
      '<amount>'    || :new.amount       || '</amount>'      ||
      '<quantity>'  || :new.quantity     || '</quantity>'    || '</PO_Insert>';

ELSIF DELETING THEN
    payload := '<?xml version="1.0" standalone="no"?>'        ||
      '<PO_Delete>'   ||
      '<id>'          || :old.id         || '</id>'          ||
      '<action>'      || 'D'             || '</action>'      || '</PO_Delete>';
ELSIF UPDATING THEN
    payload := '<?xml version="1.0" standalone="no"?>'        ||
      '<PO_Update>'   ||
      '<id>'          || :old.id          || '</id>'         ||
      '<action>'      || 'U'              || '</action>'     ||
      '<item>'        || :new.item        || '</item>'       ||
      '<amount>'      || :new.amount      || '</amount>'     ||
      '<quantity>'    || :new.quantity    || '</quantity>'   ||
      '<last_updated>'|| :new.last_updated|| '</last_updated>'|| '</PO_Update>';
END IF;

raw_payload := UTL_RAW.CAST_TO_RAW( payload );

DBMS_AQ.ENQUEUE(  queue_name            => qname
                 ,enqueue_options       => enqueue_options
                 ,message_properties    => message_properties
                 ,payload               => raw_payload
                 ,msgid                 => msgid );
EXCEPTION
     WHEN OTHERS THEN NULL;
END;
```

B.3.2 Create a Project

A project is a container for the integration logic pertaining to an integration scenario. The following steps describe creating the `PO_Integration` project using iStudio.

1. From the **File** menu, select **New Project**. The Create Project dialog box is displayed.
2. Enter `PO_Integration` in the **Project Name** field, and click **OK**. The Repository Information dialog box is displayed.
3. Enter the correct repository information, and click **OK**.

See Also: "Creating a New Project" on page 2-11

B.3.3 Create the Common View Business Object

Each application has its own semantics and syntax. In order to integrate the data from multiple sources, a common view that is semantically compatible is required. The common views are events or procedures that are grouped in a business object, located under the Common Views node in iStudio. In this scenario, all events are grouped under the `Purchase_Order` business object.

The following steps describe creating the `Purchase_Order` business object.

1. From the **File** menu, select **New**, then select **Business Object**. The Create Business Object dialog box is displayed.
2. Enter `Purchase_Order` in the **Business Object Name** field, and click **OK**.

See Also: "Creating Business Objects" on page 3-2

B.3.4 Create Business Object Events

In order to integrate data between two or more systems, a semantically compatible view, or common view, is required. In this scenario, the insert, update, delete, and cancel events are grouped under the `Purchase_Order` business object. The following four events must be created:

- `PO_Cancel`
- `PO_Insert`
- `PO_Update`
- `PO_Delete`

Note: When an event is created, a Common Data Type representing its structure is automatically created. This common data type can then be reused to define the structure of other events.

The following steps describe creating the `PO_Insert` event using an XML DTD (Data Type Definition). The user can also use the database or other common data type to describe the structure of the event.

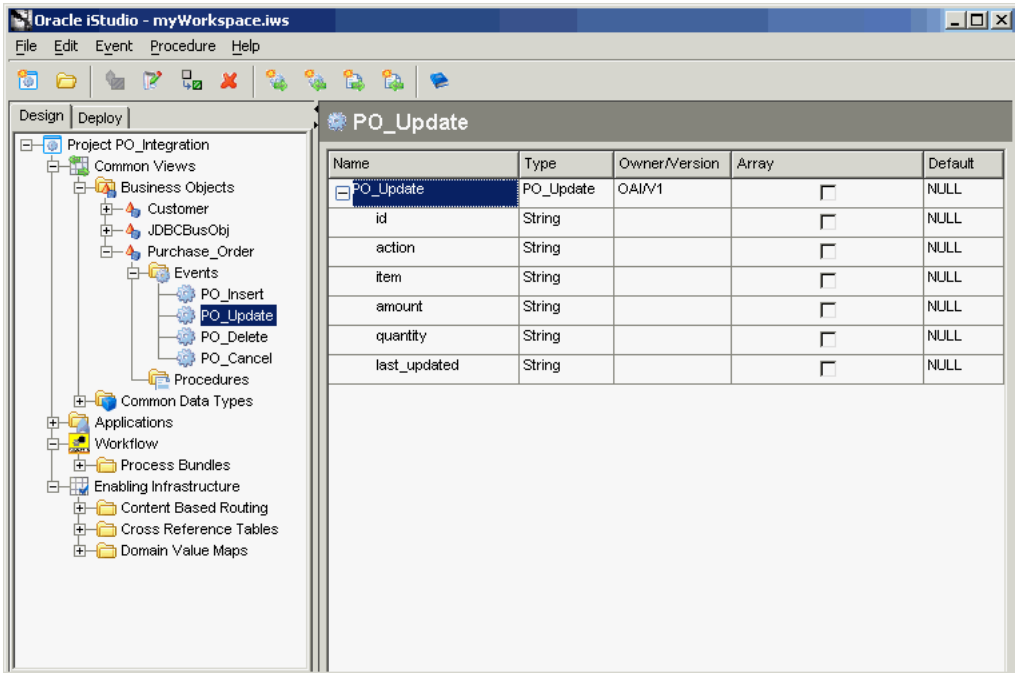
1. From the **File** menu, click **New**, and then select **Event**. The Create Event dialog box is displayed.
2. Select **Purchase_Order** as the Business Object.
3. Enter `PO_Insert` in the **Event Name** field.
4. Click **Import**, and select **XML**.
5. Select the predefined file, `PO_Insert_CV.dtd` in the Open dialog box, and click **Open**.
6. Select **PO_Insert** in the Select Root dialog box, and click **OK** to return to the Create Event dialog box.
7. Click **OK**.

Use similar steps for the `PO_Update`, `PO_Delete`, and `PO_Cancel` events, substituting the following correct XML DTD for each event. The `PO_Cancel`, `PO_`

`Delete`, `PO_Insert`, and `PO_Update` events appear in the Design Object Navigator under the Events node as shown in Figure B–3.

See Also: "Creating Events" on page 4-2

Figure B–3 Completed Event Node in iStudio



B.3.4.1 DTD Code

Each event has its own XML DTD. The code for each event is shown.

- `PO_Cancel`

```
<!ELEMENT PO_Cancel (id, action, item, amount, quantity)>
<!ELEMENT id                (#PCDATA)>
<!ELEMENT action            (#PCDATA)>
<!ELEMENT item              (#PCDATA)>
<!ELEMENT amount            (#PCDATA)>
<!ELEMENT quantity          (#PCDATA)>
```

- `PO_Update`

```
<!ELEMENT PO_Update (id, action, item, amount, quantity, last_updated)>
<!ELEMENT id                (#PCDATA)>
<!ELEMENT action            (#PCDATA)>
<!ELEMENT item              (#PCDATA)>
<!ELEMENT amount            (#PCDATA)>
<!ELEMENT quantity          (#PCDATA)>
<!ELEMENT last_updated      (#PCDATA)>
```

- `PO_Delete`

```
<!ELEMENT PO_Delete (id, action)>
<!ELEMENT id                (#PCDATA)>
<!ELEMENT action            (#PCDATA)>
```

- `PO_Insert`

```
<!ELEMENT PO_Insert (id, action, item, amount, quantity)>
```

```
<!ELEMENT id (#PCDATA)>
<!ELEMENT action (#PCDATA)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT amount (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
```

B.3.5 Create Applications

An application in iStudio represents an instance of an adapter communicating with an application. When you install an adapter, a unique name is supplied, and in iStudio, this name is used as the name of the application. This section demonstrates creating the AQAPP and DBAPP applications.

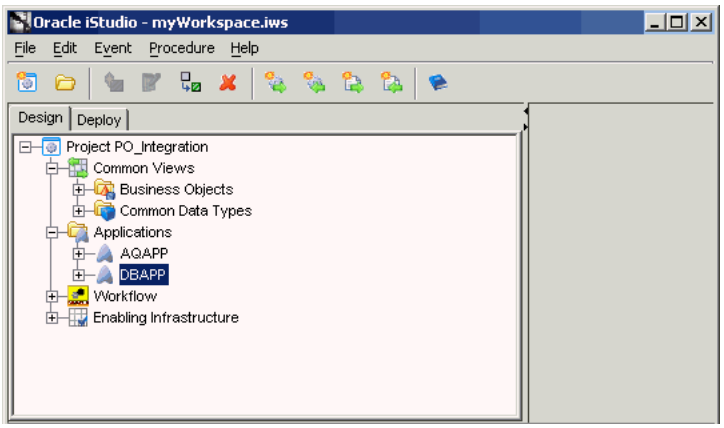
See Also: "Creating an Application" on page 3-1

The following steps describe creating the AQAPP application using iStudio.

1. From the **File** menu, select **New**, and then select **Application**. The Create Application dialog box is displayed.
2. Enter AQAPP in the **Application Name** field, and click **OK**.

Similarly, create the DBAPP application. The AQAPP and DBAPP applications appear in the Design Object Navigator under the Applications node as shown in Figure B–4.

Figure B–4 AQAPP and DBAPP Applications in iStudio



B.3.6 Create a Cross Reference Table

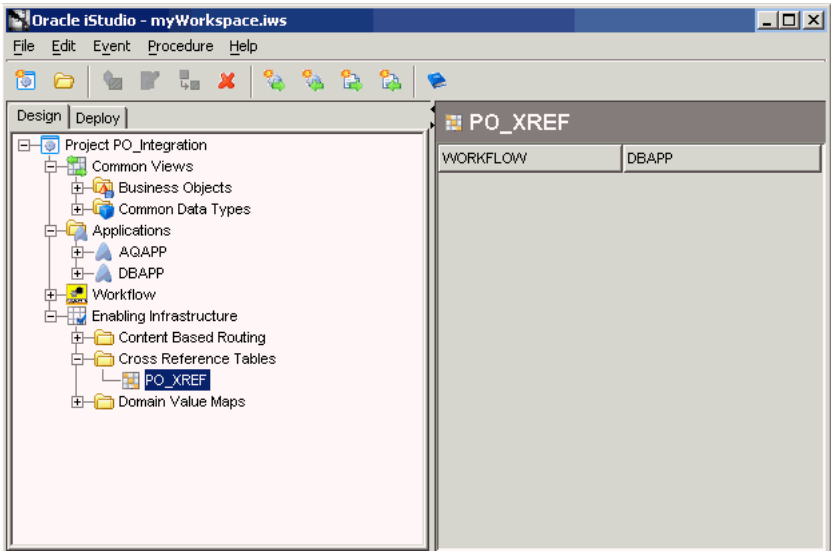
Each system has its own unique identifier or primary key. In most cases, an administrator does not allow any changes to the structure of the system. As a result, using a cross reference table, the keys of both systems can be maintained and cross-referenced for subsequent updates and deletes.

The following steps describe creating the PO_XREF cross reference table using iStudio. The table is automatically created in the repository schema and is referenced by the subscribing application. The WORKFLOW and DBAPP applications are added to the table, as the publisher and subscriber respectively.

1. From the **File** menu, click **New**, and then select **Cross Reference Tables**. The Create Cross Reference Table dialog box is displayed.
2. Enter PO_XREF in the **Cross Referenced Table Name** field, and click **Next**.

3. Select **WORKFLOW** and **DBAPP** from the Available Components list and click the right arrow button.
4. Click **Finish**. The `PO_XREF` cross reference table is displayed in the Design Object Navigator under the Cross Reference Tables node as shown in Figure B–5.

Figure B–5 PO_XREF Cross Reference Table in iStudio



See Also: "Working with Cross Referencing" on page 6-8

B.3.7 Create Publish Events

The database trigger in the Legacy Application, `AQAPP`, publishes messages when records are inserted, updated, or deleted in the purchase order table. This process happens outside the OracleAS Integration InterConnect environment. The OracleAS Integration InterConnect Advanced Queuing adapter is configured to read these messages. The publish events under the iStudio application will:

- Map the application view to the common view.
- Perform transformations.
- Publish the new event to subscribers in the OracleAS Integration InterConnect environment.

The following steps describe how the message received from the Legacy Application queue is processed.

Starting the Publish Wizard

To start the Publish Wizard:

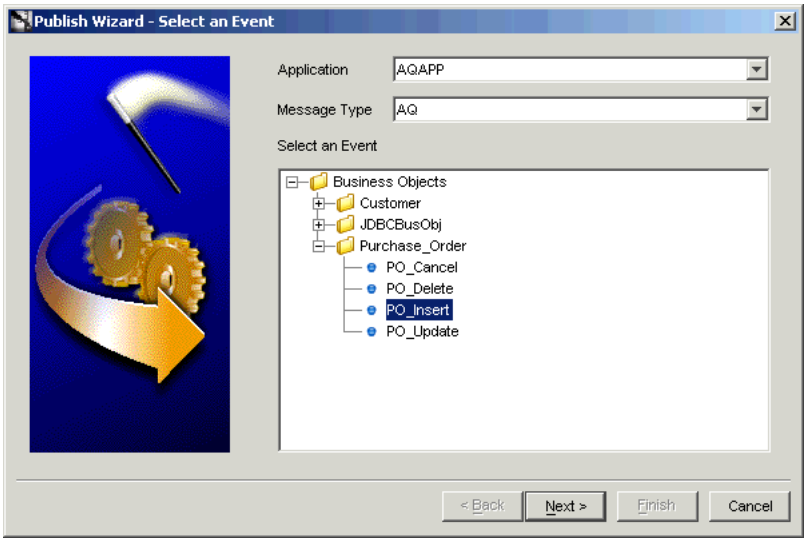
1. Expand the **Applications** node in the Design Object Navigator.
2. Select and expand the **AQAPP** application.
3. Select the **Published Events** node.
4. Right-click Published Events, and select **New**. The Publish Wizard is displayed.

Using the Publish Wizard to Publish the PO_Insert Event

To publish the `PO_Insert` Event:

1. Select an Event Page:
   a. Enter information in the following fields:
      * **Application**: Select **AQAPP** for the application.
      * **Message Type**: Select **AQ** for the message type.
   b. Expand the Business Objects list in the Select an Event dialog box and navigate to `PO_Insert`.
   c. Select `PO_Insert` and click **Next**.

Figure B–6 Publish Wizard - Select an Event page



2. Define Application View Page
   a. Import Attributes

      Import attributes from the common view by clicking **Import** and selecting **Common View**. The structure of the `PO_Insert` common view event is displayed. If the application view is different from the common view, then use the database or an XML DTD to define the structure.

   b. Create an Event Map

      An event is received and converted into a common view. This common view can be mapped by any application. If the structure of one or more events is identical, then routing becomes an issue. An event map is used to distinguish the routing in this situation. The Action field in the application view contains I for insert, U for update, or D for delete. Complete the following steps to create an event map:

   c. Click **Event Map**, then click **Add**. The Add Event Map page is displayed.
   d. Select the **Action** field, and enter `I` in the **Value** field.
   e. Click **OK** in the Add Event Map page.
   f. Click **OK** in the Event Map page.
   g. Click **Next**.
3. Define Mapping Page

Use the Define Mapping page to map fields from the **AQAPP View** to the **Common View** by using transformations. In this scenario the structure is identical, as a result, the `ObjectCopy` transformation is used to map all the fields at once. To define new mappings:

a. Click **DefineMappings**. The Mapping Parameters dialog box is displayed.

b. Expand the **PO_Insert** list and select the **PO_Insert** node in the AQAPP View box.

c. Select **ObjectCopy** in the Transformations box.

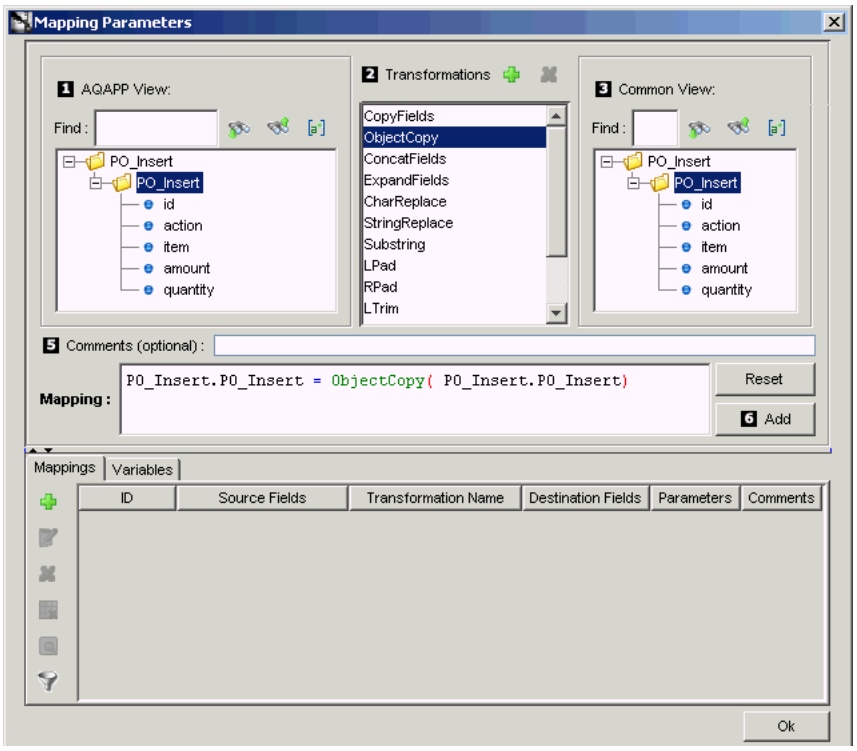
d. Expand the **PO_Insert** list and select the **PO_Insert** node in the Common View box.

e. Click **Add**.

f. Click **OK**. The new mapping is displayed in the Define Mapping page.

g. Click **Finish**.

Figure B–7 Publish Wizard - Mapping Parameters



To create the `PO_Update` and `PO_Delete` publish events, repeat the same steps, using the following values for steps 2 and 3.

- `PO_Update`
  - Use the `PO_Update` common view.
  - The event map value is `U`.
  - Use the `ObjectCopy` transformation and map to `PO_Update`.
- `PO_Delete`
  - Use the `PO_Delete` common view.

- The event map value is `D`.
- Use the `ObjectCopy` transformation and map to `PO_Delete`.

See Also: "Publishing an Event" on page 4-3

B.3.8 Subscribe to Events

The `DBAPP` application subscribes to the following three events:

- PO_Insert
- PO_Update
- PO_Delete

The `AQAPP` application subscribes only to the PO_Cancel event.

See Also: "Subscribing to an Event" on page 4-8

B.3.8.1 DBAPP Application Subscriptions

The following steps describe how the Order Fulfillment Application subscribes to messages.

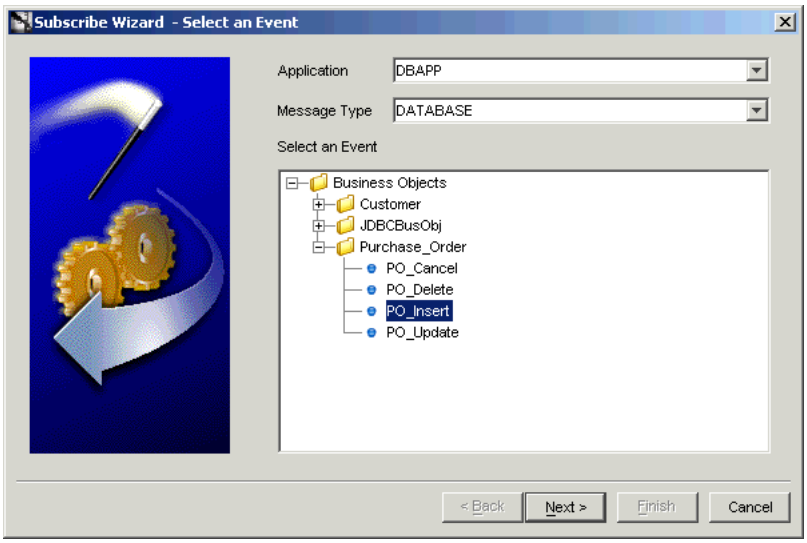
Starting the Subscribe Wizard:

1. In the Design Object navigator, expand the **Application** node to display the Subscribed Events node.
2. Right-click **Subscribed Events**, and select **New**. The Subscribe Wizard is displayed as shown in Figure B–8.

Using the Subscribe Wizard to Subscribe to the PO_Insert Event

1. Select an Event Page
   a. Enter information in the following fields:
      * **Application**: Select **DBAPP**.
      * **Message** Type: Select **Database**.
   b. Expand the **Business Objects** node in the Select an Event dialog box and navigate to **PO_Insert**.
   c. Select **PO_Insert**, and click **Next.**

Figure B–8 Subscribe Wizard - Select an Event page

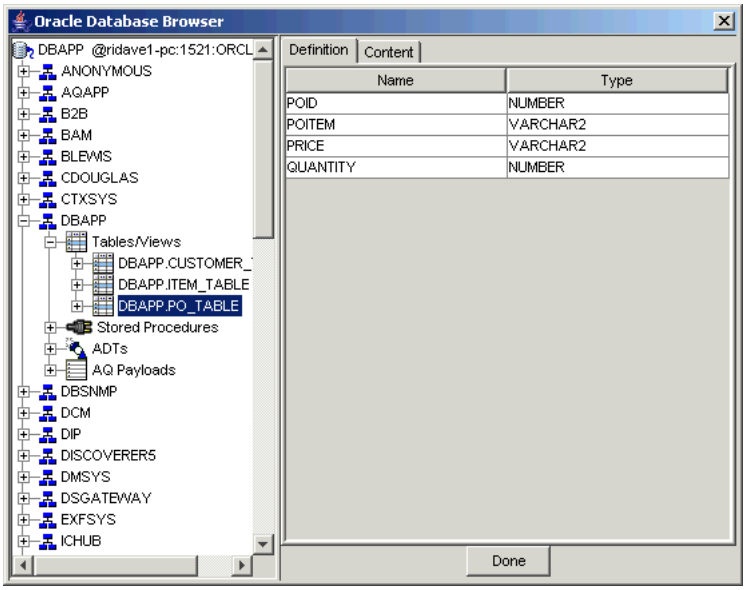


2. Define Application View Page

   a. Import attributes from the database.

      * Click **Import**, and select **Database**. The Database Login dialog box is displayed.
      * Enter the information to login to the database, and click **Login**. The Oracle Database Browser dialog box is displayed as shown in Figure B–9.
      * In the Browser dialog box, expand the **Tables/Views** node and select **DBAPP.PO**.
      * Click **Done**.

Figure B–9 Subscribe Wizard - Oracle Database Browser

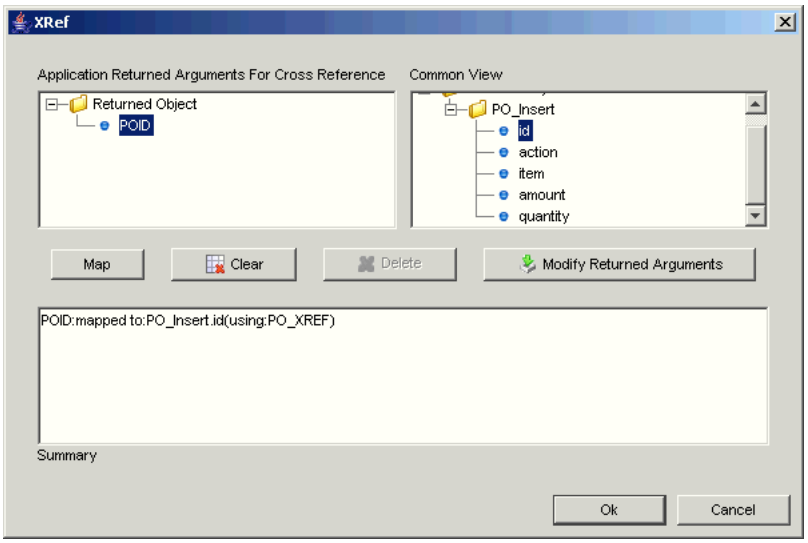


   b. Create a cross-reference.

In "Create a Cross Reference Table" on page B-7, the PO_XREF cross reference table was created. This table synchronizes the primary keys on the source and target systems.

* Click **Cross Reference** and select **PO_XREF**. The XRef dialog box is displayed as shown in Figure B–10.
* Select **POID** in the Application Returned Arguments For XRef box.
* Select **id** in the Common View box.
* Click **Map**.
* Click **OK**.

Figure B–10 Subscribe Wizard - Cross Reference



c. Click **Next**.

3. Define Mapping Page

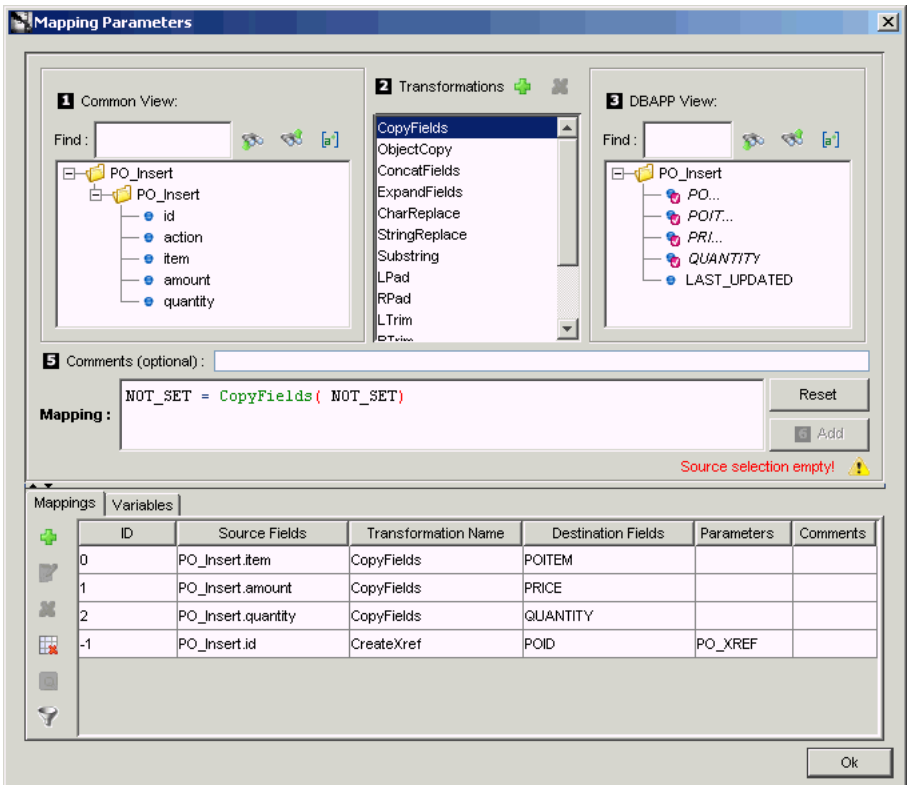
a. Define a new mapping:

* Click **Define Mappings**. The Mapping Parameters dialog box is displayed as shown in Figure B–11.
* Expand the **PO_Insert** list and the **PO_Insert** node in the Common View box. Map the following:

Common View	Transformation	DBAPP View
item	CopyFields	POITEM
amount	CopyFields	PRICE
quantity	CopyFields	QUANTITY

* Click **OK**.

Figure B–11 Subscribe Wizard - Mapping Parameters



b. Click **Next**.

4. Define Stored Procedure Page

a. Select `sub_PO_Insert_OAI_V1` from the SQL code list. The SQL code is displayed in the box.

b. Add the following code at end of the existing SQL code:

