BEA WebLogic Integration™

Programming BPM Client Applications

Release 7.0
Document Date: June 2002

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Programming BPM Client Applications

Part Number	Date	Software Version
N/A	June 2002	7.0

Contents

5. Using Value Objects

6. Establishing JMS Connections

7. Understanding the BPM Transaction Model

8. Disconnecting from the Process Engine

9. Configuring the Security Realms

11. Configuring Event Keys

12. Configuring Business Calendars

16. Managing Task Routing

17. Managing the XML Repository

18. Publishing Workflow Objects

23. Monitoring Run-Time Variables

24. Monitoring Workflow Exceptions

A. DTD Formats

B. Value Object Summary

C. EJB and Java Class Descriptors

About This Document

This document introduces the WebLogic Integration business process management (BPM) API, and describes how to use the API to create a custom clients for configuration, design, run-time management, or monitoring.

This document is organized as follows:

- Chapter 1, "Business Process Management API Development," provides an introduction to developing applications with the WebLogic Integration BPM API. Specifically, this chapter offers an overview of both the WebLogic Integration process engine and the public API, a summary of the main tasks in the application development process, and descriptions of the API examples from which the code samples presented in this document are taken.
- Part I, "API Development Fundamentals," provides detailed chapters describing each of the fundamental tasks required to develop a BPM client application:
  - Chapter 2, "Importing Packages and Interfaces," describes the packages and interfaces commonly used by BPM client applications that you should consider importing.
  - Chapter 3, "Connecting to the Process Engine," explains how to connect to the WebLogic Integration process engine, specifically by accessing the session EJBs.
  - Chapter 4, "Accessing Process Engine Information," explains how to access information about the server using the `ServerProperties` EJB.
  - Chapter 5, "Using Value Objects," explains how to create and/or access state information using the BPM value objects.
  - Chapter 6, "Establishing JMS Connections," explains how to connect to JMS from your client applications.
  - Chapter 7, "Understanding the BPM Transaction Model," describes the BPM transaction model, including examples.

- Chapter 8, “Disconnecting from the Process Engine,” explains how to disconnect from the WebLogic Integration process engine and release other resources.

- Part II, “Configuration,” provides detailed chapters describing each of the four EJB methods—`Admin`, `EJBCatalog`, `Permission`, and `WLPIPrincipal`—used for configuration:
  - Chapter 9, “Configuring the Security Realms,” explains how to configure the security realms.
  - Chapter 10, “Configuring Business Operations,” explains how to configure business operations.
  - Chapter 11, “Configuring Event Keys,” explains how to configure event keys.
  - Chapter 12, “Configuring Business Calendars,” explains how to configure business calendars.
- Part III, “Design,” provides detailed chapters describing each of the `Admin` EJB methods used for the design of business processes:
  - Chapter 13, “Creating and Managing Workflow Templates,” explains how to create and manage workflow templates.
  - Chapter 14, “Creating and Managing Workflow Template Definitions,” explains how to create and manage workflow template definitions.
  - Chapter 15, “Managing Tasks,” explains how to manage tasks that are defined as part of the template definition creation process.
  - Chapter 16, “Managing Task Routing,” explains how to define and manage tasks reroutes, and reroute a task from one user or role to another for a specified period of time.
  - Chapter 17, “Managing the XML Repository,” explains how to manage the XML repository, the data storage facility for WebLogic Integration business process components.
  - Chapter 18, “Publishing Workflow Objects,” explains how to create, export, and import workflow objects.
- Part IV, “Run-Time Management,” provides detailed chapters describing each of the `Worklist` EJB methods used for run-time management:

- Chapter 19, “Managing the Active Organization,” explains how to manage active organizations.
- Chapter 20, “Manually Starting Workflows,” explains how to manually start workflows.
- Chapter 21, “Managing Run-Time Tasks,” describes how to get, execute, assign, and sort tasks; respond to a client request; mark tasks as done or undone; set task properties; and invoke an exception handler.

- Part V, “Monitoring,” provides detailed chapters describing each of the methods—`Admin` EJB and `com.bea.wlpi.common.WorkflowException`—used for monitoring:
  - Chapter 22, “Monitoring Run-Time Workflow Instances,” explains how to monitor run-time workflow instances.
  - Chapter 23, “Monitoring Run-Time Variables,” explains how to monitor run-time variables.
  - Chapter 24, “Monitoring Workflow Exceptions,” describes the error handling and auditing facilities.
- Appendix A, “DTD Formats,” provides detailed descriptions of the BPM DTD formats.
- Appendix B, “Value Object Summary,” provides detailed descriptions of the BPM value object get and set methods.
- Appendix C, “EJB and Java Class Descriptors,” describes the EJB and Java class descriptor objects and their methods.
- Appendix D, “Database Schema,” provides the BPM database schema.
- Appendix E, “Customizing Studio and Worklist Logos and Text,” explains how to customize the logos and text for the BEA WebLogic Integration Studio and Worklist clients.

What You Need to Know

This document is intended for application developers who are interested in creating custom configuration, design, run-time management, and/or monitoring clients,or just gaining a better understanding of the BPM API. It is assumed that the reader is familiar with the WebLogic Integration product, Java programming, and XML.

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 *e-docs* Product Documentation page at the following URL:

`http://e-docs.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 WebLogic Integration documentation Home page on the e-docs Web site (and also on the documentation CD). 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 WebLogic Integration documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at the following URL:

`http://www.adobe.com/`

Related Information

The following WebLogic Integration documents contain information that may be helpful to programmers that are using and interfacing with the BPM client applications, Studio and Worklist. These applications have been built using the BPM API.

Note: The Worklist client application is being deprecated as of this release of WebLogic Integration. For information about the features that are replacing it, see the *BEA WebLogic Integration Release Notes*.

- *Using the WebLogic Integration Studio*
- *Using the WebLogic Integration Worklist*
- *Learning to Use BPM with WebLogic Integration*
- *BEA WebLogic Integration Javadoc*

For information on programming BPM plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.

For general information about Java applications, go to the Sun Microsystems, Inc. Java Web site at the following URL:

`http://java.sun.com/`

For general information about XML and XML parsers, go to the O'Reilly & Associates, Inc. XML.com Web site at the following URL:

`http://www.xml.com/`

Contact Us!

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

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

If you have any questions about this version of WebLogic Integration, or if you have problems installing and running WebLogic Integration, contact BEA Customer Support through BEA WebSupport at **www.bea.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
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

Convention	Item

`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:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

Convention	Item

. . .	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
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Business Process Management API Development

This section provides an overview of application development using the business process management (BPM) API. It includes the following topics:

- Introduction
- WebLogic Integration Process Engine
- BPM API
- BPM Application Development Tasks
- BPM API Examples

Introduction

To support business process management (BPM), WebLogic Integration provides the following components:

- Process engine—BPM server
- WebLogic Integration Studio—Design client
- Worklist—Run-time management client

  Note: The Worklist client application is being deprecated as of this release of WebLogic Integration. For information about the features that are replacing it, see the *BEA WebLogic Integration Release Notes*.

These components make up the basic BPM framework for designing, executing, and monitoring business processes, and administering related data with WebLogic Integration.

In addition to using these design and run-time management clients, you can create custom clients to manage your business processes, and/or enhance the existing set of user interface features using the BPM application programming interface (API).

This document describes how to use the BPM API to create custom configuration, design, run-time management, and monitoring clients. For more information about the API, see the *BEA WebLogic Integration Javadoc*.

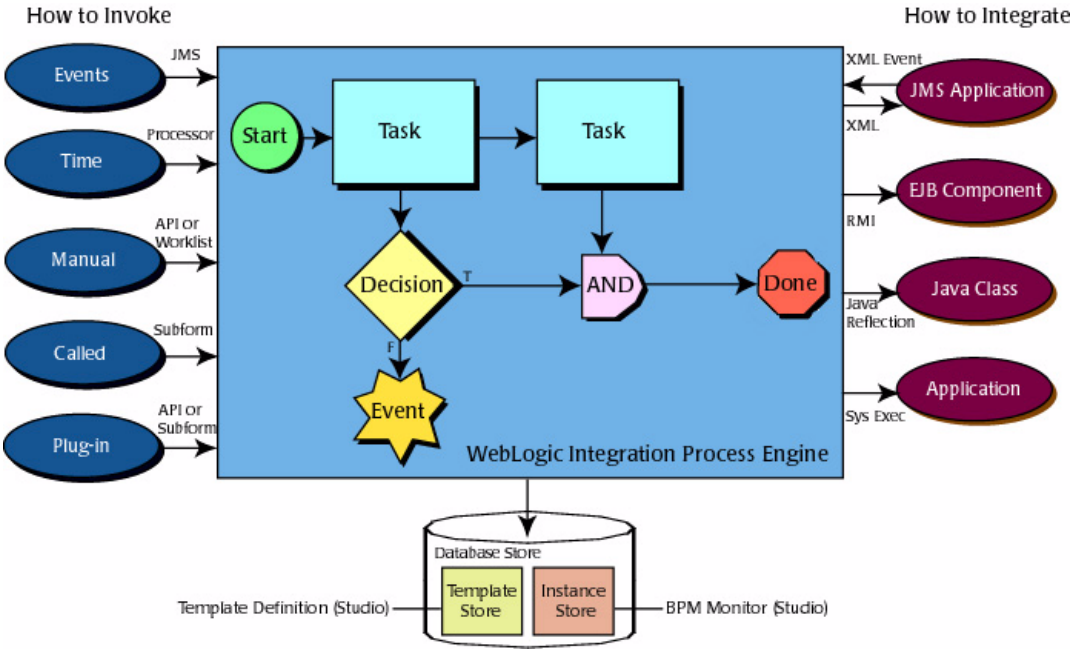
Note: For an overview of the business process model, and details about using the Studio client interface to administer, design, and monitor business processes, see *Using the WebLogic Integration Studio*.

For details about using the Worklist client interface to execute business processes, see *Using the WebLogic Integration Worklist*.

WebLogic Integration Process Engine

The following figure illustrates the BPM processing model.

Figure 1-1 BPM Processing Model



Note: For a complete description of the BPM processing model, see *Using the WebLogic Integration Studio*.

Note that the BPM processing model depends on the WebLogic Integration *process engine*, which serves as the controller for run-time instances, managing their execution and monitoring their progress.

In the previous figure, the left column lists the methods through which you can start (or *instantiate*) a business process or interact with an existing business process: in response to an XML event, as a time-based event, through a manual call, through a call from another business process, or through a plug-in interface.

The right column lists the entities that can be integrated into a business process: JMS application, EJB component, Java class, or another application.

The database store, shown at the bottom of the figure, stores templates, template definitions, and run-time instances. You can create templates and template definitions, and monitor run-time instances using the Studio or a custom design client.

WebLogic Server Infrastructure

The WebLogic Integration process engine runs on BEA WebLogic Server and takes advantage of the Java 2 Platform, Enterprise Edition (J2EE) services listed in the following table. For more information about WebLogic Server services, see *Introduction to BEA WebLogic Server*, in the BEA WebLogic Server document set, available at the following URL:

http://e-docs.bea.com/wls/docs70/intro/index.html

Table 1-1 WebLogic Server Services Used by BPM

Service	Description
Enterprise JavaBean (EJB)	Provides lifecycle management and services such as caching, persistence, and transaction management. The process engine consists of a combination of session and entity EJBs. For more information, see "Process Engine Component Architecture" on page 1-5. In addition, message-driven beans are used for providing asynchronous messaging services.
Security Realm	Supports security interfaces including principals, groups, ACLs, and permissions. The process engine builds on the WebLogic Server security realm. Roles defined in the Studio map to WebLogic Server security groups, and users, to WebLogic Server users.
Web server	Supports Web browsers and other custom design and run-time management clients that use HTTP, servlets, and JSPs.
Java Naming and Directory Service (JNDI)	Supports directory service functionality.
Java Database Connectivity (JDBC)	Supports Java database connectivity and persistence. The process engine uses JDBC for persisting data.

Table 1-1 WebLogic Server Services Used by BPM

Service	Description
Messaging	Implements the Java Message Service (JMS), including the transmission of XML content. The process engine uses JMS for communicating worklist, time, and event notifications, and error and audit messages. For more information, see "Establishing JMS Connections" on page 6-1.
Clustering	Supports scalability and high-availability by grouping multiple servers together in a way that is transparent to application developers and end users. The process engine takes advantage of clustering. To ensure data integrity and idempotence, you specify a unique transaction ID, as described in the context of the applicable methods later in this document.
Time Services	Builds on the services provided by WebLogic Server to support timed events and other timer-based activities.

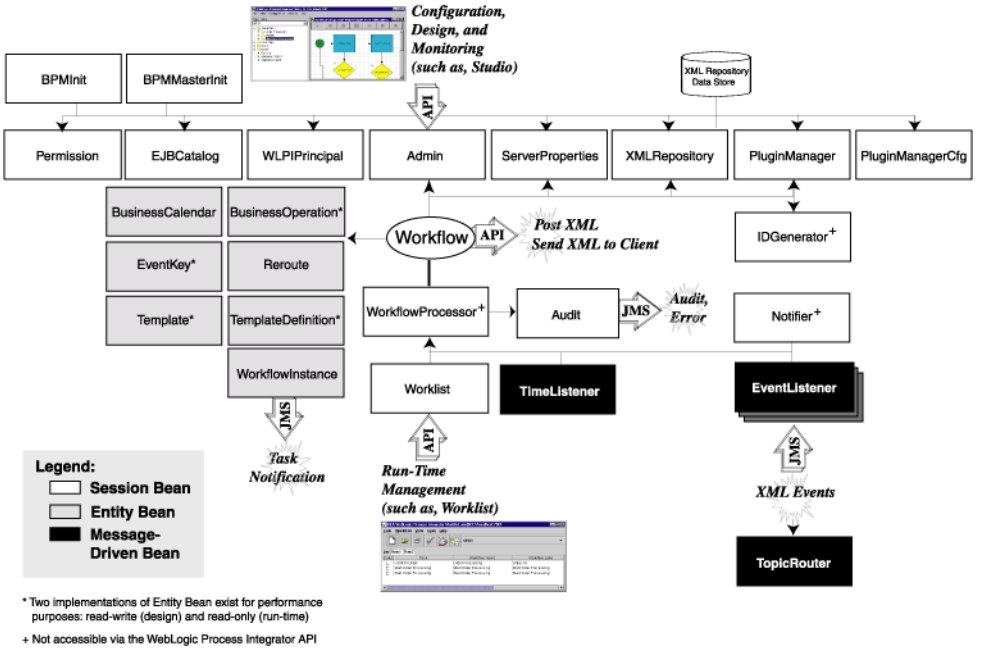
Process Engine Component Architecture

The WebLogic Integration process engine consists of the following components:

- Session EJBs
- Entity EJBs
- Message-driven beans

The architecture of the process engine, including client and JMS interactions, is illustrated in the following figure.

Figure 1-2 Process Engine Component Architecture



Note the following in the previous figure:

- The process engine manages the design, administration, and execution of a business process.
- The `Admin` and `Worklist` session EJBs are the primary interfaces for design and run-time management clients, respectively.
- The `WorkflowProcessor` session EJB provides the main interface to the process engine. (It is not part of the BPM API.)
- The `Audit` session EJB manages the audit, trace, and error messages.
- The `EventListener` message-driven bean manages XML events.

- The `TopicRouter` message-driven bean is supported for backward compatibility with pre-2.0 releases of the BEA WebLogic Process Integrator product: it reroutes all messages sent to the `wlpiEvent` JMS *topic*, which is not supported by the process engine (as of BEA WebLogic Process Integrator Release 2.0), to the `WLI_BPM_Event` message-driven bean.
- The `TimeListener` message-driven bean manage timed events.
- A number of Entity EJBs (`BusinessOperation`, `EventKey`, `Template`, and `TemplateDefinition`) maintain two implementations for performance purposes: a read-write implementation used during configuration and design, and a read-only implementation used during run time.

The following sections describe the process engine components in more detail.

Session EJBs

Session EJBs provide application services to clients upon request. The BPM session EJBs provide the primary interface to the process engine.

The following table summarizes the functions of the BPM session EJBs, which are accessible to client applications via the BPM API (except where noted). It also indicates whether each EJB is *stateful* or *stateless*.

Table 1-2 Session EJBs

Session EJB	Description	EJB Type
`com.bea.wlpi.server.admin.Admin`	Provides the primary interface for design clients, including Studio and custom design clients.	Stateful
`com.bea.wlpi.server.audit.Audit`	Sends audit and error output to WebLogic JMS.	Stateless
`com.bea.wlpi.server.catalog.EJBCatalog`	Catalogs EJBs deployed on WebLogic Server.	Stateful
`com.bea.wlpi.server.idgenerator.IDGenerator`[1]	Generates unique IDs for various WLPI system object classes. (For internal use only.)	Stateless

Table 1-2 Session EJBs (Continued)

Session EJB	Description	EJB Type
com.bea.wlpi.server.init.BPMInit	Initializes plug-in manager and license checking. In a clustered environment, this EJB is deployed across the cluster.	Stateless
com.bea.wlpi.server.init.BPMMasterInit	Sets up and starts message reapers (time-triggered clean-up services). In a clustered environment, this EJB runs on a single system.	Stateless
com.bea.wlpi.server.notify.Notifier[1]	Supports JMS queue and publishing services. (For internal use only.)	Stateless
com.bea.wlpi.server.permission.Permission	Supports getting and setting user and role permissions.	Stateless
com.bea.wlpi.server.plugin.PluginManager	Provides run-time management of plug-ins during workflow execution. For more information about programming BPM plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.	Stateless
com.bea.wlpi.server.plugin.PluginManagerCfg	Provides configuration- and design-time support and management of plug-ins. For more information about programming BPM plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.	Stateless
com.bea.wlpi.server.serverproperties.ServerProperties	Supplies the BPM server properties.	Stateless
com.bea.wlpi.server.principal.WLPIPrincipal	Links the process engine to WebLogic Server security realms.	Stateless
com.bea.wlpi.server.workflowprocessor.WorkflowProcessor[1]	Provides the main interface to the process engine.	Stateful
com.bea.wlpi.server.worklist.Worklist	Provides the primary interface for run-time management clients, including Worklist and custom run-time management clients.	Stateful
com.bea.eci.repository.ejb.XMLRepository	Provides access to the XML repository database.	Stateless

1. Not available to the BPM API.

Stateful session EJBs are available to a client for the duration of a session, until the client releases the resource.

Stateless session EJBs, on the other hand, are a shared commodity. That is, a particular EJB can be accessed by multiple clients; there is no guarantee that the same EJB will be used by the process engine to handle multiple method calls to the EJB. Stateless session EJBs facilitate clustering, as subsequent method calls can be handled by different instances of an EJB on one or more servers.

For more information about the session EJBs available via the BPM API, see "BPM API" on page 1-12.

Entity EJBs

Entity EJBs represent data objects and include state information.

The BPM entity EJBs are not accessible directly through the BPM API; however, the `com.bea.wlpi.common` package provides classes for obtaining state information. For more information about the `com.bea.wlpi.common` package, see "Client/Server Common Package" on page 1-17.

The following table describes the BPM entity EJBs.

Table 1-3 Entity EJBs

Entity EJB	Description
`com.bea.wlpi.server.businesscalendar.BusinessCalendar`	Defines business-oriented time calculations.
`com.bea.wlpi.server.businessoperation.BusinessOperation[RO,RW]`	Defines user-defined EJB or Java class actions. Two implementations of this EJB are maintained for performances purposes: a read-write implementation used during configuration and design, and a read-only implementation used during run time.

Table 1-3 Entity EJBs (Continued)

Entity EJB	Description
`com.bea.wlpi.server.eventkey.EventKey[RO,RW]`	Defines event key expressions for filtering events. Two implementations of this EJB are maintained for performances purposes: a read-write implementation used during configuration and design, and a read-only implementation used during run time.
`com.bea.wlpi.server.reroute.Reroute`	Defines task rerouting for users at predetermined time intervals.
`com.bea.wlpi.server.template.Template[RO,RW]`	Provides design-time representation of a workflow template. Used at run time to create a running instance. Two implementations of this EJB are maintained for performances purposes: a read-write implementation used during configuration and design, and a read-only implementation used during run time.
`com.bea.wlpi.server.template.TemplateDefinition[RO,RW]`	Provides design-time representation of a workflow template definition. Used at run time to create a running instance. Two implementations of this EJB are maintained for performances purposes: a read-write implementation used during configuration and design, and a read-only implementation used during run time.
`com.bea.wlpi.server.instance.WorkflowInstance`	Provides run-time representation of a workflow instance.

Message-Driven Beans

The following table describes the BPM message-driven beans.

Table 1-4 Message-Driven Beans

Message-Driven Bean	Description
`com.bea.wlpi.server.eventlistener.EventListener`	Manages event messages.
`com.bea.wlpi.server.timelistener.TimeListener`	Manages timed events.
`com.bea.wlpi.server.topicrouter.TopicRouter`	Supported for backward compatibility with the process engine provided in the BEA WebLogic Process Integrator product before Release 2.0. Reroutes all messages sent to the `wlpiEvent` JMS *topic* (for which support was dropped in WebLogic Process Integrator Release 2.0) to the `EventListener` message-driven bean, which handles messages for the newly-supported `WLI_BPM_Event` queue. For more information about the JMS queues and topics, see "Establishing JMS Connections" on page 6-1. **Note:** Although you are not required to do so, you may update your code to use the `WLI_BPM_Event` JMS queue (`com.bea.wli.bpm.Event`) in place of the `wlpiEvent` JMS topic to expedite the transmission of messages.

BPM API

The business process management (BPM) API consists of nine EJBs (summarized in the table "Session EJBs" on page 1-7) and the following seven packages:

- `com.bea.wlpi.client.common`
- `com.bea.wlpi.client.util`
- `com.bea.wlpi.common`
- `com.bea.wlpi.common.plugin`
- `com.bea.wlpi.common.security`
- `com.bea.wlpi.util`
- `com.bea.eci.repository.helper`

The `IDGenerator`, `Notifier`, and `WorkflowProcessor` EJBs are not included in the BPM API.

The API components are described in detail in the following sections. For additional information, see the *BEA WebLogic Integration Javadoc*. For information about interfacing with the API session EJBs, see "Connecting to the Process Engine" on page 3-1.

Admin Session EJB

The `com.bea.wlpi.server.admin.Admin` session EJB is a *stateful* EJB that serves as the primary interface for design clients, including the Studio and custom design clients.

Using the public methods provided by the `Admin` EJB, you can:

- Define templates and template definitions, and associated business operations
- Create business calendars and event keys, and reroute tasks
- Control the stages in the lifecycle of a task (assign, mark done, and so on)
- Get class descriptors, tasks, variables, instances, and statistics reports

- Import and export template definitions and complete workflow packages

Audit Session EJB

The `com.bea.wlpi.server.audit.Audit` session EJB is a *stateless* EJB that encapsulates a WebLogic JMS topic (`wlpiAudit`) to which the process engine sends audit and error messages on behalf of clients. The `Audit` EJB also encapsulates access to the `wlpiError` topic, to which error messages are sent. Users define audit entries via task actions when designing workflow template definitions, as described in *Using the WebLogic Integration Studio*.

Each audit message consists of an XML document that conforms to the format defined in the `Audit` DTD. For more information about the `Audit` DTD format, see "Audit DTD" on page A-2 or the *BEA WebLogic Integration Javadoc*.

Audit information (`Info`) and error (`Error`) messages are sent to the WebLogic Server log file, using the subsystem name `WLPI`.

For more information about setting up a JMS connection to the `wlipAudit` and `wlpiError` JMS topics, see "Establishing JMS Connections" on page 6-1.

EJBCatalog Session EJB

The `com.bea.wlpi.server.catalog.EJBCatalog` session EJB is a *stateful* EJB that classifies the EJBs deployed on a particular server. The EJBCatalog session EJB scans the JNDI tree, collects the EJB metadata, and stores it in a catalog. The `EJBCatalog` is used by design clients to define business operations.

Using the public methods provided by the `EJBCatalog` EJB, you can:

- List the JNDI names of all EJBs deployed on the WebLogic Server
- List the EJB metadata describing the internals of all EJBs installed on the WebLogic Server
- Set the root context in the WebLogic JNDI tree in which cataloging is to begin
- Specify whether the catalog should be regenerated with each listing request

Permission Session EJB

The `com.bea.wlpi.server.permission.Permission` session EJB is a *stateless* EJB that enables you to get and set security permissions pertaining to both role and user actions.

PluginManager Session EJB

The `com.bea.wlpi.server.plugin.PluginManager` session EJB is a *stateless* EJB that provides run-time management of plug-ins during workflow execution.

For more information about programming plug-ins and the `PluginManager` EJB, see *Programming BPM Plug-Ins for WebLogic Integration*.

PluginManagerCfg Session EJB

The `com.bea.wlpi.server.plugin.PluginManagerCfg` session EJB is a *stateless* EJB that enables you to manage the implementation of user-defined plug-ins.

For more information about programming plug-ins and the `PluginManagerCfg` EJB, see *Programming BPM Plug-Ins for WebLogic Integration*.

ServerProperties Session EJB

The `com.bea.wlpi.server.serverproperties.ServerProperties` session EJB is a *stateless* EJB that enables clients to obtain information about the properties governing BPM.

Using the public methods provided by the `ServerProperties` EJB, you can get the following information:

- Software version, including the major and minor version numbers, build number, and product name
- Software properties

- Software version required to support the features defined in a template
- Package version supported by the server

WLPIPrincipal Session EJB

The `com.bea.wlpi.server.principal.WLPIPrincipal` session EJB is a *stateless* EJB that enables clients to interface with the WebLogic Server security realm and access BPM organizations, roles, users, and other security properties. BPM roles map to WebLogic Server security groups; BPM users, to WebLogic Server users.

Using the public methods provided by the `WLPIPrincipal` EJB, you can configure, manage, and obtain information about the organizations, roles, and users that ensure secure access to the system.

Note: Information about roles and users beyond the WebLogic Server-specific information that is stored in the WebLogic Server security realm, is maintained in the BPM database.

Worklist Session EJB

The `com.bea.wlpi.server.worklist.Worklist` session EJB is a *stateful* EJB serving as the primary interface for run-time management clients, including Worklist and custom run-time management clients. The `Worklist` EJB enables interaction with running instances.

Using the public methods provided by the `Worklist` EJB, you can:

- Get startable business processes
- Instantiate business processes
- Get tasks and task counts
- Control the stages in the lifecycle of a task being executed (assign, mark done, and so on)
- Respond to notifications
- Set task properties

- Invoke exception-handling methods

XMLRepository Session EJB

The `com.bea.eci.repository.ejb.XMLRepository` session EJB is a *stateless* EJB that provides access to the XML repository database.

Using the public methods provided by the `XMLRepository` EJB, you can:

- Manage XML repository folders
- Manage XML repository entities
- Get EJB environment variable values

Client Common Package

The `com.bea.wlpi.client.common` package provides common client-side classes used by a BPM client. This package includes classes for:

- Connecting to the process engine
- Facilitating client GUI layout and development, and generating dialog boxes
- Getting exception, task, and versioning information
- Managing resources
- Caching images and icons
- Filtering files by file extension
- Generating a unique transaction ID for use in a clustered environment

Client Utility Package

The `com.bea.wlpi.client.util` package provides two JMS utilities, `JMSTest` and `JMSTestAddr`, for testing the publish and subscribe features of a JMS topic.

Client/Server Common Package

The `com.bea.wlpi.common` package provides classes used by both the BPM client and the WebLogic Integration process engine. The package includes:

- *Value objects* for describing the following:
  - Organizations, users, and roles
  - Templates and template definitions
  - Tasks and task reroutings
  - Variables
  - Running instances and functions
  - Business calendars
  - Java classes
  - Event keys
  - Versioning information
- Descriptors for Java classes, EJBs, EJB invocations, and server-side Java class methods
- Mechanism for serializing *long strings* (strings that exceed 64KB when they are UTF-8 encoded) in pre-JDK 1.3 run-time environments
- Message interface for language-independent server messaging

Package members can be serialized to facilitate the exchange of information between the client and the server.

Plug-In Common Package

The `com.bea.wlpi.common.plugin` package provides common classes used to manage user-defined plug-ins.

For more information, see *Programming BPM Plug-Ins for WebLogic Integration*.

Security Common Package

The `com.bea.wlpi.common.security` package provides common classes used to define security permissions. The package includes:

- *Value objects* for describing role and user permissions
- Security permission types

Utility Package

The `com.bea.wlpi.util` package provides general BPM utilities, including a utility for generating message-driven beans.

XML Repository Helper Package

The `com.bea.eci.repository.helper` package provides common classes used to access the XML repository. The package includes:

- *Value objects* for describing the XML repository folder and entities
- Methods for creating and managing folders and subfolders
- Methods for creating and managing entities

BPM Application Development Tasks

The BPM API enables users to accomplish the application development tasks described in the following sections.

Configuration

The BPM API, via the Studio client or a custom configuration client, enables users to configure:

- Security realms (organizations, roles, and users)
- Business operations
- Event keys
- Business calendars

Part II of this document, "Configuration," describes how to configure each of these types of entities using the BPM API. For more information about the API, see the *BEA WebLogic Integration Javadoc*.

Design

The BPM API, via the Studio client or a custom design client, enables users to perform the following types of design tasks:

- Create and manage workflow templates and template definitions
- Create and manage tasks and task reroutings
- Manage the XML repository database
- Import and export design data

Part III of this document, "Design," describes how to perform each of these types of design-related tasks using the BPM API. For more information about the API, see the *BEA WebLogic Integration Javadoc*.

Run-Time Management

The BPM API, via the Worklist or a custom run-time management client, enables users to perform the following types of run-time management tasks:

- Manage the active organization
- Start business process instances
- Manage run-time tasks, including:
  - Get, assign, and execute tasks
  - Respond to client requests
  - Mark tasks as done or undone
  - Set task properties
  - Update variables
  - Invoke exception handlers

Part IV of this document, "Run-Time Management," describes how to perform each of these types of run-time management tasks using the BPM API. For more information about the API, see the *BEA WebLogic Integration Javadoc*.

Monitoring

The BPM API, via the Studio or a custom monitoring client, enables users to perform the following types of monitoring tasks:

- Manage run-time instances
- Manage run-time variables
- Generate graphical reports based on workflow, tasks, user or role, and task status
- Generate statistical reports based on workflow, task, user or role, and date

Part V of this document, "Monitoring," describes how to perform each of these types of monitoring tasks using the BPM API. For more information about the API, see the *BEA WebLogic Integration Javadoc*.

Plug-In Development

The BPM API enables users to design and integrate custom plug-ins.

For more information, see *Programming BPM Plug-Ins for WebLogic Integration*.

BPM API Examples

The examples used throughout this document are excerpted from the BPM client examples provided with the software in the *SAMPLES_HOME*/integration/samples/bpm_api directory. The client examples include:

- Command-Line Administration Example
- Command-Line Studio Example
- Command-Line Worklist Example
- Command-Line SAX Parser Example
- JSP Worklist Example

The client examples are described in detail in the following sections.

Command-Line Administration Example

WebLogic Integration provides an example that demonstrates the basic administration tasks that you might support in a custom command-line client. These tasks include:

- Connecting to the process engine
- Managing organization, roles, and users: adding, deleting, and so on
- Managing the security realm: getting the name of a security realm class and determining whether or not the security realm is manageable

- Managing business operations: adding, deleting, and so on, and getting EJB and Java class descriptors
- Managing event keys: adding, deleting, and so on
- Managing business calendars: adding, deleting, and so on
- Managing deployed EJBs: listing names and descriptors of deployed EJBs, and enabling and/or disabling automatic regeneration
- Displaying server properties, including server version, template definition version supported, and system properties

The command-line administration example consists of one Java file, `CLAdmin.java`, located in the *`SAMPLES_HOME`*`/integration/samples/bpm_api/commandline` directory. For information about how to compile and run this example, see the `Readme.txt` file in this directory.

Some of the examples presented in Part I, "API Development Fundamentals," and Part II, "Configuration," are excerpted from the command-line administration example.

Command-Line Studio Example

WebLogic Integration provides an example that demonstrates basic design tasks that you might support in a custom command-line studio client, including:

- Managing templates: creating, deleting, and listing
- Managing task reroutings: adding, deleting, and listing

The command-line studio example consists of one Java file, `CLStudio.java`, located in the *`SAMPLES_HOME`*`/integration/samples/bpm_api/commandline` directory. For information about how to compile and run this example, see the `Readme.txt` file in this directory.

Some of the examples presented in Part III, "Design," are excerpted from the command-line studio example.

Command-Line Worklist Example

WebLogic Integration provides an example that demonstrates the basic worklist management tasks that you might support in a custom command-line worklist client. These tasks include:

- Connecting to the process engine
- Managing organizations: listing all organizations, and listing and setting the current active organization
- Managing business processes: listing all startable business processes and starting a business process
- Managing tasks: listing and executing tasks, and getting task counts

The command-line worklist example consists of one Java file, `CLWorklist.java`, located in the *`SAMPLES_HOME`*`/integration/samples/bpm_api/commandline` directory. For information about how to compile and run this example, see the `Readme.txt` file in this directory.

Some of the examples presented in Part IV, "Run-Time Management," are excerpted from the command-line worklist example.

Command-Line SAX Parser Example

WebLogic Integration provides an example that demonstrates how to use a SAX parser to parse client requests received from the BPM server. Specifically, this example illustrates how to:

- Use the Xerces SAX parser to parse client requests from the server
- Create and send a response to the server

The command-line SAX parser example consists of one Java file, `CLSaxParser.java`, located in the *`SAMPLES_HOME`*`/integration/samples/bpm_api/commandline` directory. For information about how to compile and run this example, see the `Readme.txt` file in this directory.

Some of the examples presented in Part IV, "Run-Time Management," are excerpted from the command-line worklist example.

JSP Worklist Example

WebLogic Integration provides an example that demonstrates the basic run-time management tasks that you might support in a custom Java Server Page (JSP)-based worklist client. The main component of this example is the `worklist.jsp` file, which provides the main interfaces to the JSP worklist.

The following table lists the tasks that are demonstrated by the JSP client, and identifies example files to view for more information.

Table 1-5 JSP Worklist Example

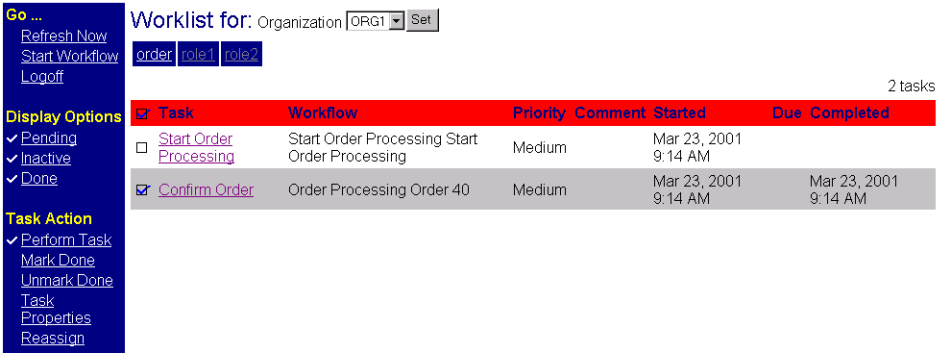
Task	Related Example Files
Logging on and connecting to the process engine, and specifying a login, password, and URL. For more information about connecting to the process engine, see "Connecting to the Process Engine" on page 3-1.	`logon.html, logon.jsp`
Starting a workflow from the list of available workflow templates.	`startworkflow.jsp`
Getting tasks.	`worklist.jsp`
Executing tasks.	`worklist.jsp`
Parsing the response XML (`responseParser`). The `responseParser` implements a document handler with the SAX (Simple API for XML) parser, enabling the application to handle specific events as they occur within the XML document. For more information about parsing XML, see the `XML.com` Web site supported by O'Reilly & Associates, Inc. at the following URL: `http://www.xml.com`	`ResponseParser.java`
Responding when input is required to set variables (`query.jsp`) or acknowledge a message (`message.jsp`).	`query.jsp, message.jsp`

Table 1-5 JSP Worklist Example (Continued)

Task	Related Example Files
Reassigning a task to a different user or role.	`reassign.jsp`
Marking a task as done or undone.	`worklist.jsp`
Setting task properties to control the run-time characteristics of a running instance.	`taskproperties.jsp`
Logging out and disconnecting from the process engine. For more information about disconnecting, see "Disconnecting from the Process Engine" on page 8-1.	`logoff.jsp`

The following figure shows the main JSP interface, `worklist.jsp`.

Figure 1-3 Main Interface to the JSP Worklist



For information about deploying the JSP worklist as a web application, see "Deploying Web Applications" in *Assembling and Configuring Web Applications* within the BEA WebLogic Server documentation set, at the following URL:

`http://e-docs.bea.com/wls/docs70/webapp/deployment.html`

Once deployed, to run the JSP worklist:

1. Append the following path to your `CLASSPATH`:
   `WLI_HOME/lib/wlpi-worklist.jar`.

For example, one way to accomplish this is by adding the following information within the `StartWebLogic` file, above the `REM Start weblogic` comment:

Windows: `set SVRCP=%WLI_HOME%\lib\wlpi-worklist.jar;%SVRCP%`
UNIX: `setenv SVRCP $WLI_HOME/lib/wlpi-worklist.jar;$SVRCP`

2. Start WebLogic Integration.
3. Use a browser to call the JSP page, with a URL that follows the pattern:

   `http://`*`WebLogicURL:WebLogicPort/example_name`*

   For example, to load the worklist example in a browser running on the same NT host as your WebLogic Server, on port 7001:

   `http://localhost:7001/worklist`

4. Log in and start using the sample.

Some of the examples presented in Part IV, "Run-Time Management," are excerpted from the JSP worklist example.

Part I API Development Fundamentals

2 Importing Packages and Interfaces

This section describes the following packages and interfaces that are available for importing into your applications:

- BPM Packages and Interfaces
- General Java Packages

BPM Packages and Interfaces

The following table lists the business process management (BPM) API packages and interfaces that are used by design and run-time client applications, and suggests when you might consider importing them.

Table 2-1 BPM Packages and Interfaces

This package or interface . . .	Consists of . . .
`com.bea.wlpi.server.admin.Admin`	Interface supporting BPM administrative functions. Import when configuring, designing, and monitoring business processes.
`com.bea.wlpi.server.catalog.EJBCatalog`	Interface supporting a catalog of EJBs installed on the application server. Import when configuring business operations.

Table 2-1 BPM Packages and Interfaces (Continued)

This package or interface . . .	Consists of . . .
com.bea.wlpi.server.permission.Permission	Interface supporting security configuration through setting permissions for roles and users. Import when configuring permissions for roles and users.
com.bea.wlpi.server.plugin.PluginManager	Interface supporting run-time management of plug-ins during workflow execution. For more information about programming plug-ins and the `PluginManager` EJB, see *Programming BPM Plug-Ins for WebLogic Integration*.
com.bea.wlpi.server.plugin.PluginManagerCfg	Interface enabling you to manage the implementation of user-defined plug-ins. For more information about programming plug-ins and the `PluginManager` EJB, see *Programming BPM Plug-Ins for WebLogic Integration*.
com.bea.wlpi.server.serverproperties.ServerProperties	Interface that allows you to access BPM server information, and inspect Java system properties remotely. **Note:** In a clustered environment, the remote inspection of Java system properties is of limited use, as the method request is directed to a server, based on the configured routing algorithm. The information returned by the method call varies, particularly if the systems in the cluster have unique specifications. Import, as necessary, to determine the compatibility of templates, template definitions, and so on.
com.bea.wlpi.server.principal.WLPIPrincipal	Interface supporting BPM security functions. Import when configuring organizations, roles, and users.
com.bea.wlpi.server.worklist.Worklist	Interface supporting BPM run-time management functions. Import when executing business processes.

Table 2-1 BPM Packages and Interfaces (Continued)

This package or interface . . .	Consists of . . .
com.bea.eci.repository.ejb.XMLRepository	Interface supporting the XML repository database. Import when XML repository database access is required.
com.bea.wlpi.client.common.*	Package containing common client-side classes, including various convenience methods. Import, as necessary.
com.bea.wlpi.client.util.*	Package containing JMS test utilities. Import, as necessary, for testing JMS.
com.bea.wlpi.common.*	Package containing classes used by both the BPM client and BPM server, including value objects. Import as necessary.
com.bea.wlpi.common.plugin.*	Package containing classes used for managing user-defined plug-ins. Import when managing user-defined plug-ins. For more information about programming plug-ins and the PluginManager EJB, see *Programming BPM Plug-Ins for WebLogic Integration*.
com.bea.wlpi.common.security.*	Package containing common classes used when defining security permissions. Import when defining security permissions.
com.bea.wlpi.util.*	Package containing general BPM utilities, including a utility for generating message-driven beans. Import when generating message-driven beans.
com.bea.eci.repository.helper.*	Package containing common classes used for accessing the XML repository. Import when using the XML repository.

General Java Packages

The following table lists the general Java packages that are used by design and run-time client applications, and suggests when you might consider importing them.

Table 2-2 General Java Packages

This package . . .	Supports . . .
`java.io.*`	System input and output. Import when performing file I/O operations.
`java.lang.*`	Fundamental design classes. Import as necessary.
`java.net.*`	Network application implementation. Import, as necessary, when performing various network-specific functions.
`java.sql.*`	Data accessing and processing API using the Java programming language. Import, as necessary, when performing database-specific functions.
`java.text.*`	Text that is independent of natural languages. Import as necessary.
`java.util.*`	Standard APIs, such as date and time utilities. Import, as necessary.
`javax.ejb.*`	Enterprise Java Beans (EJBs). Import, as necessary, when performing various EJB operations.
`javax.jms.*`	Java Message System (JMS). Import when connecting to JMS.
`javax.naming.*`	JNDI interfaces required for server and destination lookups. Import when performing JNDI lookups.

Table 2-2 General Java Packages (Continued)

This package . . .	Supports . . .
`javax.rmi.*`	Remote Method Invocation (RMI). Import as necessary to support, for example, exceptions related to remote method calls, and other RMI-related interactions.
`javax.xml.parsers.*`	Java API for XML parsing (JAXP). The JAXP API does not replace either the SAX or DOM API. Instead, it adds some convenience methods that are designed to make the SAX and DOM APIs easier to use. Import, as necessary, when parsing XML.
`org.xml.sax.*`	SAX (Simple API for XML) interfaces. Import, as necessary, when parsing XML.
`weblogic.apache.xerces.parsers.*`	Apache Xerces XML parser. Import, as necessary, when parsing XML.

Depending on the user interface requirements, you may also consider importing the packages listed in the following table.

To support . . .	Import . . .
Java applets	`java.applet.*`
Java user interface	`java.awt.*`
Java Swing GUI applications	`javax.swing.*`

3 Connecting to the Process Engine

This section explains how a to connect to the WebLogic Integration process engine and access the features of the business process management (BPM) framework. It includes the following topics:

- Accessing the API Session EJBs
- Using the Convenience Methods to Access EJBs

Accessing the API Session EJBs

As with any EJB, you must access the BPM API session EJBs using the home and remote interfaces. The following table lists the home and remote interfaces available for this purpose.

Table 3-1 API Session EJB Home and Remote Interfaces

EJB Name	Home Interface	Remote Interface
com.bea.wlpi.server.admin.Admin	AdminHome	Admin
com.bea.wlpi.server.audit.Audit	AuditHome	Audit
com.bea.wlpi.server.catalog.EJBCatalog	EJBCatalogHome	EJBCatalog
com.bea.wlpi.server.permission.Permission	PermissionHome	Permission

Table 3-1 API Session EJB Home and Remote Interfaces (Continued)

EJB Name	Home Interface	Remote Interface
com.bea.wlpi.server.plugin.PluginManager	PluginManagerHome	PluginManager
com.bea.wlpi.server.plugin.PluginManagerCfg	PluginManagerCfgHome	PluginManagerCfg
com.bea.wlpi.server.serverproperties.ServerProperties	ServerPropertiesHome	ServerProperties
com.bea.wlpi.server.principal.WLPIPrincipal	WLPIPrincipalHome	WLPIPrincipal
com.bea.wlpi.server.worklist.Worklist	WorklistHome	Worklist
com.bea.eci.repository.ejb.XMLRepository	XMLRespositoryHome	XMLRepository

To access BPM API session EJBs and their methods, you must perform the following steps:

1. Look up a session EJB home interface in JNDI.
2. Create a remote session object (`EJBObject`) using the home interface.

The following sections describe these steps in detail.

Step 1: Look Up a Session EJB Home Interface in JNDI

The BPM session EJBs are exposed to client applications as part of the WebLogic Server JNDI namespace.

You can look up a session EJB home interface by first establishing a JNDI context (`java.naming.Context`). The most common way of doing this is by instantiating a `javax.naming.InitialContext` object. For client applications that require a specific security context for authorization to access EJBs and their methods, you must also pass security credentials (user name and password, for example) to the `InitialContext()` constructor.

For example, the following method, excerpted from the JSP worklist example, creates an initial context, passing the specified URL, user ID, and password as the context environment.

```
public Context getInitialContext(
  String user,
  String password,
  String url
) throws NamingException
{
     // Get an InitialContext
     Properties h = new Properties();
     h.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextFactory");
     h.put(Context.PROVIDER_URL, url);
     if (user != null) {
         h.put(Context.SECURITY_PRINCIPAL, user);
         if (password == null)
           password = "";
     }
         h.put(Context.SECURITY_CREDENTIALS, password);

     return new InitialContext(h);
}
```

For more information, see the `javax.naming.InitialContext()` Javadoc. For more information about the JSP worklist, see "JSP Worklist Example" on page 1-24.

Once the JNDI context is defined, you can use the JNDI context `lookup()` method to access the session EJB home interface.

For example, to look up the `Worklist` session EJB home interface, and store the object reference to the `worklistHome` variable, execute the following statements:

```
Context context = getInitialContext(url, userId, password);
Object result = context.lookup("com.bea.wlpi.Worklist");
WorklistHome worklistHome =(WorklistHome)
  PortableRemoteObject.narrow(result, WorklistHome.class);
```

Similarly, to look up the `WLPIPrincipal` session EJB home interface and store the object reference to the `principalHome` variable, execute the following statements:

```
Context context = getInitialContext(url, userId, password);
object result = context.lookup("com.bea.wlpi.WLPIPrincipal");
PrincipalHome prinicipalHome=(PrincipalHome)
  PortableRemoteObject.narrow(result,
  WLPIPrinicipalHome.class);
```

Note: The `PortableRemoteObject.narrow()` method used in the previous examples ensures that the remote object is cast to the desired type. Use of this method is required if the system is configured to use RMI-IIOP; otherwise, its usage is optional (but recommended). For more information, see the `javax.rmi.PortableRemoteObject` class Javadoc.

Step 2: Create a Remote Session Object Using the Home Interface

Once you have a reference to a home interface, you can use the home interface object `create()` method to access a remote session object (`EJBObject`).

For example, to create an `EJBObject` for the `Worklist` session EJB and store the object reference to the `worklist` variable, execute the following statement:

```
Worklist worklist = worklistHome.create();
```

Similarly, to create an `EJBObject` for the `WLPIPrincipal` session EJB and store the object reference to the `principal` variable, execute the following statement:

```
WLPIPrincipal wlpiprincipal = principalHome.create();
```

Using the Convenience Methods to Access EJBs

The `com.bea.wlpi.client.common.WLPI` class provides a set of convenience methods for accessing session EJBs. The following table summarizes these methods.

Table 3-2 Convenience Methods for Accessing EJBs

Method	Description
`public com.bea.wlpi.server.admin.Admin getAdmin( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	Returns `Admin` EJB object.
`public com.bea.wlpi.server.catalog.EJBCatalog getCatalog( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	Returns `EJBCatalog` EJB object.
`public com.bea.wlpi.server.permission.Permission getPermission( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	Returns `Permission` EJB object.
`public com.bea.wlpi.server.plugin.PluginManager getPluginManager( ) throws java.lang.IllegalStateException`	Returns `PluginManager` EJB object.
`public com.bea.wlpi.server.serverproperties.ServerProperties getServerProperties( ) throws IllegalStateException, WorkflowException`	Returns `ServerProperties` EJB object.
`public com.bea.wlpi.server.principal.WLPIPrincipal getPrincipal( ) throws java.lang.IllegalStateException`	Returns `WLPIPrincipal` EJB object.

Table 3-2 Convenience Methods for Accessing EJBs (Continued)

Method	Description
`public com.bea.wlpi.server.worklist.Worklist getWorklist( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	Returns `Worklist` EJB object.
`public com.bea.eci.repository.ejb.XMLRepository getRepository( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	Returns `XMLRepository` EJB object.

Table 3-2 Convenience Methods for Accessing EJBs (Continued)

Method	Description
`public boolean connect( java.awt.Frame owner java.lang.String url, java.lang.String userId, java.lang.String password ) throws java.lang.IllegalStateException` `public boolean connect( java.awt.Frame owner java.lang.String url, java.lang.String userId, java.lang.String password, int maxRetries, boolean autoLogon, boolean graphical ) throws java.lang.IllegalStateException`	Connects to the process engine by performing the following tasks: ■ Accesses the `WLPIPrincipal` session EJB. ■ Sets the user ID and URL within the environment. ■ Performs three automatic retries if a login fails. You must specify the following arguments: ■ *`owner`* - window in which message boxes and dialog boxes will be displayed ■ *`url`* - BPM URL ■ *`userId`* - user ID ■ *`password`* - password ■ *`maxRetries`* - maximum number of retries permitted ■ *`autoLogon`* - flag specifying whether to attempt to logon using supplied values (`true`) or to prompt user for confirmation (`false`). ■ *`graphical`* - flag specifying whether to perform a graphical dialog with the user (`true`) or not (`false`). You can obtain the user ID, password, and server URL by prompting the user for this information. For your convenience, the `client.common.logon` class is provided for generating a login dialog and capturing the user input. For more information about this class, see the `com.bea.wlpi.client.common.logon` Javadoc.
`public boolean isConnected()`	Checks whether the `WLPI` instance is connected; that is, whether a reference has been made to the `WLPIPrincipal` session EJB.

Table 3-2 Convenience Methods for Accessing EJBs (Continued)

Method	Description
`public javax.naming.Context getInitialContext()`	Returns the host application JNDI context.

For more information, see the `com.bea.wlpi.client.common.WLPI` Javadoc.

CHAPTER

4 Accessing Process Engine Information

This section describes how to access information about the WebLogic Integration process engine. It includes the following topics:

- Getting the Server Version
- Getting the Package Version
- Getting the Template Definition Version
- Getting Server Properties
- Using the Convenience Methods
- Example of Accessing Information About the Process Engine

For more information about the methods described in this section, see the `com.bea.wlpi.server.serverproperties.ServerProperties` or `com.bea.wlpi.client.common.WLPI` Javadoc.

Getting the Server Version

To get the server version, use the following `com.bea.wlpi.server.serverproperties.ServerProperties` method:

```
public com.bea.wlpi.common.VersionInfo getServerVersion(
) throws java.rmi.RemoteException
```

This method returns the server version as a `VersionInfo` object. To access information about the version, use the `VersionInfo` object methods described in "VersionInfo Object" on page B-30.

For example, the following code gets the version and saves it to the `version` object. In this example, `properties` represents the `EJBObject` reference to the `ServerProperties` EJB.

```
VersionInfo version = properties.getServerVersion();
```

For more information about the `getServerVersion()` method, see the `com.bea.wlpi.server.serverproperties.ServerProperties` Javadoc.

Getting the Package Version

To get the package version that is supported by the server, use the following `com.bea.wlpi.server.serverproperties.ServerProperties` method:

```
public com.bea.wlpi.common.VersionInfo getPackageVersion(
) throws java.rmi.RemoteException
```

This method returns the package version as a `VersionInfo` object. To access information about the version, use the `VersionInfo` object methods described in "VersionInfo Object" on page B-30.

For example, the following code gets the package version and saves it to the `version` object. In this example, `properties` represents the `EJBObject` reference to the `ServerProperties` EJB.

```
VersionInfo version = properties.getPackageVersion();
```

For more information about the `getPackageVersion()` method, see the `com.bea.wlpi.server.serverproperties.ServerProperties` Javadoc.

Getting the Template Definition Version

To get the template definition version supported by the server, use the following `com.bea.wlpi.server.serverproperties.ServerProperties` method:

```
public com.bea.wlpi.common.VersionInfo getTemplateDefinitionVersion(
) throws java.rmi.RemoteException
```

This method returns the template definition version supported by the server as a `VersionInfo` object. To access information about the version, you can use the `VersionInfo` object methods described in "VersionInfo Object" on page B-30.

For example, the following code gets the template definition version supported by the server and saves it to the `version` object. In this example, `properties` represents the `EJBObject` reference to the `ServerProperties` EJB.

```
VersionInfo version = properties.getTemplateDefinitionVersion();
```

For more information about the `getTemplateDefinitionVersion()` method, see the `com.bea.wlpi.server.serverproperties.ServerProperties` Javadoc.

Getting Server Properties

To get server properties, use the following `com.bea.wlpi.server.serverproperties.ServerProperties` method:

```
public java.util.Properties getProperties(
) throws java.rmi.RemoteException
```

This method returns the server properties, including the Java system properties and any other defined properties.

Note: In a clustered environment, the remote inspection of Java system properties is of limited use, as the method request is directed to a server, based on the configured routing algorithm. The information returned by the method call varies, particularly if the systems in the cluster have unique specifications.

For example, the following code gets the server properties and saves them to the `props` object. In this example, `properties` represents the `EJBObject` reference to the `ServerProperties` EJB.

```
Properties props = properties.getProperties();
```

For more information about the `getProperties()` method, see the `com.bea.wlpi.server.serverproperties.ServerProperties` Javadoc.

Using the Convenience Methods

The `com.bea.wlpi.client.common.WLPI` class provides a set of convenience methods for accessing server information. The following table summarizes these methods.

Table 4-1 Convenience Methods for Accessing Server Information

Method	Description
`public com.bea.wlpi.common.VersionInfo getServerVersion( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	Accesses the `ServerProperties` session EJB and returns the BPM server version. This method returns a `com.bea.wlpi.common.VersionInfo` object corresponding to the template. To access information about the version, use the `VersionInfo` object methods described in "VersionInfo Object" on page B-30.
`public com.bea.wlpi.common.VersionInfo getServerTemplateDefinitionVersion( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	Accesses the `ServerProperties` session EJB and returns the template definition version supported by the current version of WebLogic Integration. This method returns a `com.bea.wlpi.common.VersionInfo` object corresponding to the template. To access information about the version, use the `VersionInfo` object methods described in "VersionInfo Object" on page B-30.

For more information, see the `com.bea.wlpi.client.common.WLPI` Javadoc.

Example of Accessing Information About the Process Engine

This section provides an excerpt from the command-line administration client example showing how to access server properties.

An input stream is defined to communicate with the user, and the user is prompted to specify what action to perform. If the user selects the `Server Properties` option, the system gets the server and template definition versions, and uses the returned `VersionInfo` objects to obtain other information about the server and template definition, including the name, and major, minor, and build versions. In addition, the server property names and values are displayed.

Important lines of code are highlighted in **bold**. In this example, the string `serverProperties` represents the `EJBObject` reference to the `ServerProperties` EJB.

```
/* Display Tool Title */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---             Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .

/**
 * Method that illustrates the Public API Methods available in the
```

```
 * ServerProperties interface (no user interaction, display mode only).
 */
public static void mngServerProperties( ) {
   Properties serverSystemProperties;
   String answer;

   try {
      /* Prompt user to select if we need to display the System properties */
      if( ( answer = askQuestion( "\nList system properties (y/n)?" ) )
         == null ) {
         /* User cancelled the operation */
         System.out.println( "*** Cancelled" );
         return;
      }

      /* Parse the answer */
      boolean isListSysProperties = ( answer.equals( "y" ) ||
         answer.equals( "Y" ) );
      /* Display all the server properties and attributes */
      System.out.println( "\nServer Version:" );

      /* WLPI Public API Method */
      /* Retrieve server attributes */
      serverVersion = serverProperties.getServerVersion( );

      /* Display all available attributes in WLPI v1.2.1 */
      System.out.println( "- Release Name: " + serverVersion.getName( ) );
      System.out.println( "- Version: " + serverVersion +
         " (Major=" + serverVersion.getMajorVersion( ) +
         " Minor=" + serverVersion.getMinorVersion( ) +
         " Build=" + serverVersion.getBuild( ) + ")" );

      /* Display all the template definition properties and attributes */
      System.out.println( "\nTemplate Definition version supported:" );

      /* WLPI Public API Method */
      /* Retrieve template definition attributes */
      serverDTDVersion = serverProperties.getTemplateDefinitionVersion( );

      /* Display all available attributes */
      System.out.println( "- Release Name: " + serverDTDVersion.getName( ) );
      System.out.println( "- Version: " + serverDTDVersion +
         " (Major=" + serverDTDVersion.getMajorVersion( ) +
         " Minor=" + serverDTDVersion.getMinorVersion( ) +
         " Build=" + serverDTDVersion.getBuild( ) + ")" );

      if( isListSysProperties ) {
         System.out.println( "\nSystem Properties: " );
```

```
      /* WLPI Public API Method */
      /* Retrieve server system properties */
      serverSystemProperties = serverProperties.getProperties( );

      /* Loop to display all properties */
      for( Enumeration e = serverSystemProperties.propertyNames( );
         e.hasMoreElements( ); ) {
         String propertyName = e.nextElement( ).toString( );
         System.out.println( "- Name: '" + propertyName + "' Value: '" +
         serverSystemProperties.getProperty( propertyName ) + "'\n" );
      }
   }
}
   catch( Exception e ) {
      System.out.println( "*** Failed to retrieve properties" );
      System.err.println( e );
   }
   return;
}
```

CHAPTER

5 Using Value Objects

This section explains how to access object data using business process management (BPM) value objects. It includes the following topics:

- Introduction to Value Objects
- Creating Value Objects
- Using Value Objects to Access Object Data
- Sorting Value Objects
- Example of Using a Value Object

Introduction to Value Objects

Three BPM packages—`com.bea.wlpi.common`, `com.bea.wlpi.common.security`, and `com.bea.eci.repository.helper`—provide classes, or *value objects*, for obtaining object data at both definition and run time. For more information about each of these packages, see "BPM API" on page 1-12.

Each value object shares the following characteristics:

- Maintains various BPM server-side objects, including session EJBs (for example, templates, template definitions, and business calendars) and entity EJBs that are used internally. Also allows you to obtain data values from these objects.
- Is represented by an individual Java class, the members of which are collectively referred to as *values*

- Is serializable: can be exchanged between client and server
- Overrides the `equals()` method for testing two objects of the same type for equality, as follows:

  ```
  public boolean equals(Object obj)
  ```

- Implements the `java.lang.comparable` interface for comparing two objects of the same type, as follows:

  ```
  public int compareTo(Object obj)
  ```

- When part of a homogeneous list, a class can be searched and sorted using the following methods:
  - `java.util.Collection.contains(Object o)`
  - `java.util.List.indexOf(Object o)`
  - `java.util.Collections.sort(List list)`
  - `java.util.Collections.sort(List list, Comparator c)`

  For more information about these classes, see the Sun Microsystems, Inc. Java Web site, available at the following URL:

  `http://java.sun.com`

- If the natural ordering of an object (as implemented by the `boolean compareTo(Object o)` method) is based on the same field used by the `boolean equals(Object o)` method, the following method

  ```
  int java.util.Collections.binarySearch(List list, Object o)
  ```

  can be used for rapidly searching a list that was sorted earlier, using the `java.util.Collections.sort(List list)` method.

- Implement the `com.bea.wlpi.common.Publishable` interface, if the import and export of the object data is supported.

  Note: If the importing and exporting of data is supported, the object also implements the `com.bea.wlpi.common.Publishable` interface. For more information, see "Publishing Workflow Objects" on page 18-1.

The following table lists the value objects that can be used to access object data.

Table 5-1 Value Objects

Use This Value Object . . .	To access . . .
`com.bea.wlpi.common.BusinessCalendarInfo`	Business calendar data
`com.bea.wlpi.common.EventKeyInfo`	Event key data
`com.bea.wlpi.common.InstanceInfo`	Workflow instance data
`com.bea.wlpi.common.OrganizationInfo`	Organization data
`com.bea.wlpi.common.security.PermissionInfo`	Permission data
`com.bea.wlpi.common.RepositoryFolderHelperInfo`	XML repository folder data
`com.bea.eci.repository.helper.RepositoryFolderInfo`	XML repository folder data
`com.bea.wlpi.common.RerouteInfo`	Task rerouting data
`com.bea.wlpi.common.RoleInfo`	Role data
`com.bea.wlpi.common.RolePermissionInfo`	Role permission data
`com.bea.wlpi.common.TaskInfo`	Workflow task data
`com.bea.wlpi.common.TemplateDefinitionInfo`	Template definition data
`com.bea.wlpi.common.TemplateInfo`	Template data
`com.bea.wlpi.common.UserInfo`	User data
`com.bea.wlpi.common.security.UserPermissionInfo`	User permission data
`com.bea.wlpi.common.VariableInfo`	Variable data
`com.bea.wlpi.common.VersionInfo`	Version number data
`com.bea.wlpi.common.XMLEntityHelperInfo`	XML repository entity data
`com.bea.eci.repository.helper.XMLEntityInfo`	XML repository entity data

Creating Value Objects

To create a value object, use the associated constructor. Each of the BPM value objects described in the table "Value Objects" on page 5-3, provides one or more constructors for creating object data. The constructors for creating value objects are described in "Value Object Summary" on page B-1.

For example, the following code creates an `OrganizationInfo` object, sets the organization ID to `ORG1`, and assigns the resulting object to `organization`.

```
OrganizationInfo organization = new OrganizationInfo("ORG1");
```

Using Value Objects to Access Object Data

Each BPM value object described in the table "Value Objects" on page 5-3 provides various methods for accessing object data. The methods for getting and setting object data for each value object are described in "Value Object Summary" on page B-1.

For example, the following code gets the organization ID for the specified `OrganizationInfo` object, `organization`.

```
String id = getOrgId(organization);
```

Sorting Value Objects

As described in "Introduction to Value Objects" on page 5-1, you can sort a list of value objects that implement the `java.lang.Comparable` interface using `sort()` methods available to the `java.util.Collections` class, as follows:

- `java.util.Collections.sort(java.util.List` *`list`*`)`
- `java.util.Collections.sort(java.util.List` *`list`*`,`
  `java.util.Comparator` *`c`*`)`

The first method enables you to sort a specified list of elements into ascending order based on the *natural ordering* of the elements, as described for `java.lang.Comparable`:

http://java.sun.com/j2se/1.3/docs/api/java/lang/Comparable.html

The second method enables you to sort a specified list of elements using a custom comparator that sorts on keys other than the default.

Note: The `com.bea.wlpi.client.common.SortTableModel` class provides a set of methods for sorting the value object data within a column of a JTable by clicking the column header. For more information, see the `com.bea.wlpi.client.common.SortableTableModel` Javadoc.

Example of Using a Value Object

This section provides an excerpt from the command-line worklist example showing how to use a value object to access task object data. It also demonstrates the use of the `TaskInfo` class methods to get a task name, template definition ID, instance ID, and task ID.

The get actions are highlighted in **bold**.

```
/* Any task assigned ? */
if( taskList.size( ) == 0 )
   System.out.println( "\nNo task assigned" );
else
   System.out.print( "\nAssigned Tasks:" );

/* Process the list to display the tasks */
for( int i = 0; i < taskList.size( ); i++ ) {
   /* Retrieve an element from the list */
   TaskInfo taskInfo = ( TaskInfo )taskList.get( i );

   /* WLPI Public API Methods */
   /* Retrieve and display a sub-set of available attributes */
   System.out.println( "\n- Name: " + taskInfo.getName( ) );
   System.out.println( "  Template Definition ID: " +
      taskInfo.getTemplateDefinitionId( ) );
   System.out.println( "  Workflow Instance ID: " +
      taskInfo.getInstanceId( ) );
   System.out.println( "  Task ID: " + taskInfo.getTaskId( ) );
```

```
      System.out.print( "  Status: " + taskInfo.getStatus( ) );

      /* Retrieve and display the task status */
      if( taskInfo.getStatus( ) == taskInfo.STATUS_PENDING )
         System.out.println( " (Pending)" );
      else if( taskInfo.getStatus( ) == taskInfo.STATUS_COMPLETE )
         System.out.println( " (Complete)" );
      else if( taskInfo.getStatus( ) == taskInfo.STATUS_OVERDUE )
         System.out.println( " (Overdue)" );
      else if( taskInfo.getStatus( ) == taskInfo.STATUS_INACTIVE )
         System.out.println( " (Inactive)" );
      else
         System.out.println( " (Unknown)" );
   }
   break;|
   .
   .
   .
```

6 Establishing JMS Connections

This section explains how to establish Java Message Service (JMS) connections. It includes the following topics:

- Overview of JMS
- JMS Destinations Used by the Process Engine
- Connecting to JMS
- Receiving Messages Asynchronously
- Generating Message-Driven Beans for Multiple Event Queues
- Guaranteeing Message Delivery
- Guaranteeing Sequential Processing of Messages
- Example of Connecting to a JMS Topic

Note: For more information about JMS, see *Programming WebLogic JMS,* in the BEA WebLogic Server document set, at the following URL:

`http://e-docs.bea.com/wls/docs70/jms/index.html`

Or the *JavaSoft JMS specification version 1.0.2*, from Sun Microsystems, Inc., available at the following URL:

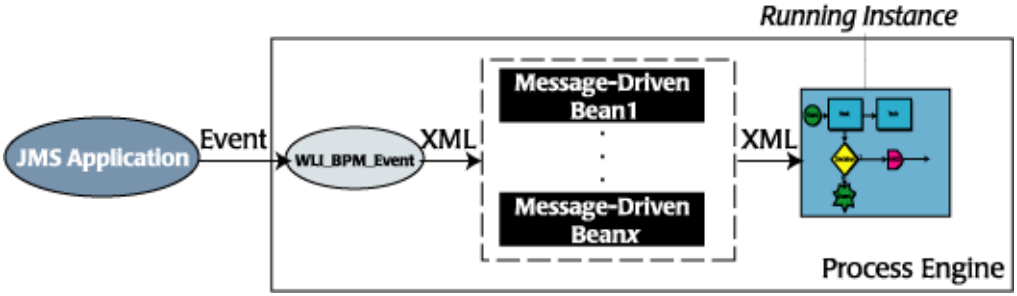
`http://www.javasoft.com/products/jms/docs.html`

Overview of JMS

WebLogic Server, on which the WebLogic Integration process engine and business process management (BPM) framework runs, implements the JMS, which supports the transmission of XML content. The process engine uses WebLogic JMS for communicating worklist, time, and event notifications; and error and audit messages.

The following figure shows how JMS enables communication, through XML messages, between running BPM workflow instances and external client applications.

Figure 6-1 Overview of JMS



As shown here, an XML event issued by a JMS application is:

1. Delivered to a predefined event queue (for example, `WLI_BPM_Event`, described in the next section).
2. Directed to the appropriate message-driven bean for processing.
3. Delivered to the running instance.

The following sections describe the JMS destinations that are used by the process engine, and explain how to connect to and use them. An example showing how to connect to a JMS topic, `WLI_BPM_Notify`, is also provided.

For more information about JMS, see *Programming WebLogic JMS,* in the BEA WebLogic Server document set, at the following URL:

`http://e-docs.bea.com/wls/docs70/jms/index.html`

JMS Destinations Used by the Process Engine

The following table summarizes the JMS destinations (queues and topics) used by the process engine.

Table 6-1 JMS Destinations Used by the Process Engine

This JMS destination . . .	Of JMS Type . . .	Is the destination for . . .
Name: `WLI_BPM_Audit` JNDI Name: `com.bea.wli.bpm.Audit`	Topic	Audit messages.
Name: `WLI_BPM_Error` JNDI Name: `com.bea.wli.bpm.Error`	Topic	Error messages. For more information, see “Monitoring Workflow Exceptions” on page 24-1.

Table 6-1 JMS Destinations Used by the Process Engine (Continued)

This JMS destination . . .	Of JMS Type . . .	Is the destination for . . .
Name: `WLI_BPM_Event` JNDI Name: `com.bea.wli.bpm.Event`	Queue	Events. All incoming messages are processed via message-driven beans. A message-driven bean is an EJB that acts as a message consumer in the WebLogic JMS messaging system. As with standard JMS message consumers, message-driven beans receive messages from a JMS queue or topic, and perform business logic based on the message contents. In addition, message-driven beans support concurrent processing for JMS destinations. A system administrator can define multiple event queues, if required, as described in "Connecting to JMS" on page 6-6. In this case, the system administrator must also generate the associated message-driven beans using the utility described in "Generating Message-Driven Beans for Multiple Event Queues" on page 6-8, and the application developer must specify the appropriate destination for a particular event. You can guarantee the delivery and sequential processing of messages using the methods described in "Guaranteeing Message Delivery" on page 6-10 and "Guaranteeing Sequential Processing of Messages" on page 6-13, respectively. **Note:** For backward compatibility, the `TopicRouter` bean automatically reroutes any messages bound for the `wlpiEvent` *topic*, which is supported by WebLogic Process Integrator Release 1.2.1 and earlier releases, to the `WLI_BPM_Event` *queue*.

Table 6-1 JMS Destinations Used by the Process Engine (Continued)

This JMS destination . . .	Of JMS Type . . .	Is the destination for . . .
Name: `WLI_BPM_Notify` JNDI Name: `com.bea.wli.bpm.Notify`	Topic	Worklist notifications. Enables a worklist client to refresh its display dynamically and obtain task object data via the delivered `com.bea.wlpi.common.TaskInfo` object. For messages posted to the `WLI_BPM_Notify` topic, the following message properties must be defined to ensure delivery to appropriate subscribers: ■ `orgId`: string specifying the ID of the organization ■ `assigneeId`: string specifying the ID of the *assignee* (user or role) ■ `role`: Boolean flag specifying whether or not the assignee is a role (`true`, `false`) ■ `action`: string specifying which of the following actions has been performed on the task: - `assigned`: task assigned to user - `update`: task state has changed - `remove`: task removed from list of user tasks A worklist client can define a selector to filter messages based on these message properties. For an example of using a selector to filter messages, see "Example of Connecting to a JMS Topic" on page 6-16. For more information about JMS, see *Programming WebLogic JMS* in the BEA WebLogic Server document set, available at the following URL: `http://e-docs.bea.com/wls/docs70/jms/index.html`
Name: `WLI_BPM_Timer` JNDI Name: `com.bea.wli.bpm.Timer`	Queue	Time processor triggers (internal use only).

Name:
`WLI_BPM_ValidatingEvent`
JNDI Name:
`com.bea.wli.bpm.ValidatingEvent`
	Queue	Events that require DTD or schema validation.
		If XML messages reference a DTD or schema, validates the content.

Table 6-1 JMS Destinations Used by the Process Engine (Continued)

This JMS destination . . .	Of JMS Type . . .	Is the destination for . . .
Name: `WLI_FailedEvent` JNDI Name: `com.bea.wli.FailedEvent`	Queue	Messages that have been unsuccessfully delivered and the maximum number of retries has been consumed. The maximum number of retries is configurable for each JMS destination using the `Redelivery-Limit` attribute. For more information, see the description of the JMSTemplate element within the *BEA WebLogic Server Configuration Reference* at the following URL: `http://e-docs.bea.com/wls/docs70/config_xml/index.html` When a failed message is received, an entry is logged to the log file. If desired, you can develop a custom event handler to handle failed messages.

Connecting to JMS

To connect to any of the JMS destinations defined in the table "JMS Destinations Used by the Process Engine" on page 6-3 (and to be able to post and receive XML messages), the WebLogic Server administrator must perform the following steps for each destination:

1. Look up a JMS connection factory in JNDI.

   A system administrator defines and configures one or more connection factories, and WebLogic Server adds them to the JNDI space during startup. A connection factory encapsulates connection configuration information, and enables JMS applications to create a connection.

2. Create a connection using the connection factory.

   A connection represents an open communication channel between an application and the messaging system. It is used to create a session for producing and consuming messages.

3. Create a session using the connection.

A session defines a serial order in which messages are produced and consumed, and can create multiple message producers and message consumers.

4. Look up destinations in JNDI.

   A destination can be either a queue or a topic, encapsulating the address syntax for a specific provider. An administrator defines and configures the destinations and WebLogic Server adds them to the JNDI space during startup.

   On the client side, destinations are handles to the objects on the server. The methods return only the destination names. To access destinations for messaging, you create message producers and consumers that can attach to destinations.

5. Create message producers and message consumers using session and destinations.

   A message producer sends messages to a queue or topic. A message consumer receives messages from a queue or topic.

6. Perform one of the following steps:

   a. If you are creating message producers, create the message object.

      A message encapsulates the information exchanged by applications.

   b. If you are creating message consumers, you may want to register an asynchronous message listener, as described in "Receiving Messages Asynchronously" on page 6-7.

7. Start the connection.

For more information about JMS, see *Programming WebLogic JMS* in the BEA WebLogic Server document set, at the following URL:

`http://e-docs.bea.com/wls/docs70/jms/index.html`

Receiving Messages Asynchronously

To receive messages asynchronously from a destination, you must register an asynchronous message listener by performing the following steps:

1. Implement the `javax.jms.MessageListener` interface, which includes an `onMessage()` method.

   Note: This step can also be accomplished using message-driven beans for WebLogic Server 6.0 or greater. For more information about EJBs, see *Programming WebLogic Enterprise JavaBeans* in the BEA WebLogic Server document set, at the following URL:

   `http://e-docs.bea.com/wls/docs70/ejb/index.html`

2. Set the message listener using the following `javax.jms.MessageConsumer` method and passing the listener information as an argument:

```
public void setMessageListener(
  javax.jms.MessageListener listener
) throws javax.jms.JMSException
```

For more information about JMS, see *Programming WebLogic JMS* in the BEA WebLogic Server document set, available at the following URL:

`http://e-docs.bea.com/wls/docs70/jms/index.html`

Generating Message-Driven Beans for Multiple Event Queues

As noted previously, the system administrator can define multiple event queues, if required, as described in “Connecting to JMS” on page 6-6. In this case, the system administrator must also generate the associated message-driven beans.

To generate message-driven beans for multiple event queues, use the `com.bea.wlpi.util.MDBGenerator` utility, as follows:

```
java com.bea.wlpi.util.MDBGenerator -queue queue_name
   [-min number] [-max number] [-order number] [-transact]
   [-validate] [-timeout seconds] [-help]
```

The following table lists the `MDBGenerator` utility arguments.

Table 6-2 MDBGenerator Utility Arguments

Argument	Description
`-queue` *`queue_name`*	Name of the custom queue for which you want to generate the message-driven bean. This argument is required.
`-min` *`minimum`*	Minimum number of message listeners allocated to handle *unordered* messages. Unordered messages are those that do not need to be processed in a specific order, as described in "Guaranteeing Sequential Processing of Messages" on page 6-13. *`minimum`* specifies an integer value that must be less than the maximum value. This argument is optional and defaults to 0.
`-max` *`maximum`*	Maximum number of message listeners allocated to handle *unordered* messages. *`maximum`* specifies an integer value that must be less than or equal to 100, and greater than the minimum value specified. This argument is optional and defaults to 5.
`-order` *`ordernum`*	Number of message listeners allocated to handle *ordered* messages. Ordered messages are those that require messages to be processed in a specific order, as described in "Guaranteeing Sequential Processing of Messages" on page 6-13. *`ordernum`* specifies a prime number less than or equal to 31. This argument is optional and defaults to 0.
`-transact`	Flag specifying that the transaction is required.
`-validate`	Flag specifying that you want to validate all messages received on the queue against the `DOC-TYPE` tag in the XML JMS message. This argument is optional.
`-timeout` *`seconds`*	Transaction timeout in seconds. This argument defaults to 30 seconds. This value is only used if the `transact` flag is not set. If the `transact` flag is set, the utility uses the WebLogic Server transaction timeout value, which can be set using the Administration Console and defaults to 30.

Table 6-2 MDBGenerator Utility Arguments (Continued)

Argument	Description
`-help`	Flag specifying that you want to display command usage syntax. This argument is optional.

