BEA WebLogic Integration™

Using Application Integration

Version 2.1
Document Date: October 2001

Using Application Integration

Part Number	Date	Software Version
N/A	October 2001	2.1

Contents

A. Migrating Application Integration Data

About This Document

Using Application Integration is organized as follows:

- "Introduction to Using Application Integration" provides an overview of BEA WebLogic Integration Framework and explains how it fits into the WebLogic Server environment and contributes to the BEA EAI solution.
- "Defining an Application View" explains how to log into an adapter, create and configure application views to represent your enterprise's business processes.
- "Using Application Views in Business Process Management" explains how to use application views in the WebLogic Server environment by setting up workflows using WebLogic Integration Studio.
- "Using Application Views by Writing Custom Code" explains how to use application views in the WebLogic Server environment by writing custom Java code.
- "Using the WebLogic Integration Application View Console" explains how to use namespaces to organize your application views by location or department instead of by adapter.
- Appendix A, "Migrating Application Integration Data" explains how to migrate application integration data between WebLogic Server domains.

What You Need to Know

This document is intended for the following users:

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

e-docs Web Site

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

How to Print the Document

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

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

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

Related Information

The following resources are also available:

BEA WebLogic Server documentation (`http://edocs.bea.com`)

BEA WebLogic Business Process Management (BPM) documentation (`http://edocs.bea.com`)

XML Schema Specification (`http://www.w3c.org/TR/xmlschema-formal/`)

The Sun Microsystems, Inc. Java site (`http://www.javasoft.com/`)

The Sun Microsystems, Inc. J2EE Connector Architecture Specification (`http://java.sun.com/j2ee/connector/`)

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Application Integration 2.0 release.

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

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

- Your name, e-mail address, phone number, and fax number
- Your company name and company address

- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

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

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

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

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

CHAPTER

1 Introduction to Using Application Integration

This document is *Using Application Integration,* the manual for using adapters built using the BEA WebLogic Integration ADK (Adapter Development Kit). This document explains how to define application view services and events and use them in your business processes in the WebLogic Integration environment.

The instructions in this document are general instructions. Because each adapter and application is different, the instructions do not cover any information specific to any particular adapter or application. If you are looking for a tour of specific adapters included with the ADK, see the following two sections in *Developing Adapters:*

- "The E-mail Adapter" in *Developing Adapters.*
- "The DBMS Adapter" in *Developing Adapters.*

This section provides information on the following subjects:

- Before You Begin
- Concepts

Before You Begin

Before you can begin using adapters to integrate your enterprise, make sure the following prerequisites are satisfied:

- You have installed WebLogic Server, including Service Pack 1.

- You have installed JDK 1.3.1. The JDK 1.3 development kit is automatically installed when you install WebLogic Server 6.1, although you may want to install your own version, as long as it is 1.3.1-compliant.
- You have installed BEA WebLogic Integration.
- Deploy each adapter for which you will define application views.

Note: For a complete list of prerequisites, see the release notes.

Concepts

This section describes important concepts with which you should familiarize yourself before you work with adapters and application views. The following concepts are discussed in detail in "Defining an Application View" on page 2-1 and "Using Application Views in Business Process Management" on page 3-1. For a broad overview of application integration, see *Introducing Application Integration.*

When to Use an Application View and When to Write Custom Code

To support service invocation and events, you can define application views, or you can write custom code that accomplishes the same functions. Application views provide the most convenient interface to an adapter's resources, but there are other ways to access an adapter. Normally, for each adapter, you will define application views to expose the application functions. However, for those who require more control, you may also write custom code to access the resources of an adapter. For your enterprise, you must decide whether to define application views or write your own code.

When to Define an Application View

You can define application views to easily integrate most enterprise information system (EIS) applications. In general, define application views in the following situations:

- You have more than one EIS in your enterprise, and you lack developers who have detailed, thorough knowledge of all of the systems.
- You want to construct business processes using WebLogic Integration Studio.
- You may need to update the parameters of the adapter or one of its processes.

When to Write Custom Code Instead of Defining an Application View

In general, write custom code as an interface to an adapter only in the following situations:

- You have only one EIS in your enterprise
- You have access to a developer who has thorough, detailed knowledge of each EIS involved in the business processes being coded.
- You do not need to use the coded functions in BPM.
- Your code will never change.

“Defining” Versus “Using” an Application View

There are two initial steps in the life cycle of an application view:

- Defining the application view.
- Using the application view.

Defining: Configuring an Application View and Adding Events and Services

When you define an application view, you configure the communication parameters, then add services and/or events. 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 will connect to the target EIS.

Defining an application view includes the following tasks:

- Entering a unique name for the application view.

- Configuring parameters that establish the network connection between the application view and the application itself.
- Configuring parameters specific to the application.
- Configuring parameters used for load balancing by the application view.
- Configuring parameters used to manage the pool of connections available to the application view.
- Defining security privileges for users of the application view.

Using an Application View in a Business Processes

After you define an application view, you can deploy it on WebLogic Server. You can use deployed application views to implement your enterprise's business processes in a business process workflow.

After using an application view in a business process workflow, the end result is a deployed electronic representation of your enterprise's business process. The workflow specifies how your applications will interact with each other to accomplish the business processes. The application views perform the transactions themselves.

Defining an Application View

When you define an application view for an adapter, you are creating an XML-based interface between WebLogic Server and a particular EIS application. For detailed steps for defining application views for adapters, see Chapter 2, "Defining an Application View."

Defining an application view involves these basic steps:

1. Naming and Configuring Connection Parameters for an Application View.
2. Adding Services and Events to an Application View.
3. Testing Services and Events.

Naming and Configuring Connection Parameters for an Application View

The first step in defining an application view for an adapter is to log on to the Application View Console, select a folder where the application view will reside, and configure its EIS connection parameters. For details on creating and configuring an application view, see the following topics:

- "Logging On to the WebLogic Integration Application View Console" on page 2-5.
- "Defining an Application View" on page 2-6.

Adding Services and Events to an Application View

After defining the EIS connection parameters of the application view, the next step is to add services and events. Services and events support a subset of an application's business processes by allowing other WebLogic Server clients to interact with the application functions you specify. The application view services and events allow specific types of transactions between WebLogic Server and the EIS application. For details on adding services and events to an application view, see the following topics:

- "Adding a Service to an Application View" on page 2-9.
- "Testing an Application View's Events" on page 2-22.

Testing Services and Events

After adding a service or event to an application view, you must make sure the service or event interacts properly with the EIS application. For details on testing services and events, see the following topics:

- "Testing an Application View's Services" on page 2-19
- "Testing an Application View's Events" on page 2-22

Using an Application View in Business Processes

Once you define an application view in your WebLogic Integration environment, you can use the application view in your enterprise's business processes. There are two ways to use application views in business processes:

- By designing business process workflows in BPM.
- By writing custom code.

Using an Application View in BPM

The most common way to use an application view in your enterprise's business processes is to design a workflow in BPM. BPM provides a GUI-based environment for designing business process workflows. These workflows can include application view services and events defined using application integration.

There are four ways to use an application view in a workflow using BPM:

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

For detailed information on each method, see Chapter 3, "Using Application Views in Business Process Management."

Using an Application View by Writing Custom Code

If you do not use BPM, the alternate way to use an application view in your enterprise is to write custom Java code to implement a business process.

For detailed steps for custom coding business processes, see Chapter 4, "Using Application Views by Writing Custom Code."

Deciding Which of the Two Methods to Use

For each business process you implement, you will need to decide which of the two implementation methods to use. You can implement any business processes as a workflow by using BPM, but you should only attempt to custom code a business process if it is extremely simple and specialized. In this document, custom coding is offered only as an alternate method for those who require it.

When to Use BPM

In general, use BPM to implement a business process in the following situations:

- When implementing the required business processes would require complicated error management, persistent processes, and sophisticated conditional branching.

  For example, if a business process receives events, selects only a subset of the events, performs complex branched actions, then generates many complex messages and sends the messages to a variety of WebLogic Server clients, then you should use BPM to implement the business process.
- When you will have to make occasional changes to the business process.

  BPM reduces the number of compile/test/debug cycles.
- When, like most organizations, your developers are valuable and scarce.

When to Write Custom Java Code

In general, write custom code to implement a business process only in the following situations:

- When the business process is simple. A simple business process is one that includes no complicated error recovery, long-lived processes, conditional branching, or joining of the process flow.

  For example, if a business process performs a limited set of actions on an incoming message, then routes the minimally transformed message to a small number of client applications, then the business process is simple enough to express by writing custom code.
- When you will not need to update the business process very often.

  When you update custom code, the change requires a full compile/test/debug cycle, which can be costly.
- When your organization can afford to dedicate developers to implement the business processes in code.

CHAPTER

2 Defining an Application View

This section contains information on the following subjects:

- Before You Begin
- Introduction to Defining an Application View
- Steps for Defining an Application View

Before You Begin

Before you attempt to define an application view, make sure the following prerequisites are satisfied.

- The appropriate adapter has been developed using the ADK. You can only create and configure application views for existing adapters.
- Determine which business processes need to be supported by the application view you are configuring. The required business processes determine the types of services and events you will include in the application views. Normally, this means gathering 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.

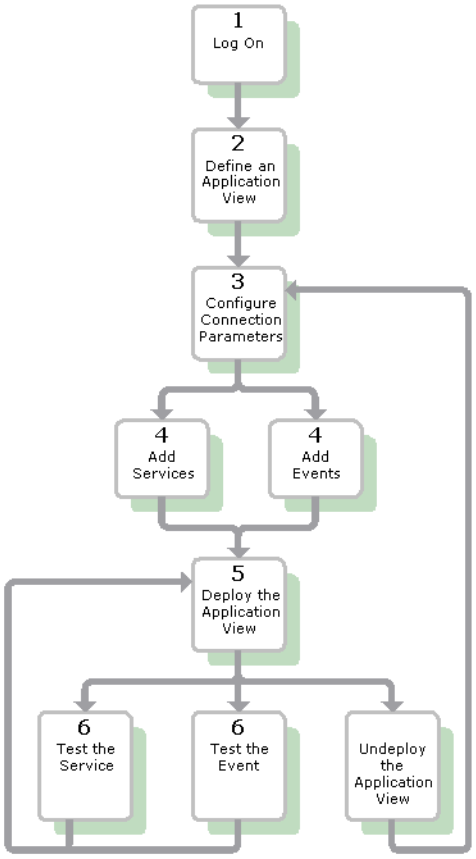
Introduction to Defining an Application View

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 with any number of services and events.

The Flow of Events

Figure 2-1 shows an overview of the steps involved in defining an application view.

Figure 2-1 The Flow of Events for Defining and Configuring Application Views


1
Log On
2
Define an Application View
3
Configure Connection Parameters
4
Add Services
4
Add Events
5
Deploy the Application View
6
Test the Service
6
Test the Event
Undeploy the Application View

4. Log on to the WebLogic Integration Application View Console. For detailed information, see "Logging On to the WebLogic Integration Application View Console" on page 2-5.

5. Click Add Application View to create a new application view for the appropriate adapter. An application view enables a set of business processes for this adapter's target EIS application. For detailed information, see "Defining an Application View" on page 2-6.

6. At the Configure Connection Parameters page, enter application connection parameters. For detailed information, see "Defining an Application View" on page 2-6.

The information is validated, and the application view is configured to connect to the system you specified.

7. Click Add Event or Add Service to define the appropriate events and services for this application view.
8. Deploy the application view on WebLogic Server so other entities can interact with it according to your security settings.

   Note: You can only test an application view if it is deployed.
9. Test the services and events to make sure they can properly interact with the target EIS application.

   Once the services and events are tested and functioning, you can use the application view in workflows. For more information, see Chapter 3, “Using Application Views in Business Process Management.”
10. Undeploy the application view if you need to reconfigure its connection parameters or add services and events.

    Note: When an application view is undeployed, no other entities can interact with it.

Steps 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.” When you create application views for your enterprise, they may look different than the screens shown in this document. This is normal, 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 on an adapter used in your enterprise, consult the relevant technical analyst or EIS specialist.

Logging On to the WebLogic Integration Application View Console

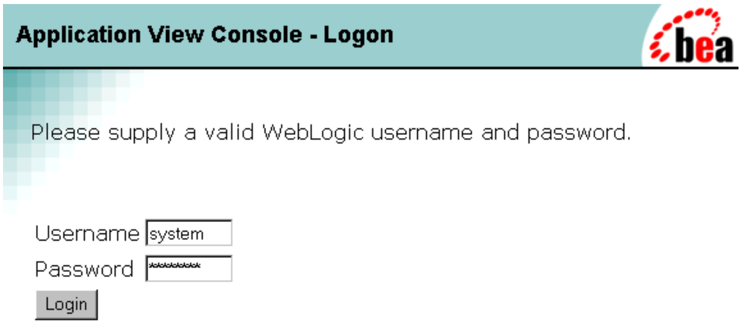
The first step in creating a new application view is to log on to the Application View Console page. The Application View Console displays all the application views in your WebLogic Integration environment, organized into folders.

To log on to the Application View Console:

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

   `http://localhost:7001/wlai`

   The Application View Console - Login page is displayed.

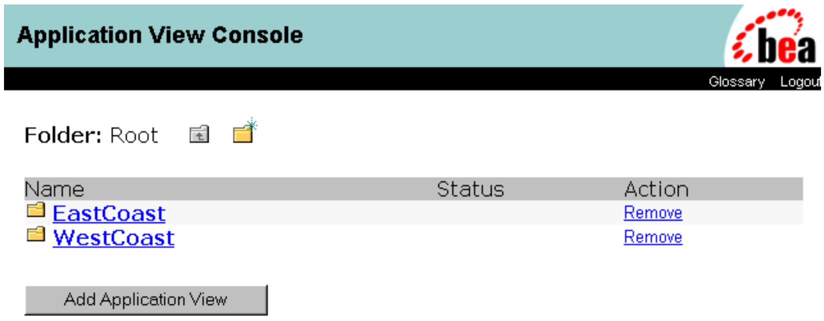

Application View Console - Logon

Please supply a valid WebLogic username and password.

Username: system

Password: ********

3. To log on to the Application View Console, enter your WebLogic Server username and password, then click Login. The Application View Console is displayed.



Application View Console

Folder: Root

Name	Status	Action
EastCoast		Remove
WestCoast		Remove

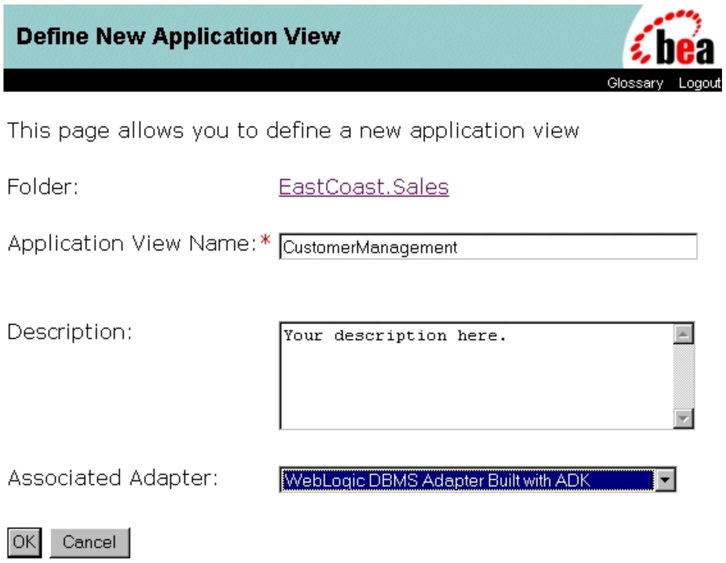
Note: If you do not see a page like this, consult the WebLogic Server administrator.

4. To add a folder, click the New Folder icon. For more information, see "Creating a Folder" on page 5-4.

Defining an Application View

After you log on to the Application View Console and navigate to a folder, click Add Application View to define an application view.

1. Log on to the Application View Console. For more information, see "Logging On to the WebLogic Integration Application View Console" on page 2-5.
2. To add a new application view to the current folder, click Add Application View. The Define New Application View page is displayed.


Define New Application View

This page allows you to define a new application view

Folder: EastCoast.Sales

Application View Name:* CustomerManagement

Description: Your description here.

Associated Adapter: WebLogic DBMS Adapter Built with ADK

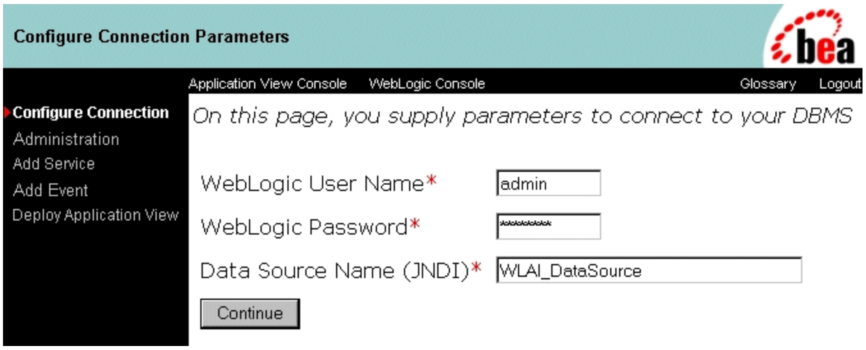
Note: Once you define the application view, you can not move it to another folder.

3. In the Application View Name field, enter a name. The name should describe the set of functions performed by this application. Each application view name must be unique to its adapter. Valid characters are a–z, A–Z, 0–9, and _ (underscore).

4. In the Description field, enter any relevant notes. These notes are viewed by users when they use this application view in workflows using business process management (BPM).