```
PROCEDURE sub_PO_Insert_OAI_V1( POID          IN OUT LONG,
                                POITEM        IN LONG,
                                PRICE         IN LONG,
                                QUANTITY      IN NUMBER,
                                LAST_UPDATED  IN DATE)
AS
 v_poid NUMBER;

 BEGIN
  SELECT PO_SEQ.NEXTVAL INTO v_poid FROM dual;
  POID :=v_POID;

  INSERT INTO PO VALUES
   ( v_POID, POITEM, PRICE, QUANTITY, SYSDATE );
 COMMIT;
END sub_PO_Insert_OAI_V1;
```

c. Click **Finish**.

Create the Subscribed PO_Update Event

To create the subscribed `PO_Update` Event:

1. Select the **PO_Update** event.

2. Import the Application View.
3. Define Mapping Page
   a. Map the same parameters as described in `PO_Insert`.
   b. In addition, map the following:
      * Expand the **PO_Update** list and node in the Common View box and select **id**.
      * Select the **LookupXref** transformation.
      * Expand the **PO_Update** list and select **POID** in the Application View box.
      * Click **Add**.
      * Click **OK**.
      * Select the Req. checkbox for table listed in the Parameters column and click **OK**.
   c. Click **Next**.
4. Define Stored Procedure Page
   a. Select `sub_PO_Update_OAI_V1` for the SQL code for field. The code is displayed in the box.
   b. Add the following code at end of the existing SQL code:

```
PROCEDURE sub_PO_Update_OAI_V1( POID          IN NUMBER,
                                POITEM        IN LONG,
                                PRICE         IN LONG,
                                QUANTITY      IN NUMBER,
                                LAST_UPDATED  IN DATE)
AS
 v_poid       NUMBER :=poid;
 v_poitem     LONG   :=poitem;
 v_price      LONG   :=price;
 v_quantity   NUMBER :=quantity;
BEGIN
  UPDATE PO SET  poitem = v_poitem, price = v_price
                 quantity = v_quantity, last_updated = sysdate
  WHERE poid = v_poid;
  COMMIT;

EXCEPTION
  WHEN OTHER THENS NULL;

END sub_PO_Update_OAI_V1;
```

   c. Click **Finish**.

Create the Subscribe PO_Delete Event

To create the subscribe PO_Delete Event:

1. Select the **PO_Delete** event.
2. Import the Application View.
3. Define Mapping Page
   a. Map the following:

* Expand the **PO_Delete** list and node in the Common View box and select `id`.
* Select the **DeleteXref** transformation.
* Expand the **PO_Delete** list and select **POID**.
* Click **Add**.
* Click **OK**.
* Select **PO_XREF** from the values column and click **OK**.

b. Click **Next**.

4. Define Stored Procedure Page

a. Select `sub_PO_Delete_OAI_V1` for the SQL code for field. The code is displayed in the box.

b. Add the following code at the end of the existing SQL code:

```
PROCEDURE sub_PO_Delete_OAI_V1( POID          IN NUMBER,
                                POITEM        IN LONG,
                                PRICE         IN LONG,
                                QUANTITY      IN NUMBER,
                                LAST_UPDATED  IN DATE)
AS
 v_poid       NUMBER :=poid;
BEGIN
  DELETE FROM WHERE PO v_poid = poid;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN NULL;

END sub_PO_Update_OAI_V1;
```

c. Click **Finish**.

B.3.8.2 AQAPP Application Subscriptions

The `AQAPP` application subscribes to the `PO_Cancel` event.

1. Select an Event Page

a. Enter information in the following fields:

* Application: Select **AQAPP**.
* Message Type: Select **AQ**.

b. Select **PO_Cancel** and click **Next**.

2. Define Application View Page

a. Import attributes from the common view and click **Next**.

3. Define Mapping Page

a. Define a new mapping:

* Click **Define Mappings** and map the following:

  Id Copyfields Id
* Click **Add**.
* Click **OK**.

b. Click **Finish**.

B.3.9 Create an Oracle Workflow Process Bundle

A process bundle enables related business processes to be grouped and transferred to the Oracle Workflow environment where user-defined business logic is applied.

Each business process enables related publish, subscribe, invoke, and implement activities to be grouped and placed in the Oracle Workflow Business Event System.

Create a Process Bundle

The following steps describe creating the PO process bundle using iStudio:

1. From the project list, expand the **Workflow** node and navigate to Process Bundle.
2. Right-click **Process Bundles** and select **New**. The Create Process Bundle dialog box is displayed.
3. Enter `PO` in the Process Bundle Name field and click **OK**.

Create a Business Process

The following steps describe creating the PO business process using iStudio:

1. Expand the **Process Bundle** node on the project list and navigate to Business Processes.
2. Right-click **Business Processes** and select **New**. The Create Business Process dialog box is displayed.
3. Enter `PO` in the Business Process Name field and click **OK**.

Create the Subscribe and Publish Activities

The Oracle Workflow business process uses the common view. As a result, transformation and mapping is not required and the only types of activities used are as follows:

- Subscribe: Oracle Workflow receives a message from OracleAS Integration InterConnect.
- Publish: Oracle Workflow sends a message to OracleAS Integration InterConnect.
- Invoke: Oracle Workflow sends a request message to OracleAS Integration InterConnect and receives a reply.
- Implement: Oracle Workflow receives a request from OracleAS Integration InterConnect and sends a reply.

In this scenario, the PO_Insert, PO_Update, and PO_Delete messages are routed to Oracle Workflow to apply business logic. Based on this logic, messages are sent to the Order Fulfillment Application or the PO_Cancel message is sent to the Legacy Application. Oracle Workflow must:

- Subscribe to and publish PO_Insert.
- Subscribe to and publish PO_Update.
- Subscribe to and publish PO_Delete.
- Publish PO_Cancel.

Create Subscribe Activity

The following steps describe creating the subscribe activity using iStudio:

1. From the Project list, expand the **Workflow** node and navigate to Business Processes.
2. Right-click **PO** business process and select **Subscribe** Activity.
3. Select Event **PO_Insert** and click **Finish**.

Repeat these steps for the `PO_Update` and `PO_Delete` events, substituting the correct values where necessary.

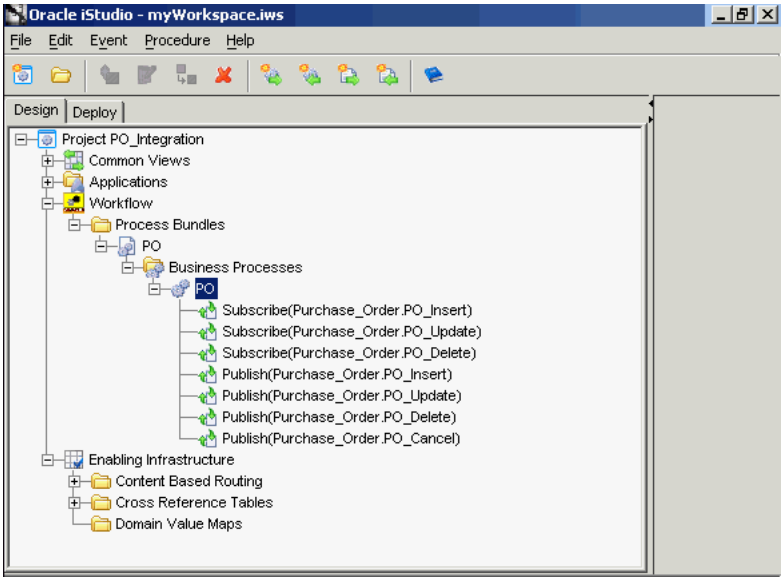
Create Publish Activity

The following steps describe creating the publish activity using iStudio:

1. From the Project list, expand the **Workflow** node and navigate to Business Processes.
2. Right-click **PO** business process and select **Publish** Activity. Right-clicking any item displays a pop-up box.
3. Select Event **PO_Insert** and click **OK**.

Repeat these steps for the `PO_Update`, `PO_Delete`, and `PO_Cancel` events, substituting the correct values where necessary. The subscribe and publish events appear in the Design Object Navigator under the `PO` node as shown in Figure B–12.

Figure B–12 Subscribe and Publish Activities in iStudio



B.3.10 Create Content-based Routing

When an event is published, it is automatically routed to any event's subscriber, by default. If the routing of an event needs to be based on a value in the message or message header, then content-based routing is required in this scenario. All changes to the purchase orders must be approved and routed to Oracle Workflow to apply business logic.

The logic to be applied for the Events `PO_Insert`, `PO_Update`, and `PO_Delete` as follows:

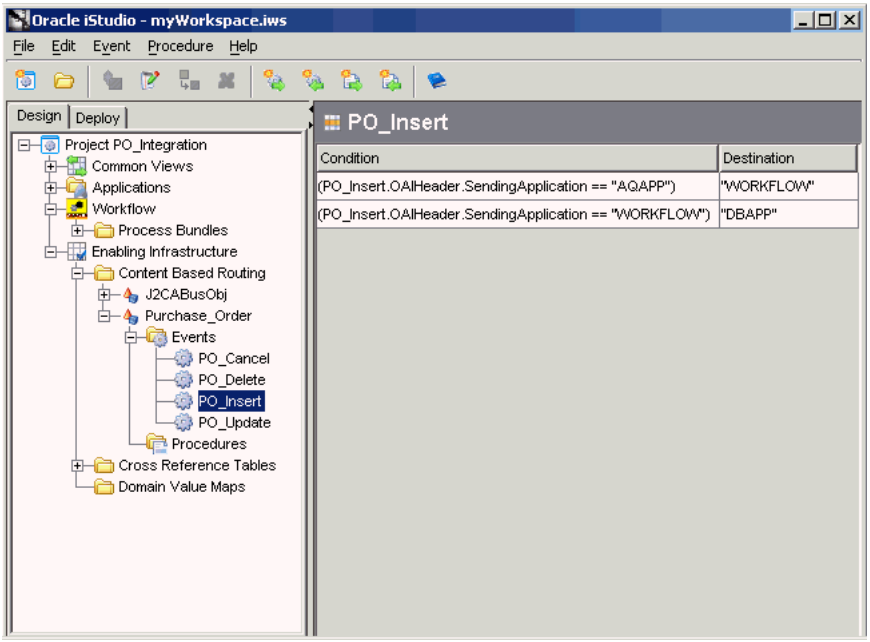
- If AQAPP is the source application, then route to the WORKFLOW destination application. The Wizard steps are as follows:

1. Source Page: Select **OAI_Header.SendingApplication**
2. Choose Operator Page: Select =
3. Choose Value Page: Enter `AQAPP`
4. Addition Condition Page: Select Radio Button Complete & press Finished
5. Destination Page: Select WORKFLOW

- If WORKFLOW is the source application, then route to the DBAPP destination application. The Wizard steps are as follows:
  1. Source Page: Select **OAI_Header.SendingApplication**
  2. Select Operator Page: Select **=**
  3. Choose Value Page: Enter `WORKFLOW`
  4. Addition Condition Page: Select Radio Button Complete & press Finished
  5. Destination Page: Select **DBAPP**

The procedure repeats for the `PO_Update` and `PO_Delete` events. Figure B–13 describes the completed content-based routing in iStudio.

Figure B–13 Completed Content Routing in iStudio



B.3.11 Deploy the Process Bundle to Oracle Workflow

Deploying the Oracle Workflow process bundle accomplishes the following:

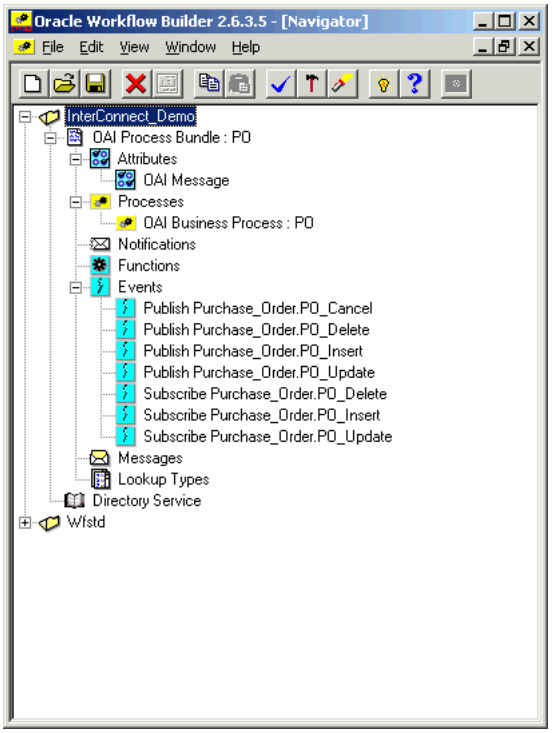
- Places the event definitions in the Oracle Workflow Business Event System.
- Creates a default Oracle Workflow file (.wft).
- Launches the Oracle Workflow Builder and Monitor.

See Also: Chapter 7, "Using Oracle Workflow"

The following steps describe deploying the process bundle to Oracle Workflow:

1. Right-click the Workflow node on the Deploy tab in iStudio and select Deploy. The Deploy dialog box is displayed.
2. Select Event Definitions to Workflow Business Event System, then Process Definitions for File in the Deploy to Workflow box.
3. Click OK. The Workflow BES Login dialog box is displayed.
4. Log in to Oracle Workflow using the correct user name, password, and URL. Click OK. The Deploy dialog box is displayed.
5. Enter a file name for the Oracle Workflow file, such as InterConnect_Demo.wft, in the File Name field, and click Open. Oracle Workflow is started with InterConnect_Demo as shown in Figure B–14.

Figure B–14 Completed Deployment in Oracle Workflow



B.3.12 Creating Objects in Oracle Workflow for Modeling

The original requirement for this scenario are as follows:

"An administrator must approve all changes such as insert, update, and delete before they are applied to the Order Fulfillment System. If a change is approved, it is sent to the Order Fulfillment System. If a change is rejected, then a cancellation notification is sent back the legacy system."

This business logic can be implemented in Oracle Workflow. The Oracle Workflow components required are:

- An Item Type equivalent to a Project
- An Attribute An object to hold the message in the event
- A Process To model the Business Logic
- Events For the modeling in the process.

- A Notification To notify the administrator in the Oracle Workflow Monitor.

Components transferred from iStudio.

- Item Type: OAI Process Bundle: PO
- Attribute: OAI Message
- Process: OAI Business Process: PO
- Events:
  - `Publish  Purchase_Order.PO_Cancel`
  - `Publish  Purchase_Order.PO_Insert`
  - `Publish  Purchase_Order.PO_Update`
  - `Publish  Purchase_Order.PO_Delete`
  - `Subscribe  Purchase_Order.PO_Insert`
  - `Subscribe  Purchase_Order.PO_Update`
  - `Subscribe  Purchase_Order.PO_Delete`

Oracle Workflow components are required to create a Notification.

B.3.12.1 Message

The message a notification activity will send.

B.3.12.2 Lookup Type

A static list of values that can be referenced by various objects. For example a message attribute can reference a lookup type as a means of providing a list of possible responses to the performer of a notification.

B.3.12.3 Notification

When the workflow engine reaches a notification activity, it issues a Send() API call to the Notification System to send the message to an assigned performer. When a performer responds to a notification activity, the Notification System processes the response and informs the workflow engine that the notification activity is complete.

B.3.12.4 What Oracle Workflow provides.

Oracle Workflow has a set of pre-defined item types with standard functionality The Standard item type contains generic activities that can be copied in a users item type. In this scenario we will be using the Lookup Type Approval.

B.3.12.5 Copy Lookup Type (Approval)

As described, the user must create a Oracle Workflow Notification. The notification has two dependent objects, A lookup Type and a Message. The Lookup Type (Approval) can be copied from the standard item type.

B.3.12.6 Create an Oracle Workflow Message

The following steps describe creating a new Oracle Workflow message called Insert_ Message

1. In the Object Navigator right-click the **Message** Node and select **New** to launch the property sheet. In each tab, add the following entries:

2. Message Tab:
   - Internal Name: `Insert_Message`
   - Display Name: `Insert Message`
   - Description: `Insert Message`
3. Body Tab:
   - Subject: `Insert Message`
   - Text Body: `A record has been Inserted in the Purchase Order Table`.
4. Result Tab:
   - Display Name: `Insert_Message`
   - Description: `Insert_Message`
   - Lookup Type: `Approval` (From Lookup Type)
5. Click **OK**.

Using the default Copy and Paste functionality create the following messages using message `Insert_Message` as the template:

- `Update_Message`: Repeats the preceding steps and use the same setting, changing all references to insert to update.
- `Delete_Message`: Repeats the preceding steps and use the same setting, changing all references to insert to Delete.

B.3.12.7 Create an Oracle Workflow Notification

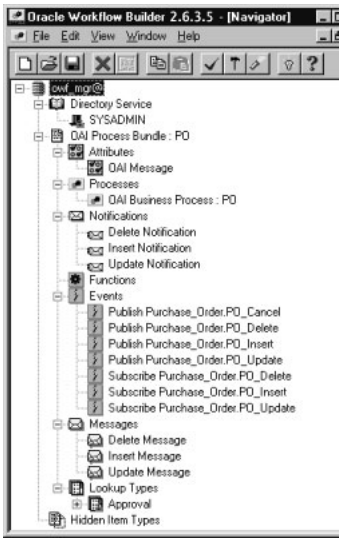
The following steps describe creating a new Oracle Workflow Notification.

1. In the Object Navigator, right-click the Notification Node and select New to launch the property sheet. In each tab, add the following entries:
2. Activity Tab:
   - Internal Name: `Insert_Notification`
   - Display Name: `Insert_Notification`
   - Description: `Insert_Notification`
   - Message: `Insert_Message` (Created previous step)
   - Result Type: `Approval` (From Lookup Type)
3. Click **OK**.

Using the default Copy and Paste functionality create the following notifications using notification Insert_Notification as the template:

- `Update_Notification`: Repeats the preceding steps and use the same setting, changing all references to insert to update.
- `Delete_Notification`: Repeats the preceding steps and use the same setting, changing all references to insert to delete.

Figure B–15 Completed Oracle Workflow Notifications



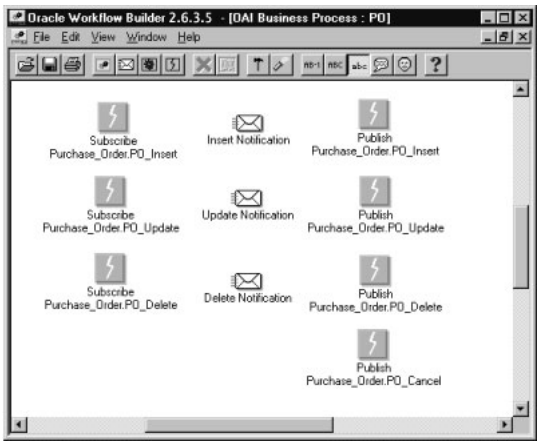
B.4 Modeling Business Logic in Oracle Workflow

Now that all of the required objects have been created, the business logic can be modeled. The following steps describe this process.

1. In the Oracle Workflow Object Navigator, expand the **OAI Process Bundle: PO** item type.
2. Expand the **Processes** node.
3. Right-click **OAI Business Process: PO** and select **Process Details**.

   Another way to display the process details is to double-click OAI Business Process: PO.
4. Move the following from the Oracle Workflow Object Navigator to the Oracle Workflow Workspace:
   - `Insert_Notification`
   - `Update_Notification`
   - `Delete_Notification`
5. Rearrange the items as shown in Figure B–16.

Figure B–16 Items Arranged in Oracle Workflow Builder



1. The subscribe events are the entry point in this process. The Start and End Property for each event must be edited and set to START. Double-click an object to launch its property sheet. The Start and End property is under the Node tab.

   Subscribe to the following events:

   - `Purchase_Order.PO_Insert`
   - `Purchase_Order.PO_Update`
   - `Purchase_Order.PO_Delete`

2. The publish events are the exit point from this process. The Start and End Property for each event must be edited and set to END. Double-click an object to launch its property sheet. The Start and End property is under the Node tab.

   Publish to the following events:

   - `Purchase_Order.PO_Insert`
   - `Purchase_Order.PO_Update`
   - `Purchase_Order.PO_Delete`
   - `Purchase_Order.PO_Cancel`

3. The notifications must be assigned in order for a person to receive the notification in Oracle Workflow Monitor. The Performer value property should be set to SYSADMIN for each notification. Double-click an object to launch its property sheet. The Performer Value field is located under the Node tab.

   Assign the following notifications:

   - `Insert_Notification`
   - `Update_Notification`
   - `Delete_Notification`

4. Mapping lines need to be drawn between the objects to define the process flow. Lines are drawn by right-clicking on an object and dragging to another object.

   a. Draw a mapping line from `Subscribe Purchase_Order. PO_Insert` to `Insert_Notification`.

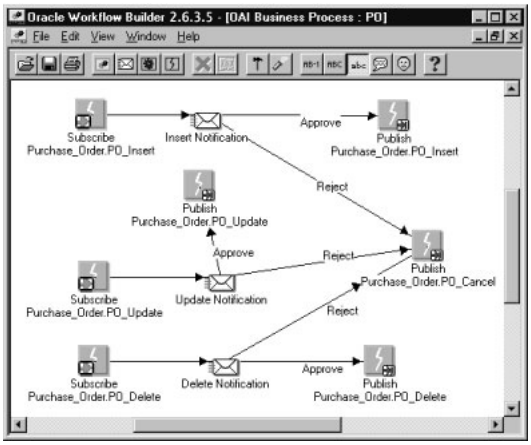
   b. Draw a mapping line from `Insert_Notification` to `Publish Purchase_Order.PO_Insert` and select **Approve** from the list that will appear when the line is drawn.

c. Draw a mapping line from `Insert_Notification` to `Publish Purchase_Order.PO_Cancel` and select **Reject** from the list that will appear when the line is drawn.

d. Repeat steps for Update & Delete objects.

5. Save your work to the database. The completed business process in Oracle Workflow Builder is shown in Figure B–17.

Figure B–17 Completed Business Process in Oracle Workflow Builder



B.5 Deployment

After modeling the business objects, it is time to deploy the OracleAS Integration InterConnect business objects. The following section describes the deployment process.

- Setting Queues
- Sync Adapters
- Exporting and Installing Code

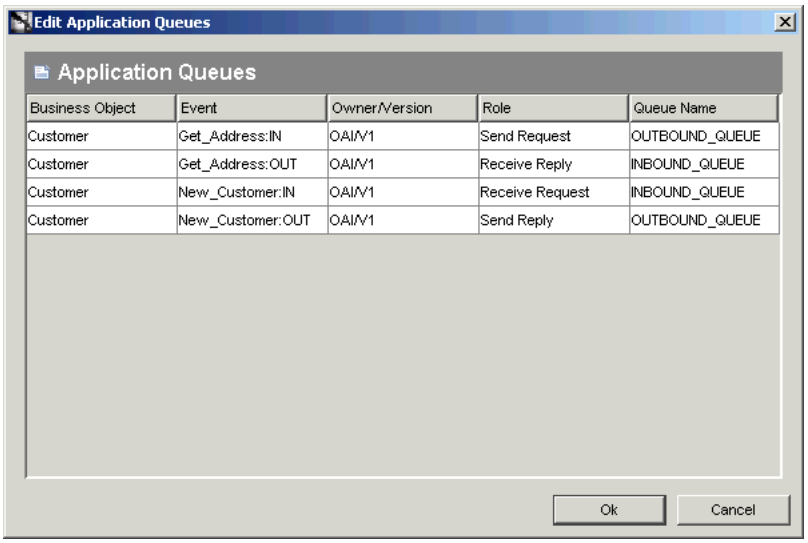
B.5.1 Setting Queues

The AQAPP application in iStudio corresponds to the Advanced Queuing adapter that communicates with the legacy application. The legacy application, through a database trigger, places inserted, updated, and deleted records into a queue using Oracle Advanced Queuing. To communicate to and from the OracleAS Integration InterConnect environment, the adapter must be configured to send and receive on those external queues.

The following steps describe this task.

1. On the Deploy tab in iStudio, expand the **Applications** list and navigate to AQAPP.
2. Expand the **AQAPP** node and navigate to the Routing node.
3. Expand the **Routing** node and select Application Queues. The Application Queues property sheet is displayed in the iStudio window.
4. Select **Edit** from the Edit menu. The Edit Application Queue dialog box is displayed as shown in Figure B–18.

Figure B–18 Application Queues in iStudio



5. Add the Queue name to each event:

Queue Name	Event
INBOUND_QUEUE	PO_Cancel
OUTBOUND_QUEUE	PO_Insert, PO_Update, and PO_Delete

6. Click **OK**.

B.5.2 Sync Adapters

Each adapter has different cache settings to minimize communication to the repository and to improve performance. If you have updated the metadata, you must synchronize the adapter and repository metadata. The following steps describe this task:

1. Select **File** from the menu, then select **Sync Adapters**. The Sync Adapters dialog box is displayed.
2. Select the applications to which to sync adapters, and click **OK**.

B.5.3 Exporting and Installing Code

Depending on the adapter type, there may be code that must be exported to a file and installed in the target application database. The following steps describe exporting the code using the Export Application dialog box in iStudio.

1. In the iStudio window, click the **Deploy** tab. Right-click **Applications** and select **Export PL/SQL**. The Export Application dialog box is displayed.
2. Select the applications to export code.
3. Enter the file prefix in the File Prefix field and click **OK**.

The resulting text file is a SQL*Plus script that is run on the target schema.

See Also: "Deploying PL/SQL Stored Procedures" on page 8-1

Example B–1 Exporting and Installing Code

The following example helps to explain exporting and installing code. This example is based on the following:

- Adapter type: Database Adapter
- iStudio application: `DBAPP`
- Subscribe event: `PO_Delete`