This utility generates a message-driven bean deployment for a specific queue as a jar file named *qname*`-mdb.jar`, where *qname* specifies the associated queue name.

To deploy the message-driven beans on WebLogic Server, edit the `ejb-jar.xml` and `weblogic-ejb.xml` files to associate the EJB with a configured JMS destination. For more information, see *Programming WebLogic Enterprise JavaBeans* in the BEA WebLogic Server document set, available at the following URL:

`http://e-docs.bea.com/wls/docs70/ejb/index.html`

Guaranteeing Message Delivery

Messages can be specified as persistent or non-persistent. A persistent message is guaranteed to be delivered at least once—it is not considered sent until it has been safely written in the file or database. WebLogic JMS writes persistent messages to a persistent backing store (file or JDBC database) assigned to each JMS server during configuration. Non-persistent messages are not stored. They are guaranteed to be delivered at least once unless there is a system failure, in which case messages may be lost. If a connection is closed or recovered, all non-persistent messages that have not yet been acknowledged will be redelivered. Once a non-persistent message is acknowledged, it will not be redelivered.

In the event that one or more receipients are not available when a message is sent, you can guarantee message delivery using *addressed* messages. Addressed messages persist an incoming event message until it is consumed by all recipients or a specified expiration time (*time-to-live*) elapses, whichever occurs first. You can guarantee message delivery on a workflow instance or template basis.

To guarantee message delivery using addressed messaging, the sending application (message producer) must define the following information:

1. The `WLPIInstanceIDs` and/or `WLPITemplateNames` fields as part of the message header, which defines the message consumer *addresses*.

   Specifically, these fields specify the ID of the instance and/or name of the template, respectively, that you want to receive the message. You can define the `WLPIInstanceIDs` and `WLPITemplateNames` JMS header fields using the following `javax.jms.Message` class method:

   ```
   public void setStringProperty(
     java.lang.String name,
     java.lang.String value
   ) throws javax.jms.JMSException
   ```

   For example, to define the JMS header fields for the `msg` message instance, use the following methods:

   ```
   String instanceID;
   //instanceID set somewhere
   msg.setStringProperty("WLPIInstanceIDs", instanceID);

   String templateName="MyTemplate";
   msg.setStringProperty("WLPITemplateNames", templateName);
   ```

   These fields must be specified as String values. You can specify multiple IDs for each field, separated by commas. You should target only those instances and/or templates that have already been instantiated.

   Note: If you want to address messages to a template, avoid using commas in the template name.

   JMS header fields are always transmitted with the message, and are available to the message consumers (including the message-driven beans). For more information about defining JMS message header fields, see "Setting and Browsing Message Header and Property Fields" in "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*, in the BEA WebLogic Server documentation set, at the following URL:

   `http://e-docs.bea.com/wls/docs70/jms/implement.html`

2. The time-to-live value as a parameter when sending the message that specifies the number of seconds to persist an addressed message. The processing engine persists the message until the time-to-live value expires or the message has been consumed by all addressed recipients, whichever occurs first. This parameter must be specified as an integer value. For more information about sending messages, see "Sending Messages" in "Developing a WebLogic JMS Application" in Programming WebLogic JMS, in the BEA WebLogic Server documentation set, at the following URL:

http://e-docs.bea.com/wls/docs70/jms/implement.html

The time-to-live value can also be set via the Studio Post XML Event dialog box, as described in "Defining Actions" in *Using the WebLogic Integration Studio*.

Once set, you can view the resulting expiration time via the `JMSExpiration` JMS header field using the following `javax.jms.Message` class method:

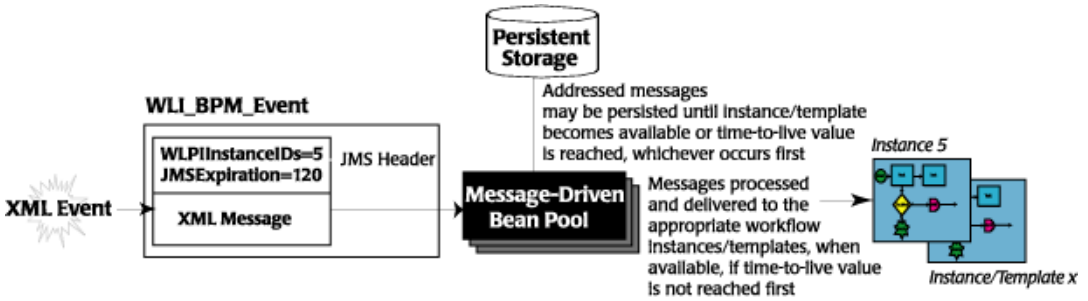
```
public long getJMSExpiration(
) throws javax.jms.JMSException
```

For more information about viewing JMS message header fields, see "Setting and Browsing Message Header and Property Fields" in "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*, in the BEA WebLogic Server documentation set, at the following URL:

http://e-docs.bea.com/wls/docs70/jms/implement.html

The following figure illustrates how to guarantee message delivery using addressed messaging.

Figure 6-2 Guaranteed Message Delivery



Note the following in this figure:

- Addressed messages are delivered to the instance or template specified by the JMS header field in the sending application (`WLPIInstanceIDs` or `WLPITemplateNames`).
- Addressed messages that cannot be delivered because a recipient is unavailable are persisted until the recipient becomes available or the time-to-live value (`JMSExpiration` header field value) is reached, whichever occurs first.

Guaranteeing Sequential Processing of Messages

One advantage of message-driven beans is that incoming messages can be processed in parallel by random message-driven bean instances. In this case, however, you cannot guarantee the *order* in which messages are processed.

If the order in which messages are received and processed is significant, you can guarantee that order by directing all messages to the same message-driven bean instance.

To guarantee that messages are delivered to the same message-driven bean instance and processed in a sequential order, the sending and receiving applications must preform the following steps:

1. Before sending the message, the sending application (message producer) must define an order key (`WLPIOrderKey`) as a field in the JMS message header. The order key value must be a long integer, such as the instance ID.

   You can define the `WLPIOrderKey` JMS header field using the following `javax.jms.Message` class method:

   ```
   public void setLongProperty(
     java.lang.String name,
     long value
   ) throws javax.jms.JMSException
   ```

   For example, to define the `WLPIOrderKey` JMS header field for the `msg` message instance, use the following method:

   ```
   msg.setLongProperty("WLPIOrderKey",
      Long.parseLong(instanceID));
   ```

   Related messages that require sequential processing should be assigned the same order key value.

   JMS header fields are always transmitted with the message, and are available to the message consumers (including the message-driven beans). For more information about defining JMS message header fields, see “Setting and Browsing Message Header and Property Fields” in “Developing a WebLogic JMS Application” in *Programming WebLogic JMS*, in the BEA WebLogic Server documentation set, at the following URL:

http://e-docs.bea.com/wls/docs70/jms/implement.html

2. The receiving application (message consumer) must define a JMS message selector to *filter* the messages that it needs to process, based on the `WLPIOrderKey` value.

   A message selector is a Boolean expression. It consists of a String with a syntax similar to the `where` clause of an `SQL select` statement.

   For example: `WLPIOrderKey=1`

   For more information about defining JMS message selectors, see "Filtering Messages" in "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*, in the BEA WebLogic Server documentation set, at the following URL:

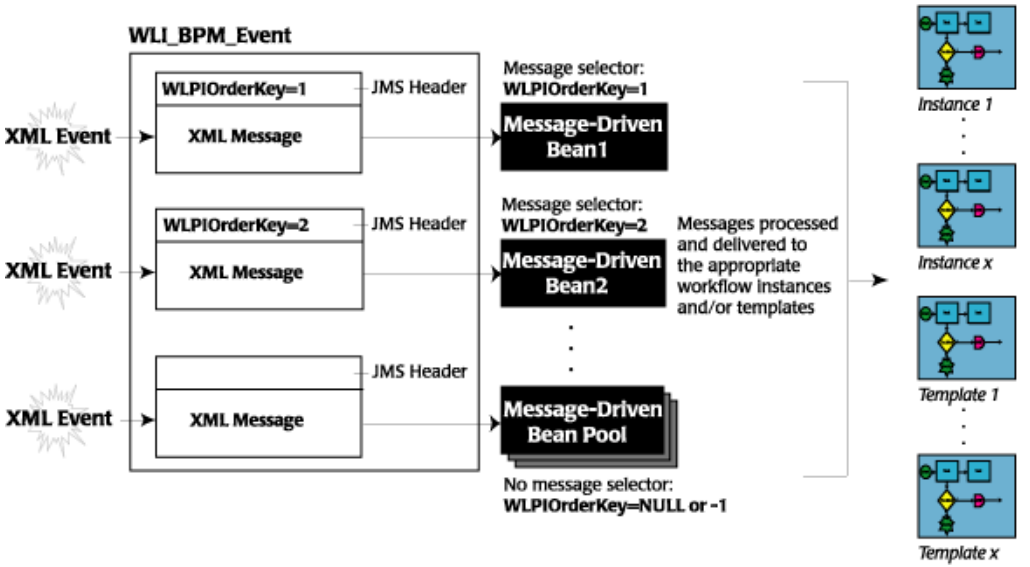
   http://e-docs.bea.com/wls/docs70/jms/implement.html

Note: All message groups that are to be processed in sequential order must be sent to the same JMS event queue.

All messages with a specific order key are processed by the same message-driven bean instance, in the order received, guaranteeing the sequential processing of the messages.

The following figure illustrates the sequential processing of messages using order keys.

Figure 6-3 Guaranteed Sequential Processing of Messages



Note the following in this figure:

- All messages that are to be processed in sequential order are sent to the same JMS event queue, `WLI_BPM_Event`, or other user-defined queue.
- All messages with the `WLPIOrderKey` header field set to `1` are processed by `Message-Driven Bean 1`; messages with the `WLPIOrderKey` header field set to `2` are processed by `Message-Driven Bean 2`; and so on.
- All messages in which the `WLPIOrderKey` header field information is not defined are processed by a pool of message-driven beans.
- Each XML message may affect one or more workflow templates and/or instances.

Example of Connecting to a JMS Topic

This section provides an example showing how to connect to a JMS destination, in this case, the worklist notification topic, `WLPI_BPM_Notify`. The `WLPI_BPM_Notify` topic is used by a worklist client to refresh its display, dynamically, upon receipt of a message. In this example a message selector (filter) is used to receive notification. Subsequently, if the properties of the received message match the current values of the organization, role, user, and action, the message selector is used to refresh the display.

Each section of code is described in detail. For more information about JMS, see *Programming WebLogic JMS* in the BEA WebLogic Server document set, available at the following URL:

http://e-docs.bea.com/wls/docs70/jms/index.html

Define the required variables, including the JNDI context, JMS connection factory, and topic static variables:

```
protected static final String JNDI_FACTORY= "weblogic.jndi.WLInitialContextFactory";
protected static final String JMS_FACTORY= "javax.jms.TopicConnectionFactory";
protected static final String NOTIFY_TOPIC="com.bea.wli.bpm.Notify";
```

Create all the objects necessary for sending messages to a JMS topic:

```
private TopicConnectionFactory tconFactory;
private TopicConnection tcon;
private TopicSession tsession;
private Topic topic;
private TopicSubscriber tsubscriber;
```

Set up the JNDI initial context, as follows:

```
try {
   Hashtable env = new Hashtable();
   env.put(Context.PROVIDER_URL, wlpi.getUrl());
   env.put(Context.INITIAL_CONTEXT_FACTORY, JNDI_FACTORY);
   env.put("weblogic.jndi.createIntermediateContexts", "true");

   Context ctx = new InitialContext(env);
```

1. Look up a connection factory using JNDI:

   ```
   tconFactory = (TopicConnectionFactory) ctx.lookup(JMS_FACTORY);
   ```

2. Create a connection using the connection factory:

```
tcon = tconFactory.createTopicConnection();
```

3. Create a session using the connection. The following method defines the session as nontransacted and specifies that messages will be acknowledged automatically.

   For more information about setting session transaction and acknowledge modes, see *Programming WebLogic JMS* in the BEA WebLogic Server document set, available at the following URL:

   http://e-docs.bea.com/wls/docs70/jms/index.html

   ```
   tsession = tcon.createTopicSession(
      false, Session.AUTO_ACKNOWLEDGE
   );
   ```

4. Look up the destination (topic) using JNDI:

   ```
   topic = (Topic)ctx.lookup(NOTIFY_TOPIC);
   ```

5. Create a reference to a message producer (topic publisher) using the session and destination (topic). If a subscriber already exists, it is closed. Information about the organization and participant is used as the selector. For more information about defining selectors, see *Programming WebLogic JMS* in the BEA WebLogic Server document set, available at the following URL:

   http://e-docs.bea.com/wls/docs70/jms/index.html

   In the following example, `wlpi` represents an instance of the `com.bea.wlpi.client.common.WLPI` class:

```
if (tsubscriber != null) tsubscriber.close();
String selector = "orgId = '" +
wlpi.getWorklist().getActiveOrganization() + "' AND assigneeId = '" +
   wlpi.getUserId() + "' AND NOT role" + " AND action = 'assigned'";
tsubscriber = tsession.createSubscriber(topic, selector, false);
```

6. Register an asynchronous message listener:

   ```
   tsubscriber.setMessageListener(this);
   ```

7. Start the connection:

   ```
   tcon.start();
  }
   ```

When a message is delivered to the topic session, it is passed to the `onMessage()` method, which you must implement as described in "Receiving Messages Asynchronously" on page 6-7.

The following code provides an example of a message listener client that uses the notification to refresh its tasklist displays.

Note: For AWT/Swing applications, you must not update the user interface on the same thread as the call to the `onMessage()` method call, or you will encounter deadlocks and/or exceptions. You should marshall the call onto the AWT Event Dispatcher Thread. The Swing utilities provide the `invokeLater()` method to enqueue a runnable object on the AWT Event Dispatcher Thread, and you can include the update logic in the runnable object `run()` method.

```
import javax.swing.SwingUtilities;
public void onMessage(Message msg) {
   String action;
   TaskInfo task;
   try {
   // We're only interested in ObjectMessages with a TaskInfo
   // payload.
      if (!(msg instanceof ObjectMessage))
         return;
      Object data = ((ObjectMessage)msg).getObject();
      if (!(data instanceof TaskInfo))
         return;
      action = msg.getStringProperty("action");
      task = (TaskInfo)data;
   } catch (JMSException e) {
      e.printStackTrace();
      return;
   }

   if (action.equals("update"))
      taskUpdated(task);
   else if (action.equals("remove"))
      taskDeleted(task);
}

private void taskDeleted(final TaskInfo task) {
   SwingUtilities.invokeLater(new Runnable() {
      public void run() {
         vTasks.remove(task);
         int row = vDisplayedTasks.indexOf(task);
         if (row != -1) {
            vDisplayedTasks.remove(row);
            model.fireTableRowsDeleted(row, row);
         }
      }
   });
}
```

For more information about JMS, see *Programming WebLogic JMS*, in the BEA WebLogic Server document set, at the following URL:

`http://e-docs.bea.com/wls/docs70/jms/index.html`

7 Understanding the BPM Transaction Model

This section explains how transactions are handled in business process management (BPM) applications, including the following topics:

- How a Transaction Is Started
- How the Transaction Is Committed
- How Exceptions Are Handled
- How to Force a New Transaction to Start
- Examples of Transactions

For an overview of transactions, and details describing transaction processing and EJBs, see *Programming WebLogic JTA* in the WebLogic Server documentation set, available at the following URL:

`http://e-docs.bea.com/wls/docs70/jta/index.html`

How a Transaction Is Started

The following three actions, which trigger the WebLogic Integration process engine to begin or resume its execution of a workflow, may mark the start of a new transaction:

- API method call
- XML event
- Time-based event

Whether or not an API method call also starts a new transaction depends on whether the calling application currently has a transaction context. If an *external* transaction context currently exists (that is, if the application has created a transaction using JTA or has inherited a JTA transaction), the WebLogic Integration Enterprise JavaBean (EJB) uses it; otherwise, the EJB container automatically creates a new transaction in which to execute the call. With one exception (the `Audit` EJB), all EJB methods in the public API use container-managed transactions with the `TX_REQUIRED` transaction attribute.

Both XML and time-based events always start a new transaction. In each case, the transaction is started prior to dequeuing the message from the queue.

Note: The WebLogic Integration Worklist client application executes user commands by calling methods on the `Worklist` EJB within a discrete transaction. When you work with tasks in the WebLogic Integration Worklist application, the EJB container creates an encompassing transaction in which to execute each command. For example, the EJB container creates one transaction in which to execute a task, another in which to mark a task as done, a third in which to reassign a task to another user or role, and so on.

The Worklist client application is being deprecated as of this release of WebLogic Integration. For information about the features that are replacing it, see the *BEA WebLogic Integration Release Notes*.

How the Transaction Is Committed

To understand how the transaction is committed, you need to understand:

- How a workflow instance is processed by WebLogic Integration
- How a workflow instance reaches a quiescent state

Each topic is described in detail in the following sections.

How a Workflow Instance Is Processed

When first instantiating a workflow, WebLogic Integration executes the Created actions for all tasks in the workflow. It then processes the Start nodes, as follows:

- In the case of an event- or time-triggered workflow, the system activates the Start node corresponding to the event.
- In the case of a manually started or called workflow (subworkflow), the system activates all manual or called Start nodes, respectively.

The progress of the transaction after the initial instantiation depends on how the template definition is defined. Activation events propagate outward along the paths emanating from the point of origin, according to the built-in processing rules for each node type.

The following table describes the processing rules for each node type.

Table 7-1 Node Type Processing Rules

Node Type	Processing Rules
Decision	Upon activation, a Decision node first evaluates its condition, executes the appropriate list of true or false actions, and finally activates its true or false successor nodes.
`Done`	Upon activation, the Done node executes any associated actions, and marks as complete the current workflow, regardless of whether all Done nodes have been reached.

Table 7-1 Node Type Processing Rules (Continued)

Node Type	Processing Rules
Event	Upon activation, an Event node checks for any addressed messages that may have previously been received, and, if none are found, registers itself in the Event Watch table. For more information on addressed messages, see "Guaranteeing Message Delivery" on page 6-10. Once triggered, an Event node performs the following tasks in sequence: ■ Initializes variables ■ Executes associated actions ■ Activates successor nodes. The activation of successor nodes proceeds according to the rules for each node type.
Join	Upon activation by any single incoming path, an OR Join node evaluates the OR condition; upon activation by all incoming paths, the AND Join node evaluates the AND condition. If conditions are met, the Join node activates its successor nodes.
Start	Upon activation, a Start node performs the following tasks in sequence: ■ Initializes variables ■ Executes associated actions ■ Activates any successor nodes The activation of successor nodes proceeds according to the rules for each node type.
Task	Upon activation, a Task node executes the list of Activated tasks in sequence. The following summarizes the subsequent actions that may be taken: ■ If the task is executed, either by an action or API request, the Execute actions listed are executed. ■ If a task is marked done by an action or API request, the MarkedDone actions are executed. ■ If an action marks the workflow as complete, or if it aborts the workflow, no further processing occurs.

How a Workflow Instance Reaches a Quiescent State

When processing a workflow, the process engine returns control to the system once there are no more actions to execute synchronously—that is, when the workflow instance enters a *quiescent* state, in which it waits for the next request.

The workflow instance enters a quiescent state when the following conditions are true:

- All Task nodes are waiting to be activated, executed, or are marked done.
- All Event nodes are waiting to be activated, or have already been triggered.
- All Decision and Join nodes are waiting for the final input to activate them, or they have already been activated.
- All Start nodes either are waiting to be activated or have already been executed.
- All Done nodes are waiting to be activated.

A quiescent state also marks the end of a transaction (and results in the transaction being committed) if and only if a new transaction was started at the point that the process engine was last triggered. If an external transaction context was exported before the API call trigger, then the external transaction remains in effect, and control is returned to the calling application. For more information, see "How a Transaction Is Started" on page 7-2.

When an action instantiates a subworkflow, all work performed by the subworkflow up to the point at which it enters a quiescent state (if ever) occurs within the same transaction as the calling workflow. This work typically includes execution of actions in all called Start nodes, activation of all successor nodes associated with each Start node, and the actions associated with each successor. This processing propagates through the subworkflow, resulting in either completion or quiescence, according to the rules previously described. If the subworkflow runs to completion (that is, if it does not quiesce), the Completed actions in the parent workflow are also performed within the same transaction. If the Completed actions include an action to mark a task (for example, the parent) as complete, processing continues, the MarkedDone actions are performed, and successor nodes are activated.

How Exceptions Are Handled

When the process engine catches or throws an exception, the decision to commit or roll back the transaction depends on the following two factors:

- Where the exception occurred
- Behavior of the active exception handler

The active exception handler can perform associated corrective actions and identify one of the following methods with which to proceed:

- Roll back the user transaction and rethrow the exception
- Stop processing and commit the return to the caller
- Retry the failed operation
- Ignore the error and continue processing

If the user transaction has not already been set for rollback only and the exception is recoverable (in other words, if the exception is a workflow warning), and the exception occurred while an action was being processed or an event variable was being set, the system allows the exception handler to decide the outcome. Otherwise, the system sets the user transaction for rollback only, invokes the active exception handler, and then rethrows the exception.

If the container creates the transaction, then it commits or rolls back the transaction when control is returned to it from the process engine. (Control is returned when the workflow instance reaches a quiescent state, as described in "How a Workflow Instance Reaches a Quiescent State" on page 7-5.)

If an application enlists a BPM EJB in an external transaction and this EJB throws an exception that does not result in the user transaction being set for rollback only, the transactional behavior is defined by the application. On the other hand, if the BPM call succeeds, there is no guarantee that the transaction will be committed by the external application (or by the EJB). An exception or other error condition may occur subsequently, resulting in the rollback of the entire transaction.

For XML and time-based events, if a transaction is rolled back, the message is returned to the queue. Redelivery of the message is attempted at specific intervals until the number of retries is exhausted. The maximum number of retries and the retry interval are configurable for each JMS destination using the Redelivery-Limit and RedeliveryDelayOverride attributes for the JMSTemplate elements. For example, for the WebLogic Integration samples domain, the WLI_JMSTemplate is defined as follows:

```
<JMSTemplate ErrorDestination="WLI_FailedEvent"
       Name="WLI_JMSTemplate"
       RedeliveryDelayOverride="60000"
       RedeliveryLimit="10"/>
```

For more information about these attributes, see the description of the JMSTemplate element within the *BEA WebLogic Server Configuration Reference* at the following URL:

http://e-docs.bea.com/wls/docs70/config_xml/index.html

How to Force a New Transaction to Start

You can force a new transaction by defining a *dummy* task action that causes the current workflow instance to reach a quiescent state. (For information describing how a workflow instance reaches a quiescent state, see "How a Workflow Instance Reaches a Quiescent State" on page 7-5.)

For example, you can force a workflow instance to enter a quiescent state by defining one of the following task actions:

- Timed event that performs no actions, with a trigger of one second in the future
- Posting of an XML event that sends a dummy trigger message to itself

Examples of Transactions

When business operations are executed within a workflow instance, the following factors help determine whether the business operations are treated as a single transaction or as multiple transactions:

- How you define the workflow

  Specifically, whether the workflow is defined such that the existing transaction enters a quiescent state before all business operations are executed.

- For an EJB business operations, how the EJBs are deployed

  If you want an application-defined EJB to participate in the current BPM transaction, then the EJB *must* be deployed with one of the the following container-managed transactional attributes: `Mandatory`, `Required`, or `Supports`. For a detailed description of these settings, see Section 16.7.2 in the Enterprise JavaBeans Specification 2.0, published by Sun Microsystems, Inc.

The following sections provide examples of workflows that are defined such that business operations are specified as part of the actions of one single task, and the actions of multiple tasks. Each example illustrates the scenarios as both a single transaction and/or multiple transactions.

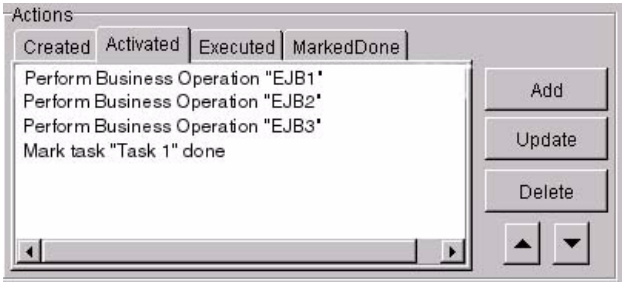
Example 1: Business Operations Defined as Actions in One Task

For the first example, suppose that you want to execute three EJBs both as separate business operations and as part of the actions of a single task. Based on how you define the task properties, the three business operations can be treated as a single transaction or as multiple transactions.

Single Transaction

To have the specified business operations treated as a *single* transaction, the following example shows how you might define the task actions on the Activated Actions tab in the Task Properties dialog box.

Figure 7-1 Transaction Example: One Task, Single Transaction

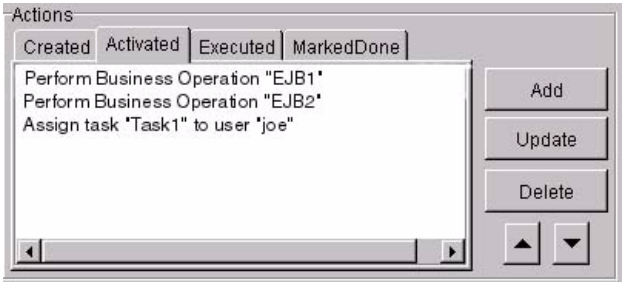


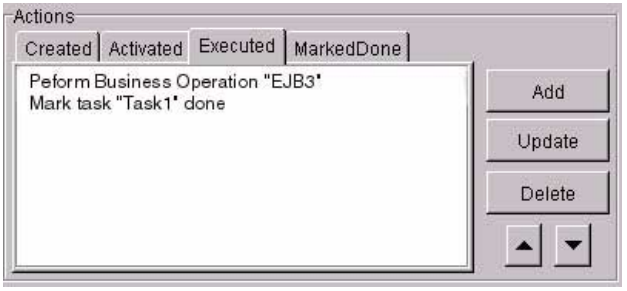
In this example, all business operations are invoked upon task activation, and are treated as a single transaction.

Multiple Transactions

To have the specified business operations treated as *multiple* transactions, the following example show how you might define the task actions on the Activated and Executed Actions tabs in the Task Properties dialog box.

Figure 7-2 Transaction Example: One Task, Multiple Transactions





In this example, a subset of the business operations (EJB1 and EJB2) are invoked upon task activation, as one transaction. Once the `Assign task "EJB1" to user "joe"` action is executed, the workflow instance enters a quiescent state, and waits for the next request. The first transaction is marked complete, and the Executed actions are performed. The EJB3 business operation is then invoked, as a separate transaction.

In this example, if an exception occurs during the processing of EJB3, the active exception handler performs the associated corrective actions and identifies the method by which to proceed. The first transaction is not affected by an exception that occurs in the second transaction. For more information, see "How Exceptions Are Handled" on page 7-6.

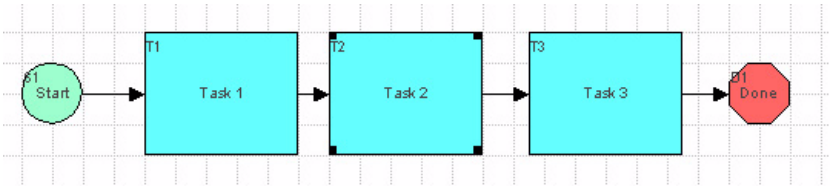
Example 2: Business Operations Defined as Actions in Multiple Tasks

For the second example, suppose you want to invoke three EJBs as separate business operations, and as actions in three separate tasks. Again, based on how you define the task properties, the three business operations can be treated as a single transaction or as multiple transactions.

Single Transaction

For this example, consider the following workflow template.

Figure 7-3 Transaction Example: Multiple Tasks, Single Transaction



Suppose the following statements are true:

- Task1 performs the EJB1 business operation and marks itself complete.
- Task2 performs the EJB2 business operation and marks itself complete.
- Task3 performs the EJB3 business operation and marks itself complete.

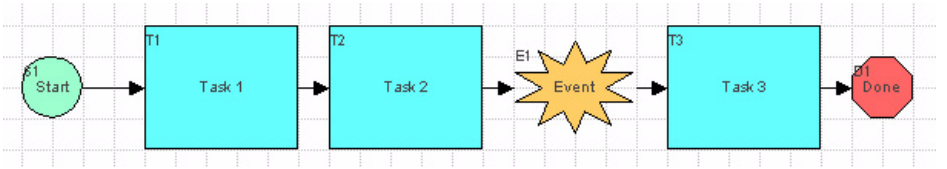
If these statements are true, then the workflow instance does not enter a quiescent state before the execution of any business operation, and all three business operations are treated as a single transaction.

Multiple Transactions

This example illustrates how you might define a dummy Post XML event to force a quiescent state (and force a new transaction to start). For more information about forcing a new transaction to start, see “How to Force a New Transaction to Start” on page 7-7.

For this example, consider the following workflow template.

Figure 7-4 Transaction Example: Multiple Tasks, Multiple Transactions



Suppose the following statements are true:

- Task1 performs the EJB1 business operation and marks itself complete.
- Task2 performs the EJB2 business operation and marks itself complete.

- The Event node is triggered, and the workflow enters a quiescent state.

  Note: It is possible that the Event Watch may not have been triggered, at the time an incoming event notification is delivered, and the associated message may be missed and/or ignored. You can use addressed messaging to guarantee message delivery by defining the required message header fields and time-to-live value, as described in "Guaranteeing Message Delivery" on page 6-10.

- Task3 performs the EJB3 business operation and marks itself complete.

If these statements are true, then EJB1 and EJB2 are treated as a single transaction, and EJB3 is treated as a separate transaction.

In this example, if an exception occurs during the processing of EJB3, the active exception handler performs the associated corrective actions and identifies the method by which to proceed. The first transaction is not affected by an exception that occurs in the second transaction. For more information, see "How Exceptions Are Handled" on page 7-6.

CHAPTER

8 Disconnecting from the Process Engine

This section explains how to disconnect from the WebLogic Integration process engine. It includes the following topics:

- Removing Session EJB References
- Releasing Other Resources

Removing Session EJB References

Once you have completed all application tasks, you can remove the session EJB references to make the system space available for use by other EJBs.

The session EJB home and remote interfaces defined in the table "API Session EJB Home and Remote Interfaces" on page 3-1 inherit methods from the `javax.ejb.EJBHome` and `javax.ejb.EJBObject` interfaces, respectively.

The following table defines the inherited methods that can be used to remove EJB references.

EJB Interface	Method	Purpose
Home	`public void remove(javax.ejb.Handle handle ) throws java.rmi.RemoteException, javax.ejb.RemoveException`	Removes the EJB object associated with the specified handle.
	`public javax.ejb.HomeHandle getHomeHandle() throws java.rmi.RemoteException`	Gets a handle for the home interface object.
Remote	`public void remove() throws java.rmi.RemoteException, javax.ebj.RemoveException`	Removes the EJB object.
	`public javax.ejb.EJBHome getEJBHome() throws java.rmi.RemoteException`	Gets the EJB home interface.
	`public javax.ejb.Handle getHandle() throws java.rmi.RemoteException`	Gets a handle for the EJB object. This value can be used as the *`handle`* parameter in the Home interface `remove()` method, described previously in this table.

For example, the following code excerpt from the JSP worklist example shows how to remove the `Worklist` EJB reference, including the session variable. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB. The `session` variable is an `HttpSession` object in which you previously stored the `worklist` remote object reference.

```
Worklist worklist = null;
worklist = (Worklist)session.getValue("worklist");

if (worklist != null) {
    try {
       worklist.remove();
    } catch (Exception e) {
    }
```

```
    worklist = null;
    session.removeValue("worklist");
}
```

For more information about the methods inherited by the home and remote interfaces, see the `javax.ejb.EJBHome` and `javax.ejb.EJBObject` Javadoc, respectively.

Releasing Other Resources

In addition to removing the session EJB references, you should perform the tasks described in the following sections:

- Stopping and Closing JMS Connections
- Closing the Context

Stopping and Closing JMS Connections

Typically, a JMS Provider allocates a significant amount of resources when it creates a connection. When a connection is no longer being used, you should close it to release the resources associated with it. A connection can be closed using the following `javax.jms.Connection` method:

```
public void close(
) throws javax.jms.JMSException
```

For example, the following code provides an example of how to disconnect from JMS by closing the connection:

```
try {
    tcon.close();
} catch (Exception e) {
       ExceptionHandler.reportException(e, this);
  }
  tcon = null;
  tsession = null;
  tsubscriber = null;
}
```

Once the connection is closed, the application sets the connection, session, and topic subscriber variables to `null` so that the garbage collector can free the resources used by these objects.

Closing the Context

To close the JNDI context resources (instead of waiting for them to be released automatically by the garbage collector), use the following `javax.naming.Context` method:

```
public void close() throws javax.naming.NamingException
```

For example, the following code closes the JNDI context resources:

```
try {
    ctx.close();
    {
    catch(Exception exp)
    {
    }
}
```

For more information, see the `javax.naming.Context` Javadoc.

CHAPTER

9 Configuring the Security Realms

This section explains how to configure the security realm, including the following topics:

- Getting Basic Security Information
- Configuring Organizations, Roles, and Users
- Mapping Security Information
- Configuring Permissions

For more information about implementing security for tasks associated with managing business processes, see "About Security Realms" in "Administering Data" in *Using the WebLogic Integration Studio*.

Getting Basic Security Information

Using the `com.bea.wlpi.server.principal.WLPIPrincipal` methods and/or the `com.bea.wlpi.client.common.WLPI` class convenience methods, you can get the following security-related information:

- Class name of the WebLogic Server security realm, which allows you to determine, for example, whether the default WebLogic Server security realm is in use
- Whether or not the WebLogic Server security realm is manageable and/or persistent
- Process engine URL
- User ID

The following sections explain how to get these types of security information, and provides an example.

Getting the Security Realm Class Name

To get the class name of the WebLogic Server security realm, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.lang.String getSecurityRealmClassName(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

This method returns the fully qualified security realm class name.

For example, the following code gets the class name of the WebLogic security realm and saves it to the string `security_class`. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB:

```
String security_class = principal.getSecurityRealmClassName();
```

For more information about the `getSecurityRealmClassName()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Determining Whether the Security Realm Is Manageable and/or Persistent

In a *manageable* security realm, applications can update group and user security information. In a *nonmanageable* security realm, applications can view the security information for groups and users only. For more information, see "Managing Security" in the *BEA WebLogic Server Administration Guide* in the BEA WebLogic Server documentation set. This document is available, in the BEA WebLogic Server document set, at the following URL:

http://e-docs.bea.com/wls/docs70/adminguide/index.html

To determine whether or not the security realm is manageable, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public boolean isManageableSecurityRealm(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

This method returns `true` if the security realm is manageable (that is, if it implements the `ManageableRealm` interface), and `false` if it is nonmanageable.

To determine whether or not the security realm is both manageable *and* persistent, use the following `com.bea.wlpi.client.common.WLPI` method:

```
public boolean allowSecurityRealmUpdates(
) throws java.lang.IllegalStateException
```

This method returns `true` if the security realm is manageable *and* persistent, and `false` if both characteristics are not present.

For example, the following code determines whether or not the security realm is manageable. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB:

```
Boolean ismanageable = principal.isManageableSecurityRealm();
```

For more information about the `isManageableSecurityRealm()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc. For more information about the `allowSecurityRealmUpdates()` method, see the `com.bea.wlpi.client.common.WLPI` Javadoc.

Getting the Server URL

To get the URL of the WebLogic Integration process engine, use the following `com.bea.wlpi.client.common.WLPI` method:

```
public java.lang.String getURL()
```

This method returns the process engine URL. If you are not currently logged on, this method returns `null`.

For example, the following code gets the process engine URL and saves it to `url`:

```
String url = com.bea.wlpi.client.common.WLPI.getURL();
```

For more information about the `getURL()` method, see the `com.bea.wlpi.client.common.WLPI` Javadoc.

Getting the User ID

To get the current user ID, use the following `com.bea.wlpi.client.common.WLPI` method:

```
public java.lang.String getUserId()
```

This method returns the user ID. If you are not currently logged on, this method returns `null`.

For example, the following code gets the user ID and saves it to `user`:

```
String user = com.bea.wlpi.client.common.WLPI.getUserId();
```

For more information about the `getUserId()` method, see the `com.bea.wlpi.client.common.WLPI` Javadoc.

Example of Getting Basic Security Information

This section provides excerpts from the command-line administration example showing how to get basic security realm information.

Note: For more information about the command-line administration example, see "Command-Line Administration Example" on page 1-21.

In this example, an input stream is defined to communicate with the user, and the user is prompted to select an action from a menu. If the user selects the `Security Realm` option, the system displays the security class name and indicates whether or not the security realm is manageable.

Important lines of code are highlighted in **bold**. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) )

/* Display Tool Title */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/* Security Realm */
         case '4' :
            /* Display WLPI security realm
             * properties and attributes */
            System.out.println( "\nSecurity Realm:" );
```

```
/* Retrieve and display WLPI security realm
 * class currently use */
System.out.println( "- Class Name: " +
/* WLPI Public API Method */
/* NOTE: Would be nice to add code to capture any
 * thrown exceptions */
principal.getSecurityRealmClassName( ) );
/* WLPI Public API Method */
/* Retrieve and display info on manageability of WebLogic Process
 * Integrator security realm currently use */
/* NOTE: Would be nice to add code to capture any
 * thrown exceptions */
if( principal.isManageableSecurityRealm( ) )
   System.out.println( "- This realm is manageable" );
else
   System.out.println( "- This realm is not manageable" );

   break;
   .
   .
   .
```

Configuring Organizations, Roles, and Users

Organizations are defined using the Studio client or a custom definition client to represent different business entities, geographical locations, or any other classifications that are relevant to the particular business of the company. You can configure roles within organizations, and users to further delineate security permissions.

This section describes the following topics:

- Configuring Organizations
- Configuring Roles
- Configuring Users

Configuring Organizations

The following sections explain how to configure organizations, including:

- Adding an Organization
- Adding a User to an Organization
- Getting All Organizations
- Getting the Roles Defined for an Organization
- Getting the Users Defined for an Organization
- Getting Organization Information
- Setting Organization Information
- Deleting a User from an Organization
- Deleting an Organization
- Example of Configuring Organizations

Adding an Organization

To add an organization to the security realm, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void addOrganization(
  com.bea.wlpi.common.OrganizationInfo orgInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `addOrganization()` method parameter for which you must specify a value.

Table 9-1 addOrganization() Method Parameter

Parameter	Description	Valid Values
orgInfo	New organization information.	An `OrganizationInfo` object. For information about defining the `OrganizationInfo` object, see "OrganizationInfo Object" on page B-9.

For example, the following code adds an organization, based on the contents of the specified `orgInfo` object. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB:

```
principal.addOrganization(orgInfo);
```

For more information about the `addOrganization()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Adding a User to an Organization

To add a user to an organization, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void addUserToOrganization(
  java.lang.String userId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `addUserToOrganization()` method parameters for which you must specify values.

Table 9-2 addUserToOrganization() Method Parameters

Parameter	Description	Valid Values
userId	ID of the user that you want to add to the organization.	String specifying a valid user ID. For information about getting a list of users, see "Getting All Users" on page 9-49.

Table 9-2 addUserToOrganization() Method Parameters (Continued)

Parameter	Description	Valid Values
orgId	ID of the organization to which you want to add the user.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

For example, the following code adds the user `joe` to the specified organization, `ORG1`. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB:

```
principal.addUserToOrganization("joe", "ORG1");
```

For more information about the `addUserToOrganization()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting All Organizations

To get a list of all organizations defined for the security realm, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.util.List getAllOrganizations(
   boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getAllOrganizations()` method parameter for which you must specify a value.

Table 9-3 getAllOrganizations() Method Parameter

Parameter	Description	Valid Values
obtainAttributes	Boolean flag specifying whether you want to get all attributes or only the organization IDs.	`true` (all attributes) or `false` (organization IDs only).

This method returns a list of `com.bea.wlpi.common.OrganizationInfo` objects. To access information about each organization, use the `OrganizationInfo` object methods described in "OrganizationInfo Object" on page B-9.

For example, the following code gets only the organization IDs (the `obtainAttributes` parameter is set to false) and saves them to the `orgList` list variable. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB:

```
List orgList = principal.getAllOrganizations(false);
```

For more information about the `getAllOrganizations()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting the Roles Defined for an Organization

You can get a list of the roles defined for an organization or determine whether a specific role is defined for an organization using the methods described in the following sections.

Getting a List of Roles Defined for an Organization

To get a list of all the roles defined for an organization, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.util.List getRolesInOrganization(
  java.lang.String orgId,
  boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getRolesInOrganization()` method parameters for which you must specify values.

Table 9-4 getRolesInOrganization() Method Parameters

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to get roles.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
obtainAttributes	Flag specifying whether you want to get all attributes or only the role IDs.	`true` (all attributes) or `false` (role IDs only).

This method returns a list of `com.bea.wlpi.common.RoleInfo` objects. To access information about each role, use the `RoleInfo` object methods described in "RoleInfo Object" on page B-17.

For example, the following code gets the roles in the `ORG1` organization, returning all attributes (as the `obtainAttributes` parameter is set to `true`). In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
List roles = principal.getRolesInOrganization("ORG1", true);
```

For more information about the `getRolesInOrganization()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Determining Whether a Role Is Defined for an Organization

To determine whether or not a specific role is defined for an organization, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public boolean isRoleInOrganization(
  java.lang.String roleId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `isRoleInOrganization()` method parameters for which you must specify values.

Table 9-5 isRoleInOrganization() Method Parameters

Parameter	Description	Valid Values
roleId	ID of the role that you want to verify.	String specifying a valid role ID. For information about getting a list of roles, see "Getting the Roles Defined for an Organization" on page 9-10.
orgId	ID of the organization that you want to check.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

This method returns `true` if the role is defined for the organization and `false` if it is not.

For example, the following code determines whether or not the `role1` role is defined for the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
List roles = principal.isRoleInOrganization("role1", "ORG1");
```

For more information about the `isRoleInOrganization()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting the Users Defined for an Organization

You can get a list of all the users defined for an organization or determine whether a specific user is defined for an organization using the methods described in the following section.

Getting a List of Users Defined for an Organization

To get a list of users defined for an organization, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.util.List getUsersInOrganization(
  java.lang.String orgId,
  boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getUsersInOrganization()` method parameters for which you must specify values.

Table 9-6 getUsersInOrganization() Method Parameters

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to get a list of users.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
obtainAttributes	Flag specifying whether you want to get all attributes or only the user IDs.	`true` (all attributes) or `false` (user IDs only).

This method returns a list of com.bea.wlpi.common.UserInfo objects. To access information about each role, use the UserInfo object methods described in "UserInfo Object" on page B-26.

For example, the following code gets the list of users in the ORG1 organization, returning all attributes (as the obtainAttributes parameter is set to true). In this example, principal represents the EJBObject reference to the WLPIPrincipal EJB.

```
List users = principal.getUsersInOrganization("ORG1", true);
```

For more information about the getUsersInOrganization() method, see the com.bea.wlpi.server.principal.WLPIPrincipal Javadoc.

Determining Whether a User Is Defined for an Organization

To determine whether or not a specific user is defined for an organization, use the following com.bea.wlpi.server.principal.WLPIPrincipal method:

```
public void isUserInOrganization(
  java.lang.String userId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the isUserInOrganization() method parameters for which you must specify values.

Table 9-7 isUserInOrganization() Method Parameters

Parameter	Description	Valid Values
userId	ID of the user that you want to verify.	String specifying a valid user ID. For information about getting a list of users, see "Getting All Users" on page 9-49.
orgId	ID of the organization that you want to check.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

This method returns `true` if the user is defined for the organization, and `false` if it is not.

For example, the following code determines whether or not the `user1` user is defined for the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
List roles = principal.isUserInOrganization("user1", "ORG1");
```

For more information about the `isUserInOrganization()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting Organization Information

To get organization information, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public com.bea.wlpi.common.OrganizationInfo getOrganizationInfo(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getOrganizationInfo()` method parameter for which you must specify a value.

Table 9-8 getOrganizationInfo() Method Parameter

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to get information.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

This method returns a `com.bea.wlpi.common.OrganizationInfo` object. To access information about the organization, use the `OrganizationInfo` object methods described in "OrganizationInfo Object" on page B-9.

For example, the following code gets the organization corresponding to the specified organization ID. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
OrganizationInfo orgInfo = principal.getOrganizationInfo(orgId)
```

For more information about the `getOrganizationInfo()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Setting Organization Information

To set organization information, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method.

```
public void setOrganizationInfo(
  com.bea.wlpi.common.OrganizationInfo orgInfo
) throws java.rmi.RemoteException
```

The following table describes the `setOrganizationInfo()` method parameter for which you must specify a value.

Table 9-9 setOrganizationInfo() Method Parameter

Parameter	Description	Valid Values
orgInfo	Organization information to be updated.	An `OrganizationInfo` object. For information about getting a list of all `OrganizationInfo` objects, see "Getting All Organizations" on page 9-9. (Be sure to set the Boolean parameter, `obtainAttributes`, to `true` to avoid inadvertently clearing any organization attributes.) For information about updating an `OrganizationInfo` object, see "OrganizationInfo Object" on page B-9.

For example, the following code updates an organization based on the contents of the specified `orgInfo` object. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.updateOrganization(orgInfo);
```

For more information about the `setOrganizationInfo()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Deleting a User from an Organization

To delete a user from an organization, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void removeUserFromOrganization(
  java.lang.String userId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `removeUserFromOrganization()` method parameters for which you must specify values.

Table 9-10 removeUserFromOrganization() Method Parameters

Parameter	Description	Valid Values
userId	ID of the user that you want to delete.	String specifying a valid user ID. For information about getting a list of users, see "Getting All Users" on page 9-49.
orgId	ID of the organization associated with the user.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

For example, the following code removes the `user1` user from the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.removeUserFromOrganization("user1", "ORG1");
```

For more information about the `removeUserFromOrganization()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Deleting an Organization

To delete an organization, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void deleteOrganization(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteOrganization()` method parameter for which you must specify a value.

Table 9-11 deleteOrganization() Method Parameter

Parameter	Description	Valid Values
orgId	ID of the organization that you want to delete.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

The following code example deletes the organization specified by the organization ID. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.deleteOrganization(organization.getId())
```

The organization ID is obtained using the methods associated with the `com.bea.wlpi.common.OrganizationInfo` object, `organization`. The `organization` object can be obtained using the methods described in "Getting All Organizations" on page 9-9.

For more information about the `deleteOrganization()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Example of Configuring Organizations

This section provides excerpts from the command-line administration example showing how to configure organizations.

Note: For more information about the command-line administration example, see "Command-Line Administration Example" on page 1-21.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Adding an Organization
- Deleting an Organization
- Setting Organization Information
- Adding a User to an Organization
- Deleting a User from an Organization
- Getting All Organizations
- Getting the Users Defined for an Organization
- Determining Whether a User Is Defined for an Organization
- Getting the Roles Defined for an Organization
- Determining Whether a Role Is Defined for an Organization

Important lines of code are highlighted in **bold**. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) )

/* Display Tool Title */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---             Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
```

```
      .
      .
public static void mngOrganizations( ) {
   String answer;
   String calendarId;
   String orgId;
   String userId;
   String eMail;
   String defaultOrgId;

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader( new InputStreamReader( System.in )
);

   try {
   /* WLPI Public API Method */
      boolean isRealmManageable = principal.isManageableSecurityRealm( );

      /* Display the menu and interact with user */
      while( true ) {
         /* Display the menu */
         System.out.println( "\n\n---        WLPI Organizations        ---" );
         System.out.println( "\nEnter choice:" );
         /* Is the realm manageable ? */
         if( isRealmManageable ) {
            /* The realm is manageable realm; Display menu options that
             * requires a manageable realm */
            System.out.println( "0) Add a new Organization" );
            System.out.println( "1) Delete an Organization" );
            System.out.println( "2) Update Organization Info" );
            System.out.println( "3) Assign a User to an Organization" );
            System.out.println( "4) Remove a User from an Organization" );
         }
         System.out.println( "5) List all Organizations" );
         System.out.println( "6) List Organization Info" );
         System.out.println( "7) List Users assigned to an Organization" );
         System.out.println( "8) Is User assigned to an Organization" );
         System.out.println( "9) List Roles defined in an Organization" );
         System.out.println( "A) Is Role in an Organization" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* Get user's selection */
         String line = stdIn.readLine( );

         /* User pressed enter without making a selection ? */
         if( line.equals( "" ) )
            continue;
```

```
/* User entered more than one char ? */
else if( line.length( ) > 1 ) {
   System.out.println( "*** Invalid selection" );
   continue;
}
/* Realm is not manageable and user entered a hidden selection ? */
else if( !isRealmManageable && line.charAt( 0 ) < '5' ) {
   System.out.println( "*** Invalid selection" );
   continue;
}

/* Convert to uppercase and to char */
char choice = line.toUpperCase( ).charAt( 0 );

/* Process user's selection */
switch( choice ) {
.
.
.
```

Adding an Organization

The following excerpt shows how to add an organization.

```
/* Add a New Organization */
case '0' :
   /* Get Organization ID for the new organization to add */
   if( ( orgId = askQuestion( "\nEnter new Organization ID" ) )
      == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Calendar ID to assign to this organization (optional) */
   calendarId = askQuestion( "Enter Calendar ID (press
      enter for none)" );

   /* Create an OrganizationInfo object; required to add
    * a new organization */
   OrganizationInfo orgInfo =
     new OrganizationInfo( orgId, calendarId );

   try {
      /* WLPI Public API Method */
      /* Add the new organization */
      principal.addOrganization( orgInfo );
```

```
   /* Success (No exception thrown) */
   System.out.println( "- Success" );
}
catch( Exception e ) {
   System.out.println( "*** Unable to add the organization\n" );
   System.err.println( e );
}
break;
.
.
.
```

Deleting an Organization

The following excerpt shows how to delete an organization.

```
/* Delete an Organization */
case '1' :
   /* Get Organization ID for the organization to remove */
   if( ( orgId = askQuestion( "\nEnter Organization
      ID to delete" ) ) == null ) {
   /* User cancelled the operation */
   System.out.println( "*** Cancelled" );
   break;
}

try {
   /* WLPI Public API Method */
   /* Remove the organization */
   principal.deleteOrganization( orgId );

   /* Success (No exception trown) */
   System.out.println( "- Success" );
}
catch( Exception e ) {
   System.out.println( "*** Unable to delete the organization\n" );
   System.err.println( e );
}
break;
.
.
.
```

Setting Organization Information

The following excerpt shows how to set information about the organization.

```
/* Update Organization Info */
case '2' :
/* Get Organization ID for the organization to update */
   if( ( orgId = askQuestion(
      "\nEnter Organization ID to update" ) ) == null ) {
   /* User cancelled the operation */
   System.out.println( "*** Cancelled" );
   break;
}

   /* Get Calendar ID; only organization attribute defined in
    * WLPI v1.2.1 thus, only attribute that can be updated */
   calendarId = askQuestion(
      "Enter new Calendar ID (press enter for none)" );

   /* Create an OrganizationInfo object; required to update
    * the organization */
   orgInfo = new OrganizationInfo( orgId, calendarId );

   try {
      /* WLPI Public API Method */
      /* Update the organization (read calendar) */
      principal.setOrganizationInfo( orgInfo );

      /* Success (No exception trown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to update the organization\n" );
      System.err.println( e );
   }
   break;            .
   .
   .
```

Adding a User to an Organization

The following excerpt shows how to assign a user to an organization.

```
/* Assign a User to an Organization */
        case '3' :
           /* Get User ID for the user to assign to an organization */
           if( ( userId = askQuestion( "\nEnter User ID to assign" )
```

```
   ) == null ) {
/* User cancelled the operation */
System.out.println( "*** Cancelled" );
break;
}

/* Get Organization ID where the user is to be assigned */
if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
   /* User cancelled the operation */
   System.out.println( "*** Cancelled" );
   break;
}

try {
   /* WLPI Public API Method */
   /* Assign the user to the organization */
   principal.addUserToOrganization( userId, orgId );

   /* Success (No exception trown) */
   System.out.println( "- Success" );
}
catch( Exception e ) {
   System.out.println( "*** Unable to
      assign user to the organization\n" );
   System.err.println( e );
}
break;
.
.
.
```

Deleting a User from an Organization

The following excerpt shows how to remove a user from an organization.

```
/* Remove a User from an Organization */
case '4' :
   /* Get User ID for the user to remove from an organization */
   if( ( userId = askQuestion(
      "\nEnter User ID to remove" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
    }

   /* Get Organization ID where the user is to be removed */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
```

```
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Remove this user from this organization */
      principal.removeUserFromOrganization( userId, orgId );

      /* Success (No exception trown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println(
         "*** Unable to remove user from organization\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Getting All Organizations

The following excerpt shows how to get a list of all organizations.

```
/* List all Organizations */
case '5' :
   /* Prompt user to select if we need to display
    * the organization attributes */
   if( ( answer = askQuestion(
      "\nList all attributes (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Parse the answer */
   boolean isGetAttributes = ( answer.equals( "y" ) ||
      answer.equals( "Y" ) );

   /* WLPI Public API Method */
   /* Retrieve all organizations */
   /* NOTE: Would be nice to add code to capture any
    * thrown exceptions */
   List orgList = principal.getAllOrganizations( isGetAttributes );
```

```
   /* Any organizations defined ? */
   if( orgList.size( ) == 0 )
      System.out.println( "\nNo Organization defined" );
   else
      System.out.println( "\nDefined organizations:" );

   /* Process the list to display organization and attributes */
   for( int i = 0; i < orgList.size( ); i++ ) {
      /* Retrieve an element from the list */
      orgInfo = ( OrganizationInfo )orgList.get( i );
      /* Retrieve and display organization id */
      System.out.println( "- ID: " + orgInfo.getOrgId( ) );

      /* Display attributes ? */
      if( isGetAttributes ) {
         /* Retrieve calendar id */
         if( ( calendarId = orgInfo.getCalendarId( ) ) == null )
            /* Assigned 'none' if none defined for this organization */
            calendarId = "None";

         /* Display calendar id */
         System.out.println( "
            Attributes: Calendar ID=" + calendarId + "\n" );
      }
   }
   break;
   .
   .
   .
```

Getting Organization Information

The following excerpt shows how to get information about an organization.

```
/* List Organization Info */
case '6' :
   /* Get Organization ID for the organization to display */
   if( ( orgId = askQuestion( "\nEnter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Retrieve info on this organization */
```

```
            orgInfo = principal.getOrganizationInfo( orgId );

            /* Retrieve and display organization id */
            System.out.println( "- ID: " + orgInfo.getOrgId( ) );

            /* Retrieve calendar id */
            if( ( calendarId = orgInfo.getCalendarId( ) ) == null )
               /* Assigned 'none' if none defined for this organization */
               calendarId = "None";

            /* Display calendar id */
            System.out.println( "
               Attributes: Calendar ID=" + calendarId + "\n" );
         }
         catch( Exception e ) {
            System.out.println(
               "*** Unable to retrieve organization info\n" );
            System.err.println( e );
         }
         break;
         .
         .
         .
```

Getting the Users Defined for an Organization

The following excerpt shows how to get a list of users defined for an organization.

```
/* List Users assigned to an Organization */
case '7' :
   /* Get Organization ID to query for */
   if( ( orgId = askQuestion( "\nEnter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Prompt user to select if we need to display
    * the user attributes */
   if( ( answer = askQuestion(
      "List user attributes (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Parse the answer */
```

```
isGetAttributes = ( answer.equals( "y" ) || answer.equals( "Y" ) );

/* WLPI Public API Method */
/* Retrieve all users assigned to this organization */
/* NOTE: Would be nice to add code to capture any
 * thrown exceptions */
List userList = principal.getUsersInOrganization(
   orgId, isGetAttributes );

/* Any user assigned ? */
if( userList.size( ) == 0 )
   System.out.println( "\nNo user assigned" );
else
   System.out.println( "\nAssigned Users:" );

/* Process the list to display users and attributes */
for( int i = 0; i < userList.size( ); i++ ) {
/* Retrieve an element from the list */
   UserInfo userInfo = ( UserInfo )userList.get( i );
   /* Retrieve and display user id */
   System.out.println( "- User ID: " + userInfo.getUserId( ) );

   /* Display attributes ? */
   if( isGetAttributes ) {
      /* Retrieve eMail address */
      if( ( eMail = userInfo.getEMailAddress( ) ) == null )
         /* Assigned 'none' if none defined for this user */
         eMail = "None";

      /* Retrieve default organization id */
      if( ( defaultOrgId = userInfo.getDefaultOrgId( ) ) == null )
         /* Assigned 'none' if none defined for this user */
         defaultOrgId = "None";

      /* Retrieve calendar id */
      if( ( calendarId = userInfo.getCalendarId( ) ) == null )
         /* Assigned 'none' if none defined for this user */
         calendarId = "None";

      /* Display email address, default ord id and calendar id */
      System.out.println( "  Attributes:\n  - eMail: " + eMail );
      System.out.println( "  - Default ORG ID: " + defaultOrgId );
      System.out.println( "  - Calendar ID=" + calendarId + "\n" );
   }
}
break;
.
.
```

```
.
```

Determining Whether a User Is Defined for an Organization

The following excerpt shows how to determine whether a user defined for an organization.

```
/* Is User assigned to an Organization */
case '8' :
   /* Get User ID to query for */
   if( ( userId = askQuestion( "\nEnter User ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Organization ID for organization to query */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
   /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* WLPI Public API Method */
   /* Is the user assigned to this organization ? */
   /* NOTE: Would be nice to add code to capture any
    * thrown exceptions */
   if( principal.isUserInOrganization( userId, orgId ) )
      System.out.println( "User is assigned to the organization" );
   else
      System.out.println( "User is not assigned to the organization" );
   break;
   .
   .
   .
```

Getting the Roles Defined for an Organization

The following excerpt shows how to get a list of roles defined for an organization.

```
/* List Roles defined in an Organization */
case '9' :
   /* Get Organization ID for organization to query */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
```

```
    System.out.println( "*** Cancelled" );
    break;
 }

 /* Prompt user to select if we need to display the user attributes */
if( ( answer = askQuestion(
   "List role attributes (y/n)?" ) ) == null ) {
    /* User cancelled the operation */
    System.out.println( "*** Cancelled" );
    break;
 }

 /* Parse the answer */
  isGetAttributes = ( answer.equals( "y" ) || answer.equals( "Y" ) );

 /* WLPI Public API Method */
 /* Retrieve all roles defined in this organization */
 /* NOTE: Would be nice to add code to capture any
  * thrown exceptions */
 List roleList = principal.getRolesInOrganization(
    orgId, isGetAttributes );

 /* Any roles defined ? */
 if( roleList.size( ) == 0 )
    System.out.println( "\nNo roles defined" );
 else
    System.out.println( "\nRoles Defined:" );

 /* Process the list to display roles and attributes */
 for( int i = 0; i < roleList.size( ); i++ ) {
    /* Retrieve an element from the list */
    RoleInfo roleInfo = ( RoleInfo )roleList.get( i );
    /* Retrieve and display role id */
    System.out.println( "- Role ID: " + roleInfo.getRoleId( ) );

    /* Display attributes ? */
    if( isGetAttributes ) {
       /* Retrieve calendar id */
       if( ( calendarId = roleInfo.getCalendarId( ) ) == null )
          /* Assigned 'none' if none defined for this role */
          calendarId = "None";

       /* Display calendar id */
       System.out.println(
         "  Attributes: Calendar ID=" + calendarId + "\n" );
    }
 }
 break;
 .
```

```
   .
   .
```

Determining Whether a Role Is Defined for an Organization

The following excerpt shows how to determine whether a role is defined for an organization.

```
/* Is Role in an Organization */
case 'A' :
   /* Get Role ID for role to query for */
   if( ( roleId = askQuestion( "\nEnter Role ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Organization ID to query */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* WLPI Public API Method */
   /* Is the role defined in this organization ? */
   /* NOTE: Would be nice to add code to capture any
      thrown exceptions */
   if( principal.isRoleInOrganization( roleId, orgId) )
      System.out.println( "Role defined in the organization" );
   else
      System.out.println( "Role not defined in the organization" );
break;
.
.
.
```

Configuring Roles

The following sections explain how to configure roles, including:

- Adding a Role

- Adding a User to a Role
- Getting the Users Defined for a Role
- Getting Role Information
- Setting Role Information
- Deleting a User from a Role
- Deleting a Role
- Example of Configuring Roles

Adding a Role

To add a role to the security realm, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method.

```
public void addRole(
  com.bea.wlpi.common.RoleInfo roleInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `addRole()` method parameter for which you must specify a value.

Table 9-12 addRole() Method Parameter

Parameter	Description	Valid Values
roleInfo	New role information.	A `RoleInfo` object. For information about defining the `RoleInfo` object, see "RoleInfo Object" on page B-17.

For example, the following code adds a role based on the contents of the specified `roleInfo` object. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.addRole(roleInfo);
```

For more information about the `addRole()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Adding a User to a Role

To add a user to a role, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void addUserToRole(
  java.lang.String userId,
  java.lang.String orgId,
  java.lang.String roleId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `addUserToRole()` method parameters for which you must specify values.

Table 9-13 addUserToRole() Method Parameters

Parameter	Description	Valid Values
userId	ID of the user that you want to add to the role.	String specifying a valid user ID. For information about getting a list of users, see "Getting All Users" on page 9-49.
orgId	ID of the organization associated with the user.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
roleId	ID of the role to which you want to add the user.	String specifying a valid role ID. For information about getting a list of roles, see "Getting Role Information" on page 9-34.

For example, the following code adds the `user1` user to the `role1` role in the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.addUserToRole("user1", "ORG1", "role1");
```

For more information about the `addUserToRole()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting the Users Defined for a Role

To get a list of users defined for a role, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.util.List getUsersInRole(
  java.lang.String roleId,
  java.lang.String orgId,
  boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getUsersInRole()` method parameters for which you must specify values.

Table 9-14 getUsersInRole() Method Parameters

Parameter	Description	Valid Values
roleId	ID of the role for which you want to get users.	String specifying a valid role ID. For information about getting a list of all role ides, see "Getting the Roles Defined for an Organization" on page 9-10.
orgId	ID of the organization associated with the role.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
obtainAttributes	Flag specifying whether you want to get all attributes or only the user IDs.	`true` (all attributes) or `false` (user IDs only).

This method returns a list of `com.bea.wlpi.common.UserInfo` objects. To access information about each user, use the `UserInfo` object methods described in "UserInfo Object" on page B-26.

For example, the following code gets a list of users defined for the `role1` role in the `ORG1` organization, returning all attributes (as the `obtainAttributes` parameter is set to `true`). In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
List users = principal.getUsersInRole("role1", "ORG1", true);
```

For more information about the `getUsersInRole()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting Role Information

To get information about a role, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public com.bea.wlpi.common.RoleInfo getRoleInfo(
  java.lang.String roleId,
  java.lang.String orgId
) throws java.rmi.RemoteException
```

The following table describes the `getRoleInfo()` method parameters for which you must specify values.

Table 9-15 getRoleInfo() Method Parameters

Parameter	Description	Valid Values
roleId	ID of the role for which you want to get information.	String specifying a valid role ID. For information about getting a list of users, see "Getting the Roles Defined for an Organization" on page 9-10.
orgId	ID of the organization associated with the role.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

This method returns a `com.bea.wlpi.common.RoleInfo` object. To access information about the role, use the `RoleInfo` object methods described in "RoleInfo Object" on page B-17.

For example, the following code gets information about the `role1` role in the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.addUserToRole("role1", "ORG1");
```

For more information about the `getRoleInfo()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Setting Role Information

To set information about a role, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method.

```
public void setRoleInfo(
  com.bea.wlpi.common.RoleInfo roleInfo
) throws java.rmi.RemoteException
```

The following table describes the `setRoleInfo()` method parameter for which you must specify a value.

Table 9-16 setRoleInfo() Method Parameter

Parameter	Description	Valid Values
roleInfo	Role information to be updated.	A `RoleInfo` object. For information about getting a list of all `RoleInfo` objects, see "Getting a List of Roles Defined for an Organization" on page 9-10. (Be sure to set the Boolean parameter, `obtainAttributes`, to `true` to avoid inadvertently clearing any organization attributes.) For information about updating a `RoleInfo` object, see "RoleInfo Object" on page B-17.

For example, the following code sets information about a role based on the contents of the specified `roleInfo` object. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.setRoleInfo(roleInfo);
```

For more information about the `setRoleInfo()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Deleting a User from a Role

To delete a user from a role, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void removeUserFromRole(
  java.lang.String userId,
  java.lang.String orgId,
  java.lang.String roleId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `removeUserFromRole()` method parameters for which you must specify values.

Table 9-17 removeUserFromRole() Method Parameters

Parameter	Description	Valid Values
userId	ID of the user that you want to delete.	String specifying a valid user ID. For information about getting a list of users, see "Getting All Users" on page 9-49.
orgId	ID of the organization associated with the user.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
roleId	ID of the role from which you want to delete the user.	String specifying a valid role ID. For information about getting a list of all role IDs, see "Getting the Roles Defined for an Organization" on page 9-10.

For example, the following code removes the `user1` user from the `role1` role within the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.removeUserFromOrganization("user1", "ORG1", "role1");
```

For more information about the `removeUserFromRole()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Deleting a Role

To delete a role, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void deleteRole(
  java.lang.String orgId,
  java.lang.String roleId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteRole()` method parameters for which you must specify values.

Table 9-18 deleteRole() Method Parameters

Parameter	Description	Valid Values
orgId	ID of the organization associated with the role to be deleted.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
roleId	ID of the role that you want to delete.	String specifying a valid role ID. For information about getting a list of all role IDs, see "Getting the Roles Defined for an Organization" on page 9-10.

For example, the following code deletes the role corresponding to the specified role ID in the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.deleteRole(role.getId(), "ORG1")
```

The role ID is obtained using the methods associated with the `com.bea.wlpi.common.RoleInfo` object, `role`. The `role` object can be obtained using the methods described in "Getting a List of Roles Defined for an Organization" on page 9-10.

For more information about the `deleteRole()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Example of Configuring Roles

This section provides excerpts from the command-line administration example showing how to configure roles.

Note: For more information about the command-line administration example, see "Command-Line Administration Example" on page 1-21.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Adding a Role
- Adding a User to a Role
- Deleting a Role
- Deleting a User from a Role
- Setting Role Information
- Getting Role Information
- Getting Users Defined for a Role

Important lines of code are highlighted in **bold**. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) )

/* Display Tool Title */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---             Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
```

```
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
public static void mngRoles( ) {
   String answer;
   String calendarId;
   String orgId;
   String roleId;
   String userId;
   String eMail;
   String defaultOrgId;

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* WLPI Public API Method */
      boolean isRealmManageable = principal.isManageableSecurityRealm( );

      /* Display the menu and interact with user */
      while( true ) {
      /* Display the menu */
         System.out.println( "\n\n---            WLPI Roles            ---" );
         System.out.println( "\nEnter choice:" );
         /* Is the realm manageable ? */
         if( isRealmManageable ) {
            /* The realm is manageable realm; Display menu options that
             * requires a manageable realm */
            System.out.println( "1) Add a new Role" );
            System.out.println( "2) Assign a User to a Role" );
            System.out.println( "3) Delete a Role" );
            System.out.println( "4) Remove a User from a Role" );
            System.out.println( "5) Update Role Info" );
         }
         System.out.println( "6) List Role Info" );
         System.out.println( "7) List Users in a Role" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* Get user's selection */
         String line = stdIn.readLine( );

         /* User pressed enter without making a selection ? */
         if( line.equals( "" ) )
```

```
   continue;
/* User entered more than one char ? */
else if( line.length( ) > 1 ) {
   System.out.println( "*** Invalid selection" );
   continue;
}
/* Realm is not manageable and user entered a hidden selection ? */
else if( !isRealmManageable && line.charAt( 0 ) < '6' ) {
   System.out.println( "*** Invalid selection" );
   continue;
}

/* Convert to uppercase and to char */
char choice = line.toUpperCase( ).charAt( 0 );

/* Process user's selection */
switch( choice ) {
   .
   .
   .
```

Adding a Role

The following excerpt shows how to add a role:

```
/* Add a new Role */
case '1' :
   /* Get Role ID for new role to add */
   if( ( roleId = askQuestion(
      "\nEnter a new Role ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Organization ID where to add new user; required */
   if( ( orgId = askQuestion(
      "Enter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Calendar ID to assign to this role (optional) */
   calendarId = askQuestion(
      "Enter Calendar ID (press enter for none)" );
```

```
   /* Create a RoleInfo object; required to add a new role */
   RoleInfo roleInfo = new RoleInfo( roleId, orgId, calendarId );

   try {
      /* WLPI Public API Method */
      /* Add the new role to the organization */
      principal.addRole( roleInfo );

      /* Success (No exception trown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to add role\n" );
      System.err.println( e );
   }
    break;
   .
   .
   .
```

Adding a User to a Role

The following excerpt shows how to add a user to a role:

```
/* Assign a User to a Role */
case '2' :
   /* Get User ID for user to assign to role */
   if( ( userId = askQuestion(
      "\nEnter User ID to assign" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Role ID where to add user */
   if( ( roleId = askQuestion( "Enter Role ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

/* Get Organization ID where role belongs */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }
```

```
   try {
      /* WLPI Public API Method */
      /* Add user to this role within this organization */
      principal.addUserToRole( userId, orgId, roleId );

      /* Success (No exception thrown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to add user to role\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Deleting a Role

The following excerpt shows how to delete a role:

```
/* Delete a Role */
case '3' :
   /* Get Role ID for role to be removed */
   if( ( roleId = askQuestion(
      "\nEnter Role ID to delete" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Organization ID where role is to be removed */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Remove role from this organization */
      principal.deleteRole( orgId, roleId );

      /* Success (No exception trown) */
      System.out.println( "- Success" );
   }
```

```
   catch( Exception e ) {
      System.out.println( "*** Unable to delete role\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Deleting a User from a Role

The following excerpt shows how to delete a user from a role:

```
/* Remove a User from a Role */
case '4' :
   /* Get User ID for user to be removed from role */
   if( ( userId = askQuestion(
      "\nEnter User ID to remove" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Role ID where to remove user */
   if( ( roleId = askQuestion( "Enter Role ID" ) ) == null ) {
   /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Organization ID where role belongs */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Remove user from this role in this organization */
      principal.removeUserFromRole( userId, orgId, roleId);

      /* Success (No exception trown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to remove user from role\n" );
```

```
            System.err.println( e );
         }
         break;
         .
         .
         .
```

Setting Role Information

The following excerpt shows how to set role information:

```
/* Update Role Info */
case '5' :
   /* Get Role ID for role to be updated */
   if( ( roleId = askQuestion(
      "\nEnter Role ID to update" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Organization ID where role belongs */
   if( ( orgId = askQuestion(
      "Enter Organization ID" ) ) == null ) {
   /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get new Calendar ID to assign to this role; only role attribute
    * defined in WLPI v1.2.1 thus only attribute that can be updated */
   calendarId = askQuestion(
      "Enter Calendar ID (press enter for none)" );

   /* Create a RoleInfo object; required to update a role */
   roleInfo = new RoleInfo( roleId, orgId, calendarId );

   try {
      /* WLPI Public API Method */
      /* Update this role (read calendar) in this organization */
      principal.setRoleInfo( roleInfo );

      /* Success (No exception trown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to update role info\n" );
```

```
            System.err.println( e );
         }
         break;
         .
         .
         .
```

Getting Role Information

The following excerpt shows how to get information about a role:

```
/* List Role Info */
case '6' :
   /* Get Role ID for the role to display */
   if( ( roleId = askQuestion( "\nEnter Role ID" ) ) == null ) {
   /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Organization ID where role belongs */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
   /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Retrieve info on this role */
      roleInfo = principal.getRoleInfo( roleId, orgId );

      System.out.println( "\nRole Info:" );
      /* Retrieve and display role id */
      System.out.println( "- Role ID: " + roleInfo.getRoleId( ) );
      /* Retrieve and display org id where role belongs */
      System.out.println( "  Org ID: " + roleInfo.getOrgId( ) );

      /* Retrieve calendar id */
      if( ( calendarId = roleInfo.getCalendarId( ) ) == null )
      /* Assigned 'none' if none defined for this role */
         calendarId = "None";

      /* Display calendar id */
      System.out.println( "  Calendar ID: " + calendarId );
   }
   catch( Exception e ) {
```

```
            System.out.println( "*** Unable to retrieve role info\n" );
            System.err.println( e );
         }
         break;
         .
         .
         .
```

Getting Users Defined for a Role

The following excerpt shows how to get a list of all users defined for a role:

```
/* List Users in a Role */
case '7' :
   /* Get Role ID for role to query */
   if( ( roleId = askQuestion( "\nEnter Role ID" ) ) == null ) {
   /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Organization ID where role belongs */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Prompt user to select if we need to display
    * the user attributes */
   if( ( answer = askQuestion(
      "List all attributes (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Parse the answer */
   boolean isGetAttributes = ( answer.equals( "y" ) ||
      answer.equals( "Y" ) );

   /* WLPI Public API Method */
   /* Retrieve all users assigned to this role */
   /* NOTE: Would be nice to add code to capture
    * any thrown exceptions */
   List userList =
      principal.getUsersInRole( roleId, orgId, isGetAttributes );
```

```
/* Any users assigned ? */
if( userList.size( ) == 0 )
   System.out.println( "\nNo users assigned" );
else
   System.out.println( "\nAssigned Users:" );

/* Process the list to display users and attributes */
for( int i = 0; i < userList.size( ); i++ ) {
   /* Retrieve an element from the list */
   UserInfo userInfo = ( UserInfo )userList.get( i );
   /* Retrieve and display user id */
   System.out.println( "- User ID: " + userInfo.getUserId( ) );

   /* Display attributes ? */
   if( isGetAttributes ) {
      if( ( eMail = userInfo.getEMailAddress( ) ) == null )
         /* Assigned 'none' if none defined for this user */
         eMail = "None";

      if( ( defaultOrgId = userInfo.getDefaultOrgId( ) ) == null )
      /* Assigned 'none' if none defined for this user */
         defaultOrgId = "None";

      /* Retrieve calendar id */
      if( ( calendarId = userInfo.getCalendarId( ) ) == null )
      /* Assigned 'none' if none defined for this user */
         calendarId = "None";

      /* Display email address, default ord id and calendar id */
      System.out.println( "  Attributes:\n  - eMail: " + eMail );
      System.out.println( "  - Default ORG ID: " + defaultOrgId );
      System.out.println( "  - Calendar ID=" + calendarId + "\n" );
   }
}
break;
.
.
.
```

Configuring Users

The following sections explain how to configure users:

- Adding a User
- Getting All Users
- Getting User Organizations
- Getting User Roles
- Getting User Information
- Setting User Information
- Deleting a User
- Example of Configuring Users

Adding a User

To create a new user and add it to the `wlpiUsers` group, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public com.bea.wlpi.common.UserInfo createUser(
  java.lang.String userId,
  java.lang.String pswd
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `createUser()` method parameters for which you must specify values.

Table 9-19 createUser() Method Parameters

Parameter	Description	Valid Values
userId	ID of the user that you want to add.	String specifying a unique user ID.
pswd	Clear-text password for the specified user ID.	String specifying a password.

For example, the following code creates a user named `sam` with the password `password` within the `wlpiUsers` group. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
UserInfo userInfo = principal.createUser("sam", "password")
```

For more information about the `createUser()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting All Users

To obtain a list of all users, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.util.List getAllUsers(
   boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getAllUsers()` method parameter for which you must specify a value.

Table 9-20 getAllUsers() Method Parameter

Parameter	Description	Valid Values
obtainAttributes	Flag specifying whether you want to get all attributes or only the user IDs.	`true` (all attributes) or `false` (user IDs only).

This method returns a list of `com.bea.wlpi.common.UserInfo` objects. To access information about each user, use the `UserInfo` object methods described in "UserInfo Object" on page B-26.