5. From the Associated Adapter list, select the adapter to use to create this application view.

6. Click OK. The Configure Connection Parameters page is displayed.



On this page, you supply parameters to connect to your DBMS

WebLogic User Name*	admin
WebLogic Password*	*********
Data Source Name (JNDI)*	WLAI_DataSource

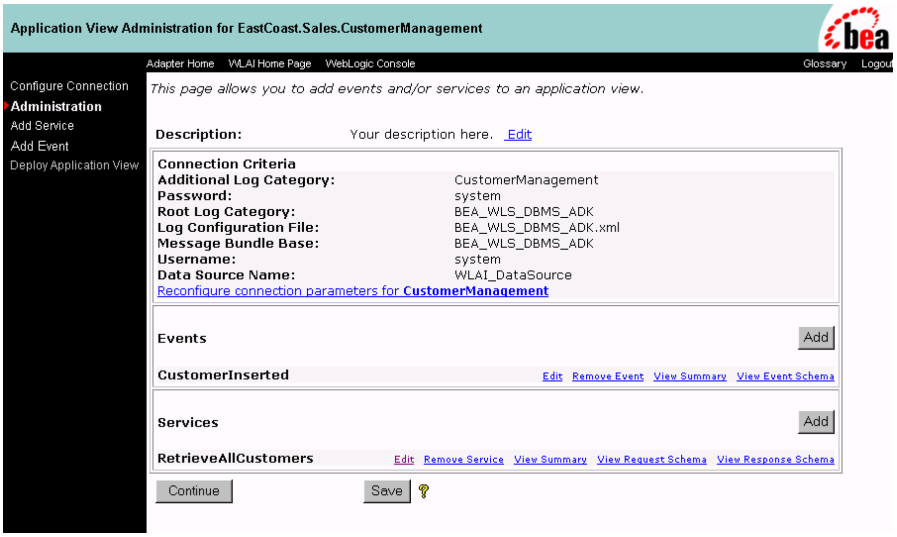
At the Configure Connection Parameters page, you define the network-related information necessary for the application view to interact with the target EIS. You need to enter this information only once per application view.

7. Enter your WebLogic Server User Name and Password.

   Note: Your page may have different fields than the ones shown. The fields are determined by the adapter.

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

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


Application View Administration for EastCoast.Sales.CustomerManagement

This page allows you to add events and/or services to an application view.

Description: Your description here. Edit

Connection Criteria	
Additional Log Category:	CustomerManagement
Password:	system
Root Log Category:	BEA_WLS_DBMS_ADK
Log Configuration File:	BEA_WLS_DBMS_ADK.xml
Message Bundle Base:	BEA_WLS_DBMS_ADK
Username:	system
Data Source Name:	WLAI_DataSource

Reconfigure connection parameters for **CustomerManagement**

Events Add

CustomerInserted Edit Remove Event View Summary View Event Schema

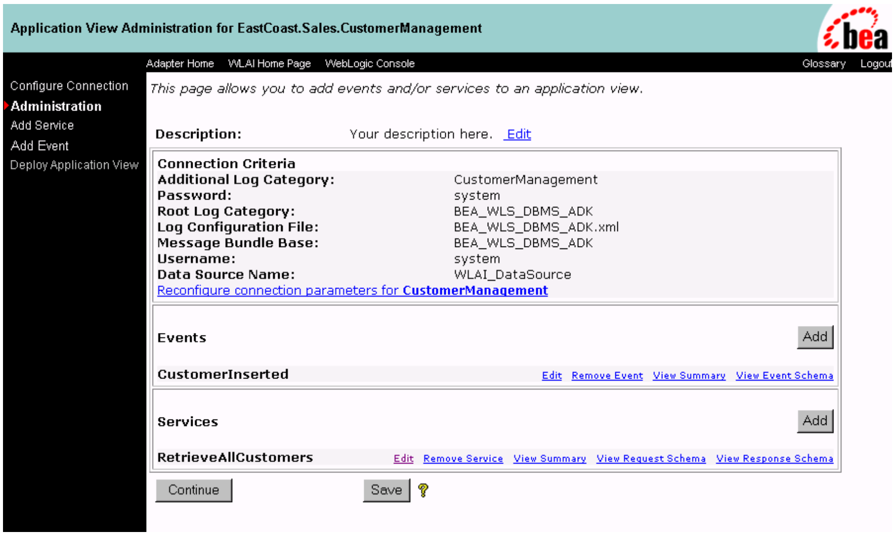
Services Add

RetrieveAllCustomers Edit Remove Service View Summary View Request Schema View Response Schema

Adding a Service to an Application View

After you create and configure an application view, add services that support the application's functions.

1. While the application view is open, click Administration. The Application View Administration page is displayed.



Application View Administration for EastCoast.Sales.CustomerManagement

This page allows you to add events and/or services to an application view.

Description: Your description here. Edit

Connection Criteria	
Additional Log Category:	CustomerManagement
Password:	system
Root Log Category:	BEA_WLS_DBMS_ADK
Log Configuration File:	BEA_WLS_DBMS_ADK.xml
Message Bundle Base:	BEA_WLS_DBMS_ADK
Username:	system
Data Source Name:	WLAI_DataSource

Reconfigure connection parameters for **CustomerManagement**

Events Add

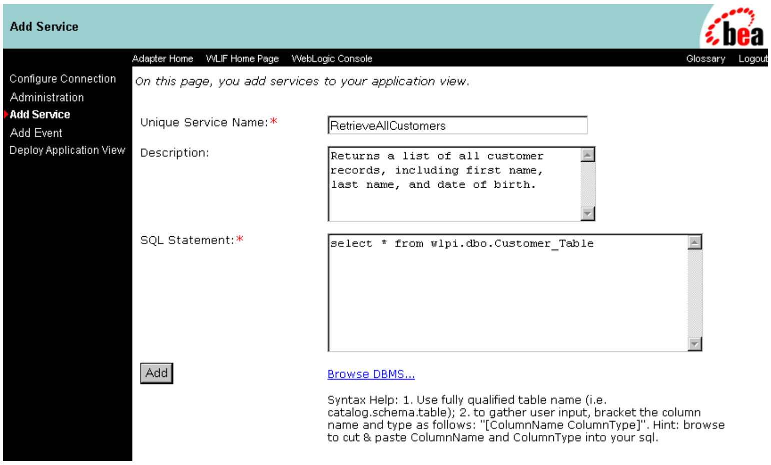
CustomerInserted Edit Remove Event View Summary View Event Schema

Services Add

RetrieveAllCustomers Edit Remove Service View Summary View Request Schema View Response Schema

Continue Save

2. Click Add Service. The Add Service page is displayed.



Add Service

On this page, you add services to your application view.

Unique Service Name:*	RetrieveAllCustomers
Description:	Returns a list of all customer records, including first name, last name, and date of birth.
SQL Statement:*	select * from wlpi.dbo.Customer_Table

Add Browse DBMS...

Syntax Help: 1. Use fully qualified table name (i.e. catalog.schema.table); 2. to gather user input, bracket the column name and type as follows: "[ColumnName ColumnType]". Hint: browse to cut & paste ColumnName and ColumnType into your sql.

Note: Your page may have different fields than the ones shown. The fields are determined by the adapter.

3. 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. Valid characters are a–z, A–Z, 0–9, and _ (underscore).

4. In the Description field, enter any relevant notes. These notes are viewed by users when they use this application view service in workflows using BPM.

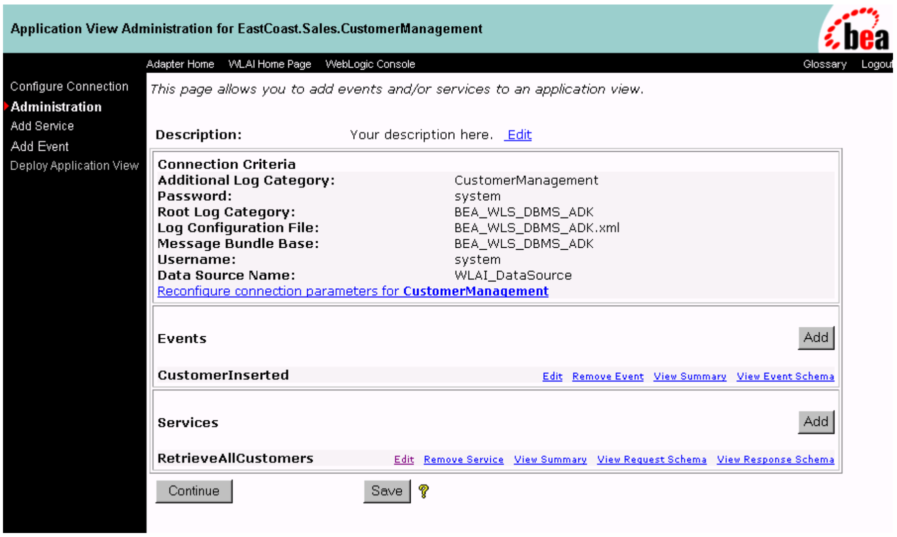
5. For any remaining fields, consult the relevant technical analyst or EIS specialist for the required information or format.

6. When finished, click Add.

Adding an Event to an Application View

After you create and configure an application view, add the appropriate events.

1. While the application view is open, click Administration. The Application View Administration page is displayed.


Application View Administration for EastCoast.Sales.CustomerManagement

This page allows you to add events and/or services to an application view.

Description: Your description here. Edit

Connection Criteria	
Additional Log Category:	CustomerManagement
Password:	system
Root Log Category:	BEA_WLS_DBMS_ADK
Log Configuration File:	BEA_WLS_DBMS_ADK.xml
Message Bundle Base:	BEA_WLS_DBMS_ADK
Username:	system
Data Source Name:	WLAI_DataSource

Reconfigure connection parameters for **CustomerManagement**

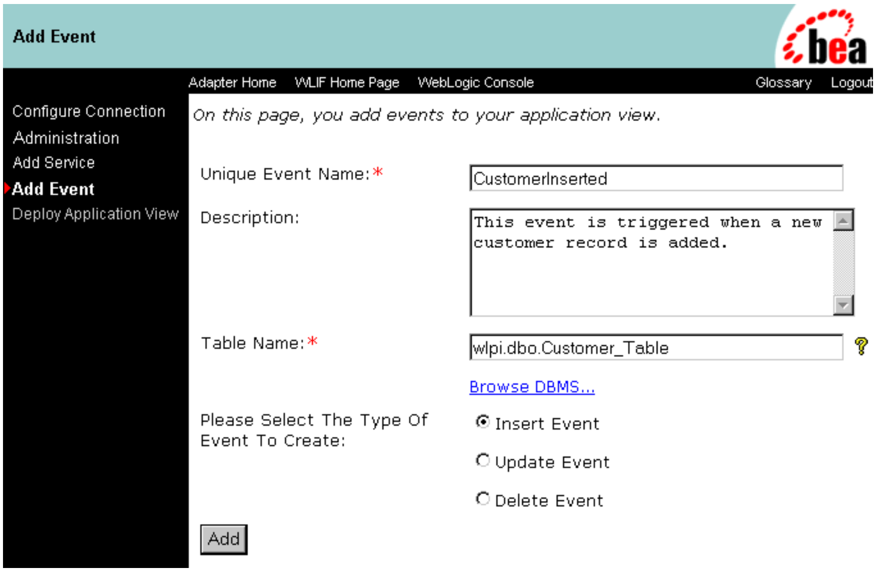
Events Add

CustomerInserted Edit Remove Event View Summary View Event Schema

Services Add

RetrieveAllCustomers Edit Remove Service View Summary View Request Schema View Response Schema

2. Click Add Event. The Add Event page is displayed.



On this page, you add events to your application view.

Unique Event Name:*	CustomerInserted
Description:	This event is triggered when a new customer record is added.
Table Name:*	wlpi.dbo.Customer_Table
Please Select The Type Of Event To Create:	[✓ Insert Event] Update Event Delete Event

Note: Your page may have different fields than the ones shown. The fields are determined by the adapter.

3. In the Unique Event Name field, enter a name. Each event name must be unique to its application view. Valid characters are a–z, A–Z, 0–9, and _ (underscore).

4. In the Description field, enter any relevant notes. These notes are viewed by users when they use this application view event in workflows using BPM.

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

6. When finished, click Add. The Application View Administration page is displayed.

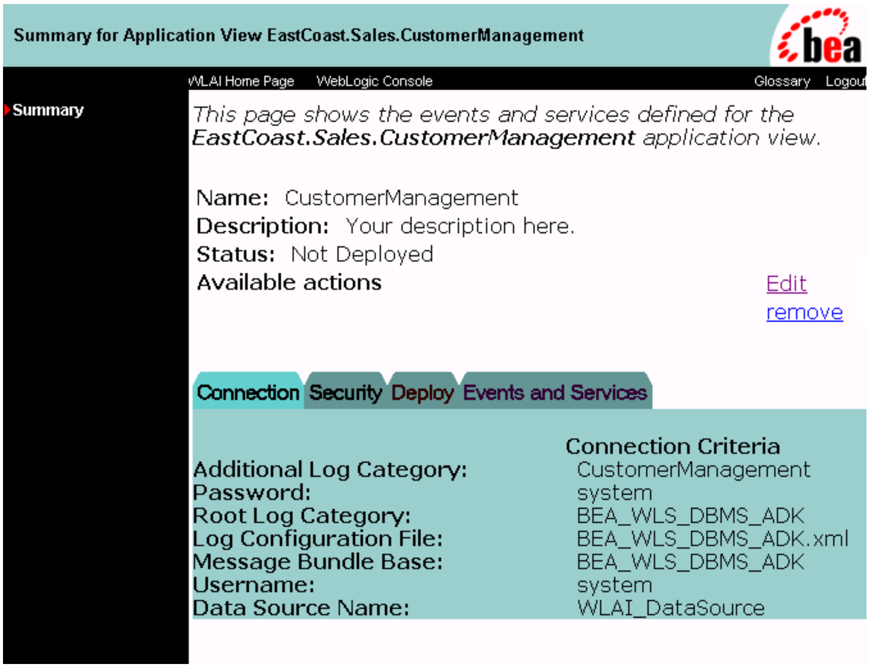
7. If you are finished adding services and events, click Continue to deploy the application view.

Deploying an Application View

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

To deploy an application view:

1. Open the application view. For more information, see "Logging On to the WebLogic Integration Application View Console" on page 2-5. The Summary for Application View page is displayed.

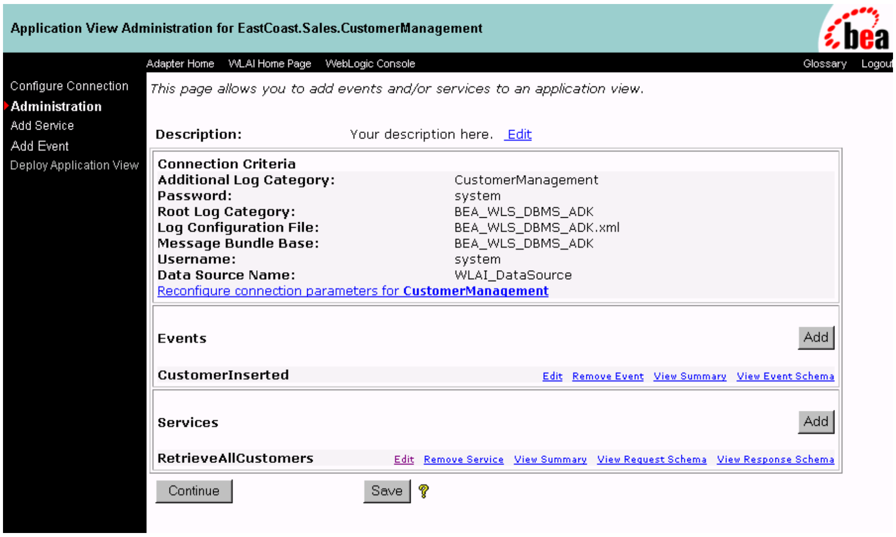


This page shows the events and services defined for the EastCoast.Sales.CustomerManagement application view.

Name: CustomerManagement
Description: Your description here.
Status: Not Deployed
Available actions

	Connection Criteria
Additional Log Category:	CustomerManagement
Password:	system
Root Log Category:	BEA_WLS_DBMS_ADK
Log Configuration File:	BEA_WLS_DBMS_ADK.xml
Message Bundle Base:	BEA_WLS_DBMS_ADK
Username:	system
Data Source Name:	WLAI_DataSource

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



This page allows you to add events and/or services to an application view.

Description: Your description here. Edit

Connection Criteria

Additional Log Category:	CustomerManagement
Password:	system
Root Log Category:	BEA_WLS_DBMS_ADK
Log Configuration File:	BEA_WLS_DBMS_ADK.xml
Message Bundle Base:	BEA_WLS_DBMS_ADK
Username:	system
Data Source Name:	WLAI_DataSource

Reconfigure connection parameters for **CustomerManagement**

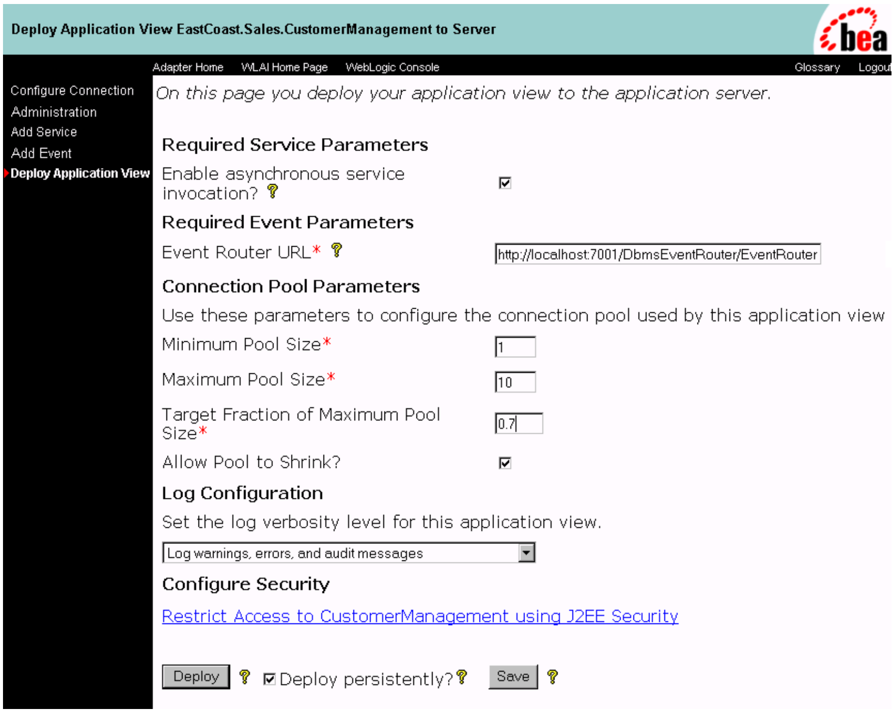
Events

CustomerInserted Edit Remove Event View Summary View Event Schema

Services

RetrieveAllCustomers Edit Remove Service View Summary View Request Schema View Response Schema

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



Deploy Application View EastCoast.Sales.CustomerManagement to Server

On this page you deploy your application view to the application server.

