



# BEA WebLogic Server® Process Edition

## Overview



# Contents

## 1. Introduction to WebLogic Server Process Edition

Key Features .....	1-2
Installation .....	1-7
Features .....	1-7
Components .....	1-12

## 2. Controls: Service Enablement

Integration Controls .....	2-2
File Control .....	2-2
Email Control .....	2-3
WLI JMS Control .....	2-3
Service Broker Control .....	2-3
HTTP Control .....	2-4
MQSeries Control .....	2-5
Tibco RV Control .....	2-6
XML MetaData Cache Control .....	2-6
Other Available Application Controls .....	2-7
Beehive Controls .....	2-7
JDBC Control .....	2-8
Timer Control .....	2-8
Service Control .....	2-9
EJB Control .....	2-9

JMS Control . . . . .	2-10
Using Asynchronous Interfaces . . . . .	2-10
Overview of Asynchrony . . . . .	2-11
Using Asynchrony. . . . .	2-11
Using Callbacks . . . . .	2-12
Using Polling . . . . .	2-12
Designing Asynchronous Interfaces . . . . .	2-13
Using Polling as an Alternative to Callbacks . . . . .	2-13
Designing Robust Asynchronous Interfaces . . . . .	2-15
Conversations . . . . .	2-17
Overview of Conversations. . . . .	2-17
Correlating Messages with a Unique Identifier . . . . .	2-18
Implementing Conversations . . . . .	2-18
Understanding Conversation Context . . . . .	2-18
Designing a Web Service to Use Conversations . . . . .	2-19

### 3. Business Process Management: Process Driven Services

Business Process Management Overview . . . . .	3-1
Business Process Management Features . . . . .	3-2
Web Services Available as Business Process Resources . . . . .	3-3
Building a Business Process. . . . .	3-4
Stateful and Stateless Processes . . . . .	3-5
Stateless Processes . . . . .	3-5
Stateful Processes . . . . .	3-6
Determining if your Business Process is Stateful or Stateless . . . . .	3-6

### 4. Data Transformation

Data Transformation Overview . . . . .	4-1
--	-----

Data Transformation Features ..... 4-3

## 5. Process Monitoring and Management

Process Configuration ..... 5-2

- Managing Process Tracking Data ..... 5-2
- Process Security Policies ..... 5-3
- Service Level Agreements ..... 5-4
- Process Versions ..... 5-5
- Dynamic Controls ..... 5-5

Process Instance Monitoring ..... 5-6

## Index



# Introduction to WebLogic Server Process Edition

WebLogic Server Process Edition provides you with the technologies and tools you need to effectively service-enable your existing resources, create composite services using process-driven development, and extend these composite services to interact with other applications and technologies. Each of these phases allows you to take distinct measurable steps that are effective for both project and enterprise level architectures.

The following sections introduce the key features of WebLogic Server Process Edition, detail the WebLogic options available when you purchase a WebLogic Server Process Edition license and provide a brief overview of WebLogic Server Process Edition:

- [Key Features](#)
- [Installation](#)
- [Features](#)
- [Features](#)
- [Components](#)

## Key Features

The following table details the key features of WebLogic Server Process Edition and outlines the benefits that these features provide.

**Table 1-1 WebLogic Server Process Edition Key Features**

Features	Benefits
<b>Service Enable Existing Resources</b>	
<p>Extensible Controls Architecture built on the Apache Beehive Control framework.</p> <p>For more information, see <a href="#">Working with Beehive Controls</a> and <a href="#">Controls: Getting Started</a> in Apache Beehive documentation.</p>	<ul style="list-style-type: none"> <li>• Access resources such as databases or other resources.</li> <li>• Encapsulate logic for reuse in other applications.</li> <li>• Modularize logic you want to keep separate from other application code.</li> <li>• Use Java 5 annotations to set control properties.</li> <li>• Provide mechanism to encapsulate business logic and to access enterprise resources such as databases, web services, EJBs, JMS message queues, and legacy applications using system controls.</li> <li>• Provide mechanism to fully customize access to a resource or encapsulate some application functionality using custom controls. You can design a custom control to do any task in an application.</li> </ul>
<p>Resource Connectivity</p> <p>For more information, see <a href="#">Using Built-In Java Controls</a> and <a href="#">Developing Adapters</a>.</p>	<ul style="list-style-type: none"> <li>• Provide out-of-the-box connectivity to Databases, EJBs, JMS, Web services, MQ Series, .NET, Tuxedo, file systems, and e-mail.</li> <li>• Provide additional J2CA-based pre-built adapters to leading enterprise applications and technologies.</li> </ul>

**Table 1-1 WebLogic Server Process Edition Key Features (Continued)**

Features	Benefits
<b>Orchestrate Services using Process-driven Development</b>	
Process Modeling for Composite Services For more information, see <a href="#">Guide to Building Business Processes</a> .	<ul style="list-style-type: none"> <li>• Build, view, and change process models, with drag-and-drop graphical construction of complex composite service scenarios and two-way editing between Design and Source views.</li> <li>• Execute sophisticated Web service orchestration scenarios using composite process driven services.</li> <li>• Automatically accessible as a Web service, can easily invoke and respond to other Web services, and can be exported to Business Process Execution Language (BPEL).</li> <li>• Built-in support for asynchronous communication, lifecycle events, security, transactions, etc.</li> <li>• Supports proven process operations such as Synchronous and Asynchronous Communication, Branching, Nesting, Looping, Parallelism, Grouping, and Exception Handling.</li> <li>• Based on Process Definition for Java (PD4J-JSR 207), providing a seamless convergence between graphical representation of process flows in XML and Java for logic execution.</li> </ul>

**Table 1-1 WebLogic Server Process Edition Key Features (Continued)**

Features	Benefits
<p>Process Monitoring and Management</p> <p>For more information, see <a href="#">Process Configuration</a> and <a href="#">Process Instance Monitoring</a> in <i>Using the WebLogic Integration Administration Console</i>.</p>	<ul style="list-style-type: none"> <li>• View process type information and locate specific processes for configuration.</li> <li>• View or update process type properties, the security policies for a process, and the selectors used to dynamically set control attributes for a Process or Service Broker control.</li> <li>• Activate or deactivate a non-versioned process.</li> <li>• Configure the activation time for a newly deployed process version, or rollback to a previous version.</li> <li>• View an interactive or printable process type graph.</li> <li>• View summary statistics that reflect system health and the summary or detailed status for selected instances.</li> <li>• Terminate or suspend instances, resume previously suspended instances, or unfreeze frozen instances.</li> </ul>
<b>Build for Integration</b>	
<p>Unified Development Environment and Run-Time Framework</p> <p>For more information, see <a href="#">Workshop for WebLogic Platform Programmer's Guide</a>.</p>	<ul style="list-style-type: none"> <li>• Use a single tool, runtime framework, and programming model to orchestrate all enterprise services including custom Java Controls, Applications, and Web services.</li> <li>• Easily produce and manage custom-fit enterprise portals through interoperability with Web applications and BEA WebLogic Portal.</li> <li>• Switch from writing code to immediately testing applications with one-button deployment, integrated debugger, and automated test harness.</li> </ul>

**Table 1-1 WebLogic Server Process Edition Key Features (Continued)**

Features	Benefits
<p>Web Services</p> <p>For more information, see <a href="#">Building Web Services with Workshop</a> for <i>Workshop for WebLogic Platform Programmer's Guide</i>.</p>	<ul style="list-style-type: none"> <li>• Automatic support for state management, message correlation, and conversation lifecycles to handle asynchronous interaction models.</li> <li>• Support for loose-coupling with a visual tool enabling a standard and flexible technique to transform between XML and Java.</li> <li>• Employ higher-level, coarse-grained messages to enhance scalability and usability.</li> <li>• Secure applications with transport-level security and message-based security (authentication, signature, and/or encryption).</li> <li>• Extensibility and integration via SOAP interceptor mode.</li> </ul>
<p>Data Transformation</p> <p>For more information, see <a href="#">Guide to Data Transformation</a>.</p>	<ul style="list-style-type: none"> <li>• Transformations packaged as controls that can be treated as resources and reused across multiple processes and integration solutions</li> <li>• Transformation of data between any of the following input-output data types: XML Data, Non-XML Data, Java Primitives, and Java classes</li> <li>• Gain a convenient Java object-based view of XML data without losing access to the richness of the native XML structure through XML Beans.</li> <li>• Support latest industry standards such as XSLT and XQuery for data transformation.</li> </ul>