For example, the following code gets the user IDs (the `obtainAttributes` parameter is set to false) and saves them to the list variable, `userList`. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
List userList = principal.getAllUsers(false)
```

For more information about the `getAllUsers()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting User Organizations

To get a list of the organizations to which a user belongs, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.util.List getOrganizationsForUser(
  java.lang.String userId,
  boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getOrganizationsForUser()` method parameters for which you must specify values.

Table 9-21 getOrganizationsForUser() Method Parameters

Parameter	Description	Valid Values
userId	ID of the user for which you want to get organizations.	String specifying a valid user ID. For information about getting a list of all organization IDs, see "Getting All Users" on page 9-49.
obtainAttributes	Flag specifying whether you want to get all attributes or only the organization IDs.	`true` (all attributes) or `false` (organization IDs only).

This method returns a list of `com.bea.wlpi.common.OrganizationInfo` objects. To access information about each organization, use the `OrganizationInfo` object methods described in "OrganizationInfo Object" on page B-9.

For example, the following code gets a list of organizations for `user1`, returning all attributes (as the `obtainAttributes` parameter is set to `true`). In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
List roles = principal.getOrganizationsForUser("user1", true);
```

For more information about the `getOrganizationsForUser()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting User Roles

To get a list of roles to which a user belongs, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.util.List getRolesForUser(
  java.lang.String orgId,
  java.lang.String userId,
  boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getRolesForUser()` method parameters for which you must specify values.

Table 9-22 getRolesForUser() Method Parameters

Parameter	Description	Valid Values
orgId	ID of the organization associated with the user.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
userId	ID of the user for which you want to get a list of roles.	String specifying a valid user ID. For information about getting a list of all organization IDs, see "Getting All Users" on page 9-49.
obtainAttributes	Flag specifying whether you want to get all attributes or only the role IDs.	`true` (all attributes) or `false` (organization IDs only).

This method returns a list of `com.bea.wlpi.common.RoleInfo` objects. To access information about each role, use the `RoleInfo` object methods described in "RoleInfo Object" on page B-17.

For example, the following code gets a list of roles for `user1` in the `ORG1` organization, returning all attributes (as `obtainAttributes` parameter is set to `true`). In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
List roles = principal.getRolesForUser("ORG1", "user1", true);
```

For more information about the `getRolesForUser()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting User Information

To get information about the user, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public com.bea.wlpi.common.UserInfo getUserInfo(
  java.lang.String userId
) throws java.rmi.RemoteException
```

The following table describes the `getUserInfo()` method parameter for which you must specify a value.

Table 9-23 getUserInfo() Method Parameter

Parameter	Description	Valid Values
userId	ID of the user for which you want to get information.	String specifying a valid user ID. For information about getting a list of users, see "Getting All Users" on page 9-49.

This method returns a `com.bea.wlpi.common.UserInfo` object. To access information about the user, use the `UserInfo` object methods described in "UserInfo Object" on page B-26.

For example, the following code gets information for the `user1` user. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
UserInfo user = principal.getUserInfo("user1");
```

For more information about the `getUserInfo()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Setting User Information

To set information about the user, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void setUserInfo(
  com.bea.wlpi.common.UserInfo userInfo
) throws java.rmi.RemoteException,
```

The following table describes the `setUserInfo()` method parameter for which you must specify a value.

Table 9-24 setUserInfo() Method Parameter

Parameter	Description	Valid Values
userInfo	User information to be updated.	A `UserInfo` object. For information about getting a list of all `UserInfo` objects, see "Getting a List of Users Defined for an Organization" on page 9-12. (Be sure to set the Boolean parameter, `obtainAttributes`, to `true` to avoid inadvertently clearing any organization attributes.) For information about updating a `UserInfo` object, see "UserInfo Object" on page B-26.

For example, the following code sets information about a user based on the contents of the specified `userInfo` object. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.setUserInfo(userInfo);
```

For more information about the `setUserInfo()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Deleting a User

You can delete users from an organization, from a role, or from the configuration database using the methods described in this section.

Deleting a User from an Organization

For information about deleting a user from an organization, see "Deleting a User from an Organization" on page 9-16, within the section "Configuring Organizations."

Deleting a User from a Role

For information about deleting a user from a role, see "Deleting a User from a Role" on page 9-36, within the section "Configuring Roles."

Deleting a User from the Database

To delete a user from the configuration database, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void deleteUser(
  java.lang.String userId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteUser()` method parameter for which you must specify a value.

Table 9-25 deleteUser() Method Parameter

Parameter	Description	Valid Values
userId	ID of the user that you want to delete.	String specifying a valid user ID. For information about getting a list of users, see "Getting All Users" on page 9-49.

For example, the following code removes the `user1` user from the database. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.deleteUser("user1");
```

For more information about the `deleteUser()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Example of Configuring Users

This section provides excerpts from the command-line administration example showing how to configure users.

Note: For more information about the command-line administration example, see "Command-Line Administration Example" on page 1-21.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Adding a User
- Deleting a User
- Setting User Information
- Getting All Users
- Getting User Information
- Getting User Organizations
- Getting User Roles

Important lines of code are highlighted in **bold**. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) )

/* Display Tool Title */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---             Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
```

```
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .

   System.out.print( ">> " ); ...
```

The `mngUsers()` method illustrates how to manage users, interacting with the user to retrieve the information required:

```
public static void mngUsers( ) {
   String answer;
   String calendarId;
   String orgId;
   String userId;
   String password;
   String eMail;
   String defaultOrgId;

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* WLPI Public API Method */
      boolean isRealmManageable = principal.isManageableSecurityRealm( );

      /* Display the menu and interact with user */
      while( true ) {
         /* Display the menu */
         System.out.println( "\n\n---            WLPI Users            ---" );
         System.out.println( "\nEnter choice:" );
         /* Is the realm manageable ? */
         if( isRealmManageable ) {
            /* The realm is manageable realm; Display menu options that
             * requires a manageable realm */
            System.out.println( "1) Add a new User" );
            System.out.println( "2) Delete a User" );
            System.out.println( "3) Update User Info" );
         }
         System.out.println( "4) List All Users" );
         System.out.println( "5) List User Info" );
         System.out.println( "6) List Organizations for a User" );
         System.out.println( "7) List Roles for a User" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );
```

```
/* Get user's selection */
String line = stdIn.readLine( );

/* User pressed enter without making a selection ? */
if( line.equals( "" ) )
   continue;
/* User entered more than one char ? */
else if( line.length( ) > 1 ) {
   System.out.println( "*** Invalid selection" );
   continue;
}
/* Realm is not manageable and user entered a hidden selection ? */
else if( !isRealmManageable && line.charAt( 0 ) < '4' ) {
   System.out.println( "*** Invalid selection" );
   continue;
}

/* Convert to uppercase and to char */
char choice = line.toUpperCase( ).charAt( 0 );

/* Process user's selection */
switch( choice ) {
   .
   .
   .
```

Adding a User

The following excerpt shows how to add a user:

```
/* Add a new User */
case '1' :
   /* Get User ID for new user to create */
   if( ( userId = askQuestion( "\nEnter new User ID" ) ) == null ) {
   /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get new user's Password */
   if( ( password = askQuestion( "Enter Password" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }
```

```
   try {
      /* WLPI Public API Method */
      /* Create the new user id using this password */
      principal.createUser( userId, password );

      /* Success (No exception thrown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to create user\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Deleting a User

The following excerpt shows how to delete a user:

```
/* Delete a User */
case '2' :
   /* Get User ID for user to remove */
   if( ( userId = askQuestion( "\nEnter User ID to delete" ) )
      == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Remove/delete user */
      principal.deleteUser( userId );

      /* Success (No exception trown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete user\n" );
      System.err.println( e );
   }
```

```
   break;
   .
   .
   .
```

Setting User Information

The following excerpt shows how to set information about a user:

```
/* Update User Info */
case '3' :
   /* Get User ID for user to update */
   if( ( userId = askQuestion( "\nEnter User ID to update" ) )
      == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get eMail address for this user (optional) */
   eMail = askQuestion(
      "Enter new eMail Address (press enter for none)" );

   /* Get Default Org ID to assign to this user (optional) */
   defaultOrgId = askQuestion(
      "Enter new default Org ID (press enter for none)" );

   /* Get Calendar ID to assign to this user (optional) */
   calendarId = askQuestion(
      "Enter new Calendar ID (press enter for none)" );

   /* Create an UserInfo object; required to update a user */
   UserInfo userInfo = new UserInfo(
      userId, eMail, defaultOrgId, calendarId );

   try {
      /* WLPI Public API Method */
      principal.setUserInfo( userInfo );

      /* Success (No exception trown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to update user\n" );
      System.err.println( e );
   }
```

```
      break;
      .
      .
      .
```

Getting All Users

The following excerpt shows how to get a list of all users:

```
/* List All Users */
case '4' :
   /* Prompt user to select if we need to display the
    * user attributes */
   if( ( answer = askQuestion(
      "\nList all attributes (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Parse the answer */
   boolean isGetAttributes = ( answer.equals( "y" ) ||
      answer.equals( "Y" ) );

   /* WLPI Public API Method */
   /* Retrieve all users */
   /* NOTE: Would be nice to add code to capture any
    * thrown exceptions */
   List userList = principal.getAllUsers( isGetAttributes );

   /* Any users defined ? */
   if( userList.size( ) == 0 )
      System.out.println( "\nNo user defined" );
   else
      System.out.println( "\nDefined Users:" );

   /* Process the list to display users and attributes */
   for( int i = 0; i < userList.size( ); i++ ) {
      /* Retrieve an element from the list */
      userInfo = ( UserInfo )userList.get( i );
      /* Retrieve and display user id */
      System.out.println( "- User ID: " + userInfo.getUserId( ) );

      /* Display attributes ? */
      if( isGetAttributes ) {
         /* Retrieve eMail address */
         if( ( eMail = userInfo.getEMailAddress( ) ) == null )
```

```
            /* Assigned 'none' if none defined for this user */
            eMail = "None";

         /* Retrieve default organization id */
         if( ( defaultOrgId = userInfo.getDefaultOrgId( ) ) == null )
            /* Assigned 'none' if none defined for this user */
            defaultOrgId = "None";

         /* Retrieve calendar id */
         if( ( calendarId = userInfo.getCalendarId( ) ) == null )
            /* Assigned 'none' if none defined for this user */
            calendarId = "None";

         /* Display email address, default ord id and calendar id */
         System.out.println( "  Attributes:\n  - eMail: " + eMail );
         System.out.println( "  - Default ORG ID: " + defaultOrgId );
         System.out.println( "  - Calendar ID=" + calendarId + "\n" );
      }
   }
   break;
   .
   .
   .
```

Getting User Information

The following excerpt shows how to get information about a user:

```
/* List User Info */
case '5' :
   /* Get User ID for the user to display */
   if( ( userId = askQuestion( "\nEnter User ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Retrieve info on this user */
      userInfo = principal.getUserInfo( userId );

      /* Retrieve and display user id */
      System.out.println( "- User ID: " + userInfo.getUserId( ) );

      /* Retrieve eMail address */
      if( ( eMail = userInfo.getEMailAddress( ) ) == null )
```

```
            /* Assigned 'none' if none defined for this user */
            eMail = "None";

         /* Retrieve default organization id */
         if( ( defaultOrgId = userInfo.getDefaultOrgId( ) ) == null )
            /* Assigned 'none' if none defined for this user */
            defaultOrgId = "None";

         /* Retrieve calendar id */
         if( ( calendarId = userInfo.getCalendarId( ) ) == null )
            /* Assigned 'none' if none defined for this user */
            calendarId = "None";

         /* Display email address, default ord id and calendar id */
         System.out.println( "  Attributes:\n  - eMail: " + eMail );
         System.out.println( "  - Default ORG ID: " + defaultOrgId );
         System.out.println( "  - Calendar ID=" + calendarId + "\n" );
      }
      catch( Exception e ) {
         System.out.println( "*** Unable to retrieve user info\n" );
         System.err.println( e );
      }
      break;
      .
      .
      .
```

Getting User Organizations

The following excerpt shows how to get a list of the user organizations:

```
/* List Organizations for a User */
case '6' :
   /* Get User ID for user to query for */
   if( ( userId = askQuestion( "\nEnter User ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Prompt user to select if we need to display the
    * organization attributes */
   if( ( answer = askQuestion(
      "List Organization attributes (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
```

```
}

/* Parse the answer */
isGetAttributes = ( answer.equals( "y" ) || answer.equals( "Y" ) );

/* WLPI Public API Method */
/* Retrieve all organizations where this user is assigned */
/* NOTE: Would be nice to add code to capture any
 * thrown exceptions */
List orgList = principal.getOrganizationsForUser(
   userId, isGetAttributes );

/* Assigned to any organizations ? */
if( orgList.size( ) == 0 )
   System.out.println( "\nNot assigned to any organization" );
else
   System.out.println( "\nAssigned to organizations:" );

/* Process the list to display organizations and attributes */
for( int i = 0; i < orgList.size( ); i++ ) {
   /* Retrieve an element from the list */
   OrganizationInfo orgInfo = ( OrganizationInfo )orgList.get( i );
   /* Retrieve and display organization id */
   System.out.println( "- Org ID: " + orgInfo.getOrgId( ) );

   /* Display attributes ? */
   if( isGetAttributes ) {
      /* Retrieve calendar id */
      if( ( calendarId = orgInfo.getCalendarId( ) ) == null )
         /* Assigned 'none' if none defined for this organization */
         calendarId = "None";

      /* Display calendar id */
      System.out.println(
         "  Attributes: Calendar ID=" + calendarId + "\n" );
   }
}
break;
.
.
.
```

Getting User Roles

The following excerpt shows how to get all roles to which a user is assigned:

```
         /* List Roles for a User */
         case '7' :
            /* Get User ID for user to query for */
            if( ( userId = askQuestion( "\nEnter User ID" ) ) == null ) {
               /* User cancelled the operation */
               System.out.println( "*** Cancelled" );
               break;
            }

            /* Get Organization ID to query */
            if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
               /* User cancelled the operation */
               System.out.println( "*** Cancelled" );
               break;
            }

//          /* Prompt user to select if we need to display the role attributes */
//          if( ( answer = askQuestion(
//             "List Role attributes (y/n)?" ) ) == null ) {
//             /* User cancelled the operation */
//             System.out.println( "*** Cancelled" );
//             break;
//          }
//
//          /* Parse the answer */
//          isGetAttributes = ( answer.equals( "y" ) ||
//             answer.equals( "Y" ) );
            isGetAttributes = false;

            /* WLPI Public API Method */
            /* Retrieve all roles where this user is assigned */
            /* NOTE: Would be nice to add code to capture any
             * thrown exceptions */
            List roleList = principal.getRolesForUser(
               orgId, userId, isGetAttributes );

            /* Assigned to any role ? */
            if( roleList.size( ) == 0 )
               System.out.println( "\nNot assigned to any role" );
            else
               System.out.println( "\nAssigned to roles:" );

            /* Process the list to display roles and attributes */
            for( int i = 0; i < roleList.size( ); i++ ) {
               /* Retrieve an element from the list */
               RoleInfo roleInfo = ( RoleInfo )roleList.get( i );
               /* Retrieve and display role id */
               System.out.println( "- Role ID: " + roleInfo.getRoleId( ) );
```

```
               /* Display attributes ? */
               if( isGetAttributes ) {
                  /* Retrieve calendar id */
                  if( ( calendarId = roleInfo.getCalendarId( ) ) == null )
                     /* Assigned 'none' if none defined for this role */
                     calendarId = "None";

                  /* Display calendar id */
                  System.out.println(
                     "  Attributes: Calendar ID=" + calendarId + "\n" );
               }
            }
            break;

         /* Return to previous menu */
         case 'B' :
            return;

         /* Exit tool */
         case 'Q' :
            /* Disconnect from the server */
            disconnect( );
            System.exit( 1 );

            default:
            System.out.println( "*** Invalid selection" );
         }
      }
   }
   /* "Unhandled" exceptions */
   catch( Exception e ) {
      System.err.println( e );
   }
   return;
}
```

Mapping Security Information

Once you have defined the users and roles, you must define the relationship between these users and roles and the users and groups defined for BEA WebLogic Server, respectively, by mapping roles to the BEA WebLogic Server security realms.

This section explains how to perform the tasks required for such mapping:

- Getting the Security Realm Groups
- Mapping a Role to a Group
- Mapping Multiple Roles to Groups
- Getting the Group Mapping for a Role
- Getting the Group Mappings for All Roles Defined for an Organization

Getting the Security Realm Groups

To get a list of BEA WebLogic Server security realm groups, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.util.List getGroups(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

This method returns a list of security realm group names.

For example, the following code gets the security realm group names and saves them to the `groups` list. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
List groups = principal.getGroups();
```

For more information about the `getGroups()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Mapping a Role to a Group

To map a BPM role to a BEA WebLogic Server security realm group, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void mapRoleToGroup(
  java.lang.String roleId,
  java.lang.String orgId,
  java.lang.String groupId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `mapRoleToGroup()` method parameters for which you must specify values.

Table 9-26 mapRoleToGroup() Method Parameters

Parameter	Description	Valid Values
roleId	ID of the role that you want to map.	String specifying a valid role ID. For information about getting a list of all role IDs, see "Getting the Roles Defined for an Organization" on page 9-10.
orgId	ID of the organization associated with the role.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
groupId	ID of the security realm group to which you want to map the role.	String specifying a new or existing group ID. If the specified security realm group does not exist, this method creates it. For information about getting a list of group IDs, see "Getting the Security Realm Groups" on page 9-66.

For example, the following code maps the `role1` role in the `ORG1` organization to the `admin` security realm. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.mapRoleToGroup("role1", "ORG1", "admin");
```

For more information about the `mapRoleToGroup()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Mapping Multiple Roles to Groups

To map multiple BPM roles to BEA WebLogic Server security realm groups, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void mapRolesToGroups(
  java.lang.String orgId,
  java.util.Map rolesToGroupMap
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `mapRoleToGroup()` method parameters for which you must specify values.

Table 9-27 mapRoleToGroup() Method Parameters

Parameter	Description	Valid Values
orgId	ID of the organization associated with the roles.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
rolesToGroupMap	Map object specifying the role and group relationships.	Map object with *key-value* pairs, specifying the role ID as the key and the group ID as the value. If the specified security realm group does not exist, this method creates it.

For example, the following code maps roles to groups as defined in the `map1` map for the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.mapRolesToGroups("ORG1", "map1");
```

For more information about the `mapRolesToGroups()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting the Group Mapping for a Role

To get the name of the group to which a role is mapped, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.lang.String getMappedGroup(
  java.lang.String roleId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getMappedGroup()` method parameters for which you must specify values.

Table 9-28 getMappedGroup() Method Parameters

Parameter	Description	Valid Values
roleId	ID of the role for which you want to get the mapped group.	String specifying a valid role ID. For information about getting a list of all organization IDs, see "Getting the Roles Defined for an Organization" on page 9-10.
orgId	ID of the organization associated with the role.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

This method returns the security realm group to which the specified role is mapped, or null if no mapping exists.

For example, the following code returns the mapped group associated with the `role1` role in the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
principal.getMappedGroups("role1", "ORG1");
```

For more information about the `getMappedGroups()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Getting the Group Mappings for All Roles Defined for an Organization

To get a list of the groups to which all roles defined for an organization are mapped, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public java.util.Map getRoleMappingsInOrg(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getRoleMappingsInOrg()` method parameter for which you must specify a value.

Table 9-29 getRoleMappingsInOrg() Method Parameters

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to get all role-to-group mappings.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

This method returns a map of *key-value* pairs in which the role ID is specified as the key, and the group ID, as the value.

For example, the following code returns the role-to-group map for the `ORG1` organization. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
Map map = principal.getRoleMappingsInOrg("ORG1");
```

For more information about the `getRoleMappingsInOrg()` method, see the `com.bea.wlpi.server.principal.WLPIPrincipal` Javadoc.

Configuring Permissions

You can configure permissions for both roles and users as a means of protecting access to specific BPM functions.

This section describes the permissions and explains how to perform the tasks associated with configuring permissions:

- Permissions Overview
- Getting Permissions for All Roles
- Getting Permissions for a Role
- Getting Permissions for All Users
- Getting the Permissions for a User
- Determining Whether a Specific Permission Is Set
- Setting Role-Specific Permissions
- Setting User-Specific Permissions

Permissions Overview

The following table describes the permissions that can be set for each role and/or user, and the associated `com.bea.wlpi.common.security.EnumPermission` static value that can be used when setting permissions. For information about setting permissions for particular roles and users, see "Setting Role-Specific Permissions" on page 9-77 and "Setting User-Specific Permissions" on page 9-79, respectively.

Table 9-30 Permissions

Permission	Description	EnumPermission Static Value
Configure System	Update application configuration, for example, by adding, updating, or deleting business calendars.	`P_Configure_System`

Table 9-30 Permissions (Continued)

Permission	Description	EnumPermission Static Value
Configure Components	■ Define, update, and delete business operations. ■ Load and configure plug-ins.	`P_Configure_Components`
Administer User	■ Manage organizations, roles, and users. ■ Specify levels of permission for roles and users. ■ Update task routes.	`P_Admininister_User`
Monitor Instance	Monitor (but not edit) instances, business calendars, workload reports, and statistics reports.	`P_Monitor_Instance`
Create Template	Create templates.	`P_Create_Template`
Delete Template	Delete templates.	`P_Delete_Template`
Execute Template	Execute templates.	`P_Execute_Template`

For more information about permissions, see the `com.bea.wlpi.common.security.EnumPermission` Javadoc.

Getting Permissions for All Roles

To get all role permissions, use the following `com.bea.wlpi.server.permission.Permission` method:

```
public java.util.List getAllRolePermissions(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Note: Execution of the `getAllRolePermissions()` method may require a substantial amount of resources, and should not be executed during run-time operations.

This method returns a list of `com.bea.wlpi.common.security.RolePermissionInfo` objects. To access information about the permissions for a role, use the `RolePermissionInfo` object methods described in "RolePermissionInfo Object" on page B-18.

For example, the following code returns all role permissions. In this example, `principal` represents the `EJBObject` reference to the `WLPIPrincipal` EJB.

```
List rolePermissions = principal.getAllRolePermissions();
```

For more information about the `getAllRolePermissions()` method, see the `com.bea.wlpi.server.permission.Permission` Javadoc.

Getting Permissions for a Role

To get a list of the permissions assigned to a role, use the following `com.bea.wlpi.server.permission.Permission` method:

```
public com.bea.wlpi.common.RolePermissionInfo getRolePermissions(
  java.lang.String roleName,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getRolePermissions()` method parameters for which you must specify values.

Table 9-31 getRolePermissions() Method Parameters

Parameter	Description	Valid Values
roleName	Name of the role for which you want permissions to be returned.	String specifying a valid role name. For information about getting a list of roles, see "Getting the Roles Defined for an Organization" on page 9-10.
orgId	ID of the organization for which you want to get all role-to-group mappings.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

This method returns a com.bea.wlpi.common.security.RolePermissionInfo object. To access information about a specific role, use the RolePermissionInfo object methods described in "RolePermissionInfo Object" on page B-18.

For example, the following code returns the permissions for the role1 role in the ORG1 organization. In this example, permission represents the EJBObject reference to the Permission EJB.

```
RolePermissionInfo rolePermissions =
  permission.getRolePermissions("role1", "ORG1");
```

For more information about the getRolePermissions() method, see the com.bea.wlpi.server.permission.Permission Javadoc.

Getting Permissions for All Users

To get all user permissions, use one of the following com.bea.wlpi.server.permission.Permission methods.

Method 1

```
public java.util.List getAllUserPermissions(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public java.util.List getAllUserPermissions(
  boolean getRoles
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Note: Execution of the getAllUserPermissions() method may require a substantial amount of resources, and should not be executed during run-time operations.

The following table describes the getAllUserPermissions() method parameter for which you must specify a value.

Table 9-32 getAllUserPermissions() Method Parameter

Parameter	Description	Valid Values
getRoles	Boolean flag specifying whether or not to include the permissions that are inherited from the roles to which the user belongs.	true (role permissions are returned) or false (role permissions are not returned).

Each method returns a list of `com.bea.wlpi.common.security.UserPermissionInfo` objects. The first method returns the permissions that are inherited from the roles to which the user is assigned, by default. The second method determines whether or not to return the inherited role permissions based on the value of the `getRoles` Boolean flag value. To access the information about user-specific permissions, use the `UserPermissionInfo` object methods described in "UserPermissionInfo Object" on page B-28.

For example, the following code returns all the permissions set for a user (and the permissions that are inherited from the roles to which the user is assigned by default). In this example, `permission` represents the `EJBObject` reference to the `Permission` EJB.

```
List userPermissions = principal.getAllUserPermissions();
```

For more information about the `getAllUserPermissions()` method, see the `com.bea.wlpi.server.permission.Permission` Javadoc.

Getting the Permissions for a User

To get a list of permissions associated with a specific user, use one of the following `com.bea.wlpi.server.permission.Permission` methods.

Method 1

```
public com.bea.wlpi.common.UserPermissionInfo getUserPermissions(
  java.lang.String userName
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public com.bea.wlpi.common.UserPermissionInfo getUserPermissions(
  java.lang.String userName,
  boolean getRoles
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getUserPermissions()` method parameters for which you must specify values.

Table 9-33 getUserPermissions() Method Parameters

Parameter	Description	Valid Values
userName	Name of the user for whom you want permissions returned.	String specifying a valid user name. For information about getting a list of users, see "Getting the Users Defined for an Organization" on page 9-12.
getRoles	Boolean flag specifying whether or not the roles to which the user is assigned are also returned.	`true` (roles are returned) or `false` (roles are not returned).

Each method returns a `com.bea.wlpi.common.security.UserPermissionInfo` object. The first method also returns a list of the roles to which the user is assigned, by default. The second method determines whether or not to return the list of roles, based on the value of the `getRoles` Boolean flag. To access information about user-specific permissions, use the `UserPermissionInfo` object methods described in "UserPermissionInfo Object" on page B-28.

For example, the following code returns all the permissions set for a user (and the roles to which the user is assigned, by default). In this example, `permission` represents the `EJBObject` reference to the `Permission` EJB.

```
List userPermissions = permission.getUserPermissions();
```

For more information about the `getUserPermissions()` method, see the `com.bea.wlpi.server.permission.Permission` Javadoc.

Determining Whether a Specific Permission Is Set

To determine whether a specific permission is set for a `PermissionInfo` object, use the following `com.bea.wlpi.common.security.PermissionInfo` method:

```
public boolean hasPermission(
  com.bea.wlpi.common.security.EnumPermission permission
)
```

The following table describes the `hasPermission()` method parameter for which you must specify a value.

Table 9-34 hasPermission() Method Parameter

Parameter	Description	Valid Values
permission	Name of the permission for which you want to check.	An `EnumPermission` object that specifies the permission for which you want to check. For more information about the permissions that can be set, see "Permissions Overview" on page 9-71.

This method returns a Boolean value indicating whether or not the specified permission exists as part of the object.

For example, the following code determines whether the specific permission is set. In this example, `permissionInfo` represents an object reference to the `PermissionInfo` class.

```
boolean hasPermission =
  permissionInfo.hasPermission(P_Administer_User);
```

For more information about the `hasPermission()` method, see the `com.bea.wlpi.server.permission.Permission` Javadoc.

Setting Role-Specific Permissions

You can set a specific permission or a group of permissions for one or more roles using the methods described in this section.

Setting Permissions for a Specific Role

To set permissions for a specific role, use the following `com.bea.wlpi.common.security.PermissionInfo` method:

```
public void setPermission(
  com.bea.wlpi.common.security.EnumPermission permission,
  boolean value
)
```

The following table describes the `setPermission()` method parameter for which you must specify a value.

Table 9-35 setPermission() Method Parameters

Parameter	Description	Valid Values
permission	Role-specific permission to be set.	Valid permission. For a list of valid permissions, see the table "Permissions" on page 9-71.

For example, the following code sets the `Administer User` permission for the `PermissionInfo` object. In this example, `permissionInfo` represents an object reference to the `PermissionInfo` class.

```
permissionInfo.setPermission(P_Administer_User);
```

For more information, see "PermissionInfo Object" on page B-10.

Setting a Group of Permissions for Multiple Roles

To set a group of permissions for one or more roles, use the following `com.bea.wlpi.server.permission.Permission` method:

```
public void setRolePermissions(
  java.util.List roleInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `setRolePermissions()` method parameter for which you must specify a value.

Table 9-36 setRolePermissions() Method Parameters

Parameter	Description	Valid Values
roleInfo	Updated information about the permissions for a role.	A list of `RolePermissionInfo` objects. For information about getting a list of all `RolePermissionInfo` objects, see "Getting Permissions for All Roles" on page 9-72. For information about defining a `RolePermissionInfo` object, see "RolePermissionInfo Object" on page B-18.

For example, the following code sets role-specific permissions according to the information defined within the `roleInfo` `RolePermissionInfo` object. In this example, `permission` represents the `EJBObject` reference to the `Permission` EJB.

```
permission.setRolePermissions(roleInfo);
```

For more information about the `setRolePermissions()` method, see the `com.bea.wlpi.server.permission.Permission` Javadoc.

Setting User-Specific Permissions

You can set a specific permission or a group of permissions for one or more users using the methods described in this section.

Setting a Single User-Specific Permission

To set a single permission for a specific user, use the following `com.bea.wlpi.common.security.PermissionInfo` method:

```
public void setPermission(
  com.bea.wlpi.common.security.EnumPermission permission,
  boolean value
)
```

The following table describes the `setPermission()` method parameter for which you must specify a value.

Table 9-37 setPermission() Method Parameters

Parameter	Description	Valid Values
permission	User-specific permission to be set.	Valid permission. For a list of valid permissions, see the table "Permissions" on page 9-71.

For example, the following code sets the `Administer User` permission for the `PermissionInfo` object. In this example, `permissionInfo` represents an object reference to the `PermissionInfo` class.

```
permissionInfo.setPermission(P_Administer_User);
```

For more information, see "PermissionInfo Object" on page B-10.

Setting a Group of Permissions for Multiple Users

To set a group of permissions for one or more users, use the following `com.bea.wlpi.server.principal.WLPIPrincipal` method:

```
public void setUserPermissions(
  java.util.List userInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `setUserPermissions()` method parameter for which you must specify a value.

Table 9-38 setUserPermissions() Method Parameters

Parameter	Description	Valid Values
userInfo	Updated information about user-specific permissions.	A list of `UserPermissionInfo` objects. For information about getting a list of all `UserPermissionInfo` objects, see "Getting Permissions for All Users" on page 9-74. For information about defining a `UserPermissionInfo` object, see "UserPermissionInfo Object" on page B-28.

For example, the following code sets the user permissions according to the information defined within the `userInfo` `UserPermissionInfo` object. In this example, `permission` represents the `EJBObject` reference to the `Permission` EJB.

```
permission.setUserPermissions(userInfo);
```

For more information about the `setUserPermissions()` method, see the `com.bea.wlpi.server.permission.Permission` Javadoc.

Part II Configuration

CHAPTER

10 Configuring Business Operations

A business operation represents a method call, implemented by an EJB or Java class instance, that allows you to create customized actions. For additional information, see "Configuring Business Operations" in "Configuring Workflow Resources" in *Using the WebLogic Integration Studio*.

This section describes the tasks associated with configuring business operations, including the following topics:

- Adding a Business Operation
- Getting Business Operations
- Updating a Business Operation
- Deleting a Business Operation
- Getting EJB Descriptors
- Getting Java Class Descriptors
- Examples of Configuring Business Operations

For more information about the methods described in this section, see the `com.bea.wlpi.server.admin.Admin` and `com.bea.wlpi.server.catalog.EJBCatalog` Javadoc.

Adding a Business Operation

To add a business operation, use one of the following `com.bea.wlpi.server.admin.Admin` methods.

Method 1

```
public java.lang.String addBusinessOperation(
  com.bea.wlpi.common.EJBInvocationDescriptor descriptor
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public java.lang.String addBusinessOperation(
  com.bea.wlpi.common.ClassInvocationDescriptor descriptor
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Each method creates a business operation. The first method creates a business operation implemented by a session or entity EJB; the second, by a Java class instance that is accessible on the server.

Note: Although session EJBs can be stored in workflow variables, stateful session EJBs are not persisted as part of a workflow instance, but exist only for the life of the transaction in which they are created. For more information about the transaction model, see "Understanding the BPM Transaction Model" on page 7-1. Entity EJBs, stateless EJBs, and Java class instances, on the other hand, are persisted as part of a workflow instance.

The following table describes the `addBusinessOperation()` method parameter for which you must specify a value.

Table 10-1 addBusinessOperation() Method Parameter

Parameter	Description	Valid Values
descriptor	Invocation descriptor for the business operation. The invocation descriptor consists of meta-data that describes the method invocation on the EJB or Java class instance that implements the business operation.	One of the following values: ■ `com.bea.wlpi.common.EJBInvocationDescriptor` when a method is being added to an EJB ■ `com.bea.wlpi.common.ClassInvocationDescriptor` when a method is being added to a Java class instance For information about getting the EJB or Java class invocation descriptor, see "Getting EJB Descriptors" on page 10-7 or "Getting Java Class Descriptors" on page 10-9, respectively. For information about defining an `EJBInvocationDescriptor` or `ClassInvocationDescriptor`, see "EJBInvocationDescriptor Object" on page C-6 or "ClassInvocationDescriptor Object" on page C-3, respectively.

Each method returns the ID of the new business operation.

For example, the following code adds the EJB business operation with the specified invocation descriptor, `descriptor`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
String operationId = admin.addBusinessOperation(descriptor);
```

For more information about the `addBusinessOperation()` methods, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting Business Operations

To get a list of defined business operations, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List getBusinessOperations(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

This method returns a list of `com.bea.wlpi.common.EJBInvocationDescriptor` or `com.bea.wlpi.common.ClassInvocationDescriptor` objects, in which the currently defined business operations are identified. To access information about each object, use the `EJBInvocationDescritor` or `ClassInvocationDescriptor` object methods described in "EJBInvocationDescriptor Object" on page C-6 or "ClassInvocationDescriptor Object" on page C-3, respectively.

For example, the following code gets the currently defined business operations. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
String operationsList = admin.getBusinessOperations();
```

For more information about the `getBusinessOperations()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Updating a Business Operation

To update a business operation, use one of the following `com.bea.wlpi.server.admin.Admin` methods.

Method 1

```
public void updateBusinessOperation(
  java.lang.String busOpId,
  com.bea.wlpi.common.EJBInvocationDescriptor descriptor
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public void updateBusinessOperation(
  java.lang.String busOpId,
  com.bea.wlpi.common.EJBInvocationDescriptor descriptor
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Each method updates a business operation. The first method updates a business operation implemented by a session or entity EJB; the second, by a Java class instance that is accessible on the server.

The following table describes the parameters for which you must specify a value for the `updateBusinessOperation()` methods.

Table 10-2 updateBusinessOperation() Method Parameters

Parameter	Description	Valid Values
busOpId	ID of the business operation to be updated.	String specifying a valid business operation ID. To get the business operation ID, use the following `com.bea.wlpi.common.EJBInvocationDescriptor` or `com.bea.wlpi.common.ClassInvocationDescriptor` method: `public java.lang.String getId()` For information about getting the EJB or Java class invocation descriptor, see "Getting EJB Descriptors" on page 10-7 or "Getting Java Class Descriptors" on page 10-9, respectively. For information about defining an `EJBInvocationDescriptor` or `ClassInvocationDescriptor`, see "EJBInvocationDescriptor Object" on page C-6 or "ClassInvocationDescriptor Object" on page C-3, respectively.
descriptor	Invocation descriptor for the business operation. The invocation descriptor consists of meta-data that describes the method invocation on the EJB or Java class instance that implements the business operation.	One of the following values: ■ `com.bea.wlpi.common.EJBInvocationDescriptor` when a method on an EJB is being updated ■ `com.bea.wlpi.common.ClassInvocationDescriptor` when a method on a Java class instance is being updated For information about getting the EJB or Java class descriptor, see "Getting EJB Descriptors" on page 10-7 or "Getting Java Class Descriptors" on page 10-9, respectively.

For example, the following code updates the EJB business operation with the specified ID and invocation descriptor, `descriptor`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.updateBusinessOperation("12345", descriptor);
```

For more information about the `updateBusinessOperation()` methods, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Deleting a Business Operation

To delete a business operation, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void deleteBusinessOperation(
  java.lang.String busOpId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteBusinessOperation()` method parameter for which you must specify a value.

Table 10-3 deleteBusinessOperation() Method Parameter

Parameter	Description	Valid Values
busOpId	ID of the business operation to delete.	String specifying a valid business operation ID. To get the business operation ID, use the following `com.bea.wlpi.common.EJBInvocationDescriptor` or `com.bea.wlpi.common.ClassInvocationDescriptor` method: `public java.lang.String getId()` For information about getting the EJB or Java class invocation descriptor, see "Getting EJB Descriptors" on page 10-7 or "Getting Java Class Descriptors" on page 10-9, respectively. For information about defining an `EJBInvocationDescriptor` or `ClassInvocationDescriptor`, see "EJBInvocationDescriptor Object" on page C-6 or "ClassInvocationDescriptor Object" on page C-3, respectively.

For example, the following code deletes the specified business operation. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.deleteBusinessOperation("124345");
```

For more information about the `deleteBusinessOperation()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting EJB Descriptors

To add, update, or delete business operations that are implemented by an EJB, you need to get the EJB descriptor. To get an EJB descriptor, use the methods available via the `com.bea.wlpi.server.catalog.EJBCatalog` EJB.

Note: The `EJBInvocationDescriptor` class also provides methods for getting, setting, and invoking EJB descriptors. For more information, see "EJBInvocationDescriptor Object" on page C-6.

The following table lists the `com.bea.wlpi.server.catalog.EJBCatalog` methods that can be used to set preferences for getting EJB descriptors.

Table 10-4 EJBCatalog EJB Methods for Setting Preferences for Getting an EJB Descriptor

Method	Description
`public void setCatalogRoot( java.lang.String root ) throws java.rmi.RemoteException`	Set the JNDI context to use when looking up installed EJBs. *`root`* specifies a valid JNDI name of context containing EJB bindings.
`public void setInspectAlways( boolean inspectAlways ) throws java.rmi.RemoteException`	Specify whether the process engine regenerates the JNDI names and `EJBMetaData` when getting an EJB descriptor or name. *`inspectAlways`* specifies a Boolean argument to enable (`true`) or disable (`false`) automatic regeneration.

Table 10-4 EJBCatalog EJB Methods for Setting Preferences for Getting an EJB Descriptor

Method	Description
`public boolean inspectAlways( ) throws java.rmi.RemoteException`	Determine whether automatic regeneration is enabled or disabled. Returns a Boolean argument specifying whether or not the JNDI names and `EJBMetaData` are regenerated when the process engine is getting an EJB descriptor or name.

The following table lists the `com.bea.wlpi.server.catalog.EJBCatalog` methods that can be used to get an EJB descriptor.

Table 10-5 EJBCatalog EJB Methods for Getting an EJB Descriptor

Method	Description
`public java.util.List getEJBDescriptors( ) throws com.bea.wlpi.common.WorkflowException, java.rmi.RemoteException`	Get the EJBs installed in the previously specified JNDI context. The default implementation recursively iterates through all JNDI bindings within this context and builds a list containing an `EJBMetaData` object for each `EJBHome` object encountered. This method returns a list of `javax.ejb.EJBMetaData` objects.
`public java.util.List getEJBNames( ) throws com.bea.wlpi.common.WorkflowException, java.rmi.RemoteException`	Get the JNDI names of the EJBs installed in the previously specified JNDI context. The default implementation recursively iterates through all JNDI bindings within this context and builds a list containing the JNDI names of all `EJBHome` objects encountered. This method returns a list of JNDI names.

For example, the following code enables hot deployment of EJBs and gets the JNDI names of all `EJBHome` objects. In this example, `catalog` represents the `EJBObject` reference to the `EJBCatalog` EJB.

```
catalog.setInspectAlways(true);
List ejbList = catalog.getEJBNames();
```

For more information about the `EJBCatalog` methods, see the `com.bea.wlpi.server.catalog.EJBCatalog` Javadoc.

Getting Java Class Descriptors

To add, update, or delete business operations that are implemented by a Java class, you need to get the class descriptor. To get a class descriptor, use the methods available via the `com.bea.wlpi.server.admin.Admin` EJB.

Note: The `ClassInvocationDescriptor` class also provides methods for getting, setting, and invoking class descriptors. For more information, see "ClassInvocationDescriptor Object" on page C-3.

To get a Java class descriptor, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public com.bea.wlpi.common.ClassDescriptor getClassDescriptor(
  java.lang.String className
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getClassDescriptor()` method parameter for which you must specify a value.

Table 10-6 getClassDescriptor() Method Parameter

Parameter	Description	Valid Values
className	Fully-qualified Java class name for which you want a descriptor returned.	String specifying a valid, fully qualified Java class name.

This method returns a `ClassDescriptor`, consisting of meta-data for the specified Java class.

For example, the following code gets the class descriptor for the specified Java class, `com.somedomain.someproduct.CreateWidget`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
ClassDescriptor descriptor = admin.getClassDescriptor
  ("com.somedomain.someproduct.CreateWidget");
```

For more information about the `getClassDescriptor()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Examples of Configuring Business Operations

The following sections provide excerpts from the command-line administration client example showing how to configure the environment:

- Example of Getting an EJB Descriptor
- Examples of Configuring Business Operations

Example of Getting an EJB Descriptor

This section provides excerpts from the command-line administration example showing how to get EJB descriptors and set related preferences.

Note: For more information about the command-line administration example, see "Command-Line Administration Example" on page 1-21.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Querying the Inspect Always Flag
- Setting the Inspect Always Flag
- Getting Deployed EJB Names
- Getting EJB Deployment Descriptors

Important lines of code are highlighted in **bold**. In this example, `catalog` represents the `EJBObject` reference to the `EJBCatalog` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) )

/* Display Tool Title */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---             Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
```

The `mngEJBCatalog()` method illustrates how to manage the EJB catalog, interacting with the user to retrieve the information required.

```
private static void mngEJBCatalog( ) {
   boolean isInspectAlways;
   List ejbList;
   String answer;

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader( new InputStreamReader( System.in )
);

   try {
   /* Display the menu and interact with user */
      while( true ) {
      /* Display the menu */
         System.out.println( "\n\n---            EJB Catalog           ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Query Inspect Always Flag" );
         System.out.println( "2) Set Inspect Always Flag" );
         System.out.println( "3) List names of deployed EJBs" );
         System.out.println( "4) List descriptors of deployed EJBs" );
         System.out.println( "B) Back to previous menu" );
```

```
      System.out.println( "Q) Quit" );
      System.out.print( ">> " );

      /* Get user's selection */
      String line = stdIn.readLine( );

      /* User pressed enter without making a selection ? */
      if( line.equals( "" ) )
         continue;
      /* User entered more than one char ? */
      else if( line.length( ) > 1 ) {
         System.out.println( "*** Invalid selection" );
      continue;
   }

   /* Convert to uppercase and to char */
   char choice = line.toUpperCase( ).charAt( 0 );

   /* Process user's selection */
   switch( choice ) {
   .
   .
   .
```

Querying the Inspect Always Flag

The following excerpt shows how to query the inspect always flag:

```
   /* Query Inspect Always Flag */
      case '1' :
         /* WLPI Public API Method */
         /* NOTE: Would be nice to add code to capture any
          * thrown exceptions */
         isInspectAlways = catalog.inspectAlways( );

         if( isInspectAlways )
            System.out.println( "\nInspect Always flag is set " +
               "(EJB hot deployment is enabled)" );
         else
            System.out.println( "\nInspect Always flag is not set " +
               "(EJB hot deployment is disabled)" );
         break;
         .
         .
         .
```

Setting the Inspect Always Flag

The following excerpt shows how to set the inspect always flag:

```
/* Set Inspect Always Flag */
case '2' :
   /* Get value to set the flag */
   /* Prompt user to select if we need to set/unset Inspect
    * Always flag */
   if( ( answer = askQuestion( "\nEnable Inspect Always (y/n)?" ) )
   == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
}

/* Parse the answer */
isInspectAlways = ( answer.equals( "y" ) || answer.equals( "Y" ) );

try {
   /* WLPI Public API Method */
   /* Set Inspect Always Flag */
   catalog.setInspectAlways( isInspectAlways );

   /* Success (No exception thrown) */
   System.out.println( "- Success" );
}
catch( Exception e ) {
   System.out.println( "*** Failed to set Inspect Always flag" );
   System.err.println( e );
}
break;
   .
   .
   .
```

Getting Deployed EJB Names

The following excerpt shows how to get the deployed EJB names:

```
/* List names of deployed EJBs */
case '3' :
   /* WLPI Public API Method */
   /* List the JNDI names of deployed EJBs */
   /* NOTE: Would be nice to add code to capture any a
```

```
         * thrown exceptions */
        ejbList = catalog.getEJBNames( );

        /* Any EJBs deployed ? */
        if( ejbList.size( ) == 0 )
           System.out.println( "\nNo EJB deployed" );
        else
           System.out.println( "\nCataloged EJBs:" );

        /* Process the list to display the names */
        for( int i = 0; i < ejbList.size( ); i++ )
           System.out.println( "- JNDI Name: " + ejbList.get( i ) );
        break;
        .
        .
        .
```

Getting EJB Deployment Descriptors

The following excerpt shows how to get the EJB deployment descriptors:

```
/* List descriptors of deployed EJBs */
case '4' :
   /* WLPI Public API Method */
   /* List the deployed EJBs */
   /* ejbList is a list of com.bea.wlpi.common.EJBDescriptor
    * objects. */
   /* NOTE: Would be nice to add code to capture any
    * thrown exceptions */
   ejbList = catalog.getEJBDescriptors( );

   /* Any EJBs deployed ? */
   if( ejbList.size( ) == 0 )
      System.out.println( "\nNo EJB deployed" );
   else
      System.out.println( "\nCataloged EJBs:" );

   /* Process the list to display miscelleanous attributes */
   for( int i = 0; i < ejbList.size( ); i++ ) {
      /* Retrieve an element from the list */
      EJBDescriptor ejbDescriptor = ( EJBDescriptor )ejbList.get( i );

      /* Display a sub-set of available attributes */
      System.out.println( "\n- Deployment Name: " +
         ejbDescriptor.getEJBDeploymentName( ) );
      System.out.println( "  Home Interface: " +
         ejbDescriptor.getEJBHomeName( ) );
```

```
            System.out.println( "  Remote Interface: " +
               ejbDescriptor.getEJBRemoteName( ) );
            /* And so on ...
             * See WLPI JavaDocs, Class EJBDescriptor,
             * for a listing of all methods available */
         }
         break;
         .
         .
         .
```

Examples of Configuring Business Operations

This section provides excerpts from the command-line administration example showing how to configure business operations.

Note: For more information about the command-line administration example, see "Command-Line Administration Example" on page 1-21.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Deleting a Business Operation
- Getting All Business Operations

Important lines of code are highlighted in **bold**. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* Display Tool Title */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
```

```
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/**
 * Method that interacts with the user to get all the required information
 * to illustrate the Public API Methods available in the Admin interface.
 * that are related to WLPI Business Operations.
*/
public static void mngBusinessOperations() {
   ClassInvocationDescriptor operationClassInfo;
   EJBInvocationDescriptor operationEjbInfo;
   List operationList;
   Object o;
   String operationId;

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
   /* Display the menu and interact with user */
      while( true ) {
      /* Display the menu */
         System.out.println( "\n\n---       Business Operations        ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Delete a Business Operation" );
         System.out.println( "2) List all Business Operations" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* Get user's selection */
         String line = stdIn.readLine();

         /* User pressed enter without making a selection ? */
         if( line.equals( "" ) )
            continue;
         /* User entered more than one char ? */
         else if( line.length() > 1 ) {
            System.out.println( "*** Invalid selection" );
            continue;
         }
```

```
/* Convert to uppercase and to char */
char choice = line.toUpperCase().charAt( 0 );

/* Process user's selection */
switch( choice ) {
.
.
.
```

Deleting a Business Operation

The following excerpt shows how to delete a business operation:

```
/* Delete a Business Operation */
case '1' :
   /* Get operation ID for the operation to delete */
   if( ( operationId = askQuestion( "\nEnter Business Operation
      ID to delete" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Delete the business operation */
      admin.deleteBusinessOperation( operationId );

      /* Success (No exception trown) */
      System.out.println( "- Deleted" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete Business Operation" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Getting All Business Operations

The following excerpt shows how to get all business operations:

```
/* List all Business Operations */
case '2' :
   try {
      /* WLPI Public API Method */
      /* Retrieve all business operations */
      operationList = admin.getBusinessOperations();

      /* Any operations defined ? */
      if( operationList.size() == 0 )
         System.out.println( "\nNo Business Operation defined" );
      else
         System.out.println( "\nDefined Business Operations:" );

      /* Process the list to display operations and attributes */
      for( int i = 0; i < operationList.size(); i++ ) {
         /* Retrieve an element from the list */
         o = operationList.get( i );

         /* Is the Business Operation a Class ? */
         if( o instanceof ClassInvocationDescriptor ) {
            operationClassInfo = ( ClassInvocationDescriptor ) o;

            /* Display business operation type */
            System.out.println( "- Business Operation Type: Class" );

            /* Retrieve and display the operation ID */
            /* WLPI Public API Method */
            System.out.println( "  ID: " +
               operationClassInfo.getId() );

            /* Retrieve and display the class name */
            /* WLPI Public API Method */
            System.out.println( "  Class Name: " +
               operationClassInfo.getClassName() );

            /* Retrieve and display the class description */
            /* WLPI Public API Method */
            System.out.println( "  Description: " +
               operationClassInfo.getDescription() );

            /* And so on ...
             * See WLPI JavaDocs, Class
             * ClassInvocationDescriptor,
             * for a listing of all methods available */
```

```
      }
      /* Is the Business Operation an EJB ? */
      else if( o instanceof EJBInvocationDescriptor ) {
         operationEjbInfo = ( EJBInvocationDescriptor ) o;

         /* Display business operation type */
         System.out.print( "- Business Operation Type: " );

         /* Is it a session or entity EJB */
         /* WLPI Public API Method */
         if( operationEjbInfo.isSessionEJB() )
            System.out.println( "Session EJB" );
         else
            System.out.println( "Entity EJB" );

         /* Retrieve and display the operation ID */
         /* WLPI Public API Method */
         System.out.println( "  ID: " +
            operationEjbInfo.getId() );

         /* Retrieve and display the EJB deployment name */
         /* WLPI Public API Method */
         System.out.println( "  EJB deployment name: " +
            operationEjbInfo.getEJBDeploymentName() );

         /* Retrieve and display the bean description */
         /* WLPI Public API Method */
         System.out.println( "  Description: " +
            operationEjbInfo.getDescription() );

         /* And so on ...
          * See WLPI JavaDocs,
          * Class EJBInvocationDescriptor,
          * for a listing of all methods available */
      }
   }
}
catch( Exception e ) {
   System.out.println( "*** Unable to list Business Operations" );
   System.err.println( e );
}
break;
.
.
.
```

CHAPTER

11 Configuring Event Keys

This section explains how to configure event keys, including the following topics:

- Overview of Event Keys
- Adding an Event Key
- Getting Event Key Information
- Updating an Event Key
- Deleting an Event Key
- Example of Configuring Event Keys

For additional information, see "Configuring Event Keys" in "Configuring Workflow Resources" in *Using the WebLogic Integration Studio*.

Overview of Event Keys

An event key enables the Event Processor to calculate a unique key value for an inbound event by evaluating a workflow expression against the event data. Typically, the workflow expression extracts information from a named field in the incoming event data using a field reference or function call.

Event keys are characterized by two attributes—Content Type and Event Descriptor—and contain the contents of the workflow expression that extracts the unique key value from the particular Content Type and Event Descriptor attributes, as well as the plug-in-supplied field that supports that data type.

The following table describes the event key attributes.

Table 11-1 Event Key Attributes

Event Key Attribute	Description
Content Type	Multipurpose Internet Mail Extensions (MIME) type for the event data that identifies the underlying data type. The default content type supported by WebLogic Integration is `text/xml` (that is, XML documents). Plug-ins can be created to handle additional content types, for example, binary COBOL Copy Book records, comma-separated values (CSV), proprietary file formats (for example, Microsoft Word or Excel), serialized Java objects, HTML, SGML, and so on. For more information about programming BPM plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.
Event Descriptor	Optionally, provides additional schema-level information to define the data format. **Note:** Some content types do not require additional information to define the data format. In the case of the default content type, `text/xml`, the event descriptor reflects the XML document type, and matches either the Public ID or System ID (if the Public ID is not defined) in the DOCTYPE DTD (Document Type Definition), if one is declared by the incoming document. If a DOCTYPE is not declared, the event descriptor matches the document root element name. Plug-in-supported content types should use the event descriptor, if necessary, to provide sufficient information to enable a field reference to extract the specified field from the event data. In cases where the data are not self-describing (for example, a raw binary record image), the event descriptor must provide sufficient information to enable the field reference(s) to access the relevant data dictionary service to retrieve, for example, the byte offset, length, and data type of the referenced field.

The Event Processor is able to calculate a unique key value for any incoming event. If the event is destined for a particular workflow instance, as in the case of an Event node, it is necessary to match the calculated key value to the ID of the target workflow instance. This match is achieved through the Event Watch table, which is used to associate the incoming content type, event descriptor, and key value with the waiting instance and Event node IDs. The Event Watch table is also used to handle triggered workflows that are instantiated in response to the receipt of a particular content type, event descriptor, and key value combination (and in this case, the instance ID is null).

Adding an Event Key

To add an event key, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void addEventKey(
  com.bea.wlpi.common.EventKeyInfo eventKeyInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `addEventKey()` method parameter for which you must specify a value.

Table 11-2 addEventKey() Method Parameter

Parameter	Description	Valid Values
`eventKeyInfo`	New event key information.	An `EventKeyInfo` object. For information about defining the `EventKeyInfo` object, see "EventKeyInfo Object" on page B-4.

For example, the following code adds an event key based on the contents of the specified `eventKeyInfo` object. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.addEventKey(eventKeyInfo);
```

For more information about the `addEventKey()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting Event Key Information

To get information about an event key, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List getEventKeyInfo(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

This method returns a list of `com.bea.wlpi.common.EventKeyInfo` objects. To access information about each event key, use the `EventKeyInfo` object methods described in "EventKeyInfo Object" on page B-4.

For example, the following code gets event key information and saves it to the `eventKeys` list object. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List eventKeys = admin.getEventKeyInfo();
```

For more information about the `getEventKey()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Updating an Event Key

To update an event key, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void updateEventKey(
  com.bea.wlpi.common.EventKeyInfo eventKeyInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `updateEventKey()` method parameter for which you must specify a value.

Table 11-3 updateEventKey() Method Parameter

Parameter	Description	Valid Values
`eventKeyInfo`	Event key information that you want to update.	An `EventKeyInfo` object. For information about defining the `EventKeyInfo` object, see "EventKeyInfo Object" on page B-4.

For example, the following code updates an event key based on the contents of the specified `eventKeyInfo` object. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.updateEventKey(eventKeyInfo);
```

For more information about the `updateEventKey()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Deleting an Event Key

To delete an event key, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void deleteEventKey(
  com.bea.wlpi.common.EventKeyInfo eventKeyInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteEventKey()` method parameter for which you must specify a value.

Table 11-4 deleteEventKey() Method Parameter

Parameter	Description	Valid Values
`eventKeyInfo`	Event key that you want to delete.	An existing `EventKeyInfo` object. For information about defining the `EventKeyInfo` object, see "EventKeyInfo Object" on page B-4.

For example, the following code deletes the specified event key. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.deleteEventKey(eventKeyInfo);
```

For more information about the `deleteEventKey()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Example of Configuring Event Keys

This section provides excerpts from the command-line administration example showing how to configure event keys.

Note: For more information about the command-line administration example, see "Command-Line Administration Example" on page 1-21.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Adding an Event Key
- Deleting an Event Key
- Getting Event Keys
- Updating Event Keys

Important lines of code are highlighted in **bold**. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* Display Tool Title */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---             Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/**
 * Method that interacts with the user to get all the required information
 * to illustrate the Public API Methods available in the Admin interface
 * that are related to WLPI Event Keys.
 * An event key is used to identify the
 * workflow definitions or instance(s) that are associated with an incoming
 * XML document. Usage of an event key avoids a sequential search of all XML
 * based workflow triggers in order to find the right one.
 */
public static void mngEventKeys() {
   EventKeyInfo eventKeyInfo;
   List eventKeyList;
   String eventKeyRoot;
   String eventKeyExpr;
   //boolean isGetDefinition;
   //BusinessCalendarInfo calendarInfo;
   //String answer;
   //String calendarDefinition;
   //String calendarId;
   //String calendarName;
   //String calendarTimezone;// XML Document

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader( new InputStreamReader(
```

```
      System.in ) );

   try {
      /* Display the menu and interact with user */
      while( true ) {
         /* Display the menu */
         System.out.println( "\n\n---            Event Keys            ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Add an Event Key" );
         System.out.println( "2) Delete an Event Key" );
         System.out.println( "3) List all Event Keys" );
         System.out.println( "4) Update an Event Key" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* Get user's selection */
         String line = stdIn.readLine();

         /* User pressed enter without making a selection ? */
         if( line.equals( "" ) )
            continue;
         /* User entered more than one char ? */
         else if( line.length() > 1 ) {
            System.out.println( "*** Invalid selection" );
            continue;
         }

         /* Convert to uppercase and to char */
         char choice = line.toUpperCase().charAt( 0 );

         /* Process user's selection */
         switch( choice ) {
         .
         .
         .
```

Adding an Event Key

The following excerpt shows how to add an event key:

```
/* Add an Event Key */
case '1' :
   /* Get Event Key XML Document root for the new Event Key */
   if( ( eventKeyRoot = askQuestion( "\nEnter the Event Key XML
      Document Root" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get Expression for the new Event Key */
  if( ( eventKeyExpr = askQuestion( "Enter the Event Key Expression" ) )
      == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* Create an EventKeyInfo object; required to invoke
       * the add method */
      eventKeyInfo = new EventKeyInfo(
         "text/xml", eventKeyRoot, eventKeyExpr, null, 0 );

      /* WLPI Public API Method */
      /* Add the Event Key */
      admin.addEventKey( eventKeyInfo );

      /* Success (No exception trown) */
      System.out.println( "- Added" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to add the Event Key" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Deleting an Event Key

The following excerpt shows how to delete event keys:

```
/* Delete an Event Key */
case '2' :
   /* Get Event Key XML Document root for the Event Key to delete */
   if( ( eventKeyRoot = askQuestion( "\nEnter the root for
      the Event Key to delete" ) ) == null ) {
   /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* To invoke the delete method, we need to create an
       * EventKeyInfo object which basically consist of an
       * XML Document root and an expression. Since we are
       * deleting the expression is not required. */
      eventKeyExpr = "";

      /* Create an EventKeyInfo object; required to
       * invoke the delete method */
      eventKeyInfo = new EventKeyInfo(
         "text/xml", eventKeyRoot, eventKeyExpr, null, 0 );

      /* WLPI Public API Method */
      /* Delete the Event Key */
      admin.deleteEventKey( eventKeyInfo );

      /* Success (No exception trown) */
      System.out.println( "- Deleted" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete the Event Key" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Getting Event Keys

The following excerpt shows how to get event keys:

```
/* List all Event Keys */
case '3' :
   try {
      /* WLPI Public API Method */
      /* Retrieve all Event Keys */
     eventKeyList = admin.getEventKeyInfo();

      /* Any Event Keys defined ? */
      if( eventKeyList.size() == 0 )
         System.out.println( "\nNo Event Keys defined" );
      else
         System.out.println( "\nDefined Event Keys:" );

      /* Process the list to display event keys and attributes */
      for( int i = 0; i < eventKeyList.size(); i++ ) {
         /* Retrieve an element from the list */
         eventKeyInfo = ( EventKeyInfo )eventKeyList.get( i );

         /* WLPI Public API Method */
         /* Retrieve and display the XML Document root */
         System.out.println( "- Content Type:       " +
           eventKeyInfo.getContentType() );

         /* WLPI Public API Method */
         /* Retrieve and display the XML Document root */
         System.out.println( "- Event Descriptor:       " +
           eventKeyInfo.getEventDescriptor() );

         /* WLPI Public API Method */
         /* Retrieve and display calendar name */
         System.out.println( "  Expression: " +
            eventKeyInfo.getExpr() );
      }
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to retrieve Event Keys" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Updating Event Keys

The following excerpt shows how to update an event key:

```
/* Update an Event Key */
case '4' :
   /* Get Event Key XML Document root for the Event Key to update */
   if( ( eventKeyRoot = askQuestion( "\nEnter the root for the
      Event Key to update" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the new Expression for the Event Key */
   if( ( eventKeyExpr = askQuestion( "Enter the new Event Key
      Expression" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* Create an EventKeyInfo object; required to invoke
       * the update method */
      eventKeyInfo = new EventKeyInfo(
         "text/xml", eventKeyRoot, eventKeyExpr, null, 0 );

      /* WLPI Public API Method */
      /* Update the Event Key */
      admin.updateEventKey( eventKeyInfo );

      /* Success (No exception trown) */
      System.out.println( "- Updated" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to update the Event Key" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

CHAPTER

12 Configuring Business Calendars

Business calendars enable you to define the operating hours for an organization. For more information about administering business calendars, see "Administering Business Calendars" in "Administering Data" in *Using the WebLogic Integration Studio*.

This section describes the tasks associated with configuring business calendars, including the following topics

- Adding a Business Calendar
- Getting Business Calendars
- Getting a Business Calendar Definition
- Updating a Business Calendar
- Deleting a Business Calendar
- Example of Configuring Business Calendars

Adding a Business Calendar

To add a business calendar, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.lang.String addBusinessCalendar(
  com.bea.wlpi.common.BusinessCalendarInfo calendarInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `addBusinessCalendar()` method parameter for which you must specify a value.

Table 12-1 addBusinessCalendar() Method Parameter

Parameter	Description	Valid Values
calendarInfo	New business calendar information.	A `BusinessCalendarInfo` object. For information about defining the `BusinessCalendarInfo` object, see "BusinessCalendarInfo Object" on page B-2.

This method returns the ID of the new business calendar.

For example, the following code adds a business calendar based on the contents of the specified `calendarInfo` object. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.addBusinessCalendar(calendarInfo);
```

For more information about the `addBusinessCalendar()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting Business Calendars

To get the contents of all business calendars that are currently defined, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List getAllBusinessCalendars(
  boolean includeDefinition
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getAllBusinessCalendars()` method parameter for which you must specify a value.

Table 12-2 getAllBusinessCalendars() Method Parameter

Parameter	Description	Valid Values
includeDefintion	Boolean flag specifying whether or not to include the XML calendar specification	`true` (include XML) or `false` (do not include XML).

This method returns a list of `com.bea.wlpi.common.BusinessCalendarInfo` objects. To access information about each business calendar, use the `BusinessCalendarInfo` object methods described in "BusinessCalendarInfo Object" on page B-2.

For example, the following code gets the business calendar information and saves it to the `calendars` list object. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List calendars = admin.getAllBusinessCalendars();
```

For more information about the `getAllBusinessCalendars()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting a Business Calendar Definition

To get a business calendar definition, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.lang.String getBusinessCalendarDefinition(
  java.lang.String calendarId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getBusinessCalendarDefinition()` method parameter for which you must specify a value.

Table 12-3 getBusinessCalendarDefinition() Method Parameter

Parameter	Description	Valid Values
calendarId	ID of the business calendar for which you want to retrieve the definition.	String specifying a valid business calendar ID. To get the calendar ID, use the following `com.bea.wlpi.common.BusinessCalendarInfo` method: `public final java.lang.String getId()` For information about getting the `BusinessCalendarInfo` object, see "Getting Business Calendars" on page 12-3. For more information about the methods available to the `BusinessCalendarInfo` object, see "BusinessCalendarInfo Object" on page B-2.

This method returns an XML file containing the business calendar definition that is compliant with the "Business Calendar DTD" on page A-11.

For example, the following code gets information about a single business calendar definition and saves it to the `calendar` string. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
String calendar = admin.getBusinessCalendarDefinition();
```

For more information about the `getBusinessCalendarDefinition()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Updating a Business Calendar

To update a business calendar, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void updateBusinessCalendar(
  com.bea.wlpi.common.BusinessCalendarInfo calendarInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `updateBusinessCalendar()` method parameter for which you must specify a value.

Table 12-4 updateBusinessCalendar() Method Parameter

Parameter	Description	Valid Values
calendarInfo	Calendar information that you want to update.	A `BusinessCalendarInfo` object. For information about defining the `BusinessCalendarInfo` object, see "BusinessCalendarInfo Object" on page B-2.

For example, the following code updates a business calendar based on the contents of the specified `calendarInfo` object. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.updateBusinessCalendar(calendarInfo);
```

For more information about the `updateBusinessCalendar()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Deleting a Business Calendar

To delete a business calendar, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void deleteBusinessCalendar(
  java.lang.String calendarId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteBusinessCalendar()` method parameter for which you must specify a value.

Table 12-5 deleteBusinessCalendar() Method Parameter

Parameter	Description	Valid Values
calendarId	ID of the business calendar that you want to delete.	String specifying a valid business calendar ID. To get the calendar ID, use the following `com.bea.wlpi.common.BusinessCalendarInfo` method: `public final java.lang.String getId()` For information about getting the `BusinessCalendarInfo` object, see "Getting Business Calendars" on page 12-3. For more information about the methods available to the `BusinessCalendarInfo` object, see "BusinessCalendarInfo Object" on page B-2.

For example, the following code deletes the specified business calendar. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.deleteBusinessCalendar(calendarInfo);
```

For more information about the `deleteBusinessCalendar()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Example of Configuring Business Calendars

This section provides excerpts from the command-line administration example that show how to configure business calendars.

Note: For more information about the command-line administration example, see "Command-Line Administration Example" on page 1-21.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Adding a Business Calendar
- Deleting a Business Calendar
- Getting a Business Calendar Definition
- Getting Business Calendars
- Updating a Business Calendar

Important lines of code are highlighted in **bold**. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* Display Tool Title */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
```

```
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/**
 * Method that interacts with the user to get all the required information
 * to illustrate the Public API Methods available in the Admin interface
 * that are related to WLPI Business Calendars.
 */
public static void mngBusinessCalendars() {\
   boolean isGetDefinition;
   BusinessCalendarInfo calendarInfo;
   List calendarList;
   String answer;
   String calendarDefinition;
   String calendarId;
   String calendarName;
   String calendarTimezone;// XML Document

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* Display the menu and interact with user */
      while( true ) {
         /* Display the menu */
         System.out.println( "\n\n---        Business Calendars        ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Add a Business Calendar" );
         System.out.println( "2) Delete a Business Calendar" );
         System.out.println( "3) List a Business Calendar Definition" );
         System.out.println( "4) List all Business Calendars" );
         System.out.println( "5) Update a Business Calendar" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );
         /* Get user's selection */
         String line = stdIn.readLine();
         /* User pressed enter without making a selection ? */
         if( line.equals( "" ) )
            continue;
         else if( line.length() > 1 ) {
            System.out.println( "*** Invalid selection" );
             continue;
         }
         /* Convert to uppercase and to char */
         char choice = line.toUpperCase().charAt( 0 );
```

```
/* Process user's selection */
switch( choice ) {
.
.
.
```

Adding a Business Calendar

The following excerpt shows how to add a business calendar:

```
/* Add Business Calendar */
case '1' :
   /* Adding a new Business Calendar, thus we do not have
      a calendar id. The server will assign it an return the
      new ID. We must set it to the word 'null' to specify no
      value (do not confuse with a null string). */
      calendarId = new String( "null" );

   /* Get Calendar Name for the new calendar to add */
  if( ( calendarName = askQuestion( "\nEnter new Calendar Name" ) )
     == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

  /* Get Calendar TimeZone for the new calendar to add */
 if( ( calendarTimezone = askQuestion( "Enter Time Zone" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* A business calendar definition is an XML document
      conforming to the BusinessCalendar.dtd document
      type (see the WLPI javadocs).
      It contains all the
      information that defines the Business Calendar */
   /* We need to build the XML Document */
   calendarDefinition = createCalendarDefinition( calendarId,
      calendarName, calendarTimezone );

   /* Create a BusinessCalendarInfo object */
   calendarInfo = new BusinessCalendarInfo( calendarId, calendarName,
      calendarTimezone, calendarDefinition );
```

```
   try {
      /* WLPI Public API Method */
      /* Create the new business calendar */
      calendarId = admin.addBusinessCalendar( calendarInfo );

      /* Success (No exception trown) */
      System.out.println( "- Business Calendar added (ID=" +
         calendarId + ")" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to add business calendar\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Deleting a Business Calendar

The following excerpt shows how to delete a business calendar:

```
/* Delete Business Calendar */
case '2' :
   /* Get Calendar ID for the calendar to delete */
   if( ( calendarId = askQuestion( "\nEnter Calendar ID to delete" ) )
      == null ) {
     /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Delete the business calendar */
      admin.deleteBusinessCalendar( calendarId );

      /* Success (No exception trown) */
      System.out.println( "- Deleted" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete Business Calendar" );
      System.err.println( e );
   }
```

```
   break;
   .
   .
   .
```

Getting a Business Calendar Definition

The following excerpt shows how to get a business calendar definition:

```
/* List a Business Calendar Definition */
case '3' :
  /* Get Calendar ID for the calendar to delete */
   if( ( calendarId = askQuestion( "\nEnter Calendar ID" ) ) == null ) {
       /* User cancelled the operation */
       System.out.println( "*** Cancelled" );
       break;
   }

   try {
       /* WLPI Public API Method */
       /* Retrieve the definition of the business calendar. */
       calendarDefinition = admin.getBusinessCalendarDefinition(
          calendarId );

   /* Success (No exception trown) */
   System.out.println( "- Business Calendar definition:\n" +
       calendarDefinition );
}
catch( Exception e ) {
   System.out.println( "*** Unable to retrieve Business " +
       "Calendar definition\n" );
   System.err.println( e );
}
break;
.
.
.
```

Getting Business Calendars

The following excerpt shows how to get business calendars:

```
/* List all Business Calendars */
case '4' :
 /* Prompt user to select if we need to display the calendar definition */
  if( ( answer = askQuestion( "\nList Definition (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Parse the answer */
   isGetDefinition = ( answer.equals( "y" ) || answer.equals( "Y" ) );

   try {
      /* WLPI Public API Method */
      /* Retrieve all business calendars */
      //Bug1 calendarList = admin.getAllBusinessCalendars(
      //isGetDefinition );
      calendarList = admin.getAllBusinessCalendars( false );

      /* Any calendars defined ? */
      if( calendarList.size() == 0 )
         System.out.println( "\nNo Business Calendar defined" );
      else
         System.out.println( "\nDefined Business Calendars:" );

      /* Process the list to display calendars and attributes */
      for( int i = 0; i < calendarList.size(); i++ ) {
         /* Retrieve an element from the list */
         calendarInfo = ( BusinessCalendarInfo )calendarList.get( i );

         /* Retrieve and display organization id */
         System.out.println( "- ID:        " + calendarInfo.getId() );

         /* Retrieve and display calendar name */
        System.out.println( "  Name:      " + calendarInfo.getName() );

         /* Retrieve and display time zone */
       System.out.println( "  Time Zone: " + calendarInfo.getTimeZone() );

         /* Display Business Calendar Definition ? */
         if( isGetDefinition ) {
            /* Retrieve and display calendar definition */
            //Bug1 Description:     getXML prefixes the returned
```

```
            // string with invalid characters */
            //Bug1    System.out.println( "  Business Calendar
            //definition:\n" +
            //Bug1    calendarInfo.getXML() );
            /* Work Around to retrieve calendar definition */
            calendarDefinition = admin.getBusinessCalendarDefinition(
               calendarInfo.getId() );
            /* Display calendar definition */
            System.out.println( "  Business Calendar definition:\n" +
              calendarDefinition );
         }
      }
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to retrieve Business Calendars" );
       System.err.println( e );
  }
   break;
   .
   .
   .
```

Updating a Business Calendar

The following excerpt shows how to update a business calendar:

```
/* Update Business Calendar */
case '5' :
   /* Get Calendar ID for the calendar to update */
   if( ( calendarId = askQuestion( "\nEnter Calendar ID to update" ) )
      == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Retrieve all business calendars in order to retrieve the
         one to update*/
      calendarList = admin.getAllBusinessCalendars( false );

      /* Any calendars defined ? */
      if( calendarList.size() == 0 ) {
         System.out.println( "\nNo Business Calendar defined" );
```

```
            break;
         }

         boolean isNotDefined = true;
         calendarInfo = null;

      /* Process the list to retrieve the info for the calendar to update */
         for( int i = 0; i < calendarList.size(); i++ ) {
            /* Retrieve an element from the list */
            calendarInfo = ( BusinessCalendarInfo )calendarList.get( i );

            /* Retrieve and display organization id */
            if( calendarId.equals( calendarInfo.getId() ) ) {
               isNotDefined = false;
               break;
            }
         }

      /* Is the calendar defined ? */
         if( isNotDefined ) {
            System.out.println( "*** This Business Calendar is
               not defined" );
            break;
         }

         System.out.println( "\nUpdating Business Calendar" );

         /* Retrieve and display calendar name */
         System.out.println( "- The current Name is: " +
            calendarInfo.getName() );

        /* Get new Calendar Name */
       if( ( calendarName = askQuestion( "  Enter a new Name " ) ) != null )
            /* User entered a new value, thus let's update the value */
            /* WLPI Public API Method */
            calendarInfo.setName( calendarName );

         /* Retrieve and display time zone */
         System.out.println( "- The current Time Zone is: " +
            calendarInfo.getTimeZone() );

         /* Get Calendar TimeZone for the new calendar to add */
      if( ( calendarTimezone = askQuestion( "  Enter a new Time Zone " ) )
            != null )
            /* User entered a new value, thus let's update the value */
            /* WLPI Public API Method */
            calendarInfo.setTimeZone( calendarTimezone );

         /* Create the new business calendar definition */
```

```
      calendarDefinition = updateCalendarDefinition( calendarId,
         calendarInfo.getName(), calendarInfo.getTimeZone() );

      /* Update the business calendar definition */
      calendarInfo.setXML( calendarDefinition );

      /* WLPI Public API Method */
      admin.updateBusinessCalendar( calendarInfo );

      /* Success (No exception trown) */
      System.out.println( "\nUpdated" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to update Business Calendar" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Part III Design

CHAPTER

13 Creating and Managing Workflow Templates

Workflow templates provide containers that hold one or more workflow template definitions.

This section explains how to create and manage workflow templates, including the following topics:

- Creating a Template
- Getting a Template
- Getting the Templates for an Organization
- Getting the Template Organizations
- Setting the Template Organizations
- Deleting a Template
- Example of Managing Templates

For more information about the methods described in this section, see the `com.bea.wlpi.server.admin.Admin` Javadoc. For information about managing workflow templates using the WebLogic Integration Studio, see "Defining Workflow Templates" in *Using the WebLogic Integration Studio*.

Creating a Template

To create a workflow template, use one of the following `com.bea.wlpi.server.admin.Admin` methods.

Method 1

```
public java.lang.String createTemplate(
  java.lang.String name,
  java.lang.String xml,
  java.util.Collection orgs
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public java.lang.String createTemplate(
  java.lang.String name,
  java.lang.String xml,
  java.util.Collection orgs,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The first method can be used in a nonclustered environment. The second method is recommended for use in a clustered environment. In this case, using the specified transaction ID, the system tracks the method execution status so that the method is not reissued after the transaction is committed, or in the event of a server crash or failover.

The following table describes the `createTemplate()` method parameters for which you must specify values.

Table 13-1 createTemplate() Method Parameters

Parameter	Description	Valid Values
name	Template name.	Non-null string.
xml	Name of the XML document that defines the plug-in data and that is compliant with the Template Definition DTD, as described in "Template Definition DTD" on page A-54. For more information about programming BPM plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.	Non-null string.
orgs	Collection of organization IDs that can access the template.	Collection of strings consisting of valid organization IDs. For information about getting a list of organization IDs, see "Getting All Organizations" on page 9-9.
transactionId	ID of the transaction. **Note:** This parameter is required only in a clustered environment.	Object specifying a unique transaction ID. To generate a unique transaction ID, create a new `com.bea.wlpi.client.common.GUID` object using the following constructor: `GUID transactionId = new GUID();` For more information about the GUID class, see the `com.bea.wlpi.client.common.GUID` Javadoc.

Each method returns the ID of the new template.

For example, the following code creates a new template named `Order Processing` that is accessible from the specified collection of organizations, `orgIds`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
String id = admin.createTemplate("Order Processing", null, orgIds);
```

For more information about the `createTemplate()` methods, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting a Template

To get a workflow template, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public com.bea.wlpi.common.TemplateInfo getTemplate(
  java.lang.String templateId,
  boolean byName
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getTemplate()` method parameters for which you must specify values.