```
PROCEDURE sub_PO_Delete_OAI_V1 (   POID         IN NUMBER,
                                   POITEM       IN LONG,
                                   PRICE        IN LONG,
                                   QUANTITY     IN NUMBER,
                                   LAST_UPDATED IN DATE,)
AS
          v_poid NUMBER :=poid;
BEGIN     DELETE FROM PO WHERE v_poid = poid;
          COMMIT;
EXCEPTION
          WHEN OTHERS THEN NULL;

END sub_PO_Delete_OAI_V1;
```

B.6 Conclusion

The final step is to test the integration.

1. A record is inserted into the Legacy System.
2. The legacy system's database trigger queues the record in its `OUTBOUND_QUEUE`.
3. OracleAS Integration InterConnect receives the message, performs transformations, converts data to a common view, and routes the message to Oracle Workflow.
4. Oracle Workflow applies the business logic and issues a notification.
5. The System Administrator logs on to Oracle Workflow Monitor, receives the Insert_notification, and approves the record.
6. OracleAS Integration InterConnect received the message, performs transformations, cross-references the primary keys, converts data to the application View, and routes the message to Order Fulfillment System.
7. The deployed code receives the message and inserts the record into the Order Fulfillment system.
8. Oracle InterConnect Manager is used to examine the inserted record and monitor the integration throughput.

This process should be repeated for Update and Delete.

C

Using the Data Definition Description Language

This appendix describes how to use the Data Definition Description Language (D3L) in its native format message to application view translations. It contains the following topics:

- About D3L
- Native Format Message and D3L File Example
- D3L File Structure
- D3L Integration with OracleAS Integration InterConnect Adapters
- Installing D3L
- Configuring D3L
- D3L Use Case
- Additional D3L Sample Files and DTD

C.1 About D3L

This section contains these topics:

- What Is D3L?
- When Is D3L Used?

C.1.1 What Is D3L?

D3L is an XML-based message description language. It describes the structure that an application's native, non-XML format message (also called the native view of the application) must follow to communicate with OracleAS Integration InterConnect. Oracle provides the following OracleAS Integration InterConnect transport adapters that interact with the D3L message description language:

- FTP
- HTTP
- MQ Series
- SMTP

OracleAS Integration InterConnect Adapters perform the following tasks:

- Validate the D3L message description files during run-time initialization.

- Use the D3L translation engine (subcomponent of the bridge) to translate messages from:
  - Native format message to application view
  - Application view to native format message
- Transport message payload data between an application and OracleAS Integration InterConnect.

> **Note:** Native format messages that are already in XML format are not translated by OracleAS Integration InterConnect Adapters if the `ota.type` parameter is set to `XML` in the `adapter.ini` file.

See Also: "D3L Integration with OracleAS Integration InterConnect Adapters" on page C-23

C.1.2 When Is D3L Used?

Some applications do not use XML as native message payload format. These applications use native formats, such as structured records of bytes and characters. For these native formats to be successfully translated into a format understood by other applications, the messages content must follow a predefined, structured set of rules. This structured format can then be translated into an application view, transformed into a common view, and understood by other applications.

D3L provides both a predefined, structured set of rules and translation capabilities for native format messages. D3L provides:

- An XML-based message description language that describes the contents of native format messages
- A translation engine that uses the instructions defined in the D3L file to translate the native format message contents to and from an application view

The D3L descriptions must comply with a syntax defined by the D3L document type definition (DTD). D3L enables you to describe the record layout of binary, string, structured, and sequence data. Use D3L only when the number of fields in the underlying native format message is fixed and known. D3L is not suitable for the following:

- Descriptions of arbitrarily structured data like regular XML
- Name-value pair data
- Conditional data structures, which require token look-aheads to parse

See Also:

- "Supported D3L Data Types" on page C-10
- "D3L DTD" on page C-49

C.1.3 D3L Features

This section describes data definition description language enhancements. It includes the following topics:

- Integrate Transport Properties
- Allow Multiple Imparrays

C.1.3.1 Integrate Transport Properties

This enhancement enables the Data Definition Description Language (D3L) author to add a new type of member, property. Syntax, to a D3L structure, in addition to `fields` and `pads`. For example:

```
<struct ...
   ...
   <property name="prop_name" />
   ...
</struct>
```

> **Note:** There is no type definition associated with this structure element.

The modified D3L Data Type Definition (DTD) for this new structure element is:

```
<!ENTITY % StructElements
   "field | property | pad"
>

<!ELEMENT property EMPTY >
   <!ATTLIST property
      %FieldAttributes;
>
```

The semantics of this new structure element is to link data in a transport protocol header with the message payload. In other words, when a D3L containing a structure with one, or more, property member(s) is imported in iStudio, it will create a String `OAI` attribute with the name specified in the property `name` attribute.

At runtime, this `OAI` attribute will be populated with the value of a transport protocol header, `inbound-to-hub`, which name matches the `name` attribute of the property member. Simlarly, for outbound messages, the OracleAS Integration InterConnect message `payload` property value will define the value of the corresponding protocol header.

For example, if using OracleAS Integration InterConnect Adapter for FTP, the file structure would be:

```
<struct ..>
   <property name="filename" />
<field ...
```

In this case, the OracleAS Integration InterConnect Application View attribute `filename`, that is derived from the D3L definition, would be assigned the name of the actual file being passed to D3L. For outbound message, the value will determine the physical filename being used to store the file.

If using OracleAS Integration InterConnect Adapter for HTTP, an example file structure would be:

```
<struct ..>
   <property name="Host" />
   <property name="Referer" />
<field name="...> < ...
```

> **Note:** This is a dynamic payload dependent feature, which will override settings in the `adapter.ini` file and Application View Meta Data Modify Fields. As a result, the property `ota.send.endpoint` could be overridden by a corresponding message attribute defined through the D3L.

C.1.3.2 Allow Multiple Imparrays

The D3L syntax enables you to create multiple nested imparrays for outbound translations (`app-to-native` or `hub-to-spoke`).

Multiple nested imparrays can be created to match multiple nested arrays in XML as XML does not have the need to declare the length of an array. For example:

```
<array1>
   <array2>..</array2>
   <array2>..</array2>
</array1>
<array1>
   <array2>..</array2>
   <array2>..</array2>
<array2>..</array2>
```

If this XML message was published by OracleAS Integration InterConnect Adapter for AQ, and consumed by OracleAS Integration InterConnect Adapter for FTP that is running in D3L mode, the preceding structure would then be matched by the following D3L structure:

```
<imparray id="array1">
   <imparray id="array2">
      ..
```

As D3L does not perform parsing for `app-to-native` translation, also known as `production`, the preceding D3L is entirely possible. However, for `native-to-app` translations, the preceding D3L would be invalid as a single imparray by itself would consume the rest of the native message.

> **Note:** This new imparray semantics depart from the design principle that D3L is a fully bidiretional symmetric translator and can perform both `native-to-app` and `app-to-native` translation using just one the same D3L definition.

Finally, the D3L translator will determine, at runtime, whether multiple nested imparrays exist in a D3L. If multiple nested imparrays are detected, the translator will prevent the D3L from being used for parsing purposes (`native-to-app` translations). If not detected, the D3L translator will flag an error condition.

C.2 Native Format Message and D3L File Example

This section provides an example of how the contents of a native format message are:

- Described in a D3L file
- Configured with the required D3L file

To successfully translate the native format message, you need to satisfy both the preceding conditions.

This section contains the following topics:

- Description of Native Format Message Contents in a D3L File
- Configuration of Native Format Message with a D3L File

C.2.1 Description of Native Format Message Contents in a D3L File

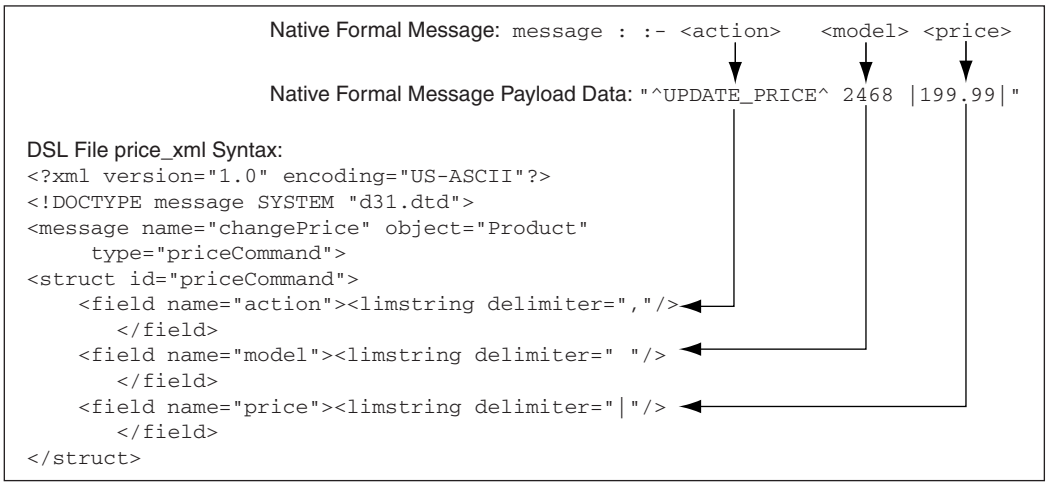
This section describes an application's native format message (named `price`) that contains data for updating the price of personal computer model number 2468 to 199.99. The native message uses the following format to describe the data:

```
message ::=  action  model  price
```

Where...	Is...
`action`	`UPDATE_PRICE`
`model`	`2468`
`price`	`199.99`

The data must strictly follow the structure defined in a D3L file, for this example, `price.xml`. This D3L translation engine translates the data into an application view. Figure C–1 shows how a D3L file defines the structure that the native format message `price` must follow to successfully define the data elements.

Figure C–1 Native Format Message Payload Data and D3L File Syntax



All three data elements are defined as strings with different delimiters for separating their data.

C.2.2 Configuration of Native Format Message with a D3L File

When the D3L translation engine receives a native format message, such as `price`, it must determine the exact D3L file to verify the native format message contents, such as, `price.xml`.

This section describes the methods for configuring the correct D3L file with the native format message. It contains the following topics:

- adapter.ini Parameter File Setting

- Message Header Attributes

C.2.2.1 adapter.ini Parameter File Setting

The `ota.d3ls` parameter in the `%ORACLE_HOME%\integration\interconnect\adapters\`*`application`*`\adapter.ini` file enables you to define the D3L file to use with the native format message. For example:

```
ota.d3ls=price.xml
```

When the D3L translation engine receives the native format message from the bridge, it retrieves the correct D3L file based on this parameter setting. Multiple D3L files can also be defined using the ota.d3ls parameter. For example:

```
ota.d3ls=price.xml,emp.xml,booking.xml
```

Unless one of the methods described in "Message Header Attributes" is used, the D3L translation engine compares the data structure in the native format message to each D3L file until it finds the correct one to use for translation.

C.2.2.2 Message Header Attributes

The D3L file includes message header attributes work with the D3L engine to choose the correct D3L file for translating a native format message to an application view. The values for these message header attributes match the settings in the native format message.

Message header attribute values override the approach of comparing each D3L file defined with the `ota.d3ls` parameter in the `adapter.ini` file with a native format message.

Two methods to set the message header attribute values are available:

- Name/Value Pair Message Header Attributes
- Magic Value Message Header Attribute

Both methods enable the D3L translation engine to use the correct D3L file for translation after receiving the native format message.

Note: When the correct D3L file is selected and a successful translation has taken place, the `message` element attributes `name` and `object` in the D3L file define the OracleAS Integration InterConnect event name and business object, respectively.

C.2.2.2.1 Name/Value Pair Message Header Attributes

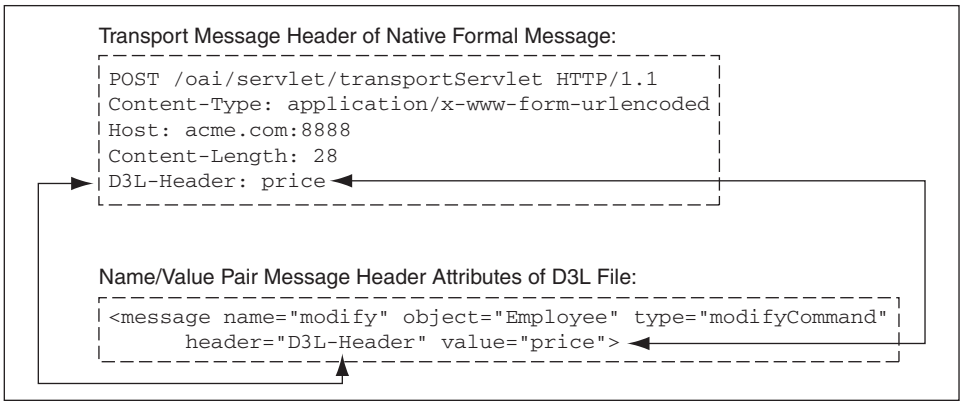
OracleAS Integration InterConnect Adapters, such as the HTTP adapter, make their protocol level transport properties available to the D3L translation engine, including custom properties added by a sender application, such as an HTTP client. The D3L file `message` element enables the user to specify two attributes, `header` and `value`, which match the protocol level headers in a received native format message.

For example, a third-party application uses the custom transport header D3L-Header to communicate with the D3L translation engine which D3L file to use to translate an incoming native format message. The following steps must be performed to set custom values in the transport header:

- Set the D3L-Header parameter in the transport `message` header to a value that matches the `value` attribute setting of the `<message>` element in the D3L file.
- Set the `header` attribute of the `<message>` element in the D3L file to D3L-Header to match the D3L-Header parameter name in the transport message header.

Figure C–2 illustrates using the HTTP adapter where D3L-Header and `price` are the header name and header value, respectively. The header name and value are used to match a native format message with the correct D3L file. The D3L translation engine retrieves the correct D3L file based on these settings.

Figure C–2 Name/Value Pair Message Header Attributes



The D3L engine supports a rudimentary pattern matching capability in the `value` attribute of the D3L `message` element.

A D3L author can create a D3L definition, such as:

```
<message type="CrtCust" header="filename" value="cust_create%" ...
```

This definition is for the FTP adapter, which provides a header property called `filename` that holds the name of a received file.

The preceding D3L destination will be selected to parse incoming files whose filenames match the name pattern in the `value` attribute, such as,

```
cust_create01
cust_create02
po_int_ext01
cust_create03
po_int_ext02
```

The "wildcard" character in the pattern ("`%`") can only appear at two places in the attribute string-value, as either the first or the last character, or both the first and last characters.

For example, the following patterns are acceptable:

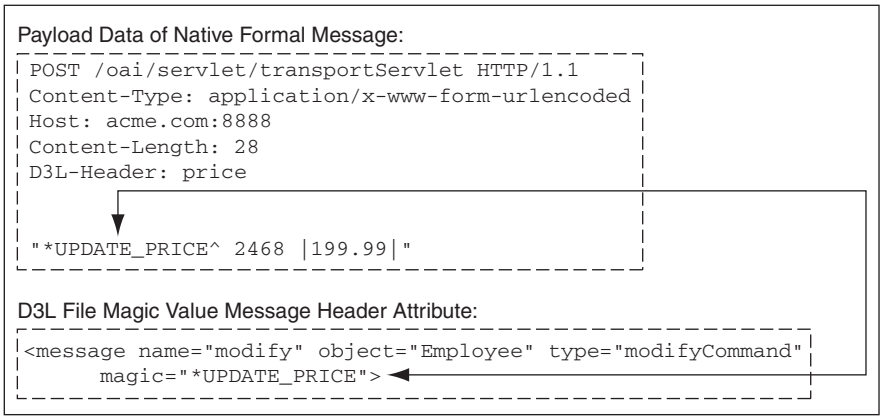
- `%endstring` : such as `%.csv`
- `startstring%`: such as `po_int%`
- `%substring%`: such as `%create%`

C.2.2.2.2 Magic Value Message Header Attribute You can set the magic attribute of the `message` element in the D3L file to match the first *n* bytes of data in a native format message. This feature enables you to define the D3L file to use with the native format

message. When a native format message is received by the D3L translation engine, the `magic` values of all D3L files are compared with the first *n* bytes of the native format message. The `magic` values must be long enough to be unique across all registered D3Ls for a given adapter instance.

Figure C–3 provides an example where `*UPDATE_PRICE` is the value that configures the native format message with the correct D3L file.

Figure C–3 Magic Value Message Header Attribute



The D3L translation engine retrieves the correct D3L file based on these settings.

The D3L attribute `startsat` of the `message` element enables the D3L author to specify the byte location to start the magic matching.

Having this attribute allows the D3L definition:

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">

<message name="newBook" type="BookType" object="BookObj" magic="ISBN#"
   startsat="12">
```

This D3L definition will trigger if a native message contains the byte character sequence `ISBN#` in byte positions 12 to 16.

See Also:

- "Native Format Message to Common View Incoming Message Translations" on page C-24
- "Additional D3L Sample Files and DTD" on page C-45

C.3 D3L File Structure

This section describes the contents of a sample D3L file named `book_reply.xml`.

```
1   <?xml version="1.0" encoding="US-ASCII"?>
2   <!DOCTYPE message SYSTEM "d3l.dtd">
3   <message name="replyFlight" type="BookingReplyType" object="Booking"
4   header="D3L-Header" value="replyOptions">
5      <unsigned4 id="u4" />
6      <unsigned2 id="u2" />
7      <struct id="DateTimeRecord">
8          <field name="DateInfo">
9             <date format="MMDDYY">
```

```
10              <pfxstring id="datstr" length="u4" />
11           </date>
12        </field>
13        <field name="TimeHour"><limstring delimiter="*" /></field>
14        <field name="TimeMinute"><limstring delimiter="*" /></field>
15    </struct>
16    <struct id="ItinRecord">
17        <field name="DepartureTime"><typeref type="DateTimeRecord" /></field>
18        <field name="ArrivalTime"><typeref type="DateTimeRecord" /></field>
19    </struct>
20    <pfxarray id="ItinArray" length="u2">
21        <typeref type="ItinRecord" />
22    </pfxarray>
23    <struct id="BookingReplyType">
24        <field name="AirportCodeFrom"><limstring delimiter="*" /></field>
25        <field name="AirportCodeTo"><limstring delimiter="*" /></field>
26        <field name="Itineraries"><typeref type="ItinArray" /></field>
27    </struct>
28   </message>
```

Lines 1 and 2

These lines define standard information, such as the Prolog and Document Type Declaration (DTD) that must have the specified values (for example, specifying `d3l.dtd` as the DTD).

Lines 3 and 4

These lines define the following message element attributes:

- `name`: It must correspond to the associated Oracle Application Server Integration InterConnect application view event name defined in iStudio. The D3L file can also be imported in iStudio when defining the message attributes of an event (the name of which must match the `name` attribute of the D3L message element).
- `type`: It names a structure that is defined in subsequent lines of this D3L file.
- `object`: It must match the Oracle Application Server Integration InterConnect business object defined in iStudio.
- `header`: It is identified in the set of protocol level transport message headers associated with a native format message.
- `value`: It must match the actual value of the corresponding protocol level transport message header defined through the `header` attribute.

See Also:

- "Creating Business Objects" on page 3-2
- "Creating Events" on page 4-2
- "Name/Value Pair Message Header Attributes" on page C-6
- "Task 6: Import a D3L File in iStudio" on page C-29

Lines 5 and 6

These lines define an unsigned, four-byte integer and unsigned, two-byte integer. These data type declarations are named `u4` and `u2`, respectively, so they can be referred to later.

Lines 7 Through 15

These lines define the following fields of a structure named `DateTimeRecord`:

- `DateInfo` defines a date format of `MMDDYY` and a length prefixed by an unsigned four-byte integer.
- `TimeHour` defines a string delimited by the character `*`.
- `TimeMinute` defines a string delimited by the character `*`.

Lines 16 Through 19

These lines define the fields of the structure named `ItinRecord`. The fields `DepartureTime` and `ArrivalTime` both consist of the `DataTimeRecord` structure.

Lines 20 Through 22

These lines define a length-prefixed array named `InitArray`, where each array element is of type `ItinRecord`.

