BEA WebLogic Integration™

Using Application Integration

Release 7.0
Document Date: June 2002

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Using Application Integration

Part Number	Date	Software Version
N/A	June 2002	7.0

Contents

About This Document

1. Introduction to Application Integration

2. Defining an Application View

3. Using Application Views in the Studio

B. Importing and Exporting Application Views

C. Modular Deployment of Application Integration

Index

About This Document

Using Application Integration is organized as follows:

- "Introduction to Application Integration" provides an overview of the BEA WebLogic Integration Framework and explains how it fits into the WebLogic Server environment and contributes to the BEA EAI solution.
- "Defining an Application View" explains how to log in to the Application View Console, and create and configure Application Views to represent your enterprise's business processes.
- "Using Application Views in the Studio" explains how to use Application Views in the WebLogic Server environment by setting up workflows using the WebLogic Integration Studio.
- "Using Application Views by Writing Custom Code" explains how to use Application Views in the WebLogic Server environment by writing custom Java code.
- "Using the Application View Console" explains how to use namespaces to organize your Application Views by location or department instead of by adapter.
- Appendix A, "Migrating Application Integration Data" explains how to migrate application integration data between WebLogic Server domains.
- Appendix B, "Importing and Exporting Application Views," explains how to use the Import/Export utility to export Application View metadata objects from the repository and how to import those objects back into the repository.
- Appendix C, "Modular Deployment of Application Integration," describes how to deploy an application integration enterprise application outside a WebLogic Integration domain.

What You Need to Know

This document is intended for the following users:

- *Business Analysts*–Business analysts work with technical analysts to ensure accuracy of the business interface functionality, to create Application Views, and to use Application Views within an enterprise.
- *Technical Analysts*–Technical analysts are responsible for configuring an adapter, for setting up WebLogic Integration services to execute information transfers with a legacy system, for configuring solutions using adapters, and for evaluating, mapping, deploying, and maintaining the WebLogic Server environment. This guide is based on the assumption that the technical analyst has thorough knowledge of the entire system.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the "edocs" Product Documentation page at http://edocs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—Print option on your Web browser.

A PDF version of this document is available on the BEA WebLogic Integration documentation home page on the edocs Web site. You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the BEA WebLogic Integration documentation home page, click the PDF Files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

The following resources are also available:

- BEA WebLogic Server documentation (`http://edocs.bea.com`)
- BEA WebLogic Integration documentation (`http://edocs.bea.com`)
- XML Schema Specification (`http://www.w3c.org/TR/xmlschema-formal/`)
- Sun Microsystems, Inc. Java site (`http://www.javasoft.com/`)
- Sun Microsystems, Inc. J2EE Connector Architecture Specification (`http://java.sun.com/j2ee/connector/`)

Contact Us!

Your feedback on the BEA WebLogic Integration documentation is important to us. Send us e-mail at **docsupport@beasys.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the BEA WebLogic Application Integration documentation.

In your e-mail message, please indicate which release of the BEA WebLogic Application Integration documentation you are using.

If you have any questions about this version of BEA WebLogic Integration, or if you have problems installing and running BEA WebLogic Application Integration, contact BEA Customer Support through BEA WebSupport at **www.beasys.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`chmod u+w *`
	`c:\startServer`
	`.doc`
	`wls.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`

Convention	Item
monospace italic text	Identifies variables in code. *Example*: `String expr`
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
. . .	Indicates one of the following in a command line: ■ That an argument can be repeated several times in a command line ■ That the statement omits additional optional arguments ■ That you can enter additional parameters, values, or other information *Example*: `import com.sap.rfc.exception.*;`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Introduction to Application Integration

This document provides instructions for using adapters built with the BEA WebLogic Integration Adapter Development Kit (ADK). It explains how to define Application View services and events and use them in your business processes in a WebLogic Integration environment.

This section provides the following topics:

- Before You Begin
- Creating an Interface to an Adapter
- Defining an Application View
- Using an Application View in a Workflow
- Using an Application View with Web Services

Note: Because all adapters and applications are different, the instructions provided in this document are generic: they are not written for a specific adapter or application. For details about the DBMS adapter provided with the ADK, see "The DBMS Adapter" in *Developing Adapters*.

Before You Begin

Before you can begin using adapters to integrate your enterprise, you must set up your environment and learn about how WebLogic Integration uses adapters and Application Views to help achieve integration.

This section provides the following information:

- Software Prerequisites
- Familiarizing Yourself with Basic Concepts

Software Prerequisites

Note: For a detailed list of prerequisites, see the *BEA WebLogic Platform Release Notes*.

Make sure the following prerequisites are satisfied:

- You have installed WebLogic Platform.
- You have installed JDK 1.3.1. The JDK 1.3 development kit is automatically installed when you install WebLogic Platform. If you prefer, however, you may install your own version, as long as it is 1.3.1-compliant.
- You have included BEA WebLogic Integration as part of your WebLogic Platform installation.
- You have deployed each adapter for which you will define Application Views.

In this release, the application integration functionality of WebLogic Integration is packaged in a single, self-contained J2EE EAR file. This packaging enables you to deploy application integration capabilities on any valid WebLogic Server domain. For example, Web services developers and WebLogic Portal developers can use Application Views to interact with EIS applications. For more information, see Appendix C, “Modular Deployment of Application Integration.”

Familiarizing Yourself with Basic Concepts

If you are not familiar with the basic concepts of application integration, we recommend that you take the time to read the overview of application integration provided in *Introducing Application Integration*. Then you will be ready to learn how to address practical issues, such as when to use one application integration method rather than another, and how to implement the method you select.

Creating an Interface to an Adapter

For each adapter to be used in your enterprise, you must provide an interface to the services and events that it provides. You can create such an interface in either of two ways: by defining Application Views or by writing custom code.

Application views provide the most convenient method of accessing an adapter's resources. In most situations you will probably choose this method for exposing the application functions provided by each adapter. However, if you require more control over an adapter's functions than that afforded by Application Views, you may also write custom code.

You are responsible for deciding whether your enterprise can derive greater benefit from Application Views or custom code. The following sections provide basic guidelines for choosing between these two methods. For details, see Chapter 2, "Defining an Application View."

When to Define an Application View

Most enterprise information system (EIS) applications can be integrated easily by defining Application Views. In general, you should define Application Views if one or more of the following criteria are true:

- You have more than one EIS in your enterprise, and you lack developers with detailed, thorough knowledge of all systems.

- You want to construct business processes using the WebLogic Integration Studio.
- You need to update the parameters of an adapter or one of its processes.

When to Write Custom Code

You should write custom code as an interface to an adapter only if one or more of the following criteria are true:

- You have only one EIS in your enterprise and your developer has thorough, detailed knowledge of the EIS involved in the business processes being coded.
- You do not need to use the business process management (BPM) functions provided by WebLogic Integration.
- Your code will never require changes.

Defining an Application View

An Application View for an adapter is an XML-based interface between WebLogic Server and a particular EIS application. You must define an Application View for each adapter used by your enterprise.

This section describes:

- What Is Defined by an Application View Definition
- How to Define an Application View

What Is Defined by an Application View Definition

When you define an Application View, you must configure communication parameters for it, and then add services and/or events to it. The Application View's services and events expose specific functions of the application. The communication parameters of the Application View govern how the Application View connects to the target EIS.

An Application View definition specifies:

- A unique name for the Application View
- Security privileges for users of the Application View
- Parameters for the:
  - Application
  - Network connections between the application and the Application View
  - Management of the pool of connections available to the Application View
  - Load balancing to be performed by the Application View

How to Define an Application View

This section provides a high-level overview of the procedure you must complete to define Application Views for adapters. For detailed instructions, see Chapter 2, "Defining an Application View."

Defining an Application View involves the following steps:

- Step 1: Name and Configure Connection Parameters for an Application View
- Step 2: Add Services and Events to the Application View
- Step 3: Test Services and Events

Step 1: Name and Configure Connection Parameters for an Application View

The first step in defining an Application View for an adapter is to log on to the Application View Console, optionally create or select one or more folders in which the Application View will reside, and configure EIS connection parameters for it.

For details about creating and configuring an Application View, see the following topics:

- "Step 1: Log On to the Application View Console" on page 2-5
- "Steps 2 and 3: Define an Application View and Configure Connection Parameters" on page 2-7

Step 2: Add Services and Events to the Application View

Services and events support a subset of an application's business processes by enabling WebLogic Server clients to interact with the application functions you specify. The services and events offered by an Application View allow specific types of transactions between WebLogic Server and the EIS application.

For details about adding services and events to an Application View, see the following topics:

- "Step 4A: Add a Service to an Application View" on page 2-11
- "Step 6B: Test an Application View's Events" on page 2-24

Step 3: Test Services and Events

Verify that your services or events interact properly with the EIS application.

For details about testing services and events, see the following topics:

- "Step 6A: Test an Application View's Services" on page 2-21
- "Step 6B: Test an Application View's Events" on page 2-24

Using an Application View in a Workflow

Once you define an Application View in your WebLogic Integration environment, you can deploy it on WebLogic Server and use it to implement your enterprise's business processes in a business process workflow.

You can use Application Views in business processes in either of the following ways:

- By designing business process workflows in the WebLogic Integration Studio
- By writing custom code

When an Application View is used in your business process workflow, the end result is a deployed electronic representation of your enterprise's business process. The workflow specifies the transactions to be performed by your applications to accomplish the business processes. The Application Views perform the transactions themselves.

Using an Application View in the WebLogic Integration Studio

The most common way to use an Application View in your enterprise's business processes is by designing a workflow in the WebLogic Integration Studio. The Studio is a graphical user interface (GUI) for designing business process workflows. These workflows can include Application View services and events.

You can use an Application View to support services and events in any of the following four ways:

- Task 1: Set Up a Task Node to Call an Application View Service
- Task 2: Set Up an Event Node to Wait for a Response from an Asynchronous Application View Service
- Task 3: Create a Workflow Started by an Application View Event
- Task 4: Set Up an Event Node to Wait for an Application View Event

For detailed information about each task, see Chapter 3, "Using Application Views in the Studio."

Using an Application View by Writing Custom Code

If you do not implement your business process by using an Application View through the Studio, you must write custom Java code, instead. For instructions, see Chapter 4, "Using Application Views by Writing Custom Code."

Choosing a Method for Implementing a Business Process

WebLogic Integration allows you to implement your business processes by using either of two methods: by creating a workflow in the Studio or by writing custom code. Any business process can be implemented as a Studio workflow.

Custom coding, however, should be attempted only if the target business process is extremely simple and specialized. In this document, custom coding is described only as an alternate method to be used in situations that require it. For a list of such situations, see "When to Write Custom Java Code" on page 1-9.

When to Use the WebLogic Integration Studio

Use the WebLogic Integration Studio to implement a business process if one or more of the following criteria are true:

- Your business processes require complicated error management, persistent processes, and sophisticated conditional branching.

  For example, if your business process must receive numerous events, select a subset of them, perform complex branched actions, generate many complex messages, and send the messages to various WebLogic Server clients, then you should use the Studio.

- Your business process requires periodic changes.

  The Studio reduces the number of required compile/test/debug cycles.

- Your developers (like those in most organizations) are valuable and scarce.

When to Write Custom Java Code

Write custom code to implement a business process only if one or more of the following criteria are true:

- Your business process is simple; that is, it includes no complicated error recovery, long-lived processes, conditional branching, or joining of the process flow.

  For example, if your business process performs a limited set of actions on an incoming message, and then routes the message to a small number of client applications, you can safely write custom code for it.

- You do not anticipate the need for frequent updates to the business process.

  Whenever you update custom code, a full compile/test/debug cycle, which can be costly, is required.

- Your organization can afford to allocate developers for the job of implementing business processes in code.

Using an Application View with Web Services

A developer of Web Services can use an AppView Control to provide users of BEA WebLogic Workshop with a Web Service that interacts with an EIS application. The interaction is implemented using a Java API. A Web services developer is not required to be an expert on the EIS to use its capabilities. A developer can invoke Application View services both synchronously and asynchronously, and can subscribe to Application View events using simple Java objects. For more information about using AppView Controls, see "Application View Control: Accessing an Enterprise Application from a Web Service" in BEA WebLogic Workshop online documentation at the following location:

```
http://edocs.bea.com/workshop/docs70/help/index.html#guide/
controls/appview/conAppViewCtrlAccessAnEnterpriseAppFromAJWS.html
```

CHAPTER

2 Defining an Application View

This section presents the following topics:

- Before You Begin
- High-Level Procedure for Defining an Application View
- Sample Detailed Procedure for Defining an Application View
- Editing an Application View

Before You Begin

When you define an Application View, you are creating an XML-based interface between WebLogic Server and a particular EIS application within your enterprise. Once you create the Application View, a business analyst can use it to create business processes that use the application. For any adapter, you can create any number of Application Views, each of which may contain any number of services and events.

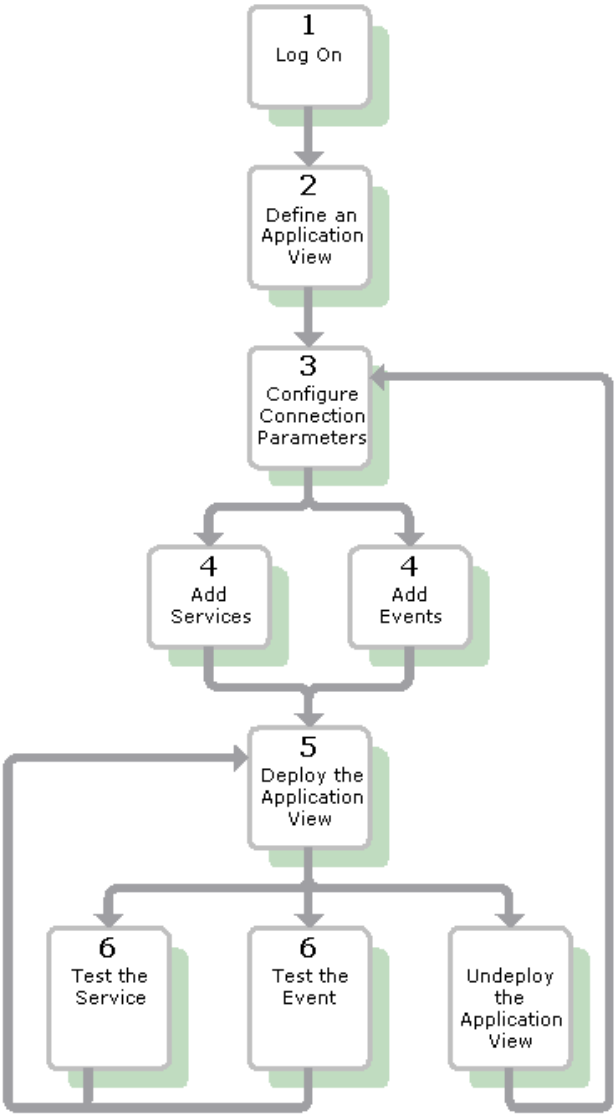
Before you attempt to define an Application View, make sure the following prerequisites are satisfied:

- The appropriate adapter has been developed using the ADK. You can create and configure Application Views only for existing adapters.
- Determine which business processes need to be supported by the Application View you are configuring. The required business processes determine the types of services and events you include in your Application Views. Therefore, you must gather information about the application's business requirements from the business analyst. Once you determine the necessary business processes, you can define and test the appropriate services and events.

High-Level Procedure for Defining an Application View

Figure 2-1 summarizes the procedure for defining and configuring an Application View.

Figure 2-1 Procedure for Defining and Configuring an Application View



1. Log on to the WebLogic Integration Application View Console. For detailed information, see “Step 1: Log On to the Application View Console” on page 2-5.

2. Click Add Application View to create a new Application View for the appropriate adapter. An Application View enables a set of business processes for the specified adapter's target EIS application. For detailed information, see "Steps 2 and 3: Define an Application View and Configure Connection Parameters" on page 2-7.

3. On the Configure Connection Parameters page, enter application connection parameters. Alternatively, you can select a previously deployed connection using the Select Existing Connection page. For detailed information, see "Steps 2 and 3: Define an Application View and Configure Connection Parameters" on page 2-7.

   The information is validated, and the Application View is configured to connect to the specified system.

4. Click Add Event or Add Service to define the appropriate events and services for this Application View.

5. Deploy the Application View on WebLogic Server so other entities can interact with it according to your security settings.

   Note: You cannot test an Application View unless it is deployed.

6. Test all services and events to make sure they can properly interact with the target EIS application.

   Once your services and events are tested and functioning, you can use the Application View in workflows. For more information, see Chapter 3, "Using Application Views in the Studio."

7. Undeploy the Application View if you need to reconfigure its connection parameters or add services and events to it.

   Note: When an Application View is undeployed, no other entities can interact with it.

Sample Detailed Procedure for Defining an Application View

This section explains how to define and maintain Application Views using an EIS adapter for a hypothetical database EIS called simply *DBMS*. The steps in the procedure presented here correspond to the steps shown in Figure 2-1.

When you create Application Views for your enterprise, they may look different from those shown in this document. Such differences are to be expected, because the Application View's adapter determines the information required for each Application View page, and each enterprise has its own specialized adapters. For details about an adapter used in your enterprise, consult the relevant technical analyst or EIS specialist.

Note: Before performing the following steps, ensure that WebLogic Server is running on your system.

Step 1: Log On to the Application View Console

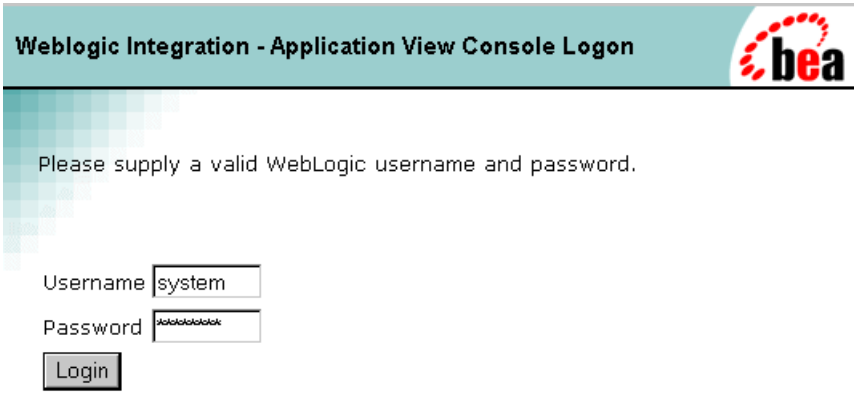
The Application View Console displays all the Application Views in your WebLogic Integration environment, organized in folders.

To log on to the Application View Console:

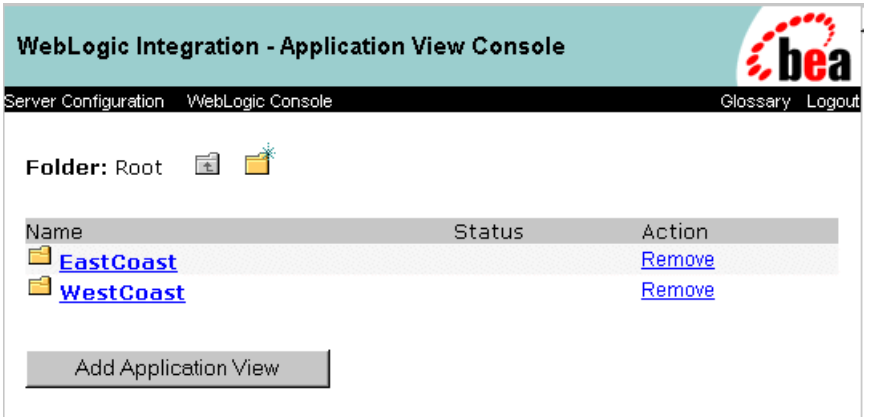
1. Open a new browser window.
2. Enter the URL for your system's Application View Console. The actual URL you enter depends on your system. It should conform to the following format:

   `http://host:port/wlai`

   The Application View Console Logon page is displayed.