Table 13-2 getTemplate() Method Parameter

Parameter	Description	Valid Values
templateId	ID or name of the template that you want to retrieve.	String specifying a valid template ID or name.
byName	Boolean flag specifying whether or not the value specified for the `templateId` parameter is the template name (`true`) or ID (`false`).	`true` (name) or `false` (name).

Each method returns the `com.bea.wlpi.common.TemplateInfo` object corresponding to the template. To access information about the template, use the `TemplateInfo` object methods described in "TemplateInfo Object" on page B-25.

For example, the following code gets the template with the name `Order Processing` (note, the *byName* parameter is set to `true`). In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
TemplateInfo template = admin.getTemplate(
  "Order Processing", true);
```

For more information about the `getTemplate()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting the Templates for an Organization

To get a list of workflow templates for an organization, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List getTemplates(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getTemplates()` method parameter for which you must specify a value.

Table 13-3 getTemplates() Method Parameter

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to get templates.	String specifying a valid organization ID. For information about getting a list of organization IDs, see "Getting All Organizations" on page 9-9.

This method returns a list of `com.bea.wlpi.common.TemplateInfo` objects. To access information about each template, use the `TemplateInfo` object methods described in "TemplateInfo Object" on page B-25.

For example, the following code gets all templates associated with the `ORG1` organization. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List templates = admin.getTemplates("ORG1");
```

For more information about the `getTemplates()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting the Template Organizations

To get the organizations that have access to a template, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public List getTemplateOrgs(
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getTemplateOrgs()` method parameter for which you must specify a value.

Table 13-4 getTemplateOrgs() Method Parameter

Parameter	Description	Valid Values
templateId	ID of the template for which you want to retrieve the organizations.	String specifying a valid template ID. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateInfo` object, see "Getting the Templates for an Organization" on page 13-5. For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25.

Each method returns a list of organization IDs.

For example, the following code gets a list of organizations that have access to a template and assigns the result to `orgs`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List orgs = admin.getTemplateOrgs(template.getId());
```

The template ID is obtained using the methods associated with the `com.bea.wlpi.common.TemplateInfo` object, `template`. The `template` object can be obtained using the methods described in "Getting the Templates for an Organization" on page 13-5.

For more information about the `getTemplateOrgs()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Setting the Template Organizations

To set the organizations that have access to a template, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void setTemplateOrgs(
  java.lang.String templateId,
  java.util.Collection orgs
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `setTemplateOrgs()` method parameters for which you must specify values.

Table 13-5 setTemplateOrganizations() Method Parameters

Parameter	Description	Valid Values
templateId	ID of the template for which you want to set organizations.	String specifying a valid template ID. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateInfo` object, see "Getting the Templates for an Organization" on page 13-5. For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25.

Table 13-5 setTemplateOrganizations() Method Parameters (Continued)

Parameter	Description	Valid Values
orgs	Collection of organization IDs that can access the template.	Collection of strings consisting of valid organization IDs. For information about getting a list of organization IDs, see "Getting All Organizations" on page 9-9.

For example, the following code sets the organizations that have access to a template using the collection, `organizations`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List orgs = admin.setTemplateOrgs(
  template.getId(), template.getName(), organizations);
```

The template ID and name are obtained using the methods associated with the `com.bea.wlpi.common.TemplateInfo` object, `template`. The `template` object can be obtained using the methods described in "Getting the Templates for an Organization" on page 13-5.

For more information about the `setTemplateOrgs()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Updating a Template

To update a workflow template, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void updateTemplate(
  com.bea.wlpi.common.TemplateInfo info
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `updateTemplate()` method parameter for which you must specify a value.

Table 13-6 updateTemplate() Method Parameter

Parameter	Description	Valid Values
info	Template name, ID, and new information.	A `TemplateInfo` object with the `name` and `ID` fields set to identify the existing template that you want to update, and the remaining fields set to the new information.

For example, the following code updates an existing template, as defined by the `TemplateInfo` object, `info`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.updateTemplate(info);
```

For more information about the `updateTemplate()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Deleting a Template

To delete a template, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void deleteTemplate(
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteTemplate()` method parameter for which you must specify a value.

Table 13-7 deleteTemplate() Method Parameter

Parameter	Description	Valid Values
templateId	ID of the template that you want to delete.	String specifying a valid template ID. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateInfo` object, see "Getting the Templates for an Organization" on page 13-5. For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25.

For example, the following code deletes the specified template. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.deleteTemplate(template.getId());
```

The template ID is obtained using the methods associated with the `com.bea.wlpi.common.TemplateInfo` object, `template`. The `template` object can be obtained using the methods described in "Getting the Templates for an Organization" on page 13-5.

For more information about the `deleteTemplate()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Example of Managing Templates

This section provides excerpts from the command-line Studio example showing how to manage templates.

Note: For more information about the command-line Studio example, see "Command-Line Studio Example" on page 1-22.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Creating a Template
- Deleting a Template
- Getting Templates for an Organization

Important lines of code are highlighted in **bold**. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* Display Tool Title */
System.out.print( "\n---      Command Line Studio v1.0      ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---             Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Templates" );
   System.out.println( "2) Task Reroutes" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/**
 * Method that interacts with the user to get all the required information
 * to illustrate the Public API Methods available in the Admin interface
 * that are related to WLPI Workflow Templates.
 */
public static void mngTemplates() {
   ArrayList orgsList = new ArrayList();
   String answer;
   String orgId;
   String templateId;
   String templateName;

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader( new InputStreamReader(
      System.in ) );

   try {
      /* Display the menu and interact with user */
```

```
while( true ) {
   /* Display the menu */
   System.out.println( "\n\n---       Workflow Templates         ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Create a Template" );
   System.out.println( "2) Delete a Template" );
   System.out.println( "3) List Templates for an Organization" );
   System.out.println( "B) Back to previous menu" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );

   /* Get user's selection */
   String line = stdIn.readLine();

   /* User pressed enter without making a selection ? */
   if( line.equals( "" ) )
      continue;
   /* User entered more than one char ? */
   else if( line.length() > 1 ) {
      System.out.println( "*** Invalid selection" );
      continue;
   }

   /* Convert to uppercase and to char */
   char choice = line.toUpperCase().charAt( 0 );

   /* Process user's selection */
   switch( choice ) {
```

Creating a Template

The following excerpt shows how to create a template:

```
   /* Create a Template */
   case '1' :
      /* Get Template name */
      if( ( templateName = askQuestion( "\nEnter Template Name" ) )
         == null ) {
         /* User cancelled the operation */
         System.out.println( "*** Cancelled" );
         break;
      }

      System.out.println( "\nDefining organizations where
         the template is accessible" );
      boolean isEnterMore = true;
```

```
boolean isCancelled = false;

/* Loop to get the list of Organizations ID */
while( isEnterMore ) {
   /* Get Organization ID for the organization to set as active */
   if( ( orgId = askQuestion( "Enter an Organization ID" ) )
      == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      isCancelled = true;
      break;
   }

   orgsList.add( orgId );

   /* Should we keep looping to enter more org ID ? */
   if( ( answer = askQuestion( "Enter more (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      isCancelled = true;
      break;
   }

   /* Evaluate the answer */
   isEnterMore = ( answer.equals( "y" ) || answer.equals( "Y" ) );
}

/* Has the user cancelled the operation ? */
if( isCancelled )
   break;

try {
   /* WLPI Public API Method */
   /* Create the new Template */
  templateId = admin.createTemplate( templateName, null, orgsList );

   /* Success (No exception trown) */
   System.out.println( "- Created (template Id = " +
      templateId + ")" );
}
catch( Exception e ) {
   System.out.println( "*** Unable to create Template" );
   System.err.println( e );
}
break;
   .
   .
   .
```

Deleting a Template

The following excerpt shows how to delete a template:

```
/* Delete a Template */
case '2' :
   /* Get Template ID for the template to delete */
   if( ( templateId = askQuestion( "\nEnter Template ID
      to delete" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Delete the Template */
      admin.deleteTemplate( templateId );

      /* Success (No exception trown) */
      System.out.println( "- Deleted" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete Template" );
      System.err.println( e );
   }
   break;

      .
      .
      .
```

Getting Templates for an Organization

The following excerpt shows how to get templates for an organization:

```
/* List Templates for an Organization */
case '3' :
   /* Get Organization ID to query for */
   if( ( orgId = askQuestion( "\nEnter Organization ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
```

```
}

/* WLPI Public API Method */
/* Retrieve all templates defined in this organization */
/* NOTE: Would be nice to add code to capture any
 * thrown exceptions */
List templateList = admin.getTemplates( orgId );

/* Any templates defined ? */
if( templateList.size() == 0 )
   System.out.println( "\nNo template defined" );
else
   System.out.println( "\nDefined Templates:" );

/* Process the list to display Templates */
for( int i = 0; i < templateList.size(); i++ ) {
   /* Retrieve an element from the list */
   TemplateInfo templateInfo =
      ( TemplateInfo )templateList.get( i );

   /* Retrieve and display template id */
   System.out.println( "- Template ID: " + templateInfo.getId() );

   /* Retrieve and display template name */
   System.out.println( "  Name:        " +
      templateInfo.getName() + "\n" );
}
break;
   .
   .
   .
```

CHAPTER

14 Creating and Managing Workflow Template Definitions

A workflow template definition specifies the operations of a business process. One or more definitions can be created for each workflow template.

This section explains how to create and manage workflow template definitions, including the following topics:

- Creating a Template Definition
- Getting Template Definition Information
- Getting Definitions for a Template
- Getting the Template Definition Content
- Setting the Template Definition Content
- Getting the Template Definition Owner
- Getting Callable Workflow
- Finding a Callable Workflow
- Locking and Unlocking a Template Definition
- Deleting a Template Definition

For more information about the methods described in this section, see the `com.bea.wlpi.server.admin.Admin` Javadoc. For information about managing workflow templates definitions using the WebLogic Integration Studio, see "Defining Workflow Templates" in *Using the WebLogic Integration Studio*.

Creating a Template Definition

To create a workflow template definition, use one of the following `com.bea.wlpi.server.admin.Admin` methods.

Method 1

```
public java.lang.String createTemplateDefinition(
  java.lang.String templateId,
  java.lang.String name,
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public java.lang.String createTemplateDefinition(
  java.lang.String templateId,
  java.lang.String name,
  java.lang.String xml,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The first method can be used in a nonclustered environment. The second method is recommended for use in a clustered environment. In this case, using the specified transaction ID, the system tracks the method execution status so that the method is not reissued after the transaction is committed, or in the event of a server crash or failover.

The following table describes the `createTemplateDefinition()` method parameters for which you must specify values.

Table 14-1 createTemplateDefinition() Method Parameters

Parameter	Description	Valid Values
templateId	ID of the template for which you want to create a template definition.	String specifying a valid template ID. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateInfo` object, see "Getting a Template" on page 13-4. For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25.
name	Name of the new template definition.	Non-null string.
xml	Name of the XML document that defines the template definition and that is compliant with the Template Definition DTD, as described in "Template Definition DTD" on page A-54.	Non-null string.
transactionId	ID of the transaction. **Note:** This parameter is required only in a clustered environment.	Object specifying a unique transaction ID. To generate a unique transaction ID, create a new `com.bea.wlpi.client.common.GUID` object using the following constructor: `GUID transactionId = new GUID();` For more information about the GUID class, see the `com.bea.wlpi.client.common.GUID` Javadoc.

The method returns the ID of the new template definition.

For example, the following code creates a new template definition named `Order Processing` based on the `orderprocessing.xml` file. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
String id = admin.createTemplateDefinition(
  template.getId(),"Order Processing", "orderprocessing.xml");
```

The template ID is obtained using the methods associated with the `com.bea.wlpi.common.TemplateInfo` object, `template`. The `template` object can be obtained using the methods described in "Getting a Template" on page 13-4.

For more information about the `createTemplateDefinition()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting Template Definition Information

To get a workflow template definition, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public com.bea.wlpi.common.TemplateDefinitionInfo getTemplateDefinition(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getTemplateDefinition()` method parameter for which you must specify a value.

Table 14-2 getTemplateDefinition() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition that you want to retrieve.	String specifying a valid template definition ID.

The method returns a `com.bea.wlpi.common.TemplateDefinitionInfo` object. To access information about the template definition, use the `TemplateDefinitionInfo` object methods described in "TemplateDefinitionInfo Object" on page B-23.

For example, the following code gets the template definition associated with the specified ID. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
TemplateDefinitionInfo info =
  admin.getTemplateDefinitions(templateDefinitionID);
```

For more information about the `getTemplateDefinition()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting Definitions for a Template

To get the definitions for a specific workflow template, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List getTemplateDefinitions(
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getTemplateDefinitions()` method parameter for which you must specify a value.

Table 14-3 getTemplateDefinitions() Method Parameter

Parameter	Description	Valid Values
templateId	ID of the template for which you want to retrieve template definitions.	String specifying a valid template ID. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateInfo` object, see "Getting a Template" on page 13-4. For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25.

This method returns a list of `com.bea.wlpi.common.TemplateDefinitionInfo` objects. To access information about each template, use the `TemplateDefinitinoInfo` object methods described in "TemplateDefinitionInfo Object" on page B-23.

For example, the following code gets all templates associated with the `Order Processing` template. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List tempdefs = admin.getTemplateDefinitions("Order Processing");
```

For more information about the `getTemplateDefinitions()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting the Template Definition Content

To get the content of a template definition, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.lang.String getTemplateDefinitionContent(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getTemplateDefinitionContent()` method parameter for which you must specify a value.

Table 14-4 getTemplateDefinition() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition for which you want to get the content.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateDefinitionInfo` object, see "Getting Definitions for a Template" on page 14-5. For more information about the methods available to the `TemplateDefinitionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.

This method returns an XML document that is compliant with the Template Definition DTD, as described in "Template Definition DTD" on page A-54.

For example, the following code gets the contents for the specified template definition. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
String content =
  admin.getTemplateDefinitionContent(tempDef.getId());
```

The template definition ID is obtained using the methods associated with the `com.bea.wlpi.common.TemplateDefinitionInfo` object, `definition`. The `definition` object can be obtained using the methods described in "Getting Definitions for a Template" on page 14-5.

For more information about the `getTemplateDefinitionContent()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Setting the Template Definition Content

To set the template definition content, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void setTemplateDefinitionContent(
  java.lang.String templateDefinitionId,
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `setTemplateDefinitionContent()` method parameter for which you must specify a value.

Table 14-5 setTemplateDefinition() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition for which you want to set the content.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateDefinitionInfo` object, see "Getting Definitions for a Template" on page 14-5. For more information about the methods available to the `TemplateDefinitionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.
xml	New template definition contents.	XML file that is compliant with the Template Definition DTD, as described in "Template Definition DTD" on page A-54.

For example, the following code sets the contents for the specified template definition using the contents of the specified XML file, `new.xml`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.setTemplateDefinitionContent(tempDef.getId(), "new.xml");
```

The template definition ID is obtained using the methods associated with the `com.bea.wlpi.common.TemplateDefinitionInfo` object, `definition`. The `definition` object can be obtained using the methods described in "Getting Definitions for a Template" on page 14-5.

For more information about the `setTemplateDefinitionContent()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting the Template Definition Owner

To get the current owner of a workflow template definition, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.lang.String getTemplateOwner(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getTemplateOwner()` method parameter for which you must specify a value.

Table 14-6 getTemplateOwner() Method Parameter

Parameter	Description	Valid Values
templateDefinitionId	ID corresponding to the template definition for which you want to get the owner.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateDefinitionInfo` object, see "Getting Definitions for a Template" on page 14-5. For more information about the methods available to the `TemplateDefinitionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.

This method returns the ID of the current template definition owner.

For example, the following code gets the template definition owner ID and assigns the result to the `owner` string. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
String owner = admin.getTemplateOwner(templateDef.getId());
```

For more information about the `getTemplateOwner()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting Callable Workflow

To get a list of callable workflows, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List getCallableWorkflows(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getCallableWorkflows()` method parameter for which you must specify a value.

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to get callable workflows.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

This method returns a list of callable workflows.

For example, the following code gets the a list of callable workflows for the organization specified by the value of the `OrgID` variable. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List callablewf = admin.getCallableWorkflows(orgId);
```

For more information about the `getCallableWorkflows()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Finding a Callable Workflow

To return a list of the most appropriate (active and effective) callable workflow, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List findCallableWorkflows(
  java.lang.String templateName
  java.lang.String templateID
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getTemplateOwner()` method parameter for which you must specify a value.

Parameter	Description	Valid Values
templateName	Name of the template for which you want to find callable workflows.	Valid template name. To get the template name, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateInfo` object, see "Getting a Template" on page 13-4. For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25.

Parameter	Description	Valid Values
templateID	ID of the template for which you want to find callable workflows.	String specifying a valid template ID. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateInfo` object, see "Getting a Template" on page 13-4. For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25.
orgId	ID of the organization for which you want to find callable workflows.	Valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

This method returns a list of callable workflows.

For example, the following code finds a list of the most appropriate callable workflows for the specified template. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List callablewf = admin.findCallableWorkflows(templateName,
templateID, orgId);
```

For more information about the `findCallableWorkflows()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Locking and Unlocking a Template Definition

You can place a persistent write lock on a workflow template definition to protect it from unwanted edits. To lock and unlock template definitions, use the following `com.bea.wlpi.server.admin.Admin` methods, respectively:

```
public void lockTemplate(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public void unlockTemplate(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Note: To get the current template definition owner, use the `getTemplateOwner()` method described in "Getting the Template Definition Owner" on page 14-9.

The following table describes the parameter for which you must specify a value for the `lockTemplate()` and `unlockTemplate()` methods.

Table 14-7 lockTemplate() and unlockTemplate() Method Parameter

Parameter	Description	Valid Values
templateDefinitionId	ID corresponding to the template definition that you want to lock or unlock.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateDefinitionInfo` object, see "Getting Definitions for a Template" on page 14-5. For more information about the methods available to the `TemplateDefinitionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.

For example, the following code locks the specified template definition. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.lockTemplate(templateDef.getId());
```

The following code unlocks the specified template definition:

```
admin.unlockTemplate(templateDef.getId());
```

For more information about the `lockTemplate()` and `unlockTemplate()` methods, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Deleting a Template Definition

To delete a template definition, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void deleteTemplateDefinition(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteTemplateDefinition()` method parameter for which you must specify a value.

Table 14-8 getTemplateDefinition() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition that you want to delete.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final java.lang.String getId()` For information about getting the `TemplateDefinitionInfo` object, see "Getting Definitions for a Template" on page 14-5. For more information about the methods available to the `TemplateDefinitionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.

For example, the following code deletes the specified template definition. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.deleteTemplateDefinition(definition.getId());
```

The template definition ID is obtained using the methods associated with the `com.bea.wlpi.common.TemplateDefinitionInfo` object, `definition`. The `definition` object can be obtained using the methods described in "Getting Definitions for a Template" on page 14-5.

For more information about the `deleteTemplateDefinition()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

CHAPTER

15 Managing Tasks

Tasks are defined as part of the template definition creation process, as described in "Creating a Template Definition" on page 14-2.

This section describes how to manage tasks, including the following topics:

- Getting Tasks
- Assigning a Task
- Getting Task Counts
- Marking a Task Complete or Incomplete
- Setting Task Properties

For more information about the methods described in this section, see the `com.bea.server.admin.Admin` Javadoc. For information about managing tasks using the WebLogic Integration Studio, see "Defining Workflow Templates" in *Using the WebLogic Integration Studio*.

Getting Tasks

To get a list of the tasks defined for a particular workflow template, use one of the following `com.bea.wlpi.server.admin.Admin` methods:

```
public javautil.List getTasks(
  java.lang.String assigneeId,
  java.lang.String orgId,
  boolean role
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

```
public javautil.List getTasks(
  java.lang.String assigneeId,
  java.lang.String orgId,
  boolean role,
  boolean incompleteonly,
  boolean sortAscending
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the parameters for the `getTasks()` method for which you must specify values.

Table 15-1 getTasks() Method Parameters

Parameter	Description	Valid Values
assigneeId	ID of the *assignee* (user or role) for whom you want to get a list of tasks.	String specifying a valid user or role. For information about getting a list of current users or roles, see "Configuring the Security Realms" on page 9-1.
orgId	ID of the organization for which you want to get tasks.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.
role	Boolean flag specifying whether the assignee specified for the `assigneeId` parameter is a role (`true`) or user (`false`).	`true` (role) or `false` (user).
incompleteonly	Boolean flag specifying whether you want to return only those tasks that are incomplete (`true`) or all tasks (`false`).	`true` (incomplete tasks only) or `false` (all tasks).
sortAscending	Boolean flag specifying whether to sort tasks in ascending order (`true`) or descending order (`false`) based on time and date stamp.	`true` (ascending order) or `false` (descending order).

Each method returns a list of `com.bea.wlpi.common.TaskInfo` objects. To access information about each task, use the `TaskInfo` object methods described in "TaskInfo Object" on page B-20.

For example, the following code gets a list of the tasks for the ROLE1 role (notice that the role parameter is set to true) in the ORG1 organization. In this example, admin represents the EJBObject reference to the Admin EJB.

```
List taskList = admin.getTasks("ROLE1", "ORG1", true);
```

For more information about the getTasks() method, see the com.bea.wlpi.server.admin.Admin Javadoc.

Assigning a Task

To change the user or role to which a task is assigned, or to unassign a task, use the following com.bea.wlpi.server.admin.Admin methods, respectively:

```
public void taskAssign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.String assignTo,
  boolean bRole,
  boolean bLoadBalance
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public void taskUnassign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the parameters required by the taskAssign() and taskUnassign() methods for which you must specify values.

Table 15-2 taskAssign() and taskUnassign() Method Parameters

Parameter	Description	Valid Values	Valid For . . .	
			taskAssign()	taskUnassign()
templateDefinitionId	ID of the template definition from which the workflow instance containing the task was instantiated.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTemplateDefinitionId()` For information about getting the `TaskInfo` object, see "Getting Tasks" on page 15-1. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.	+	+

Table 15-2 taskAssign() and taskUnassign() Method Parameters (Continued)

Parameter	Description	Valid Values	Valid For . . .	
			taskAssign()	taskUnassign()
instanceId	ID of the workflow instance containing the task.	String specifying a valid instance ID. To get the instance ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getInstanceId()` For information about getting the `TaskInfo` object, see "Getting Tasks" on page 15-1. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.	+	+
taskId	ID of the task that you want to assign or unassign.	String specifying a valid task ID. To get the task ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTaskId()` For information about getting the `TaskInfo` object, see "Getting Tasks" on page 15-1. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.	+	+

Table 15-2 taskAssign() and taskUnassign() Method Parameters (Continued)

Parameter	Description	Valid Values	Valid For . . .	
			taskAssign()	taskUnassign()
assignTo	ID of the *assignee* (user or role) to whom you want to assign tasks.	String specifying a valid user or role. For information about getting a list of current users or roles, see "Configuring the Security Realms" on page 9-1.	+	
bRole	Boolean flag specifying whether or not the assignee specified for the `assigneeId` parameter is a role (`true`) or user (`false`).	`true` (role) or `false` (user).	+	
bLoadBalance	Boolean flag specifying whether or not load balancing should be performed within the specified role. (This setting is valid only if the `bRole` value is set to `true`.)	`true` (perform load balancing) or `false` (do not perform load balancing).	+	

The actual assignee to whom the task is assigned depends on the following:

- Whether any task reroutings currently exist for the specified assignee.
- Whether or not the *bRole* and *bLoadBalance* arguments are set to `true`, indicating that load balancing is in effect. In this case, the WebLogic Integration process engine reviews the number of tasks assigned to all users in a role (for whom there are no reroutings currently in effect), identifies the user with the least number of assigned tasks, and assigns the task to this user.

For example, the following code assigns a task to the user `joe`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.taskAssign(
  task.getTemplateDefinitionId(),
  task.getInstanceId(),
  task.getTaskId(),
  "joe",
  false,
  false
);
```

The following code unassigns the same task.

```
admin.taskUnassign(
  task.getTemplateDefinitionId(),
  task.getInstanceId(),
  task.getTaskId(),
  "joe",
  false,
  false
);
```

The template definition, workflow instance, and task IDs are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The `task` object can be obtained using the methods described in "Getting Tasks" on page 15-1.

For more information about the `com.bea.wlpi.common.TaskInfo` methods, see "TaskInfo Object" on page B-20, or the `com.bea.wlpi.common.TaskInfo` Javadoc.

For more information about the `taskAssign()` and `taskUnassign()` methods, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting Task Counts

To get the number tasks assigned to a particular user, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public int[] getTaskCounts(
       java.lang.String assigneeId,
       java.lang.String orgId,
       boolean isRole
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the parameters for the second `getTaskCounts()` method for which you must specify values.

Table 15-3 getTasks() Method Parameters

Parameter	Description	Valid Values
assigneeId	ID of *assignee* (role or user) for whom to get tasks.	Valid user or role ID. For information about getting a list of current users or roles, see "Configuring the Security Realms" on page 9-1.
orgId	ID of the organization for which you want to get tasks.	String specifying a valid organization ID. For information about getting a list of all organization ids, see "Managing the Active Organization" on page 19-1.
isRole	Boolean flag specifying whether or not the assignee specified for the `assigneeId` parameter is a role or user.	`true` (role) or `false` (user)

This method returns an array of five elements, as defined in the following table.

Table 15-4 getTaskCounts() Method Array Elements

Element	Description
TASKCOUNT_TOTAL	Total number of tasks assigned to the user/role.

Table 15-4 getTaskCounts() Method Array Elements (Continued)

Element	Description
TASKCOUNT_PENDING	Number of assigned tasks that are pending.
TASKCOUNT_INACTIVE	Number of assigned tasks that are inactive.
TASKCOUNT_COMPLETED	Number of assigned tasks that are completed.
TASKCOUNT_OVERDUE	Number of assigned tasks that are overdue.

For example, the following code gets the task counts for the current user, and stores the counts in the `taskCounts[]` array. In this example, `admin` represents the `EJBObject` reference to the `Worklist` EJB.

```
int taskCounts[] = admin.getTaskCounts("ROLE1", "ORG1", true);
```

Task counts are stored in the elements fo the `taskCounts[]` array. For example, `taskCounts[TASKCOUNT_TOTAL]` specifies the task count total, `taskCounts[TASKCOUNT_PENDING]` specifies the pending task count total, and so on.

For more information about the `getTaskCounts()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Marking a Task Complete or Incomplete

To mark a task as complete (done) or incomplete (undone), use the following `com.bea.wlpi.server.admin.Admin` methods, respectively:

```
public void taskMarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public void taskUnmarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
```

```
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `taskMarkDone()` and `taskUnmarkDone()` method parameters for which you must specify values.

Table 15-5 taskMarkDone() and taskUnmarkDone() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition from which the workflow instance containing the task was instantiated.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTemplateDefinitionId()` For information about getting the `TaskInfo` object, see "Getting Tasks" on page 15-1. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.
instanceId	ID of the workflow instance containing the task.	String specifying a valid instance ID. To get the instance ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getInstanceId()` For information about getting the `TaskInfo` object, see "Getting Tasks" on page 15-1. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

Table 15-5 taskMarkDone() and taskUnmarkDone() Method Parameters (Continued)

Parameter	Description	Valid Values
taskId	ID of the task that you want to mark complete or incomplete.	String specifying a valid task ID. To get the task ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTaskId()` For information about getting the `TaskInfo` object, see "Getting Tasks" on page 15-1. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

The `taskMarkDone()` method sets the task `Completed` value to the current date and time, and results in the sequential execution of all the actions associated with the specified task's `Marked Done` event. The `taskMarkDone()` method, however, has no effect on a task that has already been marked complete. For information about defining the `Marked Done` event for a task, see "Working with Nodes" in "Defining Workflow Templates in *Using the WebLogic Integration Studio*.

The `taskUnmarkDone()` method clears the `Completed` date. The method *does not* result in the execution of the actions associated with the specified task's `Activated` event.

For example, the following code marks the specified task as complete, sets the `Completed` value to the current date and time, and executes the actions associated with the specified task's `Marked Done` event. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.taskMarkDone(
  task.getTemplateDefinitionId(),
  task.getInstanceId(),
  task.getTaskId()
);
```

The following code marks the same task as incomplete, and clears the `Completed` date:

```
admin.taskMarkUndone(
  task.getTemplateDefinitionId(),
```

```
  task.getInstanceId(),
  task.getTaskId()
);
```

The template definition, workflow instance, and task IDs are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The `task` object can be obtained using the methods described in "Getting Tasks" on page 15-1.

For more information about the `com.bea.wlpi.common.TaskInfo` methods, see "TaskInfo Object" on page B-20, or the `com.bea.wlpi.common.TaskInfo` Javadoc.

For more information about the `taskMarkDone()` and `taskMarkUndone()` methods, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Setting Task Properties

To set task properties, use following `com.bea.wlpi.server.admin.Admin` method:

```
public void taskSetProperties(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  int priority,
  boolean doneWithoutDoit,
  boolean doitIfDone,
  boolean unmarkDone,
  boolean modify,
  boolean reassign
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `taskSetProperties()` method parameters for which you must specify values.

Table 15-6 taskSetProperties() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition corresponding to the task.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTemplateDefinitionId()` For information about getting the `TaskInfo` object, see "Getting Tasks" on page 15-1. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.
instanceId	ID of the workflow instance corresponding to the task.	String specifying a valid instance ID. To get the instance ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getInstanceId()` For information about getting the `TaskInfo` object, see "Getting Tasks" on page 15-1. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

Table 15-6 taskSetProperties() Method Parameters (Continued)

Parameter	Description	Valid Values
taskId	ID of the task that you want to mark complete or incomplete.	String specifying a valid task ID. To get the task ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTaskId()` For information about getting the `TaskInfo` object, see "Getting Tasks" on page 15-1. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.
priority	Task priority.	`0`=low, `1`=medium, or `2`=high.
doneWithoutDoIt	Permission to mark task as complete using the methods described in "Marking a Task Complete or Incomplete" on page 15-9.	`true` (enabled) or `false` (disabled).
doitIfDone	Permission to execute a task after it has been marked as complete using the method described in "Executing a Task" on page 21-6.	`true` (enabled) or `false` (disabled).
unmarkDone	Permission to mark a task as incomplete using the methods described in "Marking a Task Complete or Incomplete" on page 15-9.	`true` (enabled) or `false` (disabled).
modify	Permission to modify the task run-time properties using the `taskSetProperties()` method.	`true` (enabled) or `false` (disabled).
reassign	Permission to reassign a task to another participant using the method described in "Assigning a Task" on page 15-3	`true` (enabled) or `false` (disabled).

For example, the following code sets the default task priority to medium (`1`) and enables (sets to `true`) all other task properties. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.taskSetProperties(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId(),
   1,
   true,
   true,
   true,
   true,
   true
);
```

The template definition, workflow instance, and task IDs are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The `task` object can be obtained using the methods described in "Getting Tasks" on page 15-1.

For more information about the `com.bea.wlpi.common.TaskInfo` methods, see "TaskInfo Object" on page B-20, or the `com.bea.wlpi.common.TaskInfo` Javadoc.

For more information about the `taskSetProperties()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

CHAPTER

16 Managing Task Routing

Tasks can be rerouted from one role or user to another for a specified period of time.

This section describes how to manage task routing, including the following topics:

- Adding a Task Reroute
- Getting Task Reroutes
- Updating a Task Reroute
- Deleting a Task Reroute
- Example of Managing Task Routing

For more information about the methods described in this section, see the `com.bea.wlpi.server.admin.Admin` Javadoc. For information about managing task routing using the WebLogic Integration Studio, see "Administering Task Routings" in "Administering Data" in *Using the WebLogic Integration Studio*.

Adding a Task Reroute

To add a task reroute, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.lang.String addReroute(
  java.lang.String orgId,
  java.lang.String from,
  java.lang.String to,
  int type,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the parameters for the `addReroute()` method for which you must specify values.

Table 16-1 addReroute() Method Parameters

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to reroute a task.	String specifying a valid organization ID. For information about getting a list of organization IDs, see "Getting All Organizations" on page 9-9.
from	ID of the *assignee* (role or user) for whom you want to reroute a task.	String specifying a valid role or user. For information about getting a list of current users or roles, see "Configuring the Security Realms" on page 9-1.
to	ID of the *assignee* (role or user) to whom you want to reroute a task.	String specifying a valid role or user. For information about getting a list of current users or roles, see "Configuring the Security Realms" on page 9-1.

Table 16-1 addReroute() Method Parameters (Continued)

Parameter	Description	Valid Values
type	Type of reroute.	Integer specifying one of the following static variable values: ■ TYPE_USER: Reroute tasks to a user. ■ TYPE_USERINROLE: Reroute tasks to a user in a role, using load-balancing. ■ TYPE_ROLE: Reroute tasks to a role. The static variable values are defined as part of the com.bea.wlpi.common.RerouteInfo object:
effective	Date and time that the reroute becomes effective.	Valid java.sql.Timestamp object.
expiry	Date and time that the reroute expires.	Valid java.sql.Timestamp object.

This method returns the ID of the new reroute object.

For example, the following code adds a reroute from user joe to user mary. In this example, admin represents the EJBObject reference to the Admin EJB.

```
List rerouteId = admin.addReroute( "ORG1", "joe", "mary",
  TYPE_USER, tsEffective, tsExpiry);
```

The *type* parameter is set to TYPE_USER, indicating that the task is being rerouted to a user. The reroute becomes effective on the date and time specified by the tsEffective timestamp and expires on the date and time specified by the tsExpiry timestamp.

For more information about the addReroute() method, see the com.bea.wlpi.server.admin.Admin Javadoc.

Getting Task Reroutes

To get task reroutes that are defined within an organization, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List getReroutes(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the parameter for the `getReroutes()` method for which you must specify a value.

Table 16-2 getReroutes() Method Parameter

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to get reroutes.	String specifying a valid organization ID. For information about getting a list of organization IDs, see "Getting All Organizations" on page 9-9.

Each method returns a list of `com.bea.wlpi.common.TaskInfo` objects. To access information about each task, use the `TaskInfo` object methods described in "TaskInfo Object" on page B-20.

For example, the following code gets the rerouted tasks for the `ORG1` organization. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List taskList = admin.getReroutes("ORG1");
```

For more information about the `getReroutes()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Updating a Task Reroute

To update a task reroute, use one of the following `com.bea.wlpi.server.admin.Admin` methods.

Method 1

```
public void updateReroute(
  java.lang.String rerouteId,
  java.lang.String to,
  int type,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public void updateReroute(
  java.lang.String rerouteId,
  java.lang.String from,
  java.lang.String to,
  int type,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

These methods are identical, except that in the second case, you specify the `from` parameter to define the assignee for whom you want to reroute a task.

The following table describes the parameters for the `updateReroute()` methods for which you must specify values.

Table 16-3 updateReroute() Method Parameters

Parameter	Description	Valid Values
rerouteId	ID of the task reroute that you want to update.	String specifying a valid reroute ID. For information about getting a list of all task reroute IDs, see "Getting Task Reroutes" on page 16-4.
from	ID of the *assignee* (role or user) for whom you want to reroute a task.	String specifying a valid role or user. For information about getting a list of current users or roles, see "Configuring the Security Realms" on page 9-1.

Table 16-3 updateReroute() Method Parameters (Continued)

Parameter	Description	Valid Values
to	ID of the *assignee* (role or user) to whom you want to reroute a task.	String specifying a valid role or user. For information about getting a list of current users or roles, see "Configuring the Security Realms" on page 9-1.
type	Type of reroute.	Integer value specifying one of the following static variable values: ■ TYPE_USER: Reroute tasks to a user. ■ TYPE_USERINROLE: Reroute tasks to a user in a role, using load-balancing. ■ TYPE_ROLE: Reroute tasks to a role. The static variable values are defined as part of the com.bea.wlpi.common.RerouteInfo object:
effective	Date and time that the reroute becomes effective.	Valid java.sql.Timestamp object.
expiry	Date and time that the reroute expires.	Valid java.sql.Timestamp object.

For example, the following code updates the specified task reroute, rerouting the task from user Mary to joe. In this example, admin represents the EJBObject reference to the Admin EJB.

```
List rerouteId = admin.addReroute( "ORG1", "mary", "joe",
  TYPE_USER, tsEffective, tsExpiry);
```

The *type* parameter is set to TYPE_USER, indicating that the task is being rerouted to a user. The reroute becomes effective starting on the date and time specified by the tsEffective timestamp and expires on the date and time specified by the tsExpiry timestamp.

For more information about the updateReroutes() methods, see the com.bea.wlpi.server.admin.Admin Javadoc.

Deleting a Task Reroute

To delete a task reroute, use the following com.bea.wlpi.server.admin.Admin method:

```
public void deleteReroute(
  java.lang.String rerouteId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the deleteReroute() method parameter for which you must specify a value.

Table 16-4 deleteReroute() Method Parameter

Parameter	Description	Valid Values
rerouteId	ID of the task reroute that you want to delete.	String specifying a valid reroute ID. For information about getting a list of all task reroute IDs, see "Getting Task Reroutes" on page 16-4.

For example, the following code deletes the specified reroute. In this example, admin represents the EJBObject reference to the Admin EJB.

```
admin.deleteReroute(reroute.getId());
```

The reroute ID is obtained using the methods associated with the com.bea.wlpi.common.RerouteInfo object, reroute. The reroute object can be obtained using the methods described in "Getting Task Reroutes" on page 16-4.

For more information about the deleteReroute() method, see the com.bea.wlpi.server.admin.Admin Javadoc.

Example of Managing Task Routing

This section provides excerpts from the command-line Studio example showing how to manage task routing.

Note: For more information about the command-line Studio example, see "Command-Line Studio Example" on page 1-22.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Adding a Task Reroute
- Deleting a Task Reroute
- Getting Task Reroutes

Important lines of code are highlighted in **bold**. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* Display Tool Title */
System.out.print( "\n---      Command Line Studio v1.0      ---" );

/* Display the main menu and interact with user */
while( true ) {
/* Display the menu */
   System.out.println( "\n---             Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Templates" );
   System.out.println( "2) Task Reroutes" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/**
 * Method that interacts with the user to get all the required information
 * to illustrate the Public API Methods available in the Admin interface
 * that are related to WLPI Task Reroutes.
 */
public static void mngReroutes() {
```

```
int assigneeType;
List rerouteList;
RerouteInfo rerouteInfo;
String orgId;
String rerouteId;
String answer;
String fromAssignee;
String toAssignee;
Timestamp effectiveDate;
Timestamp expiryDate;

/* Create an input stream to communicate with the user */
BufferedReader stdIn = new BufferedReader( new InputStreamReader(
   System.in ) );

try {
   /* Display the menu and interact with user */
   while( true ) {
      /* Display the menu */
      System.out.println( "\n\n---          Task Routings          ---" );
      System.out.println( "\nEnter choice:" );
      System.out.println( "1) Add a Task Reroute" );
      System.out.println( "2) Delete a Task Reroute" );
      System.out.println( "3) List all Task Reroutes" );
      System.out.println( "B) Back to previous menu" );
      System.out.println( "Q) Quit" );
      System.out.print( ">> " );

      /* Get user's selection */
      String line = stdIn.readLine();

      /* User pressed enter without making a selection ? */
      if( line.equals( "" ) )
         continue;
         /* User entered more than one char ? */
      else if( line.length() > 1 ) {
         System.out.println( "*** Invalid selection" );
         continue;
      }

      /* Convert to uppercase and to char */
      char choice = line.toUpperCase().charAt( 0 );

      /* Process user's selection */
      switch( choice ) {
      .
      .
      .
```

Adding a Task Reroute

The following excerpt shows how to add a task reroute:

```
/* Add a Task Reroute */
case '1' :
   /* Get Organization ID where to add the reroute */
   if( ( orgId = askQuestion( "\nEnter Organization ID where to
      add the Reroute" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Prompt user for User ID to reroute from */
   if( ( fromAssignee = askQuestion( "Enter User to reroute from" ) )
      == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Prompt user for Assignee to reroute to */
   if( ( toAssignee = askQuestion( "Enter Assignee to reroute to" ) )
      == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Prompt user to determine if the assignee is a user */
   if( ( answer = askQuestion( "- Is this a user (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Parse the answer */
   if( answer.equals( "y" ) || answer.equals( "Y" ) )
      assigneeType = RerouteInfo.TYPE_USER;
   else {
      /* Prompt user to determine if the assignee is user in a role */
      if( ( answer = askQuestion( "- Reroute to user in role (y/n)?" )
         ) == null ) {
         /* User cancelled the operation */
         System.out.println( "*** Cancelled" );
         break;
```

```
   }

   /* Parse the answer */
   if( answer.equals( "y" ) || answer.equals( "Y" ) )
      assigneeType = RerouteInfo.TYPE_USERINROLE;
   else
      assigneeType = RerouteInfo.TYPE_ROLE;
}

/* Get Effective Date */
/* Note: The Effective Date is a Timestamp. To keep
 * things simple, I have decided to deal with dates only. */
if( ( answer = askQuestion( "Enter Effective Date (yyyy-mm-dd)" )
   ) == null ) {
   /* User cancelled the operation */
   System.out.println( "*** Cancelled" );
   break;
}

/* Convert to a timestamp */
effectiveDate = Timestamp.valueOf( answer + " 0:0:0.0" );

/* Get Expiry Date */
/* Note: The Expiry Date is a Timestamp. To keep things simple,
 *       I have decided to deal with dates only. */
if( ( answer = askQuestion( "Enter Expiry Date (yyyy-mm-dd)" ) )
   == null ) {
/* User cancelled the operation */
   System.out.println( "*** Cancelled" );
   break;
}

/* Convert to a timestamp */
expiryDate = Timestamp.valueOf( answer + " 0:0:0.0" );

try {
   /* WLPI Public API Method */
   /* Add the Task Reroute */
   admin.addReroute( orgId, fromAssignee, toAssignee,
      assigneeType, effectiveDate, expiryDate );

   /* Success (No exception trown) */
   System.out.println( "- Added" );
}
catch( Exception e ) {
   System.out.println( "*** Unable to add Task Reroute" );
   System.err.println( e );
}
```

```
   break;
   .
   .
   .
```

Deleting a Task Reroute

The following excerpt shows how to delete a task reroute:

```
/* Delete a Task Reroute */
case '2' :
   /* Get ID for the Task Reroute to delete */
   if( ( rerouteId = askQuestion( "\nEnter Task Reroute ID to delete"
      ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI Public API Method */
      /* Delete the Task Reroute */
      admin.deleteReroute( rerouteId );

      /* Success (No exception trown) */
      System.out.println( "- Deleted" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete Task Reroute" );
      System.err.println( e );
   }
   break;
```

Getting Task Reroutes

The following excerpt shows how to get tasks routes:

```
/* List all Task Reroutes */
case '3' :
```

```
/* Get Organization ID for the organization to query */
if( ( orgId = askQuestion( "\nEnter Organization ID to list Task
   Reroutes" ) ) == null ) {
   /* User cancelled the operation */
   System.out.println( "*** Cancelled" );
   break;
}
try {
   /* WLPI Public API Method */
   /* Retrieve all Task Reroutes */
   rerouteList = admin.getReroutes( orgId );

   /* Any Task Reroutes defined ? */
   if( rerouteList.size() == 0 )
      System.out.println( "\nNo Task Reroute defined" );
   else
      System.out.println( "\nDefined Task Reroutes:" );

   /* Process the list to display Task Reroutes and attributes */
   for( int i = 0; i < rerouteList.size(); i++ ) {
      /* Retrieve an element from the list */
      rerouteInfo = ( RerouteInfo )rerouteList.get( i );

      /* WLPI Public API Method */
      /* Retrieve and display the task reroute ID */
      System.out.println( "- ID:       " + rerouteInfo.getId() );

      /* WLPI Public API Method */
      /* Retrieve and display Assignee for whom the task will be
       * reassigned */
      System.out.println( "  From user: " + rerouteInfo.getFrom() );

      /* Retrieve and display Assignee to whom the rerouted task
       * will be assigned */
      System.out.print( "  To" );

      /* WLPI Public API Method */
      /* Retrieve reroute type */
      assigneeType = rerouteInfo.getType();

      if( assigneeType == RerouteInfo.TYPE_ROLE )
         System.out.print( " role: " );
      else if( assigneeType == RerouteInfo.TYPE_USER )
         System.out.print( " user: " );
      else if( assigneeType == RerouteInfo.TYPE_USERINROLE )
         System.out.print( " user in role: " );
      else
         System.out.print( ": " );
```

```
            /* WLPI Public API Method */
            /* Retrieve and display Assignee to whom the rerouted task
             * will be assigned */
            System.out.println( rerouteInfo.getTo() );

            /* WLPI Public API Method */
            /* Retrieve and display the period when this reroute
             * is valid */
            System.out.println( "  Valid From " +
               rerouteInfo.getEffective().toString() +
               " to " + rerouteInfo.getExpiry().toString() );
      }
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to retrieve Task Reroutes" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

CHAPTER

17 Managing the XML Repository

The XML repository provides a data storage facility for the business process components of WebLogic Integration.

This section explains how to manage the XML repository, including the following topics:

- Managing XML Repository Folders
- Managing XML Repository Entities
- Getting the EJB Environment Variable Values

For more information about the methods described in this section, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Managing XML Repository Folders

The following sections describe how to create, update, display, and delete XML repository folders and subfolders.

Creating a Folder or Subfolder

To create a new folder or subfolder in the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public com.bea.wlpi.repository.helper.RepositoryFolderInfo createFolder (
  java.lang.String type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  com.bea.eci.repository.helper.RepositoryFolderInfo parent
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `createFolder()` method parameters for which you must specify values.

Table 17-1 createFolder() Method Parameters

Parameter	Description	Valid Values
type	Type of the folder being created.	String specifying a valid type from `com.bea.eci.repository.helper.Types`. `ObjectFolder` is currently the only valid type.
name	Name of the folder being created.	Non-null string that is unique within the repository for all `ObjectFolder` objects.
desc	Description of the folder being created.	String, which can be set to null.
notes	Notes for the folder being created.	String, which can be set to null.
parent	Parent folder for the child being created.	A `RepositoryFolderInfo` object specifying an existing folder, or null, to create a folder at the top level.

This method returns a list of `com.bea.eci.repository.helper.RepositoryFolderInfo` objects. To access information about each folder, use the `RepositoryFolder` object methods described in "RepositoryFolderInfo Object" on page B-11.

For example, the following code creates a new folder named `inventory` within the folder `factoryA`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
RepositoryFolderInfo newFolder = xmlrepository.createFolder(
  ObjectFolder, inventory, "Inventory of factory items.",
  "This is a note.", "factoryA");
```

For more information about the `createFolder()` method, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Getting All Folders and Subfolders

To get a list of *all* folders and subfolders in the XML repository or a list of subfolders corresponding to a particular folder, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` methods, respectively:

```
public java.util.List getAllFolders(
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

```
public java.util.List getChildFolders(
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `getChildFolders()` method parameter for which you must specify a value.

Table 17-2 getChildFolders() Method Parameter

Parameter	Description	Valid Values
rfi	Folder for which you want to view all subfolders.	An existing `RepositoryFolderInfo` object. If set to null, all folders that are *not* subfolders (that is, those that exist at the top level of the folder hierarchy) are retrieved. For information about getting a list of folders, see "Getting All Folders and Subfolders" on page 17-3.

Each method returns a list of `com.bea.eci.repository.helper.RepositoryFolderInfo` objects, or an empty list, if none are defined. To access information about each folder, use the `RepositoryFolder` object methods described in "RepositoryFolderInfo Object" on page B-11.

For example, the following code gets a list of all folders. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
List folders = xmlrepository.getAllFolders();
```

The following code gets a list of all subfolders for the specified parent, `folderA`:

```
List subfolders = xmlrepository.getChildFolders("folderA");
```

For more information about the `getAllFolders()` and `getChildFolders()` methods, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Getting Folder Tree

To get a tree structure of all the folders in the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public javax.swing.tree.DefaultMutableTreeNode getObjectFolderTree(
  javax.swing.tree.DefaultMutableTreeNode node
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `getObjectFolderTree()` method parameter for which you must specify a value.

Table 17-3 getObjectFolderTree() Method Parameter

Parameter	Description	Valid Values
node	Root node of the XML repository.	A `DefaultMutableTreeNode` object that represents the root of the XML repository.

This method returns a javax.swing.tree.DefaultMutableTreeNode representing the top node of the tree. Child folders of the specified node become the top-level folders in the tree. These folders are recursively populated with their respective children. Each child node consists of a DefaultMutableTreeNode object.

For example, the following code builds a tree from the root node. root represents a DefaultMutableTreeNode that you want to appear as the root of the tree. In this example, xmlrepository represents the EJBObject reference to the XMLRepository EJB.

```
javax.swing.tree.DefaultMutableTreeNode node =
   xmlrepository.getObjectFolderTree(root);
```

For more information about the getObjectFolderTree method, see the com.bea.eci.repository.ejb.XMLRepository Javadoc.

Getting Folder Information

To get information for a folder in the XML repository, use one of the following com.bea.wlpi.eci.repository.ejb.XMLRepository methods.

Method 1

```
public com.bea.wlpi.eci.repository.helper.RepositoryFolderInfo
getFolderInfo(
  java.lang.String name
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

Method 2

```
public com.bea.wlpi.eci.repository.helper.RepositoryFolderInfo
getFolderInfo(
  java.lang.String type,
  java.lang.String name
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the getFolderInfo() method parameters for which you must specify values.

Table 17-4 getFolderInfo() Method Parameters

Parameter	Description	Valid Values
type	Type of the folder for which you want to get information.	String specifying a valid type from com.bea.eci.repository.helper.Types. ObjectFolder is currently the only valid type.
name	Name of the folder for which you want to get information.	An existing ObjectFolder object.

Each method returns a com.bea.eci.repository.helper.RepositoryFolderInfo object, or null, if the specified folder is not defined. To access information about the folder, use the RepositoryFolder object methods described in "RepositoryFolderInfo Object" on page B-11.

For example, the following code gets the folder information for folderA. In this example, xmlrepository represents the EJBObject reference to the XMLRepository EJB.

```
com.bea.eci.repository.helper.RepositoryFolderInfo folderInfo =
  xmlrepository.getFolderInfo("folderA");
```

For more information about the getFolderInfo() methods, see the com.bea.eci.repository.ejb.XMLRepository Javadoc.

Reorganizing Folders

To reorganize folders within the XML repository, use the following com.bea.wlpi.eci.repository.ejb.XMLRepository methods:

```
public void addChildFolder(
  com.bea.eci.repository.helper.RepositoryFolderInfo child,
  com.bea.eci.repository.helper.RepositoryFolderInfo parent
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

```
public void removeChildFolder(
  com.bea.eci.repository.helper.RepositoryFolderInfo child,
  com.bea.eci.repository.helper.RepositoryFolderInfo parent
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The first method adds a child folder to a parent folder. The second removes a child folder from a parent folder.

Note: When a folder is removed from a parent, it is not also deleted from the XML repository.

The following table describes the `addChildFolder()` and `removeChildFolder()` method parameters for which you must specify values.

Table 17-5 addChildFolder() and removeChildFolder() Method Parameters

Parameter	Description	Valid Values
child	Folder to be added to or removed from the specified parent folder.	An existing `RepositoryFolderInfo` object. For information about getting a list of folders, see "Getting All Folders and Subfolders" on page 17-3.
parent	Parent folder.	An existing `RepositoryFolderInfo` object. For information about getting a list of folders, see "Getting All Folders and Subfolders" on page 17-3.

For example, the following code moves `folderA` into `folderB`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
xmlrepository.addChildFolder("folderA", "folderB");
```

Similarly, the following code removes `folderA` from `folderB`:

```
xmlrepository.removeChildFolder("folderA", "folderB");
```

For more information about the `addChildFolder()` and `removeChildFolder()` methods, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Renaming a Folder

To rename a folder in the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public void renameFolder(
  java.lang.String curname,
  java.lang.String newName
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `renameFolder()` method parameters for which you must specify values.

Table 17-6 renameFolder() Method Parameters

Parameter	Description	Valid Values
curName	Current name of the folder.	String specifying the name of an existing folder.
newName	New name for the folder.	String specifying a unique name for the new folder.

For example, the following code renames the existing folder, `folderA`, as `folderB`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
xmlrepository.renameFolder(folderA, folderB);
```

For more information about the `renameFolder()` method, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Updating a Folder

To update the description and/or notes fields for a folder in the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public void updateFolder(
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `updateFolder()` method parameter for which you must specify a value.

Table 17-7 updateFolder() Method Parameter

Parameter	Description	Valid Values
rfi	Folder name and type, and information to be updated.	A `RepositoryFolderInfo` object with the `name` and `type` fields set to identify the existing folder that you want to update, and the `description` and `notes` fields set to the new information.

For example, the following code updates the `description` and `notes` fields of an existing folder, as defined by the `RepositoryFolderInfo` object, `folderInfo`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
xmlrepository.updateFolder(folderInfo);
```

For more information about the `updateFolder()` method, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Deleting a Folder

To delete a folder from the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public void deleteFolder(
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `deleteFolder()` method parameter for which you must specify a value.

Table 17-8 deleteFolder() Method Parameter

Parameter	Description	Valid Values
rfi	Folder to be deleted.	An existing `RepositoryFolderInfo` object associated with the folder to be deleted. The specified folder must not contain any subfolders. For information about getting folder information, see "Getting Folder Information" on page 17-5.

XML entities contained within the folder being deleted *are not* deleted from the XML repository. XML entities become entities at the top level of the hierarchy.

For example, the following code deletes the folder defined by the `RepositoryFolderInfo` object, `folderInfo`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
xmlrepository.deleteFolder(folderInfo);
```

For more information about the `deleteFolder()` method, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Managing XML Repository Entities

The following sections describe how to create, update, display, and delete XML repository entities.

Creating an Entity

To create an XML repository entity, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public com.bea.eci.repository.helper.XMLEntityInfo createEntity(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  java.lang.String content,
  com.bea.eci.repository.helper.RepositoryFolderInfo parent
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `createEntity()` method parameters for which you must specify values.

Table 17-9 createEntity() Method Parameters

Parameter	Description	Valid Values
type	Type of the entity being created.	Integer specifying a valid type from `com.bea.eci.repository.helper.Types`. Valid values include: ■ `NO_CONTENT`: No content (default) ■ `TEXT`: Text ■ `DTD`: Document Type Definition (DTD) ■ `XML_SCHEMA`: XML schema ■ `XML`: XML ■ `XSLT`: XML Transformation Document (XSLT) ■ `MFL`: Message Format Language (MFL)
name	Name of the entity being created.	Non-null string that is unique within the repository for all `XMLEntityInfo` objects.
desc	Description of the entity being created.	String, which can be set to null.
notes	Notes for the entity being created.	String, which can be set to null.

Table 17-9 createEntity() Method Parameters (Continued)

Parameter	Description	Valid Values
parent	Parent folder for the entity being created.	A `RepositoryFolderInfo` object. If set to null, an entity is created at the top level. For information about getting folder information, see "Getting Folder Information" on page 17-5.

This method returns the `com.bea.eci.repository.helper.XMLEntityInfo` object corresponding to the new entity. To access information about the entity, use the `XMLEntityInfo` object methods described in "XMLEntityInfo Object" on page B-32.

For example, the following code creates a new XML entity named `widgets` in the `inventory` folder. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
XMLEntityInfo entity = xmlrepository.createEntity(Types.XML,
"widgets", "widgets inventory", "This is a note.", inventory);
```

For more information about the `createEntity()` method, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Getting Entities

To get a list of *all* entities in the XML repository, or a list of entities within a specified folder, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` methods, respectively:

```
public java.util.List getAllEntities(
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException

public java.util.List getChildDocs(
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `getChildDocs()` method parameter for which you must specify a value.

Table 17-10 getChildDocs() Method Parameter

Parameter	Description	Valid Values
rfi	Folder for which entities should be listed.	A `RepositoryFolderInfo` object associated with the folder for which entities should be listed. If this value is set to null, all entities *not* contained in a folder are retrieved. For information about getting folder information, see "Getting Folder Information" on page 17-5.

Each method returns a list of `com.bea.eci.repository.helper.XMLEntityInfo` objects, or an empty list, if no entities are defined. The `XMLEntityInfo` objects do not contain the entity object contents; they contain only the name, type, and meta-data. To access information about each entity, use the `XMLEntityInfo` object methods described in "XMLEntityInfo Object" on page B-32.

For example, the following code gets a list of all entities. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
List entities = xmlrepository.getAllEntities();
```

The following code gets a list of all entities within `folderA`:

```
List entities = xmlrepository.getChildDocs("folderA");
```

For more information about the `getAllEntities()` and `getChildDocs()` methods, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Getting Entity Information

To get information for an entity in the XML repository, use one of the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` methods.

Method 1

```
public com.bea.wlpi.eci.repository.helper.XMLEntityInfo getEntity(
  java.lang.String name
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

Method 2

```
public com.bea.wlpi.eci.repository.helper.XMLEntityInfo getEntity(
  java.lang.String name,
  int type
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The first method supports entities with single-content fields. If multiple-content fields exist, the method returns only the contents of the first type encountered. The second method supports entities with multiple-content fields: it allows you to specify the type of content that you want to retrieve.

The following table describes the `getEntity()` method parameters for which you must specify values.

Table 17-11 getEntity() Method Parameters

Parameter	Description	Valid Values
name	Name of the entity for which you want to get information.	String specifying the name of a valid `XMLEntityInfo` object.
type	Type of the entity for which you want to get information.	Integer specifying a valid type from `com.bea.eci.repository.helper.Types`. Valid values include: ■ `NO_CONTENT`: No content (default) ■ `TEXT`: Text ■ `DTD`: Document Type Definition (DTD) ■ `XML_SCHEMA`: XML schema ■ `XML`: XML ■ `XSLT`: XML Transformation Document (XSLT) ■ `MFL`: Message Format Language (MFL)

Each method returns a `com.bea.eci.repository.helper.XMLEntityInfo` object, or null, if the specified entity is not defined. To access information about the entity, use the `XMLEntityInfo` object methods described in "XMLEntityInfo Object" on page B-32.

For example, the following code gets entity information for `entityA`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
com.bea.eci.repository.helper.XMLEntityInfo entityInfo =
  xmlrepository.getEntity("entityA");
```

For more information about the `getEntity()` methods, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Organizing Entities Within Folders

To organize entities within folders in the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` methods:

```
public void addEntityToFolder(
  com.bea.eci.repository.helper.XMLEntityInfo xei,
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException

public void removeEntityFromFolder(
  com.bea.eci.repository.helper.XMLEntityInfo xei,
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The first method adds an entity to a folder. The second method removes an entity from a folder.

Note: When an entity is removed from a folder, it is not also deleted from the XML repository.

The following table describes the `addEntityToFolder()` and `removeEntityFromFolder()` method parameters for which you must specify values.

Table 17-12 addEntityToFolder() and removeEntityFromFolder() Method Parameters

Parameter	Description	Valid Values
xei	Entity to be added to or removed from the specified parent folder.	An existing `XMLEntityInfo` object. For information about getting a list of entities, see "Getting Entities" on page 17-12.
rfi	Parent folder.	An existing `RepositoryFolderInfo` object. For information about getting a list of folders, see "Getting All Folders and Subfolders" on page 17-3.

For example, the following code moves `entityA` into `folderA`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
xmlrepository.addEntityToFolder("entityA", "folderA");
```

Similarly, the following code removes `entityA` from `folderA`:

```
xmlrepository.removeEntityFromFolder("entityA", "folderA");
```

For more information about the `addEntityToFolder()` and `removeEntityFromFolder()` methods, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Renaming an Entity

To rename an entity in the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public void renameEntity(
  java.lang.String curname,
  java.lang.String newName
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `renameEntity()` method parameters for which you must specify values.

Table 17-13 renameEntity() Method Parameters

Parameter	Description	Valid Values
curName	Current name of the entity.	String specifying the name of an existing entity.
newName	New name for the entity.	String specifying a unique name for the new entity.

For example, the following code renames the existing entity, `entityA`, as `entityB`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
xmlrepository.renameEntity(entityA, entityB);
```

For more information about the `renameEntity()` method, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Updating an Entity

To update the description, notes, and/or contents fields for an entity in the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public void updateEntity(
  com.bea.eci.repository.helper.XMLEntityInfo xei
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `updateEntity()` method parameter for which you must specify a value.

Table 17-14 updateEntity() Method Parameters

Parameter	Description	Valid Values
xei	Entity and information to be updated.	An `XMLRepositoryInfo` object with the `name` and `type` fields set to identify the existing entity that you want to update, and the `description`, `notes`, and `contents` fields set to the new information.

You can update only a single-contents field, corresponding to the type specified in the `type` field of the `XMLRepositoryInfo` object, regardless of whether or not the entity has multiple content types.

For example, the following code updates the `description`, `notes`, and `contents` fields of an existing entity, as defined by the `XMLRepository` object, `entityInfo`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
xmlrepository.updateEntity(entityInfo);
```

For more information about the `updateEntity()` method, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Deleting an Entity

To delete an entity from the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public void deleteEntity(
  com.bea.eci.repository.helper.XMLEntityInfo xei
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

The following table describes the `deleteEntity()` method parameter for which you must specify a value.

Table 17-15 deleteEntity() Method Parameters

Parameter	Description	Valid Values
xei	Entity to be deleted.	An existing `XMLEntityInfo` object associated with the entity to be deleted. For information about getting entity information, see "Getting Entity Information" on page 17-13. The only required field in the `XMLEntityInfo` object is the `name` field, which uniquely identifies the entity.

For example, the following code deletes the folder defined by the `XMLEntityInfo` object, `entityInfo`. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
xmlrepository.deleteEntity(entityInfo);
```

For more information about the `deleteEntity()` method, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

Getting the EJB Environment Variable Values

To get a list of the EJB environment variable values used for connections to the XML repository, use the following `com.bea.wlpi.eci.repository.ejb.XMLRepository` method:

```
public java.util.List getEnvVars(
) throws java.rmi.RemoteException
```

This method returns the following list of environment variable values used to connect to the repository, in the order specified:

- Connection
- Transactional data source name
- JDBC class
- JDBC URL
- JDBC server

For example, the following code gets a list of the environment variables being used. In this example, `xmlrepository` represents the `EJBObject` reference to the `XMLRepository` EJB.

```
List envVars = xmlrepository.getEnvVars();
```

For more information about the `getEnvVars()` method, see the `com.bea.eci.repository.ejb.XMLRepository` Javadoc.

CHAPTER

18 Publishing Workflow Objects

This section describes publishable objects, and explains how to publish workflow objects, including the following topics:

- What Is a Publishable Object?
- Creating a Package Entry
- Exporting a Package of Publishable Objects
- Importing a Package of Publishable Objects
- Reading a Package of Publishable Objects

For more information about the methods described in this section, see the `com.bea.wlpi.server.admin.Admin` Javadoc. For information about publishing workflow objects using the WebLogic Integration Studio, see "Defining Workflow Templates" in *Using the WebLogic Integration Studio*.

What Is a Publishable Object?

A publishable object consists of any object that implements the `com.bea.wlpi.common.Publishable` interface. The `Publishable` interface provides support for creating, exporting, and importing packages.

Using the WebLogic Integration Studio or a custom design client, you can create, export, or import workflow objects that are valid *publishable* objects.

The following table lists the publishable objects defined by the `Publishable` interface, along with their corresponding types.

Table 18-1 Publishable Objects and Types

Publishable Object	Type
`BusinessCalendarInfo`	`BUSINESS_CALENDAR`
`ClassInvocationDescriptor`	`BUSINESS_OPERATION`
`EJBInvocationDescriptor`	`BUSINESS_OPERATION`
`EventKeyInfo`	`EVENT_KEY`
`OrganizationInfo`	`ORG`
`RepositoryFolderInfoHelper`	`XML_REPOSITORY_FOLDER`
`RoleInfo`	`ROLE`
`TemplateDefinitionInfo`	`TEMPLATE_DEFINITION`
`TemplateInfo`	`TEMPLATE`
`XMLEntityInfoHelper`	`XML_REPOSITORY_ENTITY`
`UserInfo`	`USER`

The following table lists the methods, provided by the `Publishable` interface, that can be used to get publishable object information.

Table 18-2 Publishable Interface Methods

Method	Description
`public java.lang.Object getContents()`	Gets the publishable object contents.
`public java.lang.String getEntryName()`	Gets the publishable object entry name.
`public java.lang.String getOwnerName()`	Gets the publishable object owner name.
`public java.util.List getReferencedPublishables( java.util.Map publishables )`	Gets all referenced publishables based on the specified publishables map object.
`public int getType()`	Gets publishable object type. For a list of valid types, see the table "Publishable Objects and Types" on page 18-2.

For more information, see the `com.bea.wlpi.common.Publishable` Javadoc.

Creating a Package Entry

To create a package entry in preparation of exporting, you must create a `com.bea.wlpi.common.PackageEntry` object.

The following example provides the constructor for creating a new `PackageEntry` object:

```
public PackageEntry(
  com.bea.wlpi.common.Publishable p,
  java.util.Map r,
  boolean b
)
```

The following table describes the `PackageEntry` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Table 18-3 PackageEntry Object Data

Object Data	Constructor Parameter	Get Method	Set Method
Publishable object For a list of valid publishable objects, see "What Is a Publishable Object?" on page 18-2.	*p*	`public com.bea.wlpi.common.Publishable getPublishable()`	`public void setPublishable(com.bea.wlpi.common.Publishable p)`
Referenced publishable objects For information about getting publishable object data, see the table "Publishable Interface Methods" on page 18-3.	*r*	`public java.util.Map getReferences()`	`public void setReferences(java.util.Map r)`
Boolean flag specifying whether or not the publishable object should be locked	*b*	`public boolean getPublished()`	`public void setPublished(boolean b)`
Entry type For a list of valid types, see the table "Publishable Objects and Types" on page 18-2.	`N/A`	`public java.lang.String getEntryType()`	N/A

For more information, see the `com.bea.wlpi.common.PackageEntry` Javadoc.

Exporting a Package of Publishable Objects

To export a package of publishable objects to a JAR file, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public byte[] exportPackage(
  com.bea.wlpi.common.PublishPackage publishables,
  java.lang.Object credential
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `exportPackage()` method parameters for which you must specify values.

Table 18-4 exportPackage() Method Parameters

Parameter	Description	Valid Values
publishables	Publishable workflow objects to be exported.	An object of type `com.bea.wlpi.common.PublishPackage` containing the entries of type `com.bea.wlpi.common.PackageEntry` to be exported into the Jar package. For a list of valid publishable types, see the table "Publishable Objects and Types" on page 18-2.
credential	Password value, if required.	A `java.lang.Object` object specifying a valid password, or null, if a password is not required.

This method returns a `byte[]` array that is an image of a JAR file containing all exported objects.

For example, the following code exports a package of publishable objects specified by the previously defined `publishable` Map object. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
byte[] exportResults = admin.exportPackage(publishableMap);
```

For more information about the `exportPackage()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Importing a Package of Publishable Objects

To import a package of publishable objects, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.lang.String importPackage(
  byte[] pkg,
  java.util.Map publishables,
  java.lang.String orgId,
  boolean activate,
  java.lang.Object credential
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `importPackage()` method parameters for which you must specify values.

Table 18-5 importPackage() Method Parameters

Parameter	Description	Valid Values
pkg	JAR file of publishable objects to be imported.	A valid `byte[]` object consisting of an image of a JAR file containing publishable objects. For information about exporting a package of publishable objects, see "Exporting a Package of Publishable Objects" on page 18-5.
publishables	Publishable workflow objects to be imported.	A `java.util.Map` object with *key-value* pairs, specifying the publishable type as the key and the `PackageEntry` object as the value: For a list of valid publishable types, see the table "Publishable Objects and Types" on page 18-2.

Table 18-5 importPackage() Method Parameters (Continued)

Parameter	Description	Valid Values
orgId	ID of the organization to which the package is being imported.	String specifying a valid organization ID. For information about getting a list of organization IDs, see "Getting All Organizations" on page 9-9.
activate	Boolean flag specifying whether or not the package should be activated on import.	`true` (activate) or `false` (do not activate).
credential	Password value, if required.	A `java.lang.Object` object specifying a valid password, or null if a password is not required.

This method returns a string value containing any reported issues, such as unresolved references.

For example, the following code imports the publishable objects to the `ORG1` organization as specified in the `shipping.jar` file and `publishPackage` map. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB:

```
//read JAR file into a byte array
File jarFile = new File("shipping.jar");
pkg = new byte[(int)jarFile.length()];
FileInputStream fin = new FileInputStream(jarFile);
fin.read(pkg);
fin.close();
//Map and import the contents
Map contents = admin.readPackage(pkg, "password");
String importResults = admin.importPackage(
  pkg, contents, "ORG1", true, "password");
```

The publishable objects are activated upon import (the activate value is set to `true`), and the password used is `abcd`.

For more information about the `importPackage()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Reading a Package of Publishable Objects

To read a package of publishable objects, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.Map readPackage(
  byte[] pkg,
  java.lang.Object credential
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `readPackage()` method parameters for which you must specify values.

Table 18-6 readPackage() Method Parameters

Parameter	Description	Valid Values
pkg	JAR file of publishable objects to be read.	A valid `byte[]` object consisting of an image of a JAR file containing publishable objects. For information about exporting a package of publishable objects, see "Exporting a Package of Publishable Objects" on page 18-5.
credential	Password value, if required.	A `java.lang.Object` object specifying a valid password, or null, if a password is not required.

This method returns a `java.util.Map` object with *key-value* pairs, specifying the publishable types as the key, and a hetereogenous `java.util.List` object containing the *xxx*Info object corresponding to the publishable type, as the value. For a list of valid publishable objects and types, see the table "Publishable Objects and Types" on page 18-2.

For example, the following code reads the publishable objects specified in the `shipping.jar` file, using the password `abcd`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
byte[] pkg = readFile("shipping.jar");
Map contents = admin.readPackage(pkg, "abcd");
```

For more information about the `readPackage()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Part IV Run-Time Management

CHAPTER

19 Managing the Active Organization

This section explains how to manage the active organization, including the following topics:

- What Is an Active Organization?
- Getting the Active Organization
- Getting All Organizations
- Setting the Active Organization
- Example of Managing the Active Organization

For information about managing the active organization using the WebLogic Integration Worklist, see "Working with Workflows" in *Using the WebLogic Integration Worklist*.

Note: The Worklist client application is being deprecated as of this release of WebLogic Integration. For information about the features that are replacing it, see the *BEA WebLogic Integration Release Notes*.

What Is an Active Organization?

Organizations are defined, using either the WebLogic Integration Studio or a custom configuration client, to represent different business entities, geographical locations, or any other class of entities that is relevant to the particular business of the company. You can also define users and roles within organizations, to further fortify the permissions that provide security.

The current active organization is the default organization for both the Worklist and/or custom run-time management client requests in which an organization is not specified. The active organization is initially set to the default organization of the user who invoked the run-time management client. The default organization is specified when the user is initially created using the definition client.

You can get and set the active organization using the methods described in the following sections.

For more information about any of the methods described in the following sections, see the *BEA WebLogic Integration Javadoc*. For more information about maintaining organizations within a Studio client, see "Maintaining Organizations" in "Administering Data" in *Using the WebLogic Integration Studio*.

Getting the Active Organization

To get the currently active organization, use the following `com.bea.wlpi.server.worklist.Worklist` method:

```
public java.lang.String getActiveOrganization(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

This method returns the ID of the active organization.

For example, the following code gets the active organization ID and saves it to the `activeOrgId` string variable:

```
String activeOrgId = worklist.getActiveOrganization();
```

In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB. For more information about the `getActiveOrganization()` method, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Getting All Organizations

In order to set the active organization, you need to know the ID of the organization that you want to set as active. To get a list of all defined organization IDs, see "Getting All Organizations" on page 9-9.

Setting the Active Organization

To set the active organization, use the following `com.bea.wlpi.server.worklist.Worklist` method:

```
public void setActiveOrganization(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `setActiveOrganization()` method parameter for which you must specify a value.

Table 19-1 setActiveOrganization() Method Parameter

Parameter	Description	Valid Values
orgId	ID of the organization that you want to set as active.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Getting All Organizations" on page 9-9.

For example, the following code sets the active organization to `ORG1`. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
worklist.setActiveOrganization("ORG1");
```

For more information about the `setActiveOrganization()` method, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Example of Managing the Active Organization

This section provides excerpts from the command-line worklist example showing how to manage the active organization.

Note: For more information about the command-line worklist example, see "Command-Line Worklist Example" on page 1-23.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Getting the Active Organization
- Getting All Organizations
- Setting the Active Organization

Important lines of code are highlighted in **bold**. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) )

/* Display Tool Title */
System.out.print( "\n---  Command Line Worklist v1.2  ---" );

/* Display the main menu and interact with user */
while( true ) {
   /* Display the menu */
   System.out.println( "\n---           Main Menu          ---" );
   System.out.println( "\nEnter choice:" );
```

```
   System.out.println( "1) Organizations" );
   System.out.println( "2) Workflows" );
   System.out.println( "3) Tasks" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
public static void mngOrganization( ) {
   String orgId;

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* Display the main menu and interact with user */
      while( true ) {
         /* Display the menu */
         System.out.println( "\n\n---          Organizations         ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) List active organization" );
         System.out.println( "2) List all organizations" );
         System.out.println( "3) Set active organization" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* Get the user's selection */
         String line = stdIn.readLine( );

         /* User pressed enter without making a selection ? */
         if( line.equals( "" ) )
            continue;
         /* User entered more than one char ? */
         else if( line.length( ) > 1 ) {
            System.out.println( "*** Invalid choice" );
            continue;
         }

         /* Convert to uppercase and to char */
         char choice = line.toUpperCase( ).charAt( 0 );

         /* Process user's selection */
         switch( choice ) { ...
```

Getting the Active Organization

The following excerpt shows how to get the name of the active organization:

```
/* List active organization */
case '1' :
   /* WLPI Public API Method */
   /* NOTE: Would be nice to add code to capture any
    * thrown exceptions */
   String activeOrgId = worklist.getActiveOrganization( );
   System.out.println( "\nActive organization is " + activeOrgId );
   break; ...
```

Getting All Organizations

The following excerpt shows how to get a list of all organizations:

```
/* List all organizations */
case '2' :
   /* WLPI Public API Method */
  /* NOTE: Would be nice to add code to capture any thrown exceptions */
   List orgList = principal.getAllOrganizations( false );

   /* Any organizations defined ? */
   if( orgList.size( ) == 0 )
      System.out.println( "\nNo Organization defined" );
   else
      System.out.println( "\nDefined organizations:" );

   /* Process the list to display organization and attributes */
   for( int i = 0; i < orgList.size( ); i++ ) {
      /* Retrieve an element from the list */
      OrganizationInfo orgInfo = ( OrganizationInfo )orgList.get( i );
      /* Retrieve and display organization id */
      System.out.println( "- ID: " + orgInfo.getOrgId( ) );
   }
   break; ...
```

Setting the Active Organization

The following excerpt shows how to set the active organization:

```
/* Set active organization */
case '3' :
   /* Get Organization ID for the organization to set as active */
   if( ( orgId = askQuestion( "\nEnter Organization ID" ) ) == null ) {
```

```
   /* User cancelled the operation */
   System.out.println( "*** Cancelled" );
   break;
}

try {
   /* WLPI Public API Method */
   worklist.setActiveOrganization( orgId );

   /* Success (No exception trown) */
   System.out.println( "- Success" );

   /* WLPI Public API Method */
   /* Confirm that the operation was succesful */
   activeOrgId = worklist.getActiveOrganization( );
   System.out.println(
      "- The active organization is now " + activeOrgId );
}
catch( Exception e ) {
  System.out.println(
      "*** Failed to set the Active Organization (ID: " +
      orgId + ")" );
   System.err.println( e );
}
break;
   .
   .
   .
```

CHAPTER

20 Manually Starting Workflows

This section describes how to start workflows manually, including the following topics:

- Getting Startable Workflows
- Manually Starting a Workflow
- Examples of Manually Starting a Workflow

Note: You can also automate the launch of a workflow using one of the following mechanisms: an external or timed event, or the Start Workflow action, `ActionStartWorkflow`, which starts one workflow from another. The latter method offers greater control because it allows business rules to be considered in determining whether a called workflow is ever started. This mechanism also enables you to set variables using the XML content, and to start multiple workflows. For more information about the actions available for starting workflows, see "Template Definition DTD" on page A-54, or "Defining Actions" in *Using the WebLogic Integration Worklist.*

For information about manually starting workflows using the WebLogic Integration Worklist, see "Working with Workflows" in *Using the WebLogic Integration Worklist*.

Note: The Worklist client application is being deprecated as of this release of WebLogic Integration. For information about the features that are replacing it, see the *BEA WebLogic Integration Release Notes*.

Getting Startable Workflows

To get a list of all the workflows that can be started manually from either the Worklist or a custom run-time management client, use the following `com.bea.wlpi.server.worklist.Worklist` method:

```
public java.util.List getStartableWorkflows(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getStartableWorkflows()` method parameter.