**Table 1-1 WebLogic Server Process Edition Key Features (Continued)**

Features	Benefits
<p>Standards Support For more information, see <a href="#">Standards</a>.</p>	<ul style="list-style-type: none"> <li>• Support latest standards including SOAP 1.2, WSDL 1.2, UDDI 2.0, and WS-Security.</li> <li>• Annotations based on JSR-175 standards.</li> <li>• XML productivity tools based on emerging XML Beans, XML Schema, and XQuery standards.</li> <li>• Comprehensive support for Enterprise Java Bean (EJB) development and deployment, including design views for session and entity beans.</li> <li>• Implement Web applications based on the Apache open-source Struts framework.</li> <li>• Support for BPEL export for cross platform interoperability.</li> </ul>
<hr/> <p><b>An Enterprise Class Foundation</b></p> <hr/>	
<p>Industry Leading Application Server</p>	<ul style="list-style-type: none"> <li>• Robust J2EE-certified application server functionality provides the essential capabilities and underlying infrastructure for enterprise-class applications: Security, system management and monitoring, failover via clustering, performance, scalability, transactions.</li> <li>• Native clustering that is completely transparent to the application.</li> <li>• Proven scalability and reliability in the most demanding enterprise environments.</li> <li>• Ensures that users experience no service interruption in mission critical applications. Use of in-memory replication to scale large clusters while ensuring high availability.</li> <li>• Allows reuse of different types of connections required to establish communications with clients, to databases, application adaptors, and message factories.</li> <li>• Delivers proven performance, scalability, flexibility, clustering, security, transaction management, and reliability to meet enterprise needs with confidence.</li> </ul> <hr/>

# Installation

There is no separate installation procedure for WebLogic Server Process Edition. The availability of features is dependent on the WebLogic Platform product components and licenses installed.

When you install WebLogic Server, WebLogic Workshop, and WebLogic Integration (the components of WebLogic Platform on which WebLogic Server Process Edition functionality depends), a development license is included that enables a complete development framework for the installed components.

To enable the WebLogic Server Process Edition capabilities for production, you must have a WebLogic Server Process Edition production license installed.

In the development environment, the features available in the WebLogic Workshop IDE are dependent on:

- The WebLogic Platform components installed.
- The type of domain in which the server is running.

If you have installed WebLogic Integration and are running the server in a WebLogic Integration domain, the WebLogic Workshop IDE not only provides access to the controls and other functionality supported by WebLogic Server Process Edition, it also provides access to WebLogic Integration features that are not supported by the WebLogic Server Process Edition production license. To ensure that no unsupported features are used, you must restrict the development environment by using only the features as described in the following section.

# Features

The WebLogic Server Process Edition production license provides business process management, data transformation, and process monitoring capabilities that are available in WebLogic Integration, in addition to all the capabilities of WebLogic Server Premium. WebLogic Server Process Edition does not include the Message Broker, WorkList, application integration framework, and trading partner integration capabilities that are available in WebLogic Integration. (For a description of the various offerings, see [Licensing](#).)

The following table details the differences between WebLogic Server Process Edition and WebLogic Integration.

**Table 1-2 Feature Comparison Matrix**

<b>Category</b>	<b>Feature</b>	<b>WebLogic Server Process Edition</b>	<b>WebLogic Integration</b>
<b>Business Processes</b>	Stateless process modeling and automation	Y	Y
	Stateful process modeling and automation	Y	Y
	Web application-based human interaction	Y	Y
	Task-based human interaction (Worklist)	N	Y
	Business calendars	N	Y
<b>Data Transformation</b>	XML and Java data transformation	Y	Y
	Non-XML data Transformation	Y	Y
	Format Builder for Non-XML data transformation	Y	Y
<b>Application Integration</b>	Application connectivity via iWay 5.5 Adapters	Y	Y
	Application connectivity via BEA WebLogic Adapters	N	Y
	Application View Design Console	N	Y
	Adapter development kit	N	Y
	RDBMS adapter	N	Y

Table 1-2 Feature Comparison Matrix (Continued)

Category	Feature	WebLogic Server Process Edition	WebLogic Integration
<b>Message Broker</b>	Inter-process pub/sub	N	Y
	Event generators for external events	N	Y
<b>BPEL Import/Export</b>	Supports the BPEL 1.1 specification	N	Y
<b>Event Generators</b>	File	N	Y
	Email	N	Y
	HTTP	N	Y
	JMS	N	Y
	MQ Series	N	Y
	TIBCO	N	Y
	Timer	N	Y
<b>Trading Partner Integration</b>	B2B protocols (e.g., ebXML, RosettaNet)	N	Y
	Trading partner management	N	Y

**Table 1-2 Feature Comparison Matrix (Continued)**

<b>Category</b>	<b>Feature</b>	<b>WebLogic Server Process Edition</b>	<b>WebLogic Integration</b>
<b>WebLogic Integration Administration Console Modules</b>	System Configuration	Y	Y
	User Management	Y	Y
	Process Configuration	Y	Y
	Process Monitoring	Y	Y
	Message Broker Management	N	Y
	Event Generators	N	Y
	Worklist Administration	N	Y
	Application Integration Management	N	Y
	Trading Partner Management	N	Y
	Business Calendar Configuration	N	Y
<b>Integration Controls</b>	File control	Y	Y

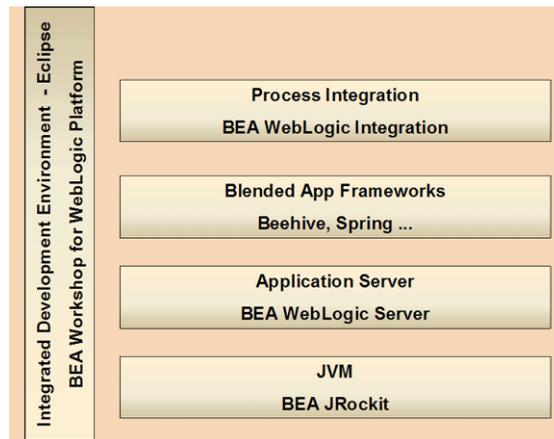
Table 1-2 Feature Comparison Matrix (Continued)

Category	Feature	WebLogic Server Process Edition	WebLogic Integration
	Email control	Y	Y
	Http Control	Y	Y
	WLI JMS control	Y	Y
	Service Broker control	Y	Y
	Dynamic Transformation control	Y	Y
	Process control	Y	Y
	MQSeries control	Y	Y
	Message Broker controls	N	Y
	TPM control	N	Y
	Application View control	N	Y
	Worklist Control	N	Y
	XML Metadata Cache control	Y	Y
	TIBCO Rendezvous control	Y	Y
	RosettaNet control	N	Y
	ebXML control	N	Y

## Components

The WebLogic Platform contains numerous component products, as shown in the following figure. You can use a combination of these components, or just use an individual component, to build an application.

**Figure 1-1 WebLogic Platform Component Products**



BEA WebLogic Server, the industry-leading J2EE application server, provides the critical infrastructure needed to develop integrated solutions, including security, transaction management, fault tolerance, persistence, and clustering.

Leveraging WebLogic Server as the underlying deployment environment, WebLogic Integration uses Web services to integrate distributed systems inside and outside an organization and utilizes the BEA Workshop for WebLogic Platform framework to simplify application development.

WebLogic Integration works seamlessly in the BEA Workshop for WebLogic Platform environment. WebLogic Integration uses the power of BEA Workshop for WebLogic Platform to provide a robust set of tools to develop and extend integration applications. WebLogic Integration provides graphical tools you use to edit your business processes, and human interaction task plans. These tools are available in the same BEA Workshop for WebLogic Platform environment as your controls, Web services, and portal-building tools

WebLogic Server Process Edition contains an IDE framework based on Eclipse 3.1.2.

For more information on the product components displayed in the preceding figure, see the [WebLogic Platform 9.2](#) documentation.

# Controls: Service Enablement

WebLogic Server Process Edition provides a set of out-of-the box controls that enable you to start integration projects with a portfolio of resources. WebLogic Server Process Edition integration controls provide easy access to enterprise resources like databases, file systems, etc., from within your application. WebLogic Integration applications also have access to external resources through the WebLogic Workshop high-level controls. BEA Workshop for WebLogic Platform provides a consistent mechanism for interacting with resources across all BEA Workshop for WebLogic Platform, WebLogic Integration, and WebLogic Portal components. The control handles the work of connecting to the enterprise resource for you, so that you can focus on the logic of your business process.

WebLogic Integration controls are now based on the Apache Beehive open source framework (<http://beehive.apache.org/>). Beehive is a lightweight component framework that helps programmers encapsulate application logic and leverage metadata annotations into their programming model. Developers can create custom controls or use the pre-built system controls.

The following sections describe the WebLogic Server Process Edition integration controls and the other available WebLogic controls in more detail, before introducing asynchronous interfaces and conversations:

- [Integration Controls](#)
- [Other Available Application Controls](#)
- [Using Asynchronous Interfaces](#)
- [Conversations](#)

## Integration Controls

The following integration controls are described in this section:

- [File Control](#)
- [Email Control](#)
- [WLI JMS Control](#)
- [Service Broker Control](#)
- [HTTP Control](#)
- [MQSeries Control](#)
- [Tibco RV Control](#)
- [XML MetaData Cache Control](#)

### File Control

A File control is used to perform operations on a file. It allows business processes to read, write, or append to a file in a file system. In addition, the File control supports file manipulation operations such as copy, rename, and delete. You can also retrieve a list of the files stored in a specific directory. The files can be one of the following types: `XmlObject`, `RawData` (binary), or `String`.