Required Service Parameters

Enable asynchronous service invocation? ☑

Required Event Parameters

Event Router URL* http://localhost:7001/DbmsEventRouter/EventRouter

Connection Pool Parameters

Use these parameters to configure the connection pool used by this application view

Minimum Pool Size* 1

Maximum Pool Size* 10

Target Fraction of Maximum Pool Size* 0.7

Allow Pool to Shrink? ☑

Log Configuration

Set the log verbosity level for this application view.

Log warnings, errors, and audit messages

Configure Security

Restrict Access to CustomerManagement using J2EE Security

Note: On the Deploy Application View page, the actual fields you see depend on the adapter. For an explanation of all fields, consult the relevant technical analyst or EIS specialist.

4. To enable BPM tor other authorized clients to asynchronously call the services (if any) of this application view, select Enable Asynchronous Service Invocation.

   An entity that calls an application view service asynchronously will continue its process without waiting for a response from the service.

5. If this application view has events, enter the URL of the adapter’s event router. For example,
   `http://localhost:7001/YourEIS_EventRouter/EventRouter`

   Note: If this field is not displayed, it means the application view has no events defined.

6. In the Minimum Pool Size field, enter the minimum number of connection pools to be used by this application view. For example, 1.

7. In the Maximum Pool Size field, enter the maximum number of connection pools to be used by this application view. For example, 10.

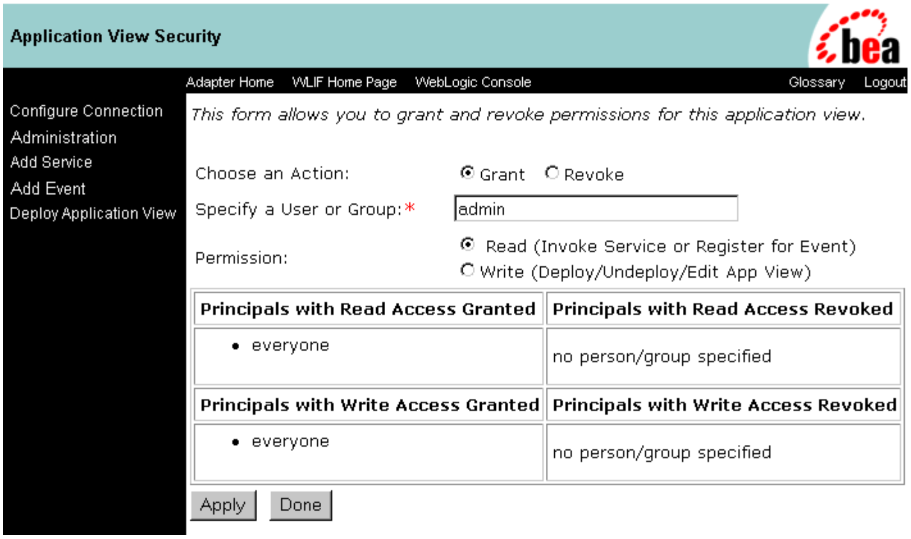
8. In the Target Fraction of Maximum Pool Size field, enter the ideal pool size, measured from 0 to 1.0. For example, 0.7. If the Maximum Pool Size is 10 and the Target Fraction is 0.7, this means the adapter will perform load balancing to attempt to maintain the connection pool size at 70% of the maximum, which in this case means 7 connections.

9. To automatically delete unused connections, select Allow Pool to Shrink.

10. On the Log Configuration area, select one of the following options according to your logging preferences:

    - Log errors and audit messages
    - Log warnings, errors, and audit messages
    - Log informationals, warnings, errors, and audit messages
    - Log all messages

11. If necessary, click Restrict Access using J2EE Security. The Application View Security page is displayed.



This form allows you to grant and revoke permissions for this application view.

Choose an Action: [✓ Grant] Revoke

Specify a User or Group:* admin

Permission: [✓ Read (Invoke Service or Register for Event)] Write (Deploy/Undeploy/Edit App View)

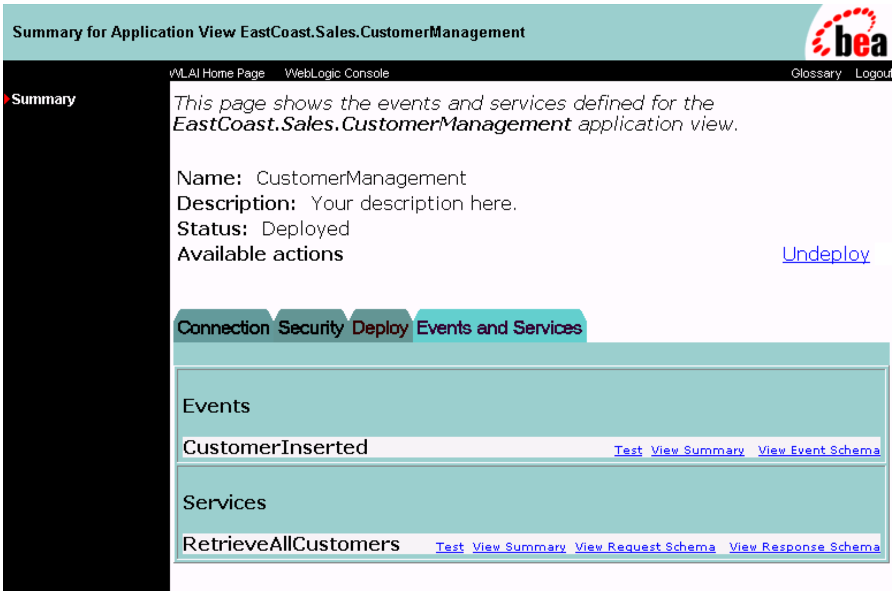
Principals with Read Access Granted	Principals with Read Access Revoked
• everyone	no person/group specified
Principals with Write Access Granted	**Principals with Write Access Revoked**
• everyone	no person/group specified

Use this page to grant or revoke a WebLogic Server user or group's read and write access to this application view.

12. When finished setting up permissions, click Apply to save your changes.
13. To return to the Deploy Application View page, click Done.
14. To save the Application View without deploying it, click Save.
15. To automatically redeploy this application view whenever WebLogic Server is restarted, select Deploy Persistently.

    Note: To save the application view for later completion without deploying it now, click Save at any time.

16. To deploy the application view, click Deploy Application View. The Summary for Application View page is displayed.



This page shows the events and services defined for the ***EastCoast.Sales.CustomerManagement*** *application view.*

Name: CustomerManagement
Description: Your description here.
Status: Deployed
Available actions

Events

CustomerInserted

Services

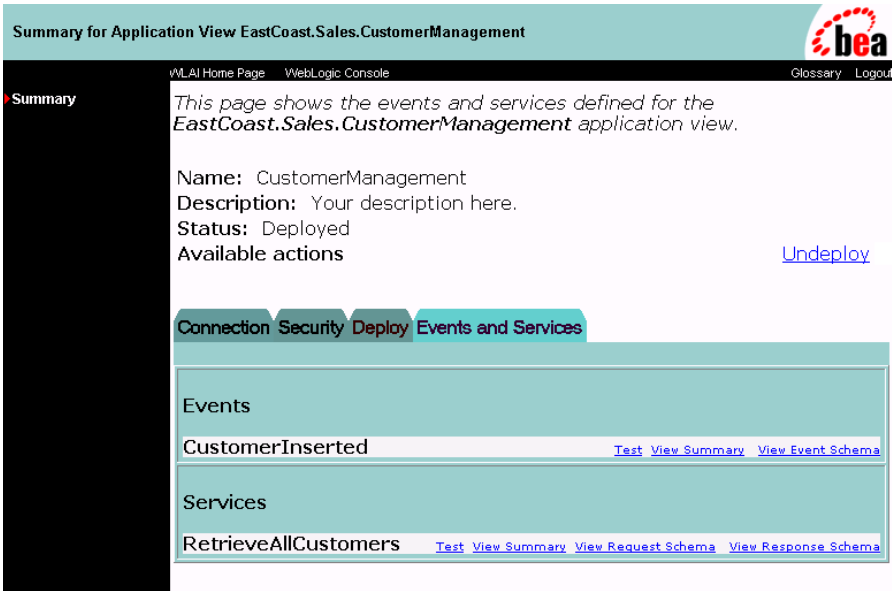
RetrieveAllCustomers

Undeploying an Application View

Undeploy an application view when you want to edit its connection parameters, add services and events, or disable clients from using the application view. For information on editing connection parameters, see "Defining an Application View" on page 2-6. When an application view is undeployed, no other WebLogic Server clients can interact with it, and you can not test its services or events.

To undeploy an application view:

1. Click Summary. The Summary for Application View page is displayed.



Summary for Application View EastCoast.Sales.CustomerManagement

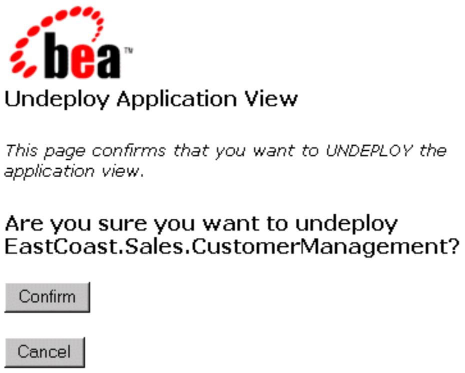
This page shows the events and services defined for the ***EastCoast.Sales.CustomerManagement*** *application view.*

Name: CustomerManagement
Description: Your description here.
Status: Deployed
Available actions Undeploy

Events
CustomerInserted Test View Summary View Event Schema

Services
RetrieveAllCustomers Test View Summary View Request Schema View Response Schema

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


Undeploy Application View

This page confirms that you want to UNDEPLOY the application view.

Are you sure you want to undeploy EastCoast.Sales.CustomerManagement?

3. Click Confirm. The Summary for Application View page is displayed, indicating you may deploy the application view again.

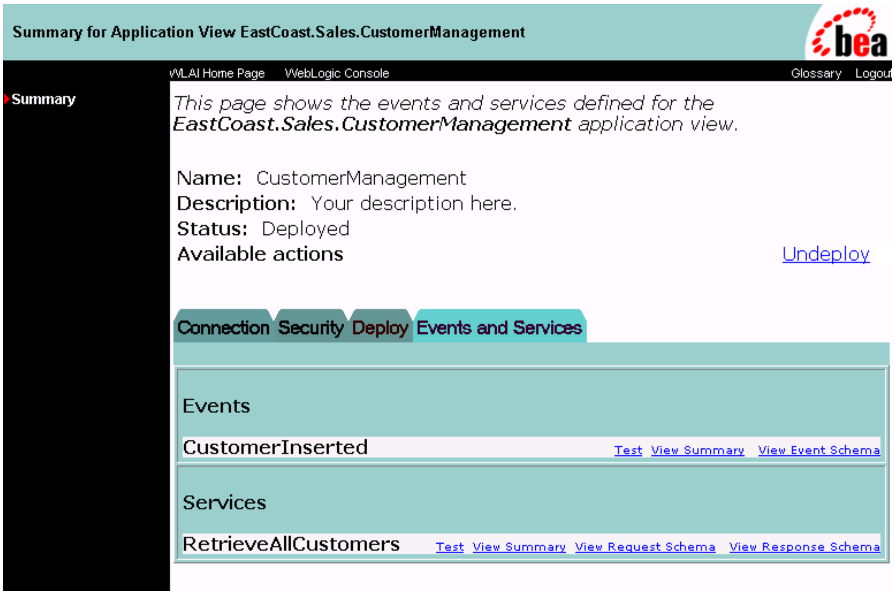
Testing an Application View's Services

After you create and deploy an application view that contains services, test the application view services. Testing evaluates whether or not the application view service interacts properly with the target EIS. To test application view services:

1. Define an application view (See "Defining an Application View" on page 2-6.), add the appropriate services, and deploy the application view (See "Deploying an Application View" on page 2-13.) if you have not done so already.

   You can test an application view only if the application view is deployed and it contains at least one event or service.

2. On the left navigation area, click Summary. The Summary for Application View page is displayed.


Summary for Application View EastCoast.Sales.CustomerManagement

This page shows the events and services defined for the ***EastCoast.Sales.CustomerManagement*** *application view.*

Name: CustomerManagement
Description: Your description here.
Status: Deployed
Available actions Undeploy

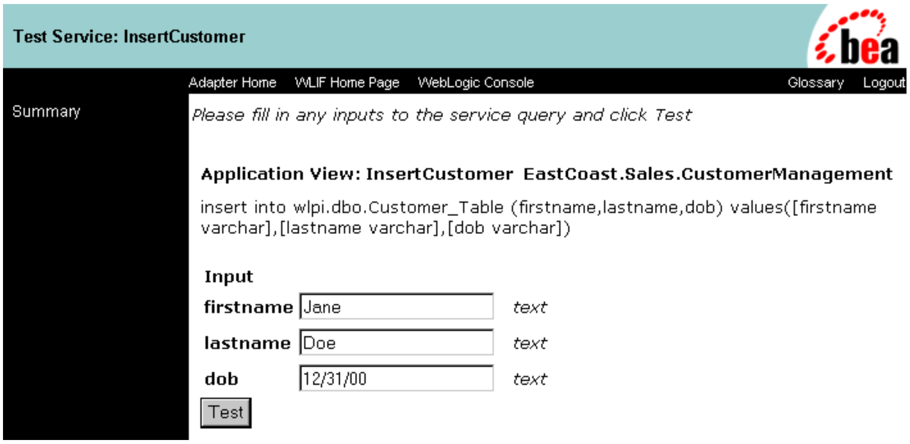
Events

CustomerInserted — Test · View Summary · View Event Schema

Services

RetrieveAllCustomers — Test · View Summary · View Request Schema · View Response Schema

3. In the Current Services area, find the service and click Test. The Test Service page is displayed.


Test Service: InsertCustomer

Please fill in any inputs to the service query and click Test

Application View: InsertCustomer EastCoast.Sales.CustomerManagement

insert into wlpi.dbo.Customer_Table (firstname,lastname,dob) values([firstname varchar],[lastname varchar],[dob varchar])

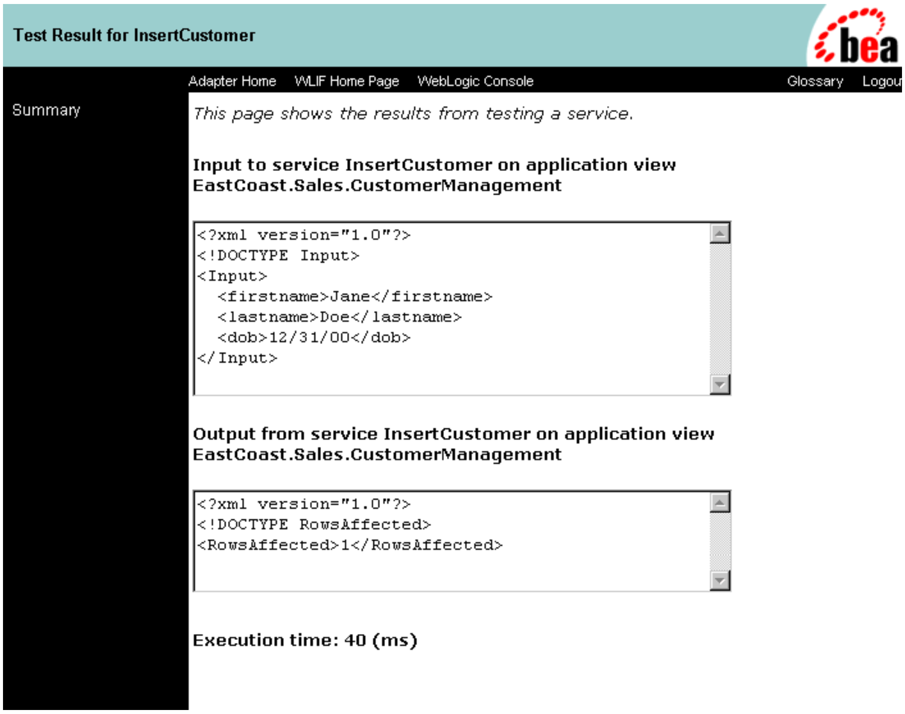
Input

firstname	Jane	*text*
lastname	Doe	*text*
dob	12/31/00	*text*

4. If necessary, enter the service input data in the Input fields. If the application view service processes this data correctly, the test is successful.

Note: Your Test Service page may have different fields than the ones shown. The fields are determined by the application view service. For an explanation of the fields, consult the relevant technical analyst or EIS specialist.