Table 20-1 getStartableWorkflows() Method Parameter

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to get startable workflows.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Managing the Active Organization" on page 19-1.

The method returns a list of `com.bea.wlpi.common.TemplateInfo` objects that:

- Correspond to the template definitions that can be started manually
- Are currently active

To access information about template definitions, use the `TemplateInfo` object methods described in "TemplateInfo Object" on page B-25.

For example, the following code gets a list of all startable workflows for the organization specified by the organization ID `ORG1`. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
List template = worklist.getStartableWorkflows("ORG1");
```

For more information about the `getStartableWorkflows()` method, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Manually Starting a Workflow

To start (or instantiate) a workflow manually, use one of the following `com.bea.wlpi.server.worklist.Worklist` methods.

Method 1

```
public java.lang.String instantiateWorkflow(
  java.lang.String orgId,
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public java.lang.String instantiateWorkflow(
  java.lang.String orgId,
  java.lang.String templateId,
  java.lang.String xml,
  java.util.Map initialValues,
  java.util.Map pluginData
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 3

```
public java.lang.String instantiateWorkflow(
  java.lang.String orgId,
  java.lang.String templateId,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 4

```
public java.lang.String instantiateWorkflow(
  java.lang.String orgId,
  java.lang.String templateId,
  java.lang.String xml,
  java.util.Map initialValues,
  java.util.Map pluginData,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The first method can be used in a nonclustered environment. The second method is recommended for use in a clustered environment. In this case, using the specified transaction ID, the system tracks the method execution status so that the method is not reissued after the transaction is committed, or in the event of a server crash or failover.

The following table describes the `instantiateWorkflow()` method parameters.

Table 20-2 instantiateWorkflow() Method Parameters

Parameter	Description	Valid Values
orgId	ID of the organization associated with the workflow that you want to start.	String specifying a valid organization ID. For information about getting a list of all organization IDs, see "Managing the Active Organization" on page 19-1.
templateId	ID of the workflow template for which you want to start a workflow.	String specifying a valid template ID. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final String getId()` The `getId()` method returns a `templateId`. Each `templateId` may have multiple definitions. To determine which template definition to use, WebLogic Integration selects, from the set of active template definitions, the version with the most relevant effective date (current or past) and expiration date (current or future). For information about defining the effective and expiration dates for the template definitions and making the template definitions active, see "Template Definition DTD" on page A-54, or *Using the WebLogic Integration Studio*. For information about getting the `TemplateInfo` object, see "Getting the Templates for an Organization" on page 13-5. For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25.
xml	Initialization values for the workflow instance.	XML document containing initialization values that are meaningful to the workflow instance, or null if no initialization values are required.
initialValues	Initial values for workflow variables.	Map object with *key-value* pairs, specifying variables to be initialized as a key, and the desired value as the value, or null, if no variables require initialization.

Table 20-2 instantiateWorkflow() Method Parameters (Continued)

Parameter	Description	Valid Values
pluginData	Plug-in instance data.	Map object with *key-value* pairs, specifying the plug-in name as the key, and the instance data as the value, or null, if no plug-data needs to be defined. For more information about programming BPM plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.
transactionId	ID of the transaction. **Note:** This parameter is required only in a clustered environment.	Object specifying a unique transaction ID. To generate a unique transaction ID, create a new `com.bea.wlpi.client.common.GUID` object using the following constructor: `GUID transactionId = new GUID();` For more information about the GUID class, see the `com.bea.wlpi.client.common.GUID` Javadoc.

In addition to starting the workflow template, the `instantiateWorkflow()` method performs the following tasks, if specified in the template definition, for every manual `Start` node listed in the template definition:

- Initializes variable values to specified constant values
- Executes specified actions
- Activates successors

For information about creating a template definition and defining tasks to be performed upon instantiation, see “Creating and Managing Workflow Template Definitions” on page 14-1.

The `instantiateWorkflow()` methods return an XML document that is compliant with the Client Request DTD, and that contains information about the template definition instance, including the following:

- Template, template definition, and instance IDs
- Information about defined `Send XML to Client` actions

For more information about the Client Request DTD, see "Client Request DTD" on page A-34.

For example, the following code instantiates the template definition for the organization specified by the `activeOrgId` and `templateId` variables. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
String clientReq = worklist.instantiateWorkflow(activeOrgId,
templateId);
```

If multiple template definitions are represented by the same `templateId`, WebLogic Integration selects the version that is active and that has the most relevant effective and expiration dates.

For more information about the `instantiateWorkflows()` methods, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Examples of Manually Starting a Workflow

The following sections provide excerpts from the command-line and JSP worklist examples showing how to manually start a workflow.

Note: For more information about the command-line and JSP worklist examples, see "BPM API Examples" on page 1-21.

Command-Line Worklist Example

This section provides excerpts from the command-line worklist example showing how to start a workflow.

Note: For more information about the command-line worklist example, see "Command-Line Worklist Example" on page 1-23.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

- Getting Startable Workflows

- Manually Starting a Workflow

Important lines of code are highlighted in **bold**. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) )

/* Display Tool Title */
System.out.print( "\n---  Command Line Worklist v1.2  ---" );

/* Display the main menu and interact with user */
while( true ) {
   /* Display the menu */
   System.out.println( "\n---           Main Menu          ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Workflows" );
   System.out.println( "3) Tasks" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
public static void mngWorkflows( ) {
   String templateId;

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* Display the main menu and interact with user */
      while( true ) {
         /* Display the menu */
         System.out.println( "\n\n---           Workflows          ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) List Startable workflow templates" );
         System.out.println( "2) Start a workflow" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* Get the user's selection */
         String line = stdIn.readLine( );

         /* User pressed enter without making a selection ? */
         if( line.equals( "" ) )
            continue;
```

```
/* User entered more than one char ? */
else if( line.length( ) > 1 ) {
   System.out.println( "*** Invalid choice" );
   continue;
}

/* Convert to uppercase and to char */
char choice = line.toUpperCase( ).charAt( 0 );

/* Process user's selection */
switch( choice ) { ...
```

Getting Startable Workflows

The following excerpt shows how to get a list of startable workflows:

```
/* List Startable workflow templates */
case '1' :
   /* WLPI Public API Method */
   /* NOTE: Would be nice to add code to capture any
    * thrown exceptions */
   String activeOrgId = worklist.getActiveOrganization( );
   List templateList;

   /* WLPI Public API Method */
   /* NOTE: Would be nice to add code to capture any
    * thrown exceptions */
   templateList = worklist.getStartableWorkflows( activeOrgId );

   /* Any startable workflows ? */
   if( templateList.size( ) == 0 )
      System.out.println( "\nNo Startable Workflows." );
   else
      System.out.println( "\nStartable Workflows:" );

   /* Process the list to display startable workflows */
   for( int i = 0; i < templateList.size( ); i++ ) {
      /* Retrieve an element from the list */
      TemplateInfo templateInfo = (
         TemplateInfo )templateList.get( i );

      /* Retrieve and display Template id and name */
      System.out.println( "- ID: " + templateInfo.getId( ) +
         " Name: " + templateInfo.getName( ) );
   }
   break; ...
```

Manually Starting a Workflow

The following excerpt shows how to manually start a workflow:

```
         /* Instantiate a Workflow Template */
         case '2' :
            /* Get Template ID for the workflow to instantiate */
            if( ( templateId = askQuestion(
               "\nEnter Workflow Template ID" ) ) == null ) {
               /* User cancelled the operation */
               System.out.println( "*** Cancelled" );
               break;
            }

            try {
               /* WLPI Public API Method */
               /* Retrieve the Active Organization ID */
               activeOrgId = worklist.getActiveOrganization( );

               /* WLPI Public API Method */
               /* Instantiate the workflow */
               String clientReq = worklist.instantiateWorkflow(
                  activeOrgId, templateId );

               /* Success (No exception thrown) */
               System.out.println( "- Success" );
//             System.out.println( "- Client Request:\n" + clientReq );

               /* Parse the reply from the server to process client
                * request (if any) */
               parser.handleRequest( clientReq );
            }
            catch( Exception e ) {
               System.out.println(
               "*** Failed to instantiate Workflow Template (ID: " +
                  templateId + ")" );
               System.err.println( e );
            }
            break;

         /* Return to previous menu */
         case 'B' :
            return;

         /* Exit tool */
         case 'Q' :
            /* Disconnect from the server */
            disconnect( );
```

```
            System.exit( 1 );

         default:
            System.out.println( "*** Invalid choice" );
         }
      }
   }
   /* "Unhandled" exceptions */
   catch( Exception e ) {
      System.err.println( e );
   }
   return;
}
```

JSP Worklist Example

This section provides excerpts from the JSP worklist example showing how to start a workflow manually.

Important lines of code are highlighted in **bold**. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

As shown in the figure "Main Interface to the JSP Worklist" on page 1-25, a workflow is started when a user clicks on the `Start Workflow` link in the JSP worklist, and selects a startable workflow. The `Start Workflow` link references the `startworkflow.jsp` file, as follows:

```
<a href="startworkflow.jsp"><font color= "white">Start Workflow</font></a>
```

The following excerpt from the `startworkflow.jsp` file in the JSP worklist example shows how to start a workflow:

```
<%@ page import="
   javax.ejb.*,
   java.rmi.RemoteException,
   java.util.*,
   com.bea.wlpi.common.*
   com.bea.wlpi.server.worklist.*
" session="true"%>

<jsp:usebean id="responseParser" class="jsp_servlet._worklist.ResponseParser"
scope="session"/>

<%
   Worklist worklist = null;
```

```
   String name;
   String error = null;
   String currentOrg = (String)session.getValue("currentOrg");
   worklist = (Worklist)session.getValue("worklist");
   name = (String)session.getValue("name");

   if (worklist != null) {
      Hashtable h = new Hashtable();
      h.put(javax.naming.Context.SECURITY_PRINCIPAL, name);
      h.put(javax.naming.Context.SECURITY_CREDENTIALS,
      (String)session.getValue("password"));
      new javax.naming.InitialContext(h);
```

The `start` variable is used to store the template ID of the workflow template definition that the user wants to start. When the JSP is initially loaded, the `start` variable is set to `null`, and the start workflow `try` loop is skipped. Next, the startable workflows are listed. When the user selects a startable workflow, the `startworkflow.jsp` file is invoked with the `start` variable set to the value of the template ID, which is obtained from the `TemplateInfo` object. The resulting XML file (which is compliant with the Client Request DTD) is returned from the `instantiateWorkflow()` method call and passed to the response parser, `ResponseParser`, for parsing. For more information about the response parser, see "Assigning a Task" on page 21-51.

```
      /* Start Workflow if start parameter set */
      String start = request.getParameter("start");
      if (start != null)
         try {
            responseParser.parse(worklist.instantiateWorkflow(
               currentOrg, start));
            session.removeValue("error");
            if (responseParser.isQuery())
               pageContext.forward("query.jsp");
            else
               pageContext.forward("worklist.jsp");
            return;
         } catch (Exception e) {
         error = e.getMessage();
      } out.println(error);
%>
<html>
<title>eProcess Integrator</title>
<head>
</head>

<body bgcolor=#ffffff>
<font face="Arial" size="4">
```

```
<table dir="ltr" border="0" cellpadding="1" cellspacing="1">
   <tr>
      <td><img src="logo.gif"></td>
      <td>
         <font color="red" size="7"><strong><em>Weblogic</em></strong></font>
         <font color="black" size="7"><strong><em>Process Integrator
         </em></strong></font></td>
   </tr>
   <tr>
      <td bgcolor="navy" valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" >
         <tr>
            <td><font color="yellow"><strong>Go ...</strong></font></td>
         </tr>
         <tr>
            <td><a href="worklist.jsp?worklist">
            <font color="white">Worklist</font></a></td>
         </tr>
         <tr>
            <td><a href="logoff.jsp"><font color="white">Logoff</font></a></td>
         </tr>
      </table>
      </td>

      <td valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" valign="top">
         <tr>
            <td><font color="navy" size="5" face="Arial">
            Start Workflow</font></td>
         </tr>
```

The startable workflows are listed. When the user selects one, the `startworkflow.jsp` file is invoked with the `start` variable set to the value of the template ID obtained from the `TemplateInfo` object.

```
/* Get startable workflows and list them./*
<%
   try {
      List templates = worklist.getStartableWorkflows(currentOrg);
      for (int i = 0; i < templates.size(); i++) {
    TemplateInfo info = (TemplateInfo)templates.get(i);

%>
         <tr>
            <td><a href="startworkflow.jsp?start=<%=info.getId()%>">
            <%=info.getName()%></a></td>
         </tr>
<%
```

```
   }
   } catch (Exception e) {
   out.print("Error 3: " + e);
   }
   }
%>
      </table>
      </td>
   </tr>
</table>
</font>
</body>
</html>
```

CHAPTER

21 Managing Run-Time Tasks

Once you have started a workflow, as described in "Manually Starting Workflows" on page 20-1, you can manage the tasks associated with the running instance. This section describes how to manage run-time tasks, including the following topics:

- Getting a Task
- Getting All Tasks
- Getting Task Counts
- Executing a Task
- Responding to a Client Request
- Assigning a Task
- Marking a Task as Complete or Incomplete
- Setting Task Properties
- Updating an Instance Variable
- Invoking an Exception Handler
- Examples of Managing Run-Time Tasks

For information about managing run-time tasks using Worklist, see "Working with Tasks" in *Using the WebLogic Integration Worklist*.

Getting a Task

To get a task, use the following `com.bea.wlpi.server.worklist.Worklist` method:

```
public com.bea.wlpi.common.TaskInfo getTask(
       java.lang.String taskName,
       java.lang.String instanceId,
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the parameters for the `getTask()` method for which you must specify values.

Table 21-1 getTask() Method Parameters

Parameter	Description	Valid Values
taskName	Name of the task that you want to get.	String specifying a valid organization id. To get the task name, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTaskName()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

Table 21-1 getTask() Method Parameters (Continued)

Parameter	Description	Valid Values
instanceId	ID of the workflow instance associated with the task that you want to get.	String specifying a valid instance ID. To get the instance ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getInstanceId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

This method returns a `com.bea.wlpi.common.TaskInfo` object. To access information about the task, use the `TaskInfo` object methods described in "TaskInfo Object" on page B-20.

For example, the following code gets the `Task1` task for the specified workflow instance. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
TaskInfo task = worklist.getTask("Task1",instanceID);
```

For more information about the `getTask()` method, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Getting All Tasks

To get a list of tasks associated with a particular participant (user or role) and organization, use one of the following `com.bea.wlpi.server.worklist.Worklist` methods:

```
public java.util.List getTasks(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public java.util.List getTasks(
       java.lang.String orgId,
       java.lang.String assigneeId,
       boolean isRole
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The first method gets the tasks assigned to the user in the active organization that created the `EJBObject` reference to the `Worklist` EJB. The second method gets the tasks assigned to the specified organization and *assignee* (role or user).

The following table describes the parameters for the second `getTasks()` method for which you must specify values.

Table 21-2 getTasks() Method Parameters

Parameter	Description	Valid Values
orgId	ID of the organization for which you want to get tasks.	String specifying a valid organization ID. For information about getting a list of all organization ids, see "Managing the Active Organization" on page 19-1.
assigneeId	ID of *assignee* (role or user) for whom to get tasks.	Valid user or role ID. For information about getting a list of current users or roles, see "Configuring the Security Realms" on page 9-1.
isRole	Boolean flag specifying whether or not the assignee specified for the `assigneeId` parameter is a role or user.	`true` (role) or `false` (user)

Each method returns a list of com.bea.wlpi.common.TaskInfo objects. To access information about each task, use the TaskInfo object methods described in "TaskInfo Object" on page B-20.

For example, the following code gets the tasks for the current user in the active organization. In this example, worklist represents the EJBObject reference to the Worklist EJB.

```
List taskList = worklist.getTasks();
```

The following code gets a list of the tasks for the ROLE1 role in the ORG1 organization (as the isRole parameter is set to true). In this example, worklist represents the EJBObject reference to the Worklist EJB.

```
List taskList = worklist.getTasks("ORG1", "ROLE1", true);
```

For more information about the getTasks() methods, see the com.bea.wlpi.server.worklist.Worklist Javadoc.

Getting Task Counts

To get the number of assigned and overdue tasks for the current user, use the following com.bea.wlpi.server.worklist.Worklist method:

```
public int[] getTaskCounts(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The method returns a two-element array. The first element specifies the number of tasks assigned to the user; the second element specifies the number of those tasks that are overdue.

For example, the following code gets the counts for the assigned and overdue tasks for the current user, and stores the counts in the taskCounts[] array. In this example, worklist represents the EJBObject reference to the Worklist EJB.

```
int taskCounts[] = worklist.getTaskCounts();
```

Counts of assigned and overdue tasks are stored in the taskCounts[0] and taskCounts[1] elements, respectively.

For more information about the `getTaskCounts()` method, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Executing a Task

To execute a task, use one of the following `com.bea.wlpi.server.worklist.Worklist` methods:

```
public java.lang.String taskExecute(
       java.lang.String taskName,
       java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public java.lang.String taskExecute(
       java.lang.String templateDefinitionId,
       java.lang.String instanceId,
       java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `taskExecute()` method parameters for which you must specify values.

Table 21-3 taskExecute() Method Parameters

Parameter	Description	Valid Values
taskName	Name of the task that you want to execute.	String specifying a valid task name. To get the task name, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTaskName()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.
templateDefinitionId	ID of the template definition from which the workflow instance containing the task was instantiated.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTemplateDefinitionId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

Table 21-3 taskExecute() Method Parameters (Continued)

Parameter	Description	Valid Values
instanceId	ID of the workflow instance containing the task.	String specifying a valid instance ID. To get the instance ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getInstanceId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.
taskId	ID of the task that you want to execute.	String specifying a valid task ID. To get the task ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTaskId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

This method results in the sequential execution of all the actions associated with the specified task's `Executed` event for the specified task. For information about defining the `Executed` event for a task, see "Template Definition DTD" on page A-54.

This method returns an XML document that is compliant with the Client Request DTD, `ClientReq.dtd`, as described in "Client Request DTD" on page A-34. The XML document contains information about the running instance, and can be accessed by parsing the document using an XML parser, such as a SAX (Simple API for XML) parser. For example implementations of a SAX parser, see "Examples of Managing Run-Time Tasks" on page 21-25.

For example, the following code executes the specified task and assigns the resulting XML to the `clientReq` string. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
String clientReq = worklist.taskExecute(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId()
);
```

The template definition, workflow instance, and task IDs are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The `task` object can be obtained using the methods described in "Getting All Tasks" on page 21-4.

For more information about the `com.bea.wlpi.common.TaskInfo` methods, see "TaskInfo Object" on page B-20.

For more information about the `taskExecute()` methods, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Responding to a Client Request

When defining a workflow, you can establish a communication channel between the running instance (via the process engine) and the client program, by creating *integration* actions. These actions, typically Send XML to Client actions, send a request to a client through an XML document that is compliant with a standard or custom DTD. For example, the action of sending a request to a client may be implemented through a dialog box.

A response from the client is expected. The originating action typically uses the `XPath` function to extract the required values from the response. For information about XPath, see:

`http://www.w3.org/TR/xpath`

For information about defining integration actions within a workflow, see "Template Definition DTD" on page A-54.

To respond to a request raised by a `Send XML to Client` action (`ActionSendXMLToClient`), use one of the following `com.bea.wlpi.server.worklist.Worklist` methods:

Method 1
```
public java.lang.String response(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String nodeId,
  java.lang.Object data
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2
```
public java.lang.String response(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String nodeId,
  java.lang.Object data,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 3
```
public java.lang.String response(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String nodeId,
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `response()` method parameters for which you must specify values.

Table 21-4 response() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition.	String specifying a valid template ID.
instanceId	ID of the workflow instance.	String specifying a valid instance ID.
nodeId	ID of the workflow object (or *node*) that raised the client request to which the response call is responding.	String specifying a valid node ID.

Table 21-4 response() Method Parameters (Continued)

Parameter	Description	Valid Values
data	Client's request-specific response.	Any object that is compliant with the receiving action. Typically, the request has been raised by an ActionSendXMLToClient action, using one of the standard DTDs, or a custom DTD. In such cases, the response is an XML document, as described under `xml`.
xml	Client's request-specific response.	XML document that is compliant with one of the following four predefined DTDs, or with a user-defined custom DTD: ■ Client Call Addin Response DTD ■ Client Call Program Response DTD ■ Client Message Box Response DTD ■ Client Set Variables Response DTD For more information about the DTDs listed above, see "DTD Formats" on page A-1.
transactionId	ID of the transaction. **Note:** This parameter is required only in a clustered environment.	Object specifying a unique transaction ID. To generate a unique transaction ID, create a new `com.bea.wlpi.client.common.GUID` object using the following constructor: `GUID transactionId = new GUID();` For more information about the GUID class, see the `com.bea.wlpi.client.common.GUID` Javadoc.

The `response()` method sets the callback variables and sequentially executes all of the actions. For information about defining callback variables and actions, see "Template Definition DTD" on page A-54.

This method returns an XML document that is compliant with the Client Request DTD, `ClientReq.dtd`, as described in "Client Request DTD" on page A-34. The XML document contains information about the running instance, and can be accessed

by parsing the document using an XML parser, such as a SAX (Simple API for XML) parser. For example implementations of a SAX parser, see "Examples of Managing Run-Time Tasks" on page 21-25.

For example, the following code sends a response to the server and stores the return value in the `clientReq` variable.

```
String clientReq = worklist.response( templateDefinitionId,
   instanceId, actionId, response.toString() );
```

For a more complete example, see "Responding to a Client Request" on page 21-47. For more information about the `response()` methods, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Assigning a Task

To change the user or role to which a task is assigned or to revoke a task assignment, use one of the following `com.bea.wlpi.server.worklist.Worklist` methods:

Method 1

```
public java.lang.String taskAssign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.String assigneeId,
  boolean isRole,
  boolean bLoadBalance
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public java.lang.String taskAssign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.String assigneeId,
  boolean isRole,
  boolean bLoadBalance,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 3

```
public java.lang.String taskUnassign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
```

```
  java.lang.String taskId
) throws RemoteException, WorkflowException
```

The following table describes the parameters required by the `taskAssign()` and `taskUnassign()` methods for which you must specify values.

Table 21-5 taskAssign() and taskUnassign() Method Parameters

Parameter	Description	Valid Values	Valid For . . .	
			taskAssign()	taskUnassign()
templateDefinitionId	ID of the template definition for which you want to assign tasks.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTemplateDefinitionId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.	+	+

Table 21-5 taskAssign() and taskUnassign() Method Parameters (Continued)

Parameter	Description	Valid Values	Valid For . . .	
			taskAssign()	taskUnassign()
instanceId	ID of the workflow instance corresponding to the task.	String specifying a valid instance ID. To get the instance ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getInstanceId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.	+	+

Table 21-5 taskAssign() and taskUnassign() Method Parameters (Continued)

Parameter	Description	Valid Values	Valid For . . .	
			taskAssign()	taskUnassign()
taskId	ID of the task that you want to assign.	String specifying a valid task ID. To get the task ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTaskId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.	+	+
assigneeId	ID of the *assignee* (role or user) for whom to assign tasks.	String specifying a valid role or user ID. For information about getting a list of current users or roles, see "Configuring the Security Realms" on page 9-1.	+	
isRole	Boolean flag specifying whether the assignee specified for the `assigneeId` parameter is a role or user.	`true` (role) or `false` (user).	+	

Table 21-5 taskAssign() and taskUnassign() Method Parameters (Continued)

Parameter	Description	Valid Values	Valid For . . .	
			taskAssign()	taskUnassign()
bLoadBalance	Boolean flag specifying whether or not load balancing should be performed within the specified role.	`true` (perform load balancing) or `false` (do not perform load balancing). This setting is valid only if the `isRole` value is set to `true`.	+	
transactionId	ID of the transaction. **Note:** This parameter is required only in a clustered environment.	Object specifying a unique transaction ID. To generate a unique transaction ID, create a new `com.bea.wlpi.client.common.GUID` object using the following constructor: `GUID transactionId = new GUID();` For more information about the GUID class, see the `com.bea.wlpi.client.common.GUID` Javadoc.	+	

When selecting an assignee, consider the following:

- Whether any task reroutes currently exist for the specified assignee. In this case, the tasks are rerouted, as specified.
- Whether or not the `isRole` and `bLoadBalance` arguments are set to `true`, indicating that load balancing is in effect. In this case, the process engine reviews the number of tasks assigned to all users in a role (for whom no reroutes are in effect currently), identifies the user with the least number of assigned tasks, and assigns the task to this user.

These methods return an XML document that is compliant with the Client Request DTD, `ClientReq.dtd`, as described in "Client Request DTD" on page A-34. The XML document contains information about the running instance, and can be accessed

by parsing the document using an XML parser, such as a SAX (Simple API for XML) parser. For example implementations of a SAX parser, see "Examples of Managing Run-Time Tasks" on page 21-25.

For example, the following code assigns a task to the user `joe`, and assigns the resulting XML to the `clientReq` string. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
String clientReq = worklist.taskAssign(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId(),
   "joe",
   false,
   false
);
```

The following code revokes the assignment for the same task.

```
String clientReq = worklist.taskUnassign(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId(),
   "joe",
   false,
   false
);
```

The template definition, workflow instance, and task IDs are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The `task` object can be obtained using the methods described in "Getting All Tasks" on page 21-4.

For more information about the `com.bea.wlpi.common.TaskInfo` methods, see "TaskInfo Object" on page B-20, or the `com.bea.wlpi.common.TaskInfo` Javadoc.

For more information about the `taskAssign()` and `taskUnassign()` methods, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Marking a Task as Complete or Incomplete

To mark a task as complete (done) or incomplete (undone), use one of the following `com.bea.wlpi.server.worklist.Worklist` methods:

Method 1

```
public java.lang.String taskMarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public java.lang.String taskMarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 3

```
public java.lang.String taskUnmarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `taskMarkDone()` and `taskMarkUndone()` method parameters for which you must specify values.

Table 21-6 taskMarkDone() and taskMarkUndone() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition from which the workflow instance containing the task was instantiated.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTemplateDefinitionId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.
instanceId	ID of the workflow instance containing the task.	String specifying a valid instance ID. To get the instance ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getInstanceId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

Table 21-6 taskMarkDone() and taskMarkUndone() Method Parameters (Continued)

Parameter	Description	Valid Values
taskId	ID of the task that you want to mark as complete or incomplete.	String specifying a valid task ID. To get the task ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTaskId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.
transactionId	ID of the transaction. **Note:** This parameter is required only in a clustered environment.	Object specifying a unique transaction ID. To generate a unique transaction ID, create a new `com.bea.wlpi.client.common.GUID` object using the following constructor: `GUID transactionId = new GUID();` For more information about the GUID class, see the `com.bea.wlpi.client.common.GUID` Javadoc.

The `taskMarkDone()` method sets the task `Completed` value to the current date and time, and initiates the sequential execution of all the actions associated with the `Marked Done` event for the specified task. The `taskMarkDone()` method, however, has no effect on a task that has already been marked complete. For information about defining the `Marked Done` event for a task, see "Template Definition DTD" on page A-54.

The `taskUnmarkDone()` method clears the `Completed` date. The method *does not* execute the actions associated with the `Activated` event for the specified task.

This method returns an XML document that is compliant with the Client Request DTD, `ClientReq.dtd`, as described in "Client Request DTD" on page A-34. The XML document contains information about the running instance, and can be accessed

by parsing the document using an XML parser, such as a SAX (Simple API for XML) parser. For example implementations of a SAX parser, see "Examples of Managing Run-Time Tasks" on page 21-25.

For example, the following code marks the specified task as complete, sets the `Completed` value to the current date and time, executes the actions associated with the `Marked Done` event for the specified task, and assigns the resulting XML to the `clientReq` string. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
String clientReq = worklist.taskMarkDone(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId()
);
```

The following code marks the same task as incomplete, clears the `Completed` date, and assigns the resulting XML to the `clientReq` string.

```
String clientReq = worklist.taskMarkUndone(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId()
);
```

The template definition, workflow instance, and task IDs are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The `task` object can be obtained using the methods described in "Getting All Tasks" on page 21-4.

For more information about the `com.bea.wlpi.common.TaskInfo` methods, see "TaskInfo Object" on page B-20, or the `com.bea.wlpi.common.TaskInfo` Javadoc.

For more information about the `taskMarkDone()` and `taskMarkUndone()` methods, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Setting Task Properties

To set task properties, use the following `com.bea.wlpi.server.worklist.Worklist` method:

```
public java.lang.String taskSetProperties(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  int priority,
  boolean doneWithoutDoit,
  boolean doitIfDone,
  boolean unmarkDone,
  boolean modifiable,
  boolean reassignable
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `taskSetProperties()` method parameters for which you must specify values.

Table 21-7 taskSetProperties() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition corresponding to the task.	String specifying a valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTemplateDefinitionId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

Table 21-7 taskSetProperties() Method Parameters (Continued)

Parameter	Description	Valid Values
instanceId	ID of the workflow instance corresponding to the task.	String specifying a valid instance ID. To get the instance ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getInstanceId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.
taskId	ID of the task that you want to mark complete or incomplete.	String specifying a valid task ID. To get the task ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getTaskId()` For information about getting the `TaskInfo` object, see "Getting Workflow Instance Tasks" on page 22-10. For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.
priority	Task priority.	`0` (low), `1` (medium), or `2` (high).
doneWithoutDoit	Permission to mark task as complete using the methods described in "Marking a Task as Complete or Incomplete" on page 21-18.	`true` (enabled) or `false` (disabled).
doitIfDone	Permission to execute a task after it has been marked as complete using the method described in "Executing a Task" on page 21-6.	`true` (enabled) or `false` (disabled).

Table 21-7 taskSetProperties() Method Parameters (Continued)

Parameter	Description	Valid Values
unmarkDone	Permission to mark a task as incomplete using the methods described in "Marking a Task as Complete or Incomplete" on page 21-18.	`true` (enabled) or `false` (disabled).
modifiable	Permission to modify run-time properties for the task using the `taskSetProperties()` method.	`true` (enabled) or `false` (disabled).
reassignable	Permission to reassign a task instance to another participant using the method described in "Assigning a Task" on page 21-12.	`true` (enabled) or `false` (disabled).

This method returns an XML document that is compliant with the Client Request DTD, `ClientReq.dtd`, as described in "Client Request DTD" on page A-34. The XML document contains information about the running instance, and can be accessed by parsing the document using an XML parser, such as a SAX (Simple API for XML) parser. For example implementations of a SAX parser, see "Examples of Managing Run-Time Tasks" on page 21-25.

For example, the following code sets the default task priority to medium (`1`), enables (sets to `true`) all other task properties, and stores the resulting XML document in the `clientReq` variable. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
String clientReq = worklist.taskSetProperties(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId(),
   1,
   true,
   true,
   true,
   true,
   true
);
```

The template definition, workflow instance, and task IDs are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The `task` object can be obtained using the methods described in "Getting All Tasks" on page 21-4.

For more information about the `com.bea.wlpi.common.TaskInfo` methods, see "TaskInfo Object" on page B-20, or the `com.bea.wlpi.common.TaskInfo` Javadoc.

For more information about the `taskSetProperties()` method, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

Updating an Instance Variable

For information about updating workflow instance variables, see "Monitoring Run-Time Variables" on page 23-1.

Invoking an Exception Handler

For information about invoking an exception handler, see "Invoking a Workflow Exception Handler" on page 24-10.

Examples of Managing Run-Time Tasks

The following sections provide excerpts from the command-line and JSP worklist examples showing how to manage run-time tasks, including an implementation of the Xerces SAX parser.

Command-Line Worklist Example

This section provides excerpts from the command-line worklist example showing how to manage run-time tasks.

Note: For more information about the command-line worklist example, see "Command-Line Worklist Example" on page 1-23.

This example prompts the user to specify one of the following actions to perform:

- Getting Task Counts
- Getting All Tasks
- Assigning a Task
- Executing a Task
- Marking a Task as Complete
- Setting the Task Properties
- Unassigning a Task
- Marking a Task as Incomplete

The applicable sections of code are provided below, and the notable lines are shown in **bold**. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

In this example, an input stream is defined to communicate with the user, and the user is prompted to specify one of the following actions to be performed:

```
/* Create an input stream to communicate with the user */
stdIn = new BufferedReader( new InputStreamReader( System.in ) )

/* Display Tool Title */
System.out.print( "\n---  Command Line Worklist v1.2  ---" );

/* Display the main menu and interact with user */
while( true ) {
   /* Display the menu */
   System.out.println( "\n---           Main Menu          ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Workflows" );
```

```
   System.out.println( "3) Tasks" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
   .
   .
   .
```

The `mngTasks()` method shows how to manage run-time tasks, interacting with the user to retrieve the information required.

```
public static void mngTasks( ) {
   boolean isForCurrentUser;
   boolean isRole, isUserInRole;
   boolean isDoneWithoutDoit, isDoitIfDone, isUnmarkDone, isModifiable,
      isReassignment;
   int priority;
   String answer;
   String assigneeId, instanceId, orgId, taskId, templateDefId;

   /* Create an input stream to communicate with the user */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* Display the main menu and interact with user */
      while( true ) {
         /* Display the menu */
         System.out.println( "\n\n---              Tasks             ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Get Tasks Count" );
         System.out.println( "2) List Tasks" );
         System.out.println( "3) Assign a Task" );
         System.out.println( "4) Execute a Task" );
         System.out.println( "5) Mark a Task as Done" );
         System.out.println( "6) Set Task Properties" );
         System.out.println( "7) Unassign a Task" );
         System.out.println( "8) Unmark a Task as Done" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* Get the user's selection */
         String line = stdIn.readLine( );

         /* User pressed enter without making a selection ? */
         if( line.equals( "" ) )
            continue;
         /* User entered more than one char ? */
```

```
else if( line.length( ) > 1 ) {
   System.out.println( "*** Invalid choice" );
   continue;
}

/* Convert to uppercase and to char */
char choice = line.toUpperCase( ).charAt( 0 );

/* Process user's selection */
switch( choice ) {
.
.
.
```

Getting Task Counts

The following excerpt shows how to get task counts.

```
/* Get Tasks Count */
case '1' :
   /* WLPI Public API Method */
   /* NOTE: Would be nice to add code to capture any
    * thrown exceptions */
   /* Retrieve the task count for the current user in an array */
   /* Array element 0 = Total tasks assigned */
   /* Array element 1 = Tasks assigned that are overdue */
   int taskCounts[] = worklist.getTaskCounts();

   /* Any task assigned ? */
   if( taskCounts[0] == 0 )
      System.out.println( "\nNo task assigned" );
   else {
      System.out.println( "\nTasks count:" );

   if( taskCounts[0] == 1 )
      System.out.println( "- 1 task assigned" );
   else
      System.out.println( "- " + taskCounts[0] + " tasks assigned" );

   /* Any overdue task ? */
   if( taskCounts[1] == 0 )
      System.out.println( "- No overdue task" );
   else if( taskCounts[1] == 1 )
      System.out.println( "- 1 overdue task" );
   else
      System.out.println( "- " + taskCounts[1] + " overdue tasks" );
   }
```

```
   break;
   .
   .
   .
```

Getting All Tasks

The following excerpt shows how to get a list of all tasks.

```
/* List all tasks */
case '2' :
   /* List the Task for the current user? */
   if( ( answer = askQuestion(
      "\nList Tasks for the current user (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Evaluate the answer */
   isForCurrentUser = ( answer.equals( "y" ) || answer.equals( "Y" ) );

   List taskList;

   if( isForCurrentUser )
      /* WLPI Public API Method */
      /* NOTE: Would be nice to add code to capture any
       * thrown exceptions */
      /* Get Tasks list for the active user and the
       * active organization. */
      taskList = worklist.getTasks( );
   else {
      /* Get Organization ID */
      if( ( orgId = askQuestion(
         "\nEnter Organization ID" ) ) == null ) {
         /* User cancelled the operation */
         System.out.println( "*** Cancelled" );
         break;
      }

      /* Get the Asignee ID */
      if( ( assigneeId = askQuestion(
         "Enter Assignee ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
      }
```

```
      /* Is the Assignee a Role? */
      if( ( answer = askQuestion(
         "Is the Assignee a Role (y/n)?" ) ) == null ) {
         /* User cancelled the operation */
         System.out.println( "*** Cancelled" );
         break;
      }

      /* Evaluate the answer */
      isRole = ( answer.equals( "y" ) || answer.equals( "Y" ) );

      /* WLPI Public API Method */
      /* NOTE: Would be nice to add code to capture any
       * thrown exceptions */
      /* Get Tasks list for the specified user and organization. */
      taskList = worklist.getTasks( orgId, assigneeId, isRole );
   }

   /* Any task assigned ? */
   if( taskList.size( ) == 0 )
      System.out.println( "\nNo task assigned" );
   else
      System.out.print( "\nAssigned Tasks:" );

   /* Process the list to display the tasks */
   for( int i = 0; i < taskList.size( ); i++ ) {
      /* Retrieve an element from the list */
      TaskInfo taskInfo = ( TaskInfo )taskList.get( i );

      /* WLPI Public API Methods */
      /* Retrieve and display a sub-set of available attributes */
      System.out.println( "\n- Name: " + taskInfo.getName( ) );
      System.out.println( "  Template Definition ID: " +
         taskInfo.getTemplateDefinitionId( ) );
      System.out.println( "  Workflow Instance ID: " +
         taskInfo.getInstanceId( ) );
      System.out.println( "  Task ID: " + taskInfo.getTaskId( ) );
      System.out.print( "  Status: " + taskInfo.getStatus( ) );

      /* Retrieve and display the task status */
      if( taskInfo.getStatus( ) == taskInfo.STATUS_PENDING )
         System.out.println( " (Pending)" );
      else if( taskInfo.getStatus( ) == taskInfo.STATUS_COMPLETE )
         System.out.println( " (Complete)" );
      else if( taskInfo.getStatus( ) == taskInfo.STATUS_OVERDUE )
         System.out.println( " (Overdue)" );
      else if( taskInfo.getStatus( ) == taskInfo.STATUS_INACTIVE )
         System.out.println( " (Inactive)" );
```

```
      else
         System.out.println( " (Unknown)" );
   }
   break;
   .
   .
   .
```

Assigning a Task

The following excerpt shows how to assign a task.

```
/* Assign a Task */
case '3' :
   /* Get the Template Definition ID */
   if( ( templateDefId = askQuestion(
      "\nEnter Template Definition ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the Workflow Instance ID */
   if( ( instanceId = askQuestion(
      "Enter Workflow Instance ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the Task ID */
   if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the Assignee ID */
   if( ( assigneeId = askQuestion( "Enter Assignee ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Is the Assignee a Role? */
  if( ( answer = askQuestion( "Assign to a role (y/n)?" ) ) == null ) {
      /* User cancelled the operation */
```

```
            System.out.println( "*** Cancelled" );
            break;
         }

         /* Evaluate the answer */
         isRole = ( answer.equals( "y" ) || answer.equals( "Y" ) );

         /* If the Assignee a role, assign to user in this role? */
         if( isRole ) {
            if( ( answer = askQuestion(
               "Assign to a user in this role (y/n)?" ) ) == null ) {
               /* User cancelled the operation */
               System.out.println( "*** Cancelled" );
               break;
            }

            /* Evaluate the answer */
            isUserInRole = ( answer.equals( "y" ) || answer.equals( "Y" ) );
         }
         else
         isUserInRole = false;

         /* Assign the Task */
         try {
            /* WLPI Public API Method */
            String clientReq = worklist.taskAssign(
               templateDefId, instanceId, taskId, assigneeId,
               isRole, isUserInRole );

            /* Success (No exception thrown) */
            System.out.println( "- Success" );

            /* Parse the reply from the server to process client
             * request (if any) */
            parser.handleRequest( clientReq );
         }
         catch( Exception e ) {
            System.out.println( "*** Failed to assign the task" );
            System.err.println( e );
         }
         break;
.
.
.
```

Executing a Task

The following excerpt shows how to execute a task.

```
/* Execute a Task */
case '4' :
   /* Get the Template Definition ID */
   if( ( templateDefId = askQuestion(
      "\nEnter Template Definition ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the Workflow Instance ID */
   if( ( instanceId = askQuestion(
      "Enter Workflow Instance ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the Task ID */
   if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Execute the Task */
   try {
       /* WLPI Public API Method */
      String clientReq = worklist.taskExecute(
         templateDefId, instanceId, taskId );

      /* Success (No exception trown) */
      System.out.println( "- Success" );

      /* Parse the reply from the server to process client
       * request (if any) */
      parser.handleRequest( clientReq );
   }
   catch( Exception e ) {
      System.out.println( "*** Failed to execute the task" );
      System.err.println( e );
   }
   break;
   .
```

```
.
.
```

Marking a Task as Complete

The following excerpt shows how to mark a task as complete.

```
/* Mark a Task as done */
case '5' :
   /* Get the Template Definition ID */
   if( ( templateDefId = askQuestion(
      "\nEnter Template Definition ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the Workflow Instance ID */
   if( ( instanceId = askQuestion(
      "Enter Workflow Instance ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the Task ID */
   if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Mark the Task as Done */
   try {
      /* WLPI Public API Method */
      String clientReq = worklist.taskMarkDone(
         templateDefId, instanceId, taskId );

      /* Success (No exception trown) */
      System.out.println( "- Success" );

      /* Parse the reply from the server to process client
       * request (if any) */
      parser.handleRequest( clientReq );
   }
   catch( Exception e ) {
      System.out.println( "*** Failed to mark the task as done" );
```

```
            System.err.println( e );
        }
        break;
        .
        .
        .
```

Setting the Task Properties

The following excerpt shows how to set task properties.

```
/* Set Task Properties */
case '6' :
    /* Get the Template Definition ID */
    if( ( templateDefId = askQuestion(
        "\nEnter Template Definition ID" ) ) == null ) {
        /* User cancelled the operation */
        System.out.println( "*** Cancelled" );
        break;
    }

    /* Get the Workflow Instance ID */
    if( ( instanceId = askQuestion(
        "Enter Workflow Instance ID" ) ) == null ) {
        /* User cancelled the operation */
        System.out.println( "*** Cancelled" );
        break;
    }

    /* Get the Task ID */
    if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
        /* User cancelled the operation */
        System.out.println( "*** Cancelled" );
        break;
    }

    /* Get Task Priority */
    if( ( answer = askQuestion(
        "Enter the Priority (0=low, 1, 2=high)" ) ) == null ) {
        /* User cancelled the operation */
        System.out.println( "*** Cancelled" );
        break;
    }

    /* Parse the answer */
    priority = Integer.parseInt( answer );
```

```
            /* Get the "Marked done without executing" Property */
            if( ( answer = askQuestion(
               "Can it be marked done without executing (y/n)?" ) ) == null ) {
            /* User cancelled the operation */
               System.out.println( "*** Cancelled" );
               break;
            }

            /* Evaluate the answer */
           isDoneWithoutDoit = ( answer.equals( "y" ) || answer.equals( "Y" ) );

            /* Get the "Re-executed if marked done" Property */
            if( ( answer = askQuestion(
               "Can it be re-executed if marked done (y/n)?" ) ) == null ) {
               /* User cancelled the operation */
               System.out.println( "*** Cancelled" );
               break;
            }

            /* Evaluate the answer */
            isDoitIfDone = ( answer.equals( "y" ) || answer.equals( "Y" ) );


            /* Get the "Unmarked if marked done" Property */
            if( ( answer = askQuestion(
               "Can it be unmarked if marked done (y/n)?" ) ) == null ) {
               /* User cancelled the operation */
               System.out.println( "*** Cancelled" );
               break;
            }

            /* Evaluate the answer */
            isUnmarkDone = ( answer.equals( "y" ) || answer.equals( "Y" ) );

//          System.out.print( "Can the properties be modified
//             at runtime (y/n) >> " );

            /* Get the answer */
//          answer = stdIn.readLine( );

            /* User cancelled the operation? */
//          if( answer.equals( "" ) )
//          {
//             System.out.println( "*** Cancelled" );
//              break;
//          }

            /* Evaluate the answer */
//          boolean isModifiable = (
```

```
//             answer.equals( "y" ) || answer.equals( "Y" ) );
            isModifiable = true;

            /* Get the "Reassign" Property */
            if( ( answer = askQuestion(
               "Can it be reassigned (y/n)?" ) ) == null ) {
               /* User cancelled the operation */
               System.out.println( "*** Cancelled" );
               break;
            }

            /* Evaluate the answer */
            isReassignment = ( answer.equals( "y" ) || answer.equals( "Y" ) );

            /* Set the Task Properties */
            try {
               /* WLPI Public API Method */
               String clientReq = worklist.taskSetProperties( templateDefId,
                 instanceId, taskId, priority, isDoneWithoutDoit, isDoitIfDone,
                  isUnmarkDone, isModifiable, isReassignment );

               /* Success (No exception trown) */
               System.out.println( "- Success" );

               /* Parse the reply from the server to process client
                * request (if any) */
               parser.handleRequest( clientReq );
            }
            catch( Exception e ) {
               System.out.println( "*** Failed to set the task properties" );
               System.err.println( e );
            }
            break;
            .
            .
            .
```

Unassigning a Task

The following excerpt shows how to revoke a task assignment.

```
         /* Unassign a Task */
         case '7' :
            /* Get the Template Definition ID */
            if( ( templateDefId = askQuestion(
               "\nEnter Template Definition ID" ) ) == null ) {
               /* User cancelled the operation */
```

```
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the Workflow Instance ID */
   if( ( instanceId = askQuestion(
      "Enter Workflow Instance ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Get the Task ID */
   if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* Unassign the Task */
   try {
      /* WLPI Public API Method */
      worklist.taskUnassign( templateDefId, instanceId, taskId );

      /* Success (No exception trown) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Failed to unassign the task" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

Marking a Task as Incomplete

The following excerpt shows how to mark a task as incomplete.

```
/* Unmark a Task as done */
case '8' :
   /* Get the Template Definition ID */
   if( ( templateDefId = askQuestion(
      "\nEnter Template Definition ID" ) ) == null ) {
      /* User cancelled the operation */
      System.out.println( "*** Cancelled" );
```

```
            break;
         }

         /* Get the Workflow Instance ID */
         if( ( instanceId = askQuestion(
            "Enter Workflow Instance ID" ) ) == null ) {
            /* User cancelled the operation */
            System.out.println( "*** Cancelled" );
            break;
         }

         /* Get the Task ID */
         if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
            /* User cancelled the operation */
            System.out.println( "*** Cancelled" );
            break;
         }

         /* Unmark the Task as Done */
         try {
            /* WLPI Public API Method */
            String clientReq = worklist.taskUnmarkDone(
               templateDefId, instanceId, taskId );

            /* Success (No exception trown) */
            System.out.println( "- Success" );

            /* Parse the reply from the server to process client
             * request (if any) */
            parser.handleRequest( clientReq );
         }
         catch( Exception e ) {
            System.out.println( "*** Failed to unmark the task as done" );
            System.err.println( e );
         }
         break;

      /* Return to previous menu */
      case 'B' :
         return;

      /* Exit tool */
      case 'Q' :
         /* Disconnect from the server */
         disconnect( );
         System.exit( 1 );

      default:
         System.out.println( "*** Invalid choice" );
```

```
            }
         }
      }
      /* "Unhandled" exceptions */
      catch( Exception e ) {
         System.err.println( e );
      }
      return;
}
```

Command-Line SAX Parser Example

This section provides excerpts from the command-line SAX Parser showing how to set up the SAX parser and respond to a `set-variables` client request.

Note: For more information about the command-line SAX Parser example, see "Command-Line SAX Parser Example" on page 1-23.

For an example of how to respond to a `message-box` client request, see "Parsing the Client Request" on page 21-46.

The SAX parser is used by the command-line worklist to parse client requests. The example excerpts in this section show the following:

- Parsing the Client Request
- Responding to the Client Request

Notable lines are shown in **bold**.

Parsing the Client Request

The following excerpt shows how to use the Xerces SAX parser with a custom run-time management client. The `handleRequest()` method uses the Xerces SAX parser to parse the client requests received from the BPM server. The `request` string is an XML document conforming to the Client Request DTD, as described in "Client Request DTD" on page A-34.

```
private static final String DEFAULT_PARSER =
   "weblogic.apache.xerces.parsers.SAXParser";
   public void handleRequest( String request ) {
      if( request == null )
```

```
            return;

         /* Get instance of handlers */
         ContentHandler contentHandler = new CLContentHandler();
         ErrorHandler errorHandler = new CLErrorHandler();

         try {
            /* Instantiate a parser */
            XMLReader parser = XMLReaderFactory.createXMLReader(
               DEFAULT_PARSER );

            /* Register the content handler */
            parser.setContentHandler( contentHandler );

            /* Register the error handler */
            parser.setErrorHandler( errorHandler );

            /* Create a character stream */
            Reader inReader = new StringReader( request );
            /* Encapsulate the stream for the SAX Parser */
            InputSource inputSource = new InputSource( inReader );

            /* Parse the request */
            parser.parse( inputSource );
         }
            catch( Exception e ) {
            System.err.println( e );
         }
      }
   }
```

Responding to the Client Request

The following methods interact with the user to retrieve the `set-variable` client request, which is triggered by an ActionSendXMLToClient event, and to respond to the `set-variable` client request.

The following method retrieves the `set-variable` client request.

```
public void setVariables( List listVarInfo ) {
   String varValue;

   /* Process the list to display the prompt and
    * retrieve the value */
   for( int i = 0; i < listVarInfo.size( ); i++ ) {
   /* Retrieve an element from the list */
      VariableInfo varInfo = ( VariableInfo )listVarInfo.get( i );
```

```
      /* Prompt user for the variable value */
      if( ( varValue = askQuestion(
         "- " + varInfo.getVarPrompt() ) ) != null ) {
         /* User entered a value; so overwrite the stored value */
         varInfo.setVarValue( varValue );
         listVarInfo.set( i, ( VariableInfo )varInfo );
      }
   }
   return;
}
   .
   .
   .
```

The following method responds to a `set-variable` client request.

```
private static String sendSetVariablesResponse(
   String templateDefinitionId, String instanceId,
   String actionId, List listVarInfo  ) {

   String clientReq = null;

   /* Create a variable to store the XML clientResponse
    * and add the document prolog */
   StringBuffer response = new StringBuffer(
      "<?xml version=\"1.0\"?>\r\n" );

   /* Add the XML document root */
   response.append( "<set-variables>\r\n" );

   /* Process the list to retrieve the varriables
    * name and value */
   for( int i = 0; i < listVarInfo.size( ); i++ ) {
      /* Add the variable element and the name attribute */
      response.append( "  <variable name=\"" );

      /* Retrieve an element from the list */
      VariableInfo varInfo = ( VariableInfo )listVarInfo.get( i );

      /* Add the value for the name attribute */
      response.append( varInfo.getVarName() + "\">" );

      /* Add the value for the variable element */
      response.append( XMLToString( varInfo.getVarValue() ) );

      /* Add end tag */
      response.append( "</variable>\r\n" );
```

```
    }

    /* Add root end tag to the XML document */
    response.append( "</set-variables>" );

    try {
       /* WLPI Public API Method */
       /* Send the client's response to the 'set-variables' client
        * request */
       clientReq = clWorklist.getWorklist().response(
          templateDefinitionId, instanceId, actionId,
          response.toString() );

       /* Success (No exception trown) */
       System.out.println( "\nResponse sent to server" );
    }
    catch( Exception e ) {
       System.out.println( "\nFailed sending response to server" );
       System.err.println( e );
    }
    return( clientReq );
}
.
.
.
```

JSP Worklist Example

This section provides excerpts from the JSP worklist example showing how to manage run-time tasks.

Note: For more information about the JSP worklist example, see "JSP Worklist Example" on page 1-24.

The example excerpts illustrate the following features:

- Getting Tasks
- Executing a Task
- Parsing the Client Request
- Responding to a Client Request

- Assigning a Task
- Marking a Task as Complete or Incomplete
- Setting Task Properties

In the following examples, all noteable lines of code are shown in **bold**. `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

Getting Tasks

The following code excerpt from the JSP worklist (`worklist.jsp`) shows how to get a list of tasks and use the `com.bea.wlpi.common.TaskInfo` methods to obtain information about those tasks.

To get a list of tasks, set the `Display Options`, as shown in the figure "Main Interface to the JSP Worklist" on page 1-25.

```
try {
   List v = worklist.getTasks(currentOrg, currentRole, isRole);
   session.putValue("tasklist", v);

   // Get item count
   int numItems = 0;
   for (int i = 0, n = v.size(); i < n; i++) {
      TaskInfo task = (TaskInfo)v.get(i);
      if (task.getStarted() != null) {
         if (task.getCompleted() != null) {
            if (!done)
               continue;
         }
         else {
            if (!pending)
               continue;
            }
         }
         else if (!inactive)
            continue;
         numItems++;
    }
}
.
.
.
```

For more information about the `com.bea.wlpi.common.TaskInfo` methods, see "TaskInfo Object" on page B-20, or the `com.bea.wlpi.common.TaskInfo` Javadoc.

Executing a Task

The following code excerpt from the JSP worklist (`worklist.jsp`) shows how to execute a task.

A user can execute a task by selecting the `Perform Task` link, as shown in the figure "Main Interface to the JSP Worklist" on page 1-25, and selecting the task to execute.

```
try {
   List vTaskList = (List)session.getValue("tasklist");
   TaskInfo task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
   session.putValue("action", ACTION_DOIT);
   if (action.equals(ACTION_DOIT)) {
      System.out.println(worklist.taskExecute(task.getTemplateDefinitionId(),
         task.getInstanceId(), task.getTaskId()));
      responseParser.parse( worklist.taskExecute(
                              task.getTemplateDefinitionId(),
                              task.getInstanceId(),
                              task.getTaskId() )
      );
      if (responseParser.isQuery()) {
         pageContext.forward("query.jsp");
         return;
      }
      if (responseParser.isMessage()) {
         pageContext.forward("message.jsp");
         return;
    }
}
.
.
.
```

The template definition, workflow instance, and task IDs are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The resulting XML document (which is compliant with the Client Request DTD, as described in "Client Request DTD" on page A-34) is returned from the `taskExecute()` method call and passed to the response parser, `ResponseParser`, for parsing. For more information about the response parser, see "Parsing the Client Request" on page 21-46.

For more information about the `com.bea.wlpi.common.TaskInfo` methods, see "TaskInfo Object" on page B-20, or the `com.bea.wlpi.common.TaskInfo` Javadoc.

Parsing the Client Request

The following excerpts from the JSP SAX Parser, `ResponseParser`, show how to set up the SAX parser and respond to a `message-box` client request. For an example of how to respond to a `set-variables` client request, see "Parsing the Client Request" on page 21-40, within the description of the command-line SAX parser.

```
package jsp_servlet._worklist;

import java.io.*;
import java.util.*;
import javax.servlet.http.*;
import com.sun.xml.parser.Parser;
import org.xml.sax.*;

public class ResponseParser implements DocumentHandler {
      .
      .
      .
   public boolean isMessage() {
      return message;
   }
      .
      .
      .
   public String getMessageOption() {
      return ((String)messageBox.getOption());
   }
   public String getMessageStyle() {
      return ((String)messageBox.getStyle());
   }  .
      .
      .
   public String generateMessageResponse(HttpServletRequest request) {
      StringBuffer response = new StringBuffer();
      response.append(
         "<?xml version=\"1.0\"?>\r\n<message-box
         option=\""+request.getParameter("option")+"\"/>");
      System.out.println((String) response.toString());
      return((String) response.toString());
   }
      .
      .
      .
```

```
   public void parse(String xml) {
      try {
         parser = new Parser();
         parser.setDocumentHandler(this);
         parser.parse(new InputSource(new ByteArrayInputStream(
           xml.getBytes())));
      } catch (SAXParseException spe) {
      } catch (Exception e) {
      }
   }
.
.
.
```

Responding to a Client Request

This section provides two code examples, excerpted from the JSP worklist, that show how to respond to a client request.

The first excerpt is taken from the `worklist.jsp` file. The `responseParser` parses the XML document. If a `set-variables` request is encountered , the context is passed to the `query.jsp` file. If a `message-box` request is encountered, the context is passed to the `message.jsp` file. For more information about the response parser, see "Parsing the Client Request" on page 21-46.

```
if (action.equals(ACTION_DOIT)) {
   responseParser.parse( worklist.taskExecute(
                           task.getTemplateDefinitionId(),
                           task.getInstanceId(),
                           task.getTaskId() )
   );
   if (responseParser.isQuery()) {
      pageContext.forward("query.jsp");
      return;
   }
   if (responseParser.isMessage()) {
      pageContext.forward("message.jsp");
     return;
   }
}
.
.
.
```

The second excerpt provides the contents of the `message.jsp` file, illustrating the `response()` method call. The method makes use of the XML parser, `responseParser`, to parse the client request XML (obtaining the template definition ID, instance ID, and workflow object ID), and to generate a response based on the request. For more information about the response parser, see "Parsing the Client Request" on page 21-46.

```
<%@ page import="
   javax.ejb.*,
   java.rmi.RemoteException,
   java.util.*,
   java.text.*,
   com.bea.wlpi.common.*
   com.bea.wlpi.server.worklist.*
"%>

<jsp:usebean id="responseParser" class="jsp_servlet._worklist.ResponseParser"
scope="session"/>

<%
   Worklist worklist = null;
   TaskInfo task = null;
   String error = null;
   String resp = null;
   String messageOption = null;
   worklist = (Worklist)session.getValue("worklist");

   if (worklist != null) {
      Hashtable h = new Hashtable();
      h.put(javax.naming.Context.SECURITY_PRINCIPAL, session.getValue("name"));
      h.put(javax.naming.Context.SECURITY_CREDENTIALS,
         session.getValue("password"));
      new javax.naming.InitialContext(h);

   if (request.getParameter("option") != null) {
      try {
         System.out.println(responseParser.generateMessageResponse(request));
         System.out.println(responseParser.getTemplateDefinitionId());
         System.out.println(responseParser.getInstanceId());
         System.out.println(responseParser.getActionId());

         resp = worklist.response(
                   responseParser.getTemplateDefinitionId(),
                   responseParser.getInstanceId(),
                   responseParser.getActionId(),
                   responseParser.generateMessageResponse(request)
         );
         pageContext.forward("worklist.jsp");
```

```
            return;
         } catch (Exception e) {
            e.printStackTrace();
         }
         }
      }
%>
<html>
<head>
<title>Message</title>
</head>
<body bgcolor=#ffffff>
<font face="Arial" size="4">
<form action="message.jsp" method="POST">
<table dir="ltr" border="0" cellpadding="1" cellspacing="1">
   <tr>
      <td><img src="logo.gif"></td>
      <td><font color="red" size="7">
         <strong><em>Weblogic</em></strong></font>
         <font color="black" size="7">
         <strong><em>Process Integrator</em></strong></font>
      </td>
   </tr>
   <tr>
      <td bgcolor="navy" valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2">
         <tr>
            <td><font color="yellow">
               <strong>Go ...</strong></font></td>
         </tr>
         <tr>
            <td><a href="worklist.jsp">
               <font color="white">Worklist</font></a></td>
         </tr>
         <tr>
            <td><a href="startworkflow.jsp">
               <font color="white">Start Workflow</font></a></td>
         </tr>
         <tr>
            <td><a href="logoff.jsp">
               <font color="white">Logoff</font></a></td>
         </tr>
      </table>
      </td>
      <td valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" valign="top">
<%
    if (error != null) {
%>
```

```
         <tr>
            <td><font color="red size="4" face="Arial"><%=error%></td>
         </tr>
<%
        }
%>
         <tr>
            <td><font color="navy" size="4" face="Arial">
<%=responseParser.getHeader()%></font></td>
         </tr>
         <tr>
            <td>
            <table border="0" cellpadding="2" cellspacing="2">
               <tr>
                  <td></td>
<%
   messageOption = responseParser.getMessageOption();
   if (messageOption.equals("ok")) {
%>
                  <td width="10%">
                     <input type="submit" name="option" value="ok"></td>
                  <td width="90%"> </td>
<%
   }
%>
<%
   if (messageOption.equals("ok_cancel")) {
%>
                  <td width="10%">
                     <input type="submit" name="option" value="ok"></td>
                  <td width="10%">
                     <input type="submit" name="option" value="cancel"></td>
                  <td width="80%"> </td>

<%
   }
%>
<%
   if (messageOption.equals("yes_no")) {
%>
                  <td width="10%">
                     <input type="submit" name="option" value="yes"></td>
                  <td width="10%">
                     <input type="submit" name="option" value="no"></td>
                  <td width="80%"> </td>
<%
   }
%>
<%
```

```
   if (messageOption.equals("yes_no_cancel")) {
%>
                  <td width="10%">
                     <input type="submit" name="option" value="yes"></td>
                  <td width="10%">
                     <input type="submit" name="option" value="no"></td>
                  <td width="10%">
                     <input type="submit" name="option" value="cancel"></td>
                  <td width="70%"> </td>
<%
   }
%>
               </tr>
            </table>
         </td>
      </tr>
   </table>
   </td>
</tr>
</table>
</form>
</font>
</body>
</html>
```

Assigning a Task

This section provides four code examples, excerpted from the JSP worklist, that show how to assign a task to a different participant (user or role).

Users can change the assignment of a task to a different participant by selecting the `Reassign` link, as shown in the figure "Main Interface to the JSP Worklist" on page 1-25, selecting the task to reassign, and specifying the user, role, or user in role when prompted to whom you want to reassign the task.

In the following example, the `Reassign` link references the `worklist.jsp` file with the `SetAction` parameter set to `4`, and a checkmark is added, as a visual cue, to the left of the selected action.

```
<tr>
   <td><%=action.equals("4") ? "<img src=\"check.gif\">" : ""%></td>
   <td><a href="worklist.jsp?SetAction=4">
      <font color="white">Reassign</font></a></td>
</tr>
```

The `SetAction` value is extracted and stored in the `action` variable, as follows:

```
if (worklist != null) {
   String cmd = request.getParameter("SetFilter");
   if (cmd != null) {
      .
      .
      .
   }
   else if (request.getParameter("SetAction") != null)
   {
      action = request.getParameter("SetAction");
      session.putValue("action", action);
   }
.
.
.
```

Based on the value of the `action` variable, the context gets passed to the `reassign.jsp` file, as shown in the following excerpt.

```
static final String ACTION_DOIT = "0";
static final String ACTION_MARKDONE = "1";
static final String ACTION_UNMARKDONE = "2";
static final String ACTION_TASKPROPERTIES = "3";
static final String ACTION_REASSIGN = "4";
    .
    .
    .
try {
   List vTaskList = (List)session.getValue("tasklist");
   TaskInfo task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
   session.putValue("action", ACTION_DOIT);
   if (action.equals(ACTION_DOIT)) {
      .
      .
      .
   }
   else if (action.equals(ACTION_REASSIGN)) {
      session.removeValue("error");
      pageContext.forward("reassign.jsp?task= "+cmd);
      return;
   }
.
.
.
```

The following code, taken from the contents of the `reassign.jsp` file, shows how to use the `taskAssign()` method to reassign a task. The template definition, workflow instance, and task IDs are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The `task` object can be obtained using the methods described in "Getting All Tasks" on page 21-4.

```
<%@ page import="
   javax.ejb.*,
   java.rmi.RemoteException,
   java.util.*,
   java.text.*,
   com.bea.wlpi.common.*
   com.bea.wlpi.server.worklist.*
   com.bea.wlpi.server.principal.*
   javax.naming.*
"%>
<%!
   static final String TYPE_USER       = "0";
   static final String TYPE_USERINROLE = "1";
   static final String TYPE_ROLE       = "2";
%>
<%
   Worklist worklist = null;
   TaskInfo task = null;
   String error = null;
   String assignType = TYPE_USER;

   worklist = (Worklist)session.getValue("worklist");
   if (worklist != null) {
      Hashtable h = new Hashtable();
      h.put(javax.naming.Context.SECURITY_PRINCIPAL, session.getValue("name"));
       h.put(javax.naming.Context.SECURITY_CREDENTIALS,
          session.getValue("password"));
      new javax.naming.InitialContext(h);
      String cmd = request.getParameter("task");
      if (cmd != null) {
         try {
            List vTaskList = (List)session.getValue("tasklist");
            task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
            session.putValue("task", task);
         } catch (Exception e) {
         }
      }
      else
         task = (TaskInfo)session.getValue("task");
   }
   if (request.getParameter("Set") != null)
      assignType = request.getParameter("Type");
```

```
   if (request.getParameter("Reassign") != null) {
      // Set properties
      try {
         assignType = request.getParameter("Type");
         boolean bRole       = false;
         boolean bUserInRole = false;
         if (!assignType.equals(TYPE_USER)) {
            bRole = true;
            if (assignType.equals(TYPE_USERINROLE))
               bUserInRole = true;
         }
         worklist.taskAssign(task.getTemplateDefinitionId(),
                             task.getInstanceId(),
                             task.getTaskId(),
                             request.getParameter("Assignee"),
                             bRole,
                             bUserInRole
         );
         session.removeValue("task");
         session.removeValue("error");
         pageContext.forward("worklist.jsp");
         return;
      } catch (Exception e) {
         error = e.getMessage();
      }
   }
   if (task != null) {
%>
<html>
<head>
<title>Reassign Task</title>
</head>
<body bgcolor=#ffffff>
<font face="Arial" size="4">
<form action="reassign.jsp" method="POST">
<table dir="ltr" border="0" cellpadding="1" cellspacing="1">
   <tr>
      <td><img src="logo.gif"></td>
      <td><font color="red" size="7"><strong><em>Weblogic</em></strong></font>
         <font color="black" size="7"><strong><em>
         Process Integrator</em></strong></font></td>
   </tr>
   <tr>
      <td bgcolor="navy" valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" >
         <tr>
            <td><font color="yellow">
            <strong>Go ...</strong></font></td>
         </tr>
```

```
         <tr>
            <td><a href="worklist.jsp">
            <font color="white">Worklist</font></a></td>
         </tr>
         <tr>
            <td><a href="startworkflow.jsp">
            <font color="white">Start Workflow</font></a></td>
         </tr>
         <tr>
            <td><a href="logoff.jsp">
            <font color="white">Logoff</font></a></td>
         </tr>
      </table>
      </td>
      <td valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" valign="top">
<%
   if (error != null) {
%>
         <tr>
            <td><font color="red size="4" face="Arial"><%=error%></td>
         </tr>
<%      }
%>
         <tr>
            <td><font color="navy" size="5" face="Arial">Reassign Task:
               <strong><%=task.getName()%></strong></font></td>
         </tr>
         <tr>
            <td>
            <table border="0" cellpadding="2" cellspacing="2">
               <tr>
                  <td><p align="right">Type:</p></td>
                  <td>
                     <select name="Type" size="1">
<option <%=assignType.equals(TYPE_USER) ? "selected" : ""%> value="0">
   User
</option>
<option <%=assignType.equals(TYPE_USERINROLE) ? "selected" : ""%> value="1">
   User in Role
</option>
<option <%=assignType.equals(TYPE_ROLE) ? "selected" : ""%> value="2">
   Role
</option>
                     </select>
                     <input type="submit" name="Set" value="Set "></td>
               </tr>
               <tr>
                  <td valign="top"><p align="right">Assign To:</p></td>
```

```
                  <td><select name="Assignee" size="10">
<%
   if (assignType.equals(TYPE_USER)) {
      List vUsers = (List)session.getValue("users");
      if (vUsers == null)
      try {
         Context ctx = (Context)session.getValue("ctx");
         WLPIPrincipalHome pHome =
            (WLPIPrincipalHome)ctx.lookup("com.bea.wlpi.WLPIPrincipal");
         WLPIPrincipal principal = (WLPIPrincipal)pHome.create();
         vUsers = principal.getUsersInOrganization("ORG1", false);
         session.putValue("users", vUsers);
         principal.remove();
      } catch (Exception e) {
      }
      for (int i = 0, n = vUsers.size(); i < n; i++) {
         String name = ((UserInfo)vUsers.get(i)).getUserId();
         out.print("<option value=\"" + name + "\">" + name + "</option>");
      }
   }
   else {
      List vRoles = (List)session.getValue("roles");
      if (vRoles == null)
         try {
            Context ctx = (Context)session.getValue("ctx");
            WLPIPrincipalHome pHome =
               (WLPIPrincipalHome)ctx.lookup("com.bea.wlpi.WLPIPrincipal");
            WLPIPrincipal principal = (WLPIPrincipal)pHome.create();
            vRoles = principal.getRolesInOrganization("ORG1", false);
            session.putValue("roles", vRoles);
            principal.remove();
         } catch (Exception e) {}
      for (int i = 0, n = vRoles.size(); i < n; i++) {
         String name = ((RoleInfo)vRoles.get(i)).getRoleId();
         out.print("<option value=\"" + name + "\">" + name + "</option>");
      }
   }
%>
                     </select>
                  </td>
               </tr>
               <tr>
                  <td></td>
                  <td><input type="submit" name="Reassign" value="Reassign ">
                  </td>
               </tr>
            </table>
         </td>
      </tr>
```

```
   </table>
   </td>
</tr>
</table>
</form>
<%
}
%>
</font>
</body>
</html>
```

Marking a Task as Complete or Incomplete

The following code examples, excerpted from the JSP worklist, show how to mark a task as complete (done) or incomplete (undone).

Users can mark a task as complete or incomplete by selecting the `Mark Done` or `Mark Undone` links, as shown in the figure "Main Interface to the JSP Worklist" on page 1-25.

In the following examples, the `Mark Done` and `Mark Undone` links reference the `worklist.jsp` file with the `SetAction` parameter set to `1` and `2`, respectively, and a checkmark, as a visual cue, is added to the left of the selected action.

```
<tr>
   <td><%=action.equals("1") ? "<img src=\"check.gif\">" : ""%></td>
   <td><a href="worklist.jsp?SetAction=1"><font color="white">
      Mark Done</font></a>
   </td>
</tr>
<tr>
   <td><%=action.equals("2") ? "<img src=\"check.gif\">" : ""%></td>
   <td><a href="worklist.jsp?SetAction=2"><font color="white">
      Unmark Done</font></a>
   </td>
</tr>
```

The `SetAction` value is extracted and stored in the `action` variable, as follows:

```
if (worklist != null) {
   String cmd = request.getParameter("SetFilter");
   if (cmd != null) {
   .
   .
   .
   else if (request.getParameter("SetAction") != null) {
```

```
      action = request.getParameter("SetAction");
      session.putValue("action", action);
   }
```

The following code, taken from the `worklist.jsp` file, shows how to use the `taskMarkDone()` and `taskUnmarkDone()` methods to mark a task as complete or incomplete, respectively.

```
try {
   List vTaskList = (List)session.getValue("tasklist");
   TaskInfo task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
   session.putValue("action", ACTION_DOIT);
   if (action.equals(ACTION_DOIT)) {
      .
      .
      .
   }
   else if (action.equals(ACTION_MARKDONE))
      worklist.taskMarkDone(task.getTemplateDefinitionId(),
                            task.getInstanceId(),
                            task.getTaskId());

   else if (action.equals(ACTION_UNMARKDONE))
    worklist.taskUnmarkDone(task.getTemplateDefinitionId(),
                            task.getInstanceId(),
                            task.getTaskId());

.
.
.
```

Setting Task Properties

This section provides several code excerpts from the JSP worklist that show how to set task properties.

Users can set the task properties by selecting the `Task Properties` link, as shown in the figure "Main Interface to the JSP Worklist" on page 1-25, selecting the task, and specifying the desired properties when prompted.

In this example, the `Task Properties` link references the `worklist.jsp` file with the `SetAction` parameter set to `3`, and a checkmark, as a visual cue, is added to the left of the selected action.

```
<tr>
   <td><%=action.equals("4") ? "<img src=\"check.gif\">" : ""%></td>
```

```
   <td><a href="worklist.jsp?SetAction=3"><font color="white">
      Task Properties</font></a></td>
</tr>
```

Next, the `SetAction` value is extracted and stored in the `action` variable as follows:

```
if (worklist != null) {
   String cmd = request.getParameter("SetFilter");
   if (cmd != null) {
      .
      .
      .
   else if (request.getParameter("SetAction") != null) {
      action = request.getParameter("SetAction");
      session.putValue("action", action);
   }
.
.
.
```

Based on the value of the `action` variable, the context is passed to the `taskproperties.jsp` file, as follows:

```
static final String ACTION_DOIT = "0";
static final String ACTION_MARKDONE = "1";
static final String ACTION_UNMARKDONE = "2";
static final String ACTION_TASKPROPERTIES = "3";
static final String ACTION_REASSIGN = "4";
   .
   .
   .
try {
   List vTaskList = (List)session.getValue("tasklist");
   TaskInfo task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
   session.putValue("action", ACTION_DOIT);
   if (action.equals(ACTION_DOIT)) {
      .
      .
      .
   else if (action.equals(ACTION_TASKPROPERTIES)) {
      session.removeValue("error");
      pageContext.forward("taskproperties.jsp?task= "+cmd);;
```

Finally, task properties must be set. The following code, taken from the `taskproperties.jsp` file, shows how to use the `taskSetProperties()` method to set task properties.

