BEA WebLogic Integration™

Using the Application Integration Design Console

Version 8.1 Service Pack 6
Revised: June 2006

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

1. Introduction to Application Integration

Familiarizing Yourself with Basic Concepts 1-2
Creating an Interface to an Adapter 1-2
When to Define an Application View 1-2
When to Write Custom Code 1-2
Defining an Application View 1-3
What Is Defined by an Application View Definition 1-3
How to Define an Application View 1-4
Step 1: Log On to the Application Integration Design Console 1-4
Step 2: Define the Application Context for an Application View 1-4
Step 3: Add Folders 1-5
Step 4: Define Application View 1-5
Step 5: Create New or Select Existing Connection 1-5
Step 6: Add Services and Events to the Application View 1-5
Step 7: Perform Final Configuration 1-5
Step 8: Test Services and Events 1-6
Step 9: Publish Application View 1-6
Using an Application View in a Business Process 1-6
Using an Application View in WebLogic Workshop 1-6
Using an Application View by Writing Custom Code 1-7
Choosing a Method for Implementing a Business Process 1-7
When to Use WebLogic Workshop to Create a Business Process 1-7

When to Write Custom Java Code 1-7
Using an Application View Control in Business Processes 1-8

2. Defining an Application View

Before You Begin 2-2
Application View Design Considerations 2-2
High-Level Procedure for Defining an Application View 2-3
Sample Detailed Procedure for Defining an Application View 2-5
Step 1: Log On to the Application Integration Design Console 2-5
Step 2: Select an Application 2-7
Step 3: Add a Folder 2-8
Step 4: Define an Application View. 2-9
Step 5: Establishing a Browsing Connection 2-11
Step 5A: Create a New Browsing Connection 2-12
Step 5B: Reuse an Existing Browsing Connection. 2-15
Step 6: Add Services and Events 2-17
Step 6A: Add a Service to an Application View. 2-17
Step 6B: Add an Event to an Application View 2-19
Step 7: Perform Final Configuration Tasks 2-21
Step 8A: Test an Application View's Services. 2-26
Step 8B: Test an Application View's Events 2-29
If You Select Service. 2-30
If You Select Manual 2-30
Step 9: Publish an Application View 2-31
Editing an Application View. 2-33
Setting Transaction Timeout Values for Application Views 2-33
Environment Variables for Application Views. 2-34
Database-Specific Error Messages 2-34

3. Using Application Views with Business Processes

Before You Begin 3-1

Integrating Application Views and Business Processes Using a Control 3-2

Sample Application View Control Files 3-4

Application View Control Interface 3-4

Application View Control (JCX) and Business Process (JPD) Samples 3-6

Receiving Events 3-6

Handling Application View Local Transactions in Business Processes 3-8

Local Transaction Management Contracts 3-9

Connector Support for Local Transactions with No User Defined Transaction Demarcation 3-9

Connector Support for XA Transactions 3-9

4. Using the Application Integration Design Console

Logging On to the Application Integration Design Console 4-1

Creating a Folder 4-4

Removing an Application View 4-5

Removing a Folder 4-5

Administering an Application View 4-6

5. Using Application Views by Writing Custom Code

Scenario 1: Creating Connections with Specific Credentials 5-1

Implementing ConnectionSpec 5-2

Calling setConnectionSpec() and getConnectionSpec() 5-2

Using the ConnectionSpec Class. 5-3

Scenario 2: Custom Coding a Business Process 5-4

About This Scenario 5-5

Before You Begin 5-5

Creating the SyncCustomerInformation Class 5-6
Code for Sample Java Class 5-8

A. Import-Export Utility A-1
Migrating Application Views Using the Import-Export Utility A-2
Import-Export Methods and Command Line A-2
Invoking the Import-Export Utility from the Command Line A-2
Editing on Import A-6
Using the Import-Export API A-8
Connecting to the Server Instance A-8
Printing Objects in a Namespace A-8
Exporting Objects A-8
Importing Objects A-9
Importing and Editing Objects A-9
Specify File for Import-Export A-9
Choosing Where to Print Messages A-9
Choosing Whether to Print Messages A-10

Index

CHAPTER 1

Introduction to Application Integration

This document provides instructions for using adapters via application views in a WebLogic Integration environment. Adapters may be purchased or developed using the BEA WebLogic Integration Adapter Development Kit (ADK). This document 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:

- Familiarizing Yourself with Basic Concepts
- Creating an Interface to an Adapter
- Defining an Application View
- Using an Application View in a Business Process
- Using an Application View Control in Business Processes

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. You can create application views for standards-based BEA WebLogic Adapters for WebLogic Integration. Documentation for these adapters is available at `http://edocs.bea.com`. For details about the DBMS adapters provided with the ADK, see Learning to Develop Adapters Using the DBMS Sample Adapters in *Developing Adapters.*

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," and Chapter 5, "Using Application Views by Writing Custom Code."

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 WebLogic Workshop.
- You need to be able 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 implemented as business processes using WebLogic Workshop.
- 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:

- Where application view information is to be stored
- A unique name for the application view
- Parameters for the following:
  - 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: Log On to the Application Integration Design Console
- Step 2: Define the Application Context for an Application View
- Step 3: Add Folders
- Step 4: Define Application View
- Step 5: Create New or Select Existing Connection
- Step 6: Add Services and Events to the Application View
- Step 7: Perform Final Configuration
- Step 8: Test Services and Events
- Step 9: Publish Application View

Step 1: Log On to the Application Integration Design Console

The Application Integration Design Console displays all the application views in your WebLogic Integration environment, organized in folders. You must log in to view the console. For details about logging on to the Application Integration Design Console, see "Step 1: Log On to the Application Integration Design Console" on page 2-5.

Step 2: Define the Application Context for an Application View

Application view files are stored in a file system related to a WebLogic Workshop project. The first step in defining an application view for an adapter is to define the application context; you can either specify an existing project or define a new file repository for application view information.

For details about specifying a project, see "Step 2: Select an Application" on page 2-7.

Step 3: Add Folders

Folders are provided to help you organize and manage application views. Create or select an existing folder in which the application view will reside. For details about adding a folder, see "Step 3: Add a Folder" on page 2-8.

Step 4: Define Application View

Create a new application view for the appropriate adapter. An application view enables business processes to make use of the specified adapter's target EIS. For detailed information, see "Step 4: Define an Application View" on page 2-9.

Step 5: Create New or Select Existing Connection

Choose the type of connection factory to associate with the application view. You can select a connection factory within an existing instance of the adapter or create a connection factory within new adapter instance.

For details about creating and selecting connections, see the following topics:

- "Step 5A: Create a New Browsing Connection" on page 2-12
- "Step 5B: Reuse an Existing Browsing Connection" on page 2-15

Step 6: 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 6A: Add a Service to an Application View" on page 2-17
- "Step 6B: Add an Event to an Application View" on page 2-19

Step 7: Perform Final Configuration

Perform final configuration tasks that will allow you to test services and events. For details about performing final configuration tasks, see "Step 7: Perform Final Configuration Tasks" on page 2-21.

Step 8: 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 8A: Test an Application View's Services" on page 2-26
- "Step 8B: Test an Application View's Events" on page 2-29

Step 9: Publish Application View

Publish the application view to the target WebLogic Workshop application. For more information, see "Step 9: Publish an Application View" on page 2-31.

Using an Application View in a Business Process

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.

You can use application views in business processes in either of the following ways:

- By designing business processes in WebLogic Workshop
- By writing custom code

When an application view is used in your business process, the end result is a deployed electronic representation of your enterprise's business process. The business process uses the Application View control to specify the transactions to be performed by your applications to accomplish specific business tasks. The application views perform the transactions themselves.

Using an Application View in WebLogic Workshop

The most common way to use an application view in your enterprise's business is by designing a business process in WebLogic Workshop. A graphical user interface (GUI) is used for designing business processes. These business processes can include application view services and events accessed through the Application View control and the Message Broker Subscription control.

For detailed information about each task, see Chapter 3, "Using Application Views with Business Processes."

Using an Application View by Writing Custom Code

If you do not implement your business process model by using an application view through a business process, you must write custom Java code, instead. For instructions, see Chapter 5, "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 business process in WebLogic Workshop or by writing custom code. Any business operation can be implemented as a business process.

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 1-7.

When to Use WebLogic Workshop to Create a Business Process

Use WebLogic Workshop 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 WebLogic Workshop.

- Your business process requires periodic changes.

  WebLogic Workshop 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 Control in Business Processes

A business process developer can use an Application View control to provide users of BEA WebLogic Workshop with the means to interact with an EIS application. The interaction is implemented using a Java API. A business process 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 Application View controls, see the Building Integration Applications topic in the WebLogic Workshop Help at the following URL:

http://edocs.bea.com/workshop/docs81/index.html.

CHAPTER 2

Defining an Application View

This section presents the following topics:

- Before You Begin
- Application View Design Considerations
- High-Level Procedure for Defining an Application View
- Sample Detailed Procedure for Defining an Application View
- Editing an Application View
- Setting Transaction Timeout Values for Application Views
- Environment Variables for Application Views
- Database-Specific Error Messages

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 define an application view, make sure the following prerequisites are satisfied:

- The appropriate adapter has been purchased or has been developed using the ADK. You can create and configure application views only for existing, deployed 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.

Application View Design Considerations

Before you start defining an application view, please read the following design considerations:

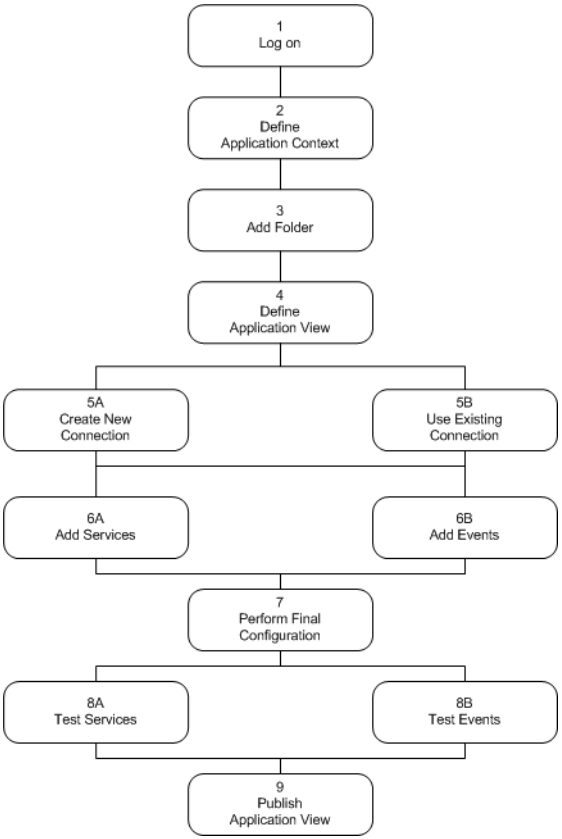
- **Use a single instance of the design console per client machine**—You should only have one instance of the Application Integration Design Console running on a single client machine. Running multiple consoles on a single machine may interfere with proper navigation between screens in your web browser.
- **Do not use underscores in application view service names**—Application views with services that are published to a WebLogic Workshop application must not contain underscores in the application view service names. Also, no underscores are allowed in the Application View control name.
- **Use short application view and folder names**—When you create folders and assign application view names, you are creating a path that describes a namespace, for example: `FolderOne.FolderTwo.MyAppView`. Limit the application view path to no more than 260 characters. Application view path names over 260 characters may cause errors in compiling the application view.

- **Assign the system administrator role to Microsoft SQL Server user accounts**—To deploy an application integration event for a Microsoft SQL Server database, the user account from which event generator tables are created must be assigned the system administrator role. If the user account is not assigned the system administrator role, the deployment fails and an error message is generated stating that the tables are invalid objects.