3. Enter your WebLogic Server username and password, then click Login. The Application View Console is displayed.



Note: If you do not see a page such as this, consult the WebLogic Server administrator.

4. To add a folder, click the New Folder icon:

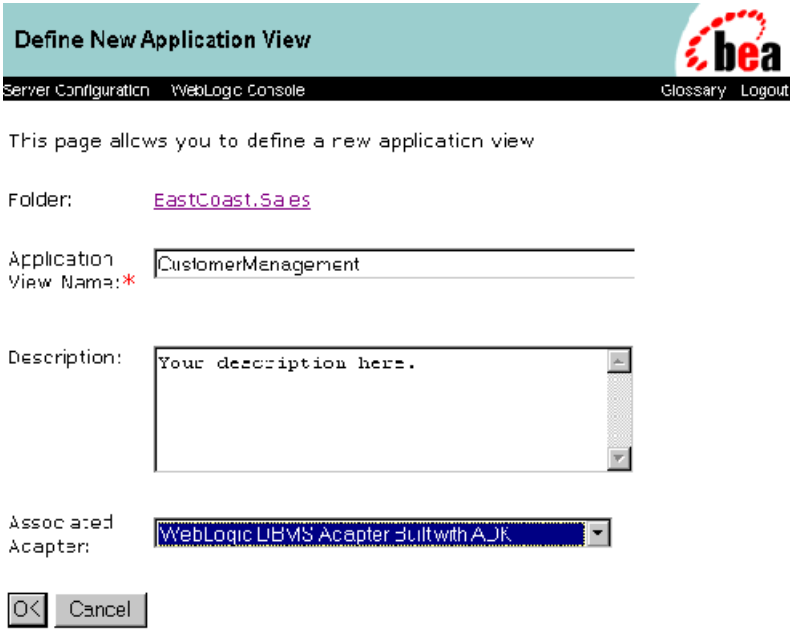
For more information, see "Creating a Folder" on page 5-3.

Steps 2 and 3: Define an Application View and Configure Connection Parameters

1. Add a new Application View to the current folder by clicking Add Application View.

   Note: Make sure you are working in the appropriate folder before performing this step. Once you define an Application View, you cannot move it to another folder.

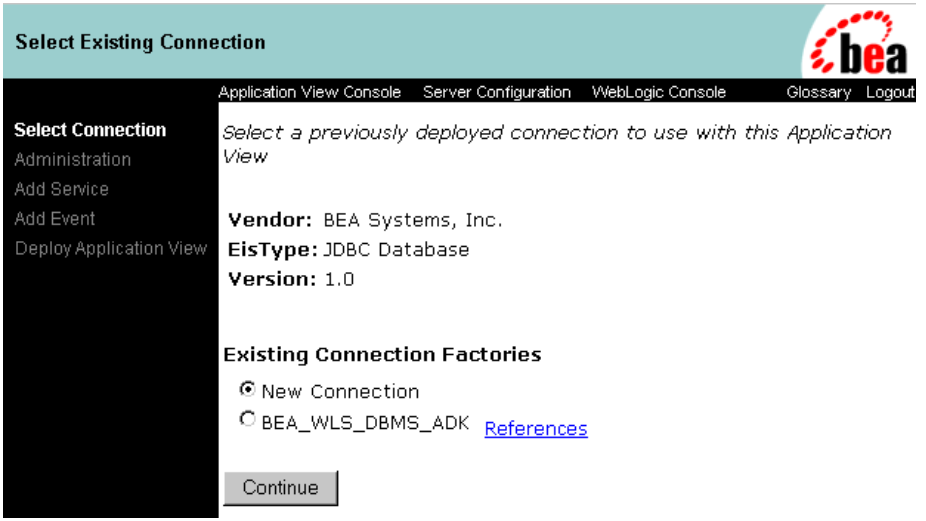
   The Define New Application View page is displayed.



2. In the Application View Name field, enter a name. The name should describe the functions performed by this application. Each Application View name must be unique to its adapter. Any valid Java Identifier is allowed in a name.

Note: The name Root is a reserved word, and cannot be used for an Application View name. If you use Root as a name, you cannot import or export the Application View using the import/export utility.

3. In the Description field, enter any notes that may be helpful to people using this Application View in workflows created in the WebLogic Integration Studio.
4. From the Associated Adapter list, select an adapter to be used to create this Application View.
5. Click OK. The Select Existing Connection page is displayed.



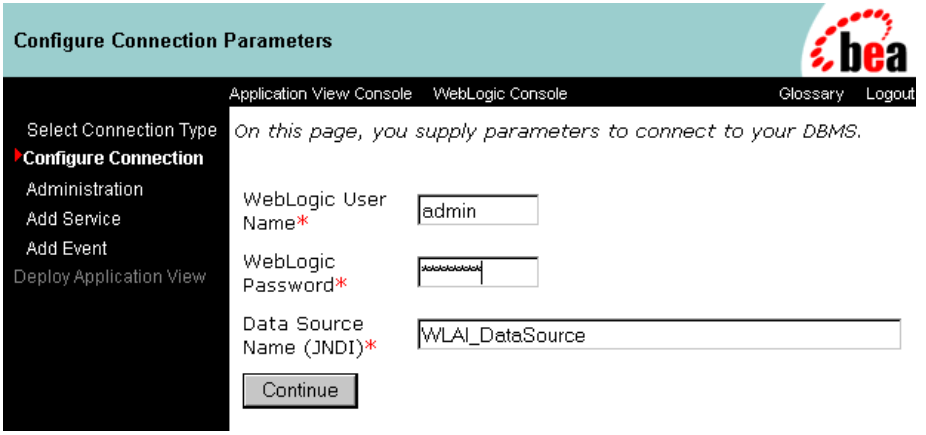
The Select Existing Connection page allows you to choose the type of connection factory to associate with the Application View.

- Select the New Connection option to create a new connection factory. Once the new connection factory is created, other Application Views cannot create a new connection with the same name.
- Select the option for an existing connection factory to share a connection factory with other Application Views. To display the names of Application Views that are deployed with the existing connection factory, click the References link beside the name of an existing connection factory.

From the Connection Factory selection page, you can display the Select Connection or Connection Configuration pages at any time. You can switch between a new connection factory and an existing one at any time before the Application View is deployed.

Using an existing connection factory can simplify server administration, especially in cases where multiple adapters interact with a single EIS. Also, using a shared connection factory allows an administrator to set the connection factory configuration parameters and direct users to select an existing connection. In this case, the users do not have to know how to configure connection parameters.

6. Click Continue. If you choose to create a new connection factory, the Configure Connection Parameters page is displayed.



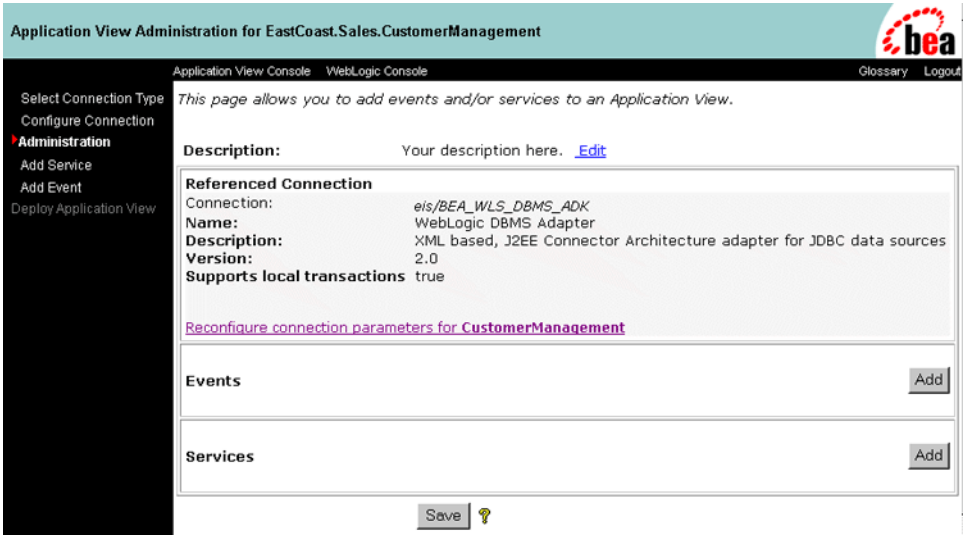
If you choose to use an existing connection factory, the Application View Administration page is displayed. (For information about the Application View Administration page, see step 9.) On the Configure Connection Parameters page, define the network-related information necessary to enable the Application View to interact with the target EIS. You need to enter this information only once per Application View.

7. Enter your WebLogic Server username and password.

   Note: The fields displayed on the page you see may differ from those shown here. Which fields are displayed is determined by the adapter.

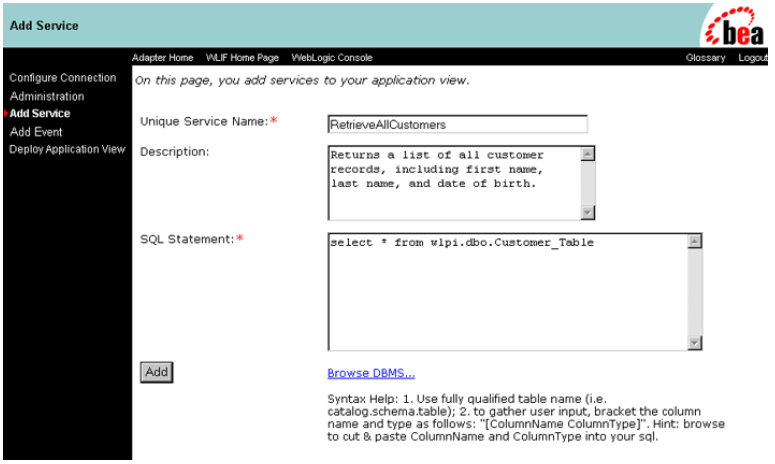
8. For the required information for any remaining fields, consult the relevant technical analyst or EIS specialist.

9. Click Continue. The Application View Administration page is displayed.



Step 4A: Add a Service to an Application View

1. On the Application View Administration page, click Add in the Services row. The Add Service page is displayed.



Note: The fields displayed on the page you see may differ from those shown here. Which fields are displayed is determined by the adapter.

2. In the Unique Service Name field, enter a name. The name should describe the function performed by this service. Each service name must be unique to its Application View. Any valid Java Identifier is allowed in a name.

3. In the Description field, enter any notes which may be helpful to people using this Application View in workflows created in the WebLogic Integration Studio.

4. For the required information for any remaining fields, consult the relevant technical analyst or EIS specialist.

   In many cases, this required information consists of an SQL statement for retrieving information from or updating information in a database. The following sample SQL statement retrieves customer information from a customer table based on a user-specified country value:

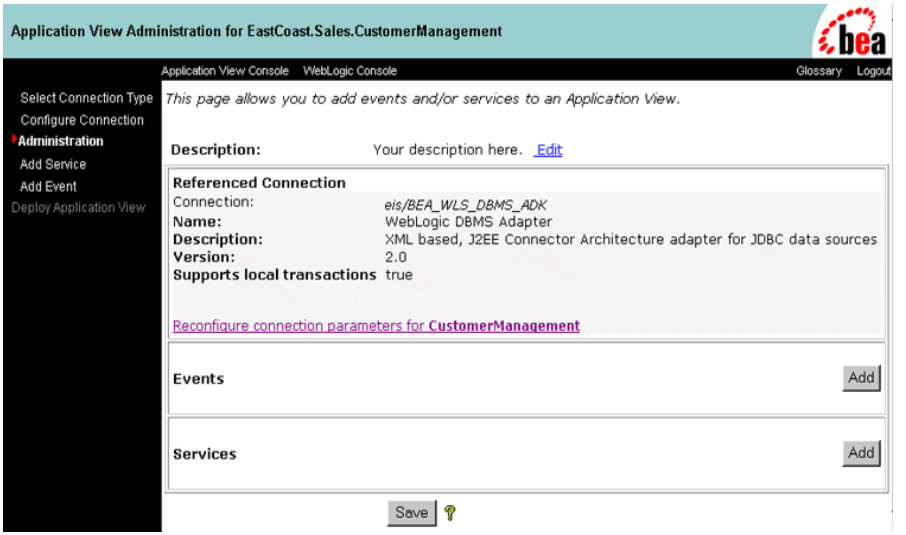
   ```
   select * from PBPUBLIC.CUSTOMER_TABLE
   where COUNTRY=[country varchar]
   ```

   The sample Application Views provided with WebLogic Integration include services which use SQL statements. Display the Application View Administration page and click the View Summary link for a service. The Summary page includes an SQL statement.

5. When finished, click Add.

Step 4B: Add an Event to an Application View

1. In the Application View Console, select Administration. The Application View Administration page is displayed.



2. Click Add in the Events row. The Add Event page is displayed.

Note: The fields displayed on the page you see may differ from those shown here. Which fields are displayed is determined by the adapter.

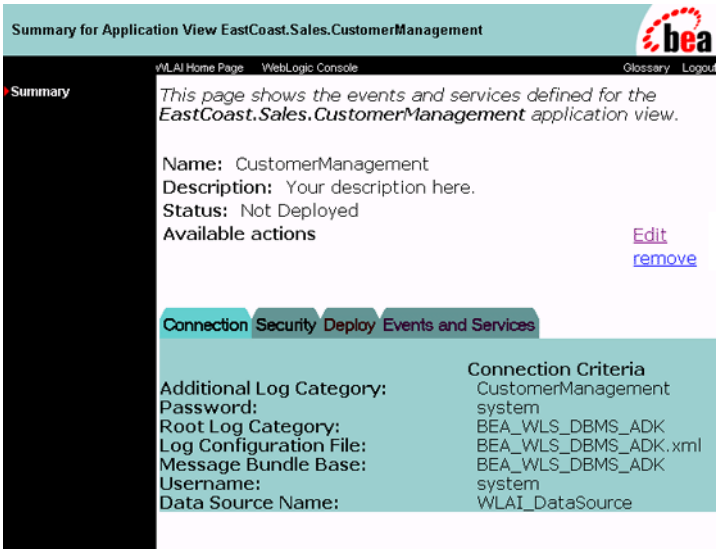
3. In the Unique Event Name field, enter a name. Each event name must be unique to its Application View. Any valid Java Identifier is allowed in a name.
4. In the Description field, enter any notes that may be helpful to people using this Application View in workflows created in the WebLogic Integration Studio.
5. For the required information for any remaining fields, consult the relevant technical analyst or EIS specialist.
6. When finished, click Add. The Application View Administration page is displayed.
7. If you are finished adding services and events, click Continue to deploy the Application View.

Step 5: Deploy an Application View

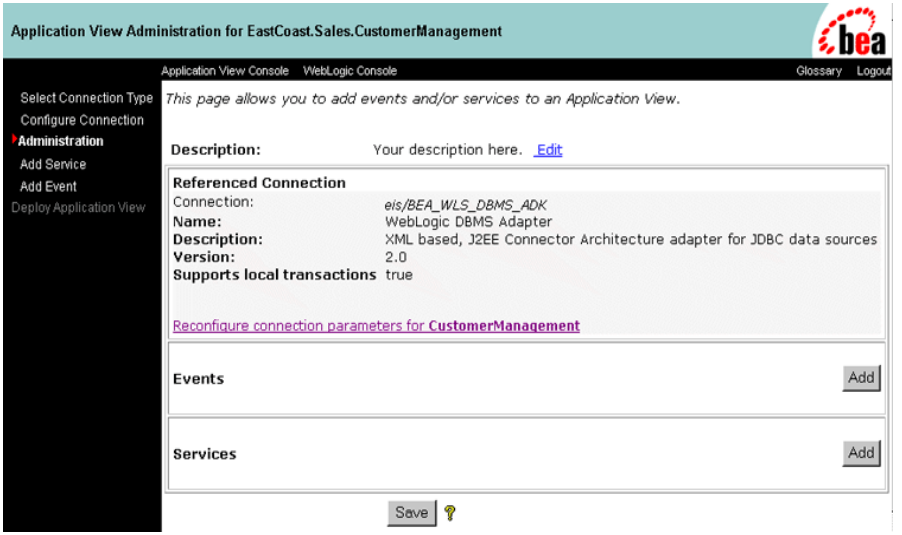
You may deploy an Application View when you have added at least one event or service to it. You must deploy an Application View before you can test its services and events or use the Application View in the WebLogic Server environment. By deploying an Application View, you place relevant metadata about its services and events into a run-time metadata repository. Deployment also makes the Application View available to other WebLogic Server clients. As a result, business processes can interact with the Application View, and you can test the Application View's services and events.

To deploy an Application View:

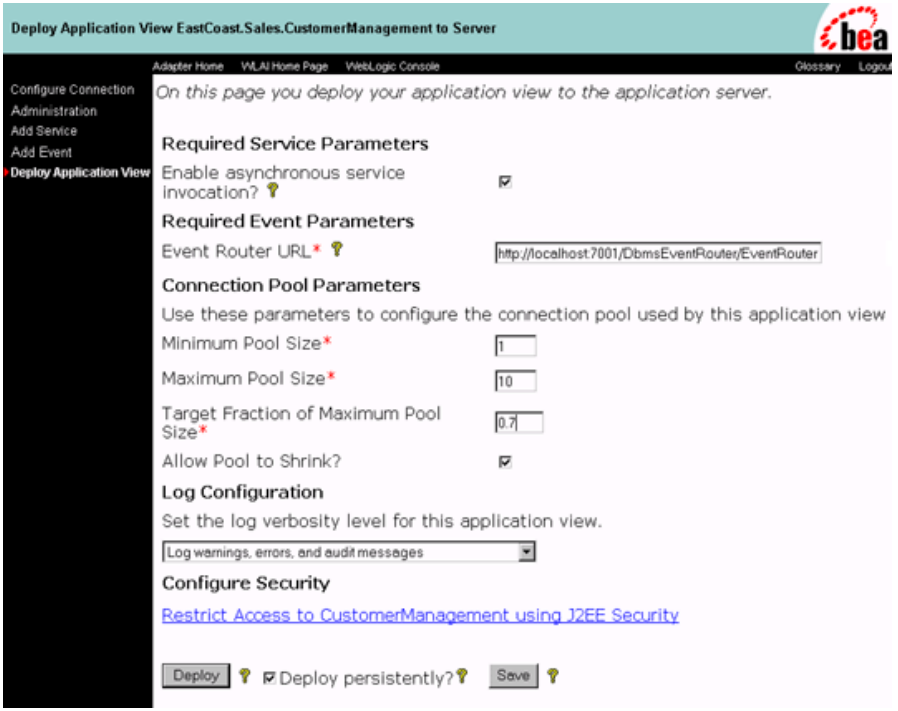
1. Open the Application View as described in "Step 1: Log On to the Application View Console" on page 2-5. The Summary for Application View page is displayed.



2. Click Edit. The Application View Administration page is displayed.



3. Click Continue. The Deploy Application View page is displayed.



Note: Which fields you see on the Deploy Application View page depends on the adapter being used. For a description of all fields, consult the relevant technical analyst or EIS specialist. If the Application View uses a shared connection factory, the connection factory properties are displayed.