```
<%@ page import="
   javax.ejb.*,
   java.rmi.RemoteException,
   java.util.*,
   java.text.*,
   com.bea.wlpi.common.*
   com.bea.wlpi.server.worklist.*
"%>

<%!
   DateFormat df = new SimpleDateFormat("MMM d, yyyy h:mm a");
%>

<%
   Worklist worklist = null;
   TaskInfo task = null;
   String error = null;

   worklist = (Worklist)session.getValue("worklist");

   if (worklist != null) {
      Hashtable h = new Hashtable();
      h.put(javax.naming.Context.SECURITY_PRINCIPAL, session.getValue("name"));
      h.put(javax.naming.Context.SECURITY_CREDENTIALS, session.getValue("password"));
      new javax.naming.InitialContext(h);

      String cmd = request.getParameter("task");
      if (cmd != null) {
         try {
            List vTaskList = (List)session.getValue("tasklist");
            task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
         } catch (Exception e) {
         }
      }
   }

   if (request.getParameter("Set") != null) {
   // Set properties
   task = (TaskInfo)session.getValue("task");
   try {
      worklist.taskSetProperties( task.getTemplateDefinitionId(),
         task.getInstanceId(), task.getTaskId(),
         Integer.parseInt(request.getParameter("Priority")),
         request.getParameter("P0") != null,
         request.getParameter("P1") != null,
         request.getParameter("P2") != null,
         request.getParameter("P3") != null,
         request.getParameter("P4") != null);
      session.removeValue("task");
```

```
      session.removeValue("error");
      pageContext.forward("worklist.jsp");
      return;
   } catch (Exception e) {
      error = e.getMessage();
   }
   }
   if (task != null) {
      session.putValue("task", task);
%>
<html>
<head>
<title>Task Properties</title>
</head>

<body bgcolor=#ffffff>
<font face="Arial" size="4">

<form action="taskproperties.jsp" method="POST">
<table dir="ltr" border="0" cellpadding="1" cellspacing="1">
   <tr>
      <td><img src="logo.gif"></td>
      <td>
         <font color="red" size="7">
         <strong><em>Weblogic</em></strong></font>
         <font color="black" size="7">
         <strong><em>Process Integrator</em></strong></font>
      </td>
   </tr>
   <tr>
      <td bgcolor="navy" valign="top">
         <table dir="ltr" border="0" cellpadding="2" cellspacing="2" >
            <tr>
               <td><font color="yellow">
            <strong>Go ...</strong></font>
         </td>
      </tr>
      <tr>
         <td><a href="worklist.jsp">
            <font color="white">Worklist</font></a>
         </td>
      </tr>
      <tr>
         <td><a href="startworkflow.jsp">
            <font color="white">Start Workflow</font></a>
         </td>
      </tr>
      <tr>
         <td><a href="logoff.jsp">
```

```
            <font color="white">Logoff</font></a></td>
      </tr>
   </table>
   </td>
   <td valign="top">
   <table dir="ltr" border="0" cellpadding="2" cellspacing="2" valign="top">
<%
    if (error != null) {
%>
   <tr>
      <td><font color="red size="4" face="Arial"><%=error%></td>
   </tr>
<%      }
%>
   <tr>
      <td><font color="navy" size="5" face="Arial">
         Task Properties for:
         <strong><%=task.getName()%></strong></font>
      </td>
   </tr>
   <tr>
      <td>
      <table border="0" cellpadding="2" cellspacing="2">
         <tr>
            <td><p align="right">Workflow:</p></td>
            <td><%=task.getWorkflow()%> <%=task.getWorkflowId()%></td>
         </tr>
         <tr>
            <td><p align="right">Comment:</p></td>
            <td><%=task.getComment()%></td>
         </tr>
         <tr>
            <td><p align="right">Started:</p></td>
           <td><%=task.getStarted() == null ? "" : df.format(task.getStarted())%>
            </td>
         </tr>
         <tr>
            <td><p align="right">Completed:</p></td>
            <td><%=task.getCompleted() == null ? "" : df.format(task.getCompleted())%>
            </td>
         </tr>
         <tr>
            <td><p align="right">Due:</p></td>
            <td><%=task.getDue() == null ? "" : df.format(task.getDue())%></td>
         </tr>
         <tr>
            <td><p align="right">Priority:</p></td>
            <td><select name="Priority" size="1">
               <option
```

```
                <%=task.getPriority()==0 ? "selected" : ""%> value="0">Low
              </option>
              <option
                 <%=task.getPriority()==1 ? "selected" : ""%> value="1">Medium
              </option>
              <option
                 <%=task.getPriority()==2 ? "selected" : ""%> value="2">High
              </option>
              </select>
           </td>
        </tr>
        <tr>
           <td valign="top"><p align="right">Permissions:</p></td>
           <td>
           <table border="0" cellpadding="0" cellspacing="0">
           <tr>
              <td>
                 <input type="checkbox" name="P0"
                 <%=task.getDoneWithoutDoit() ? "checked" : ""%> value="ON">
                 Mark done without executing
              </td>
           </tr>
           <tr>
              <td>
                 <input type="checkbox" name="P1"
                    <%=task.getDoitIfDone() ? "checked" : ""%> value="ON">
                    Re-execute if marked done
              </td>
           </tr>
           <tr>
              <td><input type="checkbox" name="P2"
                    <%=task.getUnmarkDone() ? "checked" : ""%> value="ON">
                    Unmark if marked done
              </td>
           </tr>
   <tr>
      <td><input type="checkbox" name="P3"
         <%=task.getModifiable() ? "checked" : ""%> value="ON">
         Allow modification
      </td>
   </tr>
   <tr>
      <td><input type="checkbox" name="P4"
         <%=task.getReassignment() ? "checked" : ""%> value="ON">
         Allow reassignment
      </td>
   </tr>
</table>
</td>
```

```
</tr>
<tr>
   <td></td>
   <td><input type="submit" name="Set" value="Set Properties "></td>
</tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
</table>
</form>
<%
    }
%>
</font>
</body>
</html>
```

The template definition, instance, and task IDs, and current task properties are obtained using the methods associated with the `com.bea.wlpi.common.TaskInfo` object, `task`. The `task` object can be obtained using the methods described in "Getting All Tasks" on page 21-4.

Part V Monitoring

CHAPTER

22 Monitoring Run-Time Workflow Instances

This section describes how to monitor run-time workflow instances, including the following topics:

- Getting Workflow Instances
- Checking for Workflow Instances
- Getting Workflow Instance Tasks
- Getting Workflow Instance Information
- Getting a Count of Workflow Instances
- Deleting Workflow Instances
- Querying the Run-Time Workload
- Querying the Run-Time Statistics

For information about monitoring run-time workflow instances using the WebLogic Integration Studio, see "Monitoring Workflows" in *Using the WebLogic Integration Studio*.

Getting Workflow Instances

To get a list of template and template definition instances in an organization, use the following `com.bea.wlpi.server.admin.Admin` methods, respectively.

Method 1

```
public java.util.List getTemplateInstances(
  java.lang.String templateId,
  java.lang.String orgId,
  boolean bStarted,
  java.util.Date from,
  java.util.Date to,
  int start,
  int max
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public java.util.List getTemplateDefinitionInstances(
  java.lang.String templateDefinitionId,
  java.lang.String orgId,
  boolean bStarted,
  java.util.Date from,
  java.util.Date to,
  int start,
  int max
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getTemplateInstances()` and `getTemplateDefinitionInstances()` method parameters for which you must specify values.

Table 22-1 getTemplateInstances() and getTemplateDefinitionInstances() Method Parameters

Parameter	Description	Valid Values
templateId	ID of the template for which you want to get instances.	Valid template ID. To get the template ID, you need to obtain the `TemplateInfo` object corresponding to the template for which you want to get instances, as described in "Getting the Templates for an Organization" on page 13-5. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25. The `getId()` method returns the `templateId`. Each `templateId` may have multiple definitions. To determine which template definition to use, WebLogic Integration selects the version with the most relevant effective date (current or past) and expiration date (current or future) from the set of active template definitions. For information about defining the effective and expiration dates for the template definitions and about how to make them active, see "Defining Workflow Templates" in *Using the WebLogic Integration Studio.*

Table 22-1 getTemplateInstances() and getTemplateDefinitionInstances() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition for which you want to get instances.	Valid template definition ID. To get the template definition ID, you need to obtain the `TemplateDefinitionInfo` object corresponding to the template definition for which you want to get instances, as described in "Creating a Template Definition" on page 14-2. To get the template definition ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `TemplateDefintionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.
orgId	ID of the organization for which you want to get template or template definition instances.	Valid organization ID. For information about getting the organization IDs, see "Managing the Active Organization" on page 19-1.
bStarted	Boolean flag specifying whether to query by start date or completion date.	`true` (instance start date is used for the query) or `false` (instance completion date is used for the query).
from	Lower bound of the date range.	`java.util.Date` object.
to	Upper bound of the date range.	`java.util.Date` object.
start	Offset at which the returned list starts, enabling clients to display long lists incrementally.	Integer value.

Table 22-1 getTemplateInstances() and getTemplateDefinitionInstances() Method Parameters

Parameter	Description	Valid Values
max	Maximum number of items to return, enabling clients to limit the number of items displayed at one time.	Integer value.

These methods return a list of `com.bea.wlpi.common.InstanceInfo` objects corresponding to the template and template definition instances, respectively. To access information about each instance, use the `InstanceInfo` object methods described in "InstanceInfo Object" on page B-6.

For example, the following code gets all template and template definition instances, respectively, corresponding to the specified ID value, for the organization specified by the value of the `activeOrgId` variable. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List tempinst = admin.getTemplateInstances(templateId,
   activeOrgId, true, dateFrom, dateTo, 20, 20);

List tempdefinst = admin.getTemplateDefinitionInstances(
   templatedefId, activeOrgId, true, dateFrom,
   dateTo, 20, 20);
```

In the examples, the following parameters are set:

- `bStarted` is set to `true`, so information is queried by the start date.
- Lower and upper bounds of the date range are set using the `dateFrom` and `dateTo` `java.util.Date` object variables, respectively.
- Offset value is set to `20`.
- Maximum number of values displayed at any given time is set to `20`.

For more information about the `getTemplateInstances()` and `getTemplateDefinitionInstances()` methods, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Checking for Workflow Instances

This section describes the methods that you can use to check for a workflow template or template definition instance.

Checking for a Workflow Template Instance

To check whether a workflow template instance is currently running, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public boolean checkForTemplateInstances(
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `checkForTemplateInstances()` method parameter for which you must specify a value.

Table 22-2 checkForTemplateInstances() Method Parameter

Parameter	Description	Valid Values
templateId	ID of the template for which you want to check for running instances.	Valid template ID. To get the template ID, you need to obtain the `TemplateInfo` object corresponding to the template for which you want to check for instances, as described in "Getting the Templates for an Organization" on page 13-5. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25. The `getId()` method returns the `templateId`. Each `templateId` may have multiple definitions. To determine which template definition to use, WebLogic Integration selects the version with the most relevant effective (current or past) and expiration (current or future) dates from the set of active template definitions. For information about defining the effective and expiration dates for the template definitions, and about how to make them active, see "Defining Workflows Templates" in *Using the WebLogic Integration Studio.*

The method returns a Boolean value indicating whether or not a template instance exists.

For example, the following code checks whether a template instance, corresponding to the specified ID value, is currently running. `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
boolean tempexists = admin.checkForTemplateInstances(templateId);
```

For more information about the `checkForTemplateInstances()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Checking for a Workflow Template Definition

To check for a workflow template definition instance, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List checkForTemplateDefinitionInstances(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `checkForTemplateDefinitionInstances()` method parameter for which you must specify a value.

Table 22-3 checkForTemplateInstances() Method Parameter

Parameter	Description	Valid Values
`templateDefinitionId`	ID of the template definition for which you want to check for instances.	Valid template definition ID. To get the template definition ID, you need to obtain the `TemplateDefinitionInfo` object corresponding to the template definition for which you want to get instances, as described in "Creating a Template Definition" on page 14-2. To get the template definition ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `TemplateDefintionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.

The method returns a Boolean value indicating whether or not a template or template definition instance exists.

For example, the following code checks whether or not any template definition instances exist corresponding to the specified ID value. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
boolean tempexists = admin.checkForTemplateInstances(templateId);

boolean tempdefexists = admin.checkForTemplateDefinitionInstances(
   templateDefinitionId);
```

For more information about the `checkForTemplateDefinitionInstances()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting Workflow Instance Tasks

To get a list of tasks in a workflow instance, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List getInstanceTasks(
  java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getInstanceTasks()` method parameter for which you must specify a value.

Table 22-4 getInstanceTasks() Method Parameter

Parameter	Description	Valid Values
instanceId	ID of the workflow instance for which you want to get tasks.	Valid workflow instance ID. To obtain the workflow instance ID, you need to obtain the `InstanceInfo` object corresponding to the instance for which you want to get tasks, as described in "Getting Workflow Instances" on page 22-2. To get the instance ID, use the following `com.bea.wlpi.common.InstanceInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `InstanceInfo` object, see "InstanceInfo Object" on page B-6.

The method returns a list of `com.bea.wlpi.common.TaskInfo` objects. To access information about each task, use the `TaskInfo` object methods described in "TaskInfo Object" on page B-20.

For example, the following code gets the tasks for the workflow instance specified by the value of the `instanceId` variable. `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List tasks = admin.getInstanceTasks(instanceId);
```

For more information about the `getInstanceTasks()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting Workflow Instance Information

To get workflow instance information, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public com.bea.wlpi.common.InstanceInfo getInstanceInfo(
  java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getInstanceInfo()` method parameter for which you must specify a value.

Table 22-5 getInstanceTasks() Method Parameter

Parameter	Description	Valid Values
instanceId	ID of the workflow instance for which you want to get information.	Valid workflow instance ID. To obtain the workflow instance ID, you need to obtain the `TaskInfo` object corresponding to the task for which you want to get workflow instance information, as described in "Getting Tasks" on page 15-1. To get the task ID, use the following `com.bea.wlpi.common.TaskInfo` method: `public final java.lang.String getInstanceId()` **Note:** For more information about the methods available to the `TaskInfo` object, see "TaskInfo Object" on page B-20.

The method returns a `com.bea.wlpi.common.InstanceInfo` object. To access information about the workflow instance, use the `InstanceInfo` object methods described in "InstanceInfo Object" on page B-6.

For example, the following code gets the information for the workflow instance specified by the value of the `instanceId` variable. `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
InstanceInfo = admin.getInstanceInfo(instanceId);
```

For more information about the `getInstanceInfo()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Getting a Count of Workflow Instances

To get a count of workflow instances and/or completed workflow instances, use the following `com.bea.wlpi.server.admin.Admin` methods, respectively.

Method 1

```
public int getInstanceCount(
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public int getInstanceCount(
  java.lang.String templateI,
  boolean completed
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getInstanceCount()` method parameters for which you must specify values.

Table 22-6 getInstanceCount() Method Parameters

Parameter	Description	Valid Values
templateId	ID of the template for which you want to get instances.	Valid template ID. To get the template ID, you need to obtain the `TemplateInfo` object corresponding to the template for which you want to get instances, as described in "Getting the Templates for an Organization" on page 13-5. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25. The `getId()` method returns the `templateId`. Each `templateId` may have multiple definitions. To determine which template definition to use, WebLogic Integration selects the version with the most relevant effective date (current or past) and expiration date (current or future) from the set of active template definitions. For information about defining the effective and expiration dates for the template definitions and about how to make them active, see "Defining Workflow Templates" in *Using the WebLogic Integration Studio.*
completed	Boolean flag specifying whether to count completed workflow instances only.	`true` (count completed workflow instances only) or `false` (count all workflow instances).

Each method returns an integer value corresponding to the number of workflow instances counted based on the specified criteria.

For example, the following code gets the count of all workflow instances and all completed workflow instances, respectively, corresponding to the specified ID value. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
int count = admin.getInstanceCount(templateId);

int completeCount = admin.getInstanceCount(templateId, true);
```

For more information about the `getInstanceCount()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Deleting Workflow Instances

Using the methods described in this section, you can delete:

- A specific workflow instance
- All workflow instances in an organization

Deleting a Specific Workflow Instance

To delete a specific workflow instance, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public void deleteInstance(
  java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteInstance()` method parameter for which you must specify a value.

Table 22-7 deleteInstance() Method Parameter

Parameter	Description	Valid Values
instanceId	ID of the workflow instance for which you want to get tasks.	Valid workflow instance ID. To obtain the workflow instance ID, you need to obtain the `InstanceInfo` object corresponding to the instance for which you want to get tasks, as described in "Getting Workflow Instances" on page 22-2. To get the instance ID, use the following `com.bea.wlpi.common.InstanceInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `InstanceInfo` object, see "InstanceInfo Object" on page B-6.

For example, the following code deletes the instance specified by `instanceId`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.deleteInstance(instanceId);
```

Deleting All Instances of a Workflow Template or Template Definition

To delete instances of a workflow template or template definition in an organization, use the following `com.bea.wlpi.server.admin.Admin` methods, respectively.

Method 1

```
public void deleteTemplateInstances(
  java.lang.String templateId,
  java.lang.String orgId,
  boolean bStarted,
  java.util.Date from,
  java.util.Date to
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public void deleteTemplateDefinitionInstances(
  java.lang.String templateDefinitionId,
  java.lang.String orgId,
```

```
  boolean bStarted,
  java.util.Date from,
  java.util.Date to
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `deleteTemplateInstances()` and `deleteTemplateDefinitionInstances()` method parameters for which you must specify values.

Table 22-8 deleteTemplateInstances() and deleteTemplateDefinitionInstances() Method Parameters

Parameter	Description	Valid Values
templateId	ID of the template for which you want to delete instances.	Valid template ID. To get the template ID, you need to obtain the `TemplateInfo` object corresponding to the template for which you want to delete instances, as described in "Getting the Templates for an Organization" on page 13-5. To get the template ID, use the following `com.bea.wlpi.common.TemplateInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `TemplateInfo` object, see "TemplateInfo Object" on page B-25. The `getId()` method returns the `templateId`. Each `templateId` may have multiple definitions. To determine which template definition to use, WebLogic Integration selects the version with the most relevant effective date (current or past) and expiration date (current or future) from the set of active template definitions. For information about defining the effective and expiration dates for the template definitions and about how to make them active, see "Defining Workflow Templates" in *Using the WebLogic Integration Studio*.

Table 22-8 deleteTemplateInstances() and deleteTemplateDefinitionInstances() Method Parameters (Continued)

Parameter	Description	Valid Values
templateDefinitionId	ID of the template definition for which you want to delete instances.	Valid template definition ID. To get the template definition ID, you need to obtain the `TemplateDefinitionInfo` object corresponding to the template definition for which you want to delete instances, as described in "Creating a Template Definition" on page 14-2. To get the template definition ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `TemplateDefintionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.
orgId	ID of the organization for which you want to delete template instances or template definition instances.	Valid organization ID. For information about getting organization IDs, see "Managing the Active Organization" on page 19-1.
bStarted	Boolean flag specifying whether to delete by start or completion date.	`true` (instance start date is used for the delete) or `false` (instance completion date is used for the delete).
from	Lower bound of the date range.	`java.util.Date` object.
to	Upper bound of the date range.	`java.util.Date` object.

For example, the following code deletes all template instances or template definition instances, respectively, corresponding to the specified ID value, and for the organization specified by value of the `activeOrgId` variable. In this example:

- `bStarted` is set to `true`, so information is queried by the start date.
- Lower and upper bounds of the date range are set using the `dateFrom` and `dateTo` `java.util.Date` object variables, respectively.

In the following example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.deleteTemplateInstances(templateId, activeOrgId, true,
  dateFrom, dateTo);

admin.deleteTemplateDefinitionInstances(
   templatedefId, activeOrgId, true, dateFrom, dateTo);
```

For more information about the `deleteInstance()`, `deleteTemplateInstances()`, and `deleteTemplateDefinitionInstances()` methods, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Querying the Run-Time Workload

To query the run-time workload, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List workloadQuery(
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `workloadQuery()` method parameter for which you must specify a value.

Table 22-9 workloadQuery() Method Parameter

Parameter	Description	Valid Values
xml	Run-time workload query request.	XML document that is compliant with the Workload Request DTD, as described in "Workload Request DTD" on page A-120.

This method returns a workload report in XML format that is compliant with the Workload Response DTD, as described in "Workload Response DTD" on page A-123.

For example, the following code initiates a query on the run-time workload, based on the contents of the `workloadReq` XML document, which is compliant with the Workload Request DTD.

```
String workloadresp = workloadQuery(workloadReq);
```

This method writes its response in the file `workloadresp`, in a format that is compliant with the Workload Response DTD. For more information about the `workloadQuery()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Querying the Run-Time Statistics

To query the run-time statistics, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List statisticsQuery(
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `statisticsQuery()` method parameter for which you must specify a value.

Table 22-10 statisticsQuery() Method Parameter

Parameter	Description	Valid Values
xml	Run-time statistics query request.	XML document that is compliant with the Statistics Request DTD, as described in "Statistics Request DTD" on page A-44.

This method returns a statistics report in XML format that is compliant with the Statistics Response DTD, as described in "Statistics Response DTD" on page A-48.

For example, the following code initiates a query on the run-time statistics, based on the contents of the `statisticsReq` XML document, which is compliant with the Statistics Request DTD.

```
java.lang.String statisticsresp = statisticsQuery(statisticsReq);
```

This method writes its response in the file `statisticsresp`, in a format that is compliant with the Statistics Response DTD. For more information about the `statisticsQuery()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

CHAPTER

23 Monitoring Run-Time Variables

This section explains how to monitor run-time variables, including the following topics:

- Getting Workflow Instance Variables
- Setting Workflow Instance Variables

For information about monitoring run-time variables using the WebLogic Integration Studio, see "Monitoring Workflows" in *Using the WebLogic Integration Studio*.

Getting Workflow Instance Variables

To get a list of the variables associated with a workflow instance, use the following `com.bea.wlpi.server.admin.Admin` method:

```
public java.util.List getInstanceVariables(
  java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `getInstanceVariables()` method parameter for which you must specify a value.

Table 23-1 getInstanceVariables() Method Parameter

Parameter	Description	Valid Values
instanceId	ID of the workflow instance for which you want to get variables.	Valid workflow instance ID. To obtain the workflow instance ID, you need to obtain the `InstanceInfo` object corresponding to the instance for which you want to get tasks, as described in "Getting Workflow Instances" on page 22-2. To get the instance ID, use the following `com.bea.wlpi.common.InstanceInfo` method: `public final java.lang.String getId()` **Note:** For more information about the methods available to the `InstanceInfo` object, see "InstanceInfo Object" on page B-6.

The method returns a list of `com.bea.wlpi.common.VariableInfo` objects. To access information about each variable, use the `VariableInfo` object methods described in "VariableInfo Object" on page B-29.

Note: For variables of type XML, the value returned is of type `byte[]`. In order to print or display the returned value, you must convert it to a `String` value. For example, the following code converts the `variable` value of type `byte[]` to a `String` value:

```
String value=new String((byte[])variable.getValue())
```

For example, the following code gets the variables for the workflow instance corresponding to the specified instance ID. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
List list = admin.getInstanceVariables(instance.getId());
```

The instance ID is obtained using the `getInstanceId()` method associated with the `com.bea.wlpi.common.InstanceInfo` object, `instance`. The `instance` object can be obtained using the methods described in "Getting Workflow Instances" on page 22-2.

For more information about the `getInstanceVariables()` method, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

Setting Workflow Instance Variables

To set variables associated with a workflow instance, use one of the following `com.bea.wlpi.server.admin.Admin` methods:

```
public void setInstanceVariable(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String variable,
  java.lang.Object value
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public void setInstanceVariables(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String variables
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `setInstanceVariable()` method parameters for which you must specify values.

Table 23-2 setInstanceVariable() Method Parameters

Parameter	Description	Description
templateDefinitionId	ID of the template definition for which you set workflow instance variables.	Valid template definition ID. To obtain the template definition instance ID, you need to obtain the `InstanceInfo` object corresponding to the instance for which you want to set instance variables, as described in "Getting Workflow Instances" on page 22-2. To get the instance ID, use the following `com.bea.wlpi.common.InstanceInfo` method: `public final java.lang.String getTemplateDefinitionId()` **Note:** For more information about the methods available to the `InstanceInfo` object, see "InstanceInfo Object" on page B-6.

Table 23-2 setInstanceVariable() Method Parameters (Continued)

Parameter	Description	Description
instanceId	ID of the workflow instance corresponding to the task.	Valid workflow instance ID. To obtain the template definition instance ID, you need to obtain the `InstanceInfo` object corresponding to the instance for which you want to set instance variables, as described in "Getting Workflow Instances" on page 22-2. To get the instance ID, use the following `com.bea.wlpi.common.InstanceInfo` method: `public final java.lang.String getInstanceId()` **Note:** For more information about the methods available to the `InstanceInfo` object, see "InstanceInfo Object" on page B-6.
variable	Name of the variable that you want to set.	Valid variable name. To get the variable name, use the following `com.bea.wlpi.common.VariableInfo` method: `public final String getName()` For information about getting the `VariableInfo` object, see "Getting Workflow Instance Variables" on page 23-1. For more information about the methods available to the `VariableInfo` object, see "VariableInfo Object" on page B-29.
variables	Variable names and values.	Map object with *key-value* pairs, specifying variables to be set as a key, and the desired value as the value.

Table 23-2 setInstanceVariable() Method Parameters (Continued)

Parameter	Description	Description
value	Desired value for the variable.	Value that is valid for the specified variable. For a list of supported conversions for the *value* parameter, see the com.bea.wlpi.server.admin.Admin Javadoc.

For example, the following code updates the specified variable, setting it to the specified `value`. In this example, `admin` represents the `EJBObject` reference to the `Admin` EJB.

```
admin.setInstanceVariable(
   instance.getTemplateDefinitionId(),
   instance.getInstanceId(),
   variable.getName(),
   value
);
```

The variable name is obtained using the `getName()` method associated with the `com.bea.wlpi.common.VariableInfo` object, `variable`. The variable object can be obtained using the method described in "Getting Workflow Instance Variables" on page 23-1. The template definition and workflow instance IDs are obtained using the methods associated with the `com.bea.wlpi.common.InstanceInfo` object, `instance`. The `instance` object can be obtained using the methods described in "Getting Workflow Instances" on page 22-2.

For more information about the `com.bea.wlpi.common.VariableInfo` and `com.bea.wlpi.common.TaskInfo` methods, see "Value Object Summary" on page B-1.

For more information about the `setInstanceVariable()` and `setInstanceVariables()` methods, see the `com.bea.wlpi.server.admin.Admin` Javadoc.

CHAPTER

24 Monitoring Workflow Exceptions

This section explains how to monitor workflow exceptions, including the following topics:

- Overview of Exception Handling
- Creating a Workflow Exception
- Getting Workflow Exception Information
- Invoking a Workflow Exception Handler

For information about monitoring workflow exceptions using the WebLogic Integration Studio, see "Handling Workflow Exceptions" in *Using the WebLogic Integration Studio*.

Overview of Exception Handling

The BEA WebLogic Integration exception handling facility enables you to generate, trap, and respond to generated exception conditions at run time. Exception handling is performed at the workflow level, rather than the task level.

The following sections describe workflow exceptions and exception handlers. For more information about exception handling, see "Handling Workflow Exceptions" in *Using the WebLogic Integration Studio*.

Workflow Exception

Workflow exceptions are either abnormal conditions that you define within a workflow, or typical run-time server exceptions that you trap and handle accordingly. A workflow exception may be thrown in response to one of numerous conditions, each of which is associated with an appropriate severity level.

The following table describes the levels of severity defined for workflow exceptions, including the severity code integer equivalents.

Table 24-1 Workflow Exception Severity Levels

Severity Level	Severity Code	Description
`System error`	`ERROR_SYSTEM` (1)	Fatal exception occurred during processing of a user request.
`Workflow error`	`ERROR_WORKFLOW` (2)	Fatal, illegal condition occurred, such as inconsistent workflow state.
`Workflow warning`	`WARNING_WORKFLOW` (3)	Nonfatal workflow condition occurred; the user can rectify it manually.
`Unknown error`	`ERROR_UNKNOWN` (0)	System-specific error occurred; information about it is available for internal use only.
`Custom error`	`ERROR_CUSTOM` (4)	Custom error raised either by an application calling the `com.bea.wlpi.server.worklist.Worklist.invokeWorkflowExceptionHandler()` method or by the `Invoke Error Handler` action within the Studio.

The `com.bea.wlpi.common.WorkflowException` object encapsulates a workflow exception that has occurred during the execution of a remote EJB call.

Exceptions are sent to the `wlpiError` JMS topic. For more information about connecting to JMS, see "Establishing JMS Connections" on page 6-1.

Workflow Exception Handler

All workflow template definitions specify at least one *exception handler*, the system exception handler. The system exception handler is, by default, the initial exception handler; it is invoked whenever an exception occurs. You can also define custom exception handlers. The active exception handler for a workflow template definition can be changed repeatedly in the course of the workflow.

Creating a Workflow Exception

The following table lists the constructors for creating a new `WorkflowException` object.

Table 24-2 WorkflowException Object Constructors

This Constructor	Creates a . . .
`public WorkflowException (java.lang.Exception e)`	Workflow exception containing a nested exception. The specified exception is assigned the `ERROR_SYSTEM` severity, unless it belongs to the `WorkflowException` class, in which case the new exception inherits its severity from the nested exception.
`public WorkflowException (java.lang.Exception e, int severity)`	Workflow exception containing a nested exception with the specified severity. For a list of valid severity levels, see the table "Workflow Exception Severity Levels" on page 24-2.
`public WorkflowException( java.lang.Exception e, int msgNum, int severity)`	Workflow exception containing a nested exception with the specified message code and severity. For a list of valid message codes, see the `com.bea.wlpi.common.Messages` Javadoc. For a list of valid severity levels, see the table "Workflow Exception Severity Levels" on page 24-2.

Table 24-2 WorkflowException Object Constructors

This Constructor	Creates a . . .
`public WorkflowException( java.lang.Exception e, int msgNum, java.lang.Object[] args)`	Workflow exception containing a nested exception with the specified message code and a list of arguments for substitution into a localized message string. For a list of valid message codes, see the `com.bea.wlpi.common.Messages` Javadoc.
`public WorkflowException( java.lang.Exception e, int msgNum, java.lang.Object[] args, int severity)`	Workflow exception containing a nested exception with the specified message number, a list of arguments for substitution into a localized message string, and a message severity. For a list of valid message codes, see the `com.bea.wlpi.common.Messages` Javadoc. For a list of valid severity levels, see the table "Workflow Exception Severity Levels" on page 24-2.
`public WorkflowException(int msgNum)`	Workflow exception with the specified error code. The specified exception is assigned the `ERROR_SYSTEM` severity, by default. For a list of valid message codes, see the `com.bea.wlpi.common.Messages` Javadoc.
`public WorkflowException(int msgNum, java.lang.Object[] args)`	Workflow exception with the specified error code and list of arguments for substitution into a localized message string. The specified exception is assigned the `ERROR_SYSTEM` severity, by default. For a list of valid message codes, see the `com.bea.wlpi.common.Messages` Javadoc.

Table 24-2 WorkflowException Object Constructors

This Constructor	Creates a . . .
`public WorkflowException(int msgNum, int severity)`	Workflow exception with the specified error code and severity level. For a list of valid message codes, see the `com.bea.wlpi.common.Messages` Javadoc. For a list of valid severity levels, see the table "Workflow Exception Severity Levels" on page 24-2.
`public WorkflowException(int msgNum, java.lang.Object[] args, int severity)`	Workflow exception with the specified error code, a list of arguments for substitution into a localized message string, and a message severity. For a list of valid message codes, see the `com.bea.wlpi.common.Messages` Javadoc. For a list of valid severity levels, see the table "Workflow Exception Severity Levels" on page 24-2.
`public WorkflowException(int msgNum, java.lang.Object[] args, int severity, java.lang.String origin)`	Workflow exception with the specified error code, a list of arguments for substitution into a localized message string, a message severity, and message origin. For a list of valid message codes, see the `com.bea.wlpi.common.Messages` Javadoc. For a list of valid severity levels, see the table "Workflow Exception Severity Levels" on page 24-2.

Getting Workflow Exception Information

The following sections explain how to get information about a workflow exception (such as severity, message text, message number, origin, and whether or not the workflow exception was caused by a deadlock) and how to print a stack trace.

Getting the Workflow Exception

To get the original or nested workflow exception, use the following `com.bea.wlpi.common.WorkflowException` methods, respectively:

```
public java.lang.Exception getOriginalException()

public java.lang.Exception getNestedException()
```

The first method returns the original workflow exception. The second method returns the nested workflow exception, or null, if no nested workflow exception exists.

For example, the following code gets the nested workflow exception for a given workflow exception:

```
try {
   // some methods;
}
catch(WorkflowException e) {
   Exception nested e.getNestedException();
}
```

For more information about the `getOriginalException()` and `getNestedException()` methods, see the `com.bea.wlpi.common.WorkflowException` Javadoc.

Getting the Severity

To get the message severity for a workflow exception, use one of the following `com.bea.wlpi.common.WorkflowException` methods:

Method 1 `public java.lang.String getSeverity()`

Method 2 `public java.lang.String getSeverityDescription()`

Method 3 `public java.lang.String getLocalizedSeverityDescription()`

The first method returns the message severity code as an integer value. The second method returns the message severity value as a string value. The third method returns the localized message severity code as a string value. For information about severity level values and codes, see the table "Workflow Exception Severity Levels" on page 24-2.

For example, the following code gets the message severity code for a given workflow exception:

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String severity e.getSeverity();
}
```

For more information about the `getSeverity()` method, see the `com.bea.wlpi.common.WorkflowException` Javadoc.

Getting the Message Text

To get the message text associated with a workflow exception, use one of the following `com.bea.wlpi.common.WorkflowException` methods:

Method 1 `public java.lang.String getMessage()`

Method 2 `public java.lang.String getLocalizedMessage()`

The first method returns one of the following: the message text associated with a nested message, if available; the message text in the language of the system locale from which the exception was thrown; or null if no message text was created.

The second method returns the localized message text, if available, the message text in the language of the system locale from which the exception was thrown, or null, if no message text was created.

For example, the following code gets the message text for a given workflow exception:

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String message e.getMessage();
}
```

For more information about the `getMessage()` or `getLocalizedMessage()` method, see the `com.bea.wlpi.common.WorkflowException` Javadoc.

Getting the Message Number

To get the message number for a workflow exception, use the following `com.bea.wlpi.common.WorkflowException` method:

```
public java.lang.String getMessageNumber()
```

For example, the following code gets the message text for a given workflow exception:

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String number e.getMessageNumber();
}
```

For more information about the `getMessageNumber()` method, see the `com.bea.wlpi.common.WorkflowException` Javadoc.

Getting the Origin

To get the origin (workflow component) from which the workflow exception originated, use the following `com.bea.wlpi.common.WorkflowException` method:

```
public java.lang.String getOrigin()
```

For example, the following code gets the origin text for a given workflow exception:

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String origin e.getOrigin();
}
```

For more information about the `getOrigin()` method, see the `com.bea.wlpi.common.WorkflowException` Javadoc.

Determining Whether a Workflow Exception Resulted from a Database Deadlock

To determine whether or not the workflow exception resulted from a deadlock, use one of the following `com.bea.wlpi.common.WorkflowException` methods:

Method 1 `public boolean isDeadlock()`

Method 2 `public static boolean isDeadlock(java.lang.Exception e)`

Each method returns `true` if the workflow exception resulted from a database deadlock, and `false` otherwise.

The second method calls the `getOrginalException()` method to retrieve the original exception. If the original exception is an instance of `java.sql.SQLException`, the method calls the `getSQLState()` method to determine whether the original problem was a database deadlock.

For example, the following code determines whether a method is deadlocked for a given workflow exception:

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String boolean e.isDeadlock();
}
```

For more information about the `isDeadlock()` methods, see the `com.bea.wlpi.common.WorkflowException` Javadoc.

Printing the Stack Trace

To print a stack trace, use one of the following `com.bea.wlpi.common.WorkflowException` methods:

Method 1 `public void printStackTrace()`

Method 2 `public void printStackTrace(java.io.PrintStream stream)`

Method 3 `public void printStackTrace(java.io.PrintWriter writer)`

Each method prints the workflow exception and its back-trace. The first method prints the information to the standard error stream (`System.err`); the second, to the specified print stream; and the third, to the specified print writer.

The first line of the output contains the result of calling the `toString()` method for this object. The remaining content represents the data previously recorded by the `fillInStackTrace()` method. If the workflow exception contains a nested workflow exception, the method prints its stack trace, as well.

For example, the following code prints the stack trace for a given workflow exception:

```
try {
   // some methods;
}
catch(WorkflowException e) {
   e.printStackTrace();
}
```

For more information about the `printStackTrace()` methods, see the `com.bea.wlpi.common.WorkflowException` Javadoc.

Invoking a Workflow Exception Handler

To invoke a workflow exception handler for a workflow instance, use one of the following `com.bea.wlpi.server.worklist.Worklist` methods:

Method 1

```
public java.lang.String invokeWorkflowExceptionHandler(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String handlerName,
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

Method 2

```
public java.lang.String invokeWorkflowExceptionHandler(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String handlerName,
  java.lang.String xml,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

The following table describes the `invokeWorkflowExceptionHandler()` method parameters for which you must specify values.

Table 24-3 invokeWorkflowExceptionHandler() Method Parameters

Parameter	Description	Valid Values
templateDefinitionId	ID of the workflow template definition for which you are invoking the error handler.	Valid template definition ID. To get the template definition ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final String getId()` For more information about the methods available to the `TemplateDefinitionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.
instanceId	ID of the workflow instance.	ID of the workflow instance corresponding to the template definition. To get the instance ID, use the following `com.bea.wlpi.common.TemplateDefinitionInfo` method: `public final String getInstanceId()` For information about getting the `TemplateDefinitionInfo` object, see "Getting Workflow Instances" on page 22-2. For more information about the methods available to the `TemplateDefinitionInfo` object, see "TemplateDefinitionInfo Object" on page B-23.
handlerName	Name of the error handler to invoke.	Valid error handler.
xml	User-defined subtree.	XML document defining a subtree.

Table 24-3 invokeWorkflowExceptionHandler() Method Parameters (Continued)

Parameter	Description	Valid Values
transactionId	ID of the transaction. **Note:** This parameter is required only in a clustered environment.	Object specifying a unique transaction ID. To generate a unique transaction ID, create a new `com.bea.wlpi.client.common.GUID` object using the following constructor: `GUID transactionId = new GUID();` For more information about the GUID class, see the `com.bea.wlpi.client.common.GUID` Javadoc.

This method returns an XML document that is compliant with the Client Request DTD, `ClientReq.dtd`, as described in "Client Request DTD" on page A-34. The XML document contains information about the running instance, and can be accessed by parsing the document using an XML parser, such as a SAX (Simple API for XML) parser. For example implementations of a SAX parser, see "Examples of Managing Run-Time Tasks" on page 21-25.

For example, the following code invokes the error handler, `MyErrorHandler`, passes the user-defined tree via the `myXml` string variable, and stores the resulting XML document to the `clientReq` variable. In this example, `worklist` represents the `EJBObject` reference to the `Worklist` EJB.

```
String clientReq = worklist.invokeWorkflowExceptionHandler(
  definition.getId(), instanceID, "MyErrorHandler", myXml);
```

The template definition ID is obtained using the methods associated with the `com.bea.wlpi.common.TemplateDefinitionInfo` object, `definition`. An `InstanceInfo` object can be obtained using the methods described in "Getting Workflow Instances" on page 22-2. The `instanceID` can be obtained using the `InstanceInfo` object methods.

For more information about the `com.bea.wlpi.common.TemplateDefinitionInfo` methods, see "TemplateDefinitionInfo Object" on page B-23. For more information about the `com.bea.wlpi.common.InstanceInfo` methods, see "InstanceInfo Object" on page B-6.

For more information about the `invokeWorkflowExceptionHandler()` methods, see the `com.bea.wlpi.server.worklist.Worklist` Javadoc.

A DTD Formats

This appendix describes the WebLogic Integration DTD formats, including the following:

- Audit DTD
- Business Calendar DTD
- Client Call Addin Request DTD
- Client Call Addin Response DTD
- Client Call Program Request DTD
- Client Call Program Response DTD
- Client Message Box Request DTD
- Client Message Box Response DTD
- Client Request DTD
- Client Set Variables Request DTD
- Client Set Variables Response DTD
- Import Response DTD
- Statistics Request DTD
- Statistics Response DTD
- Template DTD
- Template Definition DTD
- Workload Request DTD

- Workload Response DTD

Audit DTD

The Audit DTD describes the format of the XML document that is used by the auditing facility when generating auditing statistics.

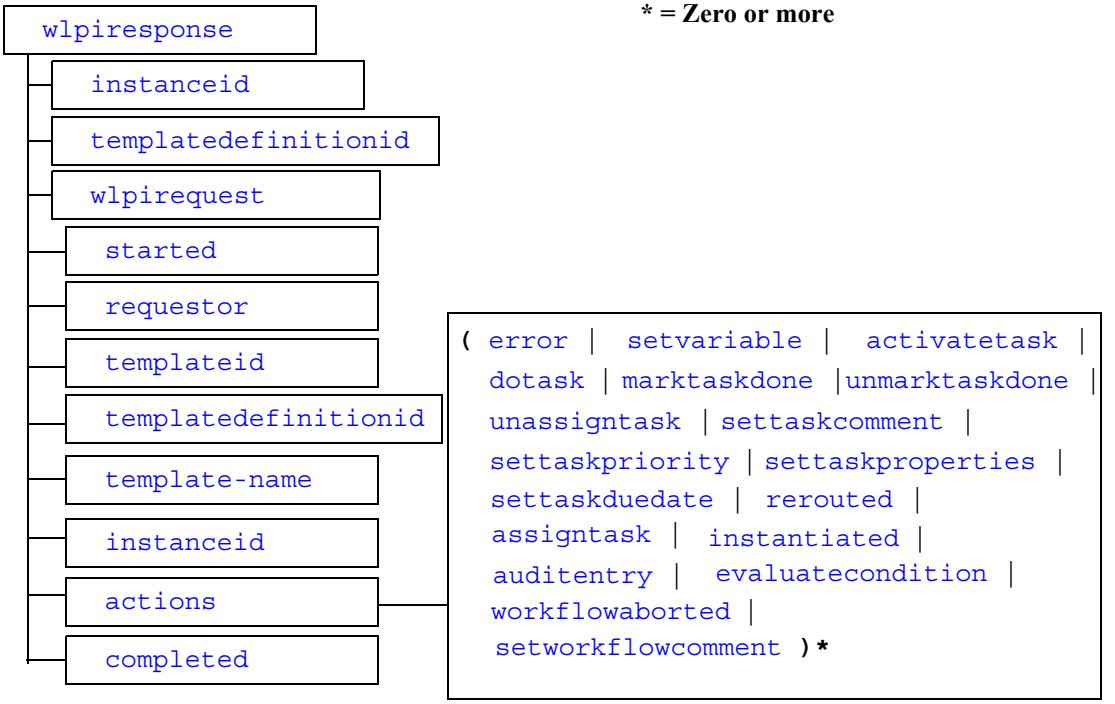
The following sections describe the Audit DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions
- Audit DTD Example

Hierarchy Diagram

The following diagram illustrates the Audit DTD hierarchy.

Figure A-1 Audit DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Audit DTD, `Audit.dtd`.

```
<!ELEMENT wlpiresponse (instanceid, templatedefinitionid, wlpirequest)>
<!ELEMENT wlpirequest (started, requestor, templateid, template-name,
                       templatedefinitionid, instanceid, actions, completed)>
<!ELEMENT actions ((error | setvariable | activatetask | dotask | marktaskdone |
                    unmarktaskdone | unassigntask | settaskcomment |
                    settaskpriority | settaskproperties | settaskduedate |
                    rerouted | assigntask | instantiated | auditentry |
                    evaluatecondition | workflowaborted | setworkflowcomment)*)>
<!ELEMENT completed (#PCDATA)>
<!ELEMENT instanceid (#PCDATA)>
<!ELEMENT requestor (#PCDATA)>
<!ELEMENT started (#PCDATA)>
```

```
<!ELEMENT templatedefinitionid (#PCDATA)>
<!ELEMENT templateid (#PCDATA)>
<!ELEMENT template-name (#PCDATA)>
<!ELEMENT error (#PCDATA)>
<!ATTLIST error time CDATA #REQUIRED id CDATA #REQUIRED>
<!ELEMENT setvariable (#PCDATA)>
<!ATTLIST setvariable time CDATA #REQUIRED variable NMTOKEN #REQUIRED)>
<!ELEMENT activatetask (#PCDATA)>
<!ATTLIST activatetask time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT dotask (#PCDATA)>
<!ATTLIST dotask time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT marktaskdone (#PCDATA)>
<!ATTLIST marktaskdone time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT unmarktaskdone (#PCDATA)>
<!ATTLIST unmarktaskdone time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT unassigntask (#PCDATA)>
<!ATTLIST unassigntask time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT settaskcomment (#PCDATA)>
<!ATTLIST settaskcomment time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT settaskpriority (#PCDATA)>
<!ATTLIST settaskpriority time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT settaskproperties (#PCDATA)>
<!ATTLIST settaskproperties time CDATA #REQUIRED taskid CDATA #REQUIRED
  name CDATA>
<!ELEMENT settaskduedate (#PCDATA)>
<!ATTLIST settaskduedate time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT rerouted (#PCDATA)>
<!ATTLIST rerouted time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT assigntask (#PCDATA)>
<!ATTLIST assigntask time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT instantiated (#PCDATA)>
<!ATTLIST instantiated time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT auditentry ANY>
<!ATTLIST auditentry time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT evaluatecondition (#PCDATA)>
<!ATTLIST evaluatecondition time CDATA #REQUIRED taskid CDATA #REQUIRED
  name CDATA>
<!ELEMENT workflowaborted (#PCDATA)>
<!ATTLIST workflowaborted time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT setworkflowcomment (#PCDATA)>
<!ATTLIST setworkflowcomment time CDATA #REQUIRED taskid CDATA #REQUIRED
  name CDATA>
```

Element Descriptions

The following table describes the elements of the Audit DTD.

Table A-1 Audit DTD Element

Element	Description	Example Value
`actions`	Defines the actions that were performed. Defines zero or more occurrences of the following subelements: ■ `error` ■ `setvariable` ■ `activatetask` ■ `dotask` ■ `marktaskdone` ■ `unmarktaskdone` ■ `unassigntask` ■ `settaskcomment` ■ `settaskpriority` ■ `settaskproperties` ■ `settaskduedate` ■ `rerouted` ■ `assigntask` ■ `instantiated` ■ `auditentry` ■ `evaluatecondition` ■ `workflowaborted` ■ `setworkflowcomment`	See "Audit DTD Example" on page A-11.
`activatetask`	Defines the following attributes: ■ `time`: Time at which the task was activated. ■ `taskid`: ID of the task that has been activated. ■ `name`: Name of the task that has been activated.	`<activatetask time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`

Table A-1 Audit DTD Element (Continued)

Element	Description	Example Value
`assigntask`	Defines the following attributes: ■ `time`: Time at which the task was assigned. ■ `taskid:` ID of the task that has been assigned. ■ `name`: Name of the task that has been assigned.	`<assigntask time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`
`auditentry`	Defines the following attributes: ■ `time`: Time at which the audit entry was logged. ■ `taskid:` ID of the task for which an audit entry has been logged. ■ `name`: Name of the task that for which an audit entry has been logged.	`<auditentry time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`
`completed`	Time at which the task execution completed.	`<completed>2001-06-12 12:45:45.115 </completed>`
`dotask`	Defines the following attributes: ■ `time`: Time at which the task was executed. ■ `taskid:` ID of the task that has been executed. ■ `name`: Name of the task that has been executed.	`<dotask time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`
`error`	Defines the following attributes: ■ `time`: Time at which the error occurred. ■ `id:` Error ID.	`<error time="2001-06-12 12:45:44.824" id="2"/>`

Table A-1 Audit DTD Element (Continued)

Element	Description	Example Value
`evaluatecondition`	Defines the following attributes: ■ `time`: Time at which a condition was evaluated. ■ `taskid`: ID of the task for which a condition has been evaluated. ■ `name`: Name of the task for which the condition has been evaluated.	`<evaluatecondition time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`
`instanceid`	ID of the workflow instance that is the target of the response document.	`<instanceid> 2 </instanceid>`
`instantiated`	Defines the following attributes: ■ `time`: Time at which the task was instantiated. ■ `name`: ID of the task that has been instantiated.	`<instantiated time="2001-06-12 12:45:44.824" name="Start Test"/>`
`marktaskdone`	Defines the following attributes: ■ `time`: Time at which the task was marked complete. ■ `taskid`: ID of the task that has been marked as complete. ■ `name`: Name of the task that has been marked as complete.	`<marktaskdone time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`
`requestor`	ID requestor.	`<requestor> wlisystem </requestor>`
`rerouted`	Defines the following attributes: ■ `time`: Time at which the task was rerouted. ■ `taskid`: ID of the task that has been rerouted. ■ `name`: Name of the task that has been rerouted.	`<rerouted time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`

Table A-1 Audit DTD Element (Continued)

Element	Description	Example Value
settaskcomment	Defines the following attributes: ■ time: Time at which the task comment was set. ■ taskid: ID of the task for which a comment has been set. ■ name: Name of the task for which a comment has been set.	<settaskcomment time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> This is a comment. </settaskcomment>
settaskduedate	Defines the following attributes: ■ time: Time at which the task due date was set. ■ taskid: ID of the task for which a task due date has been set. ■ name: Name of the task for which a task due date has been set.	<settaskduedate time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> </settaskduedate>
settaskpriority	Defines the following attributes: ■ time: Time at which the task priority was set. ■ taskid: ID of the task for which a priority has been set. ■ name: Name of the task for which a priority has been set.	<settaskpriority time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> </settaskpriority>
settaskproperties	Defines the following attributes: ■ time: Time at which the task properties were set. ■ taskid: ID of the task for which properties have been set. ■ name: Name of the task for which properties have been set.	<settaskproperties time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> </settaskproperties>
setvariable	Defines the following attributes: ■ time: Time at which the variable was set. ■ variable: Name of the variable in XML name token (NMTOKEN) format.	<setvariable time="2001-06-12 12:45:44.824" variable="test1"> value1 </setvariable>

Table A-1 Audit DTD Element (Continued)

Element	Description	Example Value
`setworkflowcomment`	Defines the following attributes: ■ `time`: Time at which the workflow comment was set. ■ `taskid:` ID of the task for which a comment has been set. ■ `name`: Name of the task for which a comment has been set.	`<setworkflowcomment time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> </setworkflowcomment>`
`started`	Time at which the task execution started.	`<started> 2001-06-12 12:45:44.824 </started`
`templatedefinitionid`	ID of the workflow template definition that is the target of the response document.	`<templatedefinitionid> 2 </templatedefinitionid >`
`templateid`	ID of the workflow template that is the target of the response document.	`<templateid> 2 </templateid>`
`template-name`	Name of the workflow template.	`<template-name> Start Test </template-name>`
`unassigntask`	Defines the following attributes: ■ `time`: Time at which the task was unassigned. ■ `taskid:` ID of the task for which assignment has been revoked. ■ `name`: Name of the task for which assignment has been revoked.	`<unassigntask time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`

Table A-1 Audit DTD Element (Continued)

Element	Description	Example Value
`unmarktaskdone`	Defines the following attributes: ■ `time`: Time at which the task was marked as incomplete. ■ `taskid`: ID of the task that has been marked as incomplete. ■ `name`: Name of the task that has been marked as incomplete.	`<unmarktaskdone time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`
`wlpirequest`	Defines the request, including the following subelements: ■ `started` ■ `requestor` ■ `templateid` ■ `templatedefinitionid` ■ `instanceid` ■ `actions` ■ `completed`	See "Audit DTD Example" on page A-11.
`wlpiresponse`	Root element. Defines the following subelements: ■ `instanceid` ■ `templatedefinitionid` ■ `wlpirequest`	See "Audit DTD Example" on page A-11.
`workflowaborted`	Defines the following attributes: ■ `time`: Time at which the workflow was aborted. ■ `taskid`: ID of the task that caused the workflow to be aborted. ■ `name`: Name of the task that has been marked as incomplete.	`<workflowaborted time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>`

Audit DTD Example

The following example illustrates a valid application of the Audit DTD:

```
<wlpirequest>
   <started>2001-06-12 12:45:44.824</started
   <requestor>wlisystem</requestor>
   <templateid>2</templateid>
   <template-name>Start Test</template-name>
   <templatedefinitionid>2</templatedefinitionid>
   <instanceid>8010</instanceid>
   <actions>
      <instantiated time="2001-06-12 12:45:44.824"
         name="Start Test"/>
      <setvariable time="2001-06-12 12:45:44.824"
         variable="test1">value1</setvariable>
      <setvariable time="2001-06-12 12:45:44.824"
         variable="test2">value2</setvariable>
      <activatetask time="2001-06-12 12:45:44.824" taskid="2"
         name="Task 1"/>
      <dotask time="2001-06-12 12:45:44.824" taskid="2"
         name="Task 1"/>
      <marktaskdone time="2001-06-12 12:45:44.824"
         taskid="2" name="Task 1"/>
      <workflowdone time="2001-06-12 12:45:45.115"
         name="Start Test"/>
   </actions>
   <completed>2001-06-12 12:45:45.115</completed>
</wlpirequest>
```

Business Calendar DTD

The Business Calendar DTD describes the format of the XML document that is used to create business calendars. Business calendars are used to define the operating hours for an organization. For more information about configuring business calendars, see “Configuring Business Calendars” on page 12-1.

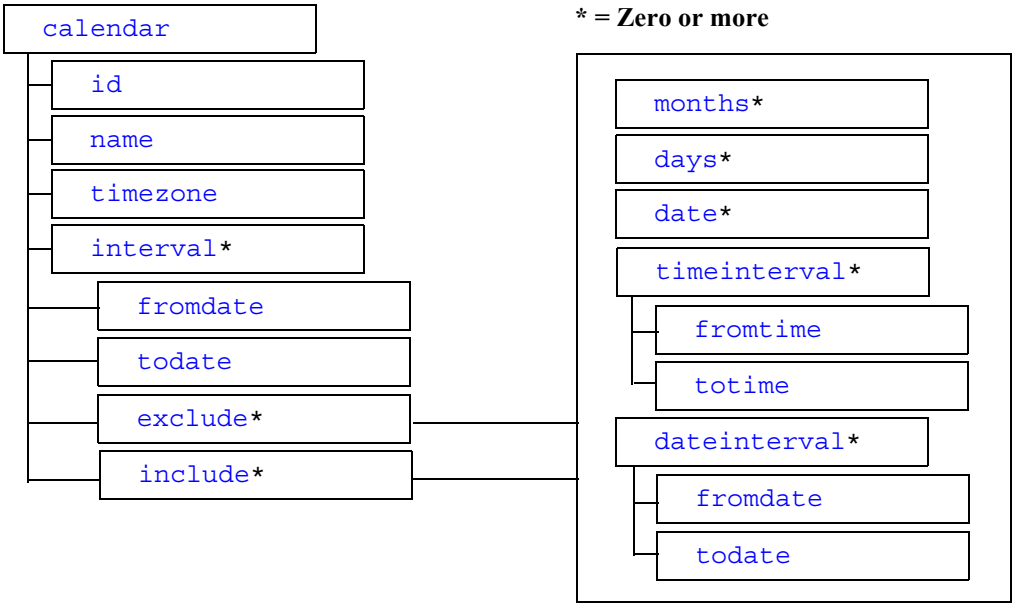
The following sections describe the Business Calendar DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions
- Business Calendar DTD Example

Hierarchy Diagram

The following diagram illustrates the Business Calendar DTD hierarchy.

Figure A-2 Business Calendar DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Business Calendar DTD, `BusinessCalendar.dtd`:

```
<!ELEMENT calendar (id, name, timezone, interval*)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT dateinterval (fromdate, todate)>
<!ELEMENT days (#PCDATA)>
<!ELEMENT exclude (months*, days*, date*, timeinterval*,
dateinterval*)>
<!ELEMENT fromdate (#PCDATA)>
<!ELEMENT fromtime (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT include (months*, days*, date*, timeinterval*,
dateinterval*)>
<!ELEMENT interval (fromdate, todate, exclude*, include*)>
<!ELEMENT months (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT timeinterval (fromtime, totime)>
<!ELEMENT timezone (#PCDATA)>
<!ELEMENT todate (#PCDATA)>
<!ELEMENT totime (#PCDATA)>
```

Element Descriptions

The following table describes the elements of the Business Calendar DTD.

Table A-2 Business Calendar DTD Elements

Element	Description	Example Value
`calendar`	Root element. You must define the following subelements: ■ `id` ■ `name` ■ `timezone` ■ `interval` (zero or more)	See "Business Calendar DTD Example" on page A-19.

Table A-2 Business Calendar DTD Elements (Continued)

Element	Description	Example Value

`date`	Date to be included or excluded.
	This date must be specified using the following format: *`month dd, yyyy`* where *`month`* specifies the name of the month (such as January), *`dd`* specifies the day, and *`yyyy`* specifies the year.
		`<date>`
		`December 25, 2001`
		`</date>`

`dateinterval`	Interval of time to be included in or excluded from the calendar.
	You must define the following subelements:
	■ `fromdate`
	■ `todate`
		`<dateinterval>`
		`<fromdate>`
		`January 1, 2001`
		`</fromdate>`
		`<todate>`
		`January 1, 2002`
		`</todate>`
		`</dateinterval>`

`days`	Day to be included or excluded. The value of this element is a text string specifying the name of the day (such as Sunday).	`<days>Saturday</days>`

`exclude`	Rule that defines an interval of time to be excluded from the main `interval`.
	You can define the following subelements:
	■ `months` (zero or more)
	■ `days` (zero or more)
	■ `date` (zero or more)
	■ `timeinterval` (zero or more)
	■ `dateinterval` (zero or more)
		`<exclude>`
		`<days>`
		`Saturday, Sunday`
		`</days>`
		`<date>`
		`January 1, 2001`
		`</date>`
		`</exclude>`

Table A-2 Business Calendar DTD Elements (Continued)

Element	Description	Example Value

`fromdate`	Subelement of `interval` or `dateinterval`.
	When used as subelement of `interval`, specifies the start date for an interval. When used as subelement of `dateinterval`, it is specific to the `include` or `exclude`.
	This date must be specified using the following format: *`month dd, yyyy`* where *`month`* specifies the name of the month (such as January), *`dd`* specifies the day, and *`yyyy`* specifies the year.
		`<fromdate>`
		`January 1, 2001`
		`</fromdate>`

`fromtime`	Start time from which the business calendar is in effect, or a subelement of `timeinterval`, in which case it specifies a range of time to include or exclude. This time must be specified using the following format: *`hh:mm`*, where *`hh`* specifies the hour and *`mm`* specifies the minutes.	`08:00`
`id`	Unique ID.	`<id>10234</id>`

`include`	Rule that defines an interval of time to be included in the main `interval`.
	You can define the following subelements:
	■ `months` (zero or more)
	■ `days` (zero or more)
	■ `date` (zero or more)
	■ `timeinterval` (zero or more)
	■ `dateinterval` (zero or more)
		`<include>`
		`<timeinterval>`
		`08:00, 17:00`
		`</timeinterval>`
		`</include>`

Table A-2 Business Calendar DTD Elements (Continued)

Element	Description	Example Value

`interval`	Interval of time that the calendar is in effect.
	You must define the following subelements:
	■ `fromdate`
	■ `todate`
	■ `exclude` (zero or more)
	■ `include` (zero or more)
		`<interval>`
		`<fromdate>`
		`20010101`
		`</fromdate>`
		`<todate>`
		`20020101`
		`</todate>`
		`<exclude>`
		`<date>`
		`20011225`
		`</date>`
		`<days>`
		`Saturday, Sunday`
		`</days>`
		`</exclude>`
		`<include>`
		`<timeinterval>`
		`08:00. 17:00`
		`</timeinterval>`
		`</include>`
		`</interval>`

`months`	Month to be included or excluded.
	The value of this element is a text string specifying the name of the month (such as January).
		`<months>`
		`August`
		`</months>`

`name`	Calendar name.	`<name>`
		`MyCalendar`
		`</name>`

`timeinterval`	Interval of time to be included in or excluded from the calendar.
	You must define the following subelements:
	■ `fromtime`
	■ `totime`
		`<timeinterval>`
		`<fromtime>`
		`17:00:00`
		`</fromtime>`
		`<totime>`
		`08:00:00`
		`</totime>`
		`</timeinterval>`

Table A-2 Business Calendar DTD Elements (Continued)

Element	Description	Example Value

`timezone`	Calendar time zone. (See Note following table for valid values.)	`<timezone>`
		`EST`
		`</timezone>`

`todate`	Date until which the business calendar remains in effect.
	This date must be specified using the following format: *`month dd, yyyy`* where *`month`* specifies the name of the month (such as January), *`dd`* specifies the day, and *`yyyy`* specifies the year.
		`<todate>`
		`January 1, 2002`
		`</todate>`

`totime`	Time until which the business calendar is in effect. This time must be specified using the following format: *`hh:mm`*, where *`hh`* specifies the hour and *`mm`* specifies the minutes.	`<totime>17:00</totime>`

Note: Valid timezone values include: Africa/Lome, GMT, UTC, Atlantic/Faeroe, Atlantic/Canary, Europe/Dublin, Europe/Lisbon, Europe/London, Africa/Luanda, Africa/Porto-Novo, Africa/Bangui, Africa/Kinshasa, Africa/Douala, Africa/Libreville, Africa/Malabo, Africa/Niamey, Africa/Lagos, Africa/Ndjamena, Africa/Tunis, Africa/Algiers, Europe/Andorra, Europe/Tirane, Europe/Vienna, Europe/Brussels, Europe/Zurich, Europe/Prague, Europe/Berlin, Europe/Copenhagen, Europe/Madrid, Europe/Gibraltar, Europe/Budapest, Europe/Rome, Europe/Vaduz, Europe/Luxembourg, Africa/Tripoli, Europe/Monaco, Europe/Malta, Africa/Windhoek, Europe/Amsterdam, Europe/Oslo, Europe/Warsaw, Europe/Stockholm, Europe/Belgrade, Europe/Paris, ECT, Africa/Bujumbura, Africa/Gaborone, Africa/Lubumbashi, Africa/Maseru, Africa/Blantyre, Africa/Maputo, Africa/Kigali, Africa/Khartoum, Africa/Mbabane, Africa/Lusaka, Africa/Harare, CAT, Africa/Johannesburg, Europe/Sofia, Europe/Minsk, Asia/Nicosia, Europe/Tallinn, Africa/Cairo, ART, Europe/Helsinki, Europe/Athens, Asia/Jerusalem, Asia/Amman, Asia/Beirut, Europe/Vilnius, Europe/Riga, Europe/Chisinau, Europe/Bucharest, Europe/Kaliningrad, Asia/Damascus, Europe/Kiev, Europe/Istanbul, EET, Asia/Bahrain, Africa/Djibouti, Africa/Asmera, Africa/Addis_Ababa, EAT, Africa/Nairobi, Indian/Comoro, Asia/Kuwait,

Indian/Antananarivo, Asia/Qatar, Africa/Mogadishu, Africa/Dar_es_Salaam, Africa/Kampala, Asia/Aden, Indian/Mayotte, Asia/Riyadh, Asia/Baghdad, Europe/Simferopol, Europe/Moscow, Asia/Tehran, MET, Asia/Dubai, Indian/Mauritius, Asia/Muscat, Indian/Reunion, Indian/Mahe, Asia/Yerevan, NET, Asia/Baku, Asia/Aqtau, Europe/Samara, Asia/Kabul, Indian/Kerguelen, Asia/Tbilisi, Indian/Chagos, Indian/Maldives, Asia/Dushanbe, Asia/Ashkhabad, Asia/Tashkent, Asia/Karachi, PLT, Asia/Bishkek, Asia/Aqtobe, Asia/Yekaterinburg, Asia/Calcutta, IST, Asia/Katmandu, Antarctica/Mawson, Asia/Thimbu, Asia/Colombo, Asia/Dacca, BST, Asia/Alma-Ata, Asia/Novosibirsk, Indian/Cocos, Asia/Rangoon, Indian/Christmas, Asia/Jakarta, Asia/Phnom_Penh, Asia/Vientiane, Asia/Saigon, VST, Asia/Bangkok, Asia/Krasnoyarsk, Antarctica/Casey, Australia/Perth, Asia/Brunei, Asia/Hong_Kong, Asia/Ujung_Pandang, Asia/Ishigaki, Asia/Macao, Asia/Kuala_Lumpur, Asia/Manila, Asia/Singapore, Asia/Taipei, Asia/Shanghai, CTT, Asia/Ulan_Bator, Asia/Irkutsk, Asia/Jayapura, Asia/Pyongyang, Asia/Seoul, Pacific/Palau, Asia/Tokyo, JST, Asia/Yakutsk, Australia/Darwin, ACT, Australia/Adelaide, Antarctica/DumontDUrville, Pacific/Truk, Pacific/Guam, Pacific/Saipan, Pacific/Port_Moresby, Australia/Brisbane, Asia/Vladivostok, Australia/Sydney, AET, Australia/Lord_Howe, Pacific/Ponape, Pacific/Efate, Pacific/Guadalcanal, SST, Pacific/Noumea, Asia/Magadan, Pacific/Norfolk, Pacific/Kosrae, Pacific/Tarawa, Pacific/Majuro, Pacific/Nauru, Pacific/Funafuti, Pacific/Wake, Pacific/Wallis, Pacific/Fiji, Antarctica/McMurdo, Asia/Kamchatka, Pacific/Auckland, NST, Pacific/Chatham, Pacific/Enderbury, Pacific/Tongatapu, Asia/Anadyr, Pacific/Kiritimati

Business Calendar DTD Example

The following example illustrates a valid application of the Business Calendar DTD:

```
<calendar>
       <id>acme2000</id>
       <name>Acme, Inc. Year 2000 Business Calendar</name>
       <timezone>EST</timezone>
       <interval>
              <fromdate>January 3, 2000</fromdate>
              <todate>December 31, 2000</todate>
              <exclude>
                     <date>January 3, 2000</date>
                     <date>December 25, 2000</date>
                     <days>Saturday, Sunday</days>
                     <dateinterval>
                            <fromdate>
                               December 22, 2000 12:00:00
                            </fromdate>
                            <todate>
                               January 1, 2001 00:00:00
                            </todate>
                     </dateinterval>
                     <timeinterval>
                            <fromtime>17:00:00</fromtime>
                            <totime>08:00:00</totime>
                     </timeinterval>
              </exclude>
       </interval>
</calendar>
```

Client Call Addin Request DTD

The Client Call Addin Request DTD describes the format of the XML document that is used when calling an addin. The Client Call Addin Response DTD describes the format of the returned value. XML documents compliant with the Client Call Addin Request DTD can be passed when a template definition is being created, as part of the `ActionSendXMLToClient` action. For details, see “Template Definition DTD” on page A-54.

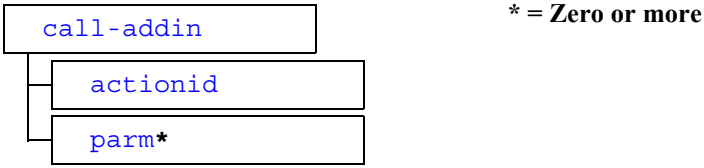
The following sections describe the Client Call Addin Request DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions
- Client Call Addin Request DTD Example

Hierarchy Diagram

The following diagram illustrates the Client Call Addin Request DTD hierarchy.

Figure A-3 Client Call Addin Request DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Client Call Addin Request DTD, `ClientCallAddInReq.dtd`:

```
<!ELEMENT call-addin (actionid, parm*)>
<!ATTLIST call-addin name CDATA #REQUIRED
                     mode (sync|async) "async">
<!ELEMENT actionid (#PCDATA)>
<!ELEMENT parm (#PCDATA)>
```

Element Descriptions

The following table describes the elements of the Client Call Addin Request DTD.

Table A-3 Client Call Addin Request DTD Elements

Element	Description	Example Value

`actionid`	Action ID.	`<actionid>`
		`959395846210`
		`</actionid>`

`call-addin`	Root element. You must define the following subelements: ■ `actionid` ■ `parm` (zero or more) You must define the following attributes: ■ `name`: Fully-qualified Java class name of the worklist addin to invoke, which must implement the `com.bea.wlpi.client.worklist.WorklistAddIn` interface. This attribute is required. ■ `mode`: Program mode specifying whether or not the run-time client blocks while executing the client request. This value can be set to `sync` (synchronous mode), to block the run-time client until completion of the request, or `async` (asynchronous mode), to execute the client request as a background task. This attribute defaults to `async`.	See "Client Call Addin Request DTD Example" on page A-22.

`parm`	Parameter.
	This value can be any character string.
		`<parm>`
		`itemNumber`
		`</parm>`

Client Call Addin Request DTD Example

The following example illustrates a valid application of the Client Call Addin Request DTD:

```
<call-addin name="com.somedomain.someproduct.WorklistAddInImpl"
mode="async">
       <actionid>959395846210</actionid>
       <parm>itemNumber</parm>
</call-addin>
```

Client Call Addin Response DTD

The Client Call Addin Response DTD describes the format of the XML document returned when an addin is called. (The Client Call Addin Request DTD describes the format of the XML document used to call the addin.)

You can use the `XPath` function to extract the returned value from the response file. For example:

```
XPath("/call-addin/child::text()")
XPath("/call-addin/text()")
```

If the return value contains objects such as XML nodes, the XPath expression must be constructed according to that structure.

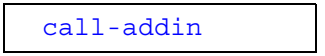
The following sections describe the Client Call Addin Response DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions

Hierarchy Diagram

The following diagram illustrates the Client Call Addin Response DTD hierarchy.

Figure A-4 Client Call Addin Response DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Client Call Addin Response DTD, `ClientCallAddInResp.dtd`.

```
<!ELEMENT call-addin ANY>
```

Element Descriptions

The following table describes the elements of the Client Call Addin Response DTD.

Table A-4 Client Call Addin Response DTD Elements

Element	Description
`call-addin`	Root element. The value of this element can consist of any combination of elements and text.

Client Call Program Request DTD

The Client Call Program Request DTD describes the format of the XML document that is used when an external program is called. (The Client Call Program Response DTD describes the format of the returned value.) XML documents compliant with the Client Call Program Request DTD can be passed when a template definition is being created, as part of the `ActionSendXMLToClient` action. For details, see “Template Definition DTD” on page A-54.

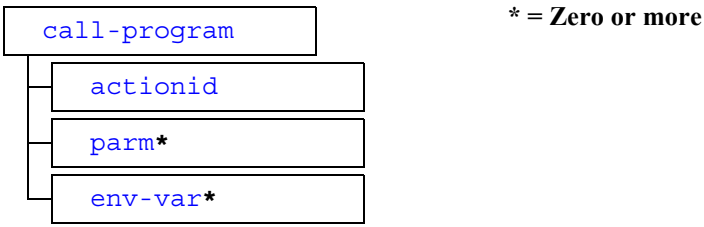
The following sections describe the Client Call Program Request DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions
- Client Call Program Response DTD Example

Hierarchy Diagram

The following diagram illustrates the Client Call Program Request DTD hierarchy.

Figure A-5 Client Call Program Request DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Client Call Program Request DTD, `ClientCallPgmReq.dtd`.

```
<!ELEMENT call-program (actionid, parm*, env-var*)>
<!ATTLIST call-program name CDATA #REQUIRED
                       mode (sync|async) "async">
<!ELEMENT actionid (#PCDATA)>
<!ELEMENT parm (#PCDATA)>
<!ELEMENT env-var (#PCDATA)>
<!ATTLIST env-var name NMTOKEN #REQUIRED>
```

Element Descriptions

The following table describes the elements of the Client Call Program Request DTD.

Table A-5 Client Call Program Request DTD Elements

Element	Description	Example Value
`call-program`	Root element. You must define the following subelements: ■ `actionid` ■ `parm` (zero or more) ■ `env-var` (zero or more) You must define the following attributes: ■ `mode`: Program mode specifying whether or not the run-time client blocks while executing the client request. This value can be set to `sync` (synchronous mode), to block the run-time client until completion of the request, or `async` (asynchronous mode), to execute the client request as a background task. This attribute defaults to `async`. ■ `name`: Program name. This attribute is required.	See "Client Call Program Request DTD Example" on page A-27.

`actionid`	Action ID.	`<actionid>`
		`  992456131534`
		`</actionid>`

`parm`	Parameter definition. This value can be any character string.	`<parm>`
		`  C:\WLPI\readme.txt`
		`</parm>`

`env-var`	Environment variable definition.
	You must define a `name` attribute that specifies the name of the environment variable in XML name token (`NMTOKEN`) format.
		`<env-var name="TEMP">`
		`  C:\TEMP`
		`</env-var>`

Client Call Program Request DTD Example

The following example illustrates a valid application of the Client Call Program Request DTD.

```
<call-program mode="async" name="notepad">
  <actionid>992456131534</actionid>
  <parm>C:\WLPI\readme.txt</parm>
  <env-var name="TEMP">C:\TEMP</env-var>
  <env-var name="TMP">C:\TEMP</env-var>
</call-program>
```

Client Call Program Response DTD

The Client Call Program Response DTD describes the format of the XML document returned when an external program is called. (The Client Call Program Request DTD describes the format of the XML document used to call the external program.)

You can use the `XPath` function to extract the returned value. For example:

```
XPath("/call-program/attribute::exit-value")
XPath("/call-program/@exit-value")
```

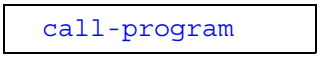
The following sections describe the Client Call Program Response DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions
- Client Call Program Response DTD Example

Hierarchy Diagram

The following diagram illustrates the Client Call Program Response DTD hierarchy.

Figure A-6 Client Call Program Response DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Client Call Program Response DTD, `ClientCallProgramResp.dtd`.

```
<!ELEMENT call-program EMPTY>
<!ATTLIST call-program exit-value NUMBER #REQUIRED>
```

Element Descriptions

The following table describes the elements of the Client Call Program Response DTD.

Table A-6 Client Call Program Response DTD Elements

Element	Description	Example Value
`call-program`	Root element. This element is empty. You must define the `exit-value` attribute to specify a numeric exit code, as retrieved by the operating system, that is meaningful within the context of the calling workflow. Consult the appropriate program documentation for more information on valid exit codes. This attribute is required.	See "Client Call Program Response DTD Example" on page A-29.

Client Call Program Response DTD Example

The following example illustrates a valid application of the Client Call Program Response DTD:

```
<call-program exit-value=0>
</call-program>
```

Client Message Box Request DTD

The Client Message Box Request DTD describes the format of the XML document that is used to prompt: (a) a user to respond to a message via a dialog box; and (b) retrieve the response. (The Client Message Box Response DTD describes the format of the returned value.) XML documents compliant with the Client Message Box Request DTD can be passed when a template definition is being created, as part of the `ActionSendXMLToClient` action. For details, see “Template Definition DTD” on page A-54.

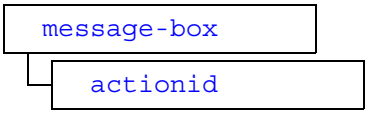
The following sections describe the Client Message Box Request DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions
- Client Message Box Request DTD Example

Hierarchy Diagram

The following diagram illustrates the Client Message Box Request DTD hierarchy.

Figure A-7 Client Message Box Request DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Client Message Box Request DTD, `ClientMsgBoxReq.dtd`.

```
<!ELEMENT message-box (#PCDATA, actionid)>
<!ATTLIST message-box title CDATA #IMPLIED
                      style (plain|information|question|warning|
                      error) "plain"
                      options (ok|ok_cancel|yes_no|
                      yes_no_cancel) "ok">
<!ELEMENT actionid (#PCDATA)>
```

Element Descriptions

The following table describes the elements of the Client Message Box Request DTD.

Table A-7 Client Message Box Request DTD Elements

Element	Description	Example Value

`actionid`	Action ID.	`<actionid>`
		`990705990915`
		`</actionid>`

Table A-7 Client Message Box Request DTD Elements (Continued)

Element	Description	Example Value
`message-box`	Root element. You must define the `actionid` subelement. You must define the following attributes: ■ `title`: Text that appears in the title field of the message box. This attribute is optional. ■ `style`: Message box style. The value of this element can be set to `error`, `information`, `plain`, `question`, or `warning`. It defaults to `plain`. ■ `options`: Message box button options. Table A-8 defines the message box options that are valid based on the `style` value assigned.	See "Client Message Box Request DTD Example" on page A-32.

The following table defines the message box options that are valid based on the `style` value assigned.

Table A-8 Valid Message Box Options Based on Style Value

Message Box Style	Message Box Options
`error`	`ok`, `ok_cancel`
`information`	`ok`
`plain`	`ok`, `ok_cancel`, `yes_no`, `yes_no_cancel`
`question`	`yes_no`, `yes_no_cancel`
`warning`	`ok`, `ok_cancel`

Client Message Box Request DTD Example

The following example illustrates a valid application of the Client Message Box Request DTD.

```
<message-box options="yes_no" style="question"
  title="Credit Check">Does customer 6831 pass credit check?|
  <actionid>990705990915</actionid>
</message-box>
```

Client Message Box Response DTD

The Client Message Box Response DTD describes the format of the XML document returned when a user is prompted for more information via a message dialog box. (The Client Message Box Request DTD describes the format of the XML document used to prompt the user.)