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 Integration Design Console. For detailed information, see "Step 1: Log On to the Application Integration Design Console" on page 2-5.

2. Select an existing application or specify a new application name and root directory. application view files are stored in a file system related to an application. For detailed information, see "Step 2: Select an Application" on page 2-7.

3. Add folders as required to help you organize application views. For detailed information, see "Step 3: Add a Folder" on page 2-8

4. Click **Add application view** to create a new application view for the appropriate adapter. An application view enables business processes to make use of the specified adapter's target EIS. For detailed information, see "Step 4: Define an Application View" on page 2-9.

5. Choose to create a new connection or to use an existing connection. For detailed information, see "Step 5: Establishing a Browsing Connection" on page 2-11.

   If you choose to create a new connection, you must enter application connection parameters. The information is validated, and the application view is configured to connect to the specified system For detailed information, see "Step 5A: Create a New Browsing Connection" on page 2-12.

   If you choose to use an existing connection, you must select an adapter instance and a connection factory. For detailed information, see "Step 5B: Reuse an Existing Browsing Connection" on page 2-15.

6. Add services and events that support business processes. For detailed information, see "Step 6: Add Services and Events" on page 2-17.

   Click **Add Event** or **Add Service** to define the appropriate events and services for this application view. For detailed information see "Step 6A: Add a Service to an Application View" on page 2-17 and "Step 6B: Add an Event to an Application View" on page 2-19.

7. Once you have finished adding events and services, display the **Connection Information** page to perform final configuration tasks. For detailed information, see "Step 7: Perform Final Configuration Tasks" on page 2-21.

8. Test all services and events to make sure they can properly interact with the target EIS application. For detailed information see "Step 8A: Test an Application View's Services" on page 2-26 and "Step 8B: Test an Application View's Events" on page 2-29.

   Once your services and events are tested and functioning, you are ready to publish the application view, allowing it to be used in business processes. For more information on using application views with business processes, see Chapter 3, "Using Application Views with Business Processes."

9. Publish the application view to the target WebLogic Workshop application. This allows business process developers within the target application to interact with the newly published application view using an Application View control. For more information, see "Step 9: Publish an Application View" on page 2-31.

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 Integration Design Console

The Application Integration Design Console displays all the application views in your WebLogic Integration environment, organized in folders.

Warning: You should only have one instance of the Application Integration Design Console running on a single client machine. Running multiple consoles on a single machine may interfere with proper navigation between screens in your web browser.

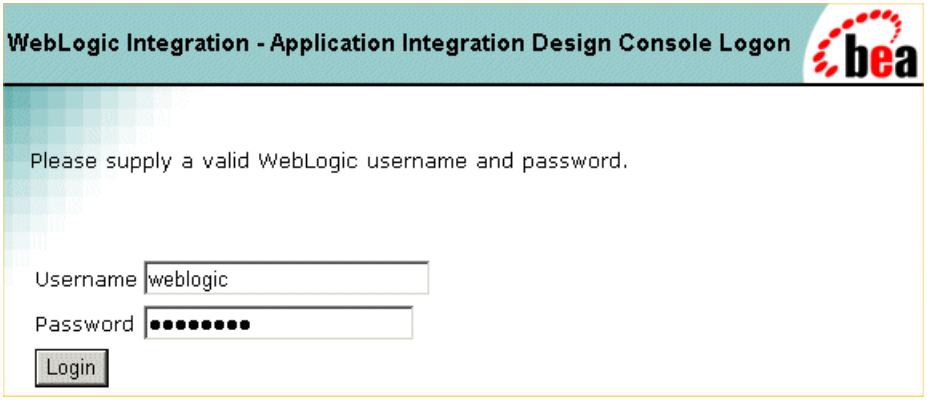
To log on to the Application Integration Design Console:

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

   `http://host:port/wlai`

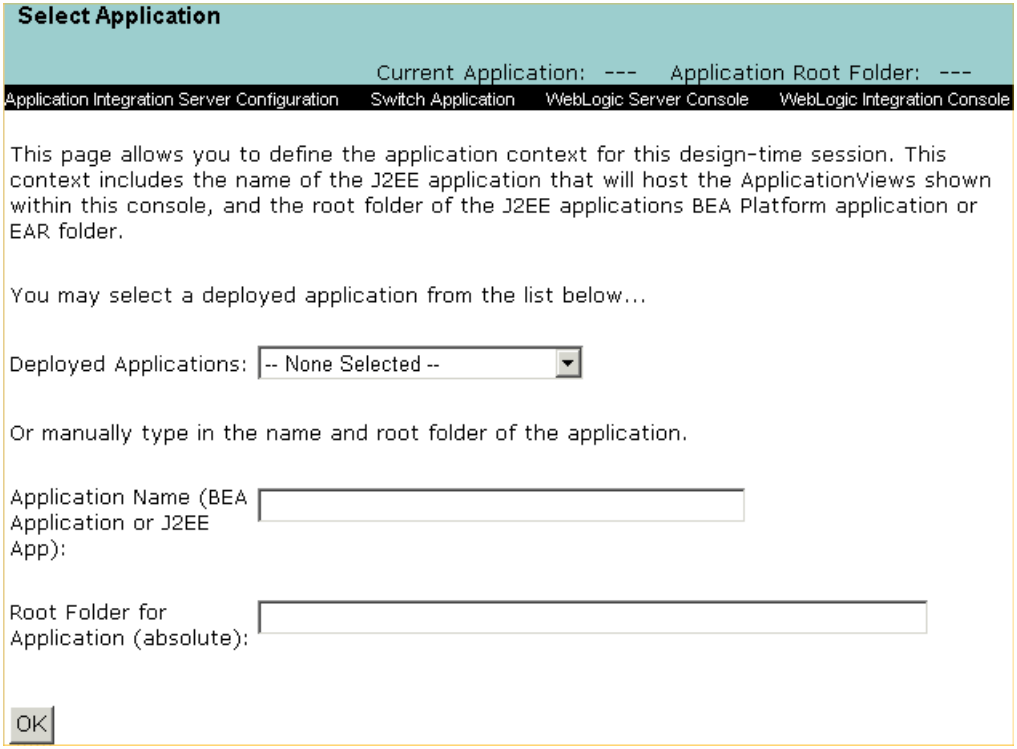
   Note: You can also invoke the Application Integration Design Console from WebLogic Workshop. Open a WebLogic Workshop application, ensure that WebLogic Server is running, and then choose Tools⇒WebLogic Integration⇒Application Integration Design Console.

   The **Application Integration Design Console Logon** page is displayed.



3. Enter your WebLogic Server username and password, then click **Login**.

   The **Select Application** page is displayed.



A menu bar appears at the top of each page which allows you to navigate to other pages and to administration consoles. The following table describes the menu items.

Menu Item	Action
Application Integration Server Configuration	Use the WebLogic Integration Administration Console to set application integration parameters. Open the WebLogic Integration Administration Console, select the System Configuration module, and then select the Application Integration module. See *Managing WebLogic Integration Solutions* for more information on setting application integration parameters.
Switch Application	Displays the Select Application page where you can choose a new application for the design-time session.
WebLogic Server Console	Invokes the WebLogic Server Administration Console.
WebLogic Integration Console	Invokes the WebLogic Integration Administration Console.
Glossary	Invokes a glossary of terms related to application integration. **Note:** The glossary is in progress and will be populated with more terms in a future release. For additional glossary terms, see the *WebLogic Platform Glossary*.
Logout	Logs you out of the current design session and displays the Logon page.

Step 2: Select an Application

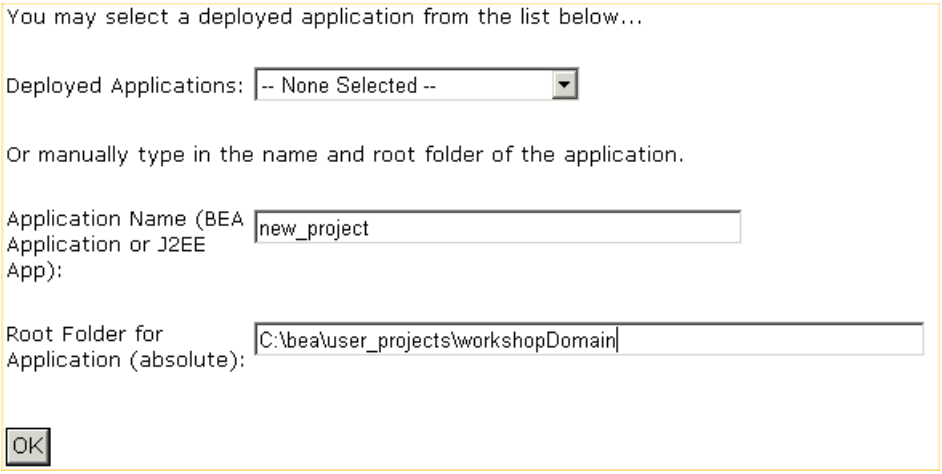
The Application Integration Design Console stores information about application views in a file-based repository. Before you create an application view, you must select an application to use in the design-time session. This determines where application view information is stored.

1. To choose an existing, deployed application, select an application from the **Deployed Applications** drop-down list.

   Note: WebLogic Workshop does not deploy an application until you add a project. You must add a project to the application in WebLogic Workshop before the application appears on the **Deployed Applications** drop-down list.

2. To create a repository for a new application, specify an application name and the root directory for the application. (You can also enter the application name and root directory for an existing, deployed application.)

   For example, the following figure shows the name and root directory for a new application.



3. Click **OK**.

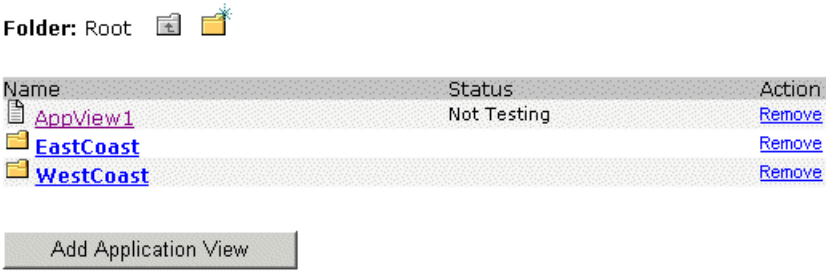
   The **Application View Design Console** page is displayed.

   Note: If you do not see the **Application View Design Console** page, consult the WebLogic Server administrator.

Step 3: Add 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.

The following figure shows the root folder with one application view and two subsidiary folders.



To create a folder:

1. To add a folder, click the new folder icon:

   The **Add Folder** page is displayed.

2. In the **New Folder** field, enter the folder 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 described in "Importing and Exporting Application Views" on page A-1.

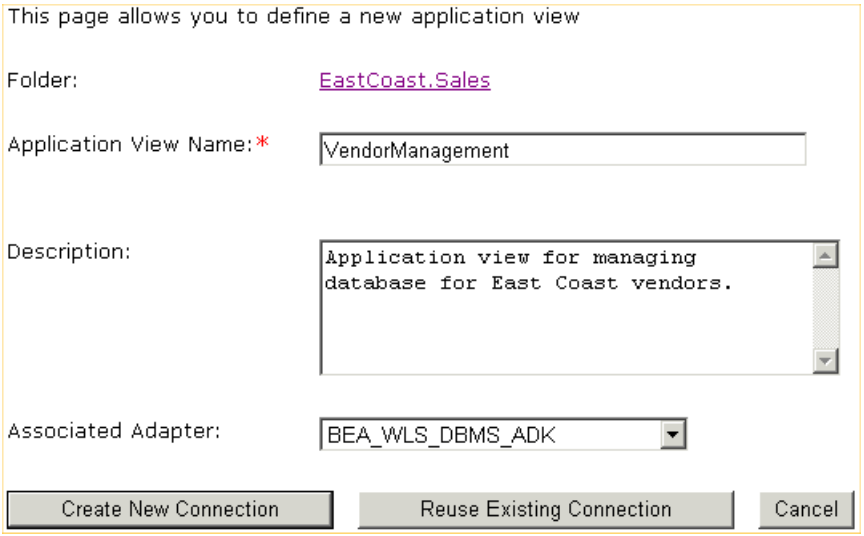
3. Click **Save**. The new folder appears on the **Application View Design Console** page

Step 4: Define an Application View

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.