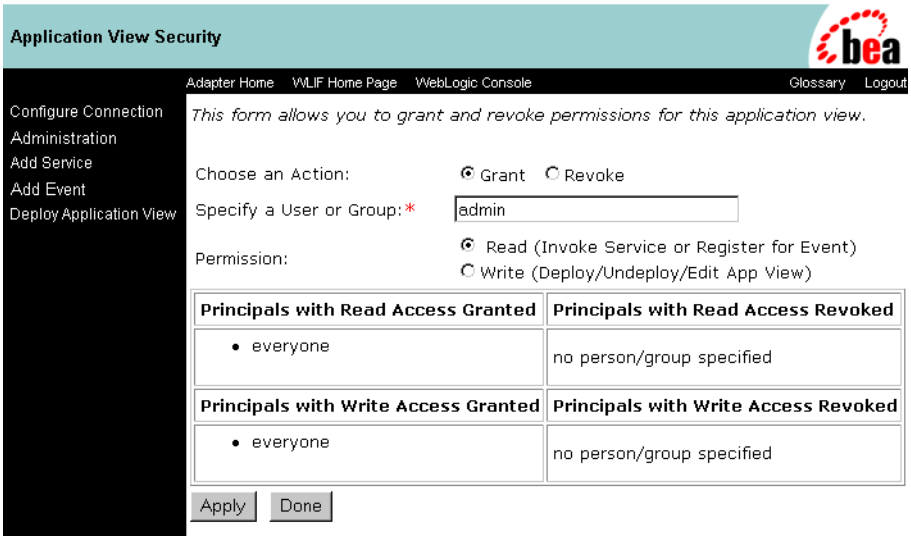
4. To enable the WebLogic Integration Studio or other authorized clients to asynchronously call any services available from this Application View, select Enable Asynchronous Service Invocation.

   An entity that calls an Application View service asynchronously continues its process without waiting for a response from the service.

5. If your Application View supports events, enter the URL of the adapter's event router. For example:
   `http://localhost:7001/YourEIS_EventRouter/EventRouter`

   Note: This field is not displayed if no events are defined for this Application View.

6. In the Minimum Pool Size field, enter the minimum number of connection pools to be used by this Application View. For example: `1`.
7. In the Maximum Pool Size field, enter the maximum number of connection pools to be used by this Application View. For example: `10`.
8. In the Target Fraction of Maximum Pool Size field, enter the ideal pool size, measured from 0 to 1.0. For example, if you specify `0.7`, then the Maximum Pool Size is defined as 10 and the Target Fraction, as 0.7. As a result, the adapter performs load balancing with the goal of maintaining the connection pool size at 70% of the maximum, which, in this case, means with 7 connections.
9. To have unused connections deleted automatically, select Allow Pool to Shrink.
10. In the Log Configuration area, select one of the following options according to your logging preferences:
    - Log errors and audit messages
    - Log warnings, errors, and audit messages
    - Log informational messages, warnings, errors, and audit messages
    - Log all messages
11. If necessary, click Restrict Access to *your_application* using J2EE Security. The Application View Security page is displayed.



Use this page to grant or revoke read and write access to this Application View for a WebLogic Server user or group.

12. When you finish setting up permissions, click Apply to save your changes.
13. Decide whether you want to deploy the Application View now or later. To deploy the Application View later, proceed to step 14. To deploy the Application View now, proceed to step 15.
14. To deploy the Application View later, click Done to return to the Deploy Application View page.

    To save the Application View without deploying it, click Save.

    Note: To save the Application View to be completed later without deploying it now, click Save at any time.

15. To deploy the Application View now, select Deploy Persistently to have this Application View automatically redeployed whenever WebLogic Server is restarted, then click Deploy Application View. The Summary for Application View page is displayed.

If the Application View uses a shared connection factory, the connection factory properties are displayed on the Deploy and Connection tabs. Click the

References link beside the name of an existing connection factory to display the names of Application Views that are deployed using that connection factory.

Optional Step: Undeploy an Application View

You must undeploy an Application View whenever you want to edit its connection parameters, add services and events to it, or disable clients from using it. For information about editing connection parameters, see "Steps 2 and 3: Define an Application View and Configure Connection Parameters" on page 2-7. When an Application View is undeployed, no other WebLogic Server clients can interact with it, and you cannot test its services or events.

To undeploy an Application View:

1. In the Application View Console, click Summary. The Summary for Application View page is displayed.

2. To undeploy the Application View from WebLogic Server, click Undeploy. The Undeploy Application View window is displayed.

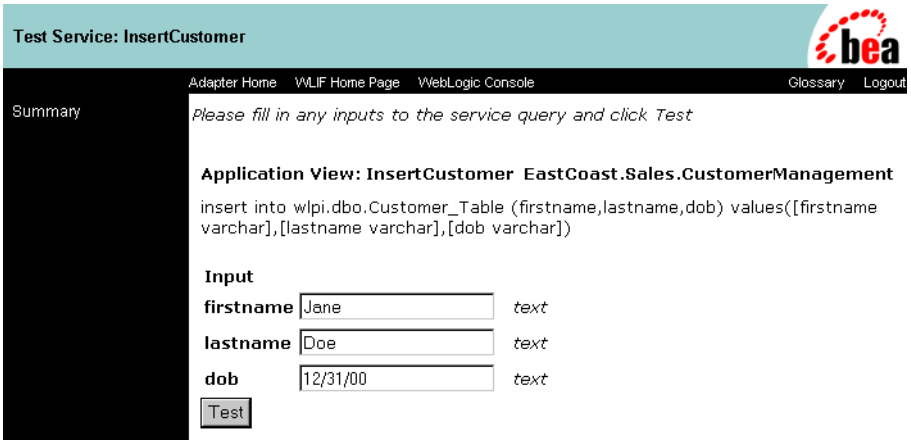
3. Click Confirm. The Summary for Application View page is displayed, giving you an opportunity to deploy the Application View again.

Step 6A: Test an Application View's Services

The purpose of testing an Application View service is to evaluate whether or not that service interacts properly with the target EIS. You can test an Application View only if it is deployed and it contains at least one event or service. To test an Application View service:

1. In the navigation area (on the left side of the window), select Summary. The Summary for Application View page is displayed.

2. In the Services area of the Events and Services tab, find the appropriate service and click Test. The Test Service page is displayed.

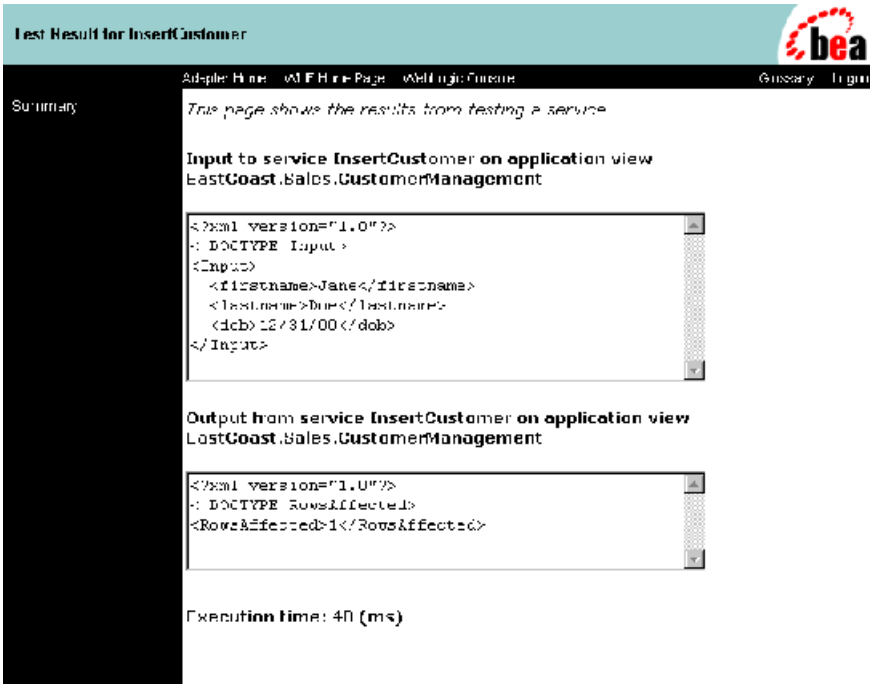


3. If necessary, enter the required data in the appropriate fields.

   Note: The fields displayed on your Test Service page may differ from those show here. Which fields are displayed is determined by the Application View

service. For a description of all fields, consult the relevant technical analyst or EIS specialist.

4. Click Test. If the Application View service correctly processes the input data that you provided in step 3, the test is successful. The Test Result page, on which all input and output documents are listed, is displayed.



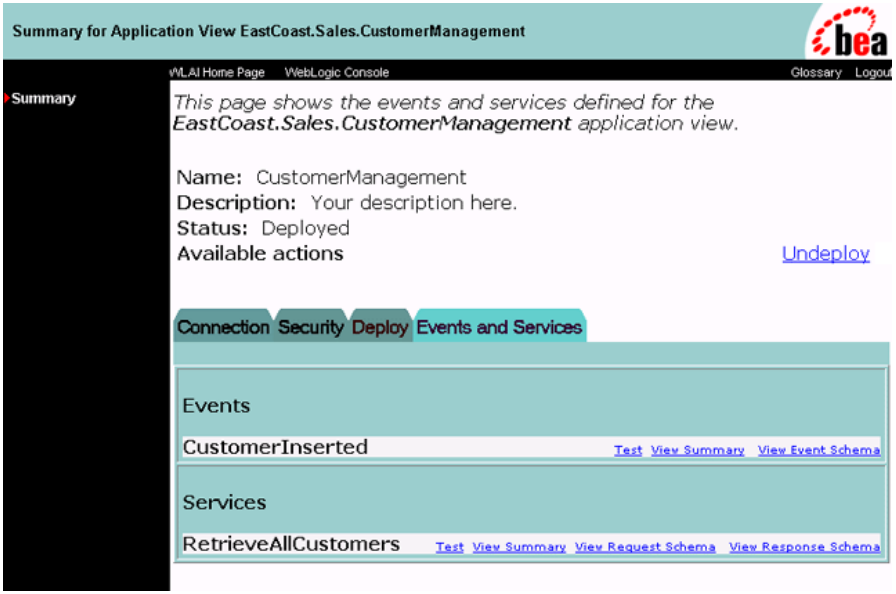
5. Repeat the test procedure (steps 1-4) for each service you want to test.

6. After you finish testing the Application View's services, you may keep the Application View deployed or, if you want to edit it, you may undeploy it.

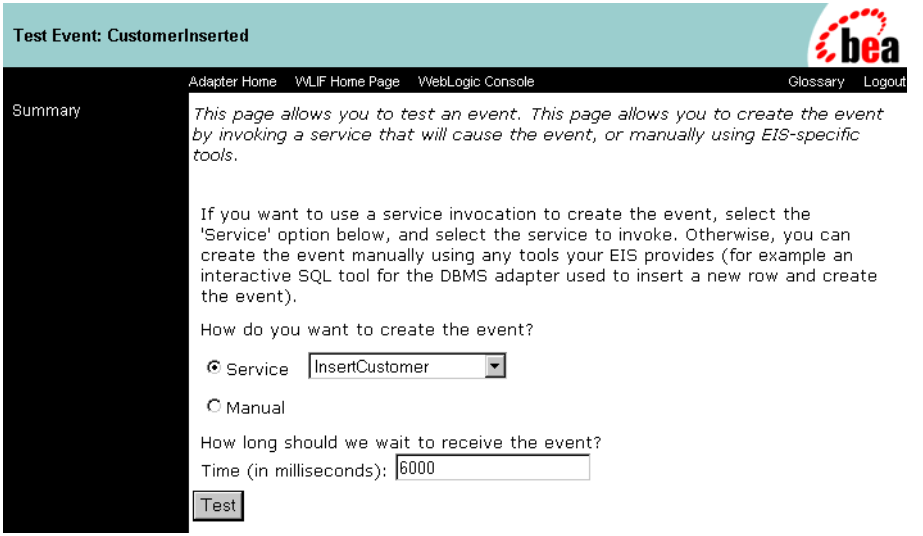
Step 6B: Test an Application View's Events

The purpose of testing your Application View events is to evaluate whether or not the Application View responds correctly to the EIS application. You can test an Application View only if it is deployed and it contains at least one event or service. To test an Application View event:

1. In the navigation area (on the left side of the window), click Summary. The Summary for Application View page is displayed.



2. In the Events area on the Events and Services tab, find your event and click Test. The Test Event page is displayed.



Note: The fields displayed on your Test Event page may differ from those shown here. Which fields are displayed is determined by the Application View event. For a description of all fields, consult the relevant technical analyst or EIS specialist.

3. Select a method for generating the test event:

   - Service: Select Service when you want to use one of the Application View's own services to generate a *canned* event. Then complete the procedure in "If You Select Service" on page 2-25.
   - Manual: Select Manual when you want to generate the event by logging on to an EIS application and performing the appropriate event-generating function. Then complete the procedure in "If You Select Manual" on page 2-28.

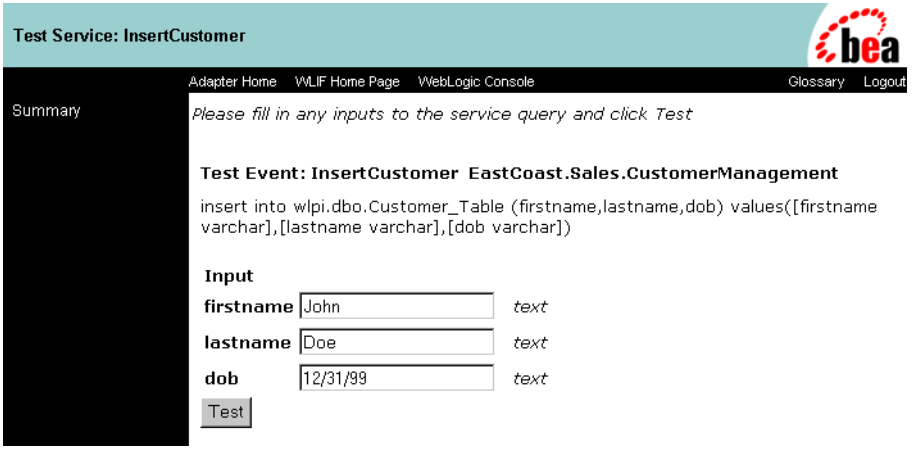
   If the Application View event responds correctly before the specified amount of time elapses, the test is successful.

If You Select Service

1. From the Service menu, select a service that triggers the event you are testing. For example, if you are testing the NewCustomer event, select a service that invokes it, such as Insert Customer.

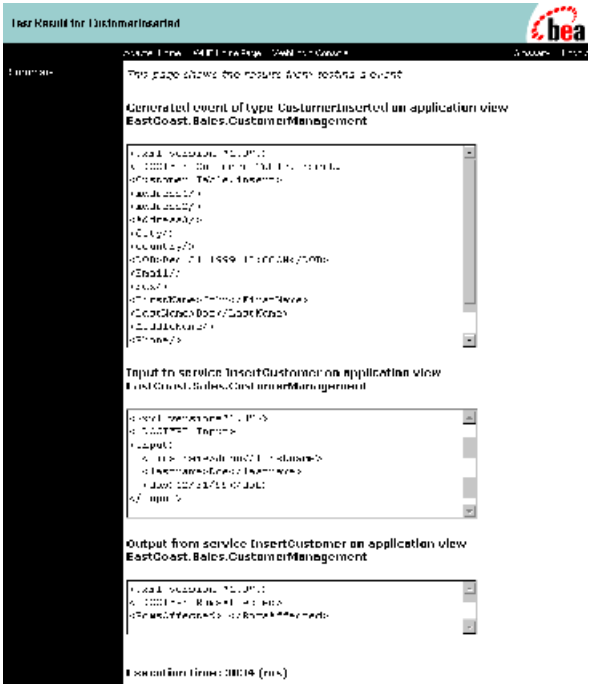
2. In the Time field, enter a reasonable period of time to wait, specified in milliseconds. (One minute = 60,000 milliseconds.) If the specified period elapses before the event succeeds, the test times out and a failure message is displayed.
3. Click Test. The triggering service is executed.

   If the service requires input data, an input page is displayed.

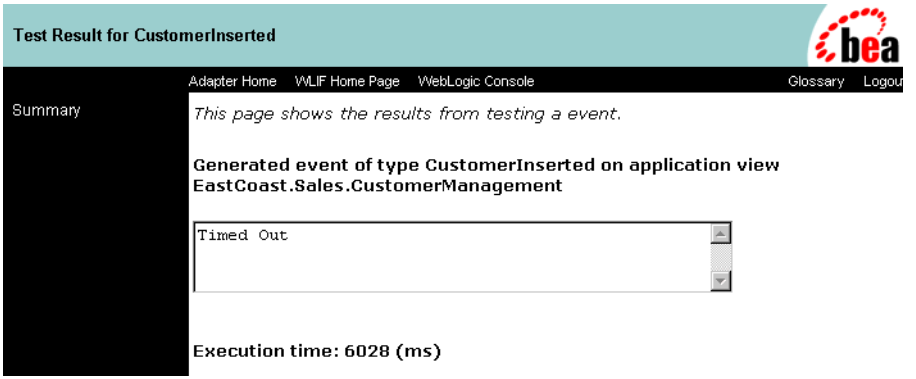


4. If service input data is required, enter it in the appropriate fields, and click Test.

   The service is executed. If the test succeeds, the Test Result page is displayed, showing the event document, the service input document, and the service output document.



If the test fails, the Test Result page displays only a Timed Out message.



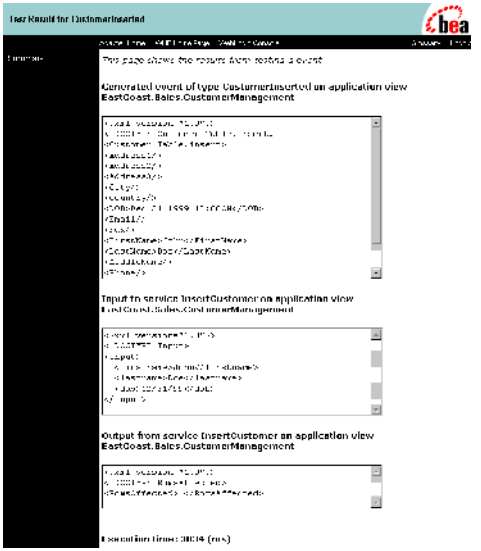
5. If the test fails, edit the event definition, or contact the system administrator or application manager.

6. If the test succeeds, repeat the test procedure for each remaining event you want to test.
7. When finished, save the Application View.