The methods available to a File control depend on the type of data contained in the file. For a file of type `String` or `XmlObject`, you can specify the character set encoding. When processing a file, you can specify the delimiter for a file as the record size. If no delimiter is specified, the file is processed one line at a time. In the case of files of type `String`, you can also specify number of bytes or any character as a delimiter.

Normally, you configure a separate File control for every individual file you want to manipulate. You can specify File control settings in several different ways. You can set the File control's properties in Design view or you can call the `setProperty` method of the `FileControl` interface. You can change File control configuration properties dynamically. To get the current property settings, use the `getProperty()` method.

You can also use the [ControlContext interface](#) to access a control's properties at run time and to handle control events. Property values set by a developer who is using the control are stored as annotations on the control's declaration in a JWS, JSP, or JPD file, or as annotations on its interface, callback, or method declarations in a Control Definition file.

For more information, see [File Control](#) in *Using Integration Controls*.

## Email Control

The Email control enables WebLogic Server Process Edition business processes to send e-mail to a specific destination. The body of the e-mail message can be text (plain, HTML, or XML) or an XML object. The Email control is customizable, which allows you to specify e-mail transmission properties in an annotation, or use dynamic properties passed as an XML variable. You can use the Email control to send a variety of content types and various combinations of body and attachments.

When you add an Email control to your business process, you can use an existing Email control definition file or create a new one.

For more information, see [Email Control](#) in *Using Integration Controls*.

## WLI JMS Control

JMS (Java Message Service) is a Java API for communicating with messaging systems, which are often packaged as products known as Message-Oriented Middleware (MOMs). WebLogic Server includes built-in messaging capabilities via WebLogic JMS, but can also work with third-party MOMs. Messaging systems are often used in enterprise applications to communicate with legacy systems, or for communication between business components running in different environments or on different hosts.

The WLI JMS control enables WebLogic Workshop business processes to easily interact with messaging systems that provide a JMS implementation. A specific WLI JMS control is associated with particular facilities of the messaging system. Once a WLI JMS control is defined, business processes may use it like any other WebLogic Workshop control.

The WLI JMS control is an extension of the JMS control, providing additional features such as RawData message type support, dynamic property configuration, and the ability to control whether to start a new transaction or remain within the calling transaction.

For more information, see [WLI JMS Control](#) in *Using Integration Controls* in the WebLogic Workshop Help.

## Service Broker Control

The Service Broker control allows a business process to send requests to, and receive callbacks from, another business process, a Web service, or a Web service or business process defined in a Web Service Description Language (WSDL) file. The Service Broker control lets you

dynamically set control attributes. This allows you to reconfigure control attributes without having to redeploy the application.

A remote Web service or business process is accessed using Web services and is described in a WSDL file. A WSDL file describes the methods and callbacks that a Web service implements, including method names, parameters, and return types. You can generate a WSDL file for any business process by right clicking on a JPD file in the Application pane and choosing **Generate WSDL File**.

**Note:** The Service Broker Control has been deprecated in the 9.2 release. It will not be supported and will be removed in the next major release of WebLogic Server Process Edition. The control has been included only for backward compatibility with WebLogic Server Process Edition 8.1. Use of this control is recommended only when communicating with WebLogic Server Process Edition 8.x JPDs.

The Service Broker Control in 9.2 does not support the latest version of web services standards, such as WS-Addressing, WS-Security, and SOAP 1.2. In order to take advantage of the latest web services standards, please use the web service control delivered with BEA Workshop for WebLogic Platform.

For more information, see [Service Broker Control](#) in *Using Integration Controls*.

## HTTP Control

The Hyper Text Transfer Protocol (HTTP) control is built using the features of the WebLogic Platform control architecture. This control's purpose is to provide outgoing HTTP access to BEA Workshop for WebLogic Platform clients. The Http control complements the other controls provided in WebLogic Integration and can be used with BEA Workshop for WebLogic Platform and business processes to work with HTTP requests and process responses. The HTTP control source file is a wrapper around the Jakarta Commons [HTTP Client](#) package. The HTTP control conforms to HTTP/1.1 specific features.

The Http control enables WebLogic Workshop and business processes to work with HTTP requests and send responses to a specific URL. The Http control supports two HTTP modes for data transfer, namely HTTP GET and HTTP POST. By using the GET mode, you can send your business data along with the URL. By using the POST mode, you can also send Binary, XML, and string documents. You can specify HTTP properties in an annotation, or pass dynamic properties via an XML variable.

Using the Http control, you can send an HTTP or HTTPS (Secure HTTP) request to a URL and receive specific HTTP response header and body data, as follows:

- Send Business data using HTTP Get and receive the HTTP response code and the message corresponding to the response code in an XML document.
- Set HTTP header values for the HTTP Post mode.
- Send Binary, XML, and String type data using HTTP Post and receive the HTTP response code and the message corresponding to the response code in an XML document.
- Configure cookies for both the HTTP Get and HTTP Post modes.
- Communicate via a secure HTTP (HTTPS) connection with both client-side and server-side authentication enabled.
- Use a proxy server for sending an HTTP or HTTPS request.
- Receive response headers in an XML document conforming to a pre-defined schema.
- Receive response body data of type Binary, XML or String.
- Receive cookies in an XML document conforming to a pre-defined schema.

For more information, see [Http Control](#) in *Using Integration Controls*.

## MQSeries Control

MQSeries is a middleware product from IBM that runs on multiple platforms. It enables message transfer between applications; the sending application PUTs a message on a Queue, and the receiving application GETs the message from the Queue. The sending and receiving applications do not have to be on the same platform, and can execute at different times. MQSeries manages all the storage, logging and communications details required to guarantee delivery of the message to the destination queue.

MQ Series is a messaging service queue provided by IBM that enables message transfer between applications. The sending application puts a message on a Queue, and the receiving application gets the message from the Queue.

The MQ Series control enables business processes to send and receive messages using MQ Series queues. Using the MQ Series control, you can send and receive Binary, XML, and String messages. You can specify MQ Series control properties while configuring the MQ Series control or dynamically at run-time. By default, the MQ Series control handles transactions implicitly for each PUT and GET method individually, without having to set an explicit transaction boundary. However, transaction boundaries can also be set explicitly.

Using SSL, you can enable both one way authentication (server-side) and two-way authentication (client-side).

For more information, see [MQSeries Control](#) in *Using Integration Controls*.

## Disclaimer

Use of the MQSeries control and event generator with BEA WebLogic Integration in no manner confers or grants the right to use MQSeries control including "dynamic libraries". In order to use such IBM products, the user of the MQSeries control and event generator must obtain a valid license from IBM.

## Tibco RV Control

TIBCO® Rendezvous™ (a product from TIBCO: [www.tibco.com](http://www.tibco.com)) enables exchange of data across applications running on distributed platforms. TIBCO Rendezvous (TIBCO RV) Control in WebLogic Server Process Edition enables seamless connection to, and transfer of data using the Rendezvous daemon. It enables communication via many of the features provided by the TIBCO Rendezvous product, including Certified Message Delivery, Distributed Queue and so on. The sending and receiving applications can be on multiple platforms, as long as the Rendezvous daemon is running on the host machine, or is remotely accessible to the host.

**Note:** The TIBCO RV control is available in BEA Workshop for WebLogic Platform only for licensed users of WebLogic Server Process Edition.

## Disclaimer

Use of the TIBCO RV control with BEA WebLogic Server Process Edition in no manner confers or grants the right to use TIBCO Rendezvous including "dynamic libraries". In order to use such TIBCO products, the user of the TIBCO RV control must obtain a valid license from TIBCO. See <http://www.tibco.com> for information on how to obtain a licensed copy of Rendezvous.

## XML MetaData Cache Control

The XML MetaData Cache control is used in business processes to access and retrieve XML metadata maintained in XML MetaData Cache. This cache is managed by WebLogic Integration Administration Console or the MBean API, which allows users to create their own NetUI-based consoles. The XML MetaData Cache is a global, domain-wide cache. So, data maintained in the cache can be accessed by any business process that is deployed in that domain. The cache can also be used for sharing data within a cluster. The cache is primarily used to maintain configuration metadata. Data is stored as key-value pairs where key is of type String and the value

contains XML data. Data from the cache is made available permanently through file-based storage. For each XML document that is added to the cache, a new XML MetaData Cache file is created. The XML MetaData Cache control in a business process uses the key to retrieve XML metadata associated with the key value, from the cache. For more information, see [XML MetaData Cache Control](#) in *Using Integration Controls*.