   The red asterisk next to the **Application View Name** field indicates that this is a required field.

   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 connecting to this application view from business processes created in WebLogic Workshop.

4. From the Associated Adapter list, select an adapter to be used to create this application view.

   Note: You can define an application view without selecting the adapter type. This allows for instances where you realize that an application view is needed, but do not have sufficient information at the time to fully define the application view. You can create, name, and enter a description of the required application view without selecting the adapter type or further defining the application view. When more information is available, you can continue defining the application view.

5. Click **Create New Connection** to create a new browsing connection.

The **Create New Browsing Connection** page is displayed, as described in "Step 5A: Create a New Browsing Connection" on page 2-12.

6. Click **Reuse Existing Connection** to use an existing browsing connection.

   The **Select Browsing Connection** page is displayed, as described in "Step 5B: Reuse an Existing Browsing Connection" on page 2-15.

7. Click **Cancel** to return to the previously displayed folder.

   The application view is not created.

Step 5: Establishing a Browsing Connection

You must choose the type of connection factory to associate with the application view. You can create a connection factory within a new adapter instance or select a connection factory within an existing instance of the adapter.

An adapter instance acts as a communications gateway between WebLogic Server and your EIS. This gateway can be unidirectional or bidirectional. Adapter instances include inbound messaging capabilities (from the EIS to WebLogic Server) that supports events, and connection factories that support services (outbound requests from WebLogic Server). An application view uses an adapter instance as a gateway for one of three purposes:

- design-time browsing
- event delivery
- service invocation

You must either create a new adapter instance or select an existing adapter instance that the application view will use. To enable design-time browsing and service invocation, you must also designate which connection factory the application view should use to get the connections used to communicate with the EIS.

When you choose to create a new connection, WebLogic Integration creates the adapter instance for you, and the connection parameters you define are used to create the connection factory within the adapter instance. For more information, see "Step 5A: Create a New Browsing Connection" on page 2-12.

When you choose to reuse an existing connection, you select the adapter instance and factory by name and do not need to specify connection parameters directly. If you do not know the proper parameters required to connect to a given EIS, have an EIS specialist set up the adapter instance and connection factory to use and the select them from the Application Integration Design

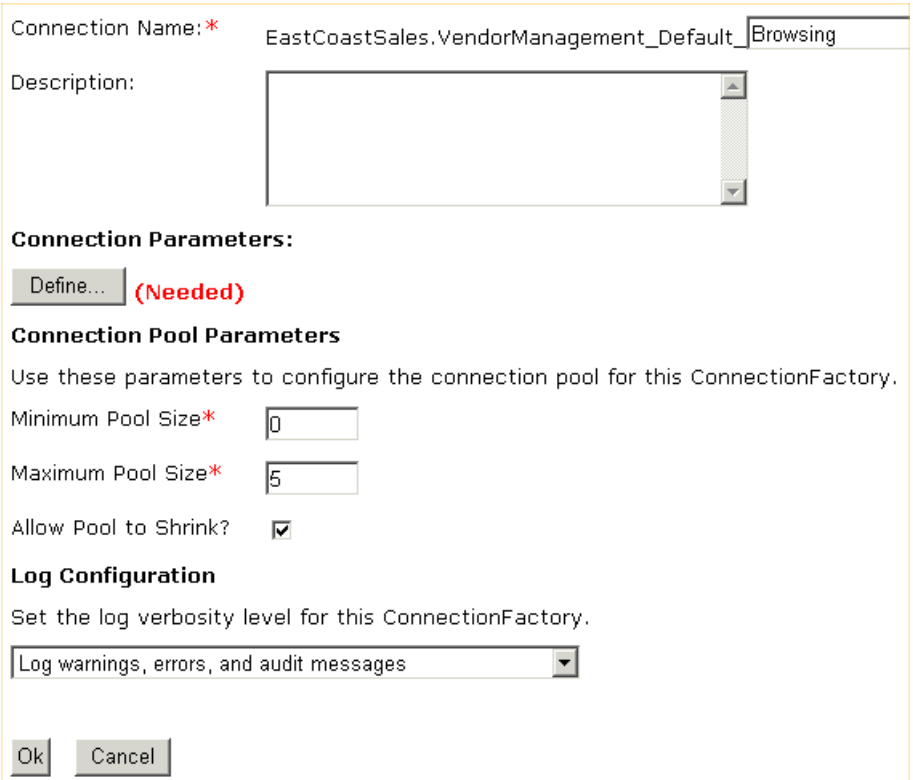
Console. For more information, see "Step 5B: Reuse an Existing Browsing Connection" on page 2-15.

When establishing connections, the Application Integration Design Console is optimized for ease of use by first-time users. When you create a new connection, the browsing connection is also assigned as the service connection. This saves the first-time user a step in the process. More advanced users are likely to reuse an existing connection. In this case, the user is more familiar with the specific connection requirements and must separately assign the service connection.

Step 5A: Create a New Browsing Connection

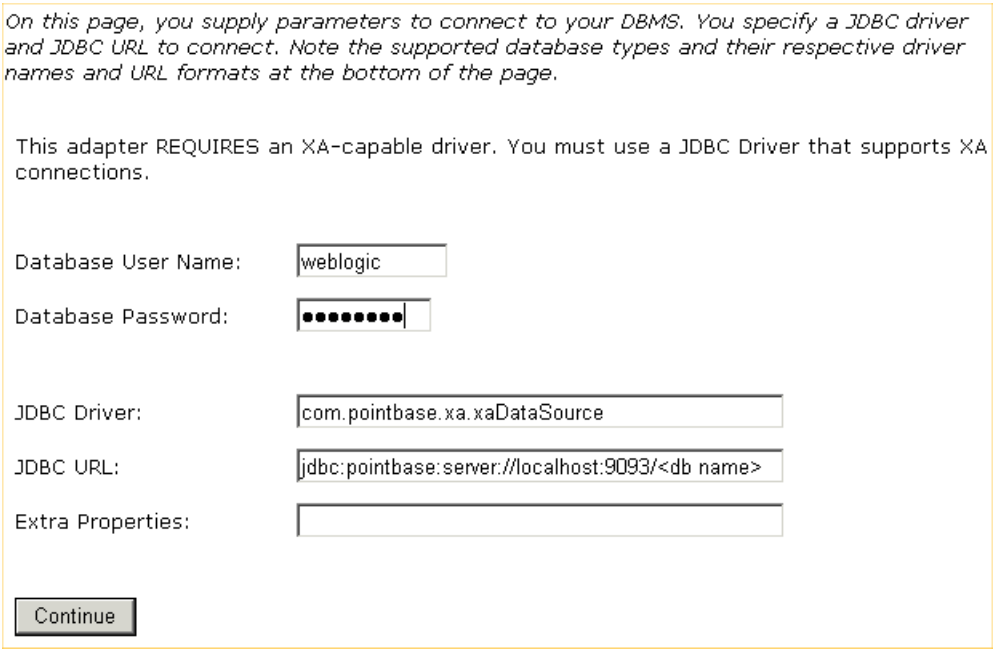
1. You use the **Create New Browsing Connection** page to configure the connection properties, pool parameters, log level, and sign-on behavior of a new connection factory. This creates a new adapter instance and connection factory within it. The adapter instance and connection factory are given default names based on the application view name.

   The **Create New Browsing Connection** page indicates whether you need to define connection parameters. If the connection needs to have adapter-specific properties set, it indicates this with a Needed label next to the Define button. This label is removed once you provide connection parameters.



2. In the **Connection Name** field, accept the default connection name suffix. (The connection names are assigned by the system and are not editable.)
3. In the **Description** field, enter a description for the new connection.
4. Click **Define** to define the connection parameters for the browsing connection.

   The **Configure Connection Parameters** page is displayed.



5. Enter the database username and password.
6. 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.

   Note: The fields displayed on the page you see may differ from those shown here. Which fields are displayed is determined by the adapter. For the required information for any remaining fields, refer to your adapter documentation. For BEA WebLogic Adapters, see "Defining Service Connection Parameters" in your adapter User Guide, available at the following URL:

   http://edocs.bea.com/wladapters/docs81/index.html

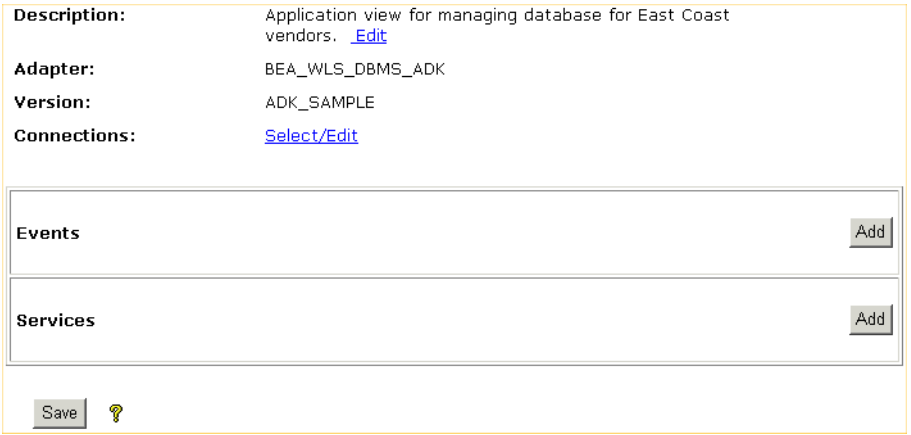
   If you are using an adapter that uses a JDBC DataSource, do not create two Tx Data Sources that point to the same connection pool. If a transaction uses two different Tx Data Sources which are both pointed to the same connection pool, you will get an XA_PROTO error when you try to access the second connection.

   For more information on JDBC DataSources and WebLogic Server, see "JDBC DataSources" in *Administration Console Online Help* at the following location:

http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc_datasources.html

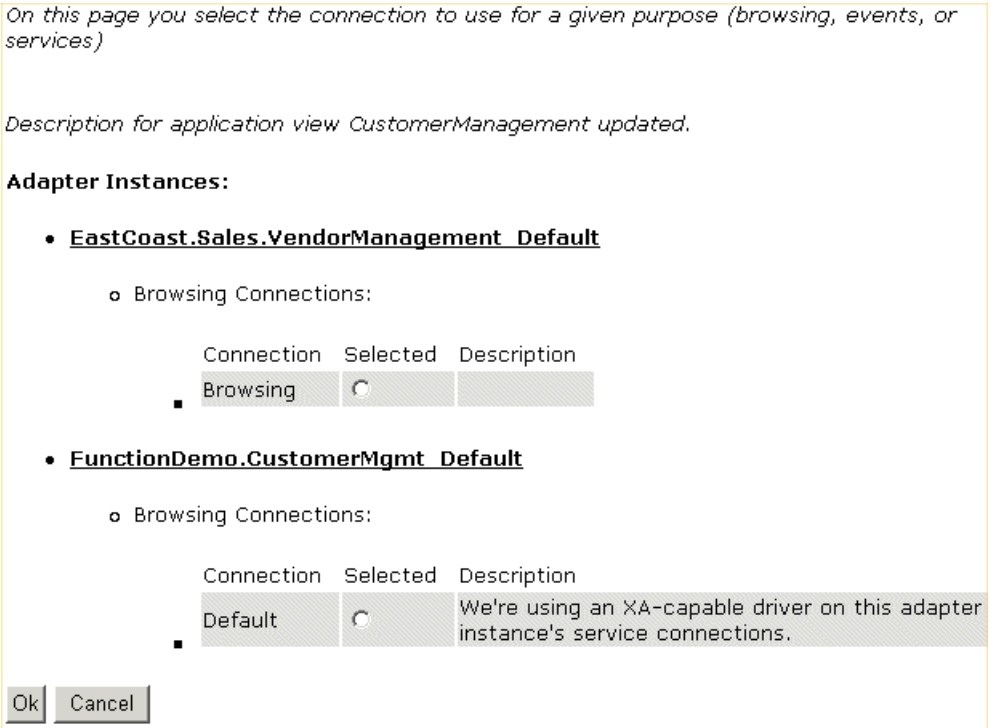
7. Click **Continue** in the **Configure Connection Parameters** page to return to the **Create New Browsing Connection** page.
8. Edit the connection pool and log parameters as required.
9. Click **OK**.