Lines 23 Through 28

These lines define the following fields of the message structure `BookingReplyType`, which satisfies the `BookingReplyType type` declaration in the message document element:

- `AirportCodeFrom` is a string delimited by the character `*`.
- `AirportCodeTo` is a string delimited by the character `*`.
- `Itineraries` is a field of type `ItinArray`, which is an array of `ItinRecord`.

C.3.1 Supported D3L Data Types

D3L supports use of the following data types and declarations in a D3L file:

- Signed or Unsigned Integers
- Floating Point Numbers
- Strings
- Structures
- Sequences

C.3.1.1 Signed or Unsigned Integers

D3L supports signed or unsigned integers that can be one, two, four, or eights octets in size, and in big or little endian octet ordering.

Example C–1 Quantity Field

```
<field name="quantity">
   <unsigned4 endian="big" align="6"/>
</field>
```

The `quantity` field defines a four byte unsigned binary integer, using big (default) `endian`, and at an alignment of 6 bytes. For example, D3L will ensure that the reading or writing of this integer will start at a position in the buffer, so that `<`*`position`*`>` `modulus <`*`alignment`*`> = 0`.

Note: *Little Endian* means that the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. whereas *Big Endian* means that the high-order byte of the number is stored in memory at the lowest address, and the low-order byte at the highest address.

Data example

Byte addresses (*hex*):

```
00  01  02  03  04  05  06  07  08  09  0A  0B
```

Byte (*hex*):

```
00  00  00  00  00  00  80  FF  FF  FF  00  00
```

Parsed value (*dec*):

quantity = $128x256^3 + 255x256^2 + 255x256^1 + 255x256^0 = 2164260863$

Example C–2 Weight and Length Field

```
<field name="weight"> <unsigned2 align="3"/> </field>
<field name="length"> <unsigned2 align="3"/> </field>
```

The `weight` and `length` fields define two 2-byte unsigned binary integers, using big endian, and an alignment of 3 bytes.

Data example

Byte addresses (*hex*):

```
..  07  08  09  0A  0B  0C  0D  0E  ..
```

Byte (*hex*):

```
..  00  00  EE  88  00  22  F0  00  ..
```

Parsed value (*dec*):

weight = $238x256^1 + 136x256^0 = 61064$
length = $34x256^1 + 240x256^0 = 8944$

Example C–3 Temperature and Pressure Field

```
<field name="temperature"> <signed2 endian="little" />       </field>
<field name="pressure">    <unsigned4 endian="big" />        </field>
<field name="wind">  <unsigned2 endian="little" align="4" /> </field>
```

The `temperature` field defines a 2-byte signed binary integer, using little endian, and no alignment.

The `pressure` field defines a 4-byte unsigned binary integer, using big endian, and no alignment.

The `wind` field defines a 2-byte unsigned binary integer, using little endian, and a 4-byte alignment.

Data example

Byte addresses (*hex*):

```
..  60  61  62  63  64  65  66  67  68  69  ..
```

Byte (*hex*):

```
..  EF  FE  00  00  04  0A  00  00  3C  00  ..
    ^               ^               ^
    little end      big end         alignment
```

Parsed value (*dec*):

```
temperature = 256x256 - (239x256^0 + 254x256^1) = -273
pressure    =   4x256^1  + 10x256^0 = 1034
wind        =   60x256^0 +  0x256^1 = 60
```

C.3.1.2 Floating Point Numbers

D3L supports single- and double-precision, IEEE format, floating-point data. Single-precision floating point numbers (floats) take up four bytes or octets. Double-precision floating point numbers (doubles) take up eight bytes or octets.

Example C–4 Distance and Age Field

```
<field name="distance"> <double align="6"/> </field>
<field name="age">      <float />           </field>
```

The `distance` field defines an 8-byte double-float (floating-point value according to the IEEE 754 floating-point `double precision` bit layout), at an alignment of 6 bytes.

The `age` field defines a 4-byte single-float (floating-point value according to the IEEE 754 floating-point `single precision` bit layout).

Note: The IEEE 754 floating-point format is parsed and produced by the following Java class methods:

`java.io.DataInput.readFloat()`

`java.io.DataInput.readDouble()`

`java.io.DataOutput.writeFloat()`

`java.io.DataOutput.writeDouble()`

Data example

Byte addresses (*hex*):

```
..  77  78  03  04  05  06  07  08  09  0A  0B  xx  xx  ..
```

Byte (*hex*):

```
00  00  D2  47  D3  CE  16  2A  B1  A1  5E  5D  6B  0B  ..
        ^ double                        ^ float
```

Parsed value (*dec*):

```
distance = 1 x 10^38
age = 1 x 10^18
```

C.3.1.3 Strings

D3L supports the following string types:

- Constant length strings without delimiters, with optional padding to fill out empty spaces.
- Delimited strings can be delimited by an arbitrary delimiter character.
- Length-prefixed strings where the length prefix is a numeric type. Numeric types are binary integer types described in "Signed or Unsigned Integers" or are a number stored as a string.
- Strings terminated by a specified character.
- Strings terminated by a delimiter defined by an enclosing `limarry` structure.
- Four date formats: `MMDDYY`, `DDMMYY`, `MMDDYYYY`, `DDMMYYYY`, where the information is stored as a string in one of these formats with any separator character between month, date, and year, such as `12!24=01`).
- Numbers not defined as binary data, but as strings. Any string format can define a number (either an integer or a floating-point entity). In iStudio, a D3L field of type `number` is handled as a double.

Example C–5 Constant length strings

```
<field name="CURRENCY_CODE">
   <padstring length="4" padchar=" " padstyle="tail"/>
</field>
<field name="COUNTRY_CODE">
   <padstring length="2" padchar="" padstyle="none"/>
</field>
<field name="TO_USD_RATE">
   <padstring length="12" padchar="0" padstyle="head"/>
</field>
```

The `CURRENCY_CODE` field defines a fixed length string of four characters. Any blank (" ") characters (pads) near the end (`padstyle="tail"`) of the string are not considered part of the data value.

The `COUNTRY_CODE` field defines a fixed length string of two characters. All characters in this field are part of the data value because padstyle is "`none`".

The `TO_USD_RATE` field defines a fixed length string of 12 characters. Any zeros at the beginning (`padstyle="head"`) of the string are not considered part of the data value.

Data example

`Native byte` (*`character`*) stream:

```
     GBP UK000012550.00
```

`Parsed value` s:

```
    CURRENCY_CODE = 'GBP'
    COUNTRY_CODE  = 'UK'
    TO_USD_RATE   = '12550.00'
```

Example C–6 Delimited strings

```
<field name="State">    <limstring delimiter="." /> </field>
<field name="Region">   <limstring delimiter="." /> </field>
<field name="City">     <limstring delimiter="|" /> </field>
<field name="Landmark"> <limstring delimiter="|" /> </field>
```

```
<field name="Street">   <limstring delimiter="+" /> </field>
```

The "State", "Region", "City", "Landmark", and "Street" fields are delimited strings enclosed by ".", ".", "|", "|", and "+", respectively .

Data example

Native byte (*character*) stream:

```
.FL..Florida Keys.|Key West||Ernest Hemingway Museum|+Whitehead St.+
```

*Parsed value*s:

```
State = 'FL'
Region = 'Florida Keys'
City = 'Key West'
Landmark = 'Ernest Hemingway Museum'
Street = 'Whitehead St.'
```

Example C–7 Length prefixed strings

```
<unsigned1 id="ubyte1" />
<unsigned2 id="ubyte2" endian="little" />
<struct>
   <field name="user">      <pfxstring length="ubyte1" /> </field>
   <field name="encr_user"> <pfxstring length="ubyte2" /> </field>
```

The user field defines a string the length of which is defined by a 1-byte binary integer preceeding the string contents.

The encr_user field defines a string the length of which is defined by a 2-byte binary integer preceeding the string contents.

Data example

Byte addresses (*hex*):

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12

Characters:

03 j o e **0D 00** D U Z a c . 1 H K V m I Y

Parsed values:

```
user = 'joe'
encr_user = 'DUZac.1HKVmIY'
```

Example C–8 Terminated strings

```
<field name="product">   <termstring endchar=","/ > </field>
<field name="ordered">   <termstring endchar=","/ > </field>
<field name="inventory"> <termstring endchar=","/ > </field>
<field name="backlog">   <termstring endchar=","/ > </field>
<field name="listprice"> <termstring endchar="\n"/> </field>
```

Each of the first four fields is populated with input characters until the terminating/ending character (endchar) is encountered (","). The last field ends with a linefeed.

Data example

Native byte (*character*) stream:

```
1020,16,18,,1580.00<LF>
```

Parsed values:

```
product = '1020'
ordered = '16'
inventory = '18'
backlog = ''
listprice = '1580.00'
```

Note: The backlog field is empty.

Example C–9 Simple strings - `simplestring`

```
<limarray id="CSV_Type" contchar="," endchar="\n">
   <simplestring/>
</limarray>
<struct>
   <field name="CSV"> <typeref type="CSV_Type"/> </field>
```

The `CSV` field references a type declaration "`CSV_Type`" to a delimited array. The array members are separated by comma (`contchar=","`) and end with linefeed (`endchar="\n"`).

Data example

Native byte (*character*) stream:

```
5,18,2.5,255,78.75,9
```

Parsed values:

```
CSV[] = { '5', '18', '2.5', '255', '78.75', '9' }
```

Example C–10 Dates - date

```
<field name="StartDate">
   <date format="MMDDYY"> <termstring endchar="\n"/> </date>
</field>
<field name="EndDate">
   <date format="DDMMYY"> <termstring endchar="\n"/> </date>
</field>
<field name="Milestone">
   <date format="MMDDYYYY"> <termstring endchar="\n"/> </date>
</field>
<field name="DueDate">
   <date format="DDMMYYYY"> <termstring endchar="\n"/> </date>
</field>
```

The fields contain dates representing the four date formats.

Data example

Byte stream (*characters*):

```
11/16/02<LF>
24/11/02<LF>
11/20-2002<LF>
23*11*2002<LF>
```

Parsed values:

```
StartDate = Sat Nov 16 00:00:00 PST 2002
EndDate   = Sun Nov 24 00:00:00 PST 2002
Milestone = Wed Nov 20 00:00:00 PST 2002
DueDate   = Sat Nov 23 00:00:00 PST 2002
```

Note: The D3L parser will accept any character between the DD, MM and YY(YY) characters in the native format, but will always produce the "/" separator when translating from application message format to native message format. Hours, minutes, and seconds are not parseable.

Example C–11 String based numbers

```
<unsigned1 id="u1" />
<pfxstring id="HueType" length="u1" />

<struct id="ColorDefinition">
   <field name="Red">
      <number> <padstring length="4" padstyle="head" padchar="0"/> </number>
   </field>
   <field name="Green">
      <number> <pfxstring length="u1" /> </number>
   </field>
   <field name="Blue">
      <number> <limstring delimiter="."/> </number>
   </field>
   <field name="Brightness">
      <number> <termstring endchar="|"/> </number>
   </field>
   <field name="Hue">
      <number> <typeref type="HueType"/> </number>
   </field>
</struct>
```

The declared type u1 is an unsigned 1-byte integer (0-255). The second type declaration "HueType" is a length prefixed string, where the string length will be defined in a one-byte binary integer preceding the string contents.

The Red field is a number defined as a fixed length string of 4 characters, which can be padded with zeros at the beginning.

The Green field is a number defined as a length prefixed string, where the string length will be defined in a one-byte binary integer preceeding the string contents.

The Blue field is a number defined as a "." delimited string. The string beginning and end is demarcated by ".".

The Brightness field is a number defined as a string which is read from the current point until the ending character ("|") is encountered.

The Hue field is a number defined as a string of type "HueType".

Data example

Byte addresses (hex) and characters (hex values shown in italics):

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 0  1  2  8 03  1  2  8  .  2  5  5  .  0  .  7
```

```
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E
5  3  3  3  3  | 08  0  .  6  6  6  6  6  6
```

Parsed values:

```
Red = 128.0
Green = 128.0
Blue = 255.0
Brightness= 0.753333
Hue = 0.666666
```

Note: The parsed numbers always internally become doubles.

C.3.1.4 Structures

D3L supports structured types, such as ordered records containing other data types (predefined or user defined). Types can be nested to arbitrary depth. This means you can use structures of sequences of structures of sequences to any finite depth. Recursive, self referencing, and data structures are not supported in D3L.

All data fields in a message format description must be named. These names are used as Oracle Application Server Integration InterConnect message attribute names. All names within the same structure must be mutually unique.

Within a D3L file the first element should be `message`. The `message` element must refer to a `struct` (using the `IDREF type` attribute), which then becomes the top-level data structure of the message.

Example C–12 ColorDefinition Field

```
<message type="ColorDefinition" name="myEV" object="myBO">
   <unsigned1 id="u1" />
   <pfxstring id="HueType" length="u1" />
   <struct id="ColorDefinition">
   <field name="Red">  ...
<field name="Green">  ...
```

Note: The top-level structure can be placed anywhere in the D3L file within the scope of the `message` element.

C.3.1.5 Sequences

D3L supports sequences, such as arrays of various types. These include:

- Delimited arrays (with arbitrary separator and terminator characters)
- Length-prefixed arrays (where the length is one of the numeric types)
- Fixed-length arrays
- Implicit-length arrays (which use all remaining data in the native format message to the end of the buffer)

The data being sequenced can be any other D3L type (predefined or user defined).

Example C–13 Delimited arrays

```
<field name="members">
   <limarray contchar=";" endchar=".">
      <limstring delimiter="." />
```

```
   </limarray>
</field>
```

The `members` field becomes an array of data elements separated by semicolons (`contchar=";"`). The end of the array is marked by a period (`endchar="."`). Each data element in the array is a string delimited by a set of periods (`delimiter="."`).

Data example

Native byte (character) stream:

```
.John.;.Steve.;.Paul.;.Todd..
```

Parsed values:

```
members[] = { 'John', 'Steve', 'Paul', 'Todd' }
```

Example C–14 Length prefixed arrays

```
<unsigned2 id="u2" endian="little" align="4" />
<struct>
   <field name="measurements">
      <pfxarray length="u2" > <signed1 /> </pfxarray>
   </field>
```

The `measurements` field becomes an array of signed 1-byte binary integers (signed1). The number of elements in the array is determined by the unsigned two-byte binary integer at the beginning of the array.

Data example

Byte addresses (*hex*):

```
..  08  09  0A  0B  0C  0D  0E  0F  10  ..
```

Bytes (*hex*):

```
..  06  00  FF  A2  6C  24  0E  77  ..
```

Values (*dec*):

```
measurements[] = { -1, -94, 108, 36, 14, 119 }
```

Example C–15 Fixed length arrays

```
<field name="digits">
   <fixarray length="10">
      <number>
         <termstring endchar="-">
      </number
   </fixarray>
```

The `digits` field becomes an array of numbers (doubles). Each number element in the native byte format is represented as a string which is terminated by a dash (`endchar="-"`). The number of elements in the array must always be 10 (`length="10"`).

Data example

Native byte (*character*) stream:

```
1-2-3-4-5-6-7-8-9-0-
```

Parsed values:

```
digits[] = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 0.0 }
```

Example C–16 Implicit length arrays

```
<message name="addOrders" object="Order" type="OrdersType">

   <number id="Number"> <termstring endchar="," />  </number>
   <number id="Price">  <termstring endchar=";" /> </number>
   <number id="Total">  <termstring endchar="\n"/> </number>

   <struct id="OrderLineType">
      <field name="LineNo">    <typeref type="Number" /> </field>
      <field name="ProductNo"> <typeref type="Number" /> </field>
      <field name="Quantity">  <typeref type="Number" /> </field>
      <field name="LinePrice"> <typeref type="Price " /> </field>
   </struct>

   <struct id="OrderType">
      <field name="OrderTotal"> <typeref type="Total" /> </field>
      <field name="OrderLines">
      <limarray contchar="\n" endchar="\n\n">
         <struct>
            <field name="OrderLine"> <typeref type="OrderLineType" /> </field>
         </struct>
      </limarray>
   </field>
</struct>

   <number id="ID"> <termstring endchar="\n" /> </number>

   <imparray id="OrdersArrayType">
      <struct>
         <field name="CustomerID"> <typeref type="ID" /> </field>
         <field name="CustomerName"> <termstring endchar="\n" /> </field>
         <field name="Order"> <typeref type="OrderType" /> </field>
      </struct>
   </imparray>

   <struct id="OrdersType">
      <field name="OrdersArray"> <typeref type="OrdersArrayType" /> </field>
   </struct>
</message>
```

The `OrdersType` structure consists of a single field `OrdersArray` which is an implicit array of three fields: a `CustomerID`, `CustomerName` and `Order` (of type `OrderType`). `OrdersArrayType` is an implicit array. As a result, it will consume all remaining bytes in the native byte input stream. The size of the array is first known when the input byte stream has been exhausted.

The `OrderLines` field is a (nested) array, where each array element is of type `OrderLineType`. The `OrderLineType` is a structure of four fields: `LineNo`, `ProductNo`, `Quantity`, and `LinePrice`.

This input stream must use the following structure to be parseable:

```
CustomerID
CustomerName
Order:
```

```
 OrderTotal
OrderLines:
LineNo, ProductNo, Quantity, LinePrice
LineNo, ProductNo, Quantity, LinePrice
...
```

Data example

Native byte (character) stream:

```
1234
Boeing
1000
1,555,10,250.00;
2,666,10,750.00;

5678
Lockheed Martin
424
1,555,5,125.00;
2,777,1,100.00;
3,888,2,199.00;
```

Parsed values:

```
{OrdersArray=
   [ { CustomerName=Boeing, CustomerID=1234.0,
      Order=
         { OrderTotal=1000.0,
            OrderLines = [
         {OrderLine={LinePrice=250.0, ProductNo=555.0, Quantity=10.0,
            LineNo=1.0}},
         {OrderLine={LinePrice=750.0, ProductNo=666.0, Quantity=10.0,
            LineNo=2.0}}
         ]
      }
   },
{ CustomerName=Lockheed Martin, CustomerID=5678.0,
   Order=
      { OrderTotal=424.0,
        OrderLines = [
      { OrderLine={LinePrice=125.0, ProductNo=555.0, Quantity=5.0,
          LineNo=1.0}},
      { OrderLine={LinePrice=100.0, ProductNo=777.0, Quantity=1.0,
          LineNo=2.0}},
        { OrderLine={LinePrice=199.0, ProductNo=888.0, Quantity=2.0,
         LineNo=3.0}}
        ]
     }
}
 ]
}
```

C.3.1.6 Data Padding

D3L supports data padding. Padding is unnamed gaps in a native format message that satisfy alignment constraints of the underlying native system. Padding is discarded in the Oracle Application Server Integration InterConnect application view message.

The following D3L example defines a number as a left-aligned string, which is padded at the end with blanks to a field width of 10.

```
<field name="Quantity">
   <number>
      <padstring length="10" padchar=' ' padstyle="tail" />
   </number>
```

The following native byte (character) stream satisfies this format:

```
9876.5____
```

Pads can also be explicitly defined between fields in a structure by using the <pad> element.

The following D3L example shows two fields, which are separated by a pad of size 10.

```
<struct id="PROD">
   <field name=PRODID"> <termstring endchar=";" /> </field>
   <pad length="10" />
   <field name=PRODDESC"> <termstring endchar=";" /> </field>
</struct>
```

The following native byte (character) stream would satisfy this format:

```
48682HW;~~~~~~~~~~WASHER AND DRYER;
[...]
```

C.3.2 Comma-Separated Values File Parsing with D3L

A comma-separated values (CSV) file consists of multiple lines. Each line contains values separated by commas that end when a new line is required:

```
a,b,c,d
1,2,3
```

The string types, termstring and simplestring, have been added to parse CSV files.

- termstring: String type `termstring` is a variation of `limstring`. It requires only a terminating delimiter, but not a beginning delimiter. For example:

  ```
  <termstring endchar="," />
  ```

  This parses any string contents until encountering a comma.

- simplestring: String type `simplestring` is a special data type. It is used when the nearest parent structure defines a valid set of delimiters, which for the current data definition description language (D3L) library is limited to `limarray`. For example:

  ```
  <limarray contchar="," endchar="\n">
     <simplestring />
</limarray>
  ```

The examples provided in the following sections use `imparray` so that input can be any number of elements, lines, or both.

C.3.2.1 CSVs are Assigned to Named Fields

This method assigns all CSVs on each line to named fields fixed number of fields per line. Example C–17 describes CSVs assigned to named fields.

Example C–17 CSVs Assigned to Named Fields

```
<message name="createPhone" object="Phone" type="phoneRecord">
```

```
<imparray id="lines">
   <struct>
      <field name="rectype"> <termstring endchar=","/ > </field>
      <field name="quantity"><termstring endchar=","/ > </field>
      <field name="endHour"> <termstring endchar=","/ > </field>
      <field name="endMin">  <termstring endchar=","/ > </field>
      <field name="cost">    <termstring endchar="\n"/> </field>
   </struct>
</imparray>
<struct id="phoneRecord">
   <field name="csv"> <typeref type="lines" /> </field>
</struct>
</message>
```

The native format message payload for Example C–17 is as follows:

```
4,,9,22,2324.29
,,,,,
55,2342,11,46,728372339.57
```

C.3.2.2 All CSVs are Read into an Array

This method read all CSVs on each line into an array. There are variable number of fields per line. Example C–18 describes all the CSVs read into an array.

Example C–18 All CSVs are Read into an Array

```
<message name="createPhone" object="Phone" type="phoneRecord">
<limarray id="linearr" contchar="#44" endchar="\n">
<simplestring />
</limarray>
<imparray id="myArray">
<struct>
<field name="line"> <typeref type="linearr" /> </field>
</struct>
</imparray>
<struct id="phoneRecord">
<field name="csv"> <typeref type="myArray" /> </field>
</struct>
</message>
```

The native format message for Example C–18 is as follows:

```
4,,9,22,2324.29
55,2342,11,46,728372339.57
55,2342,11,46,728372339.57,4,,9,22,2324.29
1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0
```

C.3.2.3 Delimiter Encoding Styles

The delimiters for `limstring`, `termstring`, and `limarray` are enhanced to allow multiple characters, as well as additional encoding styles. The associated ASCII table codes are shown in parentheses:

- Escape code using `"\"`: This works for `"\r" (13)`, `"\n" (10)`, `"\t" (9)`, and `"\f" (12)`.

  where:

  - `(13)` is the ASCII code for a carriage return (CR)

- `(10)` is the ASCII code for a line feed (LF)
- `(9)` is the ASCII code for a horizontal tab (HT)
- `(12)` is the ASCII code for a form feed (FF)

- Escape ASCII code using "#": for example, "`#13`".
- Escape ASCII hexadecimal code using "*#x*": for example, "*#x0D*".
- End-of-file delimiter "`\eof`", which maps to a virtual end-of-file character: This delimiter can only be used once. No other fields can follow once it has been used.

Example C–19 provides several examples of delimiter encoding styles.

Example C–19 Delimiter Encoding Styles

```
<termstring endchar="#x2C"/>
<termstring endchar="\n"/>
<limarray id="linearr" contchar="," endchar="\r\n">
   <simplestring/>
</limarray>