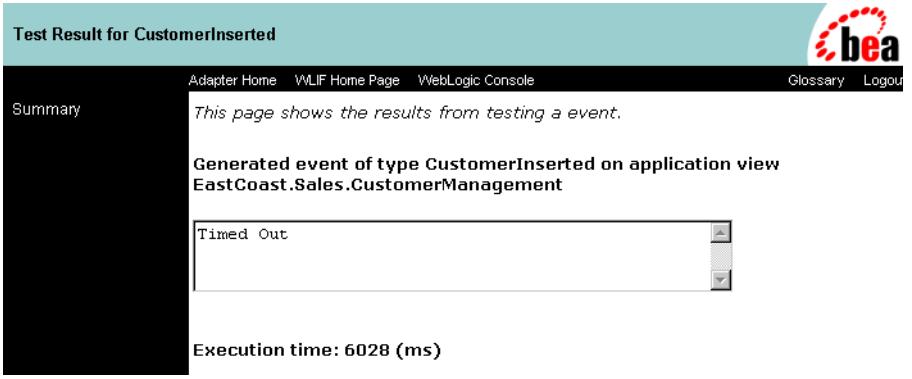
If You Select Manual

1. In the Time field, enter a reasonable time to wait, specified in milliseconds. (One minute = 60,000 milliseconds.) If this period elapses before the event succeeds, the test times out and a failure message is displayed.
2. If the application you will use to trigger the event is not already open, open it now.
3. Click Test. The test waits for an event to trigger it.
4. Using the triggering application, perform an action that executes the service that, in turn, tests the Application View event.

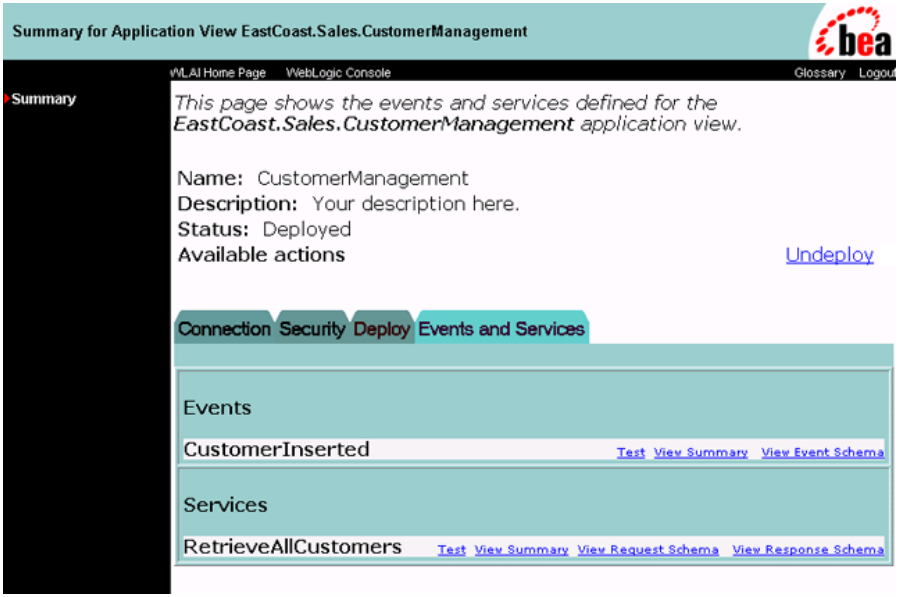
   If the test succeeds, the Test Result page is displayed. This page, in turn, displays the event document from the application, the service input document, and the service output document.



   If the test fails or takes too long, the Test Result page is displayed, showing a Timed Out message.



5. If the test fails, edit the event definition, or contact the system administrator or application manager.
6. If the test succeeds, repeat the test procedure for each remaining event you want to test.
7. When you are finished, save the Application View.

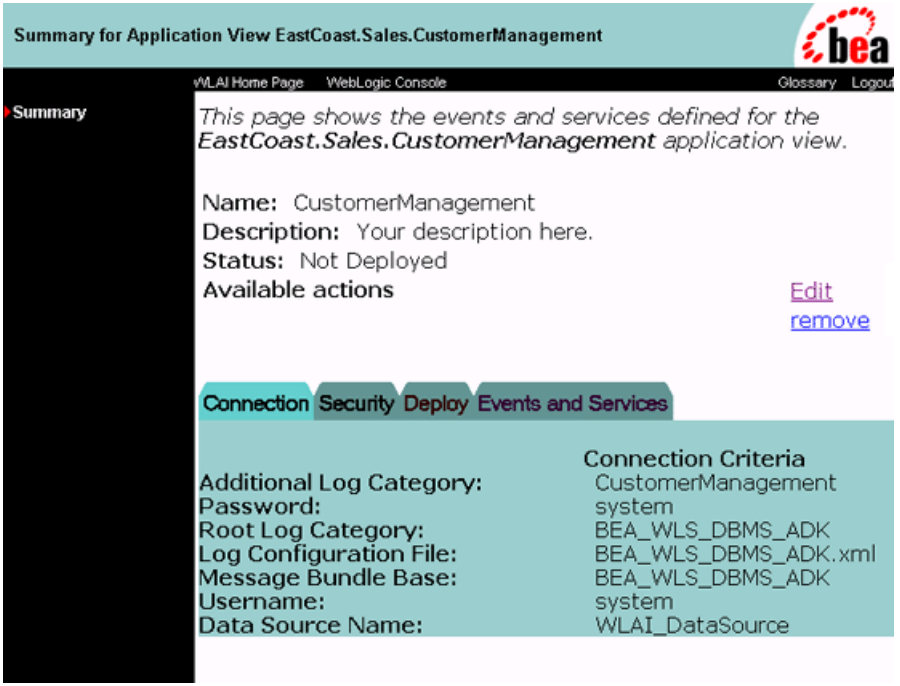


Editing an Application View

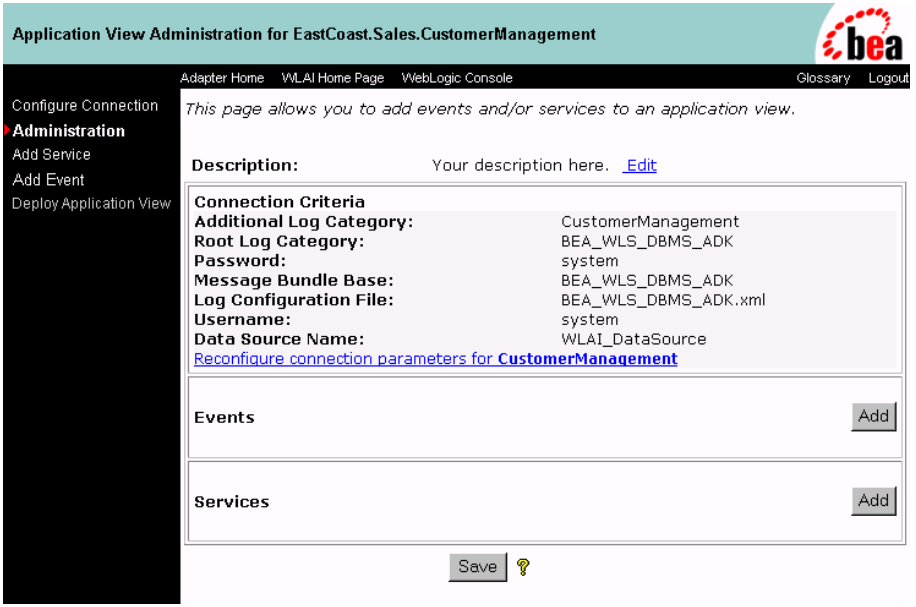
When you define an Application View, you must configure its connection parameters. After you add and test services and events, you may want to reconfigure the connection parameters or remove services and events.

To edit an existing Application View:

1. Open the Application View. The Summary for Application View page is displayed.



2. Click Edit. The Application View Administration page is displayed.



3. To reconfigure the Application View's connection parameters, select Configure Connection in the left pane. The Configuration Connection Parameters page is displayed. Follow the instructions beginning with step 6 in "Steps 2 and 3: Define an Application View and Configure Connection Parameters" on page 2-7.

4. To add services or events, click Add Service or Add Event, respectively. Follow the instructions in "Step 4A: Add a Service to an Application View" on page 2-11 or "Step 4B: Add an Event to an Application View" on page 2-13.

3 Using Application Views in the Studio

This section presents the following topics:

- Before You Begin
- Task 1: Set Up a Task Node to Call an Application View Service
- Task 2: Set Up an Event Node to Wait for a Response from an Asynchronous Application View Service
- Task 3: Create a Workflow Started by an Application View Event
- Task 4: Set Up an Event Node to Wait for an Application View Event
- Handling Application View Local Transactions in Workflows

Before You Begin

After you create all the Application View services and events that are required for your enterprise, you can use the resulting Application Views to execute your business processes. The simplest way to do this is by using the WebLogic Integration Studio to design business process workflows that use your Application View services and events.

The WebLogic Integration Studio is a graphical user interface (GUI) for designing business process workflows. These workflows can include Application View services and events defined using WebLogic Integration. For details, see *Using the WebLogic Integration Studio.*

Before you can invoke an Application View service or receive an Application View event in the WebLogic Integration Studio, you must make sure the following prerequisites have been met:

- You have created an Application View and defined services and events for it.
- The Application View and its adapter are functional and saved. If you plan to call Application View services and events from a running workflow, the Application View must be deployed, as well.
- BPM and application integration functionality is available.
- The application integration plug-in has been loaded.
- You have received information about the required business logic for the workflows you are defining from the appropriate business analyst.
- A workflow template definition is open.

Workflow Setup Tasks

The following sections describe four tasks you may perform to set up your workflow to use Application View services and events:

- Task 1: Set Up a Task Node to Call an Application View Service

- Task 2: Set Up an Event Node to Wait for a Response from an Asynchronous Application View Service
- Task 3: Create a Workflow Started by an Application View Event
- Task 4: Set Up an Event Node to Wait for an Application View Event

You can perform any combination of these tasks to create your own workflows.

This document does not fully explain how to use the business process management (BPM) functions provided by WebLogic Integration; for complete information, see *Learning to Use BPM with WebLogic Integration.*

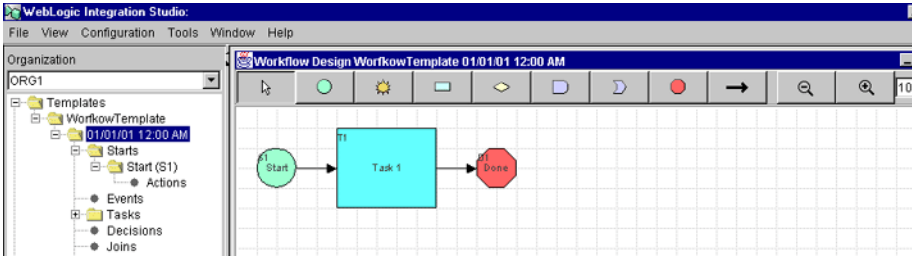
Task 1: Set Up a Task Node to Call an Application View Service

In your organization, there may be situations in which you want to call an Application View service from within a workflow. To make this type of call possible, add a task node to the workflow, then add an appropriate Application View Service action to the task node. Once the workflow is saved and activated, the Application View service is called whenever the task node is executed.

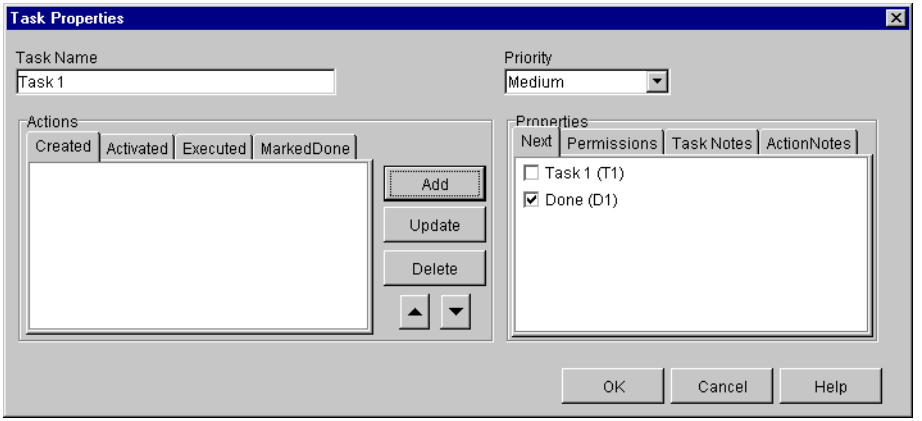
Steps for Setting up a Task Node to Call an Application View Service

To create a task node that calls an Application View service, complete the following procedure:

1. In the WebLogic Integration Studio, open a template definition. The Workflow Design window is displayed.

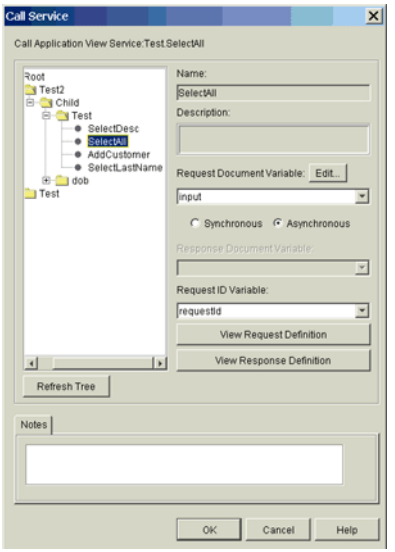


2. If there is no task node, create one.
3. Double-click the task node that calls the Application View service. The Task Properties dialog box is displayed.



4. In the Actions area, select the tab from which you want the service to be called. Your selection depends on your business processes.
5. Click Add. The Add Action dialog box is displayed.

6. Choose AI Actions→Call Application View Service and click OK. The Call Service dialog box is displayed.



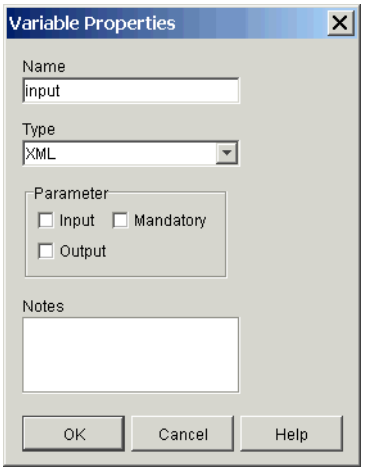
7. In the navigation tree, find and select the service you want to call.

   The navigation tree organizes Application View services by folder (for example, Test2) and Application View (for example, Test). All Application View services are at the lowest level of the navigation tree.

Note: To check for recently saved Application Views and events at any time, select the Refresh Tree option.

If the navigation tree is missing or appears too narrow, it may include an overly long XML or string variable name. Try shortening the names of your XML or string variables.

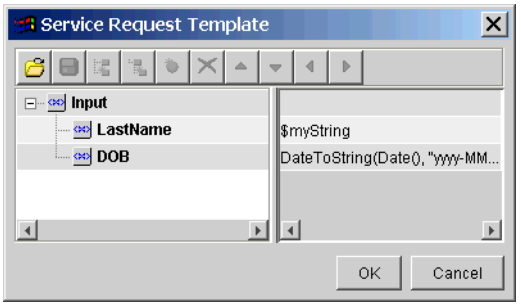
8. From the Request Document Variable list, select an existing XML variable that contains input data for the Application View service.
9. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, in which you can create a new XML variable.



10. In the Name field, enter a name for the variable.
11. In the Type menu, select XML. (XML is the only option on this menu.)

    For details about defining new variables, see *Using the WebLogic Integration Studio.*
12. Click OK to return to the Call Service dialog box.
13. To create a service request template for the specified service, or to modify an existing service request template, select one of the following:
    - Set—to create a template
    - Edit—to modify an existing template

Whichever option you select, the Service Request Template dialog box is displayed.

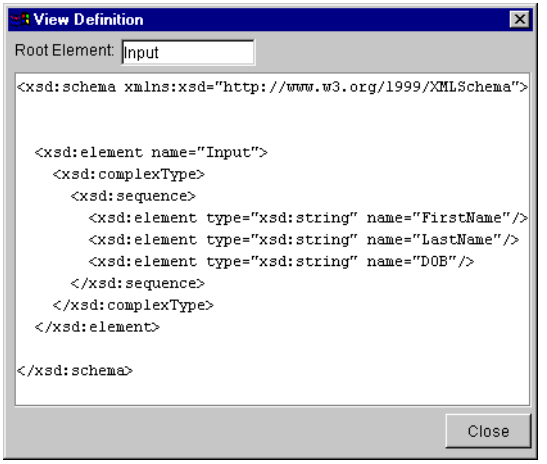


The Service Request Template dialog box displays the template to be used for all service requests of the type specified in step 11. This template is based on the input schema for the service.

When this action is executed, the template data is assigned to the specified request document variable and used as the input document for the service. This template overrides any previous setting for the variable. (If you set up a template and then want to use the previous input variable instead of the new input variable, the easiest way to do this is to delete the Call Application View Service and recreate it.)

For details about using the Service Request Template dialog box, see *Using the WebLogic Integration Studio.*

14. Click OK to return to the Call Service dialog box.
15. If you need to examine the XML schema of the input document, select View Request Definition. The View Definition dialog box is displayed.



16. Click Close when finished.

17. Call the Application View synchronously or asynchronously by selecting Synchronous or Asynchronous, respectively.

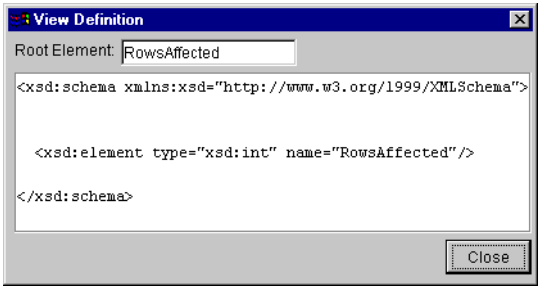
    Note: When a node calls a service synchronously, the workflow waits until the service returns a response document before continuing. If a node calls a service asynchronously, the workflow continues processing.

18. For synchronous services that require storage of the response, select a predefined XML variable from the Response Document Variable list. When the Studio receives a response from the Application View service, the response is stored in the response document variable. If you do not care about the response data, leave this field empty.

19. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, in which you can create a new XML variable. See step 9. in "Steps for Setting up a Task Node to Call an Application View Service" for details.

    For details about defining new variables, see *Using the WebLogic Integration Studio*.

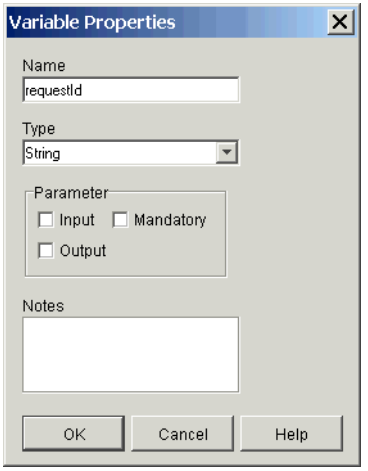
20. If you need to examine the XML schema of the response document, select View Response Definition. The View Definition dialog box is displayed.



21. Click Close when finished.

22. For asynchronous services that require storage of the request ID, select a predefined string variable from the Request ID Variable list.

23. If no suitable string variable exists, select <new> to open the Variable Properties dialog box, in which you can create a new string variable.



24. In the Name field, enter a name for the variable.

25. In the Type menu, select String. (String is the only option on this menu.)

    For details about defining new variables, see *Using the WebLogic Integration Studio.*

    Note: When you set up a task node to call an asynchronous Application View service, the result is returned to the Studio. The workflow identifies this response using the request ID variable you selected. To set up an event

node to receive the response, make sure to use the same request ID variable for the event node. For more information about creating such an event node, see "Task 2: Set Up an Event Node to Wait for a Response from an Asynchronous Application View Service" on page 3-10.

26. Click OK to save the action.

27. In the Task Properties dialog box, click OK to save the node.

Task 2: Set Up an Event Node to Wait for a Response from an Asynchronous Application View Service

This section explains how to receive an asynchronous Application View service response and handle any errors it may contain.