   The **Application View Administration** page is displayed. On the **Application View Administration** page, you can add services and events to your application view.



Step 5B: Reuse an Existing Browsing Connection

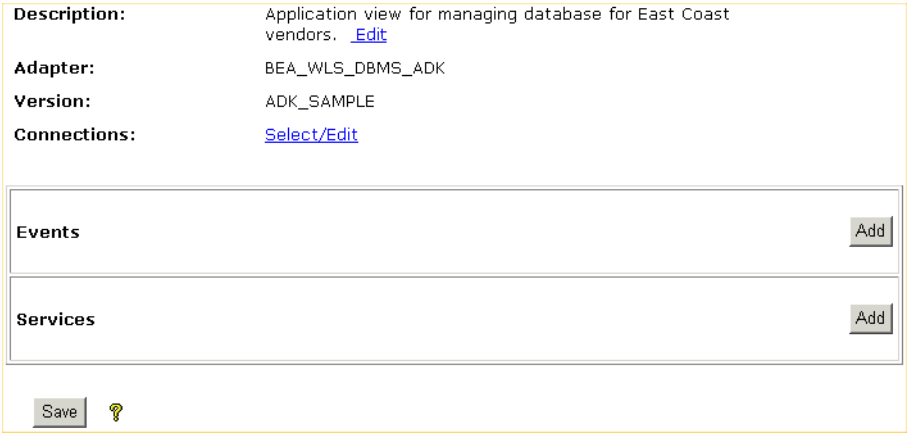
You use the **Select Browsing Connection** page to choose an existing connection. You select both the adapter instance and a connection factory within it, from which connections to the EIS are obtained. These connections are used for browsing your EIS during design-time editing sessions.



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.

1. Choose the Selected radio button next to the connection factory you want to reuse.
2. Click **OK**.

   The **Application View Administration** page is displayed. On the **Application View Administration** page, you can add services and events to your application view.



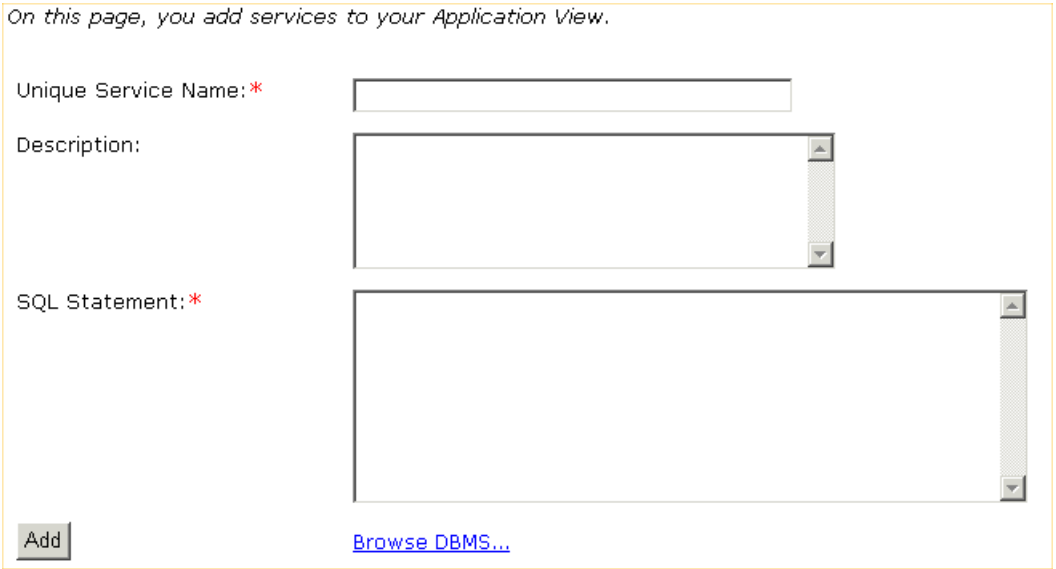
Step 6: Add Services and Events

The **Add Service** and **Add Event** pages allow you to add services and events that support specific business processes. An application view can have multiple services and events. The required business processes determine the types of services and events you include in your application views. Work with your business analyst to define the application's business requirements. Once you determine the necessary business processes, you can define the appropriate services and events.

Step 6A: 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 business processes created in WebLogic Workshop.

4. For the required information for any remaining fields, consult the relevant technical analyst or EIS specialist, or refer to your adapter documentation.

   Note: For BEA WebLogic Adapters, see "Setting Service Properties" in your adapter User Guide, available at the following URL:

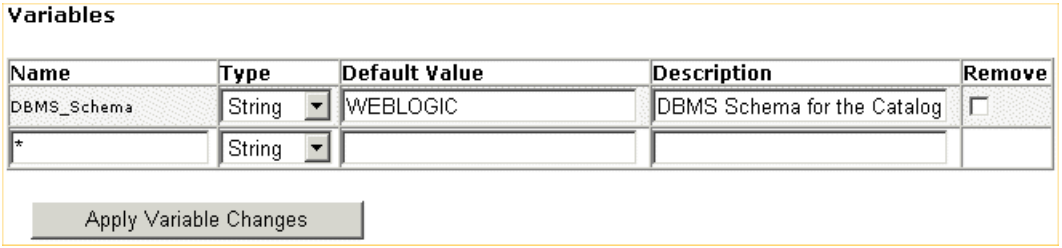
   `http://edocs.bea.com/wladapters/docs81/index.html`

   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 WEBLOGIC.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. You can optionally specify environment variables for the application view in the Variables area in the lower portion of the **Add Service** page. For more information on environment variables, see "Environment Variables for Application Views" on page 2-34.



To add an environment variable:

a. Enter a name for the environment variable in the **Name** field.

b. Select a data type for the variable from the **Type** drop-down list.

c. Optionally specify a default value and a description for the variable.

6. When finished, click **Add**.

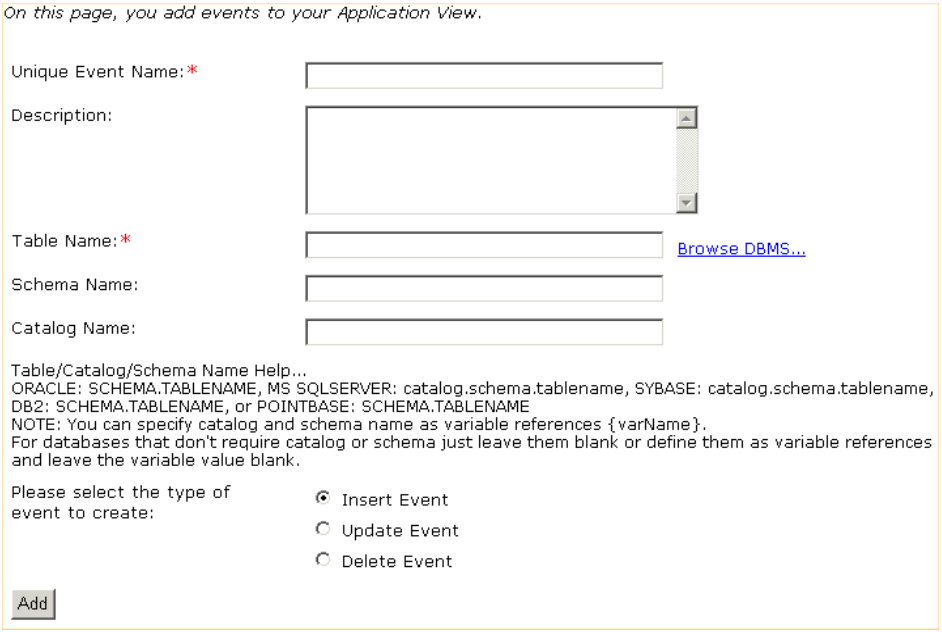
   The **Application View Administration** page is displayed. Note that the new service is shown along with links which allow you to edit, remove, and view information about the service.

7. If you are finished adding services and do not plan to add any events, click **Save** to save the current application view information. You must create or select connections for services, as described in "Step 7: Perform Final Configuration Tasks" on page 2-21, before testing and publishing the application view.

Step 6B: Add an Event to an Application View

1. On the **Application View Administration** page, 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.

2. 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.

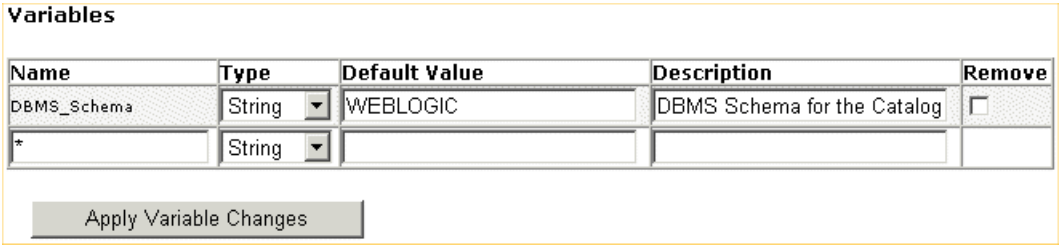
3. In the **Description** field, enter any notes that may be helpful to people using this application view in business processes created in WebLogic Workshop.

4. For the required information for any remaining fields, consult the relevant technical analyst or EIS specialist, or refer to your adapter documentation.

   Note: For BEA WebLogic Adapters, see "Setting Event Properties" in your adapter User Guide, available at the following URL:

   `http://edocs.bea.com/wladapters/docs81/index.html`

5. You can optionally specify environment variables for the application view in the Variables area in the lower portion of the **Add Event** page. For more information on environment variables, see "Environment Variables for Application Views" on page 2-34.



To add an environment variable:

a. Enter a name for the environment variable in the **Name** field.

b. Select a data type for the variable from the **Type** drop-down list.

c. Optionally specify a default value and a description for the variable.