5. Click Test after entering the service input data. The Test Result page is displayed. This page displays the input and output documents.



This page shows the results from testing a service.

Input to service InsertCustomer on application view EastCoast.Sales.CustomerManagement

```
<?xml version="1.0"?>
<!DOCTYPE Input>
<Input>
  <firstname>Jane</firstname>
  <lastname>Doe</lastname>
  <dob>12/31/00</dob>
</Input>
```

Output from service InsertCustomer on application view EastCoast.Sales.CustomerManagement

```
<?xml version="1.0"?>
<!DOCTYPE RowsAffected>
<RowsAffected>1</RowsAffected>
```

Execution time: 40 (ms)

6. Repeat the test procedure for each service you want to test.

7. When finished testing the application view's services, you may keep the application view deployed or undeploy it (See "Undeploying an Application View" on page 2-18.) to edit the application view.

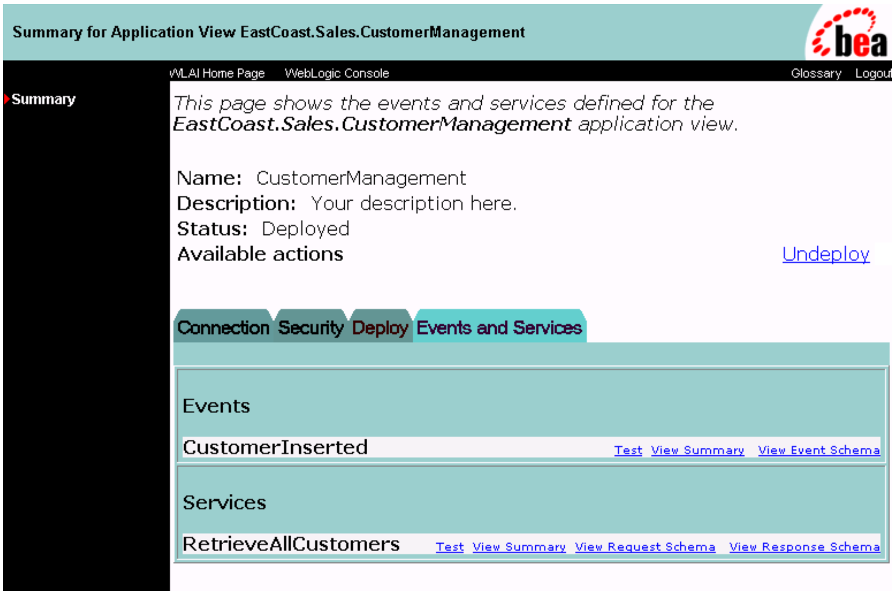
Testing an Application View's Events

After creating and deploying an application view that contains events, test the application view events. Testing evaluates whether or not the application view responds correctly to the EIS application. To test application view events:

1. Define an application view (See "Defining an Application View" on page 2-6.), add the appropriate events, and deploy the application view (See "Deploying an Application View" on page 2-13.) if you have not done so already.

   You can test an application view only if it is deployed and contains at least one event or service.

2. Click Summary. The Summary for Application View page is displayed.


Summary for Application View EastCoast.Sales.CustomerManagement

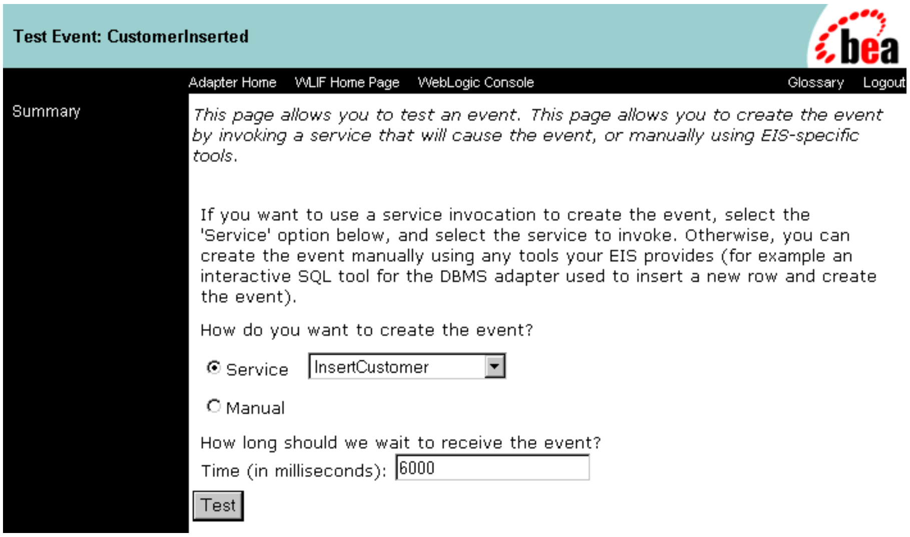
This page shows the events and services defined for the EastCoast.Sales.CustomerManagement application view.

Name: CustomerManagement
Description: Your description here.
Status: Deployed
Available actions

Events
CustomerInserted

Services
RetrieveAllCustomers

3. In the Current Events area, find your event and click Test. The Test Event page is displayed.



This page allows you to test an event. This page allows you to create the event by invoking a service that will cause the event, or manually using EIS-specific tools.

If you want to use a service invocation to create the event, select the 'Service' option below, and select the service to invoke. Otherwise, you can create the event manually using any tools your EIS provides (for example an interactive SQL tool for the DBMS adapter used to insert a new row and create the event).

How do you want to create the event?

[✓ Service] InsertCustomer

Manual

How long should we wait to receive the event?

Time (in milliseconds): 6000

Note: Your Test Event page may have different fields than the ones shown. The fields are determined by the application view service. For an explanation of the fields, consult the relevant technical analyst or EIS specialist.

4. Select the method to use to generate the test event:

- Service (See "If You Select Service" on page 2-23.): Select Service when you want to use one of the application view's own services to generate a "canned" event.
- Manual (See "If You Select Manual" on page 2-26.): Select Manual when you want to generate the event by logging on to an EIS application and perform the appropriate event-generating function.

If the application view event correctly responds before the specified time elapses, the test is successful.

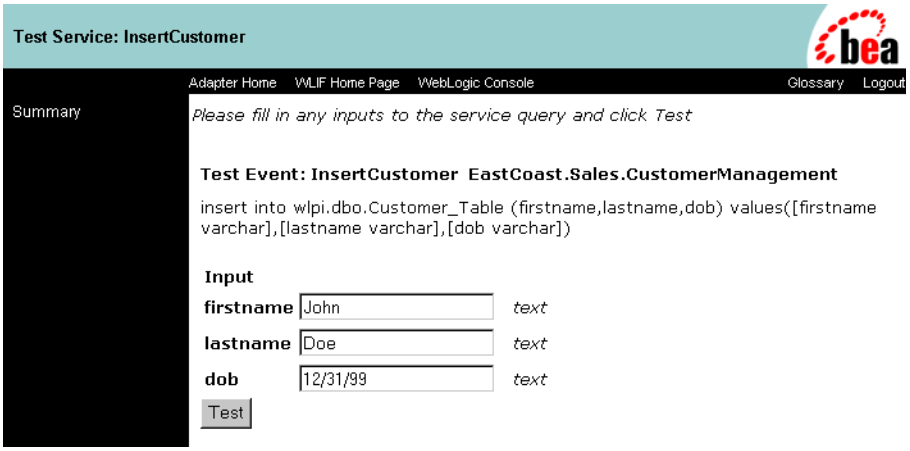
If You Select Service

a. On the Service menu, select a service that will trigger the event you are testing. For example, if you are testing the "NewCustomer" event, select a service that will invoke it, such as "Insert Customer."

b. In the Time field, enter a reasonable time to wait, in milliseconds. If this time elapses before the event succeeds, the test will time out and display a failure message.

c. Click “Test.” The triggering service is executed.

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



Please fill in any inputs to the service query and click Test

Test Event: InsertCustomer EastCoast.Sales.CustomerManagement

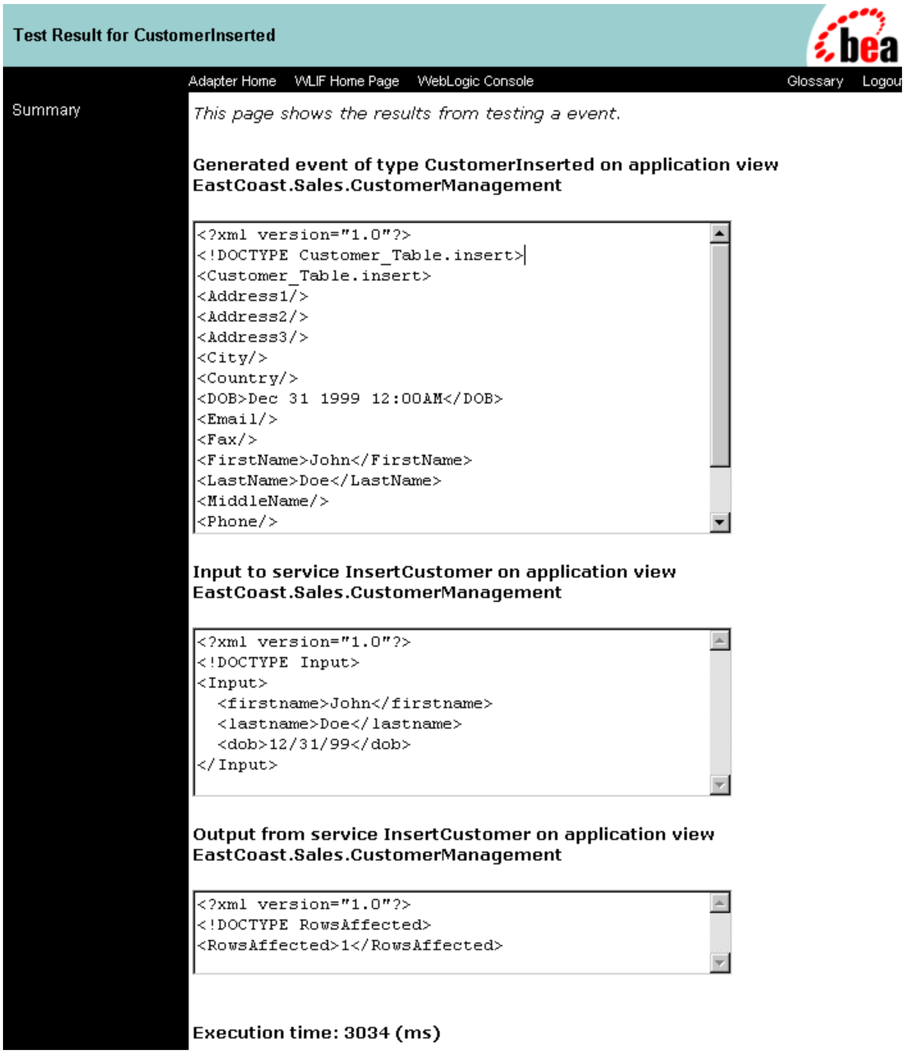
insert into wlpi.dbo.Customer_Table (firstname,lastname,dob) values([firstname varchar],[lastname varchar],[dob varchar])

Input

firstname	John	*text*
lastname	Doe	*text*
dob	12/31/99	*text*

d. If necessary, enter the service input data in the fields, then click Test.

The service executes. If the test succeeds, the Test Result page is displayed. A successful test result displays the event document, the service input document, and the service output document.



This page shows the results from testing a event.

Generated event of type CustomerInserted on application view EastCoast.Sales.CustomerManagement

```
<?xml version="1.0"?>
<!DOCTYPE Customer_Table.insert>
<Customer_Table.insert>
<Address1/>
<Address2/>
<Address3/>
<City/>
<Country/>
<DOB>Dec 31 1999 12:00AM</DOB>
<Email/>
<Fax/>
<FirstName>John</FirstName>
<LastName>Doe</LastName>
<MiddleName/>
<Phone/>
```

Input to service InsertCustomer on application view EastCoast.Sales.CustomerManagement

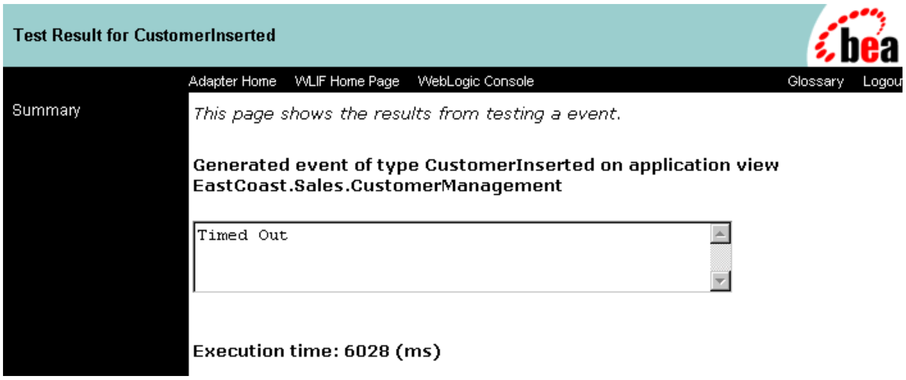
```
<?xml version="1.0"?>
<!DOCTYPE Input>
<Input>
  <firstname>John</firstname>
  <lastname>Doe</lastname>
  <dob>12/31/99</dob>
</Input>
```

Output from service InsertCustomer on application view EastCoast.Sales.CustomerManagement

```
<?xml version="1.0"?>
<!DOCTYPE RowsAffected>
<RowsAffected>1</RowsAffected>
```

Execution time: 3034 (ms)

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



This page shows the results from testing a event.

Generated event of type CustomerInserted on application view EastCoast.Sales.CustomerManagement

```
Timed Out
```

Execution time: 6028 (ms)

e. If the test failed, edit the event definition, or contact the system administrator or application manager.

f. If the test succeeded, repeat the test procedure for each remaining event you want to test.

g. When finished, save the application view.

If You Select Manual

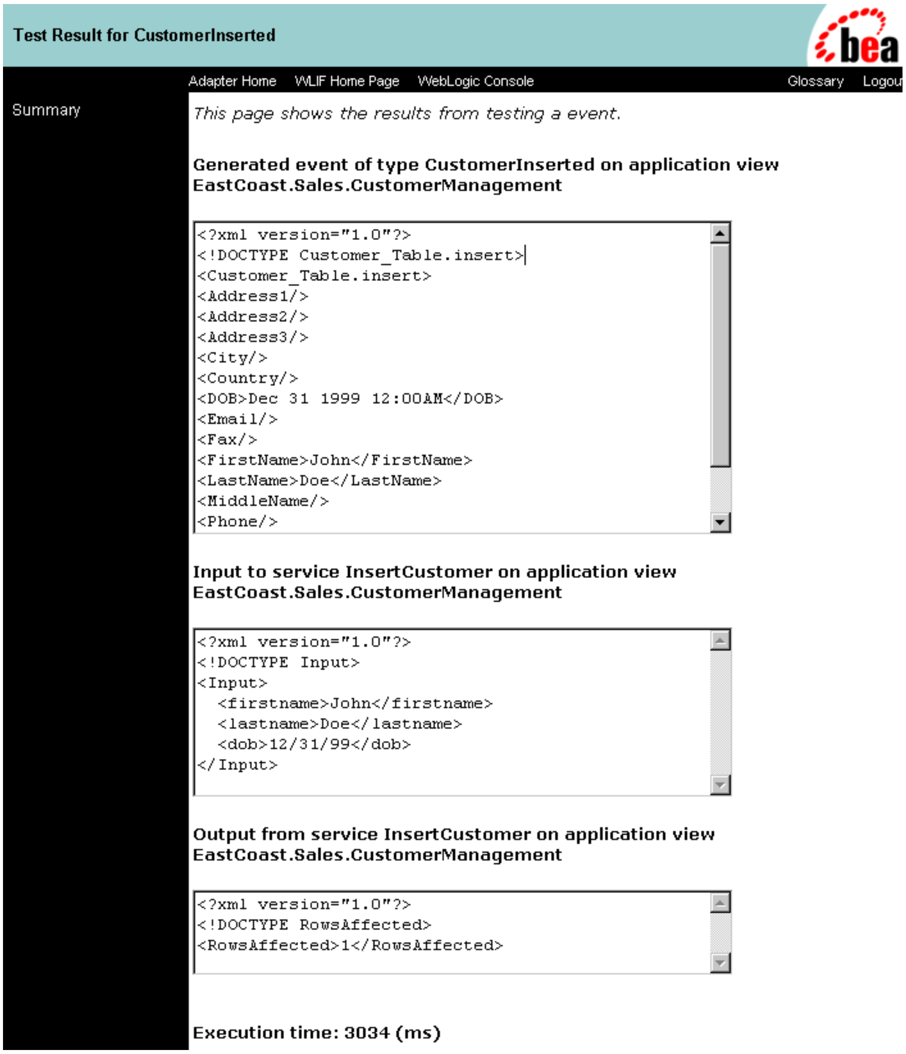
a. In the Time field, enter a reasonable time to wait, in milliseconds. (One minute = 60,000 ms.) If this time elapses before the event succeeds, the test will time out and display a failure message.

b. Open the application you will use to trigger the event, if the application is not already open.

c. Click Test. The test waits for an event to trigger it.

d. Using the triggering application, perform an action that will execute the service that will test the application view event.

   If the test succeeds, the Test Result page is displayed. A successful test result displays the event document from the application, the service input document, and the service output document.



This page shows the results from testing a event.