You can use the `XPath` function to extract the returned value. For example:

```
XPath("/message-box/attribute::option")
XPath("/message-box/@option")
```

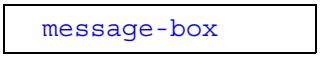
The following sections describe the Client Message Box Response DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Description
- Template Definition DTD Example

Hierarchy Diagram

The following diagram illustrates the Client Message Box Response DTD hierarchy.

Figure A-8 Client Message Box Response DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Client Message Box Response DTD, `ClientMsgBoxResp.dtd`.

```
<!ELEMENT message-box EMPTY>
<!ATTLIST message-box option (ok|yes|no|cancel) #REQUIRED>
```

Element Description

The following table describes the Client Message Box Response DTD element.

Table A-9 Client Message Box Response DTD Element

Element	Description	Example Value
`message-box`	Root element. This element is empty. You must define the `option` attribute to specify the option entered by the user. This value can be `ok`, `yes`, `no`, or `cancel`. This attribute is required.	See "Client Message Box Response DTD Example" on page A-33.

Client Message Box Response DTD Example

The following example illustrates a valid application of the Client Message Box Response DTD:

```
<message-box option=yes>
</message-box>
```

Client Request DTD

The Client Request DTD describes the format of the XML document that is returned by a subset of the runtime management methods described in Part IV, "Run-Time Management."

The returned XML file contains details about all workflow and task updates that occurred as a consequence of the runtime method call. It may also contain requests for the client to handle. Such requests are generated as part of the `ActionSendXMLToClient` action, as described in "Template Definition DTD" on page A-54.

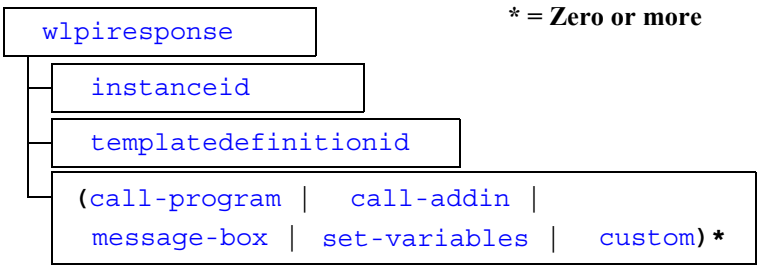
The following sections describe the Client Request DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions
- Entity Descriptions

Hierarchy Diagram

The following diagram illustrates the Client Request DTD hierarchy.

Figure A-9 Client Request DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Client Request DTD, `ClientReq.dtd`.

```
<!ENTITY callpgm   SYSTEM "ClientCallPgmReq.dtd">
<!ENTITY calladdin SYSTEM "ClientCallAddInReq.dtd">
<!ENTITY msgbox    SYSTEM "ClientMsgBoxReq.dtd">
<!ENTITY setvars   SYSTEM "ClientSetVarsReq.dtd">

<!ELEMENT wlpiresponse (instanceid, templatedefinitionid,
       (call-program | call-addin | set-variables |  message-box |
       custom)*)>
<!ELEMENT instanceid (#PCDATA)>
<!ELEMENT templatedefinitionid (#PCDATA)>
&callpgm;
&calladdin;
&msgbox;
&setvars;
<!ELEMENT custom ANY>
```

Element Descriptions

The following table describes the elements of the Client Request DTD.

Table A-10 Client Request DTD Elements

Element	Description	Example Value
`call-addin`	Request for client to call a java addin that is compliant with the "Client Call Addin Request DTD" on page A-19.	See "Client Call Addin Request DTD Example" on page A-22.
`call-program`	Request for client to call an external program that is compliant with the "Client Call Program Request DTD" on page A-24.	See "Client Call Program Request DTD Example" on page A-27.
`custom`	Custom request consisting of any combination of elements and text.	

`instanceid`	ID of the workflow instance that is the target of the response document.	`<instanceid>`
		`3`
		`</instanceid>`

Table A-10 Client Request DTD Elements (Continued)

Element	Description	Example Value
`message-box`	Request for client to respond to a message that is compliant with the "Client Message Box Request DTD" on page A-29.	See "Client Message Box Request DTD Example" on page A-32.
`set-variables`	Request for client to set variables that is compliant with the "Client Set Variables Request DTD" on page A-37.	See "Client Set Variables Request DTD Example" on page A-39.

`templatedefinitionid`	ID of the workflow template definition that is the target of the response document.	`<templatedefinitionid>`
		`2`
		`</templatedefinitionid>`

`wlpiresponse`	Root element. You must define the following subelements: ■ `wlpiresponse` ■ `instanceid` ■ `templatedefinitionid` ■ `call-program | call-addin | set-variables | message-box | custom` (zero or more)	

Entity Descriptions

The following table describes the Client Request DTD entities.

Table A-11 Client Request DTD Entities

Element	Meaning
`calladdin`	Client Call Addin Request DTD, `ClientCallAddInReq.dtd`
`callpgm`	Client Call Program Request DTD, `ClientCallPgmReq.dtd`
`msgbox`	Client Message Box Request DTD, `ClientMsgBoxReq.dtd`
`setvars`	Client Set Variables Request DTD, `ClientSetVarsReq.dtd`

Client Set Variables Request DTD

The Client Set Variables Request DTD describes the format of the XML document that is used to: (a) prompt a user to set variables via a dialog box; and (b) retrieve the response. (The Client Set Variables Response DTD describes the format of the returned value.) XML documents compliant with the Client Set Variables Request DTD can be passed when a template definition is created, as part of the `ActionSendXMLToClient` action. For details, see "Template Definition DTD" on page A-54.

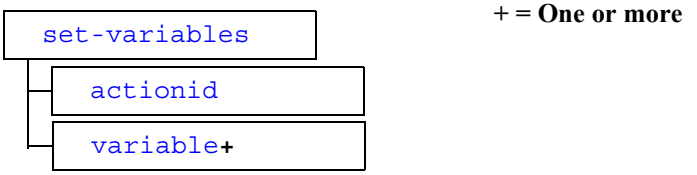
The following sections describe the Client Set Variables Request DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions
- Client Set Variables Request DTD Example

Hierarchy Diagram

The following diagram illustrates the Client Set Variables Request DTD hierarchy.

Figure A-10 Client Set Variables Request DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Client Set Variables Request DTD, `ClientSetVarsReq.dtd`.

```
<!ELEMENT set-variables (actionid, variable+)>
<!ATTLIST set-variables title CDATA #IMPLIED>
<!ELEMENT actionid (#PCDATA)>
<!ELEMENT variable (#PCDATA)>
<!ATTLIST variable name NMTOKEN #REQUIRED
                   prompt NMTOKEN #IMPLIED>
```

Element Descriptions

The following table describes the elements of the Client Set Variables Request DTD.

Table A-12 Client Set Variables Request DTD Elements

Element	Description	Example Value

`actionid`	Action ID.	`<actionid>`
		`991408825931`
		`</actionid>`

`set-variables`	Root element. You must define the following subelements: ■ `actionid` ■ `variable` (one or more) You can also define the `title` attribute to specify the text that appears in the title field of the set variables box. This attribute is optional.	See "Client Set Variables Request DTD Example" on page A-39.

Table A-12 Client Set Variables Request DTD Elements (Continued)

Element	Description	Example Value
`variable`	Variable to be set. You must define the following attributes: ■ `name`: Name of the variable to be set, specified in XML name token (`NMTOKEN`) format. This attribute is required. ■ `prompt`: Text displayed in a dialog box as a cue to the user, specified in XML name token (`NMTOKEN`) format. This attribute is optional.	`<variable name="ShipToState" prompt="An error has occurred in processing this order. Please enter the valid US state abbreviation for shipping:"> </variable>`

Client Set Variables Request DTD Example

The following example illustrates a valid application of the Client Set Variables Request DTD.

```
<set-variables title="Error Warning">
  <actionid>991408825931</actionid>
  <variable name="ShipToState" prompt="An error has occurred in
  processing this order. Please enter the valid US state
  abbreviation for shipping:"></variable>
</set-variables>
```

Client Set Variables Response DTD

The Client Set Variables Response DTD describes the format of the XML document returned when prompting a user for more information via a set-variables dialog box. (The Client Set Variables Request DTD describes the format of the XML document used when prompting the user.)

You can use the `XPath` function to extract the returned value. For example:

```
XPath("/set-variables/child::variable[attribute::name=
      "<var-name>"]/child::text()")
XPath("/set-variables/variable[@name="<var-name>"]/text()")
```

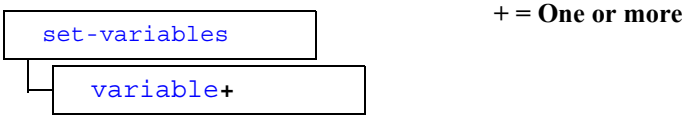
The following sections describe the Client Set Variables Response DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Description
- Client Set Variables Response DTD Example

Hierarchy Diagram

The following diagram illustrates the Client Set Variables Response DTD hierarchy.

Figure A-11 Client Set Variables Response DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Client Set Variables Response DTD, `ClientSetVarResp.dtd`.

```
<!ELEMENT set-variables (variable+)>
<!ELEMENT variable (#PCDATA)>
<!ATTLIST variable name NMTOKEN #REQUIRED>
```

Element Descriptions

The following table describes the elements of the Client Set Variables Response DTD.

Table A-13 Client Set Variables Response DTD Elements

Element	Description	Example Value
`set-variables`	Root element. You must define one or more `variable` subelements.	See "Client Set Variables Response DTD Example" on page A-41.

`variable`	Variable value.
	You must define the `name` attribute specifying the name of the variable that was set in XML name token (`NMTOKEN`) format. This attribute is required.
		`<variable name="ShipToState">`
		`  CA`
		`</variable>`

Client Set Variables Response DTD Example

The following example illustrates a valid application of the Client Set Variables Response DTD:

```
<set-variables>
  <variable name="ShipToState">CA</variable>
</set-variables>
```

Import Response DTD

The Import Response DTD describes the format of the XML document returned when the `importPackage()` method to the `com.bea.wlpi.server.admin.Admin` interface is called. For more information about the `importPackage()` method, "Importing a Package of Publishable Objects" on page 18-6.

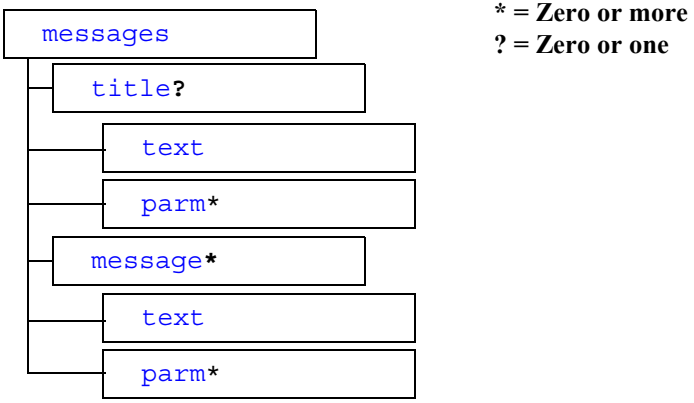
The following sections describe the Import Request DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions

Hierarchy Diagram

The following diagram illustrates the Import Response DTD hierarchy.

Figure A-12 Import Response DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Import Response DTD, `ImportResp.dtd`.

```
<!ELEMENT messages (title?, message*>
<!ELEMENT title (text, parm*)>
<!-- msg-num refers to an internationalized message number in
Messages.properties -->
<!ATTLIST text msg-num NUMBER #REQUIRED>
<!-- the text value is simply the message formatted in the locale
of the server -->
<!ELEMENT text (#PCDATA)>
<!ELEMENT message (text, parm*)>
<!-- each parm element represents a replaceable parameter value to
insert
into the corresponding parameter marker in the localized message -->
<!ELEMENT parm (#PCDATA)>
<!-- type refers to an internationalized object type in
Messages.properties -->
<!ATTLIST parm type NUMBER #IMPLIED>
```

Element Descriptions

The following table describes the elements of the Import Response DTD.

Table A-14 Import Response DTD Elements

Element	Description
`message`	Single message. You must define the following subelements: ■ `text` ■ `parm` (zero or more)
`messages`	List of messages. You must define the following subelements: ■ `title` (zero or one) ■ `message` (zero or more)

Table A-14 Import Response DTD Elements (Continued)

Element	Description
`parm`	Parameter value with which to replace the corresponding parameter marker in the localized message. You can define the following attribute. `type`: Parameter type.
`text`	Message formatted in the locale of the server. You must define the following attribute. `msg-num`: Message number.
`title`	Title for the message list. You must define the following subelements: ■ `text` ■ `parm` (zero or more)

Statistics Request DTD

The Statistics Request DTD describes the format of the XML document that is used when runtime statistics reports are generated. (The Statistics Response DTD describes the format of the returned statistics report.) For information about the method used to generate runtime statistics, see “Querying the Run-Time Statistics” on page 22-19.

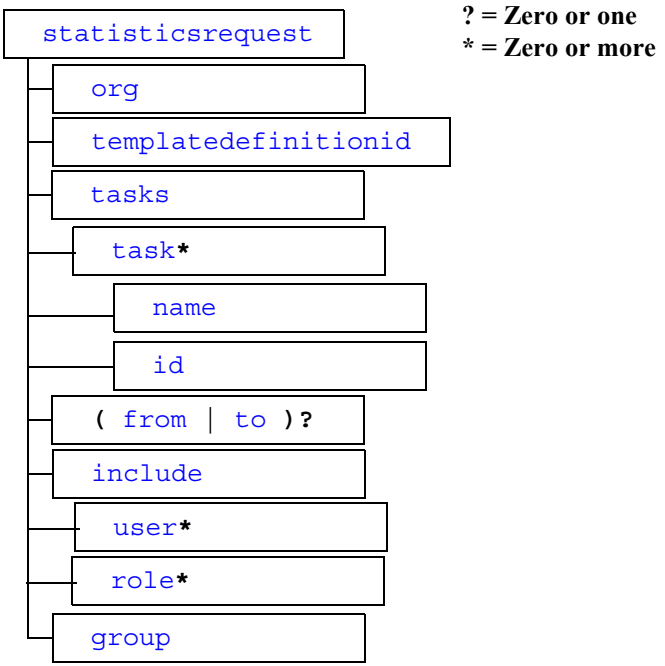
The following sections describe the Statistics Request DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions

Hierarchy Diagram

The following diagram illustrates the Statistics Request DTD hierarchy.

Figure A-13 Statistics Request DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Statistics Request DTD, `StatisticsReq.dtd`.

```
!ELEMENT statisticsrequest (org, templatedefinition,
                            tasks, (from, to)?, include, group)>
<!ELEMENT tasks (task*)>
<!ELEMENT task (name, id)>
<!ELEMENT include (#PCDATA, user* | role*)>
<!ELEMENT org (#PCDATA)>
<!ELEMENT templatedefinitionid (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT group (#PCDATA)>
```

Element Descriptions

The following table describes the elements of the Client Set Variables Response DTD.

Table A-15 Statistics Request DTD Elements

Element	Description
`from`	Date from which the statistics report should be generated. This date must be specified using the following format: *yyyyMMddHHmm*, where *yyyy* specifies the year, *MM* specifies the month, *dd* specifies the day, *HH* specifies the hour, and `mm` specifies the minutes.
`group`	Boolean flag specifying whether or not to group the totals.
`id`	ID of the task to include in the report.
`include`	List of users and/or roles to include in the report. You can define the following subelements: ■ `user` (zero or more) ■ `role` (zero or mroe)

Table A-15 Statistics Request DTD Elements (Continued)

Element	Description
`name`	Name of task to include in thes report.
`org`	Organization for which you are generating a report.
`role`	Name of role to include in the report.
`statisticsrequest`	Root element. You must define the following subelements: ■ `org` ■ `templatedefinitionid` ■ `tasks` ■ (`from`, `to`) (zero or one) ■ `include` ■ `group`
`task`	Task to include in the report. You must define the following subelements: ■ `name` ■ `id`
`tasks`	List of tasks to include in the report. You must define zero or more occurrences of the following subelement: `task`
`templatedefinitionid`	ID of the template definition for which you are generating a report.
`to`	Entity date of the period for which the statistics report should be generated. This date must be specified using the following format: *yyyyMMddHHmm*, where *yyyy* specifies the year, *MM* specifies the month, *dd* specifies the day, *HH* specifies the hour, and *mm* specifies the minutes.
`user`	Name of user to include in the report.

Statistics Response DTD

The Statistics Response DTD describes the format of the XML document returned when runtime statistics are queried. (The Statistics Request DTD describes the format of the XML document used to request the query.) For information about gathering runtime statistics, see “Querying the Run-Time Statistics” on page 22-19.

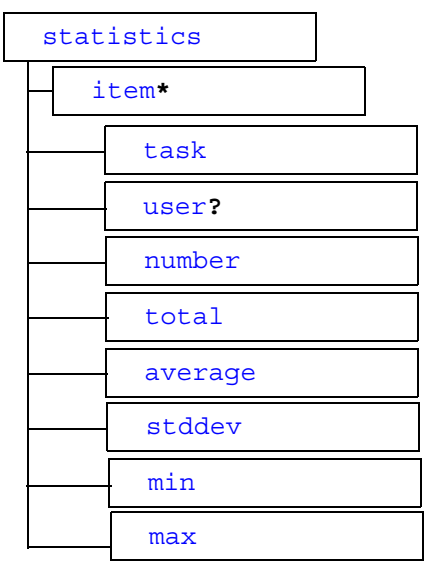
The following sections describe the Statistics Request DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions

Hierarchy Diagram

The following diagram illustrates the Statistics Response DTD hierarchy.

Figure A-14 Statistics Response DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Statistics Response DTD, `StatisticsResp.dtd`.

```
<!ELEMENT statistics (item*)>
<!ELEMENT item (task, user?, number, total, average, stddev,
          min, max)>
<!ELEMENT task (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT total (#PCDATA)>
<!ELEMENT average (#PCDATA)>
<!ELEMENT stddev (#PCDATA)>
<!ELEMENT min (#PCDATA)>
<!ELEMENT max (#PCDATA)>
```

Element Descriptions

The following table describes the elements of the Statistics Response DTD.

Table A-16 Statistics Response DTD Elements

Element	Description
`average`	Average amount of time spent performing a task.
`item`	Item for which you are gathering the workload report. You must define the following subelements: ■ `task` ■ `user` (zero or one) ■ `number` ■ `total` ■ `average` ■ `stddev` ■ `min` ■ `max`
`max`	Maximum amount of time spent performing a task.
`min`	Minimum amount of time spent performing a task.
`number`	Number of tasks performed.
`statistics`	Root element. You must define zero or more occurrences of the following subelement: `item`
`stddev`	Standard deviation of the times spent performing a task; indicates how much the AVERAGE truly represents the numbers upon which it was calculated.
`task`	Task upon which the following are calculated.
`total`	Total amount of time spent performing a task.
`user`	User that performed the task.

Template DTD

The Template DTD describes the format of the XML document that is used to create a workflow template. For information about creating workflow templates, see “Creating and Managing Workflow Templates” on page 13-1.

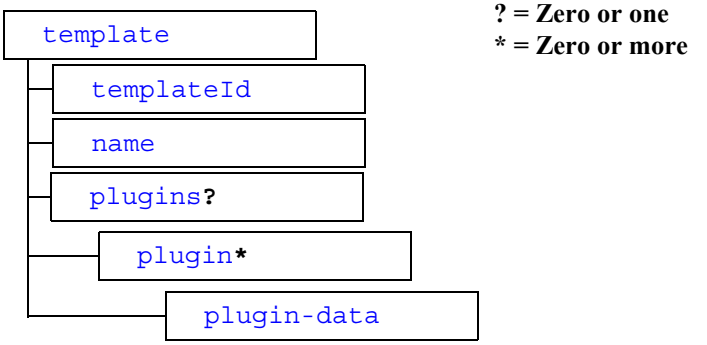
The following sections describe the Template DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions

Hierarchy Diagram

The following diagram illustrates the Template DTD hierarchy.

Figure A-15 Template DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Template DTD, `Template.dtd`.

```
<!ELEMENT template (templateId, name, plugins?)>
<!ELEMENT plugins (plugin*)>
<!ELEMENT plugin (plugin-data)>
<!ATTLIST plugin
  name CDATA #REQUIRED
  major-version CDATA #REQUIRED
  minor-version CDATA #REQUIRED
  vendor CDATA #REQUIRED
  url CDATA>
<!ELEMENT plugin-data ANY>
<!ATTLIST plugin-data
    name CDATA #REQUIRED
    id CDATA #REQUIRED
>
<!ELEMENT templateId #PCDATA>
<!ELEMENT name #PCDATA>
```

Element Descriptions

The following table describes the elements of the Template DTD.

Table A-17 Template DTD Elements

Element	Description
`name`	Name of the template.

Table A-17 Template DTD Elements (Continued)

Element	Description
`plugin`	Plug-in entry. One element appears for each plug-in referenced by a template, and for each plug-in that defines its own template property. Optionally, you can define the following subelement: `plugin-data`. You must define the following attributes: ■ `name`: Plug-in name ■ `major-version`: Major version of the plug-in software ■ `minor-version`: Minor version of the plug-in software ■ `vendor`: Vendor name ■ `url`: URL (defaults to an empty string) For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.
`plugin-data`	Plug-in data consisting of any combination of elements and text. You must define the following attributes: ■ `name`: Name of the plug-in. This attribute is required. ■ `ID`: Unique ID, provided by the plug-in, for the associated value object. This attribute is required. For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.
`plugins`	Plug-in definition, used only if plug-in is provided. You can define zero or more occurrences of the following subelements: `plugin` For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.
`template`	Root element. You must define the following subelements: ■ `templateId` ■ `name` ■ `plugins` (zero or one)
`templateId`	ID of the template.

Template Definition DTD

The Template Definition DTD describes the format of the XML document that is used to create a workflow template definition. For information about creating workflow template definitions, see "Creating and Managing Workflow Template Definitions" on page 14-1.

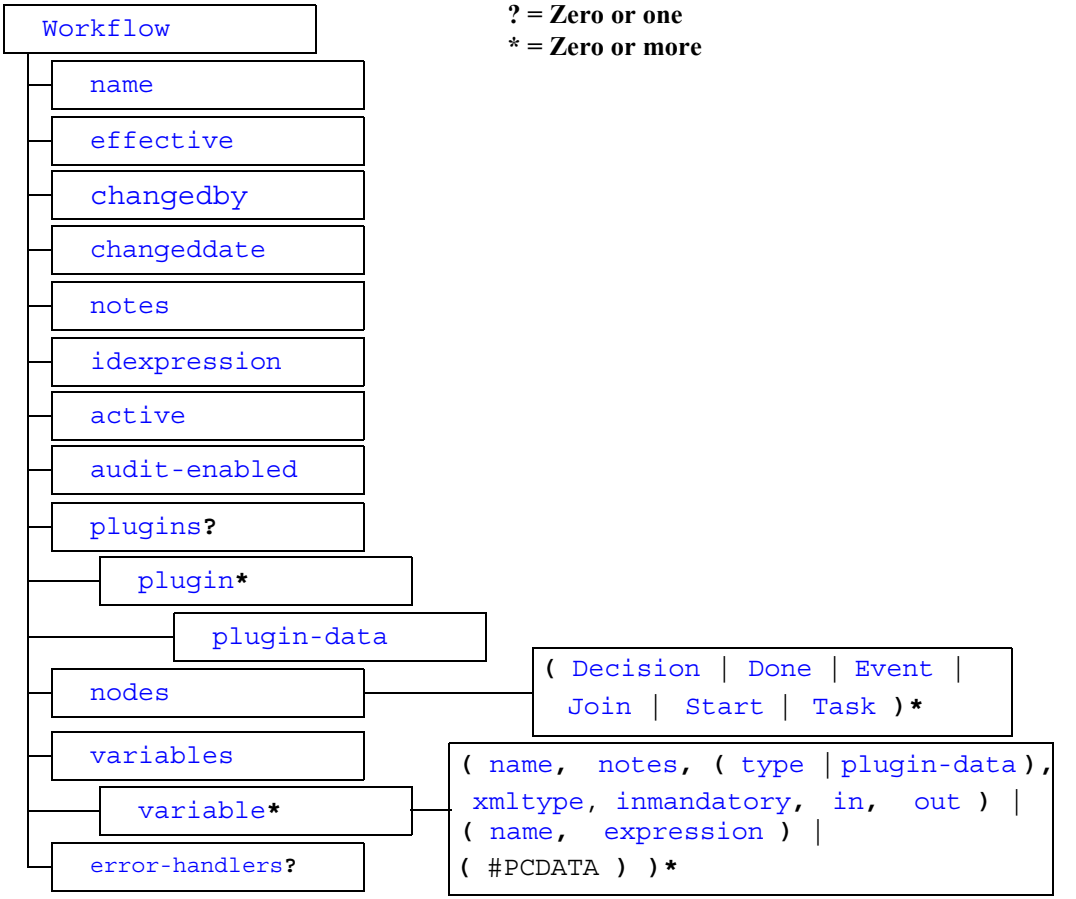
The following sections describe the Template Definition DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions
- Entity Descriptions
- Template Definition DTD Example

Hierarchy Diagram

The following diagram illustrates the Template Definition DTD hierarchy.

Figure A-16 Template Definition DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Template Definition DTD, `TemplateDefinition.dtd`.

Note: Descriptions of the DTD elements are provided in the table "Template Definition DTD Elements" on page A-63. Descriptions of the DTD entities are provided in the table "Template Definition DTD Entities" on page A-112.

```
<!ELEMENT Workflow (name, effective, changedby, changeddate, notes,
                    idexpression, active, audit-enabled, plugins?, nodes,
                    variables, error-handlers?)>
<!ATTLIST Workflow
    major-version   CDATA  "1"
    minor-version   CDATA  "0"
    build-number    CDATA  "0"
>

<!ENTITY % node "id, x, y, notes">

<!ENTITY % std-action-types "
    ActionAuditEntry |
    ActionBusinessOperation |
    ActionCallProgram |
    ActionCancelEvent |
    ActionCondition |
    ActionExpression |
    ActionNoOp |
    ActionPostXMLEvent |
    ActionPlugin |
    ActionSendEMail |
    ActionSendXMLToClient |
    ActionTaskAssignRole |
    ActionTaskAssignRoutingTable |
    ActionTaskAssignUser |
    ActionTaskDoit |
    ActionTaskDone |
    ActionTaskDueDate |
    ActionTaskSetComment |
    ActionTaskSetPriority |
    ActionTaskUnassign |
    ActionTaskUndo |
    ActionTimedEvent |
    ActionWorkflowAbort |
    ActionWorkflowDone |
    ActionWorkflowSetComment |
```

```
    ActionWorkflowStart |
    ActionXSLTransform
">

<!ENTITY % action-types "%std-action-types; | ActionInvokeErrorHandler |
                         ActionSetErrorHandler">

<!ENTITY % commit-action-types "%std-action-types; | ActionExitErrorHandler |
                               ActionSetErrorHandler">

<!ENTITY % rollback-action-types "ActionAuditEntry | ActionBusinessOperation |
                                  ActionCallProgram | ActionCondition |
                                  ActionExitErrorHandler |
                                  ActionExpression | ActionNoOp | ActionPlugin |
                                  ActionPostXMLEvent | ActionSendEMail |
                                  ActionSendXMLToClient">

<!-- Ambiguous definition of element 'actions'. -->
<!-- In context 'Task' it is:
     <!ELEMENT actions (created, activated, executed, markeddone)> -->
<!-- In context 'Decision', 'ActionCondition' it is:
     <!ELEMENT actions (false, true)> -->
<!-- In context 'Done', 'Event', 'Start', 'ActionTaskDueDate',
     'ActionTimedEvent', 'ActionWorkflowStart' it is:
     <!ELEMENT actions (%action-types;)*> -->

<!ELEMENT actions ((created, activated, executed, markeddone)
                 | (false, true)
                 | (%action-types;)*)>
<!ELEMENT ActionAuditEntry (notes, audittext)>
<!ELEMENT ActionBusinessOperation (notes, descriptorid, description,
                                   result, result-type, instance-variable,
                                   instance-variable-type, parms,
                                   iviewx?, iviewy?)>
<!ELEMENT ActionCallProgram (notes, program, arguments)>
<!ELEMENT ActionCancelEvent (notes, target)>
<!ELEMENT ActionCondition (notes, condition, actions)>

<!ELEMENT ActionExitErrorHandler (notes)>
<!ATTLIST ActionExitErrorHandler exit-type (rollback|stop|retry|continue)
                                           "rollback">

<!ELEMENT ActionExpression (notes, variable, (expression | xml)>

<!ELEMENT ActionInvokeErrorHandler (notes, xml?)>
<!ATTLIST ActionInvokeErrorHandler error-handler-name CDATA #REQUIRED>

<!ELEMENT ActionNoOp (notes, description)>
```

```
<!ELEMENT ActionPlugin (id, notes, plugin-data, actions*)>
<!-- description is displayed in action lists if the plugin is unavailable. -->
<!ATTLIST ActionPlugin description CDATA #REQUIRED>

<!ELEMENT ActionPostXMLEvent (notes, (topic | queue), transaction, addressees?,
                              message-properties?, delivery-mode,
                              priority, time-to-live, orderkey?
                             (variable | xml)),
                             iviewx?, iviewy?>
<!ELEMENT ActionSendEmail (notes, subject, message, to, cc, bcc)>
<!ELEMENT ActionSendXMLToClient (id, notes, xml, variables. iviewx?, iviewy?)>

<!ELEMENT ActionSetErrorHandler (notes)>
<!ATTLIST ActionSetErrorHandler error-handler-name CDATA #REQUIRED>

<!ELEMENT ActionTaskAssignRole (notes, target, role, expression)>
<!ELEMENT ActionTaskAssignRoutingTable (notes, target, routeconditions)>
<!ELEMENT ActionTaskAssignUser (notes, target, user, expression, role)>
<!ELEMENT ActionTaskDoIt (notes, target)>
<!ELEMENT ActionTaskDone (notes, target)>
<!ELEMENT ActionTaskDueDate (id, notes, target, expression, calendartype,
                             calendarname, actions)>
<!ELEMENT ActionTaskSetComment (notes, target, comment)>
<!ELEMENT ActionTaskSetPriority (notes, target, priority)>
<!ELEMENT ActionTaskUnassign (notes, target)>
<!ELEMENT ActionTaskUndo (notes, target)>
<!ELEMENT ActionTimedEvent (id, notes, expression, calendartype, executionunits,
                        scheduleunits, executiontime, scheduletime, recoverable?
                            stopwhentaskdone, usetimeexpression, actions)>
<!ELEMENT ActionWorkflowAbort (notes)>
<!ELEMENT ActionWorkflowDone (notes)>
<!ELEMENT ActionWorkflowSetComment (notes, comment)>
<!ELEMENT ActionWorkflowStart (id, notes, templateid, template-name, refvariable,
                               orgexpression, parameters, results, actions,
                               iviewx?, iviewy?)>
<!ELEMENT ActionXSLTransform (notes, input-expression, transform-document,
                              transform-source, output-variable,
                              xsl-parameters?)>
<!ELEMENT xsl-parameters (xsl-parameter+)>
<!ELEMENT xsl-parameter (name, expression)>
<!ELEMENT activated ((%action-types;)*)>
<!ELEMENT addressee (name, expression, type)>
<!ELEMENT addressees (instanceid)
<!ELEMENT bcc (addressee*)>
<!ELEMENT cc (addressee*)>
<!ELEMENT commit-actions ((%commit-action-types;)*)>
<!ELEMENT created ((%action-types;)*)>
<!ELEMENT Decision (%node;, condition, truenexts, falsenexts, actions)>
<!ELEMENT Done (%node;, actions, plugin-data?)>
```

```
<!ELEMENT Event (%node;, ((root, description, key, condition) | (plugin-data)),
                 actions, variables, ivewx?, iviewy?)>
<!ELEMENT error-handlers (error-handler*)>
<!ELEMENT error-handler (notes, variables, commit-actions, rollback-actions)>
<!-- No more than one error-handler can have initial set to true -->
<!ATTLIST error-handler
    name    CDATA        #REQUIRED
    initial (true|false) "false"
>

<!ELEMENT executed ((%action-types;)*)>
<!ELEMENT false ((%action-types;)*)>
<!ELEMENT falsenexts (next*)>
<!ELEMENT Join (%node;, and)>
<!ELEMENT markeddone ((%action-types;)*)>
<!ELEMENT message-properties (property+)>
<!ELEMENT nexts (next*)>
<!ELEMENT nodes (Decision | Done | Event | Join | Start | Task)*>
<!ELEMENT parameters (parm*)>

<!-- Ambiguous definition of element 'parm' -->
<!-- In context 'ActionBusinessOperation' it is <!ELEMENT parm (#PCDATA)> -->
<!-- In context 'ActionWorkflowStart' it is <!ELEMENT parm (name, value)> -->
<!-- N.B. Content model shows repeatable elements purely to conform
     with DTD syntax. -->
<!-- <parm> elements only appear in the two forms exemplified above. -->
<!ELEMENT parm ((#PCDATA | (name, value))*)>
<!ELEMENT parms (parm*)>

<!-- The 'plugins' element appears if the template definition uses
     plugin functionality or any plugin defines its own template
     definition property. -->
<!-- The Studio is responsible for updating this element when
     saving the definition. -->
<!ELEMENT plugins (plugin*)>
<!-- One 'plugin' element appears for each plugin referenced by a
     template definition and each plugin which defines its own template
    definition property. -->
<!ELEMENT plugin (plugin-data?)>
<!-- name is the plugin name, referenced means whether this template
     definition uses this plugin-specific functionality, version
     is the plugin version number. -->
<!ATTLIST plugin
    name CDATA #REQUIRED
    referenced (true|false) #REQUIRED
    major-version CDATA #REQUIRED
    minor-version CDATA #REQUIRED
    vendor CDATA #REQUIRED
    url CDATA ""
```

```
>
<!ELEMENT property (name, value)>
<!-- Ambiguous definition of element 'result'. -->
<!-- In context 'ActionBusinessOperation' it is <!ELEMENT result (#PCDATA)> -->
<!-- In context 'ActionWorkflowStart' it is <!ELEMENT result (name, value)> -->
<!ELEMENT result ((#PCDATA | (name, value))*)>
<!ELEMENT results (result*)>
<!ELEMENT rollback-actions ((%rollback-action-types;)*)>
<!ELEMENT routecondition (to, type, condition)>
<!ELEMENT routeconditions (routecondition*)>
<!ELEMENT Start (%node;, description, (manual
                                      | called
                                      | (timed, dateexpression,
                                         reschedamount, reschedunits,
                                         recoverable?, calendarname, startorg)
                                      | (event, root, key, condition, startorg)
                                      | (plugin-data, startorg)),
                 nexts, actions, variables,iviewx?, iviewy?)>
<!ELEMENT plugin-data ANY>
<!-- name is the plugin name,
     id is the plugin-provided unique ID for the xxxInfo object. -->
<!ATTLIST plugin-data
    name CDATA #REQUIRED
    ID CDATA #REQUIRED
>
<!ELEMENT Task (%node;, name, priority, donewithoutdoit, doitifdone,
                unmarkdone, modifiable, reassignment, nexts, actions)>
<!ELEMENT to (addressee*)>
<!ELEMENT true ((%action-types;)*)>
<!ELEMENT truenexts (next*)>

<!-- Ambiguous definition of element 'variable' -->
<!-- In context 'Workflow' it is
     <!ELEMENT variable (name, notes, (type|plugin-data), xmltype,
                         inmandatory, in, out)> -->
<!-- In context 'Start', 'ActionSendXMLToClient' it is
     <!ELEMENT variable (name, expression)> -->
<!-- In context 'ActionExpression' it is
     <!ELEMENT variable (#PCDATA)> -->
<!-- N.B. Content model shows repeatable elements purely to conform to
     DTD syntax. -->
<!-- <variable> elements only appear in the three forms exemplified above. -->
<!ELEMENT variable (((name, notes, (type|plugin-data), xmltype, inmandatory,
                      in, out)
                   |  (name, expression)
                   |  (#PCDATA))*)>
<!--ELEMENT variable (#PCDATA)-->
<!ELEMENT variables (variable*)>
```

```
<!ELEMENT active (#PCDATA)>
<!ELEMENT and (#PCDATA)>
<!ELEMENT arguments (#PCDATA)>
<!ELEMENT audit-enabled (#PCDATA)>
<!ELEMENT audittext (#PCDATA)>
<!ELEMENT calendarname (#PCDATA)>
<!ELEMENT calendartype (#PCDATA)>
<!ELEMENT called (#PCDATA)>
<!ELEMENT changedby (#PCDATA)>
<!ELEMENT changeddate (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT condition (#PCDATA)>
<!ELEMENT correlationid (#PCDATA)>
<!ELEMENT dateexpression (#PCDATA)>
<!ElEMENT delivery-mode (#PC-DATA)
<!ELEMENT description (#PCDATA)>
<!ELEMENT descriptorid (#PCDATA)>
<!ELEMENT doitifdone (#PCDATA)>
<!ELEMENT donewithoutdoit (#PCDATA)>
<!ELEMENT effective (#PCDATA)>
<!ELEMENT event (#PCDATA)>
<!ELEMENT executionunits (#PCDATA)>
<!ELEMENT executiontime (#PCDATA)>
<!ELEMENT expression (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT idexpression (#PCDATA)>
<!ELEMENT in (#PCDATA)>
<!ELEMENT inmandatory (#PCDATA)>
<!ELEMENT input-expression (#PCDATA)
<!ELEMENT instanceid (#PCDATA)>
<!ELEMENT instance-variable (#PCDATA)>
<!ELEMENT instance-variable-type (#PCDATA)>
<!ELEMENT iviewx (#PCDATA)
<!ELEMENT iviewy (#PCDATA)
<!ELEMENT jmstype (#PCDATA)
<!ELEMENT key (#PCDATA)>
<!ELEMENT manual (#PCDATA)>
<!ELEMENT message (#PCDATA)>
<!ELEMENT modifiable (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT next (#PCDATA)>
<!ELEMENT notes (#PCDATA)>
<!ELEMENT orderkey (#PCDATA)>
<!ELEMENT orgexpresion (#PCDATA)>
<!ELEMENT out (#PCDATA)>
<!ELEMENT output-variable (#PCDATA)
<!ELEMENT priority (#PCDATA)>
<!ELEMENT program (#PCDATA)>
<!ELEMENT queue (#PCDATA)>
```

```
<!ATTLIST queue expression (true | false) "false"
<!ELEMENT reassignment (#PCDATA)>
<!ELEMENT recoverable (#PCDATA)>
<!ELEMENT refvariable (#PCDATA)>
<!ELEMENT replyto (#PCDATA)
<!ELEMENT reschedamount (#PCDATA)>
<!ELEMENT reschedunits (#PCDATA)>
<!ELEMENT result-type (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT root (#PCDATA)>
<!ELEMENT scheduletime (#PCDATA)>
<!ELEMENT scheduleunits (#PCDATA)>
<!ELEMENT startorg (#PCDATA)>
<!ATTLIST startorg expression (true | false) "true"
<!ELEMENT stopwhentaskdone (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT target (#PCDATA)>
<!ELEMENT templateid (#PCDATA)>
<!ELEMENT template-name (#PCDATA)>
<!ELEMENT time-to-live (#PCDATA)>
<!ELEMENT timed (#PCDATA)>
<!ELEMENT topic (#PCDATA)>
<!ATTLIST topic expression (true | false) "false")
<!ELEMENT type (#PCDATA)>
<!-- The 'transaction' element has the value 'true' or 'false'. -->
<!ELEMENT transaction (#PCDATA)
<!ELEMENT transform-document (#PCDATA)
<!ELEMENT transform-source (#PCDATA)
<!ELEMENT unmarkdone (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT usetimeexpression (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT x (#PCDATA)>
<!ELEMENT xml ANY>
<!ELEMENT xmltype (#PCDATA)>
<!ELEMENT y (#PCDATA)>
```

Element Descriptions

The following table describes the elements of the Template Definition DTD.

Table A-18 Template Definition DTD Elements

Element	Description	Example Value

`ActionAuditEntry`	Action used to define audit entries that are published to the JMS `wlpiAudit` topic, in addition to the default audit entries that include major interactions and/or changes.
	In the Studio, you define this action by selecting Make Audit Entry within the Miscellaneous actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `audittext`
		`<ActionAuditEntry>`
		`  <notes></notes>`
		`  <audittext>`
		`  &quot;Workflow`
		`  cancelled.&quot;`
		`  </audittext>`
		`</ActionAuditEntry>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionBusinessOperation`	Action used to invoke predefined business operations on an EJB or Java class. The results of invoking the business operation can be assigned to workflow variables.
	In the Studio, you configure business operations by selecting Business Operations from the Configuration menu. You invoke them, when defining actions, by selecting Perform Business Operation within the Integration actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `descriptorid`
	■ `description`
	■ `result`
	■ `result-type`
	■ `instance-variable`
	■ `instance-variable-type`
	■ `parms`
	■ `iviewx` (zero or one)
	■ `iviewy` (zero or one)
		`<ActionBusinessOperation>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<descriptorid>`
		`1`
		`</descriptorid>`
		`<description>`
		`Check Inventory`
		`</description>`
		`<result>`
		`Inventory`
		`</result>`
		`<result-type>`
		`integer`
		`</result-type>`
		`<instance-variable>`
		`PobeanHandle`
		`</instance-variable>`
		`<instance-variable-type>`
		`session`
		`</instance-variable-type>`
		`<parms>`
		`<parm>`
		`ItemNumber`
		`</parm>`
		`</parms>`
		`<iviewx>150</iviewx>`
		`<iviewy>70</iviewy>`
		`</ActionBusinessOperation>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionCallProgram`	Action used to invoke predefined executable program. The results of invoking the program can be assigned to workflow variables.
	In the Studio, you define this action by selecting Call Program within the Integration actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `program`
	■ `arguments`
		`<ActionCallProgram>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<program>`
		`ProgramA`
		`</program>`
		`<arguments>`
		`CurrentUser()`
		`</arguments>`
		`</ActionCallProgram>`

`ActionCancelEvent`	Action used to cancel a workflow event that is defined within a workflow.
	In the Studio, you define this action by selecting Cancel Workflow Event within the Miscellaneous actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `target`
		`<ActionCancelEvent>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<target>`
		`987445252339`
		`</target>`
		`</ActionCancelEvent>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
ActionCondition	Action used to evaluate a conditional expression at run time and depending on the result, to perform alternative sequences of subactions. In the Studio, you define this action by selecting Evaluate Condition within the Miscellaneous actions category of the Add Actions dialog box. You must define the following subelements: ■ notes ■ condition ■ actions	<ActionCondition> <notes> This is a note. </notes> <condition> CurrentUser()= "joe" </condition> <actions> <ActionTaskUnassign> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskUnassign> </false> <true> <ActionTaskDone> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskDone> </actions> </ActionCondition>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
`ActionExitErrorHandler`	Action used to exit an exception handler. In the Studio, you define this action using the Exit Exception Handler dialog box. You can access the dialog box by clicking the Add button in the Exception Handler Properties dialog box when the Actions on Commit or Actions on Rollback tab is selected. You must define the following subelement: `notes` You must specify one of the following values for the `exit-type` attribute to specify a method to be used by the existing error handler: ■ `rollback` - Marks the user transaction as rollback only, returning the workflow to the state in which it existed prior to the start of the transaction. The exception is propagated to the client (if any) as a WorkflowException. In the rollback path, this is the only option available. Certain EJBs can either mark a transaction for rollback only, by calling a UserTransaction method, or they can throw an unchecked exception across a container boundary. In either case, WebLogic Server rolls back the transaction. ■ `stop` - Exits the error handler and does not execute any follow-on actions. ■ `retry` - Exits the error handler and attempts to retry the failed operation. (The failed operation is an action or the setting of a variable.) ■ `continue` - Stops the execution of the exception handler and attempts to continue the execution of the workflow at the next operation. (The next operation is the next action to be performed or variable to be set.)	`<ActionExitErrorHandler exit-type="rollback"> <notes> This is a note. </notes> </ActinoExitErrorHandler>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionExpression`	Action used to set a workflow variable.
	In the Studio, you define this action by selecting Set Workflow Variable within the Workflow actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `variable`
	■ `(expression | xml)`
		`<ActionExpression>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<variable>`
		`CustomerState`
		`</variable>`
		`<expression>`
		`&quot;NJ&quot;`
		`</expression>`
		`</ActionExpression>`

`ActionInvokeErrorHandler`	Action used to enable the invocation of the specified exception handler. Upon execution of this action, WebLogic Process Integrator sends the user-defined XML document to the exception processor and invokes the specified handler.
	In the Studio, you define this action by selecting Invoke Exception Handler within the Exception Handling Actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `xml` (zero or one)
	You must specify a text string specifying the name of a valid exception handler for the `error-handler-name` attribute. This attribute is required.
		`<ActionInvokeErrorHandler`
		`error-handler-name=`
		`"EH Wrong data`
		`to Pobean">`
		`<notes>`
		`This is a note.`
		`</notes>`
		`</ActionInvokeErrorHandler>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
ActionNoOp	Action used as a placeholder for a future action. This action performs no function. In the Studio, you define this action by selecting No Operation within the Miscellaneous actions category of the Add Actions dialog box. You must define the following subelements: ■ notes ■ description	<ActionNoOp> <notes> This is a note. </notes> <description> Replace with task assignment. </description> </ActionNoOp>
ActionPlugin	Generic container for all plug-in actions. You must define the following subelements: ■ id ■ plugin-data ■ notes ■ actions (zero or more) You must specify a text string providing a description of the plug-in the name for the label attribute. This attribute is required. **Note:** If a plug-in is available, the Studio displays the description of it in the actions lists. For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.	<ActionPlugin label='Send Confirm Order Event'> <id>1000936287350 </id> <plugin-data name="com.bea.wlpi.SamplePlugin" ID="2"> <statusvariable> OrderStatus </statusvariable> <totalpricevariable> OrderTotalPrice </totalpricevariable> </plugin-data> <notes></notes> </ActionPlugin>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionPostXMLEvent`	Action used to post an XML event.
	In the Studio, you define this action by selecting Post XML Event within the Integration actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `topic` | `queue`
	■ `transaction`
	■ `addressees` (zero or one)
	■ `message-properties` (zero or one)
	■ `delivery-mode`
	■ `priority`
	■ `time-to-live`
	■ `orderkey` (zero or one)
	■ `variable` | `xml`
	■ `iviewx` (zero or one)
	■ `iviewy` (zero or one)
		`<ActionPostXMLEvent>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<transaction>`
		`false`
		`</transaction>`
		`<delivery-mode>`
		`PERSISTENT`
		`</delivery-mode>`
		`<priority>4`
		`</priority>`
		`<expression>false`
		`</expression>`
		`<time-to-live>0`
		`</time-to-live>`
		`<xml>`
		`<order>`
		`&quot;new&quot;`
		`</order>`
		`</xml>`
		`<iviewx>240</iviewx>`
		`<iviewy>20</iviewy>`
		`</ActionPostXMLEvent>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`actions`	Action definition.
	For a `Task` node, you must define the following subelements:
	■ `created`
	■ `activated`
	■ `executed`
	■ `markeddone`
	For a `Decision` node or an `ActionCondition` action, you must define the following subelements:
	■ `false`
	■ `true`
	For a `Done`, `Event`, or `Start` node; or a `ActionTaskDueDate`, `ActionTimedEvent`, or `ActionWorkflowStart` event, you can define zero or more occurrences of the following subelement: `%action-types`
		`<actions>`
		`<created>`
		`</created>`
		`<activated>`
		`<ActionTaskAssignUser>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<target>`
		`2`
		`</target>`
		`<user>joe</user>`
		`<expression>`
		`false`
		`</expression>`
		`<role>`
		`false`
		`</role>`
		`</ActionTaskAssignUser>`
		`</activated>`
		`<executed>`
		`</executed>`
		`<markeddone>`
		`</markeddone>`
		`</actions>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
ActionSendEMail	Action used to send an e-mail message to a WebLogic Process Integrator user or other individual. The internet standard SMTP protocol is used to transmit the message. In the Studio, you define this action by selecting Send E-mail Message within the Miscellaneous actions category of the Add Actions dialog box. You must define the following subelements: ■ notes ■ subject ■ message ■ to ■ cc ■ bcc	<ActionSendEMail> <notes> This is a note. </notes> <subject> "Order " </subject> <message> "This is the message. " </message> <to> <addressee> <name> joe@work </name> <expression> false </expression> <type> address </type> </addressee> </to> <cc> </cc> <bcc> </bcc> </ActionSendEMail>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
ActionSendXMLToClient	Action used to send an XML document to the client, This action allows communication between the WebLogic Process Integrator and a client program via an XML message. In the Studio, you define this action by selecting Send XML to Client within the Integration actions category of the Add Actions dialog box. The XML document must be compliant with one of the following DTD formats, based on the request that is being sent: ■ "Client Call Addin Request DTD" on page A-19 ■ "Client Call Program Request DTD" on page A-24 ■ "Client Message Box Request DTD" on page A-29 ■ "Client Set Variables Request DTD" on page A-37 Custom clients can define their own services and XML formats. You must define the following subelements: ■ id ■ notes ■ xml ■ variables ■ iviewx (zero or one) ■ iviewy (zero or one)	<ActionSendXMLToClient> <id>959395846210</id> <notes>This is a note. </notes> <xml> <message-box title=""Order Confirmation"" style=""question "" options=""yes_no""> "Do you Confirm this order?" <actionid> "959395846210 " </actionid> </message-box> </xml> <variables> <variable> <name> Confirm</name> <expression> XPath(" /message-box/ attribute:: option") </expression> </variable> </variables> <actions> <ActionTaskDone> <notes> This is a note. </notes> <target>2</target> </ActionTaskDone> </actions> </ActionSendXMLToClient>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionSetErrorHandler`	Action used to set a specified exception handler as active for a specific workflow template definition. This exception handler persists for the workflow instance until a subsequent `ActionSetErrorHandler` action is executed within the same workflow. By default, the system exception handler is used.
	In the Studio, you define this action by selecting Set Workflow Exception Handler within the Exception Handling Actions category of the Add Actions dialog box.
	You must define the following subelements: `notes`.
	You must specify a text string specifying the name of a valid exception handler (as a text string) for the `error-handler-name` attribute. This attribute is required.
		`<ActionSetErrorHandler error-handler-name= "EH Wrong data to Pobean">`
		`  <notes>`
		`  This is a note.`
		`  </notes>`
		`</ActionSetErrorHandler>`

`ActionTaskAssignRole`	Action used to assign a task to a role.
	In the Studio, you define this action by selecting Assign Task to Role within the Task Actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `target`
	■ `role`
	■ `expression`
		`<ActionTaskAssignRole>`
		`  <notes>`
		`  This is a note.`
		`  </notes>`
		`  <target>`
		`  963511923442`
		`  </target>`
		`  <role>`
		`  Role1`
		`  </role>`
		`  <expression>`
		`  false`
		`  </expression>`
		`</ActionTaskAssignRole>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionTaskAssignRoutingTable`	Action used to assign a task to a user or role, depending on a set of conditions. A condition specifies the circumstances under which a task should be assigned to a specific assignee (user or role). At runtime, the system selects the assignee associated with the condition that evaluates to `true`.
	In the Studio, you define this action by selecting Assign Task to Role within the Task Actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `target`
	■ `routeconditions`
		`<ActionTaskAssignRoutingTable>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<target>2</target>`
		`<routeconditions>`
		`<routecondition>`
		`<to>Role1</to>`
		`<type>`
		`userinrole`
		`</type>`
		`<condition>`
		`a+b`
		`</condition>`
		`</routecondition>`
		`</routeconditions>`
		`</ActionTaskAssignRoutingTable>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
ActionTaskAssignUser	Action used to assign a task to either a user or user in a role. When assigning a task to a user in a role, the process engine performs load balancing by reviewing the number of tasks assigned to all users in a role (that do not currently have reroutes in effect), indentifying the user with the smallest number of assigned tasks, and assigning the task to this user. In the Studio, you define this action by selecting Assign Task to User within the Task Actions category of the Add Actions dialog box. You must define the following subelements: ■ notes ■ target ■ user ■ expression ■ role	<ActionTaskAssignUser> <notes> This is a note. </notes> <target> 963511775400 </target> <user>joe</user> <expression> false </expression> <role> false </role> </ActionTaskAssignUser>
ActionTaskDoit	Action used to execute a task. In the Studio, you define this action by selecting Execute Task within the Task Actions category of the Add Actions dialog box. You must define the following subelements: ■ notes ■ target	<ActionTaskDoIt> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskDoIt>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionTaskDone`	Action used to mark task complete. This action sets the task completed value to the current date and time, and results in the sequential execution of all of the actions associated with the specified task's `Marked Done` event.
	In the Studio, you define this action by selecting Mark Task As Done within the Task Actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `target`
		`<ActionTaskDone>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<target>`
		`963511923442`
		`</target>`
		`</ActionTaskDone>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
ActionTaskDueDate	Action used to specify a task due date as an interval (expressed in minutes, hours, days, weeks, or months) from the point at which the action is executed. This action results in the execution of all actions associated with the Marked Done event. You can also define a set of actions to be performed if a task becomes overdue. In the Studio, you define this action by selecting Set Task Due Date within the Task Actions category of the Add Actions dialog box. You must define the following subelements: ■ id ■ notes ■ target ■ expression ■ calendartype ■ calendarname ■ actions	<ActionTaskDueDate> <id>987445484073</id> <notes> This is a note. </notes> <target>2</target> <expression> DateAdd(Date(), "m",1) </expression> <calendartype> 0 </calendartype> <calendarname> </calendarname> <actions> <ActionNoOp> <notes> This is a note. </notes> <description> Replace with task assignment. </description> </ActionNoOp> </actions> </ActionTaskDueDate>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
ActionTaskSetComment	Action used to specify a comment to be displayed to the user at runtime when the action is executed. Comments typically contain explanatory text. In the Worklist, the comment is displayed as a free-form text message in the comment column. In the Studio, you define this action by selecting Set Task Comment within the Task Actions category of the Add Actions dialog box. You must define the following subelements: ■ notes ■ target ■ comment	<ActionTaskSetComment> <notes> This is a note. </notes> <target> 963511923442 </target> <comment> This is a comment. </comment> </ActionTaskSetComment>
ActionTaskSetPriority	Action used to set a task priority. In the Worklist, the comment is displayed as a free-form text message in the comment column. In the Studio, you define this action by selecting Set Task Priority within the Task Actions category of the Add Actions dialog box. You must define the following subelements: ■ notes ■ target ■ priority	<ActionTaskSetPriority> <notes> This is a note. </notes> <target> 963511923442 </target> <priority> 1 </priority> </ActionTaskSetPriority> [Sets priority to medium (1).]

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionTaskUnassign`	Action used to unassign a task.
	In the Studio, you define this action by selecting Set Task Priority within the Task Actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `target`
		`<ActionTaskUnassign>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<target>`
		`963511923442`
		`</target>`
		`</ActionTaskUnassign>`

`ActionTaskUndo`	Action used to mark task incomplete. The action *does not* result in the execution of the actions associated with the specified task's `Activated` event.
	In the Studio, you define this action by selecting Mark Task As Undone within the Task Actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `target`
		`<ActionTaskUndo>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<target>`
		`963511923442`
		`</target>`
		`</ActionTaskUndo>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionTimedEvent`	Action used to create a timed event that is triggered at an exact time and date, and will execute any specified actions.
	In the Studio, you define this action by selecting Timed Event within the Miscellaneous Actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `id`
	■ `notes`
	■ `expression`
	■ `calendartype`
	■ `executionunits`
	■ `scheduleunits`
	■ `executiontime`
	■ `scheduletime`
	■ `recoverable` (zero or one)
	■ `stopwhentaskdone`
	■ `usetimeexpression`
	■ `actions`
		`<ActionTimedEvent>`
		`<id>987100768489</id>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<expression>`
		`DateAdd(Date(),`
		`&quot;m&quot;, 1)`
		`</expression>`
		`<calendartype>0`
		`</calendartype>`
		`<executionunits>3`
		`</executionunits>`
		`<scheduleunits>0`
		`</scheduleunits>`
		`<executiontime>5`
		`</executiontime>`
		`<scheduletime>`
		`</scheduletime>`
		`<recoverable>`
		`true`
		`</recoverable>`
		`<stopwhentaskdone>`
		`true`
		`</stopwhentaskdone>`
		`<usetimeexpression>`
		`true`
		`</usetimeexpression>`
		`<actions>`
		`<ActionTaskAssignUser>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<target>2`
		`</target>`
		`<user>joe</user>`
		`<expression>`
		`false`
		`</expression>`
		`<role>false`
		`</role>`
		`</ActionTaskAssignUser`
		`</actions>`
		`</ActionTimedEvent>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
active	Flag (true or false) specifying whether or not the workflow template definition is active.	<active> true </active>
activated	Actions associated with the task's Activated event. You can define zero or more occurrences of the following subelement: %action-types.	<activated> <ActionTaskAssignUser> <notes> This is a note. </notes> <target> 2 </target> <user>joe</user> <expression> false </expression> <role> false </role> </ActionTaskAssignUser> </activated>
ActionWorkflowAbort	Action used to abort a workflow. This action permanently stops a workflow and deletes it from the server. In the Studio, you define this action by selecting Abort Workflow within the Workflow Actions category of the Add Actions dialog box. You must define the following subelement: notes.	<ActionWorkflowAbort> <notes> This is a note. </notes> </ActionWorkflowAbort>
ActionWorkflowDone	Action used to mark a workflow as complete. In the Studio, you define this action by selecting Mark Workflow as Done within the Workflow Actions category of the Add Actions dialog box. You must define the following subelement: notes.	<ActionWorkflowDone> <notes> This is a note. </notes> </ActionWorkflowDone>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`ActionWorkflowSetComment`	Action used to specify a comment to be displayed to the user at runtime when the action is executed. Comments typically contain explanatory text.
	In the Studio, you define this action by selecting Mark Workflow as Done within the Workflow Actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `notes`
	■ `comment`
		`<ActionWorkflowSetComment>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<comment>`
		`&quot;Order Cancelled&quot;`
		`</comment>`
		`</ActionWorkflowSetComment>`

`ActionWorkflowStart`	Action used to start a workflow, supporting subworkflows.
	In the Studio, you define this action by selecting Start Workflow within the Workflow Actions category of the Add Actions dialog box.
	You must define the following subelements:
	■ `id`
	■ `notes`
	■ `templateid`
	■ `template-name`
	■ `refvariable`
	■ `orgexpression`
	■ `parameters`
	■ `results`
	■ `actions`
	■ `iviewx` (zero or one)
	■ `iviewy` (zero or one)
		`<ActionWorkflowStart>`
		`<id>`
		`987445540073`
		`</id>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<templateid>`
		`3`
		`</templateid>`
		`<template-name>`
		`ShipBill`
		`</template-name>`
		`<refvariable>`
		`Name`
		`</refvariable>`
		`<parameters>`
		`</parameters>`
		`<results>`
		`</results>`
		`<actions>`
		`</actions>`
		`</ActionWorkflowStart>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
ActionXSLTransform	Action used to perform an XML transformation. You must define the following subelements: ■ notes ■ input-expression ■ transform-document ■ transform-source ■ output-variable ■ xsl-parameters (one or more)	
addressee	Address of a receipient. You must define the following subelements: ■ name ■ expression ■ type	<addressee> <name> joe@work </name> <expression> false </expression> <type> address </type> </addressee>
addressees	One or more addressees. You must define the following subelement: ■ instanceid	
and	Flag specifying whether or not the Join node is an AND (true) or OR (false) node.	<and>true</and>
arguments	Valid workflow expression to be evaluated at runtime and passed to the called program defined by the program element.	<arguments> CurrentUser() </arguments>
audit-enabled	Flag (true or false) specifying whether or not auditing is enabled on the workflow template definition.	<audit-enabled> false </audit-enabled>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`audittext`	Audit entry expression to be sent to the audit listener for further retrieval at runtime. The audit entry can be an expression built with functions and workflow variables, or a string of text.	`<audittext>`
		`&quot;Workflow`
		`cancelled.&quot;`
		`</audittext>`

`bcc`	E-mail addresses of the individuals to be blind-copied on the e-mail message, as defined by the `ActionSendEMail` action.
	You can define zero or more occurrences of the following subelement: `addressee`.
		`<bcc>`
		`<addressee>`
		`<name>`
		`joe@work`
		`</name>`
		`<expression>`
		`false`
		`</expression>`
		`<type>`
		`address`
		`</type>`
		`</addressee>`
		`</bcc>`

`calendartype`	Business calendar type.	`<calendartype>`
		`Type1`
		`</calendartype>`

`calendarname`	Business calendar name.	`<calendarname>`
		`MyCalendar`
		`</calendarname>`

`called`	Flag (`true` or `false`) specifying whether or not the `Start` node is callable from another workflow.	`<called>`
		`true`
		`</called>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
cc	E-mail address(es) of the individuals to be copied on an e-mail message, as defined by the ActionSendEMail action. You can define zero or more occurrences of the following subelement: addressee.	<cc> <addressee> <name> joe@work </name> <expression> false </expression> <type> address </type> </addressee> </cc>
changedby	Valid user that last modified the workflow template definition.	<changedby> joe </changedby>
changeddate	Date at which the workflow template was changed, specified using the following format: *yyyyMMddHHmm*, where *yyyy* specifies the year, *MM* specifies the month, *dd* specifies the day, HH specifies the hour (00 through 23), and mm specifies the minutes.	<changeddate> 200103220930 </changeddate> [March 22, 2001 9:30AM]
comment	A valid workflow expression or text string that defines a comment and is evaluted at runtime. Comments typically contain explanatory text.	<comment> "Order Cancelled" </comment>
commit-actions	Actions performed when a transaction is committed. You can define zero or more occurrences of the following subelement: %commit-action-types.	<commit-actions> <ActionTaskDone> <notes> This is a note. </notes> <target> 963511775400 </target> </ActionTaskDone> </commit-actions>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`condition`	Valid workflow condition that is evaluated at runtime and determines which set of subactions to perform.	`<condition>`
		`status=&quot;`
		`complete&quot;`
		`</condition>`

`correlationid`	Correlation ID.	

`created`	Actions associated with the task's `Created` event.
	You can define zero or more occurrences of the following subelement: `%action-types`.
		`<created>`
		`<ActionTaskAssignUser>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<target>`
		`2`
		`</target>`
		`<user>joe</user>`
		`<expression>`
		`false`
		`</expression>`
		`<role>`
		`false`
		`</role>`
		`</ActionTaskAssignUser>`
		`</created>`

`dateexpression`	Date to start a time-triggered workflow. This value can be either a workflow expression or a text string, and should result in the following format: *`yyyyMMddHHmm`*, where *`yyyy`* specifies the year, *`MM`* specifies the month, *`dd`* specifies the day, `HH` specifies the hour (00 through 23), and `mm` specifies the minutes.	`<dateexpression>`
		`200007120000`
		`</dateexpression>`
		[July 12, 2000 12:00 am]

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
`Decision`	`Decision` node. You must define the following subelements: ■ `%node` ■ `condition` ■ `truenexts` ■ `falsenexts` ■ `actions`	See "Template Definition DTD Example" on page A-115.

`delivery-mode`	Mode used to deliver a posted XML event.	`<delivery-mode>`
		`PERSISTENT`
		`</delivery-mode>`

`description`	Text description of the element.	`<description>`
		`Start`
		`</description>`

`descriptorid`	Descriptor ID of the business operation that is being defined for the `ActionBusinessOperation` action.	`<descriptorid>`
		`31`
		`</descriptorid>`

`doitifdone`	Permission flag (`true` or `false`) specifying whether or not the task can be executed once it has been marked done, using the methods described in "Executing a Task" on page 21-6.	`<doitifdone>`
		`true`
		`</doitifdone>`

`Done`	`Done` node. For manually started workflows, you must define the following subelements: ■ `%node` ■ `description` ■ `actions` ■ `plugin-data` (zero or one)	See "Template Definition DTD Example" on page A-115.

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`donewithoutdoit`	Permission flag (`true` or `false`) specifying whether or not the task can be marked done without having been executed, using the methods described in "Marking a Task as Complete or Incomplete" on page 21-18.	`<donewithoutdoit>`
		`true`
		`</donewithoutdoit>`

`effective`	Effective date in the following format: *`yyyyMMddHHmm`*, where *`yyyy`* specifies the year, *`MM`* specifies the month, *`dd`* specifies the day, `HH` specifies the hour (00 through 23), and `mm` specifies the minutes.	`<effective>`
		`200007120000`
		`</effective>`
		[July 12, 2000 12:00 am]

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`error-handler`	Error handler.
	You must define the following subelements:
	■ `notes`
	■ `variables`
	■ `commit-actions`
	■ `rollback-actions`
	You define the following attributes:
	■ `name`: Error-handler name.
	■ `initial`: Flag (`true` or `false`) specifying whether or not this is the default error handler to use. This attribute can be set to `true` for only one error handler. This attribute defaults to `false`.
		`<error-handler name="EH Wrong data to Pobean" initial="false">`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<variables>`
		`</variables>`
		`<commit-actions>`
		`<ActionExitErrorHandler exit-type="continue">`
		`<notes>`
		`This is a note.`
		`</notes>`
		`</ActionExitErrorHandler>`
		`</commit-actions>`
		`<rollback-actions>`
		`<ActionPostXMLEvent>`
		`<notes>Notes.`
		`</notes>`
		`<xml>`
		`<order>`
		`&quot;new&quot;`
		`</order>`
		`</xml>`
		`</ActionPostXMLEvent>`
		`<ActionExitErrorHandler exit-type="rollback">`
		`<notes>Notes.`
		`</notes>`
		`</ActionExitErrorHandler>`
		`</rollback-actions>`
		`</error-handler>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`error-handlers`	Error handlers defined for the workflow template definition.
	You must define the following subelement: `error-handler`.
		`<error-handlers>`
		`  <error-handler> ...`
		`  </error-handler>`
		`</error-handlers>`

`Event`	`Event` node. For a *standard* `Event` node, you must define the following subelements: ■ `%node` ■ `root` ■ `description` ■ `key` ■ `condition` ■ `actions` ■ `variables` ■ `iviewx` (zero or one) ■ `iviewy` (zero or one) For a *plugin* `Event` node, you must define the following subelements: ■ `%node` ■ `plugin-data` ■ `actions` ■ `variables` ■ `iviewx` (zero or one) ■ `iviewy` (zero or one)	See "Template Definition DTD Example" on page A-115.

`event`	Flag (`true` or `false`) specifying whether or not the `Start` node is event-triggered.	`<event>`
		`  true`
		`</event>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
executed	Actions associated with the task's Executed event. You can define zero or more occurrences of the following subelement: %action-types.	<executed> <ActionTaskDone> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskDone> </executed>
executiontime	Numeric value specifying the number of executionunits units from the current time, that define the amount of time that must elapse before a time-triggered event (defined by the ActionTimedEvent action) is executed.	<executiontime> 5 </executiontime>
executionunits	Units used to express the execution time. Possible values include: ■ 0: Milliseconds ■ 1: Seconds ■ 2: Minutes ■ 3: Hours ■ 4: Days ■ 5: Weeks ■ 6: Months ■ 7: Business hours ■ 8: Business days	<executionunits> 3 </executionunits>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
expression	For ActionExpression, ActionTaskDueDate, ActionTimedEvent, or variable, specifies a workflow expression that is meaningful within the context of the parent element. For ActionTaskAssignRole, ActionTaskAssignUser, or addressee, specifies a flag (true or false) that defines whether or not an associated value is represented as a workflow expression.	<expression> XPath(" /order/text()") </expression>
false	Actions associated with a Decision node's false branch. You can define zero or more occurrences of the following subelement: %action-types.	<false> <ActionWorkflowAbort> <notes> This is a note. </notes> </ActionWorkflowAbort> </false>
falsenexts	String specifying the next node associated with a Decision node's false branch. You can define zero or more occurrences of the following subelement: next.	<falsenexts> 963410128518 </falsenexts>
id	Unique ID that is meaningful within the context of the parent element.	<id>3</id>
idexpression	Expression used for identifying the workflow template definition.	<idexpression> "Order" +"" +:OrderNumber </idexpression>
in	Flag (true or false) specifying whether or not a variable, defined by a callable Start node, is an input parameter.	<in>true</in>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`inmandatory`	Flag (`true` or `false`) specifying whether or not an input variable, defined by a callable `Start` node, is a mandatory input parameter.	`<inmandatory>`
		`true`
		`</inmandatory>`

`input-expression`	Workflow expression that identifies the XML document for which an XSL transformation will take place.	

`instanceid`	Workflow instance ID that defines the target of the XML event defined by the `ActionPostXMLEvent` action.	`<instanceid>`
		`10`
		`</instanceid>`

`instance-variable`	Instance variable (name of the variable containing the Java object or EJB) on which WebLogic Process Integrator should invoke the business operation defined by the `ActionBusinessOperation` action.	`<instance-variable>`
		`PobeanHandle`
		`</instance-variable>`

`instance-variable-type`	Instance variable type of the instance variable defined by the `instance-variable` element.	`<instance-variable-type>`
		`session`
		`</instance-variable-type>`

`iviewx`	X-coordinate of a Studio interface view shape.	`<iviewx>150</iviewx>`
`iviewy`	Y-coordinate of a Studio interface view shape.	`<iviewx>150</iviewx>`
`jmstype`	JMS destination type.	
`Join`	`Join` node. You must define the following subelements: ■ `%node` ■ `and`	See "Template Definition DTD Example" on page A-115.

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
key	Workflow expression that evaluates to a key at runtime. The system evaluates the workflow expression against the root element of an incoming XML document (of the specified type) to yield a key value that uniquely identifies that document instance.	<key> $orderid </key>
manual	Flag (true or false) specifying whether or not the Start node is manually startable.	<manual> true </manual>
markeddone	Actions associated with the task's Marked Done event. You can define zero or more occurrences of the following subelement: %action-types.	<markeddone> <ActionWorkflowSetComment> <notes> This is a note. </notes> <comment> "Order Cancelled" </comment> </ActionWorkflowSetComment>
message	E-mail message text to be sent as defined by the ActionSendEMail action. This value can be a valid workflow expression or literal string (in double quotes) that is evaluated at runtime.	<message> The order has been processed. </message>
message-properties	Message properties. You can define one or more occurrences of the following subelement: property.	
modifiable	Permission flag (true or false) specifying whether or not the task properties can be modified at run time, using the methods described in "Setting Task Properties" on page 21-22.	<modifiable> true </modifiable>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
name	Unique string specifying the element name that is meaningful within the context of the parent element.	<name> Order Processing </name>
next	String specifying the next node to activate.	<next> 963410128518 </next>
nexts	A list containing zero or more next actions. You can define zero or more occurrences of the following subelement: next.	<nexts> <next> 963410128518 </next> </nexts>
nodes	Nodes defined for the workflow template definition. You can define zero or more occurrences of the following subelements: ■ Decision ■ Done ■ Event ■ Join ■ Start ■ Task	See "Template Definition DTD Example" on page A-115.
notes	User-defined notes.	<notes> This workflow is used for order processing. </notes>
orderkey	Order key, used for ordered messaging, as described in "Establishing JMS Connections" on page 6-1.	
orgexpression	Workflow expression that specifies the organization associated with the workflow that you are attempting to start, using the ActionWorkflowStart action.	

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
out	Flag (true or false) specifying whether or not a variable, defined by a callable Start node, is an output parameter.	<out>false</out>
output-variable	Variable to which the result of the XSL transformation is stored.	
parameters	Initial parameter values to be set using an explicitly defined variable in the calling workflow. You can define zero or more occurrences of the following subelement: parm.	<parameters> <parm> <name> CustomerId </name> <value> $CustomerId </value> </parm> </parameters>
parm	Parameter definition. For an ActionBusinessOperation action, this element can be any character string. For ActionWorkflowStart action, you can define zero or more of the following subelement pairs: ■ name ■ value	<parm> :ItemNumber </parm>
parms	Parameter list, if required. You must specify zero or more occurrences of the following subelement: parm.	<parms> <parm> :ItemNumber </parm> <parm> :ItemQuantity </parm> <parm> :CustomerState </parm> </parms>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
`plugin`	Plug-in entry. One element appears for each plug-in referenced by a template definition, and each plug-in that defines its own template definition property. Optionally, you can define the following subelement: `plugin-data`. You define the following attributes: ■ `name`: Plug-in name. ■ `referenced`: Flag (`true` or `false`) specifying whether or not the template definition uses the plug-in. ■ `major-version`: Major version of plug-in software. ■ `minor-version`: Minor version of plug-in software. ■ `vendor`: Vendor name. ■ `url`: URL (defaults to an empty string). For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.	`<plugin name="com.bea.wlpi.SamplePlugin" referenced="true" major-version="1" minor-version="0"> </plugin>`
`plugin-data`	Plug-in data consisting of any combination of elements and text. You must define the following attributes: ■ `name`: Name of the plug-in. This attribute is required. ■ `ID`: Plug-in provided unique ID for the associated value object. This attribute is required. For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.	`<plugin-data name="com.bea.wlpi.SamplePlugin" ID="2"> <statusvariable> OrderStatus </statusvariable> <totalpricevariable> OrderTotalPrice </totalpricevariable> </plugin-data>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
plugins	Plug-in definition, if plug-in is provided. You can define zero or more occurrences of the following subelements: plugin. For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.	<plugins> <plugin name="com.bea.wlpi.SamplePlugin" referenced="true" major-version="1" minor-version="0"> </plugin> </plugins>
priority	Task priority. Possible values include: 0 (low), 1 (medium), and 2 (high).	<priority> 1 </priority>
program	Fully-qualified path to an executable program.	<program> MyProgram </program>
property	Message property. You must define the following subelements: ■ name ■ value	
queue	JMS queue to which an XML document is sent or retrieved. You must define the expression attribute to specify whether or not the value specified is an expression. This attribute defaults to false.	<queue> Queue1 </queue>
reassignment	Permission flag (true or false) specifying whether or not a task can be reassigned to another participant, using the methods described in "Assigning a Task" on page 21-12.	<reassignment> true </reassignment>
recoverable	Flag specifying whether or not you would like to recover all of the actions lost in the event of system failure.	

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
refvariable	Variable to which the workflow ID is saved when the workflow is started by a ActionWorkflowStart action. This variable can be used in expressions from within the calling workflow to retrieve variables. For example, NewWF.Name retrieves the value of the Name variable.	<refvariable> Name </refvariable>
replyto	Reply-to address	
reschedamount	Numeric value. This value specifies the number of reschedunits units from the dateexpression at which a time-triggered Start node is rescheduled.	<reschedamount> 30 </reschedamount>
reschedunits	Units used to express the reschedule time for a time-triggered workflow. Possible values include: ■ 0: Milliseconds ■ 1: Seconds ■ 2: Minutes ■ 3: Hours ■ 4: Days ■ 5: Weeks ■ 6: Months ■ 7: Business hours ■ 8: Business days	<reschedunits> 0 </reschedunits>
result	Result. For an ActionBusinessOperation action, this element can be any character string. For an ActionWorkflowStart action, you must define the following subelements: ■ name ■ value	<result> PobeanHandle </result>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`result-type`	Type of `result-type` variable.	`<result-type>`
		`  Session`
		`</result-type>`

`results`	Results.
	You can define zero or more occurrences of the following subelement: `result`.
		`<results>`
		`  <result>`
		`  PobeanHandle`
		`  </result>`
		`</results>`

`role`	For `ActionTaskAssignRole`, specifies an existing role or a workflow expression that evaluates to an existing role. If a workflow expression, the `expression` flag must be set to `true`.
	For `ActionTaskAssignUser`, specifies a flag (`true` or `false`) that indicates whether the `user` element specifies a user (`false`) or role (`true`).
		`<role>`
		`  &quot;Role1&quot;`
		`</role>`

`rollback-actions`	Actions performed when a transaction is rolled back.
	You can define zero or more occurrences of the following subelement: `%rollback-action-types`
		`<rollback-actions>`
		`  <ActionTaskDone>`
		`  <notes>`
		`  This is a note.`
		`  </notes>`
		`  <target>`
		`  963511775400`
		`  </target>`
		`  </ActionTaskDone>`
		`</rollback-actions>`

`root`	Root element of an XML document that triggers the associated event.	`<root>order</root>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
routecondition	Route condition. You must define the following subelements: ■ to ■ type ■ condition	<routecondition> <to>Role1</to> <type> userinrole </type> <condition> a+b </condition> </routecondition>
routeconditions	Route conditions. You can define zero or more occurrences of the following subelement: routecondition.	<routeconditions> <routecondition> <to>Role1</to> <type> userinrole </type> <condition> a+b </condition> </routecondition> </routeconditions>
scheduletime	Numeric value. This value specifies the number of scheduleunits units from the current time, that define the amount of time that must elapse before a time-triggered event (defined by the ActionTimedEvent action) is rescheduled.	<scheduletime> 5 </scheduletime>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`scheduleunits`	Units used to express the reschedule time. Possible values include:
	■ `0`: Milliseconds
	■ `1`: Seconds
	■ `2`: Minutes
	■ `3`: Hours
	■ `4`: Days
	■ `5`: Weeks
	■ `6`: Months
	■ `7`: Business hours
	■ `8`: Business days
		`<scheduleunits>`
		`0`
		`</scheduleunits>`

`subject`	Subject line for the e-mail message to be sent, as defined by the `ActionSendEMail` action. This value can be a valid workflow expression or literal string (in double quotes) that is evaluated at runtime.	`<subject>`
		`Urgent`
		`</subject>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
`Start`	`Start` node. For *manually startable* workflows, you must define the following subelements: ■ `%node` ■ `description` ■ `manual` ■ `nexts` ■ `actions` ■ `variables` ■ `iviewx` (zero or one) ■ `iviewy` (zero or one) For *callable* workflows, you must define the following subelements: ■ `%node` ■ `description` ■ `called` ■ `nexts` ■ `actions` ■ `variables` ■ `iviewx` (zero or one) ■ `iviewy` (zero or one)	See "Template Definition DTD Example" on page A-115.