In a workflow, whenever an action calls an Application View service asynchronously, the Application View service returns a response. If you need the response, you must set up a corresponding asynchronous event node to wait for it. This section describes a highly simplified scenario in which an event node receives an Application View service response without checking for errors.

To set up an asynchronous event node to wait for a response from an asynchronous Application View service, create an event node and configure it to wait for an event of type AI Async Response.

Configuring Receipt of a Response

To set up an event node to receive an asynchronous service response, you can use either of the following methods:

- Select the Response Document tab (preferred method). Then select a request ID variable (a string) and a response document variable (of type XML). For details

about using this method, see "Procedure for Configuring Receipt of an Asynchronous Service Response (Preferred Method)" on page 3-12.

- Select the Asynchronous Variable tab (legacy method). Then select a request ID variable, an asynchronous service response variable (a string), and an asynchronous service response variable (of type AsyncServiceResponse). For details about using this method, see "Procedure for Configuring Receipt of an Asynchronous Service Response (Legacy Method)" on page 3-15.

  Note: Use of the Response Document tab is preferred because it provides a universal means of receiving both asynchronous and synchronous responses. When you use this method, an XML document is received regardless of whether the response is asynchronous or synchronous, and you do not need to query the value of the asynchronous service response variable.

We recommend using a response document variable to receive asynchronous service responses whenever possible. To configure a service to wait for an event of type AI Async Response, use the Event Properties dialog box. This dialog box may or may not offer you the choice of using an asynchronous variable to receive the response. Whether this choice is available depends on the following conditions:

- If you edit an existing AI Async Response event node that was previously set up to use an Asynchronous Service Response variable to receive the response, then two tabs are displayed in the Event Properties dialog box: an Asynchronous Variable tab (legacy method) and a Response Document tab (preferred method). Thus you have a choice of two methods you can use to configure receipt of the service response.
- If you edit an existing AI Async Response event node that does not use an Asynchronous Service Response variable, or if you are creating a new AI Async Response event node, then the Event Properties dialog box displays a dialog box without tabs. In this case you must set up a response document to receive the service response (preferred method).

Handling Errors in an Asynchronous Application View Service Response

Although this task does not include the configuration of error handling for the Application View service response, you may want to handle errors in your own workflows. To handle asynchronous service response errors in workflows that use an AsyncServiceResponse variable, you can use the features provided by the application integration plug-in.

The application integration plug-in includes the variable type AsyncServiceResponse and three functions:

- `AIHasError()`
- `AIGetErrorMsg()`
- `AIGetResponseDocument()`

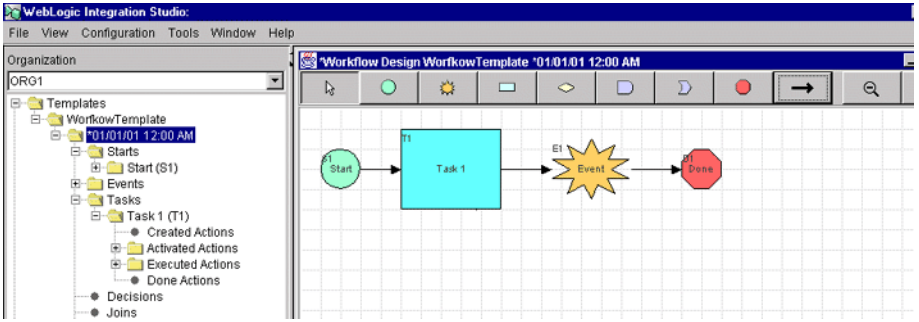
For a complete description of these functions, see "Functions Provided by the Application Integration Plug-In" on page 3-17.

Procedure for Configuring Receipt of an Asynchronous Service Response (Preferred Method)

To set up an asynchronous event node to wait for a response from an asynchronous Application View service, create an event node and configure it to wait for an event of type AI Async Response.

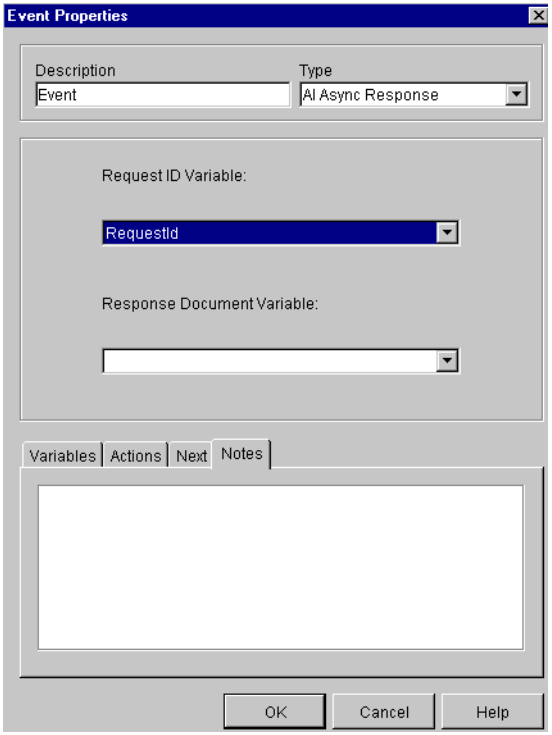
To set up an event node to use an XML variable to receive an asynchronous service response, complete the following procedure:

1. In the WebLogic Integration Studio, open a workflow template definition. The workflow design window is displayed.



2. If no event node exists, create one now. This event node will wait for an asynchronous response from a designated Application View service.

3. Double-click the event node. The Event Properties dialog box is displayed.



4. (Optional) In the Description field, enter a name.

5. In the Type list, select AI Async Response.
6. Select the Response Document (preferred) tab.

   Note: If your workflow does not use an AsyncServiceResponse variable, or if you are creating a new AI Async Response event node, then the Event Properties dialog box displays a dialog box without tabs, instead. Use this dialog box to set up a response document to receive the service response. (This method is the recommended.)

7. In the Request ID Variable list, select a string variable that is already defined. The WebLogic Integration process engine listens for an asynchronous response with an ID matching the ID stored in this variable.
8. If no suitable string variable exists, select <new> to open the Variable Properties dialog box, in which you can create a new string variable. See step 23. in "Steps for Setting up a Task Node to Call an Application View Service" for details.

   For details about defining new variables, see *Using the WebLogic Integration Studio.*

   Note: The purpose of this event node is to wait for a response to a Call Application View Service action that was called asynchronously earlier in the workflow. The Call Application View Service action sets the request ID variable. The action and this event node can work together only if both use the same request ID variable. For more information about setting up the Call Application View Service action, see "Task 1: Set Up a Task Node to Call an Application View Service" on page 3-3.

9. For asynchronous services that require storage of the response, select a predefined XML variable in the Response Document Variable list. Subsequently, whenever WebLogic Integration receives a response from the Application View service, the response is stored in the response document variable. If, on the other hand, you need the response data, skip this step.
10. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, in which you can create a new variable. See step 9. in "Steps for Setting up a Task Node to Call an Application View Service" for details.

    For details about defining new variables, see *Using the WebLogic Integration Studio.*

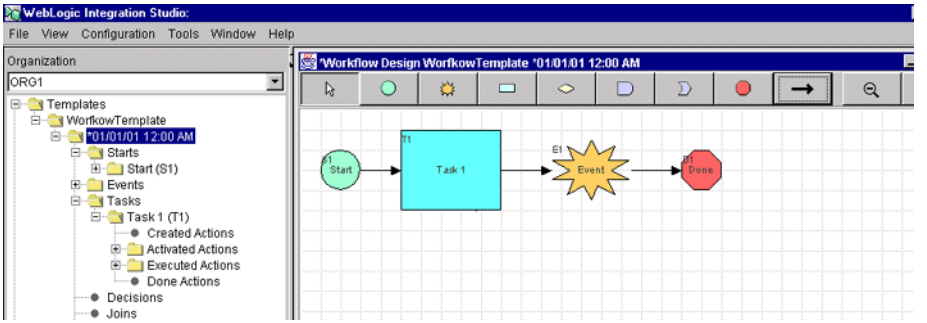
11. Click OK to save the event node.

Procedure for Configuring Receipt of an Asynchronous Service Response (Legacy Method)

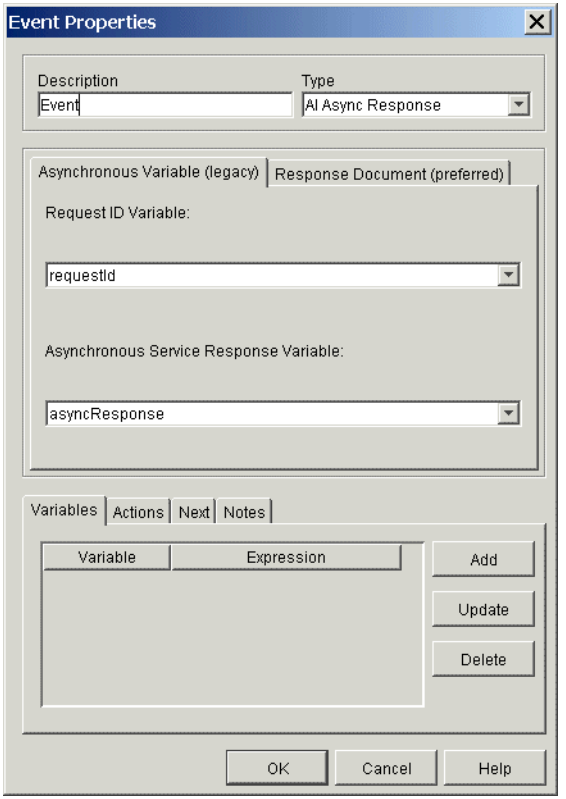
The preferred method for receiving an asynchronous service response is to use a response document variable of type XML. However, if an existing workflow contains an asynchronous event node that was previously set up to use an AsyncServiceResponse variable (to wait for a response from an asynchronous Application View service), you can modify the event node.

To modify an event node that uses an AsyncServiceResponse variable to receive an asynchronous service response, complete the following procedure:

1. In the WebLogic Integration Studio, open a workflow template definition. The Workflow Design window is displayed.



2. Double-click the asynchronous event node. The Event Properties dialog box is displayed.



3. Select the Asynchronous Variable (legacy) tab.
4. From the Request ID Variable list, select a string variable that is already defined. WebLogic Integration listens for an asynchronous response with an ID matching the ID stored in this variable.

   Note: The purpose of this event node is to wait for a response to a Call Application View Service action that was invoked asynchronously earlier in the workflow. The Call Application View Service action sets the request ID variable. The action and this event node can work together only if both use the same request ID variable. For more information about setting up the Call Application View Service action, see "Task 1: Set Up a Task Node to Call an Application View Service" on page 3-3.

5. From the Asynchronous Service Response Variable list, select an AsyncServiceResponse variable in which to store the response data.

   Note: Because you are modifying an existing asynchronous event node, the asynchronous service response variable field is already populated. If you do not need the response, select the Response Document (preferred) tab. For details about using the preferred method, see "Procedure for Configuring Receipt of an Asynchronous Service Response (Preferred Method)" on page 3-12.

6. Click OK to save the event node.

Functions Provided by the Application Integration Plug-In

If your enterprise includes AI Async Response variables and you want to interrogate those variables while using the application integration plug-in, use the following functions:

- `AIHasError()`
- `AIGetErrorMsg()`
- `AIGetResponseDocument()`

Using these functions, you can set up decision nodes to handle success and failure conditions.

Note: These functions are available only if the application integration plug-in is installed, and they support only the asynchronous variable method for receiving asynchronous service responses. For details, see "Procedure for Configuring Receipt of an Asynchronous Service Response (Legacy Method)" on page 3-15.

AIHasError()

Use `AIHasError()` to determine the status of an asynchronous service response. The following table provides details about this function.

Operands	AsyncServiceResponse variable
Preconditions	■ You have created a variable of type AsyncServiceResponse. ■ You have called an asynchronous Application View service. ■ The Application View service has returned a response, which is stored in your AsyncServiceResponse variable.
Returns	Boolean
Output explanation	False: The asynchronous Application View service call was successful. True: The asynchronous Application View service call failed.

AIGetErrorMsg()

Use `AIGetErrorMsg()` to retrieve the error message string returned by an asynchronous Application View service. The following table provides details about this function.

Operands	AsyncServiceResponse variable
Preconditions	■ You have created a variable of type AsyncServiceResponse. ■ You have called an asynchronous Application View service. ■ The Application View service has returned a response, which is stored by your AsyncServiceResponse variable.
Returns	String
Output explanation	Error string: Returns an error string explaining why the asynchronous Application View response failed. Empty string: There was no error.

AIGetResponseDocument()

Use `AIGetResponseDocument()` to retrieve the XML response document returned by an asynchronous Application View service. The following table provides details about this function.

Operands	AsyncServiceResponse variable
Preconditions	■ You have created a variable of type AsyncServiceResponse. ■ You have called an asynchronous Application View service. ■ The Application View service has returned a response, which is stored in your AsyncServiceResponse variable.
Returns	XML
Output explanation	XML document: Returns an XML document representing the asynchronous service response. Null: No response document is returned because an error occurred.

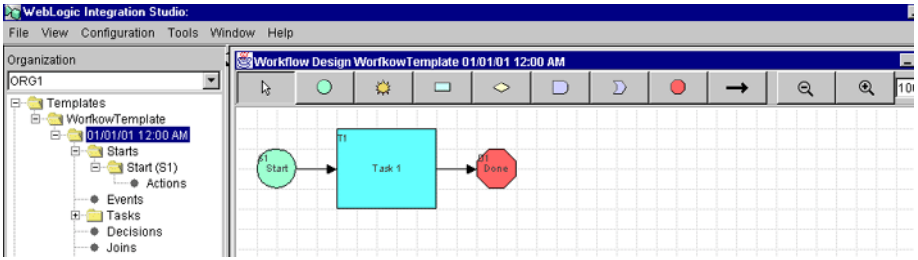
Task 3: Create a Workflow Started by an Application View Event

It is sometimes desirable to have a workflow that is started whenever a designated Application View event occurs. To create such a workflow, edit the workflow's start node so it responds to an event of type AI Start, then select the appropriate Application View event. If necessary, you can set up conditions with which the event can be filtered. After you save and activate the workflow, the start node is executed each time the Application View event occurs.

Steps for Creating a Workflow Started by an Application View Event

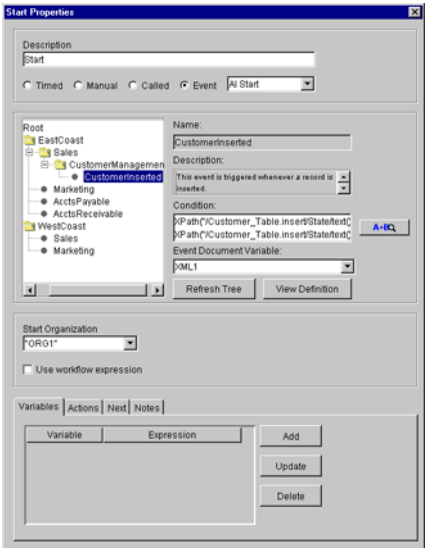
To set up a workflow with a start node that is triggered by an Application View event, complete the following procedure:

1. In the WebLogic Integration Studio, open a template definition. The Workflow Design window is displayed.



2. If no start node exists, create one now. This start node will respond to an Application View event that you specify.

3. Double-click the start node. The Start Properties dialog box is displayed.



4. (Optional) In the Description field, enter a name.
5. Click Event.
6. From the Event list, select AI Start.
7. In the navigation tree, select an Application View event.

   The navigation tree organizes Application View events by folder (such as the EastCoast and Sales folders shown in the preceding Start Properties dialog box) and Application View (for example, CustomerManagement). All Application View events are listed at the lowest level of the hierarchy.

   Note: To check for recently saved Application Views and events at any time, select Refresh Tree.

   If the navigation tree is missing or appears too narrow, it may include an overly long XML or string variable name. Try shortening the names of your XML or string variables.

8. In the Key Value Expression field, define a key value for the event. You can enter a String as the key value, or you can create an expression that is evaluated at run time to produce the key value. (Select the A + B option to display the Expression

Builder dialog box, and then create an appropriate expression.) The specified key value must match an element in the incoming Event's XML before WebLogic Integration triggers the event.

You must also define the expression that locates the key value in the XML for the incoming Event, so that WebLogic Integration can compare the incoming key value to the specified key value. If there is no key value expression, you are prompted to create one before you can save the event.

The Event Descriptor for AI Events is the fully qualified Application View event name in the following form:

`namespace`.Application View `name.event`

`Root` is not part of the fully qualified event name. The Event Descriptor field is filled in automatically if the key value expression is created directly from the event dialog box. To accept each instance of the specified Application View event regardless of its contents, leave this field blank.

9. If necessary, filter the event in one of the following ways: enter a condition in the Condition field, or select the A + B option to display the Expression Builder dialog box, and then create an appropriate expression.

   For information about setting up conditions and XPath expressions, see *Using the WebLogic Integration Studio.*

10. From the Event Document Variable list, select an XML variable. Data from the Application View event that is received by the start node is stored in this variable. If you do not need the event data, skip this step.

11. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, in which you can create a new variable. See step 9. in "Steps for Setting up a Task Node to Call an Application View Service" for details.

    For details about defining new variables, see *Using the WebLogic Integration Studio.*

12. If you need to examine the XML schema for the event document, click View Definition. The View Definition dialog box is displayed.

13. Click Close to return to the Start Properties dialog box.

14. In the Start Properties dialog box, click OK. The new or modified start node is saved.

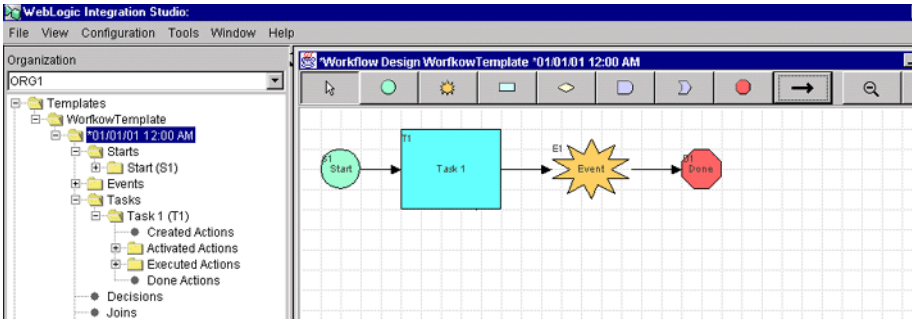
Task 4: Set Up an Event Node to Wait for an Application View Event

In a workflow, it is sometimes desirable to have an event node that is triggered by an Application View event. To create such a node, edit an event node so it responds to an event of type AI Event, then select the appropriate Application View event. If necessary, you can set up conditions with which to filter the designated Application View event. After you save and activate the workflow, the workflow progresses to this event node, waits for a specified Application View event, and continues processing.