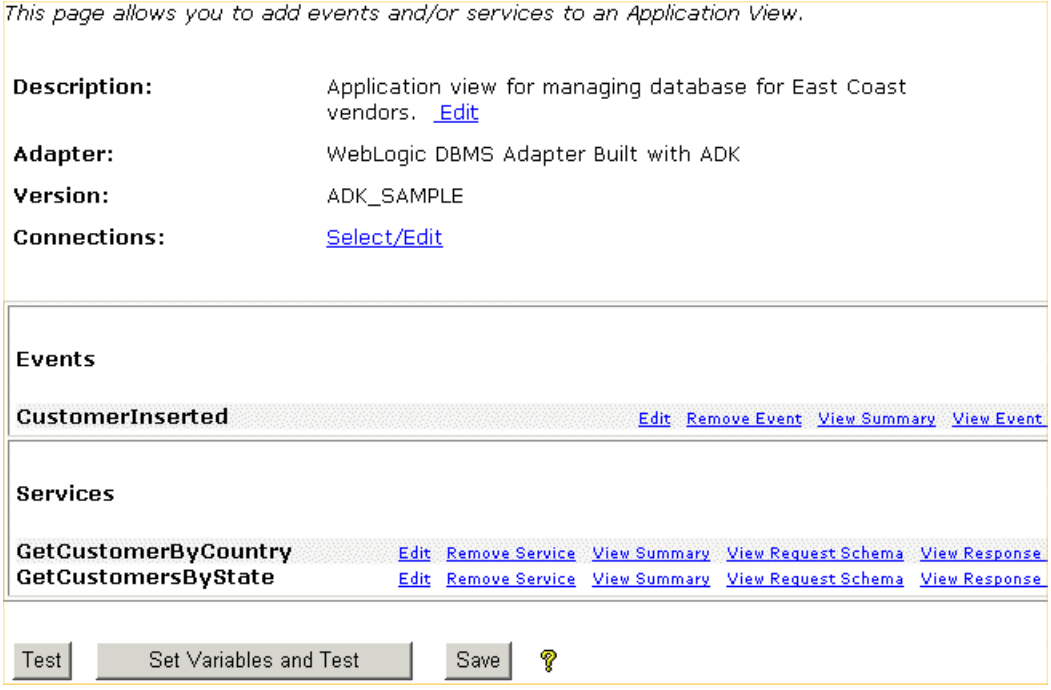
6. When finished, click **Add**.

   The **Application View Administration** page is displayed. Note that the new event is shown along with links which allow you to edit, remove, and view information about the event.

7. If you are finished adding services and events, click **Save** to save the current application view information. You must create or select connections for services and events, as described in "Step 7: Perform Final Configuration Tasks" on page 2-21, before testing and publishing the application view.

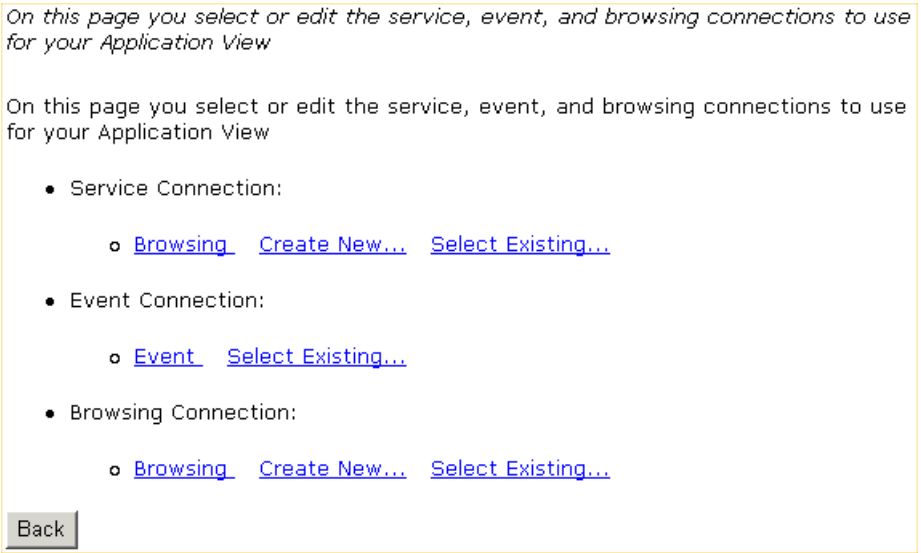
Step 7: Perform Final Configuration Tasks

Once you have finished adding services and events and have saved your application view, you must complete final configuration of connections before testing services and events. These configuration tasks are performed from the **Application View Administration** page.



1. Click **Select/Edit**.

   The **Connection Information** page is displayed.



The **Connection Information** page is organized by connection type: service invocation, event delivery, and design-time browsing connections. (For more information on how application views use adapter instances and connection factories, see "Step 5: Establishing a Browsing Connection" on page 2-11.)

2. To edit the connection for services and design-time browsing, click **Browsing**. (This is the browsing connection you defined in "Step 5A: Create a New Browsing Connection" on page 2-12.)

   The appropriate **Edit Connection** page is displayed. Note that only one connection is used for services and browsing and that changes made on the one **Edit Connection** page will appear on the other **Edit Connection** page.

3. To create a new connection for services and design-time browsing, click **Create New**. (This is the connection you defined in "Step 5A: Create a New Browsing Connection" on page 2-12.)

   The appropriate **Create New Connection** page is displayed.

   If no connection parameters have been set for a connection and the Needed label is displayed nest to the Define button, the connection parameters page is displayed when you click **OK** on either the event or service connection page. When you click **Continue**, the **Event** or **Service Connection** page is displayed without the **Needed** label.

Note: For BEA WebLogic Adapters, see "Defining Service Connection Parameters" in your adapter User Guide, available at the following URL:

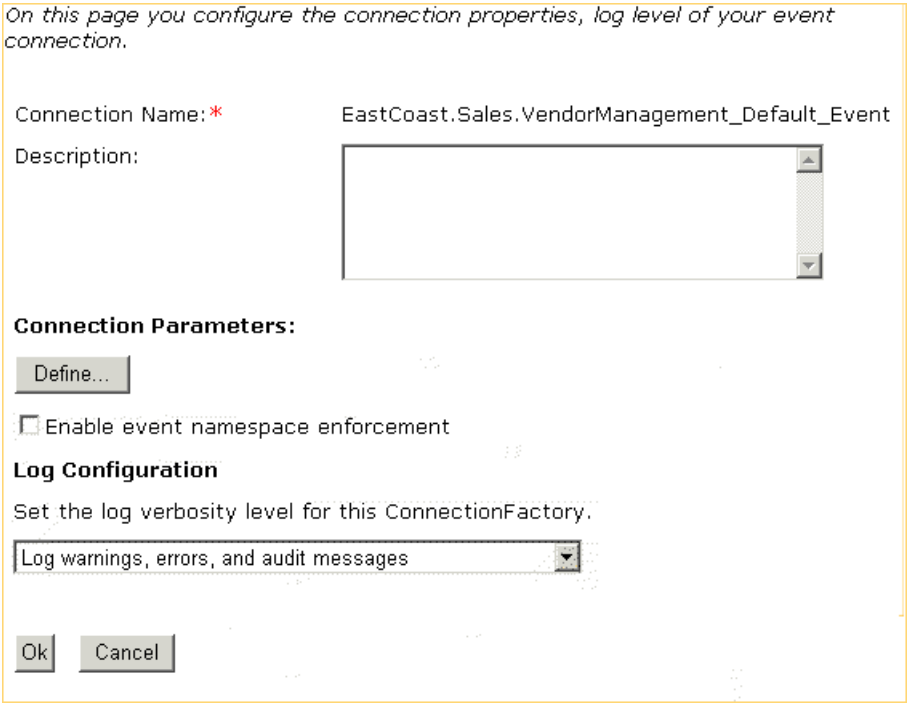
`http://edocs.bea.com/wladapters/docs81/index.html`

4. To select an existing connection that is different from the current connection, click the **Select Existing...** link.

   The appropriate **Select Connection** page is displayed. This page is the same as the page described in "Step 5B: Reuse an Existing Browsing Connection" on page 2-15.

5. Click **Event** to configure the connection properties, enable or disable the namespace enforcement option, and set the log level for your event connection.

   The **Edit Event Connection** page is displayed.



The **Edit Event Connection** page allows you to define event connection parameters, enable or disable the namespace enforcement option, and specify what information will be logged for the connection factory.

6. To enable namespace enforcement, check the **Enable event namespace enforcement** check box.

   Enable the namespace enforcement option to indicate that the client requires response documents and event documentst to declare the namespace indicated in the response/event definitions, and force the proper namespace declaration onto the response/event if needed.

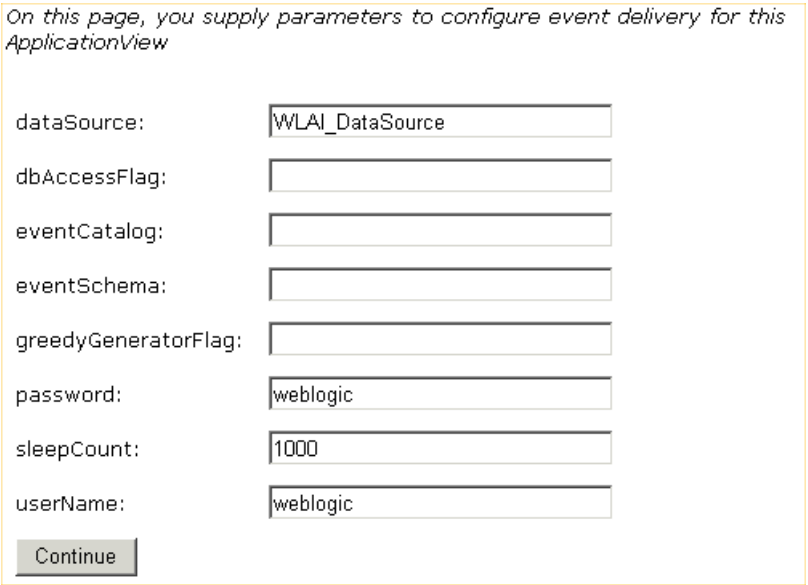
   Some legacy adapters do not provide responses/events using the namespace declared in the response/event schema This option allows clients that perform schema-based XML checking to use such adapters.

   Note: If the adapter returns raw XML text (not parsed), enabling namespace enforcement has serious performance implications because it forces a parse of the XML text in order to inject the proper namespace declaration.

7. Select one of the following settings for the log:
   - Log errors and audit messages
   - Log warnings, errors, and audit messages
   - Log informational, warning, error, and audit messages
   - Log all messages

8. Locate Connection Parameters and click **Define** to set the event delivery parameters.

   The **Configure Event Delivery Parameters** page is displayed.



The event delivery parameters you enter on this page enable connection to an EIS instance and are used when generating events. The properties are specific to the associated adapter and are defined in the `wli-ra.xml` file within the base adapter.

Note: For BEA WebLogic Adapters, see "Defining Event Connection Parameters" in your adapter User Guide, available at the following URL:

`http://edocs.bea.com/wladapters/docs81/index.html`

9. After you have set the event delivery parameters, click **Continue** to return to the **Edit Event Connection** page and then click **OK** to return to the **Connection Information** page.
10. Click **Back** to return to the **Application View Administration** page.

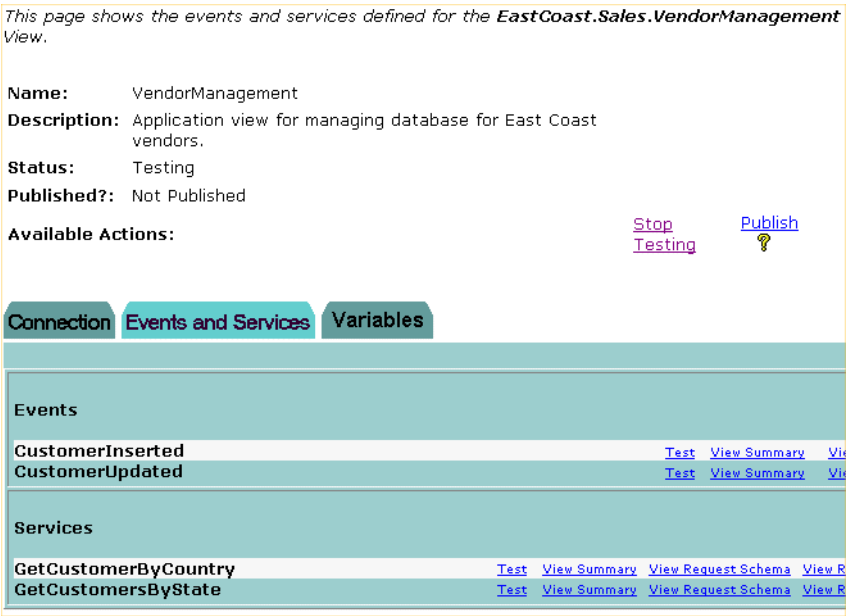
Step 8A: Test an Application View's Services

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 **Application View Administration** page, click **Test**.