<termstring id="FileContents" endchar="\eof"/>
```

The "`\r\n`" on line 3 of Example C–19 represents a DOS line break. The "\eof" on the last line of Example C–19 represents an End of File.

C.4 D3L Integration with OracleAS Integration InterConnect Adapters

This section provides information on the D3L translation engine and its files' integration with the OracleAS Integration InterConnect adapter agent and bridge subcomponents to perform events and translations. It contains the following topics:

- Runtime Initialization
- Native Format Message to Common View Incoming Message Translations
- Common View to Native Format Message Outgoing Messages Translations

C.4.1 Runtime Initialization

The OracleAS Integration InterConnect Adapter agent reads `.ini` files (such as `adapter.ini`) at runtime to access each OracleAS Integration InterConnect adapter's configuration information. The OracleAS Integration InterConnect Adapter bridge are initialized and the configuration information provided by the OracleAS Integration InterConnect adapter agent. At the completion of a successful initialization, the OracleAS Integration InterConnect adapter bridge knows the following:

- The Oracle Application Server Integration InterConnect application name and its default endpoint (message destination)
- The various Oracle Application Server Integration InterConnect events to be handled by the OracleAS Integration InterConnect Adapter bridge
- D3L files that describe each of these events
- D3L files that are accessible and valid. If a file is invalid, then the OracleAS Integration InterConnect adapter cannot start

See Also:

- Chapter 1, "Getting Started with OracleAS Integration InterConnect"
- Chapter 2, "Using iStudio"

C.4.2 Native Format Message to Common View Incoming Message Translations

When the OracleAS Integration InterConnect adapter common transport layer detects an incoming message from an application, it receives the message in its native format. The message is then passed to the OracleAS Integration InterConnect adapter bridge. The bridge performs the following functions:

- Using the D3L translation engine to translate the native format message into an application view an Oracle Application Server Integration InterConnect message object.
- Raises an application view event

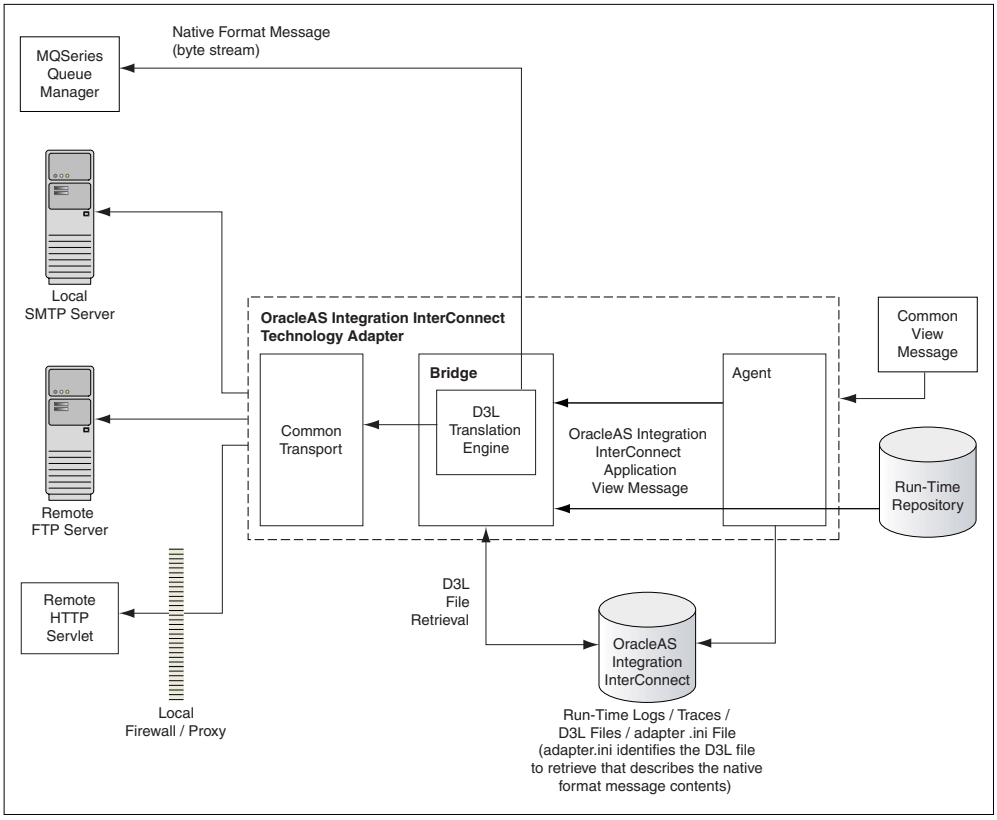
The agent transforms the application view event into a common view event and passes it on for further routing and processing. Table C–1 describes the data flow sequence if D3L message header attributes are used.

Table C–1 Message Header Attributes

If The...	Then...
Name/value pair message header attributes are used	The incoming native event might contain one of the following: ■ Transport message headers/properties (made available to the bridge by the transport layer) ■ Transport message header parameter name (for example, `D3L-Header`) matches the header attribute of the `message` element in the D3L file (header="D3L-Header") ■ Transport message header value (for example, `D3L-Header: price`) matches the value attribute of the `message` element in the D3L file (value="price") In the preceding cases, the bridge assumes that the matching D3L describes the incoming native event. Any conflicting header and value settings are detected and rejected by the bridge during initialization time. These operations are logged by OracleAS Integration InterConnect logging and tracing APIs for debugging, performance analysis, and business intelligence functions.
Magic value message header attribute is used	A magic value is specified by using the following: ■ The D3L file (length = *n* bytes) ■ The first *n* bytes of payload data in an incoming native event (for example, `*UPDATE_PRICE`) match the `magic` attribute of the `message` element in the D3L file (for example, `magic="*UPDATE_PRICE"`) In the preceding cases, the bridge assumes the native event must be processed using the matching D3L. If multiple D3Ls specify magic values that may match the same native event, the bridge randomly picks a D3L. This can lead to undesirable bridge behavior because the resulting application view event raised may not be the correct one.

Figure C–4 depicts the data flow sequence.

Figure C–4 Native Format Message to Common View Incoming Messages



See Also:

- "Message Header Attributes" on page C-6
- Section C.6.4, "Task 4: Configure a Native Format Message with a D3L File" on page C-28

C.4.3 Common View to Native Format Message Outgoing Messages Translations

When a common view event is raised, the OracleAS Integration InterConnect Adapter agent subscribing to the event, performs the following:

- Receives the associated message
- Transforms it to an Oracle Application Server Integration InterConnect message object
- Hands it to the OracleAS Integration InterConnect adapter bridge as an application view event

The OracleAS Integration InterConnect Adapter bridge queries the metadata associated with the event to determine the following:

- The D3L file for the D3L translation engine for the translation of the application view event into a native format message.
- The application to which the native format message event will be sent. There are two levels of rules to determine the application endpoint (the destination) of an OracleAS Integration InterConnect event:

- If the event contains metadata that specifies an endpoint, then the bridge uses this endpoint for the event. With the exception of the MQ Series adapter, all OracleAS Integration InterConnect adapters follow this rule.

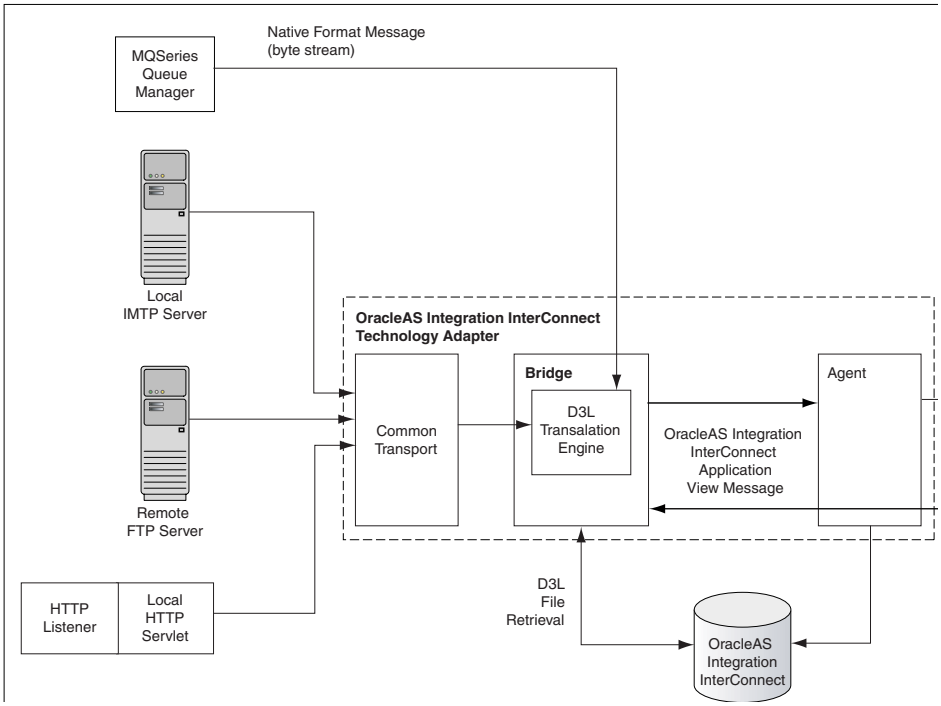
  Note: Here the metadata itself names the endpoint and the content of the event is not searched.

- If the message metadata did not specify an endpoint, then the bridge uses its default endpoint, specified in the `adapter.ini` file, and made available to the bridge during initialization.

All OracleAS Integration InterConnect adapter operations are logged using the Oracle Application Server Integration InterConnect logging and tracing APIs for debugging, performance analysis, and other business intelligence functions.

Figure C–5 shows the data flow sequence.

Figure C–5 Common View to Native Format Message Outgoing Messages



C.5 Installing D3L

D3L is automatically installed with OracleAS Integration InterConnect.

See Also:

- *Oracle Application Server Installation Guide*
- *OracleAS Integration InterConnect Adapter documentation*

C.6 Configuring D3L

After installation, perform the following tasks to configure D3L:

C.6.1 Task 1: Configure D3L with iStudio

You must define D3L in the `browsers.init` file. This enables you to import D3L files as attributes and select D3L as the message type in iStudio.

To integrate D3L with iStudio:

1. Use a text editor to open the

   ORACLE_HOME`\integration\interconnect\iStudio\browsers.init` file.

2. Add the following information at the end of the file:

   ```
   D3L;oracle.oai.agent.adapter.technology.D3LBrowser;
   ```

3. Save your changes and exit the file.

See Also: "Task 6: Import a D3L File in iStudio" on page C-29

C.6.2 Task 2: Create a Native Format Message

The native format is typically predefined by your third-party application.

1. Create a native format message. For example, this native format message updates the salary of employee number `33201` to `55000`:

   ```
   *UPDATE_EMPLOYEE_SALARY* 33201 |55000|
   ```

Where...	Is...
UPDATE_EMPLOYEE_SALARY	action
33201	EmployeeID
55000	newSalary

C.6.3 Task 3: Create a D3L File Describing the Native Format Message

To create a D3L file (for example, updemp.xml) that describes the format of the native message:

1. Use a text editor. The following D3L file describes the contents of the native format message created in "Task 2: Create a Native Format Message".

   ```
   <?xml version="1.0" encoding="US-ASCII"?>
   <!DOCTYPE message SYSTEM "d3l.dtd">
   <message name="modify" object="Employee" type="modifyCommand"
      header="D3L-Header" value="employee">
       <struct id="modifyCommand">
           <field name="action"><limstring delimiter="*"/></field>
           <field name="EmployeeID"><limstring delimiter=" "/></field>
   ```

```
            <field name="newSalary"><limstring delimiter="|"/></field>
        </struct>
    </message>
```

2. Store the D3L file in the *ORACLE_HOME*\integration\interconnect\adapters*application* directory for direct access at deployment time.

   See Also: The following sections for additional examples of D3L files:

   - Figure C–1 on page C-5
   - Example C–20 on page C-33
   - "Additional D3L Sample Files" on page C-46

C.6.4 Task 4: Configure a Native Format Message with a D3L File

Configure a native format message with the correct D3L file. This enables the D3L translation engine to use the correct D3L file to verify native format message contents. For example, the D3L file created in "Task 3: Create a D3L File Describing the Native Format Message" includes settings for name/value pair message header attributes:

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">
<message name="modify" object="Employee" type="modifyCommand"
   header="D3L-Header" value="employee">
```

These settings can match with the transport message header D3L-Header parameter name and `employee` value of a native format message:

```
POST /oai/servlet/transportServlet HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: acme.com:8888
Content-Length: 38
D3L-Header: employee
```

See Also:

- Figure C–2 on page C-7
- Table C–1 on page C-24

C.6.5 Task 5: Configure D3L with OracleAS Integration InterConnect Adapters

The `adapter.ini` file is read by the appropriate OracleAS Integration InterConnect Adapter at startup.

1. Use a text editor to open the

   ORACLE_HOME\integration\interconnect\adapters*application*\adapter.ini file.

   In the preceding directory, *application* is the name of your application and the value of the `application` parameter in the `adapter.ini` file.

2. Ensure that parameter `ota.type` is set to the following value:

   ```
   ota.type=D3L
   ```

This defines D3L as the message type for the OracleAS Integration InterConnect Adapter to handle incoming and outgoing messages.

3. Add the following line to define the D3L files for the bridge and D3L translation engine to use:

```
ota.d3ls=updemp.xml
```

where updemp.xml is the name of the D3L file created in "Task 3: Create a D3L File Describing the Native Format Message". Each event handled by the bridge must have its own D3L file. Whenever a new D3L file is imported in iStudio for use by an application, this parameter must be updated and the OracleAS Integration InterConnect adapter restarted.

4. Save your changes and exit the file.

C.6.6 Task 6: Import a D3L File in iStudio

iStudio enables you to import a D3L file for use with the following OracleAS Integration InterConnect features:

- Common data types
- Application data types
- Published/subscribed events
- Invoked/implemented procedures
- Business object events and procedures

When a D3L file is associated with Oracle Application Server Integration InterConnect common data types, application data types, events, or procedures, an iStudio OracleAS Integration InterConnect Adapter browser plug-in verifies that the file conforms to the syntax and semantics of D3L. Table C–2 identifies the iStudio tasks and locations where you can import a D3L file as an attribute and select D3L as a message type. Documentation references that describe how to perform these tasks are also provided.

Table C–2 D3L Functionality in iStudio

For this D3L Functionality...	Do This...
Common Data Type Tasks:	
■ Create a common data type that imports a D3L file as an attribute.	Refer to "Creating Common Data Types" on page 3-2
Application Data Types Tasks:	
■ Creating an application data typedata type that imports a D3L file as an attribute.	Select New from File, then select Application Data Type from the iStudio menu
Event Tasks:	
■ Create an event that imports a D3L file as an attribute.	Refer to "Creating Events" on page 4-2
■ Publish an event that uses D3L as the message type and imports a D3L file as an attribute.	Refer to "Publishing an Event" on page 4-3
■ Subscribe to an event that uses D3L as the message type and imports a D3L file as an attribute.	Refer to "Subscribing to an Event" on page 4-8
Procedure Tasks:	

Table C–2 (Cont.) D3L Functionality in iStudio

For this D3L Functionality...	Do This...
■ Create a procedure that imports a D3L file as an attribute.	Refer to "Creating a Procedure" on page 5-2
■ Invoke a procedure that uses D3L as the message type and imports a D3L file as an attribute.	Refer to "Invoking a Procedure" on page 5-3
■ Implement a procedure that uses D3L as the message type and imports a D3L file as an attribute.	Refer to "Implementing a Procedure" on page 5-7

Note: D3L functionality with procedures in iStudio is only available with the MQ Series adapter.

C.6.7 Task 7: Define Metadata Properties with Each Event (Optional)

You can associate metadata with each event in iStudio by selecting Modify Fields in the Subscribe Wizard - Define Application View. The Modify Fields is displayed after you select D3L as the Message Type in the preceding Subscribe Wizard - Select an Event. Such metadata is used for content-based routing of events at runtime.

The following application view event metadata is used by the OracleAS Integration InterConnect adapters. The property name is prefixed by `ota` to minimize namespace conflicts with user-defined metadata on application view events. The property name is considered a keyword/reserved name, and is used by both iStudio and the bridge, and must be kept consistent between these two components.

Property Name	Property Value Type	Explanation
`ota.d3lPath`	The D3L filename (string). This is automatically set. Do not modify this property.	The path name (relative or absolute) of the file that contains the D3L guidelines for this event.
`ota.isD3L`	This value is always true (boolean) and automatically set. Do not modify this property.	A flag indicating that this event is based on D3L.
`ota.send.endpoint`	The endpoint URL (string). This is mandatory. For example: `http://foo.com/servlet/test`	The actual endpoint to which this message is sent. This setting must match the type of OracleAS Integration InterConnect Adapter that subscribes to the event.
`http.sender.*` `file.sender.*`	Refer to Chapter 2 of the appropriate OracleAS Integration InterConnect Adapter documentation for the adapter being defined in the `ota.send.endpoint` parameter URL[1]). This is optional. For example: `http.sender.timeout=5000`	The properties define the transport layer configuration.

[1] The MQ Series adapter does not define any `smtp.sender` properties. This is because the MQ Series adapter does not support multiple sending endpoints in this release.

See Also: "Subscribing to an Event" on page 4-8

C.7 D3L Use Case

This section contains these topics:

- D3L Use Case Overview
- Creating Data Type Definitions for Application Views
- Configuring the aqapp_pub and fileapp_sub Applications in iStudio
- Installing the Advanced Queuing and FTP Adapters
- Running the D3L Use Case
- Using Other Adapters in D3L and XML Modes

C.7.1 D3L Use Case Overview

This use case provides an example of a minimal Oracle Application Server Integration InterConnect configuration and setup that uses D3L. This use case involves two applications using OracleAS Integration InterConnect Adapters:

- `aqapp_pub`, which is based on the Advanced Queuing adapter
- `fileapp_sub`, which is based on the FTP adapter running in D3L mode

These applications use a business object called `Employee`, which has one defined event called `newEmployee`.

`The aqapp_pub` application publishes the `newEmployee` event, while `fileapp_sub` subscribes to it. Table C–3 describes the attributes (message structure) of the `newEmployee` event:

Table C–3 Common View Attributes

Attribute Name	Attribute Type
`EmpName`	String
`EmpDept`	Integer
`EmpHiredate`	Date
`EmpSalary`	Double

All these attributes are scalar (that is, there are no arrays). This message structure represents the common view of the `newEmployee` event. For simplicity, the application views for the two applications have the exact same structure as the common view.

In "Creating Data Type Definitions for Application Views", a DTD file and a D3L file are created that match the common view attributes shown in Table C–3 on page C-31. These files are used when the application views for the two applications are defined.

C.7.2 Creating Data Type Definitions for Application Views

You must create data type definitions for the two application views.

This section contains these topics:

- Task 1: Create a DTD File for the Advanced Queuing Adapter

- Task 2: Create a D3L File for the FTP Adapter

> **Note:** This use case assumes that you have already installed and configured Oracle Application Server Integration InterConnect and iStudio.

C.7.2.1 Task 1: Create a DTD File for the Advanced Queuing Adapter

The application view for the Advanced Queuing adapter must be defined through a DTD. The DTD enables the Advanced Queuing adapter to translate a received XML (text) document into a runtime application view (Java) object. The agent component of the Advanced Queuing adapter can then transform it to a common view object before routing it to any application subscribers. A DTD is registered with (imported to) the application while defining, for example, a publication in iStudio.

1. Create a DTD file that matches the common view message structure shown in Table C–3 on page C-31:

```
<!ELEMENT NewEmpRec (EmpName, EmpDept, EmpHiredate, EmpSalary)>
<!ELEMENT EmpName     (#PCDATA)>
<!ELEMENT EmpDept     (#PCDATA)>
<!ELEMENT EmpHiredate (#PCDATA)>
<!ELEMENT EmpSalary   (#PCDATA)>
```

2. Save this DTD in a text file named `newemp.dtd`. This file can be saved to any location.

C.7.2.2 Task 2: Create a D3L File for the FTP Adapter

When running in D3L mode, the FTP adapter must have its application view defined by a D3L (XML) file. The D3L file enables a bidirectional translation between the internal runtime application view (Java) object representation and an external binary/native format message representation. The D3L file is registered with (imported to) the application while defining, for example, a subscription in iStudio.

Assume the external binary native format message of the `newEmployee` event is as follows:

```
message ::= <empname> <empdept> <emphiredate> <empsalary>
empname ::= char[20] // left adjusted string, 20 chars wide, right padded with
spaces
empdept ::= byte[2]   // unsigned 2-byte integer, little endian
emphiredate ::= '|' + <month> + <anysep> + <day> + <anysep> + <year> + '|'
empsalary ::= '$' <number> '$'
```

Where...	Is...
`<month>`, `<day>`, and `<year>`	The date format elements `MM`, `DD`, and `YYYY` (all digits)
`<anysep>`	Any single character
`<number>`	Any decimal number using the character "`.`" as a decimal separator

1. Create a D3L file that describes the structure that the native format message must follow to communicate with Oracle Application Server Integration InterConnect. The native format message can be expressed/mapped in the D3L XML definition as shown in Example C–20:

Example C–20 D3L Sample File

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">
<message type="NewEmpRec" name="newEmployee" object="Employee">
   <!-- TYPE DECLARATIONS -->
      <!-- string field 20 chars wide with trailing spaces -->
      <padstring id="str20" padchar=" " padstyle="tail" length="20" />
      <!-- unsigned 2-byte integer -->
      <unsigned2 id="uword" endian="little" />
      <!-- date format using pattern MM-DD-YYYY enclosed by '|' -->
      <date id="date" format="MMDDYYYY"><limstring delimiter="|" />
      </date>
      <!-- decimal number format enclosed by '$' -->
      <number id="number"><limstring delimiter="$" /></number>
<!-- MESSAGE STRUCTURE -->
   <struct id="NewEmpRec">
      <field name="EmpName">     <typeref type="str20" />  </field>
      <field name="EmpDept">     <typeref type="uword" />  </field>
      <field name="EmpHiredate"> <typeref type="date" />   </field>
      <field name="EmpSalary">   <typeref type="number" /> </field>
   </struct>
</message>
```

2. Save the D3L definition in Example C–20 to a file called `newemp.xml`.
3. Include a copy of this file on both the host computer where Oracle Application Server Integration InterConnect is installed and on the Windows computer where iStudio is installed.

> **Note:** `newemp.xml` is also copied to the FTP adapter application directory in "Task 4: Copy the newemp.xml D3L File to the fileapp_sub Adapter Directory" on page C-41.

The following example shows a native format message that can be translated by the `newemp.xml` D3L file (The `?` character means nonprintable):