Generated event of type CustomerInserted on application view EastCoast.Sales.CustomerManagement

```
<?xml version="1.0"?>
<!DOCTYPE Customer_Table.insert>
<Customer_Table.insert>
<Address1/>
<Address2/>
<Address3/>
<City/>
<Country/>
<DOB>Dec 31 1999 12:00AM</DOB>
<Email/>
<Fax/>
<FirstName>John</FirstName>
<LastName>Doe</LastName>
<MiddleName/>
<Phone/>
```

Input to service InsertCustomer on application view EastCoast.Sales.CustomerManagement

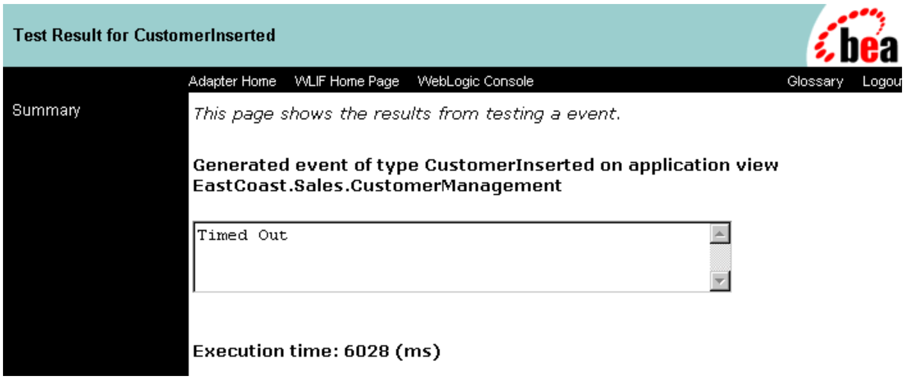
```
<?xml version="1.0"?>
<!DOCTYPE Input>
<Input>
  <firstname>John</firstname>
  <lastname>Doe</lastname>
  <dob>12/31/99</dob>
</Input>
```

Output from service InsertCustomer on application view EastCoast.Sales.CustomerManagement

```
<?xml version="1.0"?>
<!DOCTYPE RowsAffected>
<RowsAffected>1</RowsAffected>
```

Execution time: 3034 (ms)

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



Test Result for CustomerInserted

This page shows the results from testing a event.

Generated event of type CustomerInserted on application view EastCoast.Sales.CustomerManagement

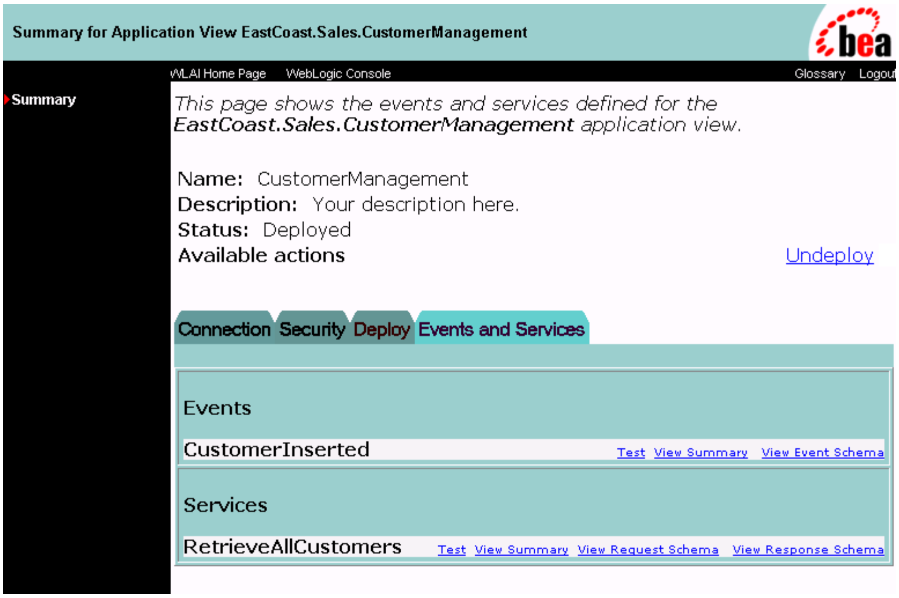
```
Timed Out
```

Execution time: 6028 (ms)

e. If the test failed, edit the event definition, or contact the system administrator or application manager.

f. If the test succeeded, repeat the test procedure for each remaining event you want to test.

g. When finished, save the application view.



Summary for Application View EastCoast.Sales.CustomerManagement

This page shows the events and services defined for the EastCoast.Sales.CustomerManagement application view.

Name: CustomerManagement
Description: Your description here.
Status: Deployed
Available actions

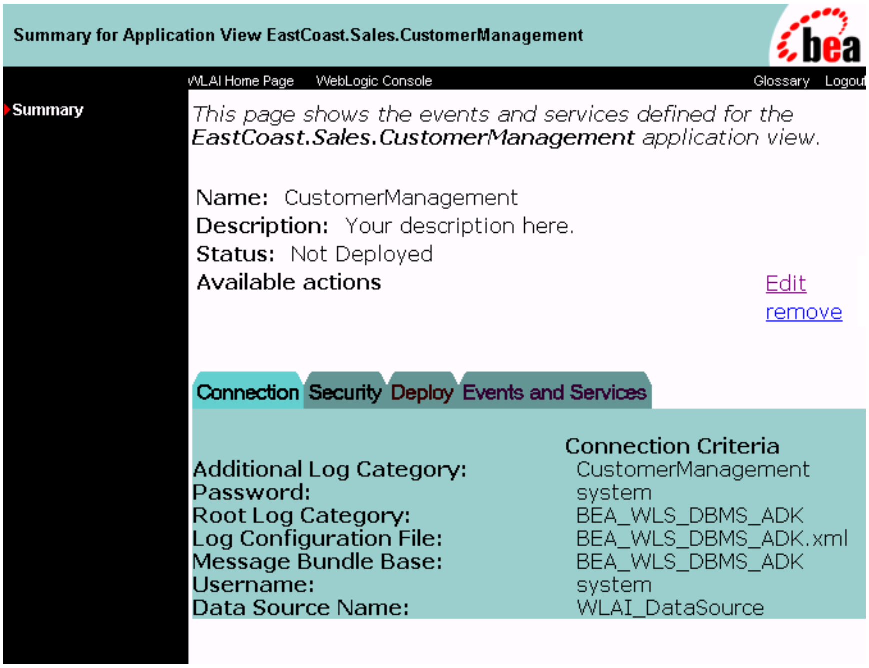
Events
CustomerInserted

Services
RetrieveAllCustomers

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.
2. Click Summary. The Summary for Application View page is displayed.

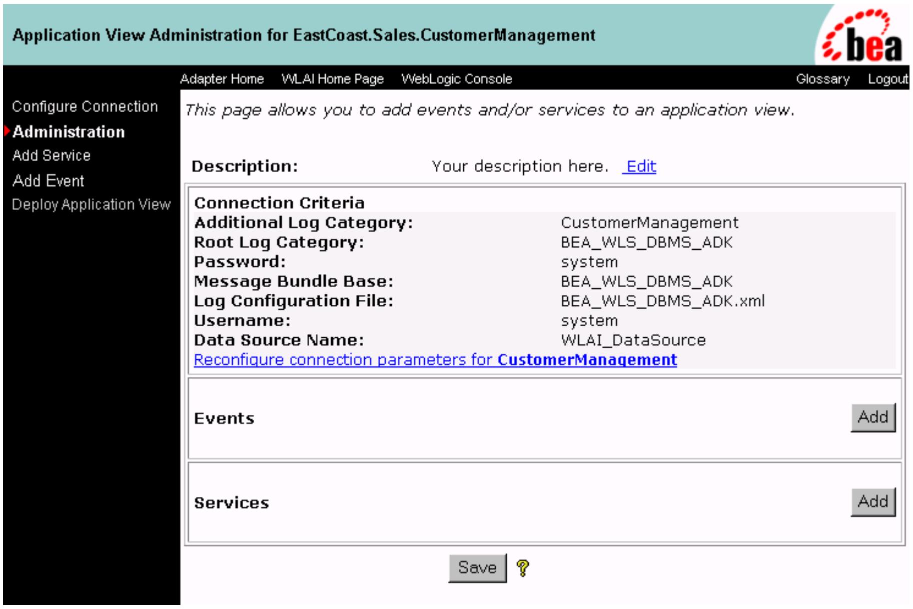

Summary for Application View EastCoast.Sales.CustomerManagement

This page shows the events and services defined for the ***EastCoast.Sales.CustomerManagement*** *application view.*

Name: CustomerManagement
Description: Your description here.
Status: Not Deployed
Available actions Edit remove

	Connection Criteria
Additional Log Category:	CustomerManagement
Password:	system
Root Log Category:	BEA_WLS_DBMS_ADK
Log Configuration File:	BEA_WLS_DBMS_ADK.xml
Message Bundle Base:	BEA_WLS_DBMS_ADK
Username:	system
Data Source Name:	WLAI_DataSource

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



This page allows you to add events and/or services to an application view.

Description: Your description here.

Connection Criteria

Additional Log Category:	CustomerManagement
Root Log Category:	BEA_WLS_DBMS_ADK
Password:	system
Message Bundle Base:	BEA_WLS_DBMS_ADK
Log Configuration File:	BEA_WLS_DBMS_ADK.xml
Username:	system
Data Source Name:	WLAI_DataSource

Reconfigure connection parameters for **CustomerManagement**

Events

Services

4. To reconfigure the application view’s connection parameters, click Configure Connection (See “Defining an Application View” on page 2-6.)

5. To add services and events, click Add Service or Add Event. For more information, see “Adding a Service to an Application View” on page 2-9 or “Adding an Event to an Application View” on page 2-11.

CHAPTER

3 Using Application Views in Business Process Management

This section contains information on the following subjects:

- Before You Begin
- Introduction to Using Application Views in BPM
- Using an Application View in BPM

Before You Begin

The following prerequisites must have been met before you can invoke an application view service or receive an application view event in business process management (BPM):

- You have created an application view and defined services and events for the application view.
- The application view and its adapter are functional and saved. If you plan on calling application view services and events from a running workflow, the application view must be deployed, as well.
- BPM is running.
- Application integration is running.
- The application integration plug-in has been loaded.
- You have received information about the required business logic for the workflows you are defining. This information usually comes from the business analyst or someone similar.
- A workflow template definition is open.

Introduction to Using Application Views in BPM

After you create all the required application view services and events for your enterprise, use the application views to execute your business processes. The simplest way to do this is by using BPM to design business process workflows that use the application view services and events.

BPM provides a GUI-based environment for designing business process workflows. These workflows can include application view services and events defined using application integration. For complete information on BPM, see *Using the WebLogic Integration Studio.*

Using an Application View in BPM

There are four ways to use application view services and events in BPM:

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

Use these scenarios in combination with each other to create your own workflows. This document does not fully explain how to use BPM. For complete information on BPM, see *Using Business Process Management* or see http://edocs.bea.com.

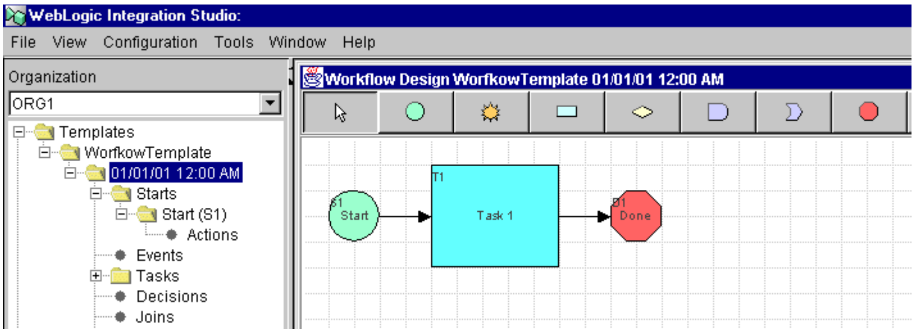
Scenario 1: Setting Up a Task Node to Call an Application View Service

In your organization, there may be situations in which you want to call an application view service from within a workflow. To do this, add a task node to the workflow, then add an appropriate Application View Service action to the task node. When the workflow is saved and activated, the application view service will be called whenever the task node executes.

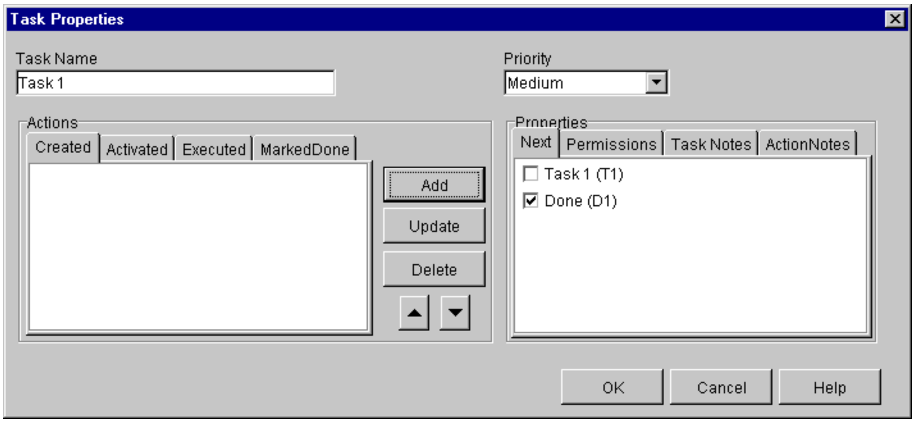
Steps for Setting up a Task Node to Call an Application View Service

Follow these steps to create a task node that calls an application view service:

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

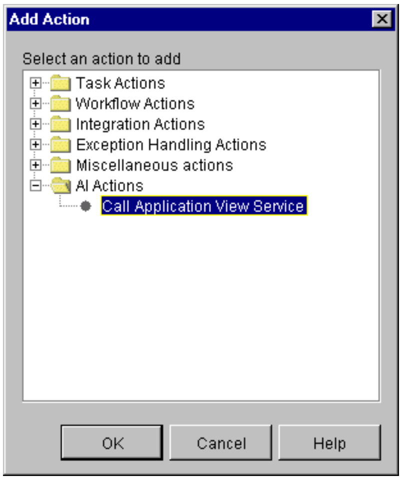


2. Create a task node if one does not already exist.
3. Double-click the task node that will call the application view service. The Task Properties dialog box is displayed.

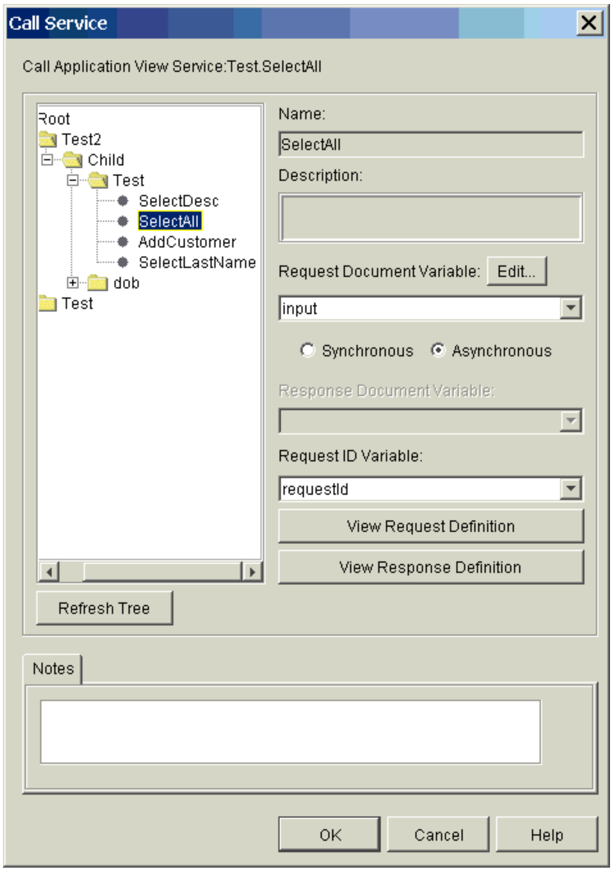


4. In the Actions area, select the tab from which you want the service to be called. Your tab choice depends on your business processes.

5. Click Add. The Add Action dialog box is displayed.



6. In the navigation tree, select AI Actions→Call Application View Service and click OK. The Call Service dialog box is displayed.



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

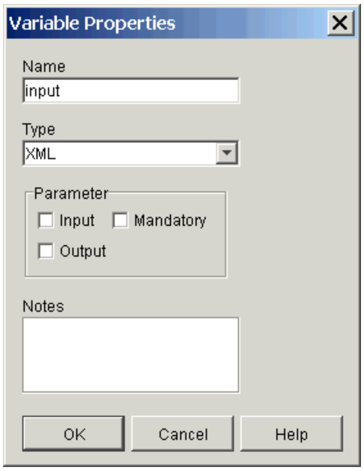
   The navigation tree organizes application view services by folder (for example, EastCoast.Sales) and application view (for example, CustomerManagement). All application view services are at the lowest level of the navigation tree.

   Note: To check for recently saved application views and events at any time, click Refresh Tree.

   If the navigation tree is missing or appears too narrow, it may be because an XML or string variable name is too long. Try renaming your XML or string variables so they are shorter.

8. In the Request Document Variable list, select an existing XML variable that contains the input data for the application view service.

9. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, where you can create a new XML variable.



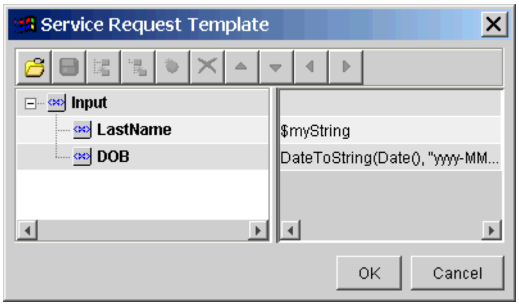
10. In the Name field, enter a name for the variable.

11. In the Type menu, select XML. XML is the only menu option.

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

12. Click OK to return to the Call Service dialog box.

13. (Optional) Click Set ... or Edit ... to display the Service Request Template dialog box, where you can create a service request template for the selected service (Set ...) or edit an existing service request template (Edit ...).