Steps for Setting Up a Node to Wait for an Application View Event

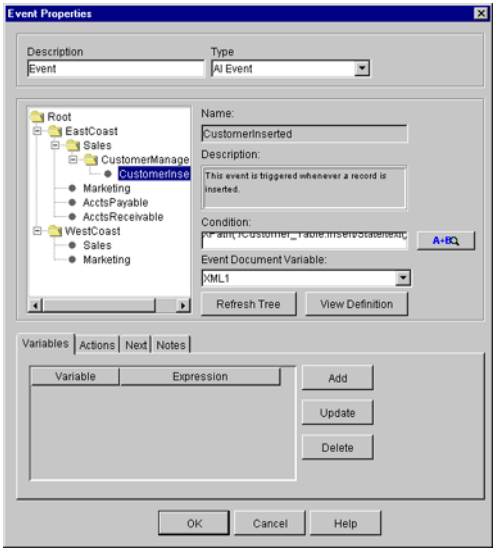
To set up an event node to be triggered by an Application View event, complete the following procedure:

1. In the WebLogic Integration Studio, open a template definition. The Workflow Design window is displayed.



2. If no event node exists, create one now. This event node will be triggered by a designated Application View event.

3. Double-click the event node. The Event Properties dialog box is displayed.



4. (Optional) In the Description field, enter a name.
5. From the Type list, select AI Event.
6. In the navigation tree, select an Application View event.

   The navigation tree organizes Application View events by folder (such as the EastCoast and Sales folders shown in the preceding Start Properties dialog box) and Application View (for example, CustomerManagement). All Application View events are listed at the lowest level of the hierarchy.

   Note: To check for recently saved Application Views and events at any time, select Refresh Tree.

   If the navigation tree is missing or appears too narrow, it may include an overly long XML or string variable name. Try shortening the names of your XML or string variables.

7. In the Key Value Expression field, define a key value for the event. You can enter a variable or a String as the key value, or you can create an expression that is evaluated at run time to produce the key value. (Select the A + B option to display the Expression Builder dialog box, and then create an appropriate expression.) The specified key value must match an element in the incoming Event's XML before WebLogic Integration triggers the event.

You must also define the expression that locates the key value in the XML for the incoming Event, so that WebLogic Integration can compare the incoming key value to the specified key value. If there is no key value expression, you are prompted to create one before you can save the event.

The Event Descriptor for AI Events is the fully qualified Application View event name in the following form:

namespace.Application View *name.event*

Root is not part of the fully qualified event name. The Event Descriptor field is filled in automatically if the key value expression is created directly from the event dialog box. To accept each instance of the specified Application View event regardless of its contents, leave this field blank.

8. If necessary, filter the event in one of the following ways: enter a condition in the Condition field, or select the A + B option to display the Expression Builder dialog box, and then create an appropriate expression.

   For information about setting up conditions and XPath expressions, see *Using the WebLogic Integration Studio.*

9. In the Event Properties dialog box, select an XML variable from the Event Document Variable list. Data from the Application View event that is received by the start node is stored in this variable. If you do not need to save the event data, skip this step.

10. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, in which you can create a new variable. See step 9. in "Steps for Setting up a Task Node to Call an Application View Service" for details.

    For details about defining new variables, see *Using the WebLogic Integration Studio.*

11. If you need to examine the XML schema for the event document, click View Definition. The View Definition dialog box is displayed.

12. Click Close when finished.

13. In the Event Properties dialog box, click OK.

Handling Application View Local Transactions in Workflows

The LocalTransaction interface is exposed to adapter clients via the Common Client Interface (CCI) Connection class. Currently the Application View interface does not use the CCI LocalTransaction interface. To manage a local transaction, a user must first acquire a LocalTransaction from the Connection object.

Local Transaction Management Contracts

A local transaction management contract is created when an adapter implements the `javax.resource.spi.LocalTransaction` interface to provide support for local transactions that are performed on the system's underlying resource manager. This type of contract enables an application server to provide the infrastructure and run-time environment for transaction management. Application components rely on this transaction infrastructure to support their component-level transaction model.

For more information about transaction demarcation support, see:

`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/transaction_management/platform/index.html`

Connector Support for Local Transactions with No User Defined Transaction Demarcation

The following is a scenario for supporting Application View local transactions within the Application Integration Plug-in. This scenario is similar to `TX_REQUIRES_NEW` for EJB transactions because the connector supports only local transactions.

In this scenario, the Connector supports only local transactions and the BPM designer does not explicitly demarcate the start and end of a local transaction. WebLogic Integration allows the Connector to participate in the global transaction by providing an XA Wrapper around the LocalTransaction object. The XA Wrapper no-ops all methods on the XAResource interface that can not be delegated to the LocalTransaction instance. WebLogic Integration allows only one non XA resource in the transaction chain. As a result, a user can have only one Application View LocalTransaction within a workflow.

Connector Support for XA Transactions

In this scenario, Application View services are not called within a local transaction. Each service invocation is automatically enlisted in the Global XA transaction because the resource adapter supports XA.

4 Using Application Views by Writing Custom Code

If you are a developer, you may want to modify an Application View by writing custom code. You can use most Application View features through the Application View Console, but some features can be used only by writing custom code.

This section presents two sample scenarios in which custom code is used:

- Scenario 1: Creating Connections with Specific Credentials
- Scenario 2: Custom Coding a Business Process

Scenario 1: Creating Connections with Specific Credentials

If you need to assign a security level to an Application View before invoking services on it, you can do so by setting credentials for the appropriate EIS. To do so, use the `ApplicationView` methods `setConnectionSpec()` and `getConnectionSpec()`. Both methods use a `ConnectionSpec` object.

You can instantiate a `ConnectionSpec` object in either of two ways: you can use the `ConnectionRequestInfoMap` class provided by the BEA WebLogic Integration Adapter Development Kit (ADK), or you can implement your own class. If you implement your own class, you must include the following four interfaces: `ConnectionSpec`, `ConnectionRequestInfo`, `Map`, and `Serializable`.

Implementing ConnectionSpec

Before you can use `setConnectionSpec()` or `getConnectionSpec()`, you must instantiate a `ConnectionSpec` object. Use the `ConnectionRequestInfoMap` class provided by the ADK, or derive your own class.

To implement `ConnectionSpec`:

1. Decide whether to use the `ConnectionRequestInfoMap` class, provided by the ADK, or to implement your own class.
2. If you are implementing your own `ConnectionSpec` class, include the following interfaces:
   - `ConnectionSpec` (JCA class)
   - `ConnectionRequestInfo` (JCA class)
   - `Map` (SDK class)
   - `Serializable` (SDK class)

Calling setConnectionSpec() and getConnectionSpec()

After you implement the `ConnectionSpec` class and instantiate a `ConnectionSpec` object, you can use both with the following `ApplicationView` methods:

- `setConnectionSpec()`
- `getConnectionSpec()`

The following listing provides the code for `setConnectionSpec()`.

Listing 4-1 Complete Code for setConnectionSpec()

```
/**
* Sets the connectionSpec for connections made to the EIS. After the
* ConnectionSpec is set it will be used to make connections to the
* EIS when invoking a service. To clear the connection spec, and use
* the default connection parameters, call this method using null.
*
* @params connectionCriteria connection criteria for the EIS.
*/
public void setConnectionSpec(ConnectionSpec connectionCriteria)
{
m_connCriteria = connectionCriteria;
}
```

The following listing provides the code for `getConnectionSpec()`.

Listing 4-2 Complete Code for getConnectionSpec()

```
/**
* Returns the ConnectionSpec set by setConnectionSpec. If no
* ConnectionSpec has been set null is returned.
*
* @returns ConnectionSpec
*/
public ConnectionSpec getConnectionSpec()
{
return m_connCriteria;
}
```

Using the ConnectionSpec Class

To set the `ConnectionSpec` class, pass it a properly initialized `ConnectionSpec` object. To clear the `ConnectionSpec` class, pass it a `ConnectionSpec` object with a null value.

Listing 4-3 shows an example of how `ConnectionSpec` is used.

Listing 4-3 Example Use of ConnectionSpec Class

```
Properties props = new Properties();
ApplicationView applicationView = new
ApplicationView(getInitialContext(props),"appViewTestSend");

ConnectionRequestInfoMap map = new ConnectionRequestInfoMap();
// map properties here
map.put("PropertyOne","valueOne");
map.put("PropertyTwo","valueTwo");
.
.
.
//set new connection spec
applicationView.setConnectionSpec(map);

IDocumentDefinition requestDocumentDef =
applicationView.getRequestDocumentDefinition("serviceName");

SOMSchema requestSchema = requestDocumentDef.getDocumentSchema();

DefaultDocumentOptions options = new DefaultDocumentOptions();
options.setForceMinOccurs(1);
options.setRootName("ROOTNAME");
options.setTargetDocument(DocumentFactory.createDocument());
IDocument requestDocument = requestSchema.createDefaultDocument(options);

requestDocument.setStringInFirst("//ROOT/ElementOne","value");
requestDocument.setStringInFirst("//ROOT/ElementTwo","value");
.
.
.
// the service invocation will use the connection spec set to connect to the EIS
IDocument result = applicationView.invokeService("serviceName",
requestDocument);
System.out.println(result.toXML());
```

Scenario 2: Custom Coding a Business Process

Although the simplest way of using Application Views in business processes is through the WebLogic Integration Studio, you always have the alternative of writing custom Java code to represent your business processes. If you are a developer who writes custom code, we recommend that you familiarize yourself with the simple example presented in this section to demonstrate how a custom business process can be written.

For a thorough comparison of the two methods for using Application Views, see "Choosing a Method for Implementing a Business Process" on page 1-8.

About This Scenario

Suppose your company uses a customer relationship management (CRM) system and an order processing (OP) system. Management wants to make sure that whenever a customer is created on the CRM system, the creation of a corresponding customer record on the OP system is triggered. Therefore, they ask you, their Java developer, to create a business process that keeps the information maintained by these two systems synchronized. The attached Java class, `SyncCustomerInformation`, implements this business logic.

This example does not cover everything you can do using custom code. It simply demonstrates the basic steps required to implement your organization's business processes and serves as a template you can use for custom coding your own business processes.

This scenario uses a concrete example class called `SyncCustomerInformation` to explain how to write custom code. In general, you must perform the following two steps to create custom code that uses an Application View in a business process:

1. Make sure you have a Java class representing the application that implements the business process.
2. Within this Java class, supply code that implements your business logic.

Before You Begin

The following prerequisites must be met before you start writing custom code to implement a business process:

- Create an Application View and define one or more events or services within the Application View.
- Obtain information, from the appropriate business analyst, about the required business logic for the business process workflow you are defining. Make sure you also get all the information needed to connect to WebLogic Server, including the host server name and port number, and a user ID and password.

In addition, this scenario is based on the assumption that the following prerequisites have been met:

- Application views for the source CRM system and the target OP system are defined and working. For details about defining Application Views, see "Defining an Application View" on page 2-1.
- Both Application Views reside in the East Coast folder. The source Application View is named East Coast.Customer Mgmt and the target Application View is named East Coast.Order Processing.

  Note: Your organization must have its own folders and Application Views.
- You are familiar with the application integration API, or you are working closely with a Java programmer who is familiar with it.
- You have all the information necessary to connect to the application integration server that hosts the Application Views.

  Note: Get the information specific to your organization from your system administrator.

Creating the SyncCustomerInformation Class

Before you can start writing custom code, you must have a Java class representing each application required for the business process. If the necessary Java classes do not exist, create them now. This example calls for one application class called `SyncCustomerInformation`. Of course, you will use different variable names in your own code. To create the `SyncCustomerInformation` Java class:

1. See "Code for Sample Java Class" on page 4-9 for the complete source code for the Java application class.

   Note: For your own projects, use the `SyncCustomerInformation` code as a template or guide. The `SyncCustomerInformation` example code is annotated with detailed comments.

2. Create code to listen for East Coast.New Customer.

3. Obtain references to the `NamespaceManager` (variable name `m_namespaceMgr`) and `ApplicationViewManager` (variable name `m_appViewMgr`) within WebLogic Server. To perform this step, use a JNDI lookup from WebLogic Server.

4. Using the `NamespaceManager` to call `nm.getRootNamespace()`, obtain a reference to the root namespace. This reference is stored in a variable called `root`.

5. Using the `root` variable to call `root.getNamespaceObject("East Coast")`, obtain a reference to the East Coast namespace. This reference is stored in a variable called `eastCoast`.

6. Using the `eastCoast` variable, obtain a temporary reference to the Customer Management `ApplicationView` and store it in a variable called `custMgmtHandle`.

7. Use this `custMgmtHandle` temporary reference to obtain a reference to an `ApplicationView` instance for Customer Management. Specifically, call the `ApplicationViewManager` as `avm.getApplicationViewInstance (custMgmtHandle.getQualifiedName())`. Store the returned reference in a variable called `custMgmt`.

8. Begin listening for New Customer events by calling `custMgmt.addEventListener("New Customer", listener)`, replacing *`listener`* with the name of an object that can respond to New Customer events. (See the application integration API for a full discussion of event listeners and the `EventListener` interface.)

9. Implement the `onEvent` method of the listener class.

   When a New Customer event is received, the `onEvent` method of the listener is called.

   The `onEvent` method calls a method to respond to the event. In this example, the `onEvent` method provides the event object that contains the data associated with the event. The method called to respond to the event is called `handleNewCustomer`.

10. Implement the `handleNewCustomer` method that will respond to the New Customer event. Specifically, write code that implements the following sequence of actions:

    a. The `handleNewCustomer` method transforms the XML document referenced in the event to the form expected by the East Coast.Order Processing.Create Customer service. This transformation may be performed using XSLT or manually, using custom transformation code. The end result of the transformation is an XML document that conforms to the schema for the request document of the East Coast.Order Processing.Create Customer service. Store this document in a variable called `createCustomerRequest`.

    b. `handleNewCustomer` obtains a reference to an instance of the East Coast.Order Processing Application View in the same way described for the East Coast.Customer Management Application View. This reference is stored in a variable called `orderProc`.

    c. `handleNewCustomer` invokes the Create Customer service on the East Coast.Order Processing Application View by calling `orderProc.invokeService("Create Customer", createCustomerRequest)`. Recall that `createCustomerRequest` is the variable holding the request document for the Create Customer service. The response document for this service is stored in a variable named `createCustomerResponse`.

    d. `handleNewCustomer` finishes executing and becomes available for the next incoming New Customer event.

Once you complete this final step, you have a new Java class called `SyncCustomerInformation`. This class implements the Sync Customer Information business logic. The `SyncCustomerInformation` class uses the application integration API to get events from the CRM system and to invoke services on the OP system.

Code for Sample Java Class

The following listing contains the full source code for the `SyncCustomerInformation` Java class. This code implements the business logic for the scenario described earlier in this section. Use it as a template for writing code to implement your enterprise's business processes.

Listing 4-4 Full Class Source Code for SyncCustomerInformation

```
import java.util.Hashtable;
import javax.naming.*;
import java.rmi.RemoteException;
import com.bea.wlai.client.*;
import com.bea.wlai.common.*;
import com.bea.document.*;

/**
 * This class implements the business logic for the 'Sync Customer Information'
 * business process. It uses the WLAI API to listen to events from the CRM
 * system, and to invoke services on the OP system. It assumes that there
 * are two ApplicationViews defined and deployed in the 'EastCoast'
 * namespace. The Application Views and their required events and services
 * are shown below.
 *
 * CustomerManagement
 *   events (NewCustomer)
 *   services (none)
 *
 * OrderProcessing
 *   events (none)
 *   services (CreateCustomer)
 */

public class SyncCustomerInformation
  implements EventListener
{
```

```
/**
 * Main method to start this application. No args are required.
 */
public static void
main(String[] args)
{
  // Check that we have the information needed to connect to the server.

  if (args.length != 3)
  {
    System.out.println("Usage: SyncCustomerInformation ");
    System.out.println("       <server url> <user id> <password>");
    return;
  }

  try
  {
    // Create an instance of SyncCustomerInformation to work with

    SyncCustomerInformation syncCustInfo =
      new SyncCustomerInformation(args[0], args[1], args[2]);

    // Get a connection to WLAI

    InitialContext initialContext = syncCustInfo.getInitialContext();

    // Get a reference to an instance of the 'EastCoast.CustomerManagement'
    // Application View

    ApplicationView custMgmt =
      new ApplicationView(initialContext, "EastCoast.CustomerManagement");

    // Add the listener for 'New Customer' events. In this case we have
    // our application class implement EventListener so it can listen for
    // events directly.

    custMgmt.addEventListener("NewCustomer", syncCustInfo);

    // Process up to 10 events and then quit.

    syncCustInfo.setMaxEventCount(10);
    syncCustInfo.processEvents();
  }
  catch (Exception e)
  {
    e.printStackTrace();
  }

  return;
```

```
}

/**
 * EventListener method to respond to 'New Customer' events
 */
public void
onEvent(IEvent newCustomerEvent)
{
  try
  {
    // Print the contents of the incoming 'New Customer' event.

    System.out.println("Handling new customer: ");
    System.out.println(newCustomerEvent.toXML());

    // Handle it

    IDocument response = handleNewCustomer(newCustomerEvent.getPayload());

    // Print the response

    System.out.println("Response: ");
    System.out.println(response.toXML());

    // If we have processed all the events we want to, quit.

    m_eventCount++;
    if (m_eventCount >= m_maxEventCount)
    {
      quit();
    }
  }
  catch (Exception e)
  {
    e.printStackTrace();
    System.out.println("Quitting...");
    quit();
  }
}

/**
 * Handles any 'New Customer' event by invoking the 'Create Customer'
 * service on the 'Order Processing' ApplicationView. The response
 * document from the service is returned as the return value of this
 * method.
 */
public IDocument
handleNewCustomer(IDocument newCustomerData)
  throws Exception
```

```
{
  // Get an instance of the 'OrderProcessing' ApplicationView.
  if (m_orderProc == null)
  {
    m_orderProc =
      new ApplicationView(m_initialContext, "EastCoast.OrderProcessing");
  }

  // Transform the data in newCustomerData to be appropriate for the
  // request document for 'Create Customer' on the 'Order Processing'
  // ApplicationView.

  IDocument createCustomerRequest =
    transformNewCustomerToCreateCustomerRequest(newCustomerData);

  // Invoke the service

  IDocument createCustomerResponse =
    m_orderProc.invokeService("CreateCustomer", createCustomerRequest);

  // Return the response

  return createCustomerResponse;
}

// ------------------------------------------
// Member Variables
// ------------------------------------------

/**
 * The url for the WLAI server (e.g. t3://localhost:7001)
 */
private String m_url;

/**
 * The user id to use when logging into WLAI.
 */
private String m_userID;

/**
 * The password to use when logging in to WLAI as the user given in
 * m_userID.
 */
private String m_password;


/**
 * The initial context to use when communicating with WLAI
 */
```

```
private InitialContext m_initialContext;

/**
 * An instance of the 'East Coast.Order Processing' ApplicationView for
 * use in handleNewCustomer.
 */
private ApplicationView m_orderProc;

/**
 * Hold the maximum number of events to be processed in handleNewCustomer
 */
private int m_maxEventCount;

/**
 * Count of the events processed in handleNewCustomer
 */
private int m_eventCount;

/**
 * A monitor variable to enable us to wait until we are asked to quit
 */
private String m_doneMonitor = new String("Done Monitor");

/**
 * A flag indicating we are done or not.
 */
private boolean m_done = false;

// ------------------------------------------------
// Utility Methods
// ------------------------------------------------

/**
 * Constructor.
 */
public SyncCustomerInformation(String url, String userID, String password)
{
  m_url = url;
  m_userID = userID;
  m_password = password;
}

/**
 * Establish an initial context to WLAI.
 */
public InitialContext
getInitialContext()
  throws NamingException
{
```

```
  // Set up properties for obtaining an InitialContext to the WLAI server.

  Hashtable props = new Hashtable();

  // Fill in the properties with the WLAI host, port, user id, and password.

  props.put(Context.INITIAL_CONTEXT_FACTORY,
            "weblogic.jndi.WLInitialContextFactory");
  props.put(Context.PROVIDER_URL, m_url);
  props.put(Context.SECURITY_PRINCIPAL, m_userID);
  props.put(Context.SECURITY_CREDENTIALS, m_password);

  // Connect to the WLAI server

  InitialContext initialContext = new InitialContext(props);

  // Store this for later

  m_initialContext = initialContext;

  return initialContext;
}


/**
 * Transform the document in the 'New Customer' event to the document
 * required by the 'Create Customer' service.
 */
public IDocument
transformNewCustomerToCreateCustomerRequest(IDocument newCustomerData)
  throws Exception
{
  // We could do an XSLT transform here, or manually move data from the
  // source to the target document. The details of this transformation
  // are out of the scope of this sample. For information on XSLT see
  // http://www.w3.org/TR/xslt. For more information on manually moving
  // data between documents, see the JavaDoc documentation for the
  // com.bea.document.IDocument interface.

  return newCustomerData;
}

/**
 * Event processing/wait loop
 */
public void
processEvents()
{
  synchronized(m_doneMonitor)
```

```
    {
      while (!m_done)
      {
        try
        {
          m_doneMonitor.wait();
        }
        catch (Exception e)
        {
          // ignore
        }
      }
    }
  }

  /**
   * Sets the max number of events we want to process.
   */
  public void
  setMaxEventCount(int maxEventCount)
  {
    m_maxEventCount = maxEventCount;
  }

  /**
   * Method to force this application to exit (cleanly)
   */
  public void
  quit()
  {
    synchronized(m_doneMonitor)
    {
      m_done = true;
      m_doneMonitor.notifyAll();
    }
  }
}
```

5 Using the Application View Console

The Application View Console is a graphical user interface (GUI) that offers an easy way to access, organize, and edit all the Application Views in your enterprise. You can use the Application View Console to create new folders and to add new Application Views to them. By storing your Application Views in folders, you can organize them according to your own navigation scheme, regardless of the adapters to which the individual Application Views belong.