## Other Available Application Controls

This section describes the following application controls:

- [Beehive Controls](#)
- [JDBC Control](#)
- [Timer Control](#)
- [Service Control](#)
- [EJB Control](#)
- [JMS Control](#)

## Beehive Controls

Beehive controls are reusable components you can use anywhere within an enterprise application. You can use built-in controls provided with Workshop for WebLogic Platform, or you can create your own. When you are building WebLogic Server Process Edition applications, Beehive controls provide a convenient way to incorporate access to resources and encapsulate business logic. Controls can:

- Provide access to resources such as databases or other resources.
- Encapsulate logic for reuse in other applications.
- Modularize logic you want to keep separate from other application code.

Controls take advantage of Java 5 annotations for setting control properties. The system controls have a rich set of properties that are parameterized through annotation settings. As for custom controls, the control author can define annotation parameterization for any set of control properties.

Controls are built on the Apache Beehive Control framework. For more information, see [Controls: Getting Started](#) in the Apache Beehive documentation.

System controls and custom controls are two main types of controls. System controls provide you a way to connect to common application resources, such as databases, EJBs, JMS queues, web services, and so on. These controls require little or no modification to use in your application.

Workshop for WebLogic Platform provides several system controls, mainly designed to provide access to enterprise resources. For example, you can use the EJB control for access to Enterprise JavaBeans, the JMS control for access to the Java Message Service, and so on. For more information about the system controls, see [Using System Controls](#).

Custom controls provide a way to fully customize access to a resource or encapsulate some application functionality. You can design a custom control to do any task in an application.

You can build your own custom controls that are based on the same framework on which system controls are based. You design a custom control from the ground up, designing its interface and implementation, adding other controls as needed. You can design a custom control for use in one project, or you can design a custom control for easy reuse in multiple projects. For more information about the custom controls, see [Building Custom Controls](#).

For more information, see [Working with Beehive Controls](#).

## JDBC Control

A JDBC control makes it easy to access a relational database from your application. Using the JDBC control, you can issue SQL commands to the database. The JDBC control automatically performs the translation from database queries to Java objects, so that you can easily access query results.

A JDBC control can operate on any database for which an appropriate Java Database Connectivity (JDBC) driver is available and for which a data source is configured in WebLogic Server. When you add a JDBC control to your application, you specify a data source for that control. The data source indicates which database the control is bound to.

For more information, see [JDBC Control](#) in *Using System Controls*.

## Timer Control

Some transactions and events require a certain amount of time to complete. Others can run indefinitely if not aborted, and eat up resources. Still others must occur at a specific time. The Timer control provides the developer with a way to respond from code when a specified interval of time has elapsed or when a specified absolute time has been reached.

A timer control can notify a web service in the following ways:

- A specific (absolute) timeout has passed (e.g., January 23, 2012 midnight)
- A relative timeout has passed (e.g., an hour and seventeen minutes)
- A recurring time interval has elapsed (e.g., every 3 minutes)
- Both an initial timeout and a recurring interval (e.g., after one hour, every 5 minutes OR after January 1, 2010, every hour thereafter)

All Timer controls are instances of the `com.bea.control.TimerControl` base class or the `com.bea.control.TimerControlBean` base class. A Timer control is declared directly in a `.java` file. Timer controls are based on the EJB 2.1 timer service. Timer controls make a best-effort to do a callback to the client when a timer elapses, suitable for application timers. However timer controls are not a true real-time time service. Timer controls are allowed only in conversational (stateful) web services.

For more information, see [Timer Control](#) in *Using System Controls*.

## Service Control

A service control is a proxy for a web service. The web service client uses this proxy to access the web service. Using a service control allows the web service client to access the operations of a web service through simple method calls to the service control. The service control manages the SOAP message exchange with the web service and returns the results of the web service operation.

All service controls are interfaces that extend the `com.bea.control.ServiceControl` base class.

A service control provides an interface between your application and a web service, which allows your application to invoke the methods and handle the callbacks of that web service. Using a service control, you can connect to any web service for which a WSDL file is available.

For more information, see [Service Control](#) in *Using System Controls*.

## EJB Control

Enterprise JavaBeans (EJBs) are Java software components of enterprise applications. The Java 2 Enterprise Edition (J2EE) specification defines the types and capabilities of EJBs as well as the environment (or container) in which EJBs are deployed and executed. From a software developer's point of view, there are two aspects to EJBs: first, the development and deployment of EJBs; and second, the use of existing EJBs from client software. The EJB control makes it easy to use an existing, deployed EJB from your application.

For more information, see [EJB Control](#) in *Using System Controls*.

## JMS Control

Java Message Service (JMS) is a Java API for communicating with messaging systems. Messaging systems are often used in enterprise applications to communicate with legacy systems or to provide communication lanes between business components running in different environments or on different hosts. The JMS control enables applications built in WebLogic Workshop to easily interact with messaging systems that provide a JMS implementation, such as WebLogic Server or Message-Oriented Middleware systems (MOMs).

The following changes have been made in the JMS control that was provided in WebLogic Workshop 8.1:

- In WebLogic Workshop 8.1, there was {parm} support in method level annotations. This is no longer supported. You have to annotate parms to specify header or property values.
- New transacted annotation needs to be false when used in a container-managed transaction.

The JMS control in Workshop for WebLogic is the standard Beehive JMS control.

For more information, see [JMS Control](#) in *Using System Controls*.

## Using Asynchronous Interfaces

Web applications, including Web services, typically use the Hypertext Transport Protocol (HTTP) to provide communication between a client and a server (the application). HTTP is a request-response protocol. In a request-response protocol, each operation consists of a request message sent from the client to a server followed by a response message returned from the server to the client. The server must always send a response for the operation to complete successfully. Such requests are called synchronous because during the request the client is synchronized with the server; the client cannot continue processing until the server responds or the request times out (the client may time out if a response is not received within a specific period of time).

In a Web application, some of the operations the application performs may be long-running such as a Local Number Portability check when one changes phone companies. These processes could span days. It would be a poor design if individual request-response cycles were allowed to span days; such requests would unnecessarily engage resources on both the client and server hosts.

With WebLogic Server Process Edition, you can design your application to be asynchronous, which means that the application notifies the client when the response is ready. This allows the client to continue performing other work while the application completes the requested operation.

It also keeps each request-response interaction between the client and application as short as possible.

The following sections provide an overview of asynchrony and asynchronous interfaces:

- [Overview of Asynchrony](#)
- [Using Asynchrony](#)
- [Designing Asynchronous Interfaces](#)
- [Using Polling as an Alternative to Callbacks](#)
- [Designing Robust Asynchronous Interfaces](#)

## Overview of Asynchrony

Interactions between software components can be synchronous or asynchronous. An interaction is synchronous if the caller of a method must wait for the method's work to complete before the caller can continue its processing. An interaction is asynchronous if the called method returns immediately, allowing the caller to continue its processing without delay. An asynchronous interaction typically initiates a computation but does not wait for the result to be available, which means it must provide some way for the caller to obtain the results of the computation at a later time.

The distributed nature of Web applications introduces unpredictable and sometimes very long latencies, which means it may take an operation a long time to complete. If a business process executing over the network involves human interaction at the back end, an operation can take on the order of days. If all interactions over the Web were synchronous, clients with pending operations could consume resources on their host systems for unacceptably long periods of time.

WebLogic Server Process Edition provides tools that make it easy for you to build asynchronous Web services and Java controls that don't require clients to block execution while waiting for results. WebLogic Server Process Edition provides multiple approaches for returning results to your Web services' and Java controls' clients; you can choose the one that best suits each situation.

## Using Asynchrony

To create an asynchronous Web service, you provide one or more methods that accept requests from clients that begin an operation but do not wait for the operation to complete. Such methods typically return immediately, supplying the response portion of the initial request-response

interaction but not supplying the actual result of the requested operation. In an asynchronous interface, you also provide a mechanism for the client to obtain the results of the long-running operation when the results are ready. There are two ways to accomplish this:

- Implement methods that initiate requests and define callbacks to send results.
- Implement methods that initiate requests, methods that return request status (for example, "pending" or "complete"), and methods that return results. This approach is referred to as a *polling interface*.

## Using Callbacks

When you define a callback for a Web service, you are defining a message for the Web service to send to the client that notifies the client of an event that has occurred in your Web service. In this design, the client first calls the Web service with a request. This request call typically returns immediately (completing the first request-response interaction), meaning that the client does not have to wait for the operation to be completed. The client can now continue doing other tasks. When the Web service or Java control has finished processing the client's request, it sends a callback, that is, it sends a message back to the client notifying it that the request has been processed and/or providing the results. Note that a callback constitutes a second request-response interaction in which the request (not the response) is sent to the client. To learn more about the callback mechanism, see "Using Callbacks to Notify Clients of Events" in [Advanced JWS Programming: Implementing Asynchronous Features](#) in *Programming Web Services for WebLogic Server*.