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
`Start (cont)`	For *time-triggered* workflows, you must define the following subelements: ■ `%node` ■ `description` ■ `timed` ■ `dateexpression` ■ `reschedamount` ■ `reschedunits` ■ `recoverable` (zero or one) ■ `calendarname` ■ `startorg` ■ `nexts` ■ `actions` ■ `variables` ■ `iviewx` (zero or one) ■ `iviewy` (zero or one) For *event-triggered* workflows, you must define the following subelements: ■ `%node` ■ `description` ■ `event` ■ `root` ■ `key` ■ `condition` ■ `startorg` ■ `nexts` ■ `actions` ■ `variables`	

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
`Start (cont)`	For *plug-in-triggered* workflows, you must define the following subelements: ■ `%node` ■ `description` ■ `plugin-data` ■ `startorg` ■ `nexts` ■ `actions` ■ `variables`	

`startorg`	Organization associated with the `Start` node.	`<startorg>`
		`&quot;ORG1&quot;`
		`</startorg>`

`stopwhentaskdone`	Flag specifying whether to stop execution (and rescheduling) when the task (`true`) or workflow (`false`) completes.	`<stopwhentaskdone>`
		`true`
		`</stopwhentaskdone>`

`target`	Unique numerical value that identifies the element upon which you are operating, and that is meaningful within the context of the parent element.	`<target>`
		`963511923442`
		`</target>`

`Task`	`Task` node. You must define the following subelements: ■ `%node` ■ `name` ■ `priority` ■ `donewithoutdoit` ■ `doitifdone` ■ `unmarkdone` ■ `modifiable` ■ `reassignment` ■ `nexts` ■ `actions`	See "Template Definition DTD Example" on page A-115.
`template-entry`	Target of an addressed JMS message.	

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value

`template-name`	Workflow template name.	`<template-name>`
		`ShipBill`
		`</template-name>`

`templateid`	Template ID of the workflow to start, using the `ActionWorkflowStart` action.	`<templateId>`
		`3`
		`</templateId>`

`time-to-live`	Time-to-live value.	`<time-to-live>`
		`0`
		`</time-to-live>`

`timed`	Flag (`true` or `false`) specifying whether or not the `Start` node is time-triggered.	`<timed>`
		`true`
		`</timed>`

`to`	E-mail address(es) of the primary receipient(s) to be sent an e-mail message, as defined by the `ActionSendEMail` action.
	You can define zero or more occurrences of the following subelement: `addressee`
		`<to>`
		`<addressee>`
		`<name>`
		`joe@work`
		`</name>`
		`<expression>`
		`false`
		`</expression>`
		`<type>`
		`address`
		`</type>`
		`</addressee>`
		`</to>`

`topic`	JMS topic to which an XML document is sent or retrieved.
	You must define the `expression` attribute to specify whether or not the value specified is an expression. This attribute defaults to `false`.
		`<topic>`
		`Topic1`
		`</topic`

`transaction`	Boolean flag specifying whether or not the workflow can be part of a transaction.	`<transaction>`
		`false`
		`</transaction>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
transform-document	XSLT document used to transform input-expression.	
transform-source	Flag that indicates whether transform-document is obtained from a workflow expression (E) or from the path to a XML repository file (R).	<transform-source> E </transform-source
true	Actions associated with a Decision node's true branch. You can define zero or more occurrences of the following subelement: %action-types.	<true> <ActionWorkflowAbort> <notes> This is a note. </notes> </ActionWorkflowAbort> </true>
truenexts	String specifying the next node associated with a Decision node's true branch. You can define zero or more occurrences of the following subelement: next.	<truenexts> 963410128518 </truenexts>
type	For a variable, specifies one of the following variable types: boolean, date, double, entity (Entity EJB), integer, object, session (Session EJB), string, or xml. For a routecondition, specifies the type of the addressee (user, userinrole, or role) specified for the to element. For an addressee, specifies the type of the addressee (address, user, or role) specified for the name element.	<type>string</type>
unmarkdone	Permission flag (true or false) specifying whether or not the task can be executed once it has been marked done, using the methods described in "Marking a Task as Complete or Incomplete" on page 21-18.	<unmarkdone> true </unmarkdone>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
`user`	User or user in role. This value can specify an existing user or role, or a workflow expression that evaluates to the user or role to which the task is assigned by the `ActionTaskAssignUser` action. If it is a workflow expression, the associated `expression` flag must be set to `true`.	`<user>true</user>`

`usetimeexpression`	Flag specifying whether to use the specified `expression` (`true`) or `executiontime` (`false`) to trigger an `ActionTimedEvent` action,	`<usetimeexpression>`
		`true`
		`</usetimeexpression>`

`value`	Parameter value that is meaningful within the context of the parent element.	`<value>`
		`5`
		`</value>`

`variable`	Variable defined for the workflow template definition.
	For a `Workflow`, you must define the following subelements:
	■ `name`
	■ `notes`
	■ `type` or `plugin-data`
	■ `xmltype`
	■ `inmandatory`
	■ `in`
	■ `out`
	For a `Start` node or `ActionSendXMLToClient` action, you must define the following subelements:
	■ `name`
	■ `expression`
	For an `ActionExpression` action, you must define the following subelements:
	■ `name`
	■ `expression`
		`<variable>`
		`<name>`
		`TotalPrice`
		`</name>`
		`<notes>`
		`This is a note.`
		`</notes>`
		`<type>`
		`double`
		`</type>`
		`<xmltype>`
		`</xmltype>`
		`<inmandatory>`
		`false`
		`</inmandatory>`
		`<in>false</in>`
		`<out>false</out>`
		`</variable>`

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
variables	Variables defined for the workflow template definition. You can define zero or more occurrences of the following subelement: variable.	<variables> <variable> <name> TotalPrice </name> <notes> This is a note. </notes> <type> double </type> <xmltype> </xmltype> <inmandatory> false </inmandatory> <in>false</in> <out>false</out> </variable> <variables>

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
`Workflow`	Root element. You must define the following subelements: ■ `name` ■ `effective` ■ `changedby` ■ `changeddate` ■ `notes` ■ `idexpression` ■ `active` ■ `audit-enabled` ■ `plugins` (zero or one) ■ `nodes` ■ `variables` ■ `error-handlers` (zero or one) You must define the following attributes: ■ `major-version`: Major version of software. Defaults to `1`. ■ `minor-version`: Minor version of software. Defaults to `0`. ■ `build-number`: Build number of software. Defaults to `0`.	See "Template Definition DTD Example" on page A-115.
`x`	X-coordinate of a Studio shape.	`<x>20</x>`

`xml`	XML document structure consisting of any combination of elements and text.	`<xml>`
		`  <order>`
		`  &quot;new&quot;`
		`  </order>`
		`</xml>`

`xmltype`	XML document type, if `variable` type is set to `xml`.	
`xsl-parameters`	Parameters used for XML transformation. You must define one or more `xsl-parameter` subelements.	

Table A-18 Template Definition DTD Elements (Continued)

Element	Description	Example Value
xsl-parameter	Parameter used for XML transformation. You must define the following subelements: ■ name ■ expression	
y	Y-coordinate of a Studio shape.	<y>40</y>

Entity Descriptions

The following table describes the Template Definition entities.

Table A-19 Template Definition DTD Entities

Entity	Description
%action-types	Defines all action types, including the %std-action-types and the following types: ■ ActionInvokeErrorHandler ■ ActionSetErrorHandler
%commit-action-types	Commit transaction action types, including the %std-action-types and the following types: ■ ActionExitErrorHandler ■ ActionSetErrorHandler
%node	Node characteristics, including the following: ■ id ■ x ■ y ■ notes

Table A-19 Template Definition DTD Entities (Continued)

Entity	Description
`%rollback-action-types`	Rollback transaction action types, including the following: ■ `ActionAuditEntry` ■ `ActionBusinessOperation` ■ `ActionCallProgram` ■ `ActionCondition` ■ `ActionExitErrorHandler` ■ `ActionExpression` ■ `ActionNoOp` ■ `ActionPlugin` ■ `ActionPostXMLEvent` ■ `ActionSendEMail` ■ `ActionSendXMLToClient`

Table A-19 Template Definition DTD Entities (Continued)

Entity	Description
%std-action-types	Standard action types, including the following: ■ ActionAuditEntry ■ ActionBusinessOperation ■ ActionCallProgram ■ ActionCancelEvent ■ ActionCondition ■ ActionExpression ■ ActionNoOp ■ ActionPlugin ■ ActionPostXMLEvent ■ ActionSendEMail ■ ActionSendXMLToClient ■ ActionTaskAssignRole ■ ActionTaskAssignRoutingTable ■ ActionTaskAssignUser ■ ActionTaskDoit ■ ActionTaskDone ■ ActionTaskDueDate ■ ActionTaskSetComment ■ ActionTaskSetPriority ■ ActionTaskUnassign ■ ActionTaskUndo ■ ActionTimedEvent ■ ActionWorkflowAbort ■ ActionWorkflowDone ■ ActionWorkflowSetComment ■ ActionWorkflowStart ■ ActionXSLTransform

Template Definition DTD Example

The following example consists of excerpts from the Order Processing template definition that is provided in the `examples` directory. This example illustrates a valid application of the Template Definition DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<Workflow major-version="1" minor-version="2" build-number="0">
  <name>Order Processing</name>
  <effective>200007120000</effective>
  <changedby>joe</changedby>
  <changeddate>20011281424</changeddate>
  <notes>This is a note.</notes>
  <idexpression>&amp;quot;Order&amp;quot;+&amp;quot; &amp;quot;+:OrderNumber</idexpression>
  <active>true</active>
  <audit-enabled>false</audit-enabled>
  <nodes>
.
.
.
  </nodes>
  <variables>
.
.
.
  </variables>
  <error-handlers>
.
.
.
  </error-handlers>
</Workflow>
```

The following sections provide detailed examples of each node type: `Decision`, `Done`, `Event`, `Join`, `Start`, and `Task`. For additional examples of each element, see "Element Descriptions" on page A-62.

Decision Node Example

The following excerpt provides an example of a `Decision` node.

```
    Decision>
      <id>963410131712</id>
      <x>100</x>
      <y>140</y>
```

```
  <notes></notes>
  <condition>:Confirm=&quot;yes&quot;</condition>
  <truenexts>
    <next>963410119665</next>
  </truenexts>
  <falsenexts>
  </falsenexts>
  <actions>
    <false>
      <ActionPostXMLEvent>
        <notes></notes>
        <xml>
          <order>
            <status>&quot;cancelled&quot;</status>
            <ordernumber>:OrderNumber</ordernumber>
          </order>
        </xml>
      </ActionPostXMLEvent>
    </false>
    <true>
    </true>
  </actions>
</Decision>
```

Done Node Example

The following excerpt provides an example of a `Done` node.

```
<Done>
  <id>3</id>
  <x>140</x>
  <y>490</y>
  <notes></notes>
  <actions>
  </actions>
</Done>
```

Event Node Example

The following excerpt provides an example of an `Event` node.

```
<Event>
  <id>991155339988</id>
  <x>370</x>
  <y>10</y>
  <notes></notes>
  <iviewx>290</iviewx>
```

```
      <iviewy>0</iviewy>
      <description>Watch for Cancellation</description>
      <root>cancelledorder</root>
      <key>$OrderID</key>
      <condition></condition>
      <nexts>
        <next>3</next>
      </nexts>
      <actions>
      </actions>
      <variables>
      </variables>
    </Event>
```

Join Node Example

The following excerpt provides an example of a `Join` node.

```
    <Join>
      <id>963511805733</id>
      <x>110</x>
      <y>230</y>
      <notes></notes>
      <and>true</and>
      <nexts>
        <next>963511923442</next>
      </nexts>
    </Join>
```

Start Node Example

The following excerpt provides an example of a `Start` node.

```
<Start>
      <id>1</id>
      <x>20</x>
      <y>40</y>
      <notes></notes>
      <iviewx>150</iviewx>
      <iviewy>50</iviewy>
      <description>Start</description>
      <called>true</called>
      <nexts>
        <next>2</next>
        <next>963410125634</next>
      </nexts>
      <actions>
```

```
    </actions>
    <variables>
      <variable>
        <name>CustomerName</name>
       <expression>XPath(&quot;/order/customer/name/text()&quot;)</expression>
      </variable>
      <variable>
        <name>CustomerId</name>
        <expression>XPath(&quot;/order/customer/id/text()&quot;)</expression>
      </variable>
      <variable>
        <name>orderstatus</name>
        <expression>XPath(&quot;/order/status/text()&quot;)</expression>
      </variable>
      <variable>
        <name>ordernumber</name>
        <expression>XPath(&quot;/order/number/text()&quot;)</expression>
      </variable>
      <variable>
        <name>CustomerMail</name>
      <expression>XPath(&quot;/order/customer/email/text()&quot;)</expression>
      </variable>
      <variable>
        <name>ItemName</name>
        <expression>XPath(&quot;/order/item/name/text()&quot;)</expression>
      </variable>
      <variable>
        <name>ItemNumber</name>
        <expression>XPath(&quot;/order/item/id/text()&quot;)</expression>
      </variable>
      <variable>
        <name>ItemQuantity</name>
       <expression>XPath(&quot;/order/item/quantity/text()&quot;)</expression>
      </variable>
      <variable>
        <name>ItemPrice</name>
        <expression>XPath(&quot;/order/item/price/text()&quot;)</expression>
      </variable>
      <variable>
        <name>CustomerState</name>
      <expression>XPath(&quot;/order/customer/state/text()&quot;)</expression>
      </variable>
    </variables>
  </Start>
```

Task Node Example

The following excerpt provides an example of a `Task` node.

```
<Task>
  <id>2</id>
  <x>100</x>
  <y>20</y>
  <notes></notes>
  <name>Confirm Order</name>
  <priority>1</priority>
  <donewithoutdoit>true</donewithoutdoit>
  <doitifdone>true</doitifdone>
  <unmarkdone>true</unmarkdone>
  <modifiable>true</modifiable>
  <reassignment>true</reassignment>
  <nexts>
    <next>963410131712</next>
  </nexts>
  <actions>
    <created>
    </created>
    <activated>
      <ActionTaskAssignUser>
        <notes></notes>
        <target>2</target>
        <user>joe</user>
        <expression>false</expression>
        <role>false</role>
      </ActionTaskAssignUser>
    </activated>
    <executed>
      <ActionSendXMLToClient>
        <id>959395846210</id>
        <notes></notes>
        <xml>
          <message-box title="&quot;Order Confirmation&quot;"
          style="&quot;question&quot;" options="&quot;yes_no&quot;">
          &quot;Do you Confirm this order?&quot;
          <actionid>&quot;959395846210&quot;</actionid>
          </message-box>
        </xml>
        <variables>
          <variable>
            <name>Confirm</name>
            <expression>XPath(&quot;/message-box/attribute::option&quot;)
            </expression>
          </variable>
        </variables>
        <actions>
          <ActionTaskDone>
            <notes></notes>
            <target>2</target>
```

```
            </ActionTaskDone>
          </actions>
        </ActionSendXMLToClient>
      </executed>
      <markeddone>
      </markeddone>
    </actions>
  </Task>
```

Workload Request DTD

The Workload Request DTD describes the format of the XML document that is used to generate runtime workload statistics. (The Workload Response DTD describes the format of the returned statistics report.) For information about the method used to query runtime workloads, see "Querying the Run-Time Workload" on page 22-18.

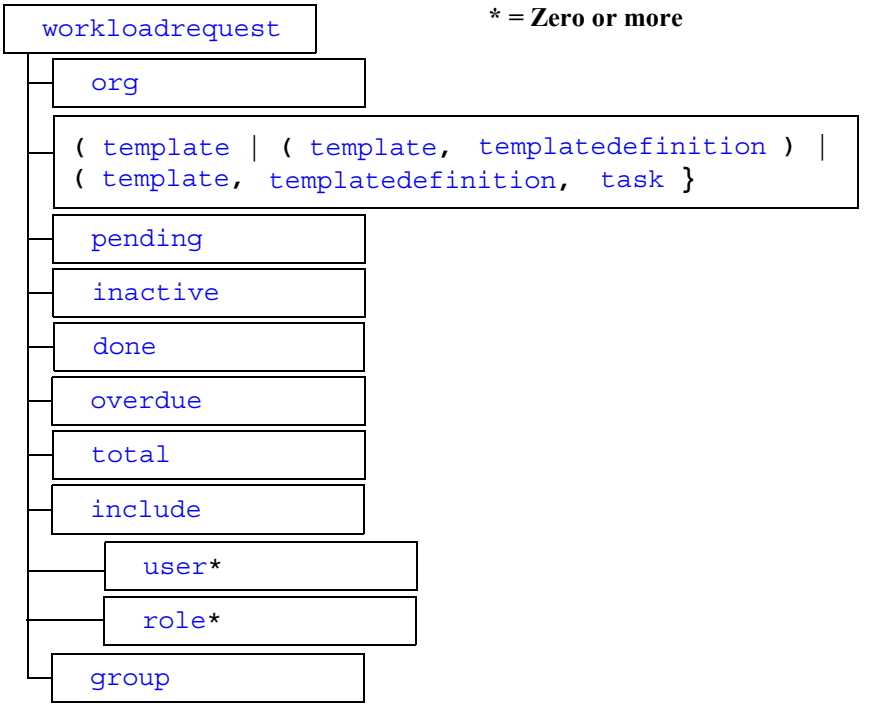
The following sections describe the Workload Request DTD, including:

- Hierarchy Diagram
- DTD Format
- Element Descriptions

Hierarchy Diagram

The following diagram illustrates the Workload Request DTD hierarchy.

Figure A-17 Workload Request DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Workload Request DTD, `WorkloadReq.dtd`.

```
<!ELEMENT workloadrequest (org, template
                           | (template, templatedefinition)
                           | (template, templatedefinition, task),
                          pending, inactive, done, overdue,
                          total, include, group)>
<!ELEMENT include (#PCDATA, user* | role*)>
<!ELEMENT org (#PCDATA)>
<!ELEMENT template (#PCDATA)>
<!ELEMENT templatedefinition (#PCDATA)>
<!ELEMENT task (#PCDATA)>
```

```
<!ELEMENT pending (#PCDATA)>
<!ELEMENT inactive (#PCDATA)>
<!ELEMENT done (#PCDATA)>
<!ELEMENT overdue (#PCDATA)>
<!ELEMENT total (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT group (#PCDATA)>
```

Element Descriptions

The following table describes the elements of the Workload Request DTD.

Table A-20 Workload Request DTD Elements

Element	Description
done	Boolean flag specifying whether or not to include workflows marked as complete.
group	Boolean flag specifying whether or not to group report and display totals only.
inactive	Boolean flag specifying whether or not to include workflows that are inactive.
include	List of users and/or roles to include in the report. You can define the following subelements: ■ user (zero or more) ■ role (zero or mroe)
org	Organization for which the workload report is being generated.
overdue	Boolean flag specifying whether or not to include workflows that are overdue.
pending	Boolean flag specifying whether or not to include workflows that are pending.
role	Name of role to include in the report.
task	ID of the task for which the workload report is being generated.

Table A-20 Workload Request DTD Elements (Continued)

Element	Description
`template`	ID of the template for which the workload report is being generated.
`templatedefinition`	ID of the template definition for which the workload report is being generated.
`total`	Boolean flag specifying whether or not to include all tasks, regardless of task state.
`user`	Name of user to include in the report.
`workloadrequest`	Root element. You must define the following subelements: ■ `org` ■ `template | (template, templatedefinition) | (template, templatedefinition, task)` ■ `pending` ■ `inactive` ■ `done` ■ `overdue` ■ `total` ■ `include` ■ `group`

Workload Response DTD

The Workload Response DTD describes the format of the XML document returned when querying runtime workloads. (The Workload Request DTD describes the format of the XML document used to request the query.) For information about querying the runtime workload, see “Querying the Run-Time Workload” on page 22-18.

The following sections describe the Workload Response DTD, including:

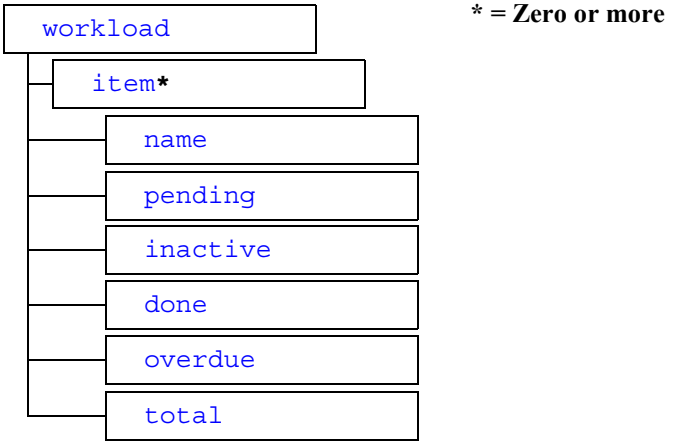
- Hierarchy Diagram
- DTD Format

- Element Descriptions

Hierarchy Diagram

The following diagram illustrates the Workload Response DTD hierarchy.

Figure A-18 Workload Response DTD Hierarchy Diagram



DTD Format

The following listing shows the format of the Workload Response DTD, `WorkloadResp.dtd`.

```
<!ELEMENT workload (item*)>
<!ELEMENT item (name, pending, inactive, done, overdue, total)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT pending (#PCDATA)>
<!ELEMENT inactive (#PCDATA)>
<!ELEMENT done (#PCDATA)>
<!ELEMENT overdue (#PCDATA)>
<!ELEMENT total (#PCDATA)>
```

Element Descriptions

The following table describes the elements of the Workload Response DTD.

Table A-21 Workload Response DTD Elements

Element	Description
done	Number of tasks marked as complete for item.
inactive	Number of inactive tasks for item.
item	Item for which you are gathering the workload report. You must define the following subelements: ■ name ■ pending ■ inactive ■ done ■ overdue ■ total
name	Name of the item for which you are gathering the workload report.
overdue	Number of overdue tasks for item.
pending	Number of pending tasks for item.
total	Total number of tasks for item.
workload	Root element. You must define zero or more occurrences of the following subelement: item

B Value Object Summary

This appendix describes the BPM value objects and their methods, including the following:

- BusinessCalendarInfo Object
- EventKeyInfo Object
- InstanceInfo Object
- OrganizationInfo Object
- PermissionInfo Object
- RepositoryFolderInfo Object
- RepositoryFolderInfoHelper Object
- RerouteInfo Object
- RoleInfo Object
- RolePermissionInfo Object
- TaskInfo Object
- TemplateDefinitionInfo Object
- TemplateInfo Object
- UserInfo Object
- UserPermissionInfo Object
- VariableInfo Object
- VersionInfo Object

- XMLEntityInfo Object
- XMLEntityInfoHelper Object

For more information about accessing object data, see "Using Value Objects" on page 5-1.

BusinessCalendarInfo Object

The `com.bea.wlpi.common.BusinessCalendarInfo` object maintains object data pertaining to business calendars.

You can use the following constructors to create a new `BusinessCalendarInfo` object:

```
public BusinessCalendarInfo(
  java.lang.String id,
  java.lang.String name,
  java.lang.String timezone,
  java.lang.String xml
)

public BusinessCalendarInfo(
  java.lang.String id,
  java.lang.String name,
  java.lang.String timezone,
  java.lang.String xml,
  boolean published
)
```

The following table describes the `BusinessCalendarInfo` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Table B-1 BusinessCalendarInfo Object Data

Object Data	Constructor Parameter	Get Method	Set Method
Business calendar ID	*id*	`public java.lang.String getId()`	`public void setId(java.lang.String id)`
Business calendar name	*name*	`public java.lang.String getName()`	`public void setName(java.lang.String name)`
Timezone in which the business calendar operates	*timezone*	`public java.lang.String getTimeZone()`	`public void setTimeZone(java.lang.String id)`
Business calendar definition (XML) For a description of the Business Calendar DTD format, see "Business Calendar DTD" on page A-11.	*xml*	`public java.lang.String getXML()`	`public void setXML(java.lang.String xml)`
Boolean flag specifying whether or not the business calendar is published.	*published*	`public boolean isPublished()`	`N/A`

For more information, see the `com.bea.wlpi.common.BusinessCalendarInfo` javadoc.

EventKeyInfo Object

The `com.bea.wlpi.common.EventKeyInfo` object maintains object data pertaining to event keys.

An event key specifies a unique value that is encoded as part of the event data, and that is used to identify workflow definitions or instances associated with an incoming XML document. The process engine generates an EventKey table that associates a workflow expression to an event based on the event content type and descriptor. The workflow expression is used to evaluate the event key value for an incoming event.

You can use the following constructor to create a new `EventKeyInfo` object:

```
public EventKeyInfo(
  java.lang.String contentType,
  java.lang.String eventDescriptor,
  java.lang.String expr,
  java.lang.String plugin,
  int fieldId
)
```

The following table describes the `EventKeyInfo` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Note: For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.

Table B-2 EventKeyInfo Object Data

Object Data	Parameter	Get Method	Set Method
MIME Content type (for example, `text/xml`)	*contentType*	`public final java.lang.String getContentType()`	N/A

Table B-2 EventKeyInfo Object Data (Continued)

Object Data	Parameter	Get Method	Set Method
If the MIME Content type is set to `text/xml`, specifies one of the following: ■ Public ID of the XML document DTD ■ System ID if no public ID is quoted ■ Root element name if no DOCTYPE is specified Otherwise, specifies an event descriptor, in a plug-in defined format	*eventDescriptor*	`public final java.lang.String getEventDescriptor( )`	N/A
Expression that yields a unique event key for a document with the specified content type and format description	*expr*	`public final java.lang.String getExpr()`	`public final void setExpr(java.lang.String expr)`
Plug-in that supplies the field type required to evaluate the expression; if null, and the content type is `text/xml`, the default XML field type is used	*plugin*	`public final java.lang.String getPlugin()`	N/A
Field ID assigned by the plug-in	*fieldId*	`public final int getFieldId()`	N/A

For more information, see the `com.bea.wlpi.common.EventKeyInfo` javadoc.

InstanceInfo Object

The `com.bea.wlpi.common.InstanceInfo` object maintains object data pertaining to workflow instances.

You can use the following constructors to create a new `InstanceInfo` object:

```
public InstanceInfo(
  java.lang.String id,
  java.lang.String templateId,
  java.lang.String templateDefinitionId,
  java.lang.String name,
  java.lang.String initiator,
  java.lang.String parentId,
  java.sql.Timestamp started,
  java.sql.Timestamp completed,
  java.lang.String idString,
  int state,
  java.lang.String comment
)

public InstanceInfo(
  java.lang.String id,
  java.lang.String templateId,
  java.lang.String templateDefinitionId,
  java.lang.String name,
  java.lang.String initiator,
  java.lang.String parentId,
  java.sql.Timestamp started,
  java.sql.Timestamp completed,
  java.lang.String idString,
  int state,
  java.lang.String comment
  java.util.Map pluginData
)
```

The following table describes the `InstanceInfo` object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Note: For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.

Table B-3 InstanceInfo Object Data

Object Data	Constructor Parameter	Get Method
Instance ID	*id*	`public final java.lang.String getId()`
Template ID	*templateId*	`public final java.lang.String getTemplateId()`
Template definition ID	*templateDefinitionId*	`public final java.lang.String getTemplateDefinitionId()`
Template definition name	*name*	`public final java.lang.String getName()`
Initiator ID	*initiator*	`public final java.lang.String getInitiator()`
Parent ID	*parentId*	`public final java.lang.String getParentId()`
Date and time at which the workflow was started	*started*	`public final java.sql.Timestamp getStarted()`
Date and time at which the workflow was completed	*completed*	`public final java.sql.Timestamp getCompleted()`

Table B-3 InstanceInfo Object Data (Continued)

Object Data	Constructor Parameter	Get Method
Instance label	*idString*	`public final java.lang.String getIdString()`
Workflow instance state that can be expressed as one of the following static integer values: ■ `WORKFLOW_STATE_ACTIVE`: instance is running normally ■ `WORKFLOW_STATE_SUSPENDED`: instance is suspended	*state*	`public final int getState()`
Comment	*comment*	`public final java.lang.String getComment()`
Plug-in data	*pluginData*	`public java.lang.Object getPluginInstanceData( java.lang.String pluginName)`

For more information, see the `com.bea.wlpi.common.InstanceInfo` javadoc.

OrganizationInfo Object

The `com.bea.wlpi.common.OrganizationInfo` object maintains object data pertaining to the organization.

You can use the following constructors to create a new `OrganizationInfo` object:

```
public OrganizationInfo(
  java.lang.String orgId
)

public OrganizationInfo(
  java.lang.String orgId,
  java.lang.String calendarId
)
```

The following table describes the `OrganizationInfo` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Table B-4 OrganizationInfo Object Data

Object Data	Constructor Parameter	Get Method	Set Method
Organization ID	*orgId*	`public final java.lang.String getOrgId()`	N/A
ID of the business calendar to be used	*calendarId*	`public final java.lang.String getCalendarId()`	`public final void setCalendarId(java.lang.String calendarId)`

For more information, see the `com.bea.wlpi.common.OrganizationInfo` javadoc.

PermissionInfo Object

The `com.bea.wlpi.common.security.PermissionInfo` object maintains permission information for a *principal* (role or user), as defined by the associated `com.bea.wlpi.common.security.EnumPermission` object.

You can use the following constructor to create a a new `PermissionInfo` object:

```
public PermissionInfo(
  java.lang.String principalId
)
```

The following table describes the `PermissionInfo` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Table B-5 PermissionInfo Object Data

Object Data	Constructor Parameter	Get Method	Set Method
Principal ID	*principalId*	`public java.lang.String getPrincipalId()`	N/A
Permission level and value The permission level can be set to one of the `com.bea.wlpi.common.security.EnumPermission` static integer values defined in the table "Permissions Overview" on page 9-71. The permission value can be set to `true` (enabled) or `false` (disabled).	N/A	N/A	`public void setPermission(com.bea.wlpi.common.security.EnumPermission permission, boolean value)`

Table B-5 PermissionInfo Object Data (Continued)

Object Data	Constructor Parameter	Get Method	Set Method
Boolean flag specifying whether or not a permission level is in effect For more information about the permission levels that can be set, see "Permissions Overview" on page 9-71.	N/A	`public boolean hasPermission(com.bea.wlpi.common.security.EnumPermission permission)`	N/A

For more information, see the `com.bea.wlpi.common.security.PermissionInfo` javadoc. See also "RolePermissionInfo Object" on page B-18 and "UserPermissionInfo Object" on page B-28.

RepositoryFolderInfo Object

Note: The `com.bea.wlpi.common.RepositoryFolderInfoHelper` object provides an auxiliary class to the `com.bea.eci.repository.helper.RepositoryFolderInfo` object to support the importing and exporting of object data. For more information, see "RepositoryFolderInfoHelper Object" on page B-13.

The `com.bea.eci.repository.helper.RepositoryFolderInfo` object maintains object data pertaining to the folders within the XML repository.

You can use the following constructor to create a new `OrganizationInfo` object:

```
public RepositoryFolderInfo(
  java.lang.String type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes
)
```

The following table describes the `RepositoryFolderInfo` object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Table B-6 RepositoryFolderInfo Object Data

Object Data	Constructor Parameter	Get Method
Folder type	*type*	`public java.lang.String getType()`
Folder name	*name*	`public java.lang.String getName()`
Folder description	*desc*	`public java.lang.String getDescription()`
Folder notes	*notes*	`public java.lang.String getNotes()`
Date on which folder was created	N/A	`public java.sql.Timestamp getCreatedOn()`
Date on which folder was last modified	N/A	`public java.sql.Timestamp getLastModifiedOn()`

For more information, see the `com.bea.eci.repository.helper.RepositoryFolderInfo` javadoc.

RepositoryFolderInfoHelper Object

The `com.bea.wlpi.common.RepositoryFolderInfoHelper` object provides an auxiliary class to the `com.bea.eci.repository.helper.RepositoryFolderInfo` object to support the importing and exporting of object data.

You can use the following constructors to create a new `RepositoryFolderInfoHelper` object:

```
public RepositoryFolderHelperInfo(
  java.lang.String type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes
)

public RepositoryFolderInfoHelper(
  java.lang.String type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  com.bea.wlpi.common.RepositoryFolderInfoHelper parent
)
```

The following table describes the `RepositoryFolderInfoHelper` object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Table B-7 RepositoryFolderInfo Object Data

Object Data	Constructor Parameter	Get Method	Set Method
Folder type	*type*	`public java.lang.String getFolderType()`	N/A
Folder name	*name*	`public java.lang.String getName()`	N/A

Table B-7 RepositoryFolderInfo Object Data (Continued)

Object Data	Constructor Parameter	Get Method	Set Method
Folder description	*desc*	`public java.lang.String getDescription()`	N/A
Folder notes	*notes*	`public java.lang.String getNotes()`	N/A
Parent	*parent*	`public com.bea.wlpi.common.RepositoryFolderInfoHelper getParent()`	`public void setParent(com.bea.wlpi.common.RepositoryFolderInfoHelper p)`
XML repository publishable object contents	N/A	`public java.lang.Object getContents()`	N/A
XML repository publishable entry name	N/A	`public java.lang.String getEntryName()`	N/A
XML repository publishable owner name	N/A	`public java.lang.String getOwnerName()`	N/A
XML repository publishables referenced by specified publishable objects	N/A	`public java.util.List getReferencedPublishables(java.util.Map publishables)`	N/A
XML repository publishable object type	N/A	`public int getType()`	N/A

For more information, see the `com.bea.wlpi.common.RepositoryFolderInfoHelper` javadoc.

RerouteInfo Object

The `com.bea.wlpi.common.RerouteInfo` object maintains object data pertaining to task reroutes.

You can use the following constructor to create a new `RerouteInfo` object:

```
public RerouteInfo(
  java.lang.String id,
  java.lang.String from,
  java.lang.String to,
  int type,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry
)
```

The following table describes the `RerouteInfo` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Table B-8 RerouteInfo Object Data

Object Data	Constructor Parameter	Get Method	Set Method
Task reroute ID	*id*	`public final java.lang.String getId()`	N/A
User or role for whom the task will be rerouted	*from*	`public final java.lang.String getFrom()`	`public void setFrom(java.lang.String from)`
User or role to whom the rerouted tasks will be assigned	*to*	`public final java.lang.String getTo()`	`public void setTo(java.lang.String to)`

Table B-8 RerouteInfo Object Data (Continued)

Object Data	Constructor Parameter	Get Method	Set Method
Type of task reroute. Can be represented by one of the following static integer values, defined by static final members of the `ReRouteInfo` class: ■ `TYPE_USER`: Reroute task to another user ■ `TYPE_USERINROLE`: Reroute task to a user in a role, using load balancing ■ `TYPE_ROLE`: Reroute task to a role	*type*	`public final int getType()`	`public void setType(int type)`
Date and time at which the task reroute becomes effective	*effective*	`public final java.sql.Timestamp getEffective()`	`public void setEffective(java.sql.Timestamp effective)`
Date and time at which the task reroute expires	*expiry*	`public final java.sql.Timestamp getExpiry()`	`public void setExpiry(java.sql.Timestamp expiry)`

For more information, see the `com.bea.wlpi.common.RerouteInfo` javadoc.

RoleInfo Object

The `com.bea.wlpi.common.RoleInfo` object maintains object data pertaining to roles.

You can use the following constructors to create a new `RoleInfo` object:

```
public RoleInfo(
  java.lang.String roleId,
  java.lang.String orgId
)

public RoleInfo(
  java.lang.String roleId,
  java.lang.String orgId,
  java.lang.String calendarId
)

public RoleInfo(
  java.lang.String roleId,
  java.lang.String orgId,
  java.lang.String calendarId,
  java.lang.String groupId
)
```

The following table describes the `RoleInfo` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Table B-9 RoleInfo Object Data

Object Data	Constructor Parameter	Get Method	Set Method
Role ID	*roleId*	`public final java.lang.String getRoleId()`	N/A
ID of the organization within which the role is defined	*orgId*	`public final java.lang.String getOrgId()`	N/A

Table B-9 RoleInfo Object Data (Continued)

Object Data	Constructor Parameter	Get Method	Set Method
ID of the business calendar to be used	*calendarId*	`public final java.lang.String getCalendarId()`	`public void setCalendarId(java.lang.String calendarId)`
ID of the security realm group to which this role maps	*groupId*	`public final java.lang.String getGroupId()`	`public final java.lang.String setGroupId(java.lang.String groupId)`

For more information, see the `com.bea.wlpi.common.RoleInfo` javadoc.

RolePermissionInfo Object

The `com.bea.wlpi.common.security.RolePermissionInfo` object maintains object data pertaining to role permissions, as defined by the associated `com.bea.wlpi.common.security.EnumPermission` object.

You can use the following constructors to create a new `RolePermissionInfo` object:

```
public RolePermissionInfo(
  java.lang.String roleId,
  java.lang.String orgId
)

public RolePermissionInfo(
  java.lang.String roleId,
  java.lang.String orgId,
  java.lang.String groupId
)
```

The following table describes the `RolePermissionInfo` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Note: The `RolePermissionInfo` class extends the `PermissionInfo` class. For information about the get and set methods, including how to set permissions for a specific role, see "PermissionInfo Object" on page B-10.

Table B-10 RolePermissionInfo Object Data

Object Data	Constructor Parameter	Get Method	Set Method
Role ID	*roleId*	N/A	N/A
ID of the organization within which the role is defined	*orgId*	`public java.lang.String getOrgId()`	`public void setOrgId(java.lang.String orgId)`
ID of the security realm group within which the role is defined	*groupId*	`public java.lang.String getGroupId()`	`public void setGroupId(java.lang.String groupId)`

For more information, see the `com.bea.wlpi.common.security.RolePermissionInfo` javadoc. See also "PermissionInfo Object" on page B-10.

TaskInfo Object

The `com.bea.wlpi.common.TaskInfo` object maintains object data pertaining to workflow tasks.

You can use the following constructor to create a new `TaskInfo` object:

```
public TaskInfo(
  java.lang.String templateId,
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.String name,
  java.lang.String assignee,
  boolean assigneeIsRole,
  java.lang.String workflow,
  java.lang.String workflowId,
  int priority,
  java.sql.Timestamp started,
  java.sql.Timestamp completed,
  java.sql.Timestamp due,
  java.lang.String comment,
  boolean doneWithoutDoit,
  boolean doitIfDone,
  boolean unmarkDone,
  boolean modifiable,
  boolean reassignment
)
```

The following table describes the `TaskInfo` object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Table B-11 TaskInfo Object Data

Object Data	Constructor Parameter	Get Method
Template ID	*templateId*	`public final java.lang.String getTemplateId()`

Table B-11 TaskInfo Object Data (Continued)

Object Data	Constructor Parameter	Get Method
Template definition ID	*templateDefinitionId*	`public final java.lang.String getTemplateDefinitionId()`
Instance ID	*instanceId*	`public final java.lang.String getInstanceId()`
Task ID	*taskId*	`public final java.lang.String getTaskId()`
Task name	*name*	`public final java.lang.String getTaskName()`
Assignee (user or role) to which the task is assigned	*assignee*	`public final java.lang.String getAssignee()`
Boolean flag specifying whether or not the assignee is a role	*assigneeIsRole*	`public final boolean isAssignedToRole()`
Workflow name	*workflow*	`public final java.lang.String getWorkflowName()`
Workflow ID	*workflowId*	`public final java.lang.String getWorkflowId()`
Task priority	*priority*	`public final int getPriority()`
Date and time at which the task was started	*started*	`public final java.sql.Timestamp getStarted()`
Date and time at which the task was completed	*completed*	`public final java.sql.Timestamp getCompleted()`

Table B-11 TaskInfo Object Data (Continued)

Object Data	Constructor Parameter	Get Method
Date and time at which the task should be due	*due*	`public final Timestamp getDue()`
Comment	*comment*	`public final java.lang.String getComment()`
Permission to mark a task as complete	*doneWithoutDoIt*	`public final boolean getDoneWithoutDoIt()`
Permission to execute a task after it has been marked as complete	*doitIfDone*	`public final boolean getDoitIfDone()`
Permission to mark a task as incomplete	*unmarkDone*	`public final boolean getUnmarkDone()`
Permission to modify the task run-time properties	*modifiable*	`public final boolean getModifiable()`
Permission to reassign a task instance to another participant	*reassignment*	`public final boolean getReassignment()`
Task status that can be represented by one of the following static integer values, defined by static final members of the `TaskInfo` class: ■ `STATUS_COMPLETE`: task is complete ■ `STATUS_INACTIVE`: task has not yet started ■ `STATUS_OVERDUE`: task has started, but is not yet complete and the due date is past ■ `STATUS_PENDING`: task has started, but is not yet complete	N/A	`public final int getStatus()`

For more information, see the com.bea.wlpi.common.TaskInfo javadoc.

TemplateDefinitionInfo Object

The com.bea.wlpi.common.TemplateDefinitionInfo object maintains object data pertaining to the template definition.

You can use the following constructors to create a new TemplateDefinitionInfo object:

```
public TemplateDefinitionInfo(
  java.lang.String id,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry,
  boolean active
)

public TemplateDefinitionInfo(
  java.lang.String id,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry,
  boolean active,
  java.lang.String templateId,
  java.lang.String templateName
)

public TemplateDefinitionInfo(
  java.lang.String id,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry,
  boolean active,
  java.lang.String templateId,
  java.lang.String templateName,
  java.lang.String xml,
  java.utilMap pluginData,
  boolean published
)
```

The following table describes the TemplateDefinitionInfo object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Note: For more information about programming plug-ins, see *Programming BPM Plug-Ins for WebLogic Integration*.

Table B-12 TemplateDefinitionInfo Object Data

Object Data	Constructor Parameter	Get Method
Template definition ID	*id*	`public final java.lang.String getId()`
Effective date for the template definition	*effective*	`public final java.sql.Timestamp getEffective()`
Expiration date for the template definition	*expiry*	`public final java.sql.Timestamp getTemplateDefinition()`
Boolean flag specifying whether the template definition is marked active	*active*	`public final boolean getActive()`
Template ID	*templateId*	`public final java.lang.String getTemplateId()`
Template name	*templateName*	`public final java.lang.String getTemplateName()`
Workflow template definition in XML format	*xml*	`public java.lang.String getContents()`
Plug-in data	*pluginData*	`public java.util.Map getPluginData()` `public com.bea.wlpi.common.plugin.PluginObject getPluginData(java.lang.String pluginName)`

Table B-12 TemplateDefinitionInfo Object Data (Continued)

Object Data	Constructor Parameter	Get Method
Boolean flag specifying whether or not the template definition is published.	*published*	`public boolean isPublished()`

For more information, see the `com.bea.wlpi.common.TemplateDefinitionInfo` javadoc.

TemplateInfo Object

The `com.bea.wlpi.common.TemplateInfo` object maintains object data pertaining to the template.

You can use the following constructors to create a new `TemplateInfo` object:

```
public TemplateInfo(
  java.lang.String id,
  java.lang.String name
)

public TemplateInfo(
  java.lang.String id,
  java.lang.String name,
  java.lang.String xml
)

public TemplateInfo(
  java.lang.String id,
  java.lang.String name,
  java.lang.String xml,
  java.util.Map pluginData,
  boolean published
)
```

The following table describes the `TemplateInfo` object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Table B-13 TemplateInfo Object Data

Object Data	Constructor Parameter	Get Method
Template ID	*id*	`public final java.lang.String getId()`
Template name	*name*	`public final java.lang.String getName()`
Plug-in defined template properties in XML form	*xml*	N/A
Plug-in data	*pluginData*	`public java.util.Map getPluginData()` `public com.bea.wlpi.common.plugin.PluginObject getPluginData(java.lang.String pluginName)`
Boolean flag specifying whether or not the template is published	*published*	`public boolean isPublished()`

For more information, see the `com.bea.wlpi.common.TemplateInfo` javadoc.

UserInfo Object

The `com.bea.wlpi.common.UserInfo` object maintains object data pertaining to the user.

You can use the following constructors to create a new `UserInfo` object:

```
public UserInfo(
  java.lang.String userId
)
```

```
public UserInfo(
  java.lang.String userId,
  java.lang.String eMailAddress,
  java.lang.String defaultOrgId,
  java.lang.String calendarId
)
```

The following table describes the `UserInfo` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Table B-14 UserInfo Object Data

Object Data	Constructor Parameter	Get Method	Set Method
User ID	*userId*	`public final java.lang.String getUserId()`	N/A
User e-mail address	*eMailAddress*	`public final java.lang.String getEMailAddress()`	`public void setEMailAddress(java.lang.String eMailAddress)`
ID of the default organization within which the user is defined	*defaultOrgId*	`public final java.lang.String getDefaultOrgId()`	`public void setDefaultOrgId(java.lang.String defaultOrgId)`
ID of the business calendar to be used	*calendarId*	`public final java.lang.String getCalendarId()`	`public void setCalendarId(java.lang.String calendarId)`

For more information, see the `com.bea.wlpi.common.UserInfo` javadoc.

UserPermissionInfo Object

The `com.bea.wlpi.common.security.UserPermissionInfo` object maintains object data pertaining to user permissions.

You can use the following constructors to create a new `UserPermissionInfo` object:

```
public UserPermissionInfo(
  java.lang.String userId
)

public UserPermissionInfo(
  java.lang.String userId,
  java.util.List roles
)
```

The following table describes the `UserPermissionInfo` object data, the constructor parameters used to define the data, and the get and set methods that can be used to access that data after the object is defined.

Table B-15 RolePermissionInfo Object Data

Object Data	Constructor Parameter	Get Method	Set Method
User ID	*userId*	N/A	N/A
ID of the roles to which the user belongs	*roleId*	`public java.util.List getRoles()`	N/A

For more information, see the `com.bea.wlpi.common.security.UserPermissionInfo` javadoc.

VariableInfo Object

The `com.bea.wlpi.common.VariableInfo` object maintains object data pertaining to the workflow variables.

You can use the following constructors to create a new `VariableInfo` object:

```
public VariableInfo(
  java.lang.String name,
  java.lang.Object value
)

public VariableInfo(
  java.lang.String name,
  java.lang.String type
)
```

The following table describes the `VariableInfo` object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Table B-16 VariableInfo Object Data

Object Data	Constructor Parameter	Get Method
Variable name	*name*	`public final java.lang.String getName()`
Variable value	*value*	`public final java.lang.String getValue()`

Table B-16 VariableInfo Object Data (Continued)

Object Data	Constructor Parameter	Get Method

Variable type	*type*	`public final java.lang.String getType()`
		You can also use the following methods to validate the type:
		`static void validateType(java.lang.String type)`
		`static void validateTypes(java.lang.String[] types)`

For more information, see the `com.bea.wlpi.common.VariableInfo` javadoc.

VersionInfo Object

The `com.bea.wlpi.common.VersionInfo` object maintains object data pertaining to the version number.

You can use the following constructors to create a new `VersionInfo` object:

```
public VersionInfo(
  int majorVersion,
  int minorVersion,
  int volume,
  java.lang.String build,
  java.lang.String name
)

public VersionInfo(
  int majorVersion,
  int minorVersion,
  java.lang.String build,
  java.lang.String name
)
```

```
public VersionInfo(
  java.lang.String version
)
```

The following table describes the `VersionInfo` object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Table B-17 VersionInfo Object Data

Object Data	Constructor Parameter	Get Method
Major version number	*majorVersion*	`public final int getMajorVersion()`
Minor version number	*minorVersion*	`public final int getMinorVersion()`
Volume number	*volume*	`public final int getVolume()`
Build number or name	*build*	`public final java.lang.String getBuild()`
Version name	*name*	`public final java.lang.String getName()`
Default version	N/A	`public static final VersionInfo getDefaultVersion()`
Version number in decimal format containing between two and four integer components (separated by dots) specifying the following data (in the order listed): `majorVersion`, `minorVersion`, `volume`, and `build` values	*version*	N/A

For more information, see the `com.bea.wlpi.common.VersionInfo` javadoc.

XMLEntityInfo Object

Note: The `com.bea.wlpi.common.XMLEntityInfoHelper` object provides an auxiliary class to the `com.bea.eci.repository.helper.XMLEntityInfo` object to support the importing and exporting of object data. For more information, see "XMLEntityInfoHelper Object" on page B-34.

The `com.bea.eci.repository.helper.XMLEntityInfo` object maintains object data pertaining to the entities within the XML repository.

You can use the following constructors to create a new `XMLEntityInfo` object:

```
public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes
)

public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  byte[] content
)

public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  java.sql.Timestamp createdOn,
  java.sql.Timestamp lastModifiedOn
)

public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  java.sql.Timestamp createdOn,
  java.sql.Timestamp lastModifiedOn,
  java.lang.String content
)
```

The following table describes the `XMLEntityInfoHelper` object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Table B-18 XMLEntityInfo Object Data

Object Data	Constructor Parameter	Get Method
Entity type	*type*	`public java.lang.String getType()`
Entity name	*name*	`public java.lang.String getName()`
Entity description	*desc*	`public java.lang.String getDescription()`
Entity notes	*notes*	`public java.lang.String getNotes()`
Entity contents (as array of bytes)	*content*	`public byte[] getContent()`
Entity creation date	*createdOn*	`public java.sql.Timestamp getCreatedOn()`
Entity last modified date	*lastModifiedOn*	`public java.sql.Timestamp getLastModifiedOn()`
Entity contents (as a string)	*content*	`public java.lang.String getContentAsString()`

For more information, see the `com.bea.eci.repository.helper.XMLEntityInfo` javadoc.

XMLEntityInfoHelper Object

The com.bea.wlpi.common.XMLEntityInfoHelper object provides an auxiliary class to the com.bea.eci.repository.helper.XMLEntityInfo object to support the importing and exporting of object data.

You can use the following constructor to create a new XMLEntityInfoHelper object:

```
public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes
)
```

The following table describes the XMLEntityInfoHelper object data, the constructor parameters used to define the data, and the get methods that can be used to access that data after the object is defined.

Table B-19 XMLEntityInfoHelper Object Data

Object Data	Constructor Parameter	Get Method	Set Method
Entity type	*type*	public java.lang.String getEntityType()	N/A
Entity name	*name*	public java.lang.String getName()	N/A
Entity description	*desc*	public java.lang.String getDescription()	N/A
Entity notes	*notes*	public java.lang.String getNotes()	N/A

Table B-19 XMLEntityInfoHelper Object Data (Continued)

Object Data	Constructor Parameter	Get Method	Set Method
Parent folder	N/A	`public java.util.List getParentFolders()`	`public void addParentFolder(com.bea.wlpi.common.RepositoryFolderInfoHelper rfih)`
XML Repository publishable object contents	N/A	`public java.lang.Object getContents()`	N/A
XML Repository publishable entry name	N/A	`public java.lang.String getEntryName()`	N/A
XML Repository publishable owner name	N/A	`public java.lang.String getOwnerName()`	N/A
XML Repository publishables referenced by specified publishable objects	N/A	`public java.util.List getReferencedPublishables(java.util.Map publishables)`	N/A
XML repository publishable object type	N/A	`public int getType()`	N/A

For more information, see the `com.bea.wlpi.common.XMLEntityInfoHelper` javadoc.

C EJB and Java Class Descriptors

Descriptors describe server-side EJBs and Java classes, and are used to pass information to the client. This appendix describes the EJB and Java class descriptor objects and their methods, including the following:

- ClassDescriptor Object
- ClassInvocationDescriptor Object
- EJBDescriptor Object
- EJBInvocationDescriptor Object
- MethodDescriptor Object

For more information about accessing EJB and Java class descriptors, see "Configuring Business Operations" on page 10-1.

ClassDescriptor Object

The `com.bea.wlpi.common.ClassDescriptor` object describes a server-side Java class. The `ClassDescriptor` is used to pass information to the client about the Java classes deployed on the server.

The following constructor can be used to for create a new `ClassDescriptor` object:

```
public ClassDescriptor(java.lang.Class javaClass)
```

The following table describes the `ClassDescriptor` object data, and the associated get methods that can be used to access that data after the object is defined.

Table C-1 ClassDescriptor Object Data

Description	Get Method
Java class method descriptors	`public final java.util.List getMethodDescriptors()`
Java class constructor descriptors	`public final java.lang.Util getConstructorDescriptors()`
Java class modifiers	`public final int getModifiers()`
Boolean flag specifying whether or not the Java class is serializable	`public final boolean isSerializable`

For more information, see the `com.bea.wlpi.common.ClassDescriptor` Javadoc.

ClassInvocationDescriptor Object

The `com.bea.wlpi.common.ClassInvocationDescriptor` object describes and implements the invocation of a Java class method.

The `ClassInvocationDescriptor` is used to instantiate and invoke associated methods on a Java class.

The following constructors can be used to create a new `ClassInvocationDescriptor` object:

```
public ClassInvocationDescriptor(
  java.lang.String description,
  java.lang.String className,
  com.bea.wlpi.common.MethodDescriptor constructorDescriptor,
  java.lang.String[] constructorParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor methodDescriptor,
  java.lang.String[] methodParmDescriptions
)

public ClassInvocationDescriptor(
  java.lang.String description,
  java.lang.String className,
  com.bea.wlpi.common.MethodDescriptor constructorDescriptor,
  java.lang.String[] constructorParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor methodDescriptor,
  java.lang.String[] methodParmDescriptions,
  boolean published
)
```

The following table describes the `ClassInvocationDescriptor` object data, the constructor parameters used to define the data, and the associated get and set methods that can be used to access that data after the object is defined.

Table C-2 ClassInvocationDescriptor Object Data

Object Data	Parameter	Get Method	Set Method
Class invocation descriptor ID	N/A	`public java.lang.String getId()`	`public void setId()`

Table C-2 ClassInvocationDescriptor Object Data (Continued)

Object Data	Parameter	Get Method	Set Method
Class description in human-readable format	`description`	`public java.lang.String getDescription()`	`public void set(...)` (See Note following table)
Fully-qualified class name	`className`	`public java.lang.String getClassName()`	`public void set(...)` (See Note following table)
Class constructor	`constructorDescriptor`	`public com.bea.wlpi.common.MethodDescriptor getConstructor()`	`public void set(...)` (See Note following table)
Class constructor parameter descriptions	`constructorParmDescriptions`	`public java.lang.String[] getConstructorParmsDescriptions()`	`public void set(...)` (See Note following table)
Class method descriptor	`methodDescriptor`	`public com.bea.wlpi.common.MethodDescriptor getMethod()`	`public void set(...)` (See Note following table)
Method parameter descriptions	`methodParmDescriptions`	`public java.lang.String[] getMethodParmsDescriptions()`	`public void set(...)` (See Note following table)
Boolean flag specifying whether or not the information is locked	`published`	`public boolean isPublished()`	`public void set(...)` (See Note following table)

Note: The following methods can be used to set specific Java class descriptor information, as specified in the previous table:

```
public void set(
  java.lang.String description,
  java.lang.String className,
```

```
    com.bea.wlpi.common.MethodDescriptor
      constructorDescriptor,
    java.lang.String[] constructorParmDescriptions,
    com.bea.wlpi.common.MethodDescriptor methodDescriptor,
    java.lang.String[] methodParmDescriptions
  )

  public void set(
    java.lang.String description,
    java.lang.String className,
    com.bea.wlpi.common.MethodDescriptor
      constructorDescriptor,
    java.lang.String[] constructorParmDescriptions,
    com.bea.wlpi.common.MethodDescriptor methodDescriptor,
    java.lang.String[] methodParmDescriptions,
    boolean published
  )
```

For more information, see the `com.bea.wlpi.common.ClassInvocationDescriptor` Javadoc.

EJBDescriptor Object

The `com.bea.wlpi.common.EJBDescriptor` object describes an EJB. The `EJBDescriptor` is used to pass information to the client about EJBs deployed on the server.

The following constructor can be used to create a new `EJBDescriptor` object:

```
public EJBDescriptor()
```

The following table describes the `EJBDescriptor` object data, and the associated get and set methods that can be used to access that data after the object is defined.

Table C-3 EJBDescriptor Object Data

Description	Get Method	Set Method
EJB deployment name	`public java.lang.String getEJBDeploymentName()`	`public void setEJBDeploymentName(java.lang.String ejbDeploymentName)`

Table C-3 EJBDescriptor Object Data (Continued)

Description	Get Method	Set Method
EJB home name	`public java.lang.String getEJBHomeName()`	`public void getEJBHomeName(java.lang.String ejbHomeName)`
EJB home interface method descriptors	`public java.util.List getEJBHomeMethodDescriptors()`	`public void setEJBHomeMethodDescriptors(java.util.List methodDescriptors)`
EJB primary key name	`public java.lang.String getEJBPrimaryKeyName()`	`public void setEJBPrimaryKeyName()`
EJB remote interface method descriptors	`public java.lang.Util getEJBRemoteMethodDescriptors()`	`public void setEJBRemoteMethodDescriptors(java.util.List methodDescriptors)`
EJB remote name	`public java.lang.String getEJBRemoteName()`	`public void setEJBRemoteName(java.lang.String ejbRemoteName)`
Session or Entity EJB	`public boolean isSessionEJB()`	`public void setSessionEJB()`

For more information, see the `com.bea.wlpi.common.EJBDescriptor` Javadoc.

EJBInvocationDescriptor Object

The `com.bea.wlpi.common.EJBInvocationDescriptor` object describes and implements the invocation of an EJB method. The `EJBInvocationDescriptor` is used to obtain an EJB remote interface and invoke associated methods.

The following constructors can be used to create a new `EJBInvocationDescriptor` object:

```
public EJBInvocationDescriptor()
```

```
public EJBInvocationDescriptor(
  java.lang.String description,
  com.bea.wlpi.common.EJBDescriptor beanDescriptor,
  com.bea.wlpi.common.MethodDescriptor homeMethodDescriptor,
  java.lang.String[] homeParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor remoteMethodDescriptor,
  java.lang.String[] remoteParmDescriptions
)

public EJBInvocationDescriptor(
  java.lang.String description,
  com.bea.wlpi.common.EJBDescriptor beanDescriptor,
  com.bea.wlpi.common.MethodDescriptor homeMethodDescriptor,
  java.lang.String[] homeParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor remoteMethodDescriptor,
  java.lang.String[] remoteParmDescriptions,
  boolean published
)
```

The following table describes the `EJBInvocationDescriptor` object data, the constructor parameters used to define the data, and the associated get and set methods that can be used to access that data after the object is defined.

Table C-4 EJBInvocationDescriptor Object Data

Object Data	Parameter	Get Method	Set Method
EJB invocation descriptor id	N/A	`public java.lang.String getId()`	`public void setId()`
EJB description in human-readable format	`description`	`public java.lang.String getDescription()`	`public void set(...)` (See Note following table)
EJB deployment name	N/A	`public java.lang.String getEJBDeploymentName()`	`public void setEJBDeploymentName(java.lang.String` *`deploymentName`*`)`
EJB deployment descriptor	`beanDescriptor`	`public java.lang.String`	`public void set(...)` (See Note following table)

Table C-4 EJBInvocationDescriptor Object Data (Continued)

Object Data	Parameter	Get Method	Set Method
Home interface name	N/A	`public java.lang.String getEJBHomeName()`	`public void setEJBHomeName(java.lang.String` *`ejbHomeName`*`)`
Home interface method descriptor	`homeMethodDescriptor`	`public com.bea.wlpi.common.MethodDescriptor getEJBHomeMethod()`	`public void setEJBHomeMethod(com.bea.wlpi.common.MethodDescriptor` *`ejbHomeMethod`*`)`
Home method parameter descriptions	`homeParmDescriptions`	`public java.lang.String[] getHomeParmsDescriptions()`	`public void set(...)` (See Note following table)
Remote interface name	`N/A`	`public java.lang.String getEJBRemoteName()`	`public void setEJBRemoteName(java.lang.String` *`ejbRemoteName`*`)`
Remote interface method descriptor	`remoteMethodDescriptor`	`public com.bea.wlpi.common.MethodDescriptor getEJBRemoteMethod()`	`public void setEJBRemoteMethod(com.bea.wlpi.common.MethodDescriptor` *`ejbRemoteMethod`*`)`
Remote method parameter descriptions	`remoteParmDescriptions`	`public java.lang.String[] getRemoteParmsDescriptions()`	`public void set(...)` (See Note following table)
Boolean flag specifying whether or not the information is locked	`published`	`public boolean isPublished()`	`public void set(...)` (See Note following table)
Primary key class name	N/A	`public java.lang.String getEJBPrimaryKeyName()`	`public void setEJBPrimaryKeyName(java.lang.String` *`ejbPrimaryKeyName`*`)`

Table C-4 EJBInvocationDescriptor Object Data (Continued)

Object Data	Parameter	Get Method	Set Method
Return type name	N/A	`public java.lang.String getReturnTypeName()`	N/A
Boolean flag specifying whether or not debugging messages are displayed when using the `invoke()` method, as described in "Getting EJB Descriptors" on page 10-7	N/A	`public boolean isVerbose()`	`public void setVerbose (boolean` *`verbose`*`)`
Boolean flag specifying whether or not the desriptor has all required values set	N/A	`public boolean isFullyFormed()`	N/A
Boolean flag specifying whether a session or entity EJB	N/A	`public boolean isSessionEJB()`	N/A

Note: The following methods can be used to set specific EJB invocation descriptor information, as specified in the previous table:

```
public void set(
  java.lang.String description,
  com.bea.wlpi.common.EJBDescriptor beanDescriptor,
  com.bea.wlpi.common.MethodDescriptor homeMethodDescriptor,
  java.lang.String[] homeParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor
    remoteMethodDescriptor,
  java.lang.String[] remoteParmDescriptions
)

public void set(
  java.lang.String description,
  com.bea.wlpi.common.EJBDescriptor beanDescriptor,
  com.bea.wlpi.common.MethodDescriptor homeMethodDescriptor,
  java.lang.String[] homeParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor
    remoteMethodDescriptor,
  java.lang.String[] remoteParmDescriptions,
  boolean published
)
```

For more information, see the `com.bea.wlpi.common.EJBInvocationDescriptor` Javadoc.

MethodDescriptor Object

The `com.bea.wlpi.common.MethodDescriptor` object describes a server-side Java class method. The `MethodDescriptor` is used to pass information to the client about the Java class constructors and/or methods.

The following constructors can be used to create a new `MethodDescriptor` object:

```
public MethodDescriptor(java.lang.reflect.Constructor constructor)
```

```
public MethodDescriptor(java.lang.reflect.Method method)
```

The following table describes the `MethodDescriptor` object data, and the associated get methods that can be used to access that data after the object is defined.

Table C-5 MethodDescriptor Object Data

Description	Get Method
Method name	`public final java.lang.String getMethodName()`
Fully-qualified exception class names	`public final java.lang.String[] getExceptionTypes()`
Fully-qualified parameter class or interface names	`public final java.lang.String[] getParameterTypes()`
Return type	`public final java.lang.String getReturnType()`
Boolean flag specifying whether or not the method is abstract	`public final boolean isAbstract()`
Boolean flag specifying whether or not the method is final	`public final boolean isFinal()`
Boolean flag specifying whether or not the method is native	`public final boolean isNative()`
Boolean flag specifying whether or not the method or constructor is private	`public final boolean isPrivate()`

Table C-5 MethodDescriptor Object Data (Continued)

Description	Get Method
Boolean flag specifying whether or not the method or constructor is protected	`public final boolean isProtected()`
Boolean flag specifying whether or not the method is static	`public final boolean isStatic()`
Boolean flag specifying whether or not the method is strict	`public final boolean isStrict()`
Boolean flag specifying whether or not the method is synchronized	`public final boolean isSynchronized()`

For more information, see the `com.bea.wlpi.common.MethodDescriptor` Javadoc.

D Customizing Studio and Worklist Logos and Text

You can customize the logos and text for the BEA WebLogic Integration Studio and Worklist clients.

Note: The Worklist client application is being deprecated as of this release of WebLogic Integration. For information about the features that are replacing it, see the *BEA WebLogic Integration Release Notes*.

To customize the existing logo images for the Studio and Worklist, replace the following file in the `wlpi-studio.jar` and `wlpi-worklist.jar` files, respectively, with a custom logo:

`com/bea/wlpi/client/common/images/logo.gif`

To customize the existing Studio and Worklist product and server message text, replace the following files, as required, with custom text files:

- `com/bea/wlpi/client/common/text/text.properties` in the `wlpi-studio.jar` and `wlpi-worklist.jar` files, respectively (product text)
- `com/bea/wlpi/client/studio/text/text.properties` in the `wlpi-studio.jar` (Studio product text)
- `com/bea/wlpi/client/worklist/text/text.properties` in the `wlpi-worklist.jar` (Worklist product text)

- `com/bea/wlpi/common/message.properties` in the `wlpi-studio.jar`, `wlpi-worklist.jar`, and `wlpi-aux.jar` (server message text)

E Database Schema

The following table describes the BPM database schema entities.

Table E-1 BPM Database Schema Entities

Entity Name	Attribute Name	Attribute Definitions	
		Column Data Type	Column Name
ACLENTRIES	PRINCIPAL	VARCHAR(254)	A_PRINCIPAL
	NAME	VARCHAR(254)	A_NAME
	PERMISSION	VARCHAR(254)	A_PERMISSION
ADDRESSEDMESSAGE	KEYVALUE	VARCHAR(254)	KEYVALUE
	EXPIRY	DATE	EXPIRY
	VALIDATED	BOOLEAN	Boolean
	MESSAGE	LARGE BINARY	MESSAGE
	MESSAGEID	VARCHAR(20)	MESSAGEID
BUSINESSCALENDAR	TIMEZONE	VARCHAR(50)	TIMEZONE
	NAME	VARCHAR(50)	NAME
	DATA	LARGE BINARY	DATA
	BUSINESSCALENDARID	VARCHAR(20)	ID

Table E-1 BPM Database Schema Entities (Continued)

Entity Name	Attribute Name	Attribute Definitions	
		Column Data Type	Column Name
BUSINESSOPERATION	DESCRIPTION	VARCHAR(50)	DESCRIPTION
	LASTCHANGED	DATE	Datetime
	DEFINITION	LARGE BINARY	DEFINITION
	BUSINESSOPERATION ID	VARCHAR(20)	ID
EVENTKEY	PLUGIN	VARCHAR(50)	PLUGIN
	EXPRESSION	TEXT	EXPRESSION
	LASTCHANGED	DATE	Datetime
	EVENTDESCRIPTOR	VARCHAR(192)	EVENTDESCRIPTOR
	FIELDID	INTEGER	FIELDID
	CONTENTTYPE	VARCHAR(128)	CONTENTTYPE

Table E-1 BPM Database Schema Entities (Continued)

Entity Name	Attribute Name	Attribute Definitions	
		Column Data Type	Column Name
EVENTWATCH	KEYVALUE	VARCHAR(254)	KEYVALUE
	CONDITION	TEXT	CONDITION
	PLUGIN	VARCHAR(50)	PLUGIN
	EVENTDESCRIPTOR	VARCHAR(192)	EVENTDESCRIPTOR
	CONTENTTYPE	VARCHAR(128)	CONTENTTYPE
	VALIDATED	BOOLEAN	Boolean
	TEMPLATEID	VARCHAR(20)	ID
	NODEID	VARCHAR(20)	NODEID
	FIELDID	INTEGER	FIELDID
	TEMPLATEDEFINITIONID	VARCHAR(20)	TEMPLATEDEFINITIONID
	STARTORG	TEXT	STARTORG
	INSTANCEID	VARCHAR(20)	INSTANCEID
GROUPMEMBER	GROUPID	VARCHAR(20)	GROUPID
	GROUPMEMBERID	VARCHAR(20)	GROUPMEMBERID
IDGENERATOR	MAXKEY	VARCHAR(20)	MAXKEY
	TABLENAME	VARCHAR(30)	TABLENAME

Table E-1 BPM Database Schema Entities (Continued)

Entity Name	Attribute Name	Attribute Definitions	
		Column Data Type	Column Name
INSTANCE	STARTED	DATE	STARTED
	COMPLETED	DATE	COMPLETED
	INITIATOR	VARCHAR(50)	INITIATOR
	TEMPLATEDEFINITIONID	VARCHAR(20)	ID
	NAME	VARCHAR(50)	NAME
	ORGID	VARCHAR(20)	ORG
	TEMPLATEID	VARCHAR(20)	ID
	PARENTID	VARCHAR(20)	PARENTID
	DATA	LARGE BINARY	DATA
	INSTANCEID	VARCHAR(20)	ID
	IDSTRING	VARCHAR(50)	IDSTRING
	STATE	INTEGER	STATE
	ATTACHMENTS	INTEGER	ATTACHMENTS
	WORKFLOWCOMMENT	VARCHAR(50)	WORKFLOWCOMMENT
MESSAGERECIPIENT	RECIPIENTTYPE	INTEGER	RECIPIENTTYPE
	CREATED	DATE	Datetime
	RECIPIENTID	VARCHAR(20)	RECIPIENTID
	MESSAGEID	VARCHAR(20)	MESSAGEID
ORG	DESCRIPTION	VARCHAR(254)	DESCRIPTION
	BUSINESSCALENDARID	VARCHAR(20)	ID
	ORGID	VARCHAR(20)	ORGID

Table E-1 BPM Database Schema Entities (Continued)

Entity Name	Attribute Name	Attribute Definitions	
		Column Data Type	Column Name
ORGMEMBER	USERID	VARCHAR(20)	U_NAME
	ORGID	VARCHAR(20)	ORGID
PLUGIN	CONFIG	TEXT	CONFIG
	STARTMODE	INTEGER	STARTMODE
	VERSION	VARCHAR(20)	VERSION
	NAME	VARCHAR(50)	NAME
REROUTE	ROUTETYPE	INTEGER	ROUTETYPE
	EFFECTIVE	DATE	EFFECTIVE
	ROUTETO	VARCHAR(50)	ROUTETO
	ROUTEFROM	VARCHAR(50)	ROUTEFROM
	REROUTEID	VARCHAR(20)	ID
	EXPIRY	DATE	EXPIRY
	ORGID	VARCHAR(20)	ORGID
STATS	STARTED	DATE	STARTED
	COMPLETED	DATE	COMPLETED
	TASKID	VARCHAR(20)	TASKID
	TEMPLATEDEFINITIONID	VARCHAR(20)	TEMPLATEDEFINITIONID
	ISROLE	BOOLEAN	ISROLE
	ORGID	VARCHAR(20)	ORG
	ASSIGNEE	VARCHAR(20)	ASSIGNEE

Table E-1 BPM Database Schema Entities (Continued)

Entity Name	Attribute Name	Attribute Definitions	
		Column Data Type	Column Name
TASK	MODIFIABLE	BOOLEAN	MODIFIABLE
	UNMARKDONE	BOOLEAN	UNMARKDONE
	REASSIGNMENT	BOOLEAN	REASSIGNMENT
	STARTED	DATE	STARTED
	COMPLETED	DATE	COMPLETED
	NAME	VARCHAR(50)	NAME
	DOITIFDONE	BOOLEAN	DOITIFDONE
	PRIORITY	INTEGER	PRIORITY
	DONEWITHOUTDOIT	BOOLEAN	DONEWITHOUTDOIT
	TEMPLATEID	VARCHAR(20)	ID
	TEMPLATEDEFINITIONID	VARCHAR(20)	ID
	WORKFLOWID	VARCHAR(50)	WORKFLOWID
	INSTANCEID	VARCHAR(20)	Id
	TASKID	VARCHAR(20)	TASKID
	TASKCOMMENT	VARCHAR(50)	TASKCOMMENT
	ASSIGNEE	VARCHAR(20)	ASSIGNEE
	ORGID	VARCHAR(20)	ORGID
	WORKFLOW	VARCHAR(50)	WORKFLOW
	ISROLE	BOOLEAN	ISROLE
	DUE	DATE	DUE

Table E-1 BPM Database Schema Entities (Continued)

Entity Name	Attribute Name	Attribute Definitions	
		Column Data Type	Column Name
TEMPLATE	NAME	VARCHAR(50)	NAME
	LASTCHANGED	DATE	Datetime
	DATA	LARGE BINARY	DATA
	TEMPLATEID	VARCHAR(20)	ID
TEMPLATEDEFINITION	ACTIVE	BOOLEAN	ACTIVE
	MANUAL	BOOLEAN	MANUAL
	EXPIRY	DATE	EXPIRY
	EFFECTIVE	DATE	EFFECTIVE
	LASTCHANGED	DATE	Datetime
	TEMPLATEDEFINITIONID	VARCHAR(20)	ID
	CALLED	BOOLEAN	CALLED
	TEMPLATEID	VARCHAR(20)	ID
	DATA	LARGE BINARY	DATA
TEMPLATEORG	TEMPLATEID	VARCHAR(20)	ID
	ORGID	VARCHAR(20)	ORGID
TIMEDEVENT	INSTANCEID	VARCHAR(20)	ID
	TEMPLATEID	VARCHAR(20)	ID
	TRIGGERTIME	DATE	TRIGGERTIME
	TEMPLATEDEFINITIONID	VARCHAR(20)	ID
	NODEID	VARCHAR(20)	NODEID
	STARTORG	TEXT	STARTORG

Table E-1 BPM Database Schema Entities (Continued)

Entity Name	Attribute Name	Attribute Definitions	
		Column Data Type	Column Name
TRANSACTIONINFO	RESULTOBJECT	LARGE BINARY	RESULTOBJECT
	COMPLETEDTIME	DATE	COMPLETEDTIME
	TRANSACTIONID	VARCHAR(40)	TRANSACTIONID
USERMEMBER	USERID	VARCHAR(20)	U_NAME
	GROUPID	VARCHAR(20)	GROUPID
WLSGROUP	GROUPID	VARCHAR(20)	GROUPID
WLSUSER	PASSWORD	VARCHAR(254)	U_PASSWORD
	USERID	VARCHAR(20)	U_NAME
WORKFLOWROLE	BUSINESSCALENDARID	VARCHAR(20)	ID
	GROUPID	VARCHAR(20)	GROUPID
	DESCRIPTION	VARCHAR(20)	DESCRIPTION
	ORGID	VARCHAR(20)	ORGID
	ROLEID	VARCHAR(20)	ROLEID
WORKFLOWUSER	FULL_NAME	VARCHAR(254)	FULL_NAME
	DEFAULTORGID	VARCHAR(20)	DEFAULTORGID
	EMAILADDRESS	VARCHAR(100)	EMAILADDRESS
	BUSINESSCALENDARID	VARCHAR(20)	ID
	USERID	VARCHAR(20)	U_NAME

Index

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X