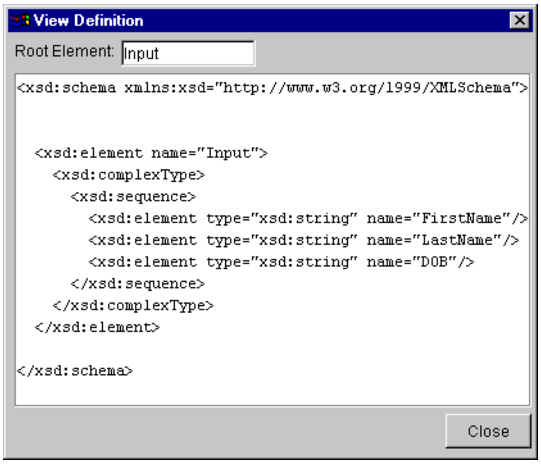
The Service Request Template dialog box displays the template to apply to all service requests of this type. This template is based on the input schema for the service.

When this action executes, the template data will be assigned to the specified request document variable and used as the input document for the service. This template will override any previous setting for the variable.

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

14. Click OK to return to the Call Service dialog box.

15. If you need to examine the XML schema of the input document, click View Request Definition. The View Definition dialog box is displayed.



Root Element: Input

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">


  <xsd:element name="Input">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element type="xsd:string" name="FirstName"/>
        <xsd:element type="xsd:string" name="LastName"/>
        <xsd:element type="xsd:string" name="DOB"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

16. Click Close when finished.

17. To call the application view synchronously, select Synchronous, or select Asynchronous to call the application view asynchronously.

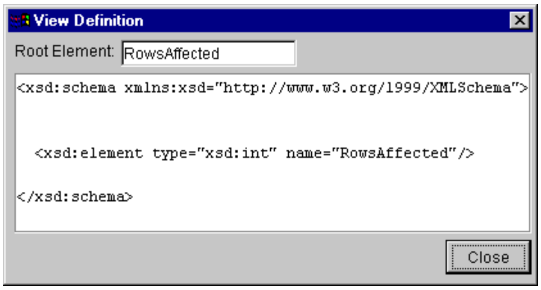
    Note: A node that synchronously calls a service will wait for the service to return a response document before the workflow can continue. If the node asynchronously calls a service, the workflow will continue.

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

19. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, where you can create a new XML variable. For details, see step 9. in "Scenario 1: Setting Up a Task Node to Call an Application View Service".

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

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

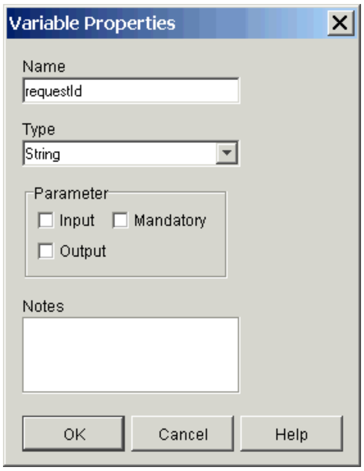


```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">


  <xsd:element type="xsd:int" name="RowsAffected"/>

</xsd:schema>
```

21. Click Close when finished.

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

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



Name: requestId

Type: String

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

25. In the Type menu, select String. String is the only menu option.

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

    Note: When you set up a task node to call an asynchronous application view service, the result will be returned to BPM. The workflow identifies this response using the request ID variable you selected. To set up an event node to receive the response, make sure to use the same request ID variable for the event node. For more information on creating such an event node, see "Scenario 2: Setting Up an Event Node to Wait for a Response from an Asynchronous Application View Service" on page 3-11.

26. Click OK to save the action.

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

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

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

Receiving an Asynchronous Application View Service Response

In a workflow, whenever an action calls an application view service asynchronously (See “Scenario 1: Setting Up a Task Node to Call an Application View Service” on page 3-4.), the application view service will return a response. Normally, if you care about the response, you will want to set up a corresponding asynchronous event node to wait for the response. This section explains a highly simplified scenario in which an event node receives an application view service response without checking for errors.

To set up an asynchronous event node to wait for a response from an asynchronous application view service, create an event node, then set up the event node to wait for an event of type AI Async Response.

You can use one of two methods to set up the event node to receive the asynchronous service response:

- By using the Response Document tab (preferred method). When you use this method, you receive the asynchronous service response by selecting the request ID variable and a response document variable. The request ID variable is a string and the response document variable is of type XML. For details on using this method, see “Steps for Receiving an Asynchronous Service Response (Preferred Method)” on page 3-13.
- By using the Asynchronous Variable tab (legacy method). When you use this method, you receive the asynchronous service response by selecting the request ID variable and an asynchronous service response variable. The request ID variable is a string and the asynchronous service response variable is of type AsyncServiceResponse. For details on using this method, see “Steps for Receiving an Asynchronous Service Response (Legacy Method)” on page 3-15.

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

Use a response document variable to receive asynchronous service responses whenever possible. Whenever you set up an Event Properties dialog box to wait for an event of type AI Async Response, you may or may not have the choice of using an asynchronous variable to receive the response.

- If you edit an existing AI Async Response event node that was previously set up to use an Asynchronous Service Response variable to receive the response, then two tabs will be displayed in the Event Properties dialog box: an Asynchronous Variable tab (legacy method) and a Response Document tab (preferred method). In this case, you can select one of the two methods to receive the service response.
- If you edit an existing AI Async Response event node that does not use an Asynchronous Service Response variable or you are creating a new AI Async Response event node, then the Event Properties dialog box will display a tabless dialog box, where you can set up a response document to receive the service response (preferred method).

Handling Errors in an Asynchronous Application View Service Response

Although this scenario does not handle errors returned in the application view service response, you will normally want to handle errors in your own workflows. To handle asynchronous service response errors in your workflows that use an AsyncServiceResponse variable, use the features included in the application integration plug-in.

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

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

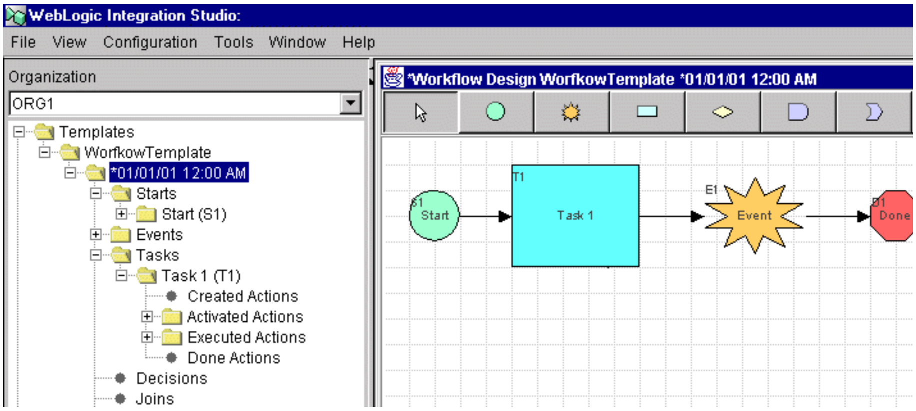
For complete documentation of these functions, see “Explanation of Functions Provided by the Application Integration Plug-in” on page 3-18.

Steps for Receiving an Asynchronous Service Response (Preferred Method)

To set up an asynchronous event node to wait for a response from an asynchronous application view service, create an event node, then set up the event node to wait for an event of type AI Async Response.

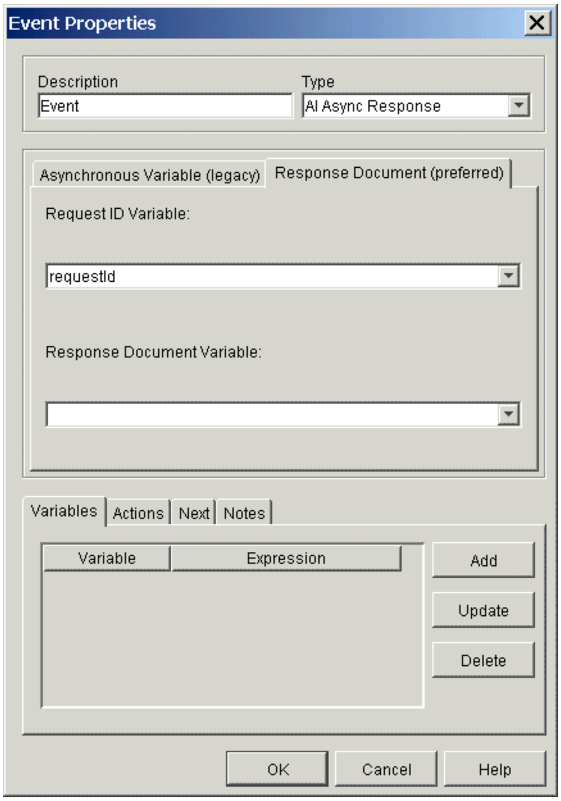
Follow these steps to set up an event node to use an XML variable to receive an asynchronous service response:

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



2. Create an event node if one does not already exist. This event node will wait for an asynchronous response from a designated application view service.

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



4. (Optional) In the Description field, enter a name.
5. In the Type list, select AI Async Response.
6. Select the Response Document (preferred) tab.

   Note: If your workflow does not use an AsyncServiceResponse variable or you are creating a new AI Async Response event node, then the Event Properties dialog box will display a tabless dialog box instead. Use it to set up a response document to receive the service response (preferred method).

7. In the Request ID Variable list, select an already-defined string variable. BPM will listen for an asynchronous response with an ID matching this variable.

8. If no suitable string variable exists, select <new> to open the Variable Properties dialog box, where you can create a new string variable. For details, see step 23. in “Scenario 1: Setting Up a Task Node to Call an Application View Service”.

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

   Note: The purpose of this event node is to wait for a response to a Call Application View Service action that was called asynchronously earlier in the workflow. The Call Application View Service action sets the request ID variable. To make the action and this event node work together, they must both use the same request ID variable. For more information on setting up the Call Application View Service action, see “Scenario 1: Setting Up a Task Node to Call an Application View Service” on page 3-4.

9. For asynchronous services that require storage of the response, select a predefined XML variable in the Response Document Variable list. When BPM receives the response from the application view service, the response document variable stores the response. If you do not care about the response data, skip this step.

10. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, where you can create a new variable. For details, see step 9. in “Scenario 1: Setting Up a Task Node to Call an Application View Service”.

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

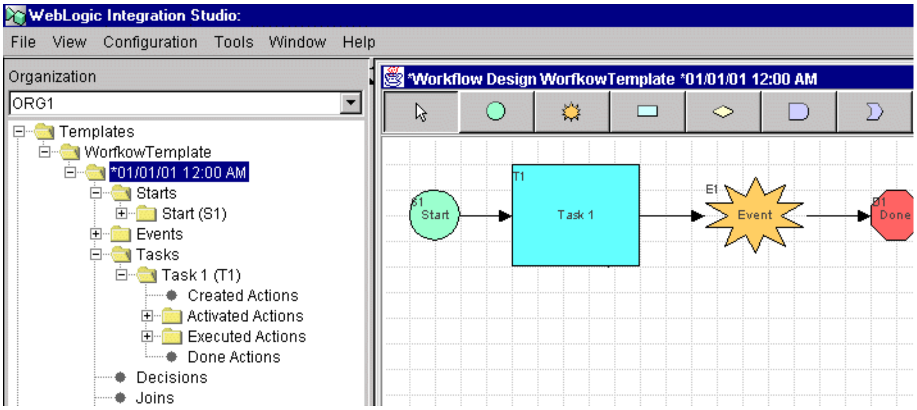
11. Click OK to save the event node.

Steps for Receiving an Asynchronous Service Response (Legacy Method)

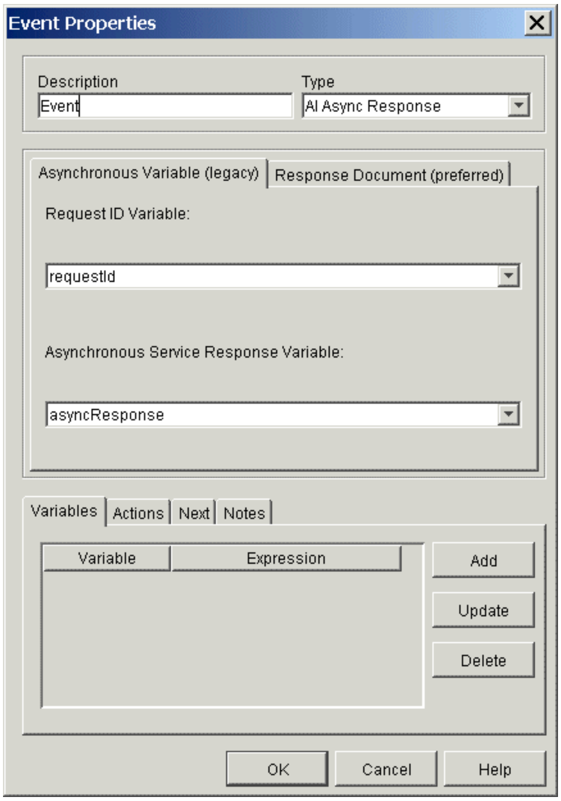
The preferred method for receiving an asynchronous service response is to use a response document variable of type XML. However, if an existing workflow contains an asynchronous event node that was previously set up to use an AsyncServiceResponse variable to wait for a response from an asynchronous application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

Follow these steps to modify an event node that uses an AsyncServiceResponse variable to receive an asynchronous service response:

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



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



3. Select the Asynchronous Variable (legacy) tab.

4. In the Request ID Variable list, select an already-defined string variable. BPM will listen for an asynchronous response with an ID matching this variable.

   Note: The purpose of this event node is to wait for a response to a Call Application View Service action that was called asynchronously earlier in the workflow. The Call Application View Service action sets the request ID variable. To make the action and this event node work together, they must both use the same request ID variable. For more information on setting up the Call Application View Service action, see "Scenario 1: Setting Up a Task Node to Call an Application View Service" on page 3-4.

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

> **Note:** Because you are modifying an existing asynchronous event node, the asynchronous service response variable field will already be populated. If you do not care about the response, select the Response Document (preferred) tab. For details on using the preferred method, see "Steps for Receiving an Asynchronous Service Response (Preferred Method)" on page 3-13.

6. Click OK to save the event node.

Explanation of Functions Provided by the Application Integration Plug-in

When using the application integration plug-in, use the functions `AIHasError()`, `AIGetErrorMsg()`, and `AIGetResponseDocument()` to interrogate AI Async Response variables, if applicable. If the application integration plug-in is installed in BPM, then you have access to these functions. Using these functions, you can set up decision nodes to handle success and failure conditions.

> **Note:** These functions support only the asynchronous variable method for receiving asynchronous service responses. For details, see "Steps for Receiving an Asynchronous Service Response (Legacy Method)" on page 3-15.

AIHasError()

Use `AIHasError()` to determine the status of an asynchronous service response.

Operands:

AsyncServiceResponse variable

Preconditions:

You have created a variable of type AsyncServiceResponse. You have called an asynchronous application view service. The application view service has returned a response, which is stored in your AsyncServiceResponse variable.

Returns:

Boolean

Output explanation:

False: The asynchronous application view service call was successful.

True: The asynchronous application view service call failed.

AIGetErrorMsg()

Use `AIGetErrorMsg()` to retrieve the error message string returned by an asynchronous application view service.

Operands:

AsyncServiceResponse variable

Preconditions:

You have created a variable of type AsyncServiceResponse. You have called an asynchronous application view service. The application view service has returned a response, which is stored by your AsyncServiceResponse variable.

Returns:

String

Output explanation:

Error string: Returns an error string explaining why the asynchronous application view response failed.

Empty string: There was no error.

AIGetResponseDocument()

Use `AIGetResponseDocument()` to retrieve the actual XML response document returned by an asynchronous application view service.

Operands:

AsyncServiceResponse variable

Preconditions:

You have created a variable of type AsyncServiceResponse. You have called an asynchronous application view service. The application view service has returned a response, which is stored by your AsyncServiceResponse variable.

Returns:

XML

Output explanation:

XML document: Returns an XML document representing the asynchronous service response.

Null: No response document was returned, because an error ocurred.

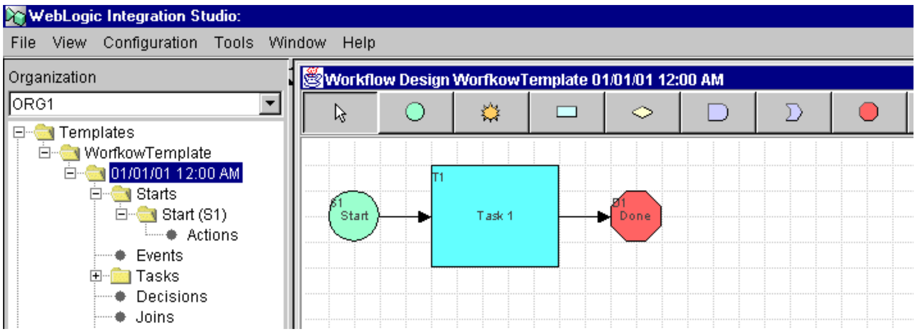
Scenario 3: Creating a Workflow Started by an Application View Event

You may want to create a workflow that starts whenever a designated application view event occurs. To set up a workflow to be started by an application view event, edit the workflow's start node so it responds to an event of type AI Start, then select the appropriate application view event. If necessary, you can set up conditions on which to filter the event. After you save and activate the workflow, the start node will execute each time the application view event occurs.