This section presents the following topics:

- Logging On to the Application View Console
- Creating a Folder
- Removing an Application View
- Removing a Folder

Logging On to the Application View Console

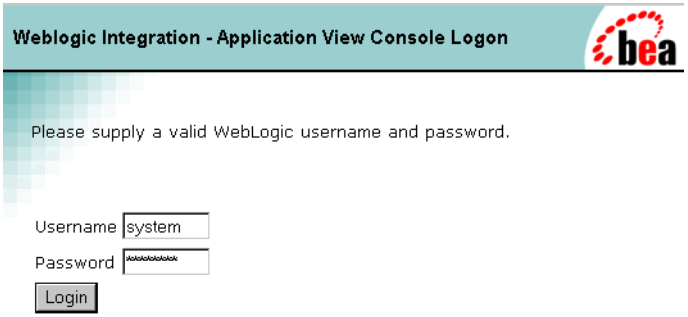
To log on to the Application View Console:

1. Launch a browser window.
2. Enter the URL for your system's Application View Console in the following format:

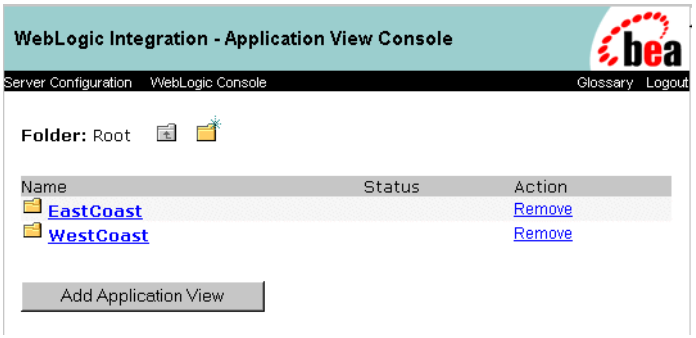
   `http://your_server:your_port/wlai`

For example: `http://wli1:7001/wlai`

The logon page is displayed.



3. Enter your WebLogic Server username and password, then click OK. The Application View Console is displayed.



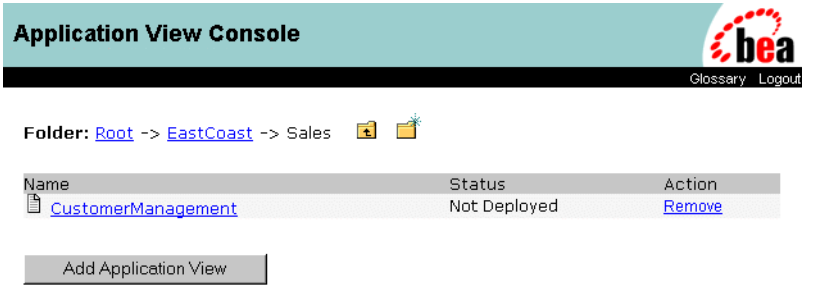
Creating a Folder

The Application Views in your enterprise are organized in folders that may contain Application Views and subsidiary folders. Once you create a folder, you cannot move it to another folder. Before removing a folder, you must remove all Application Views and subfolders.

Once you create an Application View in a folder, you can remove the Application View, but you cannot move it to another folder.

To create a folder:

1. While logged on to the Application View Console, navigate to the folder in which you want to create the new folder.



2. Click the New Folder icon:

The Add Folder page is displayed.

3. In the New Folder field, enter a name. Any valid Java Identifier is allowed in a name.

   Note: The name Root is a reserved word, and cannot be used for a folder name. If you use Root as a name, you cannot import or export the folder using the import/export utility.

4. Click Save.

Removing an Application View

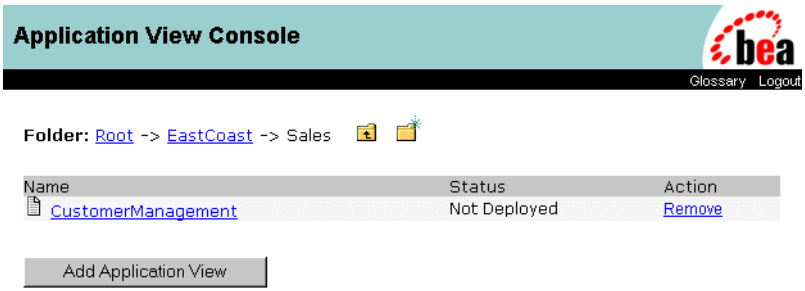
Remove Application Views when they become obsolete or when the application to which they belong is retired.

You can remove an Application View only if both of the following conditions are true:

- You have undeployed the Application View. (See "Optional Step: Undeploy an Application View" on page 2-20.) Make sure the Application View status is Not Deployed.
- You are logged on to WebLogic Server with a user account with the appropriate write privileges.

To remove an Application View:

1. While logged on to the Application View Console, navigate to the folder in which the target Application View is located.

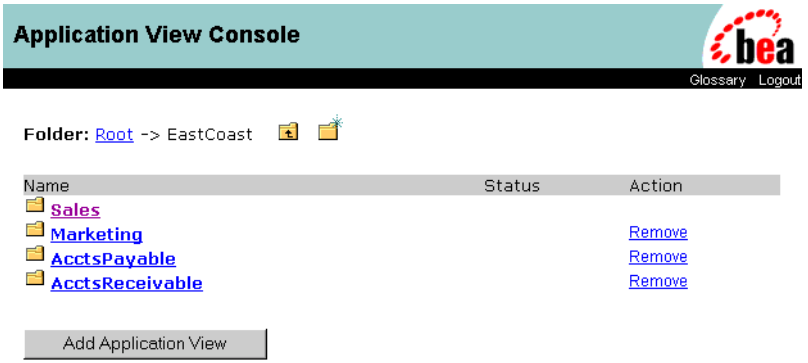


2. Click Remove. A confirmation page is displayed. Click Confirm to delete the Application View.

Removing a Folder

Remove folders that are no longer needed. To remove a folder:

1. Remove all Application Views and subfolders from the target folder.
2. Log on to the Application View Console and go to the folder in which the target folder resides.



3. Click Remove. A confirmation page is displayed.

4. Click Confirm. The folder is deleted.

A Migrating Application Integration Data

This section presents the following topics:

- Overview of Migrating Data
- Migrating Data Within a Single EIS Instance
- Migrating Data Within Multiple EIS Instances
- Recommended Practices

Overview of Migrating Data

Configuration data for application integration functions is stored in the same repository as configuration data for business process management (BPM) functions. Therefore, you can use the same tools to migrate data for both types of functions. However, several special considerations apply to the migration of application integration data and deployment of that data in the target environment.

Note: The application integration engine now supports modular deployment. It is bundled in an enterprise application archive (EAR) file and is available to any valid WebLogic domain. Because the BPM capabilities may not always be installed, a command-line interface to the import / export tool is provided for Application Views. For more information, see Appendix B, "Importing and Exporting Application Views."

Migrating application integration data is straightforward when you are migrating between WebLogic Server domains within a single Enterprise Information System (EIS) instance. When you migrate application data between WebLogic Server domains in different EIS instances, however, you must perform special procedures to ensure a working solution in the target environment.

This section provides information about migrating application integration data between WebLogic Server domains in the following scenarios:

- Migrating Data Within a Single EIS Instance
- Migrating Data Within Multiple EIS Instances

Migrating Data Within a Single EIS Instance

This section describes how to migrate application integration data among multiple WebLogic Server domains within a single EIS instance. For example, you might move Application View definitions between repositories for different domains of WebLogic Integration. In this case, only the WebLogic Integration domain changes; the target EIS instances referred to in the Application Views remain the same.

In this case, the workflow package import/export utility (accessible from the WebLogic Integration Studio) simplifies the job of migrating data. It involves exporting a package from the Studio in the source domain, and importing that package into the Studio in the target domain.

For more information about this import/export utility, see "Importing and Exporting Workflow Packages" in *Using the WebLogic Integration Studio*.

How an Application View Is Exported

When you export a workflow that includes application integration functionality, the export tool automatically identifies the Application Views and other resources on which the workflow depends. Listing A-1 and Listing A-2 show values identifying an Application View and the resources on which it depends in the export tool. In general, the Application View value is displayed (in the export tool) in the location shown in Listing A-1.

Listing A-1 Application View Location in the BPM Export Tool

```
All Workflow Objects
  |--  XML Repository
    |-- Folder: WLAI.Namespace.Root
      |-- Folder: WLAI.Namespace.Root.firstfolder
        |-- Folder: WLAI.Namespace.Root.firstfolder.nthfolder
          |-- Entity: WLAI.ApplicationView.Root.firstfolder.
                      nthfolder.appviewname
```

In general, entities related to the Application View can be found under the *nth_folder*, and are named according to the convention shown in Listing A-2. Not all Application Views follow this convention, however.

Listing A-2 Application View Resource Locations in BPM Export Tool

```
Entity: WLAI.entity_type.Root.firstfolder.nthfolder.appviewname_event/
servicename_adapterspecific
```

To fully export an Application View, you *must* select all entities that are related to it, including entities of `Schema` and `ConnectionFactory` types.

Example Application View Export

The `EastCoast.Sales` folder contains an Application View named `CustomerManagement` which is displayed in the BPM export tool at the location shown in Listing A-3.

Listing A-3 Application View in the BPM Export Tool

```
All Workflow Objects
  |--  XML Repository
    |-- Folder: WLAI.Namespace.Root
      |-- Folder: WLAI.Namespace.Root.EastCoast
        |-- Folder: WLAI.Namespace.Root.EastCoast.Sales
```

```
          |-- Entity:WLAI.ApplicationView.Root.EastCoast.
                    Sales.CustomerManagement
```

To fully export the `CustomerManagement` Application View, the export tool automatically selects all entities that conform to the following pattern:

```
Entity: WLAI.entitytype.Root.EastCoast.Sales.CustomerManagement
```

The `CustomerManagement` Application View contains several events and services: the export utility shows one `Schema` type entity for each event and two `Schema` type entities for each service. The `CustomerManagement` Application View also uses the DBMS adapter, and includes one event named `CustomerCreated` and one service named `CreateCustomer`. Therefore, the entities shown in Listing A-4 are listed by the export utility.

Listing A-4 Entities Used by the Application View

```
Entity: WLAI.Schema.Root.EastCoast.Sales.CustomerManagement_CustomerCreated_
CUSTOMER_TABLE_insert

Entity: WLAI.Schema.Root.EastCoast.Sales.CustomerManagement_CreateCustomer_input

Entity: WLAI.Schema.Root.EastCoast.Sales.CustomerManagement_
CreateCustomer_output
```

The `CustomerManagement` Application View also includes a single connection factory. The entity name for this connection factory is as follows:

```
Entity: WLAI.ConnectionFactory.Root.EastCoast.Sales.CustomerManagement.
ConnectionFactory
```

For the `CustomerManagement` Application View to be exported properly, all the entities shown in Listing A-4 must be selected.

Importing an Application View

To import an Application View, perform the following steps:

1. Using the workflow package import utility (accessible from the WebLogic Integration Studio), import a package containing application integration data. The utility automatically imports all the entities you exported into the package earlier.
2. Deploy your imported Application Views using the WebLogic Integration Application View Console (generally located at `http://server:port/wlai`).
3. Navigate through the imported folders to find the imported Application View.
4. Select the appropriate Application View. Then select the Deploy option on the Application View Summary page.

The Application View is now ready for use in the target environment.

Migrating Data Within Multiple EIS Instances

Be careful when migrating data among WebLogic Server domains and between multiple instances of an EIS, because Application Views defined for a particular EIS instance contain identifiers and other data specific to that instance. This advice also applies to the connection factory used by the Application View.

To prevent possible errors, you must manually change EIS instance-specific data in your Application View or connection factory by performing the following steps:

1. Open the Application View Console.
2. Navigate to the appropriate Application View and edit it as necessary:
   a. Identify and update all EIS-specific data in the Application View and the events, services, and connection factory associated with that Application View.
   b. Search for any EIS-instance-specific references, and replace them with references to the new EIS instance in the target environment.

Be sure to edit the Application View and connection factory definitions. The following parameters of the Application View definitions may need to be changed:

- The `EventRouterURL` parameter of the `ApplicationView` deploy screen must refer to the event router in the target environment.
- Parameters in the service definitions. These are adapter-specific data that might refer to EIS instance-specific data. Change any EIS instance-specific parameters for the service as necessary.
- Parameters in the event definitions. These are adapter-specific data that may refer to EIS instance-specific data. Use the Edit feature on the Application View Summary page to change any EIS instance-specific parameters for the service.

Example Application View Import

The `CustomerManagement` example includes a database called `CUST`, in the source environment, and a database called `CustDB` in the target environment. Listing A-5 shows the XML text that represents the Application View and connection factory. Specifically, this listing shows the Application View descriptor for the `CustomerManagement` Application View. When you use the Application View Console, you must use the appropriate fields in the design-time UI forms to view and edit this information.

Listing A-5 Example XML Text for the Application View and Connection Factory

```
<?xml version="1.0"?>
<!DOCTYPE applicationView>
<applicationView asyncEnabled="true"
connectionFactory="com.bea.wlai.connectionFactories.EastCoast.Sales.
CustomerManagement_connectionFactoryInstance"
connectionFactoryName="EastCoast.Sales.CustomerManagement_connectionFactory"
eventRouterURL="http://localhost:7001/DbmsEventRouter/EventRouter"
      name="CustomerManagement"
      ownsConnectionFactory="true">
  <description>Manages customers in the east coast sales database</description>

  …

  <service interactionSpecClass="com.bea.adapter.dbms.cci.InteractionSpecImpl"
```

```
      name="CreateCustomer"
      ownsRequestSchema="true"
      ownsResponseSchema="true"
requestDocumentType="EastCoast.Sales.CustomerManagement_CreateCustomer_input/In
put"
responseDocumentType="EastCoast.Sales.CustomerManagement_CreateCustomer_output/
RowsAffected">
   <description>create a new customer in database</description>
   <interactionSpecProperty
name="functionName">executeUpdate</interactionSpecProperty>
   <interactionSpecProperty name="sql">insert into CUST.dbo.CUSTOMER_TABLE
(FirstName, LastName, DOB) values ([FirstName varchar], [LastName varchar], [DOB
timestamp])</interactionSpecProperty>
   </service>

   <event name="CustomerCreated"
      ownsSchema="true"
      rootElementName="CUSTOMER_TABLE.insert"

schemaName="EastCoast.Sales.CustomerManagement_CustomerCreated_CUSTOMER_TABLE_i
nsert">
   <description>New customer created in database</description>
   <eventProperty name="tableName">CUSTOMER_TABLE</eventProperty>
   <eventProperty name="triggerType">insert</eventProperty>
   <eventProperty name="catalogName">CUST</eventProperty>
   <eventProperty name="schemaName">dbo</eventProperty>
  </event>

</applicationView>
```

This Application View contains:

- Explicit reference to the event router URL. (The URL is probably different in the target domain from that in the original domain if you change EIS instances.)
- `interactionSpecProperty` elements with explicit SQL statements (expressed with the `<service>` element) that refer to the `CUST` database, the `dbo` schema, and the `CUSTOMER_TABLE` table.
- `eventProperty` elements that refer to `catalogName` as `CUST` and `schemaName` as `dbo`. In this example, all references to `CUST` (highlighted in the preceding text) must be changed to `CustDB`. If a different schema is used, the schema references must also be changed.

Each adapter puts different properties in the service and event descriptors of the Application View descriptors it creates. For information about which properties must be changed to operate successfully with a new EIS instance, see your adapter documentation.

You must also change the connection factory descriptor so that it refers to the new EIS instance. Listing A-6 shows a sample connection factory.

Listing A-6 Sample Connection Factory

```
<?xml version="1.0"?>
<!DOCTYPE connection-factory-dd>
<connection-factory-dd name="CustomerManagement_connectionFactory">
  <jndi-name>com.bea.wlai.connectionFactories.EastCoast.Sales.
   CustomerManagement_connectionFactoryInstance</jndi-name>
  <pool-parms allowPoolToShrink="true"
      maxPoolSize="10"
      minPoolSize="0"
      targetFractionOfMaxPoolSize="0.1"/>
  <mcf-parm name="MessageBundleBase">
    <mcf-parm-value>BEA_WLS_DBMS_ADK</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="DataSourceName">
    <mcf-parm-value>eventSource</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="AdditionalLogContext">
    <mcf-parm-value>CustomerManagement</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="UserName">
    <mcf-parm-value>system</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="Password">
    <mcf-parm-value>security</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="RootLogContext">
    <mcf-parm-value>BEA_WLS_DBMS_ADK</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="PingTable">
    <mcf-parm-value>CUST.dbo.CUSTOMER_TABLE</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="LogLevel">
    <mcf-parm-value>WARN</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="LogConfigFile">
    <mcf-parm-value>BEA_WLS_DBMS_ADK.xml</mcf-parm-value>
  </mcf-parm>
```

```
  <adapter-logical-name>BEA_WLS_DBMS_ADK</adapter-logical-name>
</connection-factory-dd>
```

As shown in the preceding code, this connection factory descriptor refers directly to the `CUST` database and to a JDBC data source named `eventSource`. To make sure that this connection factory operates properly in the target environment, you must make the following changes:

- Change the reference to `CUST` toa reference to `CustDB`
- Change the `eventSource` JDBC data source reference so it refers to a valid JDBC data source (pointing at the new DBMS that is hosting `CustDB`) in the target domain.

At this point, you have modified all necessary references and ensured that all the resources needed by the Application View and connection factory exist in the target domain. You may now deploy all the Application Views you imported, using the Application View Console (generally located at `http://`*`host`*`:`*`port`*`/wlai`).