To use a callback, two requirements must be met. First, if a Web service defines a callback the Web service must be conversational. Conversational Web services keep track of the originator of a request and can therefore send the callback to the appropriate caller. Secondly, the client must be capable of receiving and interpreting the callback. If the callback is defined by a Web service, then in essence the client must itself be a Web service since it must be capable of receiving messages. It must also be capable of correlating an incoming message with a previous request that it initiated. For more information about conversations, see [Designing Conversational Web Services](#).

## Using Polling

When the client of a Web service or Java control is not conversational, as is the case for Web pages and non-conversational Web services, callbacks cannot be used to notify the client of request completion. In addition, if the client of your Web service resides on a host that rejects unsolicited incoming traffic or is protected by firewalls, the host will reject callbacks because callbacks are, by nature, unsolicited, and the client will not receive the callback. To handle these

scenarios, Web services and Java controls must provide a polling interface. In a polling interface, the client first calls the Web service or Java control so that an operation can be initiated. This request call is synchronous but typically returns immediately, meaning that the client does not have to wait for the operation to be completed. The client can now continue doing other tasks, but must periodically call the Web service or Java control to check the status of its pending request. When a periodic check shows that the request has been completed, the client then calls the Web service or Java control to obtain the result. To learn more about implementing a polling interface, see [Using Polling as an Alternative to Callbacks](#).

Polling and callbacks are two different mechanisms to achieve asynchrony. Unless you are absolutely certain that the clients of your Web service or Java control will always require only one of these mechanisms, you may want to implement both approaches in order to handle all situations. Doing so provides the convenience of callbacks to those clients who can handle them, and a polling interface for clients who cannot accept callbacks.

## Designing Asynchronous Interfaces

This section discusses the best practices for creating and using Web services and Java controls with asynchronous interfaces. The first topic describes how to use polling as an alternative to callbacks. Then, various design questions for designing Web services and Java controls that can be called by both Web services and JSP (web) pages are answered.

### Using Polling as an Alternative to Callbacks

Because callbacks are, by definition, separated from the original request to which the callback is a response, they appear as unsolicited messages to the client's host. Many hosts refuse unsolicited network traffic, either because they directly reject such traffic or because they are protected by firewalls or other network security apparatus. Clients that run in such environments are therefore not capable of receiving callbacks.

Another requirement for handling callbacks is that the client is persistent by being conversational. If the client is a Web application, that is, a JSP page, or a non-conversational Web service, it cannot handle callbacks.

In order to allow clients that can't accept callbacks to use your Web services, you can supply a polling interface as an alternative. In a polling interface, you provide one or more methods that a client can call periodically to determine whether the result of a previous request is ready. Although the Web service or Java control will still be asynchronous in design, the interactions with a client are handled with synchronous (unbuffered) methods.

A typical design of a polling interface will have these three methods:

- A *start\_request* method that the client will call to initiate a request. If the client calls a Web service, the *start\_request* method will start a conversation.
- A *check\_status* method that the client will periodically call to check the status of the request. The method returns a boolean value indicating whether or not the request has been handled. If the client calls a Web service, the *check\_status* method will continue the conversation.
- A *get\_results* method that the client will call to get the results of the request. The results may for instance be returned as a String or an object of some kind, or `null` if the request could not be processed successfully. If the client calls a Web service, the *get\_results* method will finish the conversation.

Notice that a client using a polling interface needs to periodically check the status of the request, because the Web service or Java control cannot notify the client when its request has been processed. Also notice that the three methods will not be buffered. The *check\_status* and *get\_results* methods do not return void and cannot be buffered, while the *start\_request* method cannot be buffered because you need to ensure that this method has been handled before the *check\_status* is handled. (Remember that the relative handling order of buffered and unbuffered methods is uncertain. For more information, see [Using Buffering with Web Service Callbacks and Control Events](#).)

There are several other ways to implement a polling interface. The following example shows one such variation:

```
public class Conversation {
    @javax.jws.WebMethod
    @weblogic.jws.Conversation(weblogic.jws.Conversation.Phase.START)
    public void startRequest()
    {
        ...
    }
    @javax.jws.WebMethod
    @weblogic.jws.Conversation(weblogic.jws.Conversation.Phase.CONTINUE)
    public String getRequestStatus()
    {
        ...
    }
}
```

```

    }

    @javax.jws.WebMethod
    @weblogic.jws.Conversation(weblogic.jws.Conversation.Phase.FINISH)
    public void terminateRequest()
    { }
}

```

A client uses the `startRequest` method to initiate a request from a conversational Web service. The client then calls `getRequestStatus` periodically to check on the result. As before, the client is free to perform other processing between calls to `getRequestStatus`. The `getRequestStatus` method returns an indication that the request is pending until the request is complete. The next time the client calls `getRequestStatus` after the request is complete, the result is returned to the client. The client then calls `terminateRequest` to finish the conversation.

## Designing Robust Asynchronous Interfaces

This section explores several typical design solutions that constitute a good design and create a successful and robust Web service or Java control.

### Do I Need an Asynchronous Interface?

The first question you might need to answer for a Web service or Java control is whether the service or control needs to be asynchronous. There are certainly cases where a non-conversational, synchronous service or control will suffice, especially when the functionality it implements is relatively small, the request handling is relatively short, and the underlying infrastructure supporting this Web service is solid; for instance if you are working on a fail-proof intranet or if your control is called by a (synchronous) Web service on the same server. However, if any of these factors are not true or uncertain at the time of design, you will want to make your service or control asynchronous.

### Do I Need to Use Callbacks?

Callbacks are a powerful approach to designing asynchronous Web services or Java controls, relieving the client from periodically checking a request's status, as is required with polling. Especially when it is unclear how long a request will take to process, or if processing times vary wildly, using a callback is likely the most elegant implementation of asynchrony and loose

coupling. Using callbacks in combination with buffering of both methods and callbacks is particularly effective in dealing with high-volume traffic. However, callbacks require that the client is designed to accept incoming messages and is hosted in an environment that supports message delivery.

## Do I Need to Use Polling?

All asynchronous Web services and Java controls should provide a polling interface. If the Web service or Java control is asynchronous, a polling interface is required by any client that cannot accept callbacks; a polling interface is the only way such a client can obtain the results of operations initiated by asynchronous method invocations. You should think of a polling interface as the foundation interface of a Web service or Java control, and callbacks as "extra" functionality that is convenient for clients who can handle callbacks.

The exception to this guideline is a Java control for which the only clients will be conversational Web services or Java controls invoked by conversational Web services. Conversational WebLogic Workshop Web services can always accept callbacks. However, Java controls should be designed to be reusable. Assuming that the only clients a Java control will ever have are WebLogic Workshop Web services limits the reusability of the Java control.

## A Robust Web Service or Java Control

To create an asynchronous Web service or Java control that is robust and handles all situations, it is recommended that you implement both a callback and a polling interface. Your design might (among others) include the following methods:

- A *start\_request\_asynch* buffered method that the client will call to initiate a request. The method starts a conversation and notes that the callback mechanism will be used when the results are ready.
- A *callback\_results* buffered callback that sends the results to the client when the request is completed and finishes the conversation.
- A *start\_request\_synch* buffered method that the client will call to initiate a request. The method starts a conversation and notes that the polling mechanism will be used when the results are ready.
- A *check\_status* unbuffered method that the client will periodically call to check the status of the request. The method continues a conversation and returns a Boolean value indicating whether or not the request has been completely handled.
- A *get\_results* unbuffered method that the client will call to get the results of the request. The method finishes the conversation.

Other implementations of this design are possible. For a variation, see [Using Polling as an Alternative to Callbacks](#).

For more information, see [Designing Asynchronous Interfaces](#) in *Workshop for WebLogic Platform Programmer's Guide*.

## Conversations

A single Web service may communicate with multiple clients at the same time, and it may communicate with each client multiple times. In order for the Web service to track data for the client during asynchronous communication, it must have a way to remember which data belongs to which client and to keep track of where each client is in the process of operations. In WebLogic Server Process Edition, you use conversations to uniquely identify a given communication between a client and your Web service and to maintain state between operations.

Conversations are essential for any Web service involved in asynchronous communication. This includes Web services that communicate with clients using callbacks or polling interfaces, and Web services that use controls with callbacks.

The following sections provide more information on conversations:

- [Overview of Conversations](#)
- [Correlating Messages with a Unique Identifier](#)
- [Implementing Conversations](#)
- [Designing a Web Service to Use Conversations](#)

## Overview of Conversations

A Web service and a client may communicate multiple times to complete a single task. Also, multiple clients may communicate with the same Web service at the same time. Conversations provide a straightforward way to keep track of data between calls and to ensure that the Web service always responds to the right client.

Conversations meet two challenges inherent in persisting data across multiple communications:

- Conversations uniquely identify a communication between a client and a Web service, so that messages are always returned to the correct client. For example, in a shopping cart application, a conversational Web service keeps track of which shopping cart belongs to which customer.