```
Pos       Bytes (in hexadecimal)                    Characters
0000000   4a6f 686e 2044 6f65 2020 2020 2020 2020   John Doe
0000020   2020 2020 4000 7c31 322f 3134 2f32 3030       @?|12/14/200
0000040   317c 2435 3432 3230 2e37 3524             1|$54220.75$
```

Where...	Is...
EmpName	John Doe
EmpDept	64 (hex: 0x40)
EmpHiredate	12/14/2001
EmpSalary	54220.75

In "Configuring the aqapp_pub and fileapp_sub Applications in iStudio" on page C-33, you complete all the steps necessary in iStudio, including defining the common view, defining the application creation, and so on.

C.7.3 Configuring the aqapp_pub and fileapp_sub Applications in iStudio

This section describes the tasks to complete in iStudio.

This section contains these topics:

C.7.3.1 Task 1: Create a New Workspace and New Project

1. Start iStudio from the Start menu.

   When iStudio starts, the last used workspace is automatically loaded. For this use case, define a new workspace and new Project.
2. Select **File**, and then **New Workspace**.
3. Enter `d3l_tests` for the Workspace Name, and click OK.
4. Select **File**, and then **New Project**.
5. Enter `d3l_test_ftp` for the Project Name, and click OK.
6. Enter the following values in the Hub Information dialog box:

For...	Enter...
Hub database username	`oaihub`
Hub database password	`oaihub` (the default)
Hub database URL	*`hubDB-host`*`:`*`hubDB-port`*`:`*`hubDB-SID`* For example: `dlsun10:1521:V904`

C.7.3.2 Task 2: Create the Employee Business Object

1. Select **File**, **New**, and then **Business Object**.
2. Enter `Employee` for the Business Object name and click OK.

> **Note:** The `Employee` Business Object name matches with the value for the `object` attribute of the `<message>` element in the D3L file created in "Task 2: Create a D3L File for the FTP Adapter" on page C-32.

C.7.3.3 Task 3: Create the newEmployee Event

Define the `newEmployee` event as described in "D3L Use Case Overview" on page C-31. Define the (common view) attributes of the event by importing the `newemp.xml` D3L file defined in "Task 2: Create a D3L File for the FTP Adapter" on page C-32. This D3L file defines the same data types as used by the common view. (Refer to Table C–3 on page C-31.)

1. Select **File**, **New**, and then **Event**.
2. Select `Employee` in the Business Object list.
3. Enter `newEmployee` in the Event Name field.
4. Click Import.
5. Select `D3L` from the list that appears.
6. Locate and select the `newemp.xml` D3L file created in "Task 2: Create a D3L File for the FTP Adapter" on page C-32. The contents of `newemp.xml` display in the Attributes fields of the Create Event dialog box. If you receive an error while importing, check if the contents of the `newemp.xml` file on your iStudio computer are identical to the text shown in Example C–20 on page C-33.

   Note: The `newEmployee` Event Name matches with the value for the `name` attribute of the `<message>` element in the D3L file created in "Task 2: Create a D3L File for the FTP Adapter" on page C-32.

7. Click **Save**.

   See Also: "Creating Events" on page 4-2

C.7.3.4 Task 4: Create the aqapp_pub Application

Now create the `aqapp_pub` application, which publishes the defined event `Employee.newEmployee`.

1. Click **File**, **New**, and then **Application**.
2. Enter `aqapp_pub` for the Application Name, and click **OK**.

C.7.3.5 Task 5: Enable the aqapp_pub Application to Publish the newEmployee Event

Use the Publish Wizard to publish the `newEmployee` event.

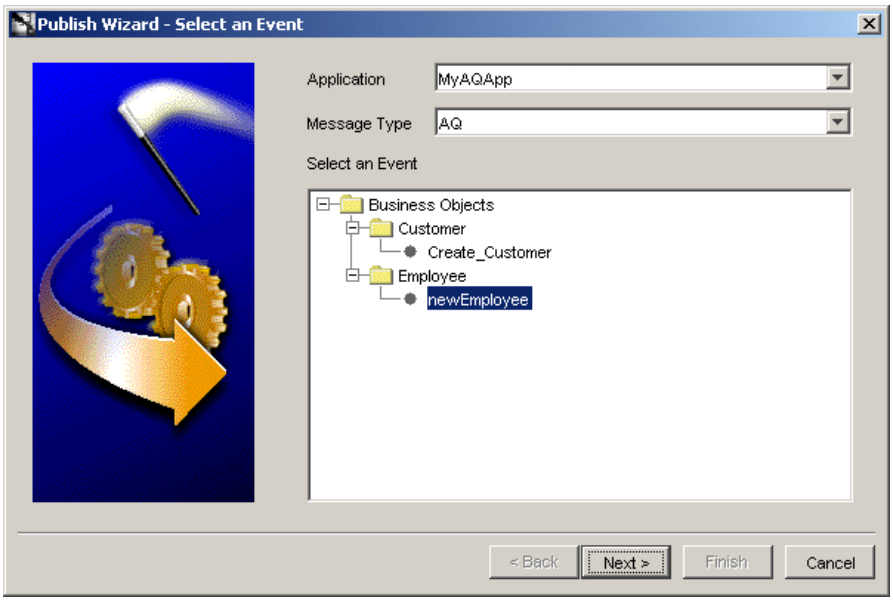
This section contains the following topics:

- Select the Event to Publish
- Define the Application View
- Define the Application View to Common View Mapping

C.7.3.5.1 Select the Event to Publish

Select the event to publish with the Publish Wizard.

1. Select **Event**, and then **Publish Event.** The Publish Wizard - Select an Event dialog box is displayed.
2. Select **aqapp_pub** from the Application list.
3. Select **AQ** from the Message Type list. This choice means that the `aqapp_pub` application is based on the Advanced Queuing adapter.
4. Click the **newEmployee** event in the Select an Event list, which is a child of the `Employee` business object.



5. Click Next. The Publish Wizard - Define Application View dialog box is displayed.

C.7.3.5.2 Define the Application View Define the application view for the Advanced Queuing adapter-based application `aqapp_pub` in this dialog box. This view was defined in "Task 1: Create a DTD File for the Advanced Queuing Adapter" on page C-32 as an XML DTD, which is a requirement of the Advanced Queuing adapter. Import this DTD to define the application view.

1. Click **Import**.
2. Select **XML** from the list that appears.
3. Locate and select the `newemp.dtd` file, which you created in "Task 1: Create a DTD File for the Advanced Queuing Adapter" on page C-32.
4. Select **NewEmpRec** in the Choose Root Element dialog box.
5. Click **OK**. The Publish Wizard - Define Application View dialog box is displayed.
6. Click **Next**. The Publish Wizard - Define Mapping dialog box is displayed.

C.7.3.5.3 Define the Application View to Common View Mapping Define the application view to common view mapping on this dialog box.

1. Click **New**. The Mapping Parameters dialog box is displayed.
2. Expand **newEmployee** and **NewEmpRec** in the aqapp_pub View pane.
3. Expand **newEmployee** and **NewEmpRec** in the Common View pane.
4. Click the `EmpName` attribute in both panes.
5. Select `CopyFields` in the Transformations list.
6. Click OK.
7. Repeat Steps 4 through 6 for the remaining attributes `EmpDept`, `EmpHiredate`, and `EmpSalary`. When complete, the Publish Wizard - Define Mapping dialog box is displayed.

8. Click **Finish**. The Publication for application `aqapp_pub` is complete. The navigation pane on the left hand side of iStudio shows the structure for the `aqapp_pub` application.

C.7.3.6 Task 6: Define the Application Queue for the aqapp_pub Application

As the `aqapp_pub` application publishes the `newEmployee` event and is based on the Advanced Queuing adapter, you must define the (Oracle Advanced Queuing) queue from which the Advanced Queuing adapter reads the event. When an XML message, which complies with the DTD defined in "Task 1: Create a DTD File for the Advanced Queuing Adapter" on page C-32, is enqueued onto the outbound queue, the Advanced Queuing adapter:

- Picks the message up
- Translates the message to an application view event
- Passes the message to the adapter agent for further transformation to the common view representation

The following steps describe how to choose the queue name. The queue does not have to exist physically at this point, as you create it in a later step. (Refer to section "Task 2: Create the Application Queue AQAPP_NEWEMP" on page C-39.)

1. Click the **Deploy** navigation tab on top of the iStudio navigation list.
2. Expand the **Applications** node.
3. Expand the **aqapp_pub** node.
4. Expand the **Routing** node.
5. Right-click the **Application Queues** node.
6. Select the **Edit** option from the list that appears. The Edit Application Queues dialog box is displayed.
7. Click in the empty field under the Queue Name column header, and enter the queue name, for example, `AQAPP_NEWEMP`:
8. Click **OK**.

C.7.3.7 Task 7: Create the fileapp_sub Application

Create the `fileapp_sub` application to subscribe to the defined event `Employee.newEmployee` (which is published by `aqapp_pub`).

1. Click **File**, **New**, and then **Application**.
2. Enter `fileapp_sub` for the Application Name and click **OK**.

C.7.3.8 Task 8: Enable the fileapp_sub Application to Subscribe to the newEmployee Event

Use the Subscribe Wizard to subscribe to the `newEmployee` event.

This section contains these topics:

- Select the Event to which to Subscribe
- Define the Application View
- Define the Application View to Common View Mapping

C.7.3.8.1 Select the Event to which to Subscribe Select the event to which to subscribe with the Subscribe Wizard.

1. Select **Event**, and then **Subscribe Event**. The Subscribe Wizard - Select an Event dialog box is displayed.
2. Select **fileapp_sub** from the Application list.
3. Select **D3L** from the Message Type list.
4. Click **newEmployee** (under `Employee`) in the Select an Event list.
5. Click **Next**. The Subscribe Wizard - Define Application View dialog box is displayed.

C.7.3.8.2 Define the Application View Define the application view for the FTP adapter-based application `fileapp_sub` in this dialog box. This view was defined in "Task 2: Create a D3L File for the FTP Adapter" on page C-32 as a D3L file. This is a requirement of any OracleAS Integration InterConnect Adapter operating in D3L mode. Import this D3L file to define the application view.

1. Enter `Employee` as the business object name in the Object Name input field.
2. Click **Import**.
3. Select **D3L** from the list that appears.
4. Locate and select the `newemp.xml` file, which you saved in "Task 2: Create a D3L File for the FTP Adapter" on page C-32.

   The contents of `newemp.xml` display in the Attributes fields:
5. Click **Next**. The Subscribe Wizard - Define Application View dialog box is displayed.

C.7.3.8.3 Define the Application View to Common View Mapping Define the application view to common view mapping in this dialog box.

1. Click **Define Mappings**. The Mapping Parameters dialog box is displayed.
2. Expand **newEmployee** node in the Common View pane.
3. Expand **newEmployee** node in the `fileapp_sub` View pane.
4. Select the **NewEmpRec** in both panes.
5. Select **ObjectCopy** in the Transformations list and click **Add**.

> **Note:** You can select `ObjectCopy` here because the common view and application view are based on the same D3L file.

6. Click **OK**.
7. Click **Finish**. This completes the necessary setup steps in iStudio.

C.7.4 Installing the Advanced Queuing and FTP Adapters

Now that iStudio setup is complete, you must install one instance of each of the two adapter types. This section contains these topics:

- Task 1: Install the Advanced Queuing Adapter for Application aqapp_pub
- Task 2: Create the Application Queue AQAPP_NEWEMP

- Task 3: Install the FTP Adapter for Application fileapp_sub
- Task 4: Copy the newemp.xml D3L File to the fileapp_sub Adapter Directory
- Task 5: Set the D3L file and Payload Type in the adapter.ini Adapter Initialization File

C.7.4.1 Task 1: Install the Advanced Queuing Adapter for Application aqapp_pub

1. Refer to "Advanced Queuing Adapter Installation" in Chapter 2 of the *Oracle Application Server Integration InterConnect Adapter for AQ Installation and User's Guide* for installation instructions.

   During installation, enter the following specific values when prompted:

   a. Enter `aqapp_pub` in the Application Name field of the Oracle Application Server Integration InterConnect AQ Adapter Configuration dialog box.

   b. Enter the database connection information to connect to the database instance on the Application Spoke Database page. The `AQAPP_NEWEMP` application queue defined in "Task 6: Define the Application Queue for the aqapp_pub Application" on page C-37 is created here.

   c. Enter the database username and password of the account and schema on the Spoke Application Database AQ Username dialog box, which owns the Application Queue (`AQAPP_NEWEMP`). Select the schema name `aqapp` and the password `aqapp`. Leave the Consumer Name field blank, as you are creating the `AQAPP_NEWEMP` queue as a single consumer queue.

2. Complete adapter installation by providing appropriate responses when prompted.

   When installation is complete, the new adapter instance is located in the following directory:

Platform	Directory
Windows	`%ORACLE_HOME%\integration\interconnect\adapters\aqapp_pub`
UNIX	`$ORACLE_HOME/integration/interconnect/adapters/aqapp_pub`

C.7.4.2 Task 2: Create the Application Queue AQAPP_NEWEMP

To create the Advanced Queuing `AQAPP_NEWEMP` application queue, you must first create the queue table, create the queue, and start the queue.

1. Ensure that the database user issuing the commands in this section has been granted the following roles:

```
RESOURCE,

CONNECT,

AQ_ADMINISTRATOR_ROLE
```

2. Use SQL*Plus to log in to the database account specified in Step 1 of "Task 1: Install the Advanced Queuing Adapter for Application aqapp_pub" on page C-39.

3. Create the queue table using the same name as the application queue:

```
SQL> EXECUTE dbms_aqadm.create_queue_table('AQAPP_NEWEMP', 'RAW');
```

4. Create the queue:

```
SQL> EXECUTE dbms_aqadm.create_queue('AQAPP_NEWEMP', 'AQAPP_NEWEMP');
```

5. Start the queue:

```
SQL> EXECUTE dbms_aqadm.start_queue('AQAPP_NEWEMP');
```

C.7.4.3 Task 3: Install the FTP Adapter for Application fileapp_sub

1. Refer to "FTP Adapter Installation" in Chapter 2 of the *Oracle Application Server Integration InterConnect Adapter for FTP Installation and User's Guide* for installation instructions.

   During installation, enter the following specific values when prompted:

   a. Enter `fileapp_sub` in the Application Name field of the Oracle Application Server Integration InterConnect FTP Adapter Configuration dialog box.

   b. Enter the following value in the URL field of the Oracle Application Server Integration InterConnect FTP Adapter Configuration Configure receiving endpoint information dialog box:

   ```
   ftp://localhost/tmp/fileapp_sub/read
   ```

   c. Enter the following value in the URL field of the Oracle Application Server Integration InterConnect FTP Adapter Configuration Configure sending endpoint information dialog box:

   ```
   ftp://localhost/tmp/fileapp_sub/write
   ```

   This action places every `newEmployee` message received by the `fileapp_sub` application (by way of its configured subscription created in "Task 8: Enable the fileapp_sub Application to Subscribe to the newEmployee Event" on page C-37) in the `/tmp/fileapp_sub/write` directory of the computer where the FTP adapter is installed. Ensure that you create these directories with global read and write permissions before starting the `fileapp_sub` application (based on the FTP adapter), for example:

   ```
   $ umask 0
   $ mkdir -p /tmp/fileapp_sub/read
   $ mkdir -p /tmp/fileapp_sub/write
   ```

2. Complete adapter installation by providing appropriate responses when prompted.

   When installation is complete, the new adapter instance is located in the following directory:

Platform	Directory
Windows	`%ORACLE_HOME%\integration\interconnect\adapters\fileapp_sub`
UNIX	`$ORACLE_HOME/integration/interconnect/adapters/fileapp_sub`

C.7.4.4 Task 4: Copy the newemp.xml D3L File to the fileapp_sub Adapter Directory

1. Copy the `newemp.xml` D3L file defined in "Task 2: Create a D3L File for the FTP Adapter" on page C-32 to the platform-specific directory mentioned in the preceding Step 2 on page C-40.

C.7.4.5 Task 5: Set the D3L file and Payload Type in the adapter.ini Adapter Initialization File

Set the `ota.d3ls` and `ota.type` parameters in the `adapter.ini` adapter initialization file for the FTP adapter. The `adapter.ini` file is located in the platform-specific directory mentioned in the preceding Step 2 on page C-40.

1. Use a text editor to set the `ota.d3ls` parameter to `newemp.xml` in `adapter.ini`:

   ```
   ota.d3ls=newemp.xml
   ```

   If the `ota.d3ls` parameter line already exists in `adapter.ini`, replace it with the preceding version.

2. Use a text editor to set the `ota.type` parameter to `D3L` in `adapter.ini`:

   ```
   ota.type=D3L
   ```

C.7.5 Running the D3L Use Case

Now that both the Advanced Queuing adapter instance `aqapp_pub` and the FTP adapter instance `fileapp_sub` have been installed, use both to run the D3L use case.

This section contains these topics:

- Task 1: Start the Adapters
- Task 2: Create PL/SQL Code to Trigger the Native newEmployee Event
- Task 3: Trigger the newEmployee Event
- Task 4: Verify Receipt of newEmployee Event

C.7.5.1 Task 1: Start the Adapters

The first task is to start the adapters.

C.7.5.1.1 To Start the Adapters on UNIX: Follow these steps to start the adapters on UNIX:

To start the `aqapp_pub` (Advanced Queuing) adapter:

1. Change directories to where the `aqapp_pub` adapter is installed:

   ```
   $ cd $ORACLE_HOME/integration/interconnect/adapters/aqapp_pub
   ```

2. Start the adapter as a background process:

   ```
   $ start &
   ```

To start the `fileapp_sub` (FTP) adapter:

1. Change directories to where the `fileapp_sub` adapter is installed:

   ```
   $ cd $ORACLE_HOME/integration/interconnect/adapters/fileapp_sub
   ```

2. Start the adapter as a background process:

   ```
   $ start &
   ```

C.7.5.1.2 To Start the Adapters on Windows: Follow these steps to start the adapters on Windows:

To start the `aqapp_pub` (Advanced Queuing) adapter:

1. Change directories to where the `aqapp_pub` adapter is installed:

```
cd %ORACLE_HOME%\integration\interconnect\adapters\aqapp_pub
```

2. Start the adapter:

```
start
```

To start the `fileapp_sub` (FTP) adapter:

1. Change directories to where the `fileapp_sub` adapter is installed:

```
cd %ORACLE_HOME%\integration\interconnect\adapters\fileapp_sub
```

2. Start the adapter:

```
start
```

> **Note:** You can also start adapters from the Windows Control Panel. Refer to the OracleAS Integration InterConnect Adapter documentation for instructions.

C.7.5.2 Task 2: Create PL/SQL Code to Trigger the Native newEmployee Event

The next task generates the native event (that is, triggers the `newEmployee` event). As configured in iStudio, the `aqapp_pub` application publishes the `newEmployee` event. It does so when it sees a new (XML) message on the `AQAPP_NEWEMP` queue that conforms to the DTD defined in "Task 1: Create a DTD File for the Advanced Queuing Adapter" on page C-32.

To generate the native event, you must enqueue a message on the application queue (`AQAPP_NEWEMP`) for the application `aqapp_pub`. You do this through an anonymous PL/SQL block.

1. Change directories to where the `aqapp_pub` application (of the Advanced Queuing adapter) is installed, for example:

On...	Change To...
UNIX	$ cd $ORACLE_HOME/integration/interconnect/adapters/aqapp_pub
Windows	cd %ORACLE_HOME%\integration\interconnect\adapters\aqapp_pub

2. Create a file (named `newemp.sql` in this example) with the contents shown in Example C–21:

Example C–21 File newemp.sql

```
DECLARE
    enqueue_options     dbms_aq.enqueue_options_t;
    message_properties  dbms_aq.message_properties_t;
    msgid               RAW(16);
```

```
    raw_payload         RAW(32767);
    payload             varchar2(2000);
BEGIN
    payload :=
      '<?xml version="1.0" standalone="no"?>
       <NewEmpRec>
         <EmpName>Scott Tiger</EmpName>
         <EmpDept>257</EmpDept>
         <EmpHiredate>05/01/2001</EmpHiredate>
         <EmpSalary>52308.75</EmpSalary>
       </NewEmpRec>';

      raw_payload := utl_raw.cast_to_raw(payload);
      dbms_aq.enqueue(queue_name         => 'AQAPP_NEWEMP',
                      enqueue_options    => enqueue_options,
                      message_properties => message_properties,
                      payload            => raw_payload,
                      msgid              => msgid);
  commit;
END;
/
```

Note: The `payload` variable is assigned a string value, which contains a valid XML document that conforms to the DTD `newemp.dtd` defined in "Task 1: Create a DTD File for the Advanced Queuing Adapter" on page C-32.

C.7.5.3 Task 3: Trigger the newEmployee Event

Everything is now defined, created, and started. You must now trigger the `newEmployee` event, which was prepared in "Task 2: Create PL/SQL Code to Trigger the Native newEmployee Event" on page C-42.

As mentioned earlier, the event is triggered when you place an XML message on the `AQAPP_NEWEMP` queue, which is what the `newemp.sql` script does.

Run the PL/SQL script to generate the event.

1. Log in to the database account `aqapp` where the `AQAPP_NEWEMP` queue was defined. (Refer to "Task 2: Create the Application Queue AQAPP_NEWEMP" on page C-39.) For example, assuming no connect string is necessary, specify the following command:

   ```
   sqlplus aqapp/aqapp
   ```

2. Run the `newemp.sql` script:

   ```
   SQL> START newemp.sql
   ```

   The following message is displayed:

   ```
   PL/SQL procedure successfully completed.
   ```

3. Exit SQL*Plus:

   ```
   SQL> EXIT
   ```

C.7.5.4 Task 4: Verify Receipt of newEmployee Event

After some time (maybe several minutes depending on the system performance), a file appears in the `/tmp/fileapp_sub/write` directory, which represents the sending endpoint for the FTP adapter. The file is named after the pattern:

app-name-timestamp

1. Verify that the `newEmployee` event has been published and received by the `fileapp_sub` application. On UNIX, for example, perform the following commands:

```
$ cd /tmp/fileapp_sub/write
$ ls -l
total 2

-rw-rw-r--  1 bstern  svrtech    44 Dec 18 15:29 FILEAPP_SUB-1008718194783
```

   The contents of the file can be displayed in different formats:

```
$ od -c FILEAPP_SUB-1008718194783
0000000   S   c   o   t   t       T   i   g   e   r
0000020               001 001   |   0   5   /   0   1   /   2   0   0
0000040   1   |   $   5   2   3   0   8   .   7   5   $
```

   or

```
$ od -x FILEAPP_SUB-1008718194783
0000000 5363 6f74 7420 5469 6765 7220 2020 2020
0000020 2020 2020 0101 7c30 352f 3031 2f32 3030
0000040 317c 2435 3233 3038 2e37 3524
```

2. Verify that this output corresponds to the D3L definition shown in "Task 2: Create a D3L File for the FTP Adapter" on page C-32 and the data enqueued by `newemp.sql`.
3. Repeat Step 2 on page C-43 to trigger and generate another event (file). The second time you trigger the event, the new file in the `/write` directory appears much faster (in approximately 3-4 seconds). This is because the adapter allocates and initializes all connections and data structures after processing the first message.
4. You have completed the use case.

C.7.6 Using Other Adapters in D3L and XML Modes

This section briefly describes how to use adapters other than the FTP adapter, and how to run them in XML mode instead of D3L mode. It contains these topics:

- Using the HTTP, SMTP, or MQ Series Adapters in D3L Mode
- Using XML Mode

C.7.6.1 Using the HTTP, SMTP, or MQ Series Adapters in D3L Mode

Perform the following steps to use the D3L use case with a different OracleAS Integration InterConnect adapter.

1. Enter another application name that indicates which adapter you are using in "Task 7: Create the fileapp_sub Application" on page C-37 (for example, `smtpapp_sub`.)

2. Specify the parameters needed for the particular adapter in Step 1 on page C-40. Refer to the installation documentation for the appropriate OracleAS Integration InterConnect Adapter.
3. In "Task 4: Verify Receipt of newEmployee Event" on page C-44, the verification process depends entirely on the adapter type, or more specifically, the exact sending endpoint defined.
4. Replace the `fileapp_sub` application name where ever it appears with the new application name.

The remaining steps are the same as decribed in the previous section.

C.7.6.2 Using XML Mode

Perform the following steps to use XML as the operational mode of the OracleAS Integration InterConnect adapters.

1. Skip "Task 2: Create a D3L File for the FTP Adapter" on page C-32.
2. Define the following common view event attributes in Step 4 and Step 5 of "Task 3: Create the newEmployee Event" on page C-34:
   a. Manually create a common data type (right-click + New) named `NewEmpRec` that has the following attributes:

Attribute Name	Attribute Type
ENAME	STRING
JOB	STRING
SAL	DOUBLE
COMM	DOUBLE

   b. Import the common data typedata type defined in Step 2 instead of importing a D3L file.
3. Select `XML` instead of `D3L` in Step 3 of "Select the Event to which to Subscribe" on page C-38.
4. Select to import `XML` and choose the file `newemp.dtd` in Step 4 of "Define the Application View" on page C-38.
5. Perform Steps 2 through 4 in "Define the Application View to Common View Mapping" on page C-38 like you did Steps 2 through 7 in "Define the Application View to Common View Mapping" on page C-36.
6. Skip Task 4: Copy the newemp.xml D3L File to the fileapp_sub Adapter Directory on page C-41 and "Task 5: Set the D3L file and Payload Type in the adapter.ini Adapter Initialization File" on page C-41.

> **Note:** Step 2 assumes that you do not have the D3L file. However, as a shortcut, you can still define the common view event attributes as they were performed in Step 6 of "Task 3: Create the newEmployee Event" on page C-34.

C.8 Additional D3L Sample Files and DTD

This section contains these topics:

C.8.1 Additional D3L Sample Files

This section provides several D3L sample files. These example files describe how to use the D3L language to define the content of native format messages.

C.8.1.1 Sample File with Structure VehicleRegistration

Sample file `msg-1.xml` represents a structure named `VehicleRegistration`. Table C–4 describes the file fields and Example C–22 shows `msg-1.xml` file contents.

Table C–4 msg-1.xml File Fields

Field	Description
`SizeWeight`	A fixed-length array of four signed, one-byte, little-endian integers, each aligned on two-byte boundaries (implying a one-byte padding between elements of the array.)
`ProductCode`	An unsigned, two-byte, big-endian integer aligned on two-byte boundaries.
`VIN`	An unsigned, eight-byte, big-endian integer aligned on two-byte boundaries.
`PreviousOwners`	A length-prefixed array of dates in the `MMDDYYYY` format (the length of the array is a signed, one-byte, little-endian integer with a two-byte alignment.)
`Miles`	An unsigned, two-byte, big-endian integer with a two-byte alignment.
`DateProduced`	A single date in the `MMDDYYYY` format.

Example C–22 Sample File msg-1.xml with Structure EmployeeRegistration