Recommended Practices

The following suggestions are offered to help you reduce the effort needed to migrate application integration data between environments.

- Whenever possible, set up identical EIS instances in both the source and target domains. For example, in the Application View in which the DBMS adapter is used, you might use two instances of MS SQL Server (with the same name, the same user accounts, and the same database object) for both your source and target databases. This type of setup eliminates the need for manual editing of Application View and connection factory descriptors.
- Change the event router URL to reflect the event router's location in the target environment. To do so, edit the Application View in the Application View Console.
- After your Application Views are imported, deploy them using the Application View Console.

B Importing and Exporting Application Views

This section presents the following topics:

- Import/Export Utility
- Import/Export Methods and Command Line

Import/Export Utility

WebLogic Integration provides a simple import/export utility for Application Views that can be executed from the command line, and incorporated into your code with the import/export API for Application Views. The output of the utility is a JAR file containing all artifacts owned by the Application View. You must manually import or export any artifacts that are used but not owned by the Application View.

Note: If the business process management (BPM) capabilities of WebLogic Integration are installed, you can use the workflow package import/export utility (accessible from the WebLogic Integration Studio) to migrate application integration data. For more information, see Appendix A, "Migrating Application Integration Data."

The import/export utility allows you to export application integration metadata objects from the repository and to import those objects back into the repository. The utility allows you to import/export the following objects:

- Application views
- Connection factories (with descriptors as stored in the repository, not those deployed from WebLogic Server)
- Schemas
- Namespaces

All exported objects for a given invocation of the utility are stored in a JAR archive. When a previously exported JAR is imported, all objects in the JAR are imported, too. You can also append objects to an existing exported JAR, and deploy Application Views and connection factories on import.

Import/Export Methods and Command Line

The utility is available as an API and as a command-line tool. In both cases, the server must be running. The following sections describe the command-line parameters and the methods on the import/export utility.

Invoking the Import/Export Utility from the Command Line

The following is the command-line syntax for the import/export utility:

```
Usage: ImportExport <server_URL> <username> <password> <file>
[-codepage=Cp<codepage_number>] [-dump=< <namespace> | <'Root'> >]
[-append] [-overwrite] [-deploy]
< [-export [object_name]*] | [-import [edit-on-import_filename]] >
```

The following table shows the command-line parameters for the import/export utility.

Parameter	Description
server_URL	URL of WebLogic Server. This is required only if you are connecting from a remote client.
username	Your username for the specified WebLogic Server. This is required only if you are connecting from a remote client.
password	Your password for the specified WebLogic Server. This is required only if you are connecting from a remote client.
file	Name of the file to be created on export or to be imported into the repository.
-codepage	Sets the codepage used when writing to the console. This insures that characters are displayed correctly. The default value is Cp437, which is used in the United States. Other valid values include: Cp850 Multilingual (Latin I) Cp852 Slavic (Latin II) Cp855 Cyrillic (Russian) Cp857 Turkish Cp860 Portuguese Cp861 Icelandic Cp863 Canadian-French Cp865 Nordic Cp866 Russian Cp869 Modern Greek MS932 Japanese
-dump	Prints a list of all objects within both the specified namespace and other namespaces nested within it. To print a list of objects for the entire folder structure, specify `Root`.
-append	Appends exported items to the file specified by *file*.
-overwrite	Overwrites items already in the repository when import is being performed.
-deploy	Deploys the Application View or connection factory on import.

Parameter	Description
`-export`	Specifies an export operation and the name of the objects to be exported. Wildcards are allowed in object names. To export the entire folder structure, include `Root` in the list of object names.
`-import`	Specifies that objects contained in *`file`* should be imported into the repository. If an edit-on-import file is specified, objects are edited according to the instructions in the file. Edits are performed before the object is added to the repository and before deployment (if requested).

Editing on Import

When an edit-on-import file is specified, you can execute editing commands on the text for objects to be imported. The resulting editing is done before the object is stored in the repository or deployed. Otherwise, the method is used as described in "Importing Objects."

The document specified must conform to the following DTD:

```
<!ELEMENT ApplicationView (replace*)>
<!ATTLIST ApplicationView name NMTOKEN #REQUIRED
                          newName NMTOKEN #IMPLIED>

<!ELEMENT ConnectionFactory (replace*)>
<!ATTLIST ConnectionFactory name NMTOKEN #REQUIRED
                            newName NMTOKEN #IMPLIED>

<!ELEMENT Schema (replace*)>
<!ATTLIST Schema name NMTOKEN #REQUIRED
                 newName NMTOKEN #IMPLIED>

<!ELEMENT replace EMPTY>
<!ATTLIST replace xpath CDATA #REQUIRED
                  old CDATA #REQUIRED
                  new CDATA #REQUIRED
```

The edit-on-import document contains sections, indicated by the `<ELEMENT>` tag, for each object to be edited. You can edit ApplicationView, ConnectionFactory, and Schema objects. Each section identifies an object by name and specifies the elements to be replaced. Optionally, each section can specify a `newName` attribute that can be used to assign a new name to an object.

Each `replace` element specifies the following:

- An xpath expression used to identify the nodes to be edited, among all the nodes in the object's XML descriptor
- An old value to search for in the selected nodes. The old value can be a Perl5 regular expression. If you specify an empty string (`""`) as the old value, all text in the selected nodes is matched.
- A new value with which to replace the old value. The new value must be a simple string.

Example Edit-on-Import Document

The following edit-on-import descriptor document describes how to edit an Application View named DBMS.DBMS1 and rename it to DBMS.DBMS1a. The XML document illustrates three replacements for the Application View and one replacement for the connection factory.

```
<?xml version="1.0"?>
<!DOCTYPE edit SYSTEM "ImportExportEditOnImport.dtd">
<edit>
   <ApplicationView name="DBMS.DBMS1" newName="DBMS.DBMS1a">
      <replace xpath="/applicationView/@connectionFactory"
         old=""
         new="com.bea.wlai.connectionFactories.DBMS.
              DBMS1a_connectionFactoryInstance"/>
      <replace xpath="/applicationView/@connectionFactoryName"
         old=""
         new="DBMS.DBMS1a_connectionFactory"/>
      <replace        xpath="/applicationView/service[@name='Service1']/
       interactionSpecProperty[@name='sql']"
         old="CAJUN."
         new="PBPUBLIC."/>
   </ApplicationView>

   <ConnectionFactory name="DBMS.DBMS1_connectionFactory"
    newName="DBMS.DBMS1a_connectionFactory">
      <replace xpath="/connection-factory-dd/jndi-name"
         old=""
         new="com.bea.wlai.connectionFactories.DBMS.
              DBMS1a_connectionFactoryInstance"/>
   </ConnectionFactory>
</edit>
```

The first replacement edits the `connectionFactory` attribute of the Application View descriptor and changes the text in this attribute to the new value. Note the use of an empty string as the old value to match all text in the node.

The second replacement edits the `connectionFactoryName` attribute of the Application View descriptor and changes the text in this attribute to the new value.

The third replacement edits the service descriptor for the service named `Service1` and changes the `interactionSpecProperty` element named `sql` by replacing `CAJUN.` with `PBPUBLIC.` wherever it occurs.

The fourth replacement edits the `jndi-name` attribute of the connection factory descriptor and changes the text in this attribute to the new value.

Using the Import/Export API

The following sections describe the methods included in the import/export API. The class name for the import/export API is `com.bea.wlai.client.ImportExport`.

Connecting to the Server Instance

`connect(<multiple signatures>)`

The `connect`() method establishes a connection method with the server instance. Depending on where you initiate connections, you may need to specify different arguments for `connect`().

If you are initiating connections. . .	Then you must specify these arguments. . .
Within the same server	No arguments
From a remote client without `InitialContext`	URL (as a string), username, and password
From a remote client with `InitialContext`	`InitialContext`

Printing Objects in a Namespace

```
dumpNamespace(String namespaceName)
```

The `dumpNamespace`() method takes a string that represents the qualified name of a namespace, and prints out all objects within that namespace, as well as any other namespaces embedded in it.

Exporting Objects

```
exportNamespaceObjects(Set objectNames, boolean append, List errors)
```

The `exportNamespaceObjects`() method takes a list of object names (qualified names as strings) and exports them to the specified output file (see "Specify File for Import/Export"). All objects listed for export are examined for dependencies. Any objects that are used but not owned by an Application View are also exported.

When an Application View is exported, the ConnectionFactory and all the Schema objects on which the Application View depends are also exported. When a Namespace is exported, all objects in the namespace (including other namespaces) are also exported.

To append the exported file to an existing archive, or to overwrite or create the file, set `append` to true.

Any nonfatal errors encountered during the export are stored as Exception objects in the specified errors List object.

Importing Objects

```
importNamespaceObjects(boolean overwrite, boolean deploy, List errors)
```

The `importNamespaceObjects`() method takes all entries in the specified input file (see "Specify File for Import/Export") and imports them into the repository. Set `overwrite` to true to overwrite existing metadata in the repository. Set `deploy` to true to deploy connection factories and Application Views on import.

Any nonfatal errors encountered during the import are stored as Exception objects in the specified errors List object.

Importing and Editing Objects

```
importNamespaceObjects(IDocument editOnImportDoc boolean overwrite,
boolean deploy, List errors)
```

When an edit-on-import document is specified, the `importNamespaceObjects`() method allows you to execute editing commands on the text for objects to be imported. The resulting editing is done before the object is stored in the repository or deployed. Otherwise, the method is used as described in "Importing Objects."

The document specified in the *`editOnImportDoc`* argument must conform to the DTD shown in "Editing on Import."

Specify File for Import/Export

```
setFile(File filename)
```

The `setFile`() method designates the file to be used as the export destination or import source.

Choosing Where to Print Messages

```
setPrintWriter(PrintWriter out)
```

The `setPrintWriter`() method designates a PrintWriter object to be used when messages (such as status, diagnostic, and error messages) are generated.

Choosing Whether to Print Messages

```
setQuiet(boolean quiet)
```

The `setQuiet`() method specifies whether to print progress and information messages. To print messages, set `quiet` to `false`. To disable message printing, set `quiet` to `true`.

C Modular Deployment of Application Integration

This section presents the following topics:

- Overview
- Deployment Components
- Deployment Configuration for Domains Outside the WebLogic Integration Environment
- JMS Resources

Overview

In pre-7.0 WebLogic Integration environments, Application Views are usually created and managed through a business process management (BPM) workflow. In a Release 7.0 environment, however, Web service developers using WebLogic Workshop need to use Application Views to access enterprise information systems. The application integration functionality of WebLogic Integration is packaged in an enterprise application archive (EAR file) and is available to any valid WebLogic domain. The following sections describe changes in the deployment process required for modular deployment. For detailed information about deployment concepts and tasks, see *Deploying BEA WebLogic Integration Solutions*.

Classpath Changes and Server Restart

You must add the `wlai-core.jar` file to the classpath for domains that are not based on WebLogic Integration. To implement the change in the classpath, you must then restart the server.

Repository

The application integration engine uses the repository for metadata persistence. The application integration engine relies on a preconfigured repository and associated JDBC connection pool and data source. For modular deployment, you must provide the JDBC data source name and credentials at the time of deployment. The application integration engine assumes that the repository has already been installed in the data source.

JMS Resources

Application views use JMS resources to handle events and asynchronous service invocations. To support these functions, Application Views use the following resources: `JMSConnectionFactory`, `JMSTemplate`, `JMSJDBCStore`, and `JMSServer`. In addition, the application integration engine defines a request and response queue for handling asynchronous service invocations. There are two modes of operation supported for determining which JMS resources to use.

- You provide the name of existing JMS resources for any or all of the following resources:
  - `JMSConnectionFactory`
  - `JMSServer`

  Note: If you provide the name for the `JMSServer`, you must also configure the `JMSJDBCStore`.
- You do not provide the name of existing JMS resources. In this case, the startup process uses JMS MBeans to define the necessary JMS resources. The full list of JMS Resources used by the application integration engine is described in "JMS Resources."

Configuration

The application integration startup and shutdown classes are replaced by an Enterprise JavaBean (EJB) named `StartupBean`. The EJB is deployed from `wlai-server-ejb.jar` and is loaded on startup. On startup, the EJB initiates the initialization sequence for the application integration engine. The initial parameters to the `StartupBean` EJB serve as the configuration parameters for the application integration engine. You can set these parameters using the standard EJB Descriptor editing tool provided in the WebLogic Administration Console. The following table lists the configuration parameters.

Property Name	Default Value	Description
`wlai.logLevel`	`warning`	Verbosity level for application integration logging.
`wlai.deploymentRepositoryRootPath`	*`$PWD`*`/wlai/deploy`; where `$PWD` is the present working directory for WebLogic Server	The location where the application integration engine saves connection factory deployment descriptors
`wlai.hostUserID`	`system`	A user identifier. Application integration allows a remote event router (deployed from a Web application) to authenticate itself to the WebLogic Server so that it can post events.
`wlai.hostPassword`	`security`	The password for a user. Application integration allows a remote event router (deployed from a Web application) to authenticate itself to the WebLogic Server so that it can post events.
`wlai.jms.autogen`	`true`	Flag that allows the application integration startup process to autogenerate JMS resources.
`wlai.jms.serverName`	`WLIJMSServer`	Name of the JMSServer on the local WebLogic Server.

wlai.jms.connectionFactoryJNDIName	com.bea.wlai.JMSConnectionFactory	JMS Connection Factory JNDI context.
wlai.repositoryDatasourceName	WLAI_DataSource	JDBC data source name.

In the Application View Console, click on Server Configuration to view the configuration parameters for the application integration engine. You can edit the `wlai.logLevel` property on the Server Configuration page. All other parameters must be edited using the WebLogic Administration Console.

Start and stop the application integration engine from the WebLogic Administration Console by deploying and undeploying the WebLogic Integration application that contains the application integration component.

Import/Export Utility

In previous releases, application integration data was migrated using the workflow package import/export utility accessible from the WebLogic Integration Studio. To support modular deployment, application integration provides a command-line import/export tool for Application Views. For more information, see Appendix B, "Importing and Exporting Application Views."

Deployment Components

The following list describes the deployment components for application integration.

- `wlai-core.jar`

  This `jar` file must be added to the WebLogic server classpath at startup. It contains the following components:

```
wlai-core.jar
|__log4j.jar contents
|__logtoolkit.jar contents
|__xmltoolkit.jar contents
|__bea.jar contents
|__com/bea/wlai/*.dtd
```

```
|__com/bea/wlai/common/*.class
|__com/bea/wlai/message/*.class
|__xcci.jar contents
```

This `jar` file is also required for all clients of application integration.

- `wlai-client.jar`

  This `jar` file contains all classes needed by clients of the application integration engine, such as Application View clients and resource adapter design time Web applications.

- `wlai-server.jar`

  This `jar` file contains all classes needed only by server-side components and should not be included in any adapter EAR files or clients.

- `wlai-mbean.jar`

  This `jar` file contains the Application View MBeans and must be included on the system classpath for WebLogic Server.

- `wlai-server-ejb.jar`

  This `jar` file contains the base server classes and the management EJBs. This `jar` file must be deployed before all other components of the AI engine. This `jar` file contains the Startup EJB which initializes application integration.

- `wlai.war`

  This `war` file contains the Application View Console Web Application and the `LifeCycleServlet` for the application integration engine.

- `wlai-eventprocessor-ejb.jar`

  This `jar` file contains the event processing message driven EJB for handling application integration events.

- `wlai-asyncprocessor-ejb.jar`

  This `jar` file contains the asynchronous service processing message driven EJB for handling asynchronous service invocations.

- `wlai-plugin-ejb.jar`

  This `jar` file contains all classes for the Application Integration Plug-in for BPM.

- `wlai-plugin.war`

  This `war` file contains the online help Web application for the Application Integration Plug-in for BPM.

Deployment Configuration for Domains Outside the WebLogic Integration Environment

To deploy an application integration enterprise application outside a WebLogic Integration domain, use `wlai.ear`. The EAR file contains the components shown in the following diagram:

```
wlai.ear
|__META-INF
|     |__application.xml (EAR file deployment descriptor)
|__wlai.war (Application View Console Web application)
|     |__WEB-INF
|         |__lib
|             |__webtoolkit.jar
|__wlai-server-ejb.jar (Application Integration Management EJBs)
|     |__Startup EJB
|     |__ApplicationView EJB
|     |__SchemaManager EJB
|     |__DeployManager EJB
|     |__ApplicationViewManager EJB
|     |__NamespaceManager EJB
|__wlai-eventprocessor-ejb.jar
|__wlai-asyncprocessor-ejb.jar
|__ecibase.jar (ECI repository base classes)
|__ecirepository.jar (ECI repository classes)
```

You must add `wlai-core.jar`, `wlai-server.jar`, and `wlai-mbean.jar` to the system classpath and restart WebLogic Server. You must also copy the `wlai-mbean.jar` file to your *`WL_HOME`*`\lib\mbeantypes` directory where *`WL_HOME`* is your Web Logic Server installation.

Once the `wlai-core.jar` has been added to the classpath, add the following Application component to the `config.xml` file for the domain, or upload it from the WebLogic Server Administration Console:

```
<Application Deployed="true" Name="WebLogic Application Integration"
Path="PATH_TO_EAR/wlai.ear">

<EJBComponent Name="WLI-AI Server" Targets="myserver"
URI="wlai-server-ejb.jar"/>
```

```
<WebAppComponent Name="wlai"
Targets="myserver" URI="wlai.war"/>

<EJBComponent Name="WLI-AI Async Processor" Targets="myserver"
URI="wlai-asyncprocessor-ejb.jar"/>

<EJBComponent Name="WLI-AI Event Processor" Targets="myserver"
URI="wlai-eventprocessor-ejb.jar"/>

</Application>
```

Note: The deployment order is specified in the `application.xml` file for the `wlai.ear`.

Note: The Application Integration Plug-In for BPM is not deployed from the `wlai.ear` file for domains outside the WebLogic Integration environment.

JMS Resources

The application integration engine uses the following JMS resources:

- `JMSServer`
- `JMSConnectionFactory`

  If the `JMSConnectionFactory` supplied by the user is not bound to JNDI location `com.bea.wlai.JMSConnectionFactory`, the factory is cloned and bound to `com.bea.wlai.JMSConnectionFactory`. As a result, internal application integration components are guaranteed access to a `JMSConnectionFactory`.

- Queues:
  - `WLAI_EVENT_QUEUE`—Must be bound at `com.bea.wlai.EVENT_QUEUE`.
  - `WLAI_ASYNC_REQUEST_QUEUE`—Must be bound at `com.bea.wlai.ASYNC_REQUEST_QUEUE`.
  - `WLAI_ASYNC_RESPONSE_QUEUE`—Must be bound at `com.bea.wlai.ASYNC_RESPONSE_QUEUE`.
- Topics:
  - `WLAI_EVENT_TOPIC`—Must be bound at `com.bea.wlai.EVENT_TOPIC`.

JMS Resource Configuration

The application integration engine automatically defines all required JMS resources if you do not define them explicitly for a standalone server and for a cluster. The system administrator can specify the following configuration parameters:

```
wlai.jms.serverName
wlai.jms.connectionFactoryJNDIName
```

In addition, the administrator has the option of disabling all autogeneration of JMS resources, by specifying the following configuration parameter:

```
wlai.jms.autogen=false
```

This parameter prevents the application integration engine from attempting to autogenerate any required JMS resources. If you specify it, you must define any required JMS resources manually.

Index