- Conversations maintain state between calls to the Web service; that is, they keep track of the data associated with a particular client between calls. Conversations ensure that the data associated with a particular client is saved until it is no longer needed or the operation is complete. For example, in a shopping cart application, a conversational Web service remembers which items are in the shopping cart while the customer continues shopping.

## Correlating Messages with a Unique Identifier

When a client begins a conversation with a service, WebLogic Server Process Edition creates a context in which to keep track of state-related data during the exchange. This new context is identified by a conversation ID, a string that uniquely identifies the conversation. The Web service uses this conversation ID to correlate messages to the client through each operation it performs. The conversation ID ensures that a message sent or received by the Web service is always associated with the appropriate client. You can see the conversation ID in action when you test an asynchronous Web service in Test View.

For more information, see [Designing Conversational Web Services](#).

## Implementing Conversations

Conversations maintain a Web service's state-related data and correlate communications between the Web service, its clients, and other resources. You should implement conversations in any Web service design that is asynchronous or involves multiple communications with a client or Java control in connection with a single request.

## Understanding Conversation Context

When a client calls a service operation that is annotated to start a conversation, WebLogic Server Process Edition creates a conversation context through which to correlate calls to and from the service and to persist its state-related data.

When a conversation starts, WebLogic Server Process Edition does the following:

- Creates a context through which to maintain the scope of the conversation and associates it with a conversation ID.
- Starts an internal timer to measure idle time.
- Starts an internal timer to measure the conversation's age.

When WebLogic Server Process Edition performs all of these tasks, it creates a context for the conversation. Each piece of information—including the conversation ID, persistent data, idle time and age—is part of the conversation's context.

The conversation ID is a particularly useful item in the conversation's context. It attaches to each communication, which helps each of the resources, Web services, and clients involved in the conversation identify which communications belong to which conversation. To learn more about conversation IDs, see [Overview of Conversations](#).

## Designing a Web Service to Use Conversations

As you build services that support conversations, you should keep in mind a few characteristics of conversations. First, WebLogic Server Process Edition automatically handles correlation between two Web services that support conversations. In other words, if your Web service supports conversations and calls the conversational methods of another Web service that supports conversations, WebLogic Server Process Edition manages the conversation, including the conversation ID, automatically.

However, the scope of conversation context is limited to the service itself. You cannot assume that the state of another Web service is being persisted simply because your service calls one of its methods during the course of a conversation. The other Web service is responsible for its own state maintenance.

Also keep in mind that a Web service's state is only updated on the successful completion of methods or callback handlers that are marked with the conversation phase attributes `start`, `continue`, or `finish`. This excludes internal methods of your service, which are not operations and so can not be conversational.

For more information, see [Designing Conversational Web Service](#).

Controls: Service Enablement

# Business Process Management: Process Driven Services

WebLogic Server Process Edition allows you to model and execute business processes that span multiple internal systems, external resources, and users. From the business process management (BPM) perspective, the enterprise is a set of business services that are accessed through controls that can be orchestrated to model a business process. Business processes allow you to orchestrate the execution of business logic and the exchange of business documents among back-end systems, users, and trading partners (systems and users) in a loosely coupled fashion.

The following sections provide further information on BPM and the Web services available as business process resources:

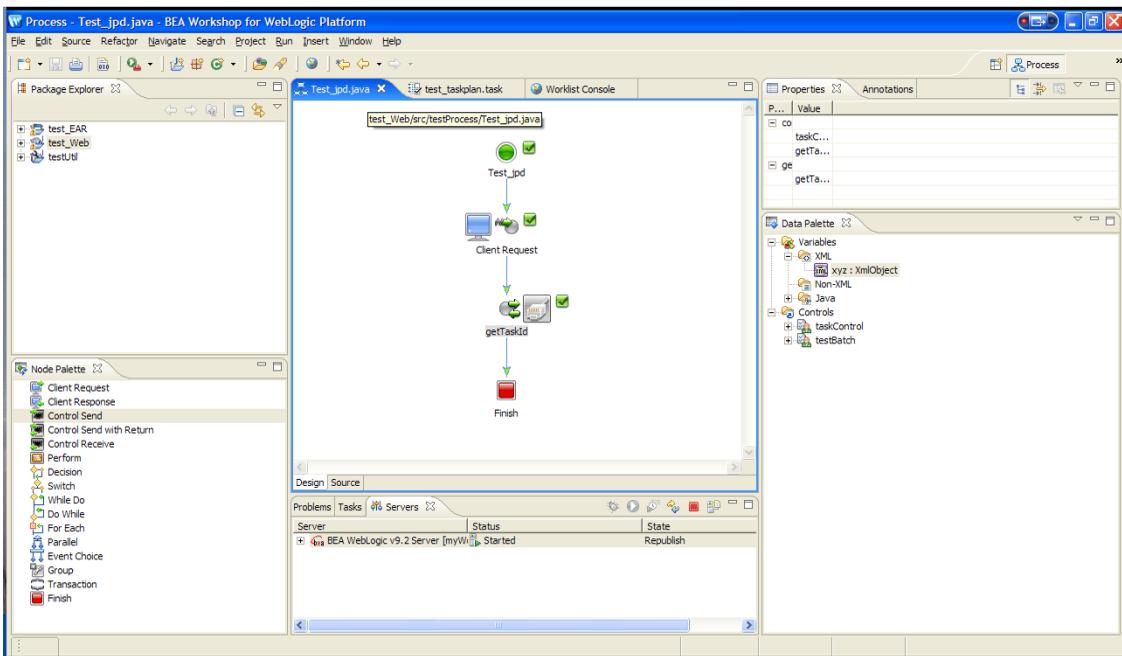
- [Business Process Management Overview](#)
- [Business Process Management Features](#)
- [Web Services Available as Business Process Resources](#)
- [Building a Business Process](#)
- [Stateful and Stateless Processes](#)

## Business Process Management Overview

The business process engine enables you to easily create a graphical representation of your business process, allowing you to focus on the application logic rather than on implementation details.

You create a graph of component nodes in your business process by dragging components from the Business Process Palette and dropping them onto the Design View pane. Program control is represented visually by these nodes (or shapes) and the connections between them. Effectively, you create a graphical representation of your business process and its interactions with clients and resources, such as databases, JMS queues, file systems, and other components. The following figure provides an example of the business process editor.

**Figure 3-1 Business Process Editor**



For more information on building business processes, see [Guide to Building Business Processes](#).

## Business Process Management Features

The following table details the key features of WebLogic Server Process Edition business process management (BPM).

**Table 3-1 BPM Key Features**

<b>Feature</b>	<b>Description</b>
Unified access to resources through Controls	This enables you to view business activities as services and model the business process to orchestrate integration and automation. Business processes seamlessly interact with users, applications, back-end resources, and resources inside and outside the firewall.
New simplified structured business processes	The new simplified structure provides XML for the business process flow and Java for the operations.
Graphical business process editing for high-level integration scenarios	This allows you to carry out message-based, transformation routing. As the business processes are Java classes, the business process (JPD) files also contain the metadata that describes the business process logic. Additionally, you can perform two-way editing from Design as well as from Source view.
Support for Java code in business process nodes	You are only a click away from Java coding.
Support for synchronous and asynchronous business process invocations	Messages (including synchronous starts) can be sent from external Java clients via Remote Method Invocation (RMI).
Process implementation optimization for performance	The following processes are supported: <ul style="list-style-type: none"> <li>• Stateless synchronous</li> <li>• Stateless asynchronous</li> <li>• Stateful asynchronous</li> </ul>

## Web Services Available as Business Process Resources

WebLogic Server Process Edition leverages Web services, asynchronous communication, and XML messaging at the platform level. At this level, you can use these services across internal and external integrations to simplify development and integration of loosely coupled and asynchronous applications.

WebLogic Server Process Edition features native support for Web services, including Web service security and reliable messaging. You can invoke Web services from within a WebLogic Server Process Edition business process. You can also expose business processes as a Web



using the graphical tools, the source code is automatically written to a business process file. When you do need to write Java code, it is always available with single click access (**Source** view).

The business process management functionality of WebLogic Server Process Edition enables corporate developers to develop, run, and maintain complex e-business processes that integrate existing enterprise systems, cross-enterprise applications, and end-user decision makers.

For more information, see [Guide to Building Business Processes](#).

## Stateful and Stateless Processes

There are two types of business processes; Stateful and Stateless. A Stateful process is a business process which is compiled into an entity bean and runs within the scope of one or more JTA transactions. A Stateless process is a business process which is compiled into a Stateless session bean and runs within one JTA transaction. By default, a business process is Stateless until you add a blocking construct to the data flow, that is, add a process that affects a transaction boundary. For more information about transaction boundaries, see [Transaction Boundaries](#) in *Guide to Building Business Process*.