```
<?xml version="1.0" encoding="US-ASCII"?>
<message type="VehicleRegistration" name="Register" object="Vehicle">
    <date format="MMDDYYYY" id="Date_T">
        <padstring id="FixString10_T" length="10" padchar='' padstyle="none" />
    </date>
    <struct id="VehicleRegistration">
        <!-- Width x Length x Height x Weight (inch/lb) -->
        <field name="SizeWeight"><typeref type="ShortArray4_T" /></field>
        <field name="ProductCode"><unsigned2 align="2" endian="big" /></field>
        <field name="VIN"><unsigned8 align="2" endian="big" /></field>
        <field name="PreviousOwners"><typeref type="StringArray_T" /></field>
        <field name="Miles"><unsigned2 align="2" endian="big" /></field>
        <field name="DateProduced"><typeref type="Date_T" /></field>
    </struct>
    <fixarray id="ShortArray4_T" length="4">
        <unsigned2 align="2" endian="little" id="" />
    </fixarray>
    <unsigned1 align="2" endian="little" id="Short_T" />
    <pfxarray id="StringArray_T" length="Short_T">
        <typeref type="FixString10_T" />
    </pfxarray>
```

```
</message>
```

The following native format message examples show a hexadecimal and character representation of the same message, which can be parsed by the `msg-1.xml` D3L file:

- Hexadecimal format:

```
0000000 4500 b200 3400 8a0b 30d9 0000 0000 0072
0000020 55ff 0200 4a6f 6e65 732c 502e 2020 536d
0000040 6974 682c 522e 2020 5208 3131 2532 3225
0000060 3139 3939
```

- Character format:

```
0000000   E  \0 262  \0   4  \0 212 013   0 331  \0  \0  \0  \0  \0   r
0000020   U 377 002  \0   J   o   n   e   s   ,   P   .           S   m
0000040   i   t   h   ,   R   .           R  \b   1   1   %   2   2   %
0000060   1   9   9   9
```

C.8.1.2 Sample File with Structure Hierarchy PersonRecord

Sample file `msg-2.xml` demonstrates a structure hierarchy named `PersonRecord`. Table C–5 describes the file fields and Example C–23 shows `msg-2.xml` file contents.

Table C–5 msg-2.xml File Fields

Field	Description
`Name`	A string delimited by a comma.
`Age`	An unsigned, one-byte integer.
`DOB`	A date in `MMDDYYYY` format, length prefixed by a signed, four-byte integer.
`Phone`	An unsigned, four-byte integer.
`City`	A structure named `CityRecord` that consists of the following fields: ■ `Name` ■ A string delimited by `*` ■ `State` ■ A string delimited by `*` ■ `Country` ■ A string delimited by `*` ■ `Population` ■ An unsigned, four-byte integer
`State`	A structure named `StateRecord` that consists of the following fields: ■ `Name` ■ A string delimited by a space ■ `Capital` ■ A string delimited by a space ■ `Population` ■ An unsigned, four-byte integer

Example C–23 Sample File msg-2.xml with Structure PersonRecord

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">
<message type="PersonRecord">
```

```
    <signed4 id="s4" />
    <struct id="CityRecord">
        <field name="Name"><limstring delimiter="*" /></field>
        <field name="State"><limstring delimiter="*" /></field>
        <field name="Country"><limstring delimiter="," /></field>
        <field name="Population"><unsigned4 /></field>
    </struct>

    <struct id="StateRecord">
        <field name="Name"><limstring delimiter=" " /></field>
        <field name="Capital"><limstring delimiter=" " /></field>
        <field name="Population"><unsigned4 /></field>
    </struct>

    <struct id="PersonRecord">
        <field name="Name"><limstring delimiter="," /></field>
        <field name="Age"><unsigned1 /></field>
        <field name="DOB">
            <date format="MMDDYYYY">
                <pfxstring id="dobstr" length="s4" />
            </date>
        </field>
        <field name="Phone"><unsigned4 /></field>
        <field name="City"><typeref type="CityRecord" /></field>
        <field name="State"><typeref type="StateRecord" /></field>
    </struct>

</message>
```

The following is a combined hexadecimal and character representation of a native message, which can be parsed by `msg-2.xml`:

```
000  2c4a 6f68 6e20 446f 652c 1e00 0000 000a  ,John Doe,_.....
020  3131 2f32 352f 3139 3635 0000 002c a155  11/25/1965...,.U
040  2a50 6f72 746c 616e 642a 2a4f 522a 2c55  *Portland**OR*,U
060  5341 2c00 000f 4240 204f 7265 676f 6e20  SA,...B@_Oregon_
100  2053 616c 656d 2000 003d 0900            _Salem_..=..
```

C.8.1.3 Sample File with Structure ProductRecord

Sample file `msg-3.xml` defines a structure named `ProductRecord`. Table C–6 describes the file fields and Example C–24 shows `msg-3.xml` file contents.

Table C–6 msg-3.xml File Fields

Field	Description
`Manufacturer`	A string delimited by a space.
`Weight`	A single-precision, floating-point number.
`Widgets`	A length-prefixed array of `WidgetRecord` structures. A `WidgetRecord` consists of: ▪ `Name` ▪ A string delimited by a space ▪ `Color` ▪ A string delimited by a space ▪ `Weight` ▪ A single-precision, floating point number

Example C–24 Sample File msg-3.xml with Structure ProductRecord

```
<?xml version="1.0" encoding="US-ASCII"?>

<!DOCTYPE message SYSTEM "d3l.dtd">

<message type="ProductRecord">
    <unsigned1 id="u1" />
    <unsigned2 id="u2" />
    <number id="pfxnum">
        <padstring length="8" padchar="" padstyle="none" />
    </number>
    <pfxarray id="Unsigned1Tab" length="u1">
        <unsigned1 />
    </pfxarray>
    <pfxarray id="Signed4Tab" length="pfxnum">
        <unsigned4 />
    </pfxarray>
    <pfxarray id="StrTab" length="u1">
        <limstring delimiter=" " />
    </pfxarray>
    <struct id="WidgetRecord">
        <field name="Name"><limstring delimiter=" " /></field>
        <field name="Color"><limstring delimiter=" " /></field>
        <field name="Weight"><float /></field>
    </struct>
    <pfxarray id="WidgetTab" length="u2">
        <typeref type="WidgetRecord" />
    </pfxarray>
    <struct id="ProductRecord">
        <field name="Manufacturer"><limstring delimiter=" " /></field>
        <field name="Weight"><float /></field>
        <field name="Widgets"><typeref type="WidgetTab" /></field>
    </struct>

</message>
```

C.8.2 D3L DTD

Example C–25 shows the DTD to which D3L (XML) files must conform.

Example C–25 D3L DTD

```
<!ENTITY % Name     "CDATA"     >
     <!ENTITY % Number   "NMTOKEN"   >
     <!ENTITY % Comment  "CDATA"     >
     <!ENTITY % DelimiterChar
         "CDATA"
     >
     <!ENTITY % QuotationCharAttribute
         "
         quote  %DelimiterChar; #IMPLIED
         "
     >
     <!-- ======================================================== -->
     <!ENTITY % GenericAttributes
         "
         name    %Name;          #IMPLIED
         comment %Comment;       #IMPLIED
         id      ID              #IMPLIED
```

```
    "
>
<!ENTITY % FieldAttributes
    "
    name    %Name;          #REQUIRED
    comment %Comment;       #IMPLIED
    id      ID              #IMPLIED
    "
>
<!ENTITY % NonTypeAttributes
    "
    name    %Name;          #IMPLIED
    comment %Comment;       #IMPLIED
    "
>
<!-- ======================================================== -->
<!ENTITY % StructAttributes
    "
    %GenericAttributes;
    %QuotationCharAttribute;
    "
>
<!-- ======================================================== -->
<!ENTITY % Align
    "%Number;"
>
<!ENTITY % IntegerSize
    "( 1 | 2 | 4 | 8 )"
>
<!ENTITY % Endian
    "( big | little )"
>
<!ENTITY % IntegerAttributes
    "
    %GenericAttributes;
    endian  %Endian;        'big'
    "
>
<!ENTITY % IntegerTypes
    " signed1 | unsigned1
    | signed2 | unsigned2
    | signed4 | unsigned4
    | signed8 | unsigned8
    "
>
<!ENTITY % FloatAttributes
    "
    %GenericAttributes;
    "
>
<!ENTITY % FloatTypes
    " float | double
    "
>
<!-- ======================================================== -->
<!ENTITY % PadStyle
    "( head | tail | none )"
>
<!ENTITY % PadChar
    "CDATA"
```

```
    >
    <!ENTITY % StringAttributes
        "
        %GenericAttributes;
        "
    >
    <!ENTITY % SimpleStringAttributes
        "
        %StringAttributes;
        "
    >
    <!ENTITY % TerminatedStringAttributes
        "
        %StringAttributes;
        endchar     %DelimiterChar; #REQUIRED
        "
    >
    <!ENTITY % QuotedTerminatedStringAttributes
        "
        %StringAttributes;
        %QuotationCharAttribute;
        endchar     %DelimiterChar; #REQUIRED
        "
    >
    <!ENTITY % PaddedStringAttributes
        "
        %StringAttributes;
        length      %Number;        #REQUIRED
        padchar     %PadChar;       #REQUIRED
        padstyle    %PadStyle;      #REQUIRED
        "
    >
    <!ENTITY % PrefixedStringAttributes
        "
        %StringAttributes;
        length      IDREF           #REQUIRED
        "
    >
    <!ENTITY % DelimitedStringAttributes
        "
        %StringAttributes;
        delimiter   %DelimiterChar; #REQUIRED
        "
    >
    <!ENTITY % StringTypes
        "padstring | pfxstring | limstring | qtdtermstring | termstring |
simplestring "
    >
    <!-- ======================================================== -->
    <!ENTITY % DateFormat
        "( DDMMYY | DDMMYYYY | MMDDYY | MMDDYYYY )"
    >
    <!ENTITY % DateAttributes
        "
        %GenericAttributes;
        format  %DateFormat;    #REQUIRED
        "
    >
    <!-- ======================================================== -->
    <!ENTITY % NumberAttributes
```

```
    "
    %GenericAttributes;
    "
>
<!-- ======================================================== -->
<!ENTITY % ArrayAttributes
    "
    %GenericAttributes;
    "
>
<!ENTITY % FixedArrayAttributes
    "
    %ArrayAttributes;
    length      %Number;        #REQUIRED
    "
>
<!ENTITY % PrefixedArrayAttributes
    "
    %ArrayAttributes;
    length      IDREF           #REQUIRED
    "
>
<!ENTITY % DelimitedArrayAttributes
    "
    %ArrayAttributes;
    contchar    %DelimiterChar; #REQUIRED
    endchar     %DelimiterChar; #REQUIRED
    "
>
<!ENTITY % ImplicitArrayAttributes
    "
    %ArrayAttributes;
    "
>
<!-- ======================================================== -->
<!ENTITY % ScalarElements
    " signed1 | unsigned1
    | signed2 | unsigned2
    | signed4 | unsigned4
    | signed8 | unsigned8
    | float   | double
    | date    | number
    | padstring
    | pfxstring
    | limstring
    | termstring
    | qtdtermstring
    | simplestring
    "
>
<!ENTITY % TypeElements
    "%ScalarElements;
    | struct
    | fixarray
    | pfxarray
    | limarray
    | imparray
    "
>
<!-- ======================================================== -->
```

```
<!ENTITY % FieldElements
    "%TypeElements;"
>
<!ENTITY % MessageElements
    "%TypeElements;"
>
<!ENTITY % StructElements
    "field | pad"
>
<!ENTITY % ArrayElements
    "%ScalarElements; | struct"
>
<!ENTITY % ImplicitArrayElements
    "%ArrayElements; | limarray"
>
<!-- ======================================================== -->
<!ELEMENT message ( %MessageElements; )* >
    <!ATTLIST message
        name        %Name;          #REQUIRED
        object      CDATA           #REQUIRED
        type        IDREF           #REQUIRED
        comment     %Comment;       #IMPLIED
        id          ID              #IMPLIED
        header      CDATA           #IMPLIED
        value       CDATA           #IMPLIED
        magic       CDATA           #IMPLIED
        startsat    %Number;        #IMPLIED
        reply       (Y|N)           "N"
        %QuotationCharAttribute;
    >
<!-- ======================================================== -->
<!ELEMENT struct ( %StructElements; )* >
    <!ATTLIST struct
        %StructAttributes;
    >
<!-- ======================================================== -->
<!ELEMENT field ( typeref | %FieldElements; ) >
    <!ATTLIST field
        %FieldAttributes;
    >
<!-- ======================================================== -->
<!ELEMENT signed1 EMPTY >
    <!ATTLIST signed1
        %IntegerAttributes;
        size    %IntegerSize;   #FIXED  "1"
        align   %Align;         "1"
    >
<!ELEMENT unsigned1 EMPTY >
    <!ATTLIST unsigned1
        %IntegerAttributes;
        size    %IntegerSize;   #FIXED  "1"
        align   %Align;         "1"
    >
<!ELEMENT signed2 EMPTY >
    <!ATTLIST signed2
        %IntegerAttributes;
        size    %IntegerSize;   #FIXED  "2"
        align   %Align;         "2"
    >
<!ELEMENT unsigned2 EMPTY >
```

```
    <!ATTLIST unsigned2
        %IntegerAttributes;
        size    %IntegerSize;   #FIXED  "2"
        align   %Align;         "2"
    >
<!ELEMENT signed4 EMPTY >
    <!ATTLIST signed4
        %IntegerAttributes;
        size    %IntegerSize;   #FIXED  "4"
        align   %Align;         "4"
    >
<!ELEMENT unsigned4 EMPTY >
    <!ATTLIST unsigned4
        %IntegerAttributes;
        size    %IntegerSize;   #FIXED  "4"
        align   %Align;         "4"
    >
<!ELEMENT signed8 EMPTY >
    <!ATTLIST signed8
        %IntegerAttributes;
        size    %IntegerSize;   #FIXED  "8"
        align   %Align;         "8"
    >
<!ELEMENT unsigned8 EMPTY >
    <!ATTLIST unsigned8
        %IntegerAttributes;
        size    %IntegerSize;   #FIXED  "8"
        align   %Align;         "8"
    >
<!-- ======================================================== -->
<!ELEMENT float EMPTY >
    <!ATTLIST float
        %FloatAttributes;
        align   %Align;     "4"
    >
<!ELEMENT double EMPTY >
    <!ATTLIST double
        %FloatAttributes;
        align   %Align;     "8"
    >
<!-- ======================================================== -->
<!ELEMENT simplestring EMPTY >
    <!ATTLIST simplestring
        %SimpleStringAttributes;
    >
<!ELEMENT qtdtermstring EMPTY >
    <!ATTLIST qtdtermstring
        %QuotedTerminatedStringAttributes;
    >
<!ELEMENT termstring EMPTY >
    <!ATTLIST termstring
        %TerminatedStringAttributes;
    >
<!ELEMENT padstring EMPTY >
    <!ATTLIST padstring
        %PaddedStringAttributes;
    >
<!ELEMENT pfxstring EMPTY >
    <!ATTLIST pfxstring
        %PrefixedStringAttributes;
```

```
    >
<!ELEMENT limstring EMPTY >
    <!ATTLIST limstring
        %DelimitedStringAttributes;
    >
<!-- ======================================================== -->
<!ELEMENT fixarray ( typeref | %ArrayElements; ) >
    <!ATTLIST fixarray
        %FixedArrayAttributes;
    >
<!ELEMENT pfxarray ( typeref | %ArrayElements; ) >
    <!ATTLIST pfxarray
        %PrefixedArrayAttributes;
    >
<!ELEMENT limarray ( typeref | %ArrayElements; ) >
    <!ATTLIST limarray
        %DelimitedArrayAttributes;
    >
<!ELEMENT imparray ( typeref | %ImplicitArrayElements; ) >
    <!ATTLIST imparray
        %ImplicitArrayAttributes;
    >
<!-- ======================================================== -->
<!ELEMENT date ( typeref | %StringTypes; ) >
    <!ATTLIST date
        %DateAttributes;
    >
<!-- ======================================================== -->
<!ELEMENT number ( typeref | %StringTypes; ) >
    <!ATTLIST number
        %NumberAttributes;
    >
<!-- ======================================================== -->
<!ELEMENT typeref EMPTY >
    <!ATTLIST typeref
        %NonTypeAttributes;
        type    IDREF           #REQUIRED
    >
<!-- ======================================================== -->
<!ELEMENT pad EMPTY >
    <!ATTLIST pad
        %NonTypeAttributes;
        length  %Number;        #REQUIRED
    >
<!-- ======================================================== -->
```

D

Transformations

This appendix describes the OracleAS Integration InterConnect transformations.

D.1 OracleAS Integration InterConnect Transformations

This section contains the following transformations:.

D.1.1 Copy Fields

Copy the source fields into the destination fields.

Parameters:

None

D.1.2 Copy Object

Copy the source object into the destination object.

Parameters:

None

D.1.3 Concat Fields

Concatenate the source fields, and copy into the destination fields.

Parameter	Description
`prefix`	An optional prefix to the concatenated string.
`separator`	The separator, a string of characters, that separate source fields in the concatenated string.
`suffix`	An optional suffix to the concatenated string.

D.1.4 Expand Fields

Expand the source field into the destination fields.

Parameter	Description
delimiter	The delimiter or string of characters around which the source field should be separated.

D.1.5 Set Constant

Copy a constant into the destination fields.

Parameter	Description
constant	The constant to be copied.

D.1.6 True Conditional Lookup XRef

Find the source field in a cross-reference table. If the condition is satisfied, then copy it into the destination field.

Parameter	Description
condition	The condition for this parameter.
table	The cross-reference table.
pass through	If there is no corresponding cross-reference, and the parameter is true, then the destination field is set to the source field. If this parameter is false, then the destination field is set to null.

D.1.7 True Conditional Lookup DVM

Find the source field in a domain value map table. If the condition is satisfied, then copy it into the destination field.

Parameter	Description
condition	The condition for this parameter.
table	The domain value map table.
pass through	If there is no corresponding domain value map and this parameter is set to true, then the destination field is set to the source field. If this parameter is set to false, then the destinations field is set to null.

D.1.8 Conditional Copy

Copy the source fields into the destination fields if the expression is satisfied.

Parameter	Description
expression	The expression.
only copy on true	If this parameter is set to true and the expression evaluates to false, then nothing is copied. If this parameter is set to false and the expression evaluates to false, then the second input object is copied.

D.1.9 True Conditional Copy

Copy the source fields into the destination fields if the condition is satisfied.

Parameter	Description
`condition`	The condition for this parameter.

D.1.10 True Conditional Concat

Concatenate the source fields into the destination field if the condition is satisfied.

Parameter	Description
`condition`	The condition for this parameter.
`prefix`	An optional prefix to the concatenated string.
`separator`	The separator, a string of characters, that separate source fields in the concatenated string.
`suffix`	An optional suffix to the concatenated string.

D.1.11 True Conditional To Number

Convert the sign, value, and precision source fields into a number, and copy it into the destination field if the condition is satisfied.

Parameter	Description
`condition`	The condition for this parameter.
`int length`	The number of digits before the decimal point excluding the sign.
`dec length`	The number of digits after the decimal point.
`character`	The padding character.
`DVM`	An optional domain value map to lookup decimal point character.

D.1.12 False Conditional Copy

Copy the source fields into the destination fields if condition is not satisfied.

Parameter	Description
`condition`	The condition for this parameter.
`condition`	The condition for this parameter.
`condition`	The condition for this parameter.

D.1.13 False Conditional Concat

Concatenate the source fields into the destination field if the condition is not satisfied.

Parameter	Description
condition	The condition for this parameter.
prefix	An optional prefix to the concatenated string.
separator	The separator, a string of characters, that separate source fields in the concatenated string.

D.1.14 False Conditional To Number

Convert the sign, value, and precision source fields into a number and copy it into the destination field if condition is not satisfied.

Parameter	Description
condition	The condition for this parameter.
int length	The number of digits before the decimal point excluding the sign.
dec length	The number of digits after the decimal point.

D.1.15 To Number

Convert the sign, value, and precision source fields into a number, and copy it into the destination field.

Parameter	Description
int length	The number of digits before the decimal point excluding the sign.
dec length	The number of digits after the decimal point.
character	The padding character.

D.1.16 Substring

Copy a substring of the source field into the destination field.

Parameter	Description
begin index	The index at which the substring begins.
length	An optional length of the substring.
begin index	The index at which the substring begins.

D.1.17 Char Replace

Replace characters in the source field, and copy it into the destination field.

Parameter	Description
targets	The string of characters to replace.
replacements	The string of replacement characters.

Parameter	Description
targets	The string of characters to replace.

D.1.18 String Replace

Replace each occurrence of a string in the source field, and copy the replacement into the destination field.

Parameter	Description
targets	The string of characters to replace.
replacements	The string of replacement characters.
targets	The string of characters to replace.

D.1.19 LTrim

Delete source field characters starting from the left until a character from the set is found, and copy the remaining string into the destination field.

Parameter	Description
characters	The string of characters to seek that stop the deletion.
characters	The string of characters to seek that stop the deletion.
characters	The string of characters to seek that stop the deletion.

D.1.20 RTrim

Delete source field characters starting from the right until a character from the set is found, and copy the remaining string into the destination field.

Parameter	Description
characters	The string of characters to seek that stop the deletion.
characters	The string of characters to seek that stop the deletion.
characters	The string of characters to seek that stop the deletion.

D.1.21 LPad

Pad source field starting from the left for a given length, and copy it into the destination field.

Parameter	Description
length	The padding length.
character	An optional character to pad with, default is <space>.
length	The padding length.

D.1.22 RPad

Pad source field starting from the right for a given length, and copy it into the destination field.

Parameter	Description
`length`	The padding length.
`character`	An optional character to pad with, default is `<space>`.
`length`	The padding length.

D.1.23 Lookup XRef

Lookup the source field in a cross-reference table, and copy it into the destination field.

Parameter	Description
`table`	The cross-reference table.
`pass through`	If there is no corresponding cross-reference and this parameter is set to `true`, then the destination field is set to the source field. If this parameter is set to `false`, then the destination field is set to `null`.
`table`	The cross-reference table.

D.1.24 Delete XRef

Delete the source field from a cross-reference table.

Parameter	Description
`table`	The cross-reference table.
`table`	The cross-reference table.
`table`	The cross-reference table.

D.1.25 Lookup DVM

Look up the source field in a domain value map table, and copy it into the destination field.

Parameter	Description
`table`	The domain value map table.
`pass through`	If there is no corresponding domain value map and this parameter is set to `true`, then the destination field is set to the source field. If this parameter is set to `false`, then the destination field is set to `null`.
`table`	The domain value map table.

D.1.26 Truncate

Truncate source field starting from the right for a given length, and copy it into the destination field.

Parameter	Description
length	The length to truncate.
length	The length to truncate.
length	The length to truncate.

D.1.27 Increment

Increment a counter, and copy the incremented value into the destination field.

Parameter	Description
start value	The initial counter value.
counter	The name of the counter. The name should distinguish it from other counters that may be at different values at a given time and may have a different step size.
step size	The increment size.

D.1.28 Lookup

Find equivalent of an item in the Lookup table and copy its value in the destination field.

Parameter	Description
table	The name of the table in which Item will be looked up.
item	The name of the item to be looked up.

D.1.29 AddTransformation

Add user-defined information in the header component of the OAI message.

Parameter	Description
field	The field to be added.
value	The value of the added field.

D.1.30 DatabaseOperation

Apply SQL or PL/SQL operations to the source fields, and copy the result to the destination fields. This transformation can be processed on any database, including the hub or database adapter instance. Connect to the database given by the connectivity parameters and bind the input variables to the corresponding bind variables of the SQL or PL/SQL given by the operation parameters.

The statement is then run. Upon successful execution, the results are copied to the destination field of the transformation. The connection to the database is then closed and the result of the transformation is returned.

Parameter	Description
db user	The database user name.

Parameter	Description
db password	The password of the database user.
db host	The hostname of the database.

Note: For the PL/SQL type of transformation, the following syntax of the PL/SQL statement is assumed:

- IN parameters are specified with a ?I
- OUT parameters are specified with a ?O:<T>
- IN/OUT parameters are specified with a ?IO:<T>

<T> is a single character type specifier denoting the type of the variable. Valid variable values are:

- S: String
- I: Integer
- F: Float
- D: Double
- T: Date
- B: Binary

E

Troubleshooting OracleAS Integration InterConnect

This appendix describes common problems that you might encounter when using OracleAS Integration InterConnect and explains how to solve them. It also gives detailed instructions on how to diagnose OracleAS Integration InterConnect problems. It contains the following topics:

E.1 Problems and Solutions

This section describes common problems and solutions. It contains the following topics:

E.1.1 iStudio Fails to Connect

iStudio is not connecting to the repository. For example, a message pops up saying "Failed to connect to the repository".