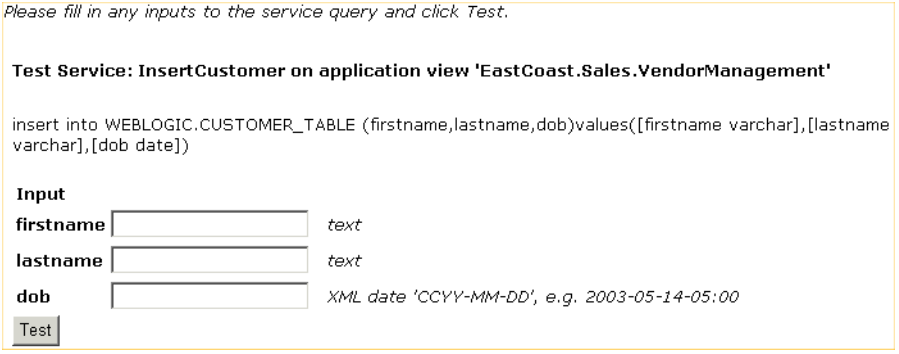
   The **Summary for Application View** page is displayed. Note that the **Status** is **Testing** and a **Stop Testing** link is displayed.

You can optionally click **Set Variables and Test.** The **Set Variables and Test Application View** page is displayed. You can edit the values of environment variables and then click **Test** to test the application view.



2. In the Services area of the Events and Services tab, find the appropriate service and click **Test**.

   The **Test Service** page is displayed.



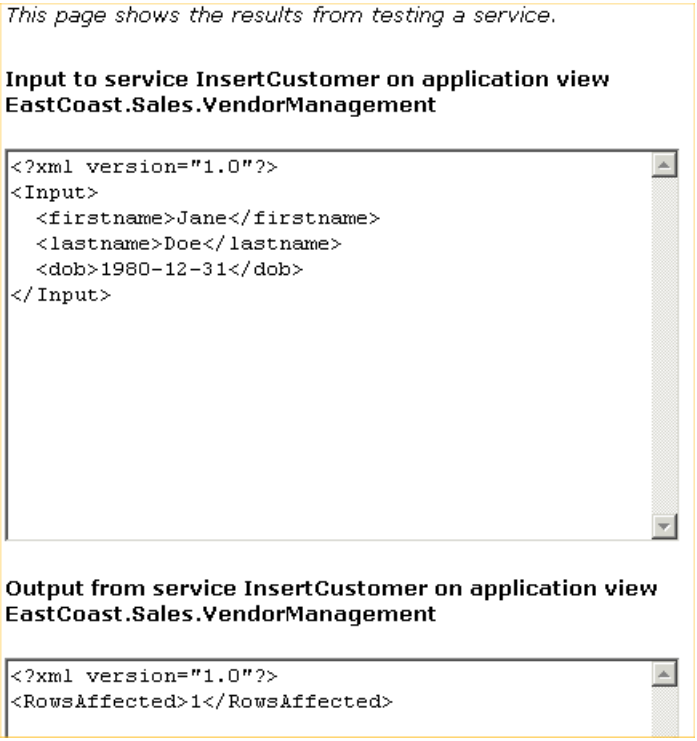
3. If necessary, enter the required data in the appropriate fields.

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. For BEA WebLogic Adapters, see "Testing Services" in your adapter User Guide, available at the following URL:

http://edocs.bea.com/wladapters/docs81/index.html

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. If the service fails, the failure will be shown in the response field.



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 publish the application view or, if you want to edit it, you may return to the **Summary** page, click **Stop Testing**.

Step 8B: Test an Application View's Events

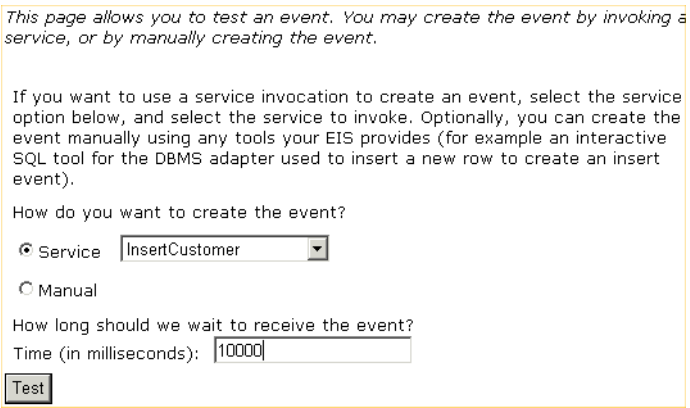
The purpose of testing your application view events is to evaluate whether or not the application view responds correctly to the EIS application. You can test an application view only if it is deployed and it contains at least one event or service. To test an application view event:

1. In the **Application View Administration** page, click **Test**.

   The **Summary for Application View** page is displayed. Note that the **Status** is **Testing** and a **Stop Testing** link 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. For BEA WebLogic Adapters, see your adapter User Guide, available at the following URL:

   `http://edocs.bea.com/wladapters/docs81/index.html`

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 2-30.

   – 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 2-30.

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** drop-down list, select a service that triggers the event you are testing. For example, if you are testing the CustomerInserted event, select a service that invokes it, such as InsertCustomer.
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.

   Note: For BEA WebLogic Adapters, see "Testing Events Using a Service" in your adapter User Guide, available at the following URL:

   `http://edocs.bea.com/wladapters/docs81/index.html`
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. Do one of the following:
   - If the test succeeds, repeat the test procedure for each remaining event you want to test.
   - If the test fails, you have several options for diagnosing the problem. The test may fail for a number of reasons. If the service fails, the failure is shown in the response field. If the event does not arrive even though the service succeeded, make sure that you invoked the service that is known to create the selected event. You can also check the server log for error messages or contact your system administrator.
6. 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.

   Note: For BEA WebLogic Adapters, see "Testing Events Manually" in your adapter User Guide, available at the following URL:

   `http://edocs.bea.com/wladapters/docs81/index.html`

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. Do one of the following:

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

   – If the test succeeds, repeat the test procedure for each remaining event you want to test.

6. When you are finished, save the application view.

Step 9: Publish an Application View

From the **Summary** page, you can publish an application view that has been tested. This generates an EJB within the application directory, publishes schema files to the WebLogic Workshop application for use in the XML Mapper, and makes the application view visible within WebLogic Workshop. The Application View control wizard browses only published application views.

When you publish an application view, it becomes a part of the WebLogic Workshop application. After publishing, the application view is controlled by the WebLogic Workshop application rather than the Application Integration Design Console. You must republish the application view if you edit it in the Application Integration Design Console.

Note: You must recreate any Application View control that uses an application view that has been modified in the Application Integration Design Console and republished.

When the application view has been published, a message similar to the following is displayed on the **Summary** page:

> EastCoast.Sales.CustomerManagement published to:
> C:\bea\weblogic81\domains\samples\integration\sampleApp\
> sampleApp_EastCoast_Sales_CustomerManagement_ApplicationView-ejb.jar

All application integration artifacts (application views, schemas, and namespaces) are contained in the application view EJB. The EJB is rebuilt as necessary to reflect changes in the application integration artifact source files. The source files are maintained in the `wlai-repository` directory under the EAR root directory. When an application view is published, the schemas needed by the application view are copied into the `Schemas/wlai` directory in the WebLogic Workshop application.

When you publish an application view, testing is automatically stopped before the application view is published. This ensures that your temporary test application view deployments are removed from the server and do not interfere with your published application view.

Note: When you publish an application view, schemas are published to the WebLogic Workshop application. Publishing an application view and opening WebLogic Workshop may cause the Schema project to be rebuilt. If you edit a business process (for example, to define a start node) before the Schemas project has finished rebuilding, you may see an error message similar to the following:

```
The type for the variable could not be resolved. This may be due to
not entering fully qualified type name. Please go to source to correct
errors.
```

Make sure that the Schema project has completed its rebuild before editing the business process.

Published application views can be monitored and managed using the WebLogic Integration Administration Console. For more information on managing application views and adapter instances, see *Managing WebLogic Integration Solutions*.

Note: When you delete a previously published application view, associated `wlai.channel` file, and EJB file, the application view ID still appears in the WebLogic Integration Administration Console and is shown in an Undeployed state after you rebuild the application. The deleted application view ID is removed from WebLogic Integration Administration Console when you reboot the system.

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 the **Edit** link under **Available Actions**.

   The **Application View Administration** page is displayed.

3. To reconfigure the application view's connection parameters, select the **Select/Edit** link under Connections.

   The **Connection Information** page is displayed. Follow the instructions in "Step 7: Perform Final Configuration Tasks" on page 2-21.

4. To add services or events, click **Add** in the **Services** or **Events** area. Follow the instructions in "Step 6A: Add a Service to an Application View" on page 2-17 or "Step 6B: Add an Event to an Application View" on page 2-19.

Setting Transaction Timeout Values for Application Views

The current default JTA transaction timeout for domains generated by the Configuration Wizard is 30 seconds. Typically, many adapter calls can exceed 30 seconds. You can increase the server's default transaction timeout using the WebLogic Server Administration Console:

1. Open the WebLogic Server Administration Console and log in.

2. In the left pane, select **Services⇒JTA**.

   The **JTA configuration** page is displayed.

3. In the right pane, edit the **Timeout Seconds** field to reflect a longer timeout value. The suggested value is 500 seconds.

To implement the increased timeout value, you must republish your application views or manually change the `weblogic-ejb-jar.xml` descriptor in your previously published application views. To manually increase the value, insert the following into

`weblogic-ejb-jar.xml` in the `weblogic-enterprise-bean` element for both the stateless and stateful session EJBs for the application view:

```
<transaction-descriptor>
   <trans-timeout-seconds>500</trans-timeout-seconds>
</transaction-descriptor>
```

Environment Variables for Application Views

Application views contain metadata about associated services and events. This metadata often contains references to environment-specific resources or data values that may need to be updated dynamically. Environment variables allow these values to be updated dynamically, without having to undeploy, edit, and redeploy the application view. Once an environment variable is created, its value can be specified at time of deployment or dynamically at runtime.

Variable definitions will contain the following:

- Variable name
- Data type
- Default value (optional)
- Description (optional)

You can define new variables when adding or editing services or events. The list of existing variables is also displayed and can be edited on the **Application View Administration** and **Application View Summary** pages.

Once the application view has been published, an administrator can display environment variables and edit their values in the WebLogic Integration Administration Console. An administrator can not add or delete environment variables.

Database-Specific Error Messages

When using your published application views, you may see database-specific error messages written to the log file. The section identifies some of the more common database-specific error messages.

From time to time, when using Sybase or MSSQL databases, warnings are issued stating that the active database and/or language has been changed. These warnings come from the Sybase and MSSQL databases when the active database or language is changed on a connection. Since the

catalog is changed at various points in the DBMS sample adapter, users of the sample adapter will inevitably see these messages. These warnings are harmless and can be ignored.

When an adapter instance in an MSSQL XA environment is automatically suspended and resumed, error messages similar to the following are thrown.

```
<Oct 15, 2003 4:40:30 PM PDT> <Error> <JDBC> <BEA-001112>
<Test "SELECT COUNT(*) FROM sysobjects" set up for pool "wlaiPool" failed with
exception: "javax.transaction.xa.XAException: [BEA][SQLServer JDBC Driver]
No more data available to read.".>
```

This is a JDBC-level error, generated as the JDBC container cleans up existing/dead connections to the restarted DBMS instance. This type of exception is normal in this case; the EIS is available, the adapter instance is resumed successfully, and the JDBC container recovers.