The following sections provide more information on Stateful and Stateless processes:

- [Stateless Processes](#)
- [Stateful Processes](#)
- [Determining if your Business Process is Stateful or Stateless](#)

## Stateless Processes

Stateless processes support business scenarios that involve short-running logic and have high performance requirements. A Stateless process is optimized for lower-latency, higher-performance execution because it does not persist its state to a database. For example, a Stateless process is one that receives a message asynchronously from a client, transforms the message, and then sends it asynchronously to a resource using a control. Another example is a process that starts with a message broker subscription, transforms a message, and publishes it to another message broker channel. Such a process is analogous to the kinds of routing rules used by traditional message brokering or message routing systems.

For information on working with variables in a Stateless process, see [Building Stateless and Stateful Business Processes](#) in *Guide to Building Business Processes*.

## Stateful Processes

Stateful processes support business scenarios that involve complex, long-running logic and therefore have specific reliability and recovery requirements. A process is made Stateful by the addition of Stateful nodes or logic that forces transaction boundaries. For more information on transaction boundaries, see [Transaction Boundaries](#) in *Guide to Building Business Process*.

For example, a process that receives a message, transforms it, sends it to a business partner, and then waits for an asynchronous response is Stateful because the act of waiting forces a transaction boundary. This is necessary to ensure that:

- The process can recover and continue execution without loss of data in the event of a system outage during this waiting period.
- System resources are used efficiently during this waiting period.

For information on working with variables in a Stateful process, see [Building Stateless and Stateful Business Processes](#) in *Guide to Building Business Processes*.

## Determining if your Business Process is Stateful or Stateless

The Start node Property Editor displays whether a business process is Stateless or Stateful. If a process is Stateless, the Property Editor displays the message “Stateless = True” and the Start node icon displays the  icon. If a process is Stateful, the Property Editor displays the message “Stateless = False” and the Start node icon displays the  icon.

# Data Transformation

Data transformation is the mapping and conversion of data from one format to another. For example, XML data can be transformed from XML data valid to one XML Schema to another XML document valid to a different XML Schema. Other examples include the data transformation from non-XML data to XML data.

Data transformation enables you to translate between XML, non-XML, and Java data formats, allowing you to rapidly integrate heterogeneous applications regardless of the format used to represent data. The data transformation functionality is available through a Transformation Control, and data transformations can be packaged as controls and re-used across multiple business processes and applications.

The following sections provide an overview of data transformation and introduce the key features of data transformation:

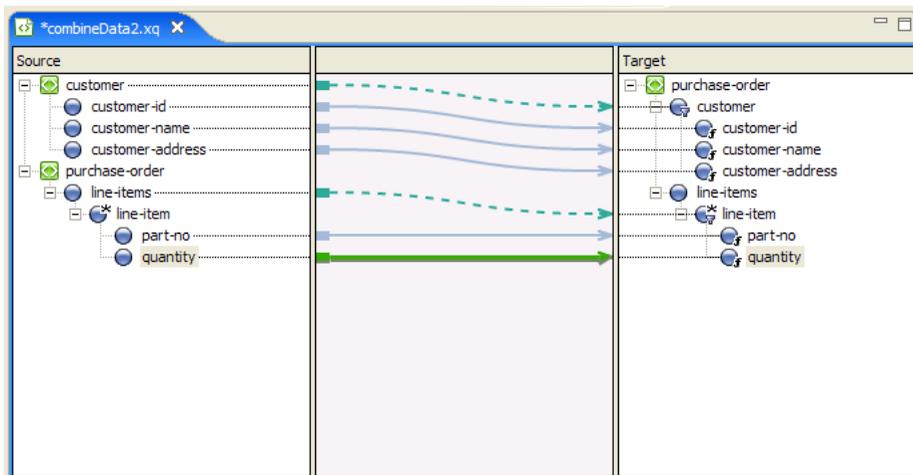
- [Data Transformation Overview](#)
- [Data Transformation Features](#)

## Data Transformation Overview

In a WebLogic Workshop business process, you can transform XML data using XQuery expressions or eXtensible Stylesheet Language Transformations (XSLTs). WebLogic Server Process Edition provides the functionality for executing existing XSLTs in business processes, and also offers a new and easier path to data transformation through XQuery. XQuery is a standards-based query language with the familiar simplicity of SQL-like expressions and an easy-to-use data mapping tool.

WebLogic Server Process Edition features a powerful visual data mapping tool, the *XQuery Transformation Mapper*, that enables you to easily generate complex transformations. The following figure shows the XQuery Transformation Mapper. The mapper functionality of WebLogic Workshop enables the conversion of data of different types. In addition, you can assign untyped data to typed variables, include parameter names and custom Java classes in maps, and display links implied from user-edited functions and structural relationships in maps.

**Figure 4-1 XQuery Transformation Mapper**



WebLogic Workshop generates a query from this graphical representation of a data transformation. The generated query is invoked during run time by the business process to transform data. The query is written in the XQuery language—a language defined by the World Wide Web Consortium (W3C) that provides a vendor independent language for the query and retrieval of XML data.

You can also import an existing XSLT into WebLogic Workshop for data transformation. An XSLT is written in the eXtensible Stylesheet Language (XSL)—an older language defined by the W3C that supports the use of stylesheets for the conversion of XML data. In WebLogic Workshop, the preferred method for data transformations is queries in the XQuery language. Data transformations using XSL transformations is supported primarily for legacy applications.

# Data Transformation Features

The following table details the key data transformation features.

**Table 4-1 Data Transformation Features**

Feature	Properties
Data transformation	<p>You can package transformations as controls that can be treated as resources and reused across multiple processes and integration solutions. Data transformation can take place between any of the following input-output data types: XML Data, Non-XML Data, Java Primitives, and Java classes. WebLogic Server Process Edition allows multiple input sources to a transformation and supports complex relations and constraints including joins, unions, and grouping by key fields. WebLogic Server Process Edition also enables transformation of XML grammars. For more information, see <a href="#">Guide to Data Transformation</a>.</p> <p>For a step-by-step tutorial that illustrates the use of business process actions for data transformation, see <a href="#">Tutorial: Building Your First Data Transformation</a>.</p>
Integration with business processes	<p>WebLogic Server Process Edition enables the transformation of data in a business process using transformations written in XQuery or eXtensible Stylesheet Language Transformations (XSLT).</p> <p>WebLogic Server Process Edition transforms data:</p> <ul style="list-style-type: none"> <li>• Received as an incoming message into the business process.</li> <li>• Before the business process sends an outgoing message.</li> <li>• Inside the business process.</li> </ul>
XQuery Mapper	<p>WebLogic Server Process Edition provides you with a visual modeling tool for transformation between any combination of XML, non-XML, and Java data formats. WebLogic Server Process Edition enables the visual transformation of data from one format to another through a drag-and-drop mechanism and engages the power of XQuery functions and operators. For more information, see <a href="#">Transforming Data Using the XQuery Mapper</a>.</p>
Format Builder	<p>WebLogic Server Process Edition provides you with a tool that enables the creation of metadata to describe non-XML data. For more information, see <a href="#">Format Builder</a>.</p>

## Data Transformation

# Process Monitoring and Management

This chapter presents an overview of the *Process Configuration* and *Process Instance Monitoring* modules of the WebLogic Server Process Edition Administration Console.

You use the Process Configuration module to:

- View process type information and locate specific processes for configuration.
- View or update process type properties, such as the display name, tracking level, and reporting data policy.
- View or update the security policies for a process.
- Activate or deactivate a non-versioned process.
- Configure the activation time for a newly deployed process version, or rollback to a previous version.
- View an interactive or printable process type graph.
- View or update the selectors used to dynamically set control attributes for a Process or Service Broker control.

**Note:** You must be logged in as a member of the Administrators, IntegrationAdministrators, or IntegrationOperators group to make changes to the configuration for a process or dynamic control. IntegrationOperators cannot modify process security policies. For more information, see *About WebLogic Integration Users, Groups, Roles, and Security Policies* in [User Management](#) in the *Worklist Console Online Help*.

You use the Process Instance Monitoring module to:

- View summary statistics that reflect system health.
- View the summary or detailed status for selected instances.
- View an interactive or printable process instance graph.
- Terminate or suspend instances, resume previously suspended instances, or unfreeze frozen instances.

The following sections provide more information on the Process Configuration and Process Instance Monitoring modules:

- [Process Configuration](#)
- [Process Instance Monitoring](#)

## Process Configuration

You must be logged in as a member of the Administrators, IntegrationAdministrators, or IntegrationOperators group to make changes to the configuration for a process or dynamic control. IntegrationOperators cannot modify process security policies.

The following sections provide an overview of information related to business process administration:

- [Managing Process Tracking Data](#)
- [Process Security Policies](#)
- [Service Level Agreements](#)
- [Process Versions](#)
- [Dynamic Controls](#)

## Managing Process Tracking Data

The data generated as process instances execute is initially stored in the run time database. The monitoring information provided in the console is based on this data. In order to optimize performance, it is important to keep the amount of tracking data stored in the run time database to a minimum. This is accomplished by:

- Capturing only the necessary data.