Problem 1

Incorrect hub database parameters entered in the Hub Information dialog box.

Solution 1

iStudio uses the user name, password, and connect string entered in the 'Hub Information' dialog box to establish a connection with the respository. Ensure that you have provided the correct hub database parameters, *hostname*:*tns_listener_port*:*db_sid*, for connecting to the Infrastructure Database. For example, `myhost.us.mycompany.com:1521:orcl`.

> **See Also:** Section 2.5.1, "Creating a New Project" for more details on the hub database parameters

Problem 2

Repository is down.

Solution 2

Ensure that the repository is properly started and check the *$ORACLE_HOME*`/intergration/interconnect/repository/reposlog.txt` file for any problems.

Problem 3

Repository is running behind a firewall.

Solution 3

Ensure that you configure the repository to use a specific RMI communication port. Specify a port number in the `repo_admin_port` parameter of the `repository.ini` file located at *$ORACLE_HOME*`/integration/interconnect/repository`. You also need to open the seven ports on the firewall starting from the port number specified in `repo_admin_port` parameter of the `repository.ini` file.

E.1.2 Mappings in iStudio

I have changed the mapping, but the adapter(s) still seem to use the old information.

Problem 1

Metadata is not refreshed.

Solution 1

In the File menu, select Sync Adapters. This pushes the updated metadata to the adapters.

Solution 2

Alternatively, you could perform the following steps to change the behavior after transformation:

1. Stop the adapters.
2. In the `adapter.ini` file, set the value of the parameter `agent_delete_file_cache_at_startup` to True, to delete all cached metadata at startup.

   This parameter specifies whether to delete the cached metadata during startup. If any of the agent caching methods such as metadata caching, DVM table caching, or lookup table caching is enabled, then metadata from the repository is cached

locally on the file system. However, if you change some metadata or DVM table using iStudio and you want the Adapter to use those changes the next time it is started, then you can either delete the cache files or set this parameter to true before restarting.

3. Restart the adapters.

E.1.3 Metadata Not Editable in iStudio

The metadata is not editable in iStudio as the "Edit" option is greyed out.

Problem 1

The metadata in the repository might belong to a different metadata owner than the one specified in the `$ORACLE_HOME/integration/interconnect/repository/repository.ini` file.

Solution 1

Ensure that the "owner" of the data is same in both repositories. Data types have an owner, usually, that is "OAI". But you can change it in the `repository.ini` file. Only data that is owned by the local repository owner may be edited.

The metadata owner information is stored along with the objects at the time of their creation. If the repository owner name does not match the creation time owner name, then you will notice that the edit buttons are greyed out. You will need to edit the `repository.ini` file to change the owner name to the one used during creation and restart the repository. You will also need to reestablish the iStudio connection to the repository.

E.1.4 Subscribing Adapter Does Not Receive Messages From the Hub

The subscribing adapter is not able to pickup any message from the hub queue (`oai_hub_queue`). The messages are queued up in the hub queue but aren't subscribed by the subscribing adapter.

Problem 1

The value for the `application`, `agent_subscriber_name`, and `agent_message_selector` parameter in the *`$ORACLE_HOME`*`/integration/interconnect/adapters/`*`adapter_name`*`/adapter.ini` file doesn't match the application name in iStudio.

Solution 1

Ensure that the value for the `application`, `agent_subscriber_name`, and `agent_message_selector` parameter in the *`$ORACLE_HOME`*`/integration/interconnect/adapters/`*`adapter_name`*`/adapter.ini` file match the application name in iStudio.

Problem 2

Subscriber name or application name may be longer than 20 characters.

Solution 2

Apply patch 2659228.

E.1.5 Messages Are Not Getting Delivered to a Spoke Application

If an adapter successfully received a message from the hub, but encounters a problem while transforming the message or delivering the message to the spoke application, then it will try to reprocess/resend the message in a certain interval until the problem has been resolved and the message has been successfully delivered. In most cases, this unfortunately means that all other pending messages that need to be processed to the spoke application won't get delivered until this one message has cleared out of the system.

Problem

OracleAS Integration InterConnect guarantees in-order-delivery of messages, meaning that if an adapter encounters a problem while processing a message, it will try to re-send the message until the problem has been resolved and the message can be delivered successfully. All other pending messages can't get delivered as this would change the order of delivery.

Solution 1

Refer to the error message and the `oailog.txt` file in *`$ORACLE_HOME`*`/integration/interconnect/adapters/`*`adapter_name`*`/logs/directory`, to resolve the error message.

Solution 2

Use the *`$ORACLE_HOME`*`/integration/interconnect/adapters/`*`adapter_name`*`/ignoreErrors` script to drop the problem message out of the system. The `ignoreErrors` script will reconfigure the adapter, so that it ignores a specific error code. The adapter will then drop all messages that fail with that specific error code rather than trying to resend them, which ensures that other pending messages can be processed.

Note: When you use `ignoremessage` script, the messages are permanently deleted from the system. These messages cannot be recovered.

E.1.6 OracleAS Integration InterConnect Repository Does Not Start

The OracleAS Integration InterConnect repository does not start properly and there are java excpetions in the `$ORACLE_HOME/integration/interconnect/repository/reposlog.txt` file. For example, "Could not register with the hub database. Please check your hub database parameters." Error message: "Retrying in 10 sec.java.sql.SQLException: Io exception: The Network Adapter could not establish the connection".

Problem 1

The information in the `$ORACLE_HOME/integration/interconnect/hub/hub.ini` file is not valid.

Solution 1

Provide the correct information in the `$ORACLE_HOME/integration/interconnect/hub/hub.ini` file.

```
[Database]

hub_username=ichub
```

encrypted_hub_password= *encrypted_password* use $ORACLE_HOME/integration/interconnect/bin/encrypt for encryption

hub_host=*hub_host_name*

hub_instance=*database_sid*

hub_port=*tns_listsner_port*

hub_use_thin_jdbc=*true*

[Repository Info]

repository_name=InterConnectRepository1012

Problem 2

The hub schema does not exist on the hub database.

Solution 2

Run the $ORACLE_HOME/integration/interconnect/repository/hubschema script to create the hub schema.

E.1.7 DB Adapter Does Not Pick Up Messages That Have Been Published from the Database

The database adapter does not pick up messages that have been published from the database, even though the publshing PL/SQL procedure was performed successfully and the message object record exists in the MESSAGEOBJECT table.

Problem

You might have published the message for the wrong application. The value provided for srcAppName+partition id parameter in the publishing PL/SQL procedure pub_*eventorprocedure_name_metadata_owner_version* does not match the application parameter value in the *$ORACLE_HOME*/integration/interconnect/adapters/*adapter_name*/adapter.ini file.

Solution

Ensure that the value provided for the srcAppName+partition id parameter matches the value of the application parameter in the *$ORACLE_HOME*/integration/interconnect/adapters/*adapter_name*/adapter.ini file.

E.1.8 DB Adapter Does Not Publish Messages

The database adapter is not publishing the messages.

Problem 1

The application name specified in the trigger that publishes the message to the OAI scheme does not match with the application name specified in the application parameter of the adapter.ini file.

Solution 1

Ensure that the application name specified in the trigger that publishes the message to the OAI schema and the application name specified in the application parameter of

the adapter.ini file are same. If the partition parameter in the adapter.ini file contains a value, then, in the trigger, the application name should be specified as the value of the application parameter followed by the value of the partition parameter. In addition, ensure that the parent-child relationships are properly handled according to the structure created in iStudio.

Problem 2

The value of the db_bridge_schema1_num_readers parameter specified in the adapter.ini file is incorrect.

Solution 2

The value of the db_bridge_schema1_num_readers parameter in the adapter.ini file should be greater than zero.

E.1.9 FTP Adapter Cannot Match Incoming Message with Any D3L Definiton

The FTP adapter throws an error message that it cannot match the incoming message with any of the D3L defintions, and drops the message.

Problem 1

The ota.d3ls parameter is either empty or points to the wrong D3L file(s).

Solution 1

Provide a valid D3L file for ota.d3ls (or list of D3L files).

Problem 2

The D3L header does not contain the correct information for OracleAS Integration InterConnect event mapping.

Solution 2

Ensure that the

- name tag in the D3L header matched the BusinessObject event name in iStudio.
- Object Name field in the Define application View window of the Publish/Subscribe/Invoke/Implement wizard is not empty.
- object tag matches the value of the Object Name field in the Define application View window of the Publish/Subscribe/Invoke/Implement wizard.

E.1.10 AQ Adapter Does Not Pick Up Mesage from the Spoke AQ

The AQ adapter does not pick up message that have been enqueued to a spoke AQ.

Problem 1

You have not provided the name of the spoke queue in the Deploy Tab in iStudio for this integration point.

Solution 1

Provide the name of the spoke queue. The AQ adapter log file should contain a line *timestamp*: AQ Adapter created a reader for queue *queue_name*.

Problem 2

The spoke queue might be a multiconsumer queue (JMS Topic) and you might connect to it with an incorrect consumer name.

Solution 2

Ensure that the `aq_bridge_consumer_name` parameter in the *`$ORACLE_HOME`*`/integration/interconnect/adapters/`*`adapter_name`*`/adapter.ini` file is correct and matches the name of the consumer that the message was enqueued for.

Problem 3

The message structure defined in iStudio does not match the message you are trying to publish.

Solution 3

Ensure that the message structure defined in iStudio matches the message you are trying to publish.

E.1.11 FTP Adapter Does Not Publish or Subscribe to Messages

The FTP adapter is not publishing or subscribing to the messages.

Problem 1

The Transport properties of the FTP adapter might not be correct.

Solution 1

Specify 2 as the value of the log level parameter `agent_log_level` in the `adapter.ini` file of the FTP adapter. Restart the adapter and try to publish a message or subscribe to a message. You can see a log starting with `TransportProperties.TransportProperties()`. This log contains all the transport properties for the FTP adapter in the form of name value pairs. Ensure that the values of these properties are correct.

Problem 2

You might not have created a publish or subscribe event in iStudio for the FTP adapter that corresponds to the message structure. The FTP adapter is not able to match the message data to any of the events defined during design time and therefore cannot publish or subscribe the message.

Solution 2

Create a publish or subscribe event for the FTP adapter in iStudio corresponding to the message structure.

Problem 3

The FTP server is not based on RFC standards.

Solution 3

Ensure that the FTP server is based on RFC standards.

E.1.12 HTTP Adapter Does Not Publish or Subscribe to Messages

The HTTP adapter is not publishing or subscribing to the messages.

Problem 1

The Transport properties of the HTTP adapter might not be correct.

Solution 1

Specify 2 as the value of the log level parameter `agent_log_level` in the `adapter.ini` file of the HTTP adapter. Restart the adapter and try to publish a message or subscribe to a message. You can see a log starting with `TransportProperties.TransportProperties()`. This log contains all the transport properties of the HTTP adapter in the form of name value pairs. Ensure that the values of these properties are correct.

Problem 2

You might not have created a publish or subscribe event in iStudio for the HTTP adapter that corresponds to the message structure. The HTTP adapter is not able to match the message data to any of the events defined during design time and therefore cannot publish or subscribe the message.

Solution 2

Create a publish or subscribe event for the HTTP adapter in iStudio corresponding to the message structure.

Problem 3

The transport servlet is not deployed properly.

Solution 3

Ensure that the transport servlet that receives the messages over HTTP and directs these messages to the HTTP receiver through RMI is deployed properly. In addition, you should never change the `http.receiver.instance_name` parameter in the `adapter.ini` file after the adapter installation.

Problem 4

The send endpoint defined by the `ota.send.endpoint` parameter in the `adapter.ini` file or in iStudio is incorrect.

Solution 4

Ensure that the send endpoint defined in the `adapter.ini` file or iStudio is correct. If the send endpoint expects the message in some particular form, then you can customize the outgoing messages using the `SenderCustomizer` parameter.

E.1.13 CBR Issue with DB Adapter

I created an event with the name ProdRelease. The root element name is BatchProdRelease. When I restart the adapter, it displays the following error:

"An exception occured while evaluating the CBR expression: (ProdRelease_CO:OAI_META/V1.BatchProdRelease_CO:OAI_META/V1.Header.PlantCode == "GFL"). Following is a detailed message. oracle.oai.agent.common.AgentRuntimeException: Specified attribute (BatchProdRelease_CO:OAI_META/V1) does not exist in Data Type (ProdRelease_CO:OAI_META/V1)."

Problem

The root element name is a substring of the event name.

Solution

Ensure that the root element name does not have a substring of the event name. The root element name in this situation is BatchProdRelease and the event name is ProdRelease. This causes the adapter to fail. A workaround for this is to change either the root element name or event name.

E.1.14 Workflow Configuration

How can I configure Oracle Workflow with OracleAS Integration InterConnect?

Solution

To configure Oracle Workflow with OracleAS Integration InterConnect, perform the following steps:

1. Navigate to your Oracle Workflow home page.
2. Click **Global Workflow Preferences**. The Global Workflow Preferences page is displayed.
3. Ensure that System Status is set to **Enabled**.
4. Open the `http://host-name/pls/DAD location/wf_setup.check_all` URL where host-name is the name of your computer. For example,

   ```
   http://workflowinstall-pc/pls/wf/wf_setup.check_all
   ```

   The Check Setup page is displayed.
5. Schedule listeners for the `WF_ERROR` and `WF_IN` agents in the Listeners for Local Inbound Agents group. To schedule a listener for the `WF_ERROR` agent:
   a. In the Listeners for Local Inbound Agents group, click **Create** for the `WF_ERROR` agent. The Edit Listener page is displayed.
   b. In the Edit Listener page, enter `10` in the **Seconds** field.
   c. Click **Submit**.

   A listener for the `WF_ERROR` agent has been scheduled. Similarly, schedule a listener for the `WF_IN` agent.

   Note: It is recommended that you set the listener to run every 10 seconds. To obtain faster response times, schedule additional listeners rather than lowering the interval between runs. For information about scheduling listeners, refer to *Oracle Workflow Administrator's Guide*.

   ---
6. In the Check Setup page, under the Database Init.ora Parameters group, ensure that the value of `aq_tm_processes` is greater than or equal to 1. Similarly, ensure that the value of `job_queue_processes` is greater than or equal to 5.
7. In the Event Subscriptions group, create subscriptions for the following events:
   - `oracle.apps.wf.event.agent.create`
   - `oracle.apps.wf.event.event.create`
   - `oracle.apps.wf.event.subscription.create`

   To add subscriptions for an event, click **Add Subscription** at the bottom of the Check Setup page. The Edit Subscription page is displayed.

8. To create a subscription for the `oracle.apps.wf.event.agent.create event`, specify the following values:
   a. Select your Oracle Workflow system from the **System** list. You can search by % to locate registered systems.
   b. Select **External** from the Source Type list.
   c. Enter the event for which you are creating the subscription in the Event Filter field. For example, if you are creating a subscription for the `oracle.apps.wf.event.agent.create event`, enter `oracle.apps.wf.event.agent.create`.
   d. Select **Enabled** from the **Status** list.
   e. Select **Key** from the **Rule Data** list.
   f. Enter `wf_event_functions_pkg.receive` in the **Rule Function** field.

A subscription for the `oracle.apps.wf.event.agent.create` event has been created. Similarly, create subscriptions for the `oracle.apps.wf.event.event.create` and the `oracle.apps.wf.event.subscription.create` events. The subscriptions that you create are displayed in the Check Setup page. The configuration of Oracle Workflow with OracleAS Integration InterConnect is complete.

E.2 Need More Help?

You can find more solutions on Oracle *MetaLink*, `http://metalink.oracle.com`. If you do not find a solution for your problem, log a service request.

See Also:

- *Oracle Application Server Release Notes*, available on the Oracle Technology Network: `http://www.oracle.com/technology/documentation/index.html`

Glossary

Advanced Business Application Programming

A programming language developed by SAP for application development purposes.

adapter

Enables third-party environments to participate in application integration. An adapter has two major tasks:

- Provide connectivity between an application and the hub.
- Transform and route messages between the application and the hub.

adapter.ini file

An initialization parameter file that an adapter uses at startup to connect to an application.

advanced queuing adapter

Enables an Oracle Advanced Queuing application to be integrated with other applications using OracleAS Integration InterConnect.

agent

A subcomponent of an adapter that handles runtime instructions. The agent is independent of the application to which the adapter connects. The agent focuses on the integration scenario based on the integration metadata in the repository.

application

A component integrated with OracleAS Integration InterConnect.

application view

A native view translated into the syntax used by an adapter. . Each application has its own application view of data that allows it to participate in the integration. The application view of data uses transformations to map into the common view.

bridge

A subcomponent of an agent adapter that transfers data between the application and OracleAS Integration InterConnect. The bridge is the protocol/application-specific piece of the adapter that communicates with the application.

Business Application Programming Interface

Standardized programming interface that enables external applications to access the business processes and data of the R/3 system.

business object

A collection of logically related integration points.

cipher suites

A set of cryptographic algorithms. SSL supports different cryptographic algorithms, or ciphers, for tasks such as authenticating the server and client to each other, transmitting certificates, and establishing session keys. Clients and servers support different cipher suites depending on factors such as the SSL version supported, company policies regarding permissible encryption strength, and government restrictions on export of SSL-enabled software.

common view

A view that is syntactically and semantically in the OracleAS Integration InterConnect format. The common view:

- Identifies the list of integration points for applications. Applications participate in integration by binding to one or more common view integration points, such as creating a purchase order and creating a new customer.
- Eliminates the complexity of multiple integration points between applications.

content-based routing

Messages routed to specific applications based on business rules or message content.

cross-reference tables

Keys for corresponding entities created in different applications can be correlated through cross-referencing.

D3L

Data Definition Description Language. An XML-based message description language that describes application message information in its native format, also known as its native view.

database adapter

Enables an Oracle Application, typically PL/SQL-based, to be integrated with other applications using OracleAS Integration InterConnect.

design time

During the design phase, a business analyst uses iStudio to define the integration objects, applications that participate in the integration, and the specifications of the data exchanged between applications.

domain value maps

Code tables mapped across different systems.

DTD

Document Type Definition. A set of rules that defines the allowable structure of an XML document. DTDs are text files that derive their format from SGML and are either embedded within an XML document.

EAI

Enterprise Application Integration. The integration of applications and business processes within the same company.

endpoints

The physical destination points for messages exchanged between OracleAS Integration InterConnect and an application.

event

An integration point used to pattern the publish/subscribe model. An event has associated data that is the common view of all the data to be exchanged through this event. An event can be published or subscribed by an application.

event map

Allows application data to be mapped to an OracleAS Integration InterConnect event without the application having to know about the OracleAS Integration InterConnect event itself.

FTP adapter

Enables an FTP to be integrated with other applications using OracleAS Integration InterConnect.

HTTP

Hypertext Protocol Transfer. The underlying format, or protocol, used by the Web to format and transmit messages and determine what actions Web servers and browsers should take in response to various commands. HTTP is the protocol used between Oracle Application Server and clients.

HTTP adapter

Enables an Oracle HTTP application to be integrated with other applications using OracleAS Integration InterConnect. This adapter is useful in all EAI environments that use the HTTP transport protocol.

IDoc Type

Indicates the SAP format used to transfer the data for a business transaction. An IDoc is a real business process in the form of an IDoc type. An IDoc type is described using the following components:

- A control record is the format of the control record, which is identical for all IDoc types.
- One or more data records consist of a fixed administration part and a data part (segment). The number and format of the segments can be different for each IDoc type.
- Status records describe the processing stages which an IDoc can pass through and have identical formats for each IDoc type.

invoke/implement model

An application involes a procedure by sending data out to the OracleAS Integration InterConnect hub and expects return of the result from an applicaion implementing the procedure. An application implements a procedure by receiving data from the OracleAS Integration InterConnect hub and returns the result once the procedure has been performed. In iStudio, a procedure is used to model this scenario.

IMAP4

Internet Message Access Protocol 4. IMAP4 is a standard protocol for accessing e-mail from a local server. IMAP4 is a client/server protocol in which e-mail is received and held for users by their Internet server. Users can view just the heading and sender of

the e-mail, and then decide whether to download the e-mail. Users can also create and manipulate folders or mailboxes on the server, delete messages, or search for certain parts or an entire note. IMAP requires continual access to the server during the time that users work with their e-mail.

iStudio

A design time integration specification tool targeted at business analysts. This tool helps business analysts specify the integration logic at a functional level, instead of a technical coding level. iStudio exposes the integration methodology using simple wizards and reduces, or eliminates, the need for writing code to specify the integration logic. This reduces the total time required to complete an integration.

metadata

A definition or description of data (essentially, data about data).

MQ Series adapter

Enables OracleAS Integration InterConnect to send message to and receive messages from the MQ Series queues and topics.

native view

An application's message information in its native format (for example, SAP IDoc). Native events are both syntactically and semantically in the native format of the application, and are defined external to OracleAS Integration InterConnect.

Oracle Wallet Manager

A Java-based application that security administrators use to manage public-key security credentials on clients and server. Security credentials consist of a public/private key pair, a certificate, and a trustpoint.

Oracle Workflow

Integrated with OracleAS Integration InterConnect and is used for business process collaborations across two or more applications.

OracleAS Integration InterConnect

The integration hub that coordinates the communication and transformation of messages between two or more heterogeneous applications. OracleAS Integration InterConnect defines business events, their associated data, and any transformations required to map one application's view of a business object to another's view.

payload

The data sent between applications. For example, the payload data for a purchase order sent from one application to another application may include the product name, the quantity ordered, and the price.

persistence

The ability to save data and restore it when needed.

procedure

An integration point used to pattern the invoke/implement model. A procedure has associated data that is the common view of all the data to be exchanged through this procedure. A procedure can be invoked or implemented by an application.

project

Encapsulates all the integration logic for one integration scenario.

proxy host

A server through which messages sent to remote Web servers must pass. A proxy server also prevents users outside a company's firewall from breaking into an organization's private network.

publish/subscribe model

An application publishes an event when it sends data out to the OracleAS Integration InterConnect hub without knowing the destination applications. Furthermore, data is not expected in return. An application subscribes to an event if it receives the data from the OracleAS Integration InterConnect hub regardless of who sent the data. Furthermore, it does not send out any data in return. Events in iStudio are used to model this scenario.

realm

Realms enable the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and authorization database.

repository

The repository has the following functionality:

- At design time, all integration logic defined in iStudio is stored in tables in the repository metadata.
- At runtime, the repository provides access to this metadata for an metadata to integrate applications.

The repository server is deployed as a standalone Java application running outside the database. The repository schema is a set of tables in the Oracle Application Server Infrastructure.

RMI

Remote Method Invocation. An interaction scheme for distributed objects written in Java. It enables a Java program running on one computer to access the methods of another Java program running on another computer.

runtime

For each application participating in a specific integration, OracleAS Integration InterConnect attaches one or more adapters to it. At runtime, the adapters retrieve the metadata from the repository to determine the format of messages, perform transformations between the various data formats, and route the messages to the appropriate queues in the OracleAS Integration InterConnect hub.

SMTP

Simple Mail Transfer Protocol. A TCP/IP protocol for sending and receiving e-mail. SMTP is typically used with one of two other protocols, Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP), that enable users to save messages in a server mailbox and periodically download them. Users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving messages on their local server.

SMTP adapter

The SMTP adapter enables an SMTP application to be integrated with other applications using OracleAS Integration InterConnect. This adapter is useful in all EAL environments where e-mail used the IMAP4 and SMTP transport protocols.

SSL

Secure Sockets Layer. SSL is a standard for the secure transmission of documents over the Internet using HTTPS (secure HTTP). SSL uses digital signatures to ensure that transmitted data is not tampered with.

tracking fields

One or more application view fields in the context of a particular event. Used to track the event instances at runtime.

wallet

A wallet is an abstraction used to store and manager security credentials for an individual entity. It implements the storage and retrieval of credentials for use with various cryptographic services.

workspace

Stores user settings and preferences such as application login credentials and last opened project.

XML

eXtensible Markup Language. XML is a set of rules for defining data markup in a plain text format.

XSLT

Extensible Stylesheet Language transformations. XSLT describes how to transform the structure of an XML document into a differently-structured XML document. XSLT is an extension of the Extensible Stylesheet Language (XSL). XSLT shows how to reorganize the XML document into another data structure (that can then be presented by following an XSL style sheet).

Index

A

B

C

D

E

F

H

I

L

M

N

O

P

R

S

T

W

X