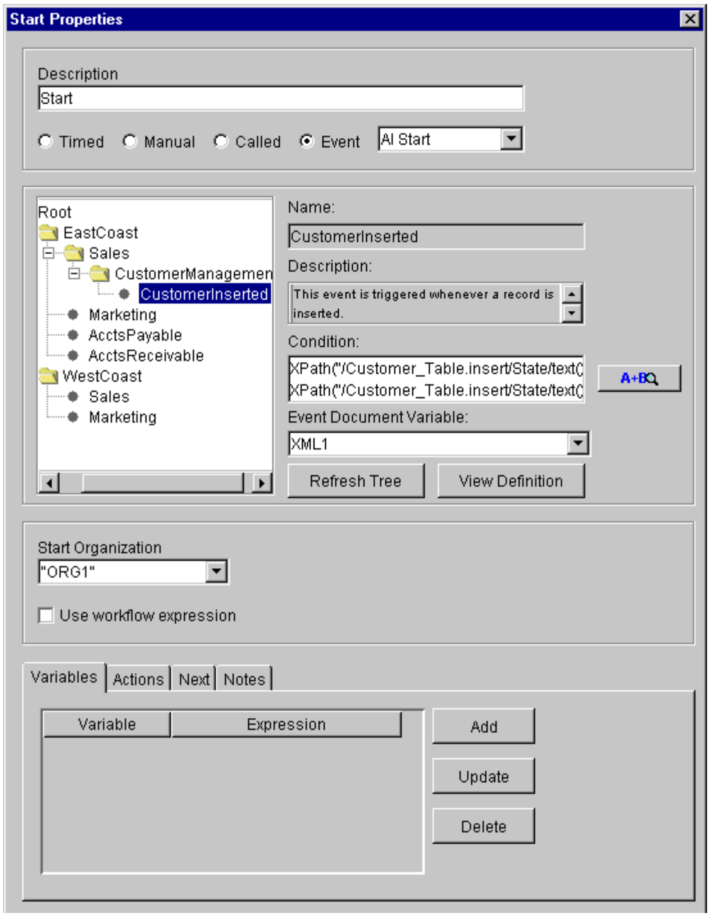
Steps for Creating a Workflow Started by an Application View Event

Follow these steps to set up a workflow with a start node that is triggered by an application view event.

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



2. Create a start node if one does not already exist. This start node will respond to an application view event that you specify.
3. Double-click the start node. The Start Properties dialog box is displayed.



4. (Optional) In the Description field, enter a name.
5. Click Event.
6. In the Event list, select AI Start.
7. In the navigation tree, navigate to and select the application view event.

The navigation tree organizes application view events by folder (for example, EastCoast.Sales) and application view (for example, CustomerManagement). All application view events are at the lowest level of the hierarchy.

Note: To check for recently saved application views and events at any time, click Refresh Tree.

If the navigation tree is missing or appears too narrow, it may be because an XML or string variable name is too long. Try renaming your XML or string variables so they are shorter.

8. If necessary, filter the event by entering a condition in the Condition field, or click the A + B button to display the Expression Builder dialog box.

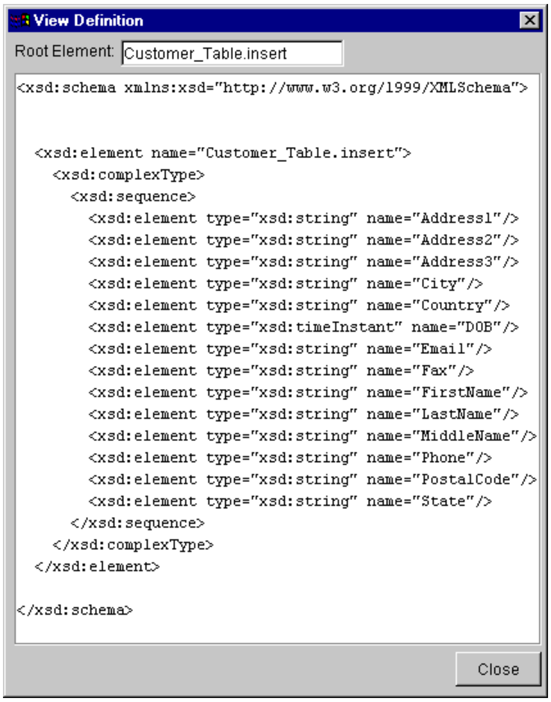
   For information on setting up conditions and XPath expressions, see *Using the WebLogic Integration Studio.*

9. In the Event Document Variable list, select an XML variable. When the start node receives data from the application view event, this variable stores the data. If you do not care about the event data, skip this step.

10. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, where you can create a new variable. For details, see step 9. in “Scenario 1: Setting Up a Task Node to Call an Application View Service”.

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

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



```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">


  <xsd:element name="Customer_Table.insert">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element type="xsd:string" name="Address1"/>
        <xsd:element type="xsd:string" name="Address2"/>
        <xsd:element type="xsd:string" name="Address3"/>
        <xsd:element type="xsd:string" name="City"/>
        <xsd:element type="xsd:string" name="Country"/>
        <xsd:element type="xsd:timeInstant" name="DOB"/>
        <xsd:element type="xsd:string" name="Email"/>
        <xsd:element type="xsd:string" name="Fax"/>
        <xsd:element type="xsd:string" name="FirstName"/>
        <xsd:element type="xsd:string" name="LastName"/>
        <xsd:element type="xsd:string" name="MiddleName"/>
        <xsd:element type="xsd:string" name="Phone"/>
        <xsd:element type="xsd:string" name="PostalCode"/>
        <xsd:element type="xsd:string" name="State"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

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

13. On the Start Properties dialog box, click OK. The start node is saved.

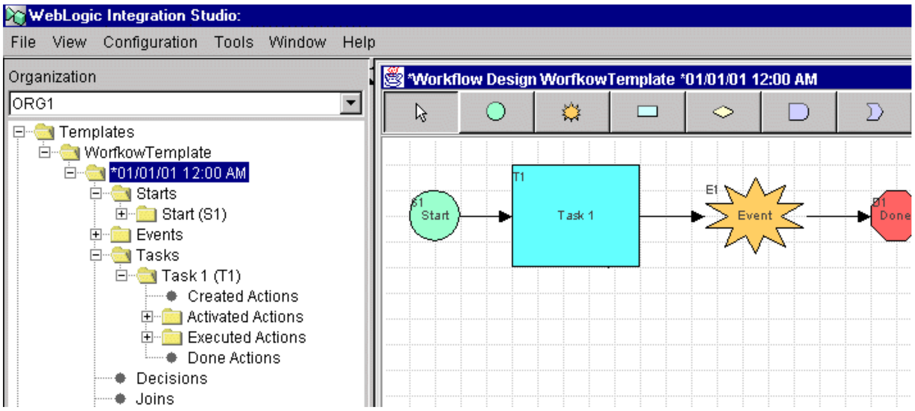
Scenario 4: Setting Up an Event Node to Wait for an Application View Event

In a workflow, you may want to create an event node that is triggered by an application view event. To set up an event node to respond to an application view event, edit the event node so it responds to an event of type AI Event, then select the appropriate application view event. If necessary, you can set up conditions on which to filter the application view event. After you save and activate the workflow, the workflow will progress to this event node, wait for a specified application view event, and continue processing.

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

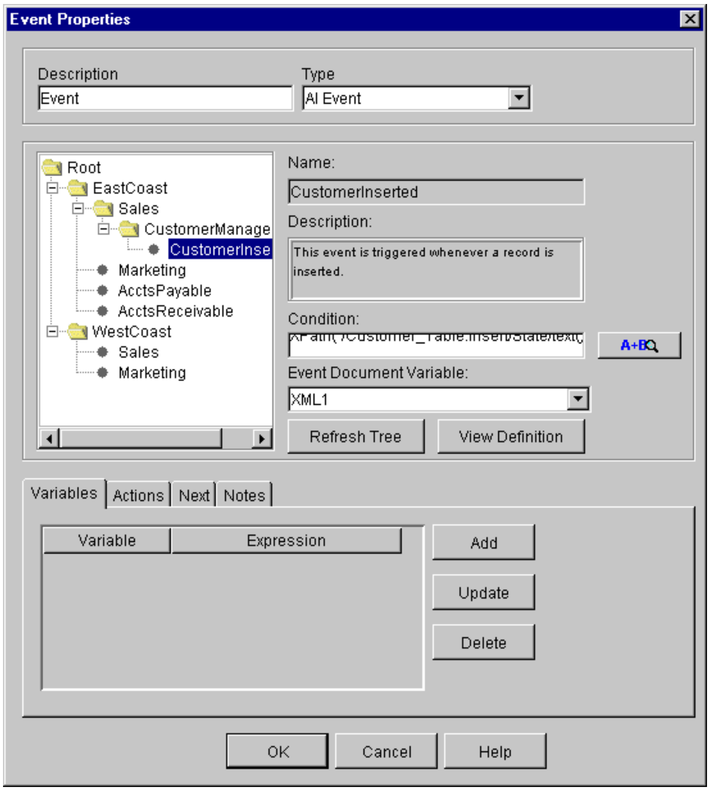
Follow these steps to set up an event node to be triggered by an application view event.

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



2. Create an event node if one does not already exist. This event node will be triggered by a designated application view event.

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



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

5. In the Type list, select AI Event.

6. In the navigation tree, navigate to and select an application view event.

   The navigation tree organizes application view events by folder (for example, EastCoast.Sales) and application view (for example, CustomerManagement). All application view events are at the lowest level of the hierarchy.

   Note: To check for recently saved application views and events at any time, click Refresh Tree.

   If the navigation tree is missing or appears too narrow, it may be because an XML or string variable name is too long. Try renaming your XML or string variables so they are shorter.

7. If necessary, filter the event by entering a condition in the Condition field, or click the A + B button to display the Expression Builder dialog box.

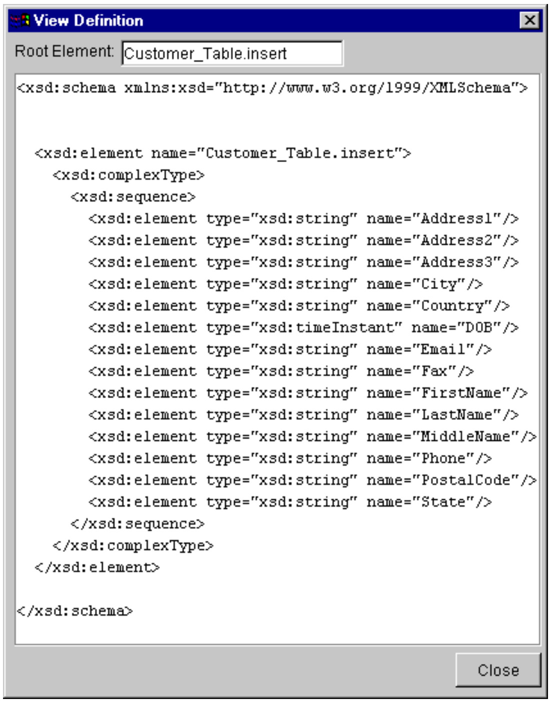
   For information on setting up conditions and XPath expressions, see *Using the WebLogic Integration Studio.*

   On the Event Properties dialog box, select an XML variable in the Event Document Variable list. When the event node receives data from the application view event, this variable stores the data. If you do not care about the event data, skip this step.

8. If no suitable XML variable exists, select <new> to open the Variable Properties dialog box, where you can create a new variable. For details, see step 9. in “Scenario 1: Setting Up a Task Node to Call an Application View Service”.

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

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



Root Element: Customer_Table.insert

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">


  <xsd:element name="Customer_Table.insert">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element type="xsd:string" name="Address1"/>
        <xsd:element type="xsd:string" name="Address2"/>
        <xsd:element type="xsd:string" name="Address3"/>
        <xsd:element type="xsd:string" name="City"/>
        <xsd:element type="xsd:string" name="Country"/>
        <xsd:element type="xsd:timeInstant" name="DOB"/>
        <xsd:element type="xsd:string" name="Email"/>
        <xsd:element type="xsd:string" name="Fax"/>
        <xsd:element type="xsd:string" name="FirstName"/>
        <xsd:element type="xsd:string" name="LastName"/>
        <xsd:element type="xsd:string" name="MiddleName"/>
        <xsd:element type="xsd:string" name="Phone"/>
        <xsd:element type="xsd:string" name="PostalCode"/>
        <xsd:element type="xsd:string" name="State"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

10. Click Close when finished.
11. On the Event Properties dialog box, click OK.

CHAPTER

4 Using Application Views by Writing Custom Code

If you are a developer, you may want to modify an application view by writing custom code. You can use most application view features by using its Web-based GUI, but there are some application view features you can use only by custom coding.

This section contains information on the following subjects:

- Scenario 1: Connecting Using Specific Credentials
- Scenario 2: Custom Coding a Business Process

Scenario 1: Connecting Using Specific Credentials

If necessary, you can invoke methods on an application view that let you set the security level before invoking services on the application view.

Use the new `ApplicationView` methods `setConnectionSpec()` and `getConnectionSpec()` to set the credentials for an EIS. Both methods use a `ConnectionSpec` object. To instantiate a `ConnectionSpec` object, 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 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 implement your own class.
2. If you are implementing your own `ConnectionSpec` class, include the following interfaces in it:
   - `ConnectionSpec` interface (JCA class)
   - `ConnectionRequestInfo` interface (JCA class)
   - `Map` interface (SDK class)
   - `Serializable` interface (SDK class)

Calling setConnectionSpec() and getConnectionSpec()

After you implement the `ConnectionSpec` class and instantiate a `ConnectionSpec` object, you can use it in conjunction with the following two new `ApplicationView` methods:

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

Listing 4-1 Complete Code for setConnectionSpec()

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

Listing 4-2 Complete Code for getConnectionSpec()

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

Using the ConnectionSpec

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

Listing 4-3 shows a specific example for using `ConnectionSpec`.

Listing 4-3 An Example That Uses ConnectionSpec

```
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 primary way to use application views in business processes is to use business process management (BPM), an alternate way is to write custom Java code to represent the business process. If you are a developer who uses the custom coding method, this section uses a simple example to demonstrate how to custom code a business process.

For a thorough comparison of the two ways to use application views, see “Deciding Which of the Two Methods to Use” on page 1-6.

About this Scenario

In the simple example used throughout this section, the following business logic is implemented:

An enterprise has a customer relationship management (CRM) system and an order processing (OP) system. You want a business process that coordinates the synchronization of customer information between these two systems. That means that whenever a customer is created on the CRM system, it should trigger the creation of a corresponding customer record on the OP system. The attached Java class `SyncCustomerInformation` implements this business logic.

This example does not cover everything you can do using custom code. It only demonstrates the basic steps you take when you implement your own organization's business processes.

Your role is to use this example code as a template 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 do the following two steps to create custom code that uses an application view in a business process:

1. Make sure a Java class exists to represent the application that implements the business process.
2. Within this Java class, supply the code to implement the business logic.

Before You Begin

The following prerequisites must be met before you write 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 about the required business logic for the business process workflow you are defining. This information usually comes from a business analyst. You have all the information necessary to connect to WebLogic Server,

including the host server name and port number, and a WebLogic Server user ID and password.

In addition, this particular scenario assumes the following prerequisites are already complete:

- Application views for the source CRM system and the target OP system are defined and working. For details on defining application views, see “Defining an Application View” on page 2-1.
- Both application views exist 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 will have its own folders and application views.
- You are familiar with the application integration API or are working closely with a Java programmer who is.
- You have all the information necessary to connect to the application integration server that hosts the application views.

  Note: For your organization, get this information from the system administrator.

Creating the SyncCustomerInformation Class

When writing custom code, a Java class must exist to represent each application required for the business process. Create the necessary Java classes if they do not exist already. This example calls for one application class called `SyncCustomerInformation.` Of course, your own code will use different variable names. To create the `SyncCustomerInformation` Java class:

1. See “Example Code for SyncCustomerInformation” on page 4-8 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 thoroughly commented.
2. Make sure the code does the following things (steps 3 through 11):
3. Create code to listen for East Coast.New Customer.

4. Obtain a reference to the `NamespaceManager` (variable name `m_namespaceMgr`) and `ApplicationViewManager` (variable name `m_appViewMgr`) within WebLogic Server. Accomplish this using a JNDI lookup from WebLogic Server.

5. Using the `NamespaceManager,` obtain a reference to the “root” namespace by calling `nm.getRootNamespace()`. This reference is stored in a variable called `root.`

6. Using the `root` variable, obtain a reference to the East Coast namespace by calling `root.getNamespaceObject(“East Coast”).` This reference is stored into a variable called `eastCoast.`

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

8. This `custMgmtHandle` temporary reference will be used to obtain an actual reference to an `ApplicationView` instance for Customer Management. Do this by calling the `ApplicationViewManager` as `avm.getApplicationViewInstance (custMgmtHandle.getQualifiedName()).` Store the returned reference into a variable called `custMgmt.`

9. Begin listening for New Customer events by calling `custMgmt.addEventListener(“New Customer”, listener)`, where `listener` is 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).

10. Implement the `onEvent` method of the listener class used in the step above.

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

    The `onEvent` method should then call 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 is called `handleNewCustomer.`

11. Implement the `handleNewCustomer` method that will respond to the New Customer event.

    The following things happen:

a. The `handleNewCustomer` method transforms the XML document 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` will then obtain a reference to an instance of the East Coast.Order Processing `ApplicationView` in the same way described for the East Coast.Customer Management `ApplicationView.` This reference is stored into a variable called `orderProc.`

c. `handleNewCustomer` will then invokes the Create Customer service on the East Coast.Order Processing `ApplicationView` 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` is finished and returns, leaving itself ready to handle the next incoming New Customer event.

When you are finished, a new Java class exists called `SyncCustomerInformation.` This class implements the Sync Customer Information business logic. This `SyncCustomerInformation` class uses the application integration API to get events from the CRM system and to invoke services on the OP system.