- Archiving the data to an offline database if required for later analysis.
- Purging the data from the runtime database when it is no longer needed for monitoring from the console.

A combination of system and process properties control the management of tracking data.

For more information, see “Managing Process Tracking Data” under “About Process Configuration” in [Process Configuration](#) in *Using the WebLogic Integration Administration Console*.

## Process Security Policies

To ensure process security, the administrator can configure the following security policies for a process:

- Execution policy for process operations

The execution policy specifies whether the operations in the process are run as the start user or the caller's ID:

- If start user is specified, each operation assumes the identity of the user that started the process.
- If the caller's ID is specified, the operation after the call in assumes the identity of that interrupting call.

In addition, the administrator configures whether or not a single principal is required. If a single principal is required, then all incoming client requests must come from the same user.

Execution policy controls the identity used to access external or backend resources. It allows the administrator to specify whether a process accesses an external system as the invoking application or as an application that called into the process later. For example, suppose a process listens for a message on a channel and then waits for a client request. The administrator can set the execution policy to use the identity from the client request when the process subsequently accesses SAP.

- Process authorization policy

The role(s) authorized to invoke the process methods (client requests). All methods in the process inherit the role(s) specified in the process authorization policy.

**Note:** If the process authorization policy is not defined, everyone is authorized.

- Method authorization policy

The role(s) authorized to invoke the process methods (client requests). All methods inherit the role(s) specified in the process authorization policy. Additional roles can be added to the authorization policy for the method.

- Callback authorization policy

The roles authorized to invoke the process callback.

**Note:** If the callback authorization policy is not defined, everyone is authorized.

To learn how to set the security policies, see “Updating Security Policies” in [Process Configuration](#) in *Using the WebLogic Integration Administration Console*.

## Service Level Agreements

A service level agreement (SLA) specifies a performance target for a process. It is typically an internal or external commitment that a process will be executed within a specified period of time.

To assist you in achieving the SLA for a process, the WebLogic Server Process Edition Administration Console allows you to set the following thresholds:

- SLA threshold, which represents the commitment applicable to the process type (number of seconds, minutes, hours, or days).
- SLA warning threshold, which is a percent of the total SLA.

Process status relative to these thresholds is tracked for each process instance as follows:

- When the elapsed time for a process instance reaches the warning threshold, a warning is displayed on the Process Instance Summary and Detail pages. The amount of time remaining until the SLA threshold will be reached is also displayed.
- When the elapsed time exceeds the SLA set, a red flag is displayed. The amount of time the SLA threshold has been exceeded is also displayed.

This ability to set SLA thresholds allows you to easily identify processes that do not execute within the target time frame. You can then make the changes necessary to meet agreements between suppliers and customers, or to achieve your own performance goals. To learn how to set the SLA for a process, see “Viewing and Changing Process Details” under “About Process Configuration” in [Process Configuration](#) in *Using the WebLogic Integration Administration Console*.

## Process Versions

When developers need to modify a deployed process, they must create a new process version and then release it into production along with older versions. When multiple versions are deployed, the system determines which version to use when creating new instances. The administrator controls the release of a process version by:

- Enabling or disabling a version.
- Setting the activation time for a version.

When creating a new instance, the system selects the version with the most recent activation time from among the enabled versions. (A disabled version is not available for selection.)

When an administrator activates a process by setting its activation time, instances currently running are not affected. Only instances that are created after the new version becomes active are created based on the new version.

If a newly activated version experiences problems, a rollback is easily accomplished by doing one of the following:

- Updating the activation time on the prior version.
- Disabling the problem version. In this case, the enabled version with the most recent activation date becomes the active version.

To learn more about how to enable or disable a version, or to configure the activation time, see “Managing Process Versions” in [Process Configuration](#) in *Using the WebLogic Integration Administration Console*.

## Dynamic Controls

Dynamic controls, which currently include the Service Broker and Process controls, provide the means to dynamically set control attributes through a combination of look-up rules and look-up values. This process is known as *dynamic binding*. In dynamic binding, the process developer specifies look-up rules, and the administrator defines the look-up values. This design pattern allows control attributes to be reconfigured for a running application, without redeployment.

The look-up or selector values are stored in the `DynamicProperties.xml` file, which is located in the `wliconfig` subdirectory of the domain root. You can manage the values stored in the `DynamicProperties.xml` file from the **View Dynamic Control Properties** page of the Process Configuration module.

Dynamic binding changes made in the WebLogic Server Process Edition Administration Console override both configuration changes made in the Workshop development environment and static annotations.

For more information about Dynamic Controls, see [Process Configuration](#) in *Using the WebLogic Integration Administration Console*.

## Process Instance Monitoring

The information displayed in the Process Monitoring module is based on the tracking data stored in the run time database. A combination of system-level and process-level properties control the capture and archiving of data. To learn more about how tracking data is managed, see “Managing Process Tracking Data” in [Process Instance Monitoring](#) in *Using the WebLogic Integration Administration Console*.

The following table lists the pages you can access from the Process Instance Monitoring module. The tasks associated with each page are detailed.

**Table 5-1 Process Instance Monitoring Tasks**

Page	Associated Tasks
Process Instance Statistics	<ul style="list-style-type: none"><li>• For each process type, the average elapsed time and a count of the number of instances in each state (running, suspended, aborted, frozen, terminated, completed, and above SLA) are displayed.</li><li>• Filter the list by URI or display name. Use ? to match any single character or * to match zero or more characters.</li></ul>
Process Instance Summary	<ul style="list-style-type: none"><li>• View a list of process instances. Instance ID, display name, process label, start time, elapse time, and status (running, completed, frozen, aborted, suspended) are displayed.</li><li>• Filter the list by process status (for example, running, frozen, or over SLA), instance ID, or process label.</li><li>• Access the Process Instance Details page for a selected process.</li><li>• Set the number of instances to display per page.</li><li>• Suspend, Resume, Terminate, or Unfreeze process instances.</li></ul>
Advanced Search	<ul style="list-style-type: none"><li>• Construct an advanced search using process properties such as status, time started or completed, elapsed time, or SLA status.</li></ul>

**Table 5-1 Process Instance Monitoring Tasks**

---

System Health	<ul style="list-style-type: none"><li>• View general indicators of system health and performance trends by process type, including the process types that are taking the longest to execute, those that have not completed within SLA thresholds, and those that are failing to complete.</li></ul>
Process Instance Details	<ul style="list-style-type: none"><li>• View process instance properties, including variable values for the running instance, worklist tasks created by or associated with the process, and business messages associated with the process.</li><li>• Suspend, Resume, Terminate, or Unfreeze the process instance.</li><li>• Access an interactive or printable process graph.</li></ul>

---

For more information, see [Process Instance Monitoring](#) in *Using the WebLogic Integration Administration Console*.

## Process Monitoring and Management

# Index

## A

- asynchronous interfaces
  - designing 2-13
  - designing robust interfaces 2-15
- asynchrony
  - overview 2-11
  - using 2-11

## B

- Beehive controls 2-7
- BPM 3-1
  - key features 3-2
  - overview 3-1
- business process editor 3-2
- business process resources 3-3

## C

- callbacks 2-12
- control
  - database 2-8
  - EJB 2-9
  - JMS 2-10
  - service 2-9
  - timer 2-8
- conversations
  - designing a web service to use 2-19
  - implementing 2-18
  - overview 2-17
  - understanding context 2-18

## D

- data transformation
  - features 4-3
  - overview 4-1
- database control 2-8

## E

- EJB control 2-9
- e-mail control 2-3

## F

- file control 2-2

## H

- HTTP control 2-4

## I

- integration controls 2-1
  - e-mail 2-3
  - file 2-2
  - HTTP 2-4
  - MQseries 2-5
  - service broker 2-3
  - Tibco RV 2-6
  - WLI JMS 2-3
  - XML metadata cache 2-6

## J

- JMS control 2-10

## **M**

- messages
  - unique identifier 2-18
- MQseries control 2-5

## **P**

- polling 2-12
- process
  - building a business 3-4
  - configuration 5-2
  - instance monitoring 5-6
  - security policies 5-3
  - versions 5-5
- process configuration 5-2
- process edition
  - component 1-12
  - options 1-7
- process tracking data
  - managing 5-2
- processes
  - stateful 3-6
  - stateless 3-5

## **S**

- service broker control 2-3
- service control 2-9
- stateful process 3-6
- stateless process 3-5

## **T**

- TIBCO
  - website 2-6
- TIBCO RV
  - multiple platforms 2-6
- Tibco RV control 2-6
- timer control 2-8

## **W**

- web service
  - designing 2-19
- WLI JMS control 2-3

## **X**

- XML metadata cache control 2-6
- XQuery transformation mapper 4-2