When using an application view with a Microsoft SQL Server or Sybase database, use the WebLogic Server Administration Console to enable the `TestReservedConnection` parameter for the connection pool used for application integration. If the parameter is not enabled, the auto resume or manual resume features do not work and a SQLException similar to the following is thrown:

```
java.sql.SQLException: [BEA][SQLServer JDBC Driver]No more data
available to read
```

CHAPTER 3

Using Application Views with Business Processes

This section presents the following topics:

- Before You Begin
- Integrating Application Views and Business Processes Using a Control
- Sample Application View Control Files
- Receiving Events
- Handling Application View Local Transactions in Business Processes

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 WebLogic Workshop to design business processes that use your application view services and events.

WebLogic Workshop provides a graphical user interface (GUI) for designing business processes as well as web services. These business processes can include application view services and events defined using the Application Integration Design Console.

Business Processes are integrated with application views through the Application View control. The Application View control allows a business process engineer to browse the hierarchy of application views, invoke a service as a business process action, and to start a new business process when an event occurs. For details, see the Application View Control topic in the WebLogic Workshop Help at the following URL:

http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsAppView.html

Before you can invoke an application view service or receive an application view event in a business process, you must make sure the following prerequisites have been met:

- You have created an application view, defined services and events for it, and published the application view as described in "Defining an Application View" on page 2-1.
- The application view and its adapter are functional and saved. If you plan to call application view services and events from a running business process, the application view must be deployed, as well.
- WebLogic Workshop and application integration functionality are available.
- You have received information about the required business logic for the business processes you are defining from the appropriate business analyst.

Integrating Application Views and Business Processes Using a Control

The Application View control allows users of WebLogic Integration, WebLogic Workshop, and WebLogic Portal to invoke functions within enterprise applications using simple Java APIs or XML documents. This allows users who are not experts in the use of a given enterprise system to use its capabilities in a manner a Java programmer can understand.

The Application View control provides a means for a business process developer to invoke application view services both synchronously and asynchronously. Synchronous services are represented as simple methods with a single parameter and a non-void return value. Asynchronous services are represented as both a method with a single parameter (the request), and a callback method with a single parameter (the response).

Application view events are delivered through the Message Broker. To receive application view events, use a Message Broker Subscription control or static subscription (from business processes only). For more information on handling events, see "Receiving Events" on page 3-6.

Perform the following steps to use an application view from within a business process:

1. Define an application in WebLogic Workshop. For more information, see the WebLogic Workshop Help.

   Any WebLogic Workshop application which uses the application integration capabilities of WebLogic Integration must contain a project explicitly named Schemas. The Schemas project is used to store the `wlai.channel` file and application view schemas (published as XML Bean classes). If the Schemas project does not exist in the application, you must create it before publishing application views.

2. Select the WebLogic Workshop application in the Application Integration Design Console using the drop-down application list. For more information, see "Step 2: Select an Application" on page 2-7.

3. Define the application view and add services and events. For more information, see "Step 4: Define an Application View" on page 2-9 through "Step 6B: Add an Event to an Application View" on page 2-19.

4. Test the application view. For more information, see "Step 8A: Test an Application View's Services" on page 2-26 and "Step 8B: Test an Application View's Events" on page 2-29.

5. Publish the application view using the **Publish** button on the **Application View Summary** page. This generates an application view EJB at the root of the WebLogic Workshop application. For more information, see "Step 9: Publish an Application View" on page 2-31.

   The published application view EJB is added to the Modules list in the WebLogic Workshop application tree. This deploys the application view EJB and, in turn, the application view.

   Note: When you publish an application view, schemas are published to the WebLogic Workshop application. Publishing an application view and opening WebLogic Workshop may cause the Schema project to be rebuilt. If you edit a business process (for example, to define a start node) before the Schemas project has finished rebuilding, you may see an error message similar to the following:

   ```
   The type for the variable could not be resolved. This may be due
   to not entering fully qualified type name. Please go to source to
   correct errors.
   ```

   Make sure that the Schema project has completed its rebuild before editing the business process.

6. Add an Application View control to your business process using the **New⇒Integration Controls⇒ApplicationView** option in the Data Palette. For more information, see the Application View Control topic in the WebLogic Workshop Help at the following URL:

   http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsAppView.html

7. Test your business process. For more information, see the WebLogic Workshop Help.

When you create an Application View control, you specify or select the following:

- A variable name associated with the control instance.
- Whether to use an existing Application View control or define a new control.
  - If you choose to use an existing Application View control, the associated application view and service are already defined in the JCX file for that control.
  - If you choose to create a new control, you must specify a name for the new JCX file. You then browse for the desired application view and select a service that is asynchronously invoked by the control.

Sample Application View Control Files

Application View control instances are stored as JCX files. Each Application View control instance is a Java interface that extends the base Application View control interface (`com.bea.wlai.control.ApplicationViewControl`).

Application View Control Interface

The following code shows the base Application View control interface.

Listing 3-1 Application Control Base Interface

```
package com.bea.wlai.control;

import com.bea.control.XMLControl;
import com.bea.control.Extensible;
import com.bea.xml.XmlObject;

/**
* ApplicationView Control base interface
*/
public interface ApplicationViewControl
```

```
extends XMLControl
{
  // -------------------------
  // Async methods
  // -------------------------

  /**
  * Is this control instance currently processing an async request?
  */
  public boolean isAsyncRequestActive();

  /**
  * Return the ID of the active asynchronous request being processed by
  * this control instance, or null if no async request is active.
  */
  public String getActiveAsyncRequestID();

  // -------------------------
  // Local transaction methods
  // -------------------------

  /**
  * Begin a local transaction on this control. This will begin a local
  * transaction on the underlying ApplicationView instance. All work done
  * by this control instance between this call, and a call to
  * commitLocalTransaction() or rollbackLocalTransaction() will be
  * committed or rolled back, respectively, as a unit. If the underlying
  * adapter used by the ApplicationView for this control does not support
  * local transactions, an exception is thrown.
  */
  public void beginLocalTransaction()
    throws Exception;

  /**
  * Commit the active local transaction for this control. All work done
  * since the last call to beginLocalTransaction() will be committed into
  * the EIS's permanent state. If the underlying
  * adapter used by the ApplicationView for this control does not support
  * local transactions, an exception is thrown.
  */
  public void commitLocalTransaction()
    throws Exception;

  /**
  * Rollback the active local transaction for this control. All work done
  * since the last call to beginLocalTransaction() will be discarded.
  * If the underlying adapter used by the ApplicationView for this control does
   not support local transactions, an exception is thrown.
  */
```

```
  public void rollbackLocalTransaction()
    throws Exception;

  /**
  * Callback interface that defines the generic events that can be delivered
  * to any ApplicationView control instance. Note, no client-visible events
  * are defined here. They see only those events defined in the control
  * interface (JCX file).
  */
  public interface Callback
{
    /**
    * Private/internal callback handler for handling async responses.
    * Clients of ApplicationViewControl should NOT implement this type
    * of handler.
    * @exclude
    */
    public void internalCallbackMethod(Object asrObj)
      throws Exception;
    }
}
```

Application View Control (JCX) and Business Process (JPD) Samples

For sample Application View controls and business processes, use WebLogic Workshop to open the sample application located in:

SAMPLES_HOME/integration/sampleApp/sampleApp.work

For instructions on running the samples, use WebLogic Workshop to open the following HTML file:

SAMPLES_HOME/integration/sampleApp/docs/index.html

Receiving Events

The JCX file for an Application View control instance does not represent events from the associated application view. Business processes receive events from application views by using an instance of the Subscribe Control for the Message Broker. When an application view which has events is published to WebLogic Workshop, a `wlai.channel` file is automatically generated. The channel file defines the channels available for subscription by business processes.

For more information on Message Broker Subscription controls, see the Message Broker Subscription Control topic in the WebLogic Workshop Help at the following URL:

http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsBrokerSubscribe.html

The following in an example of a `wlai.channel` file., consisting of subscriptions for two events.

Listing 3-2 Sample wlai.channel File

```
<?xml version="1.0" encoding="UTF-8"?>
<mb:channels channelPrefix="/wlai"
      xmlns:mb="http://www.bea.com/wli/broker/channelfile">
  <mb:channel messageType="none"
      name="InsertBasedEvents">
    <mb:channel messageType="xml"
      name="CustomerInsertEvent"
      qualifiedMessageType="et:CUSTOMER_TABLE.insert"
      xmlns:et="wlai/CustomerInsertEvent_CUSTOMER_TABLE_insert"/>
  </mb:channel>
  <mb:channel messageType="none"
    name="FunctionDemo">
    <mb:channel messageType="none"
      name="CustomerMgmt">
      <mb:channel messageType="xml"
        name="CustomerUpdated"
        qualifiedMessageType="et:CUSTOMER_TABLE.update"
        xmlns:et="wlai/FunctionDemo/CustomerMgmt_CustomerUpdated_event"/>
    </mb:channel>
  </mb:channel>
</mb:channels>
```

To enable a business process to receive events from an application view, create an instance of a Message Broker Subscription control as follows:

1. Start WebLogic Workshop.
2. Open the business process which will receive application view events.

3. On the **Controls** tab of the **Data Palette**, click **Add⇒Integration Controls⇒MB Subscription** to invoke the control wizard.
4. Enter a variable name for the control.
5. Select the radio button to create a new MB Subscription control to use and enter a name for the control.
6. Select a channel from the channel-name drop-down list that represents the event you want to receive. Look for events with a channel prefix of `wlai`, indication that the channels are defined in the generated `wlai.channel` file.

   When you select a channel, the associated message type and, if specified, metadata type are displayed.
7. If you want to filter the message, select the This subscription will be filtered checkbox.
8. Click **OK**

   The control is created using the name supplied in step 5.

Once the control is created, you can use the Property Editor to define a filter at either the class or method level.

Note: You can start a business process on receiving an event. This feature is available from the process Start node, where you can select Subscribe to Message Broker Channel and select your event from the **Channel Name** drop-down list..

Handling Application View Local Transactions in Business Processes

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 WebLogic Workshop. 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 business process.

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.

CHAPTER 4

Using the Application Integration Design Console

The Application Integration Design 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 Integration Design 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 Integration Design Console
- Creating a Folder
- Removing an Application View
- Removing a Folder

Logging On to the Application Integration Design Console

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

Warning: You should only have one instance of the Application Integration Design Console running on a single client machine. Running multiple consoles on a single machine may interfere with proper navigation between screens in your web browser.