Example Code for SyncCustomerInformation

The following code listing is the full source code for the `SyncCustomerInformation` Java class. It implements the business logic for the scenario described earlier in this chapter. Use this example code as a guide for writing your own custom code to implement your enterprise's business processes.

Listing 4-4 Full Class Source Code for SyncCustomerInformation

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

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

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

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

    try
    {
      // Create an instance of SyncCustomerInformation to work with
      SyncCustomerInformation syncCustInfo =
        new SyncCustomerInformation(args[0], args[1], args[2]);

      // Get a connection to WLAI

      InitialContext initialContext = syncCustInfo.getInitialContext();

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

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

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

      custMgmt.addEventListener("NewCustomer", syncCustInfo);

      // Process up to 10 events and then quit.

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

    return;
  }

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

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

      // Handle it

      IDocument response = handleNewCustomer(newCustomerEvent.getPayload());
      // Print the response

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

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

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

  /**
   * Handles any 'New Customer' event by invoking the 'Create Customer'
   * service on the 'Order Processing' ApplicationView. The response
   * document from the service is returned as the return value of this
   * method.
   */
  public IDocument
  handleNewCustomer(IDocument newCustomerData)
    throws Exception
  {
    // Get an instance of the 'OrderProcessing' ApplicationView.
    if (m_orderProc == null)
    {
      m_orderProc =
        new ApplicationView(m_initialContext, "EastCoast.OrderProcessing");
    }

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

    IDocument createCustomerRequest =
      transformNewCustomerToCreateCustomerRequest(newCustomerData);

    // Invoke the service

    IDocument createCustomerResponse =
      m_orderProc.invokeService("CreateCustomer", createCustomerRequest);
    // Return the response

    return createCustomerResponse;
  }

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

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

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

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


  /**
   * The initial context to use when communicating with WLAI
   */
  private InitialContext m_initialContext;

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

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

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

  /**
```

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

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

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

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

/**
 * Establish an initial context to WLAI.
 */
public InitialContext
getInitialContext()
  throws NamingException
{
  // Set up properties for obtaining an InitialContext to the WLAI server.

  Hashtable props = new Hashtable();

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

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

  // Connect to the WLAI server

  InitialContext initialContext = new InitialContext(props);

  // Store this for later

  m_initialContext = initialContext;
```

```
    return initialContext;
  }


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

    return newCustomerData;
  }

  /**
   * Event processing/wait loop
   */
  public void
  processEvents()
  {
    synchronized(m_doneMonitor)
    {
      while (!m_done)
      {
        try
        {
          m_doneMonitor.wait();
        }
        catch (Exception e)
        {
          // ignore
        }
      }
    }
  }

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

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

CHAPTER

5 Using the WebLogic Integration Application View Console

This section contains information on the following subjects:

- Before You Begin
- Introduction to Using the Application View Console
- Steps for Using the Application View Console

Before You Begin

Before you attempt to work with Application View Console, ensure that application integration is running.

Introduction to Using the Application View Console

Use the Application View Console to access, organize, and edit all application views in your enterprise. You can use the Application View Console to create new folders and to add new application views to the folders. These folders allow you to organize your application views according to your own navigation scheme, regardless of the adapter to which the application view belongs.

Steps for Using the Application View Console

This section explains how to organize application views into folders using the Application View Console. The actual folders you set up depend on your organization.

Logging On to the Application View Console

The first step in managing application views is to log on to the Application View Console. To log on:

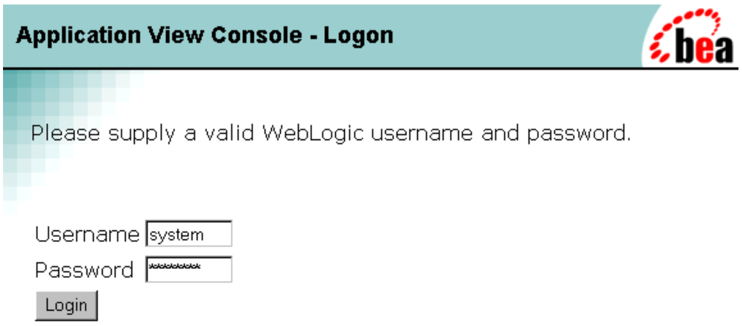
1. Launch a browser window.

2. Open the URL for your system's Application View Console. The actual URL you enter depends on your system. It should follow the format:

   `http://<yourserver>:<yourport>/wlai`

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

   The logon page is displayed.



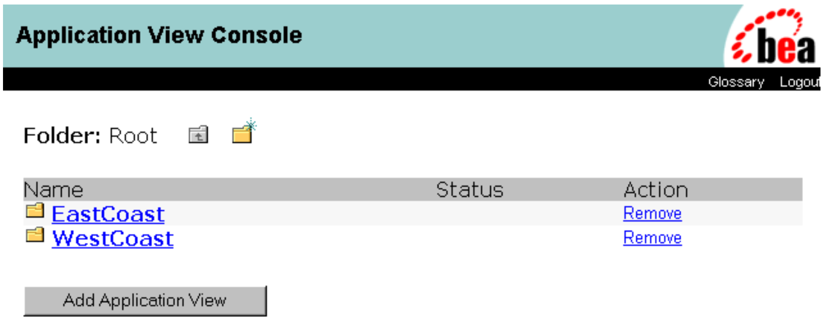
Application View Console - Logon

Please supply a valid WebLogic username and password.

Username: system
Password: ********

Note: If you have already logged in to WebLogic Server as system, then you will automatically skip the Application View Console logon page.

3. To log on to the Application View Console, enter your WebLogic Server username and password, then click OK. The Application View Console is displayed.



Application View Console

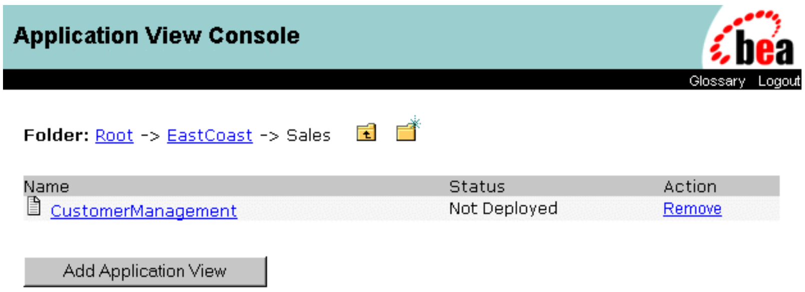
Folder: Root

Name	Status	Action
EastCoast		Remove
WestCoast		Remove

Creating a Folder

Create folders to organize the application views in your enterprise. Folders can contain application views and other folders. Once you create a folder, you cannot move it to another folder, and you can remove the folder only if it is empty. Once you create an application view in a folder, you can remove the application view, but you cannot move it to another folder. To create a folder:

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

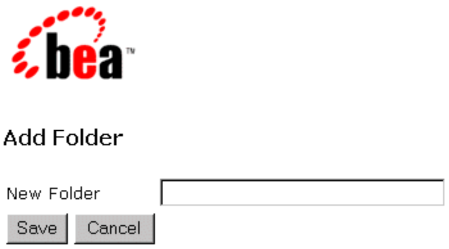

Application View Console

Glossary Logout

Folder: Root -> EastCoast -> Sales

Name	Status	Action
CustomerManagement	Not Deployed	Remove

2. Click the New Folder icon. The Add Folder page is displayed.


Add Folder

New Folder

3. In the New Folder field, enter a name. Valid characters are a–z, A–Z, 0–9, and _ (underscore).
4. Click Save.

Removing an Application View

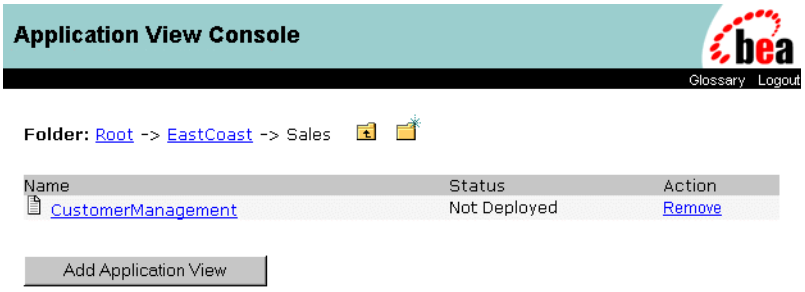
Remove application views when they become obsolete or the application is retired.

You can remove an application view only if the following conditions are true:

- You have undeployed the application view. (See “Undeploying an Application View” on page 2-18.) That is, the application view status reads Not Deployed.
- You are logged on to WebLogic Server using a user account that has the appropriate write privileges.

To remove an application view:

1. While logged on to the Application View Console, navigate to the folder where the target application view is located.


Application View Console

Folder: Root -> EastCoast -> Sales

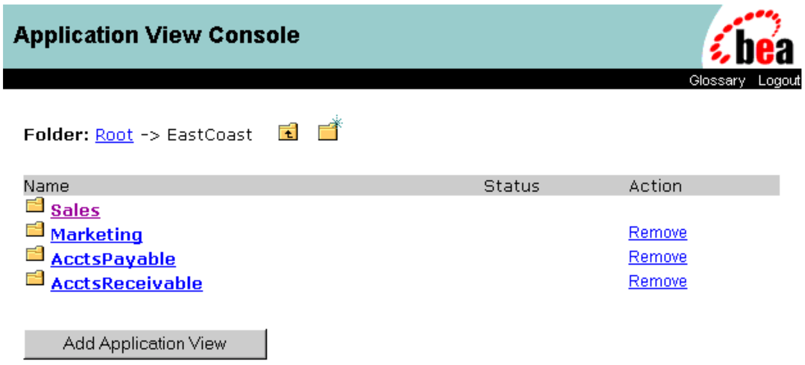
Name	Status	Action
CustomerManagement	Not Deployed	Remove

2. Click Remove to delete the application view.

Removing a Folder

Remove folders that are no longer needed. Before you can remove a folder, you must remove all of its application views and subfolders. To remove a folder:

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

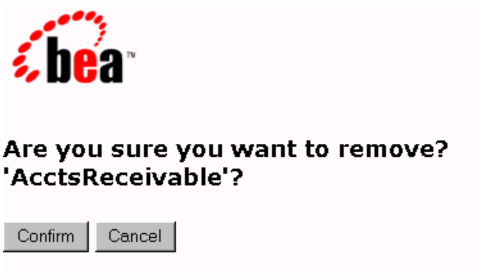

Application View Console

Glossary Logout

Folder: Root -> EastCoast

Name	Status	Action
Sales		
Marketing		Remove
AcctsPayable		Remove
AcctsReceivable		Remove

2. Click Remove to delete the folder. A confirmation page is displayed.


Are you sure you want to remove? 'AcctsReceivable'?

3. Click Confirm to delete the folder.

A Migrating Application Integration Data

This section includes information on the following topics:

- Overview
- Migrating Data Within the Same EIS Instance
- Migrating Data Within Different EIS Instances
- Recommended Practices

Overview

Application integration configuration data is stored in the same repository as data for business process management (BPM). Therefore, you can use the same tools to migrate application integration when migrating BPM data. However, there are some special considerations for migrating application integration data and deploying the migrated data in the target environment.

Migrating application integration data is straightforward when migrating between WebLogic Server domains the Enterprise Information System (EIS) when instances do not change. However, if the EIS instances do change, you must follow special procedures to ensure a working solution in the target environment.

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

- Migrating Data Within the Same EIS Instance
- Migrating Data Within Different EIS Instances

Migrating Data Within the Same EIS Instance

This section describes how to migrate application integration data between WebLogic Server domains, when the EIS instances involved do not change. An example of this type of migration is moving application view definitions between repositories for different domains of WebLogic Integration. In this case, only the WebLogic Integration domain changes, but the target EIS instances referred to in the application views remain the same.

In this case, the BPM package import/export utility makes migrating data simple. It involves exporting a package from BPM in the source domain, and importing that package into BPM in the target domain.

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

Export

When exporting a workflow that utilizes application integration, the BPM export tool automatically identifies the application views and other resources the workflow depends on. Listing A-1and Listing A-2 show general values identifying an application view and the resources it depends on in the export tool. In general, the application view will be located (in the BPM export tool) in the location shown in Listing A-1.

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

```
All Workflow Objects
  |--  XML Repository
```

```
    |-- Folder: WLAI.Namespace.Root
      |-- Folder: WLAI.Namespace.Root.<first folder>
        |-- Folder: WLAI.Namespace.Root.<first folder>.<nth folder>
          |-- Entity: WLAI.ApplicationView.Root.<first folder>.
                   <nth folder>.<appview name>
```

In general, all entities related to the application view can be found under the `<nth folder>`, and will be named according to the convention shown in Listing A-2. All application views may not follow this convention.

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

```
Entity: WLAI.<entity type>.Root.<first folder>.<nth folder>.<appview
name>_<event/service name>_<adapter_specific>
```

To fully export an application view, you *must* select all entities that are related to the application view, if not already selected. This includes entities of type `Schema` and type `ConnectionFactory`.

Export Example

For example, an application view named `CustomerManagement` in the folder `EastCoast.Sales` would be displayed in the BPM export tool at the location shown in Listing A-3.

Listing A-3 Application View in the BPM Export Tool

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

In order to fully export the `CustomerManagement` application view, the export tool automatically selects all entities that conform to the following pattern

```
Entity: WLAI.<entity type>.Root.EastCoast.Sales.CustomerManagement
```

For example, the `CustomerManagement` application view may contain several events and services. The export utility shows one `Schema` type entity for each event and two `Schema` type entities for each service. For example, where the `CustomerManagement` application view uses the DBMS adapter and has one event named`CustomerCreated` and one service named `CreateCustomer`, the entities shown in Listing A-4 are shown in the BPM export utility.

Listing A-4 Entities Used by the Application View

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

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

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

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

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

Each of these entities must be selected to properly export the `CustomerManagement` application view.

Import

Use the BPM package import utility to import a package containing application integration data. This utility automatically imports all entities you exported into the package. Before you can use the application views you just imported, you must ensure that they are deployed.

Deploy your imported application views using the WebLogic Integration Application View Console (generally located at `http://<server>:<port>/wlai`). Navigate through the imported folders to find the imported application view. Select the application view, and click the Deploy button on the Application View Summary page. This makes the application view ready for use in the target environment.

Migrating Data Within Different EIS Instances

This and the previous scenarion use the same procedures for export and import; however, some additional steps are required in the import procedure.

Special care must be taken when migrating data between WebLogic Server domains and between different instances of an EIS, because application views defined against one EIS instance contain identifiers and other data specific to that EIS instance. This is also true of the connection factory used by the application view.

You must manually change EIS-instance-specific data in your application view or connection factory. You can make these changes from the Application View Console by navigating to the desired application view and editing the application view. You must identify and update all EIS-specific data in the application view and its events, services and associated connection factory. Search for any EIS-instance-specific references, and replace them with references to the new EIS instance in the target environment.

In particular, you must edit the application view and connection factory definitions. Application view definitions may need changes in the following areas:

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

Import Example

In the CustomerManagement example, we have a database called CUST in the source environment, and we have a database called CustDB in the target environment. Listing A-5 shows the XML text that represents the application view and connection factory. More specifically, the example shows the application view descriptor for the CustomerManagement application view. When you use the Application View Console, you will need to use the appropriate fields in the design-time UI forms to see view and edit this information.

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

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

  …

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

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

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

</applicationView>
```

Note that this application view contains:

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

Each adapter places different properties into the service and event descriptors of application view descriptors it creates. Refer to your adapter documentation for information on what properties must be changed to operate successfully against a new EIS instance.

The connection factory descriptor will also need to be changed to refer to the new EIS instance. Listing A-6 shows sample connection factory.

Listing A-6 Example Connection Factory

```
<?xml version="1.0"?>
<!DOCTYPE connection-factory-dd>
<connection-factory-dd name="CustomerManagement_connectionFactory">
  <jndi-
name>com.bea.wlai.connectionFactories.EastCoast.Sales.CustomerManagement_connec
```

```
tionFactoryInstance</jndi-name>
  <pool-parms allowPoolToShrink="true"
      maxPoolSize="10"
      minPoolSize="0"
      targetFractionOfMaxPoolSize="0.1"/>
  <mcf-parm name="MessageBundleBase">
    <mcf-parm-value>BEA_WLS_DBMS_ADK</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="DataSourceName">
    <mcf-parm-value>eventSource</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="AdditionalLogContext">
    <mcf-parm-value>CustomerManagement</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="UserName">
    <mcf-parm-value>system</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="Password">
    <mcf-parm-value>security</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="RootLogContext">
    <mcf-parm-value>BEA_WLS_DBMS_ADK</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="PingTable">
    <mcf-parm-value>CUST.dbo.CUSTOMER_TABLE</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="LogLevel">
    <mcf-parm-value>WARN</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="LogConfigFile">
    <mcf-parm-value>BEA_WLS_DBMS_ADK.xml</mcf-parm-value>
  </mcf-parm>
  <adapter-logical-name>BEA_WLS_DBMS_ADK</adapter-logical-name>
</connection-factory-dd>
```

Note that this connection factory descriptor refers to directly to the `CUST` database and to a JDBC data source named `eventSource`. To ensure this connection factory will operate properly in the target environment, you must change the reference to `CUST` to be `CustDB`, and change the `eventSource` JDBC data source reference to refer to a valid JDBC data source (pointing at the new DBMS hosting `CustDB`) in the target domain.

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

Recommended Practices

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

- Wherever possible, set up identical EIS instances in both the source and target domain. For example, use a source and target database (for application views that use the DBMS adapter) that have the same type (for example, MS SQL Server to MS SQL Server), the same name, user accounts, and database objects. This eliminates the need to manually edit application view and connection factory descriptors.
- Remember to change the event router URL to reflect the event router’s location in the target environment. This can be changed by editing the application view from the application integration Application View Console.
- Remember to deploy your application views after they are imported using the Application View Console.

Index