To log on to the Application Integration Design Console:

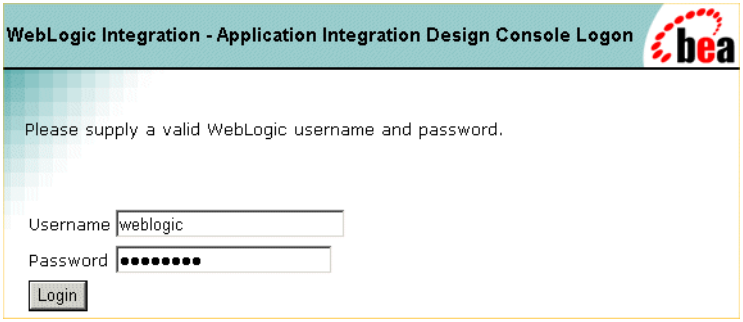
1. Launch a browser window.
2. Enter the URL for your system's Application Integration Design Console in the following format:

   `http://your_server:your_port/wlai`

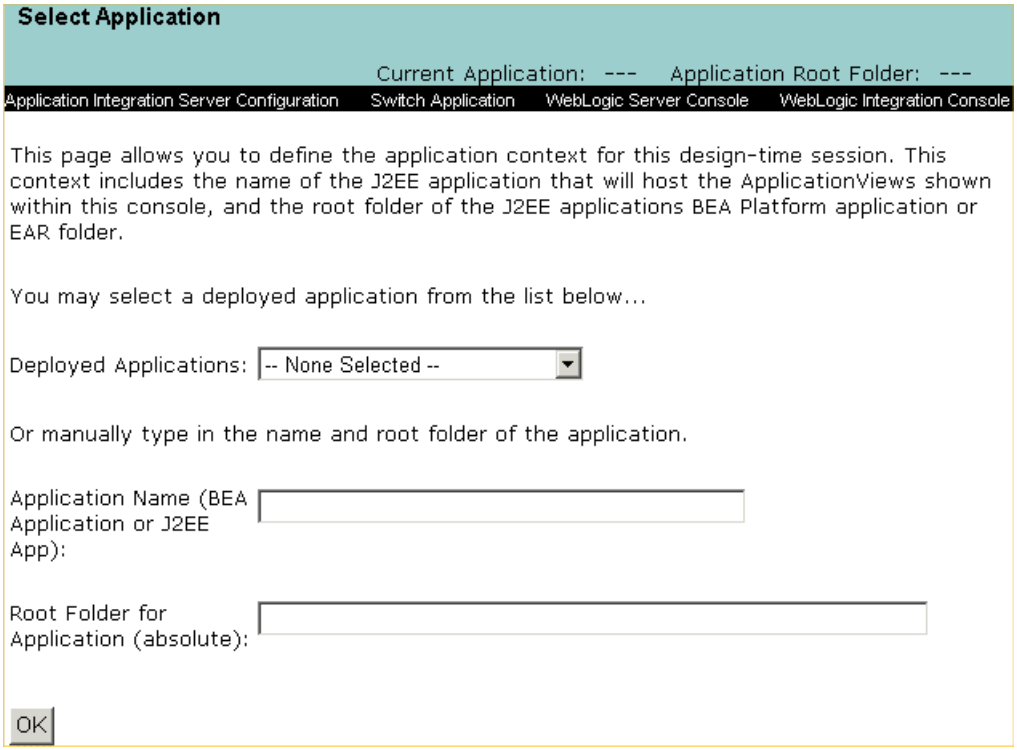
   For example: `http://wli1:7001/wlai`

   Note: You can also invoke the Application Integration Design Console from WebLogic Workshop. Open a WebLogic Workshop application, ensure that WebLogic Server is running, and then choose Tools→WebLogic Integration→Application Integration Design Console.

   The logon page is displayed.



3. Enter your WebLogic Server username and password, then click **OK**. The **Context Definition** page is displayed.

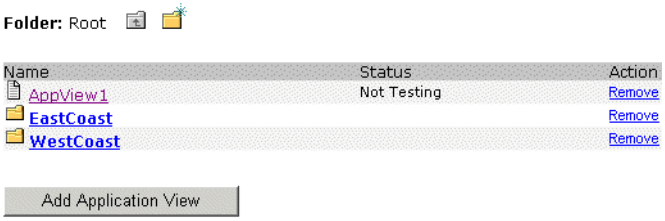


4. The Application Integration Design Console stores information about application views in a file-based repository. Before you create an application view, you must define the context for the design-time session. This determines where application view information is stored.

   To choose an existing, deployed application, select an application from the Deployed Applications menu.

   To create a repository for a new application, specify an application name and the root directory for the application.

5. When you have defined the context for the design session, click **OK**. The Application Integration Design 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 Integration Design 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.

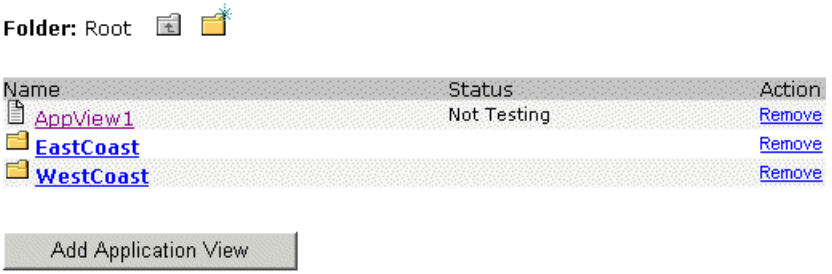
4. Click **Save**.

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 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 Integration Design Console, navigate to the folder in which the target application view is located.
2. Choose an application view and click **Remove**. A confirmation page is displayed. Click **Confirm** to delete the application view.

Note: When you delete a previously published application view, associated `wlai.channel` file, and EJB file, the application view ID still appears in the WebLogic Integration Administration Console and is shown in an Undeployed state after you rebuild the application. The deleted application view ID is removed from WebLogic Integration Administration Console when you reboot the system.

Removing a Folder

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

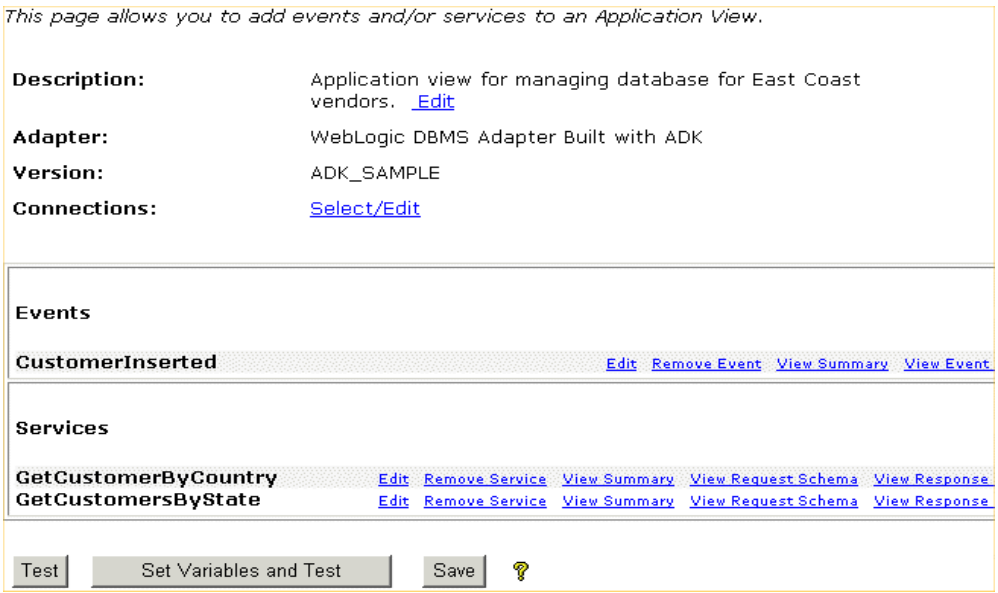
1. Remove all application views and subfolders from the target folder.
2. Log on to the Application Integration Design Console and go to the folder in which the target folder resides.
3. Click **Remove**. A confirmation page is displayed.

4. Click **Confirm**. The folder is deleted.

Administering an Application View

You can perform administration tasks on application views using the **Application View Administration** page. To administer an application view:

1. Log on to the Application View Design Console, select and application, and go to the folder containing the desired application view.
2. Click the application view name. The **Application View Summary** page is displayed.
3. Click **Edit**. The **Application View Administration** page is displayed.



4. Choose an administration task to perform:
   - Click **Edit** at the end of the **Description** field to edit the application view description.
   - Click **Select/Edit** to the right of the **Connections** label to configure connections, assign existing connections for services and events, or create new service or browsing connections.
   - Use the buttons in the services and events rows to add, edit, or remove services or events.

CHAPTER 5

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 Integration Design 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 5-1 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 5-2 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 5-3 shows an example of how `ConnectionSpec` is used.

Listing 5-3 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 to include an Application View control, 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 1-7.

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 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 5-8 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 5-4 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();
    }
  }
}
```

APPENDIX A

Importing and Exporting Application Views

This section presents the following topics:

- Import-Export Utility
- Migrating Application Views Using the 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.

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

- Application views
- 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.

Migrating Application Views Using the Import-Export Utility

Use the Import-Export utility to migrate application views from a WebLogic Integration 7.0 environment to a WebLogic Integration 8.1 environment. As part of the migration, the utility removes the ACLs from the application view and converts the use of connection factories to use of resource adapters. The migrated application views are loaded into the file-based repository.

Note: You must define a project in WebLogic Workshop before migrating application views. The project determines the location of the repository.

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 scripts used to run the utility are located as follows:

`WLI_HOME/bin/aiimportexport` (`cmd` or `sh`)

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:

```
aiimportexport <appname> <root_dir> <file>
[-codepage=Cp<codepage_number>] [-dump=< <namespace> | <'Root'> >]
[-append] [-overwrite] [-publish]
< [-export [object_name]*] | [-import [edit-on-import_filename]
[-eventProps=<connection_factory_name>,<properties_file_name>]*] >
```

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

Parameter	Description
appname	The name of J2EE application to which artifacts will be imported, or from which artifacts will be exported.
root_dir	The root directory of the AI repository within the given application.
file	Name of the JAR 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 The value specified must match your console's codepage. On Windows systems, use the `chcp` command to display the console's codepage.
-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* instead of overwriting the file. This option should only be used when `-export` is specified.

Parameter	Description
`-overwrite`	Overwrites items already in the repository and contained in the archive being imported. This option should only be used when `-import` is specified.
`-publish`	Publishes any application views after they are imported. This option should only be used when `-import` is specified.
`-export`	Specifies an export operation and the name of the objects (namespaces and application views) to be exported into *`file`*. When specifying an object within a namespace use a dot (.) as the delimiter, for example, `mynamespc.myappview`. 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).

Parameter	Description

`-eventProps`	This option is used only when importing a WebLogic Integration 7.0 or earlier export JAR. This option defines a mapping between the connection factory (qualified) name and a properties file that provides the properties needed for event generation within the adapter instance generated for the named connection factory. Multiple `-eventProps` arguments may be used as needed. These properties were specified in WebLogic Integration 7.0 within the EventRouter WAR file's `web.xml`.
	
	When creating the properties file for the `-eventProps` arguments, you should take the property name/value pairs from the `init-param` elements in the `web.xml` of the adapter's EventRouter WAR file. You can ignore the following properties from the `web.xml` file, and exclude them from the `-eventProps` properties file:
	`eventGeneratorClassName`
	`RootLogContext`
	`AdditionalLogContext`
	`LogConfigFile`
	`LogLevel`
	`MessageBundleBase`
	`LanguageCode`
	`CountryCode`
	
	If no `-eventProps` argument is found that maps a properties file to the qualified name for a connection factory, then when the connection factory is imported, the adapter instance generated has inbound messaging disabled. To specify event properties for the adapter instance at a later time, use the Edit Adapter Instance page of the Application Integration Design Console.

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" on page A-9.

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 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" on page A-9) 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" on page A-9.

The document specified in the *`editOnImportDoc`* argument must conform to the DTD shown in "Editing on Import" on page A-6.

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`.

Index

A

Adapter Development Kit (ADK) 1-1
Application Integration Design Console 4-1
application view events
    adding 2-19
    testing manually 2-30
    testing with a service 2-30
application view folders
    creating 4-4
    removing 4-5
application view services
    adding 2-17
application views
    adding events to 2-19
    adding services to 2-17
    editing 2-33
    removing 4-5
    testing events 2-29
    users of 1-6
    using by writing custom code 1-7
    using in WebLogic Workshop 1-6
    when to define 1-2

B

business process management (BPM)
    using 3-1
    when to use 1-7
business processes 1-6
    using custom code 1-7

C

custom code
    for business processes
        when to use 1-7
        writing 5-1
    for defining application views 1-2

E

events
    See application view events

J

Java
    custom coding in 5-1

N

namespace enforcement option 2-24

W

WebLogic Workshop
    using 3-1
    when to use 1-7
Workshop
    See WebLogic Workshop