



BEA WebLogic ServerTM and BEA WebLogic ExpressTM

Assembling and Configuring Web Applications

BEA WebLogic Server Version 6.1
Document Date: November 25, 2003

Copyright

Copyright © 2002 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA WebLogic Collaborate, BEA WebLogic Commerce Server, BEA WebLogic E-Business Platform, BEA WebLogic Enterprise, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Server, E-Business Control Center, How Business Becomes E-Business, Liquid Data, Operating System for the Internet, and Portal FrameWork are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Assembling and Configuring Web Applications

Part Number	Document Date	Software Version
N/A	November 25, 2003	BEA WebLogic Server Version 6.1 SP6

Contents

About This Document

1. Web Applications Basics

Overview of Web Applications	1-1
Main Steps to Create a Web Application	1-2
Directory Structure	1-3
URLs and Web Applications.....	1-4
Web Application Developer Tools	1-5
ANT Tasks to Create Skeleton Deployment Descriptors.....	1-5
Web Application Deployment Descriptor Editor.....	1-6
XML Editor.....	1-6

2. Deploying Web Applications

Overview of Deploying Web Applications	2-1
Deploying Web Applications Using Auto-Deployment.....	2-3
How to Deploy a Web Application Using Auto-Deployment	2-3
Uploading a Web Application to the applications Directory	2-5
Redeploying a Web Application Using Auto-Deployment.....	2-5
Redeploying a Web Application in a .war Archive	2-5
Redeploying a Web Application in Exploded Directory Format.....	2-6
Deploying Web Applications with Production Mode Enabled	2-7
Deploying a Web Application Using the Administration Console	2-8
Deploying a Web Application Using the weblogic.deploy Utility	2-9
Redeploying a Web Application with Production Mode Enabled	2-10
Redeploying a Web Application Using the Administration Console	2-11
Redeploying a Web Application Using the weblogic.deploy Utility	2-12
Partial Redeployment	2-13

Refreshing Static Components Without Redeploying the Application.....	2-13
Un-deploying Web Applications	2-15
Un-deploying a Web Application Using the Administration Console.....	2-15
Un-deploying a Web Application Using the weblogic.deploy Utility	2-15
Deleting a Web Application	2-16
Deleting a Web Application in the applications Directory	2-16
Deleting a Web Application Using the Administration Console	2-17
Deleting a Web Application Using the weblogic.deploy Utility.....	2-17
Deploying Stand-Alone Web Applications	2-18
 3. Configuring Web Application Components	
Configuring Servlets.....	3-2
Servlet Mapping	3-2
Servlet Initialization Parameters.....	3-4
Configuring JSP.....	3-5
Configuring JSP Tag Libraries	3-6
Configuring Welcome Pages	3-7
Setting Up a Default Servlet.....	3-8
Customizing HTTP Error Responses	3-9
Using CGI with WebLogic Server	3-9
Configuring WebLogic Server to use CGI.....	3-10
Requesting a CGI Script.....	3-11
Serving Resources from the CLASSPATH with the ClasspathServlet.....	3-12
Configuring External Resources in a Web Application	3-13
Referencing EJBs in a Web Application	3-15
Determining the Encoding of an HTTP Request.....	3-16
Mapping IANA Character Sets to Java Character Sets	3-17
 4. Using Sessions and Session Persistence in Web Applications	
Overview of HTTP Sessions	4-1
Setting Up Session Management	4-2
HTTP Session Properties.....	4-2
Session Timeout	4-2
Configuring Session Cookies	4-3
Using Longer-Lived Cookies	4-3

Logging Out and Ending a Session	4-4
Configuring Session Persistence	4-4
Common Properties.....	4-5
Using Memory-based, Single-server, Non-replicated Persistent Storage ..	4-6
Using File-based Persistent Storage	4-6
Using a Database for Persistent Storage (JDBC persistence)	4-7
Using Cookie-Based Session Persistence	4-9
Secure Cookies.....	4-10
Using URL Rewriting.....	4-11
Coding Guidelines for URL Rewriting	4-11
URL Rewriting and Wireless Access Protocol (WAP)	4-12
 5. Configuring Security in Web Applications	
Overview of Configuring Security in Web Applications	5-1
Setting Up Authentication for Web Applications	5-2
Multiple Web Applications, Cookies, and Authentication.....	5-3
Single Sign-on Requirements.....	5-4
Restricting Access to Resources in a Web Application	5-5
Using Users and Roles Programmatically in Servlets.....	5-6
 6. Application Events and Listeners	
Overview of Application Events and Listeners.....	7-1
WebLogic Server 6.1 with J2EE 1.2 and J2EE 1.3 Functionality	7-2
WebLogic Server 6.1 with J2EE 1.2 Plus Additional J2EE 1.3 Features.	
7-3	
WebLogic Server 6.1 with J2EE 1.2 Certification.....	7-3
Servlet Context Events	7-3
HTTP Session Events	7-4
Configuring an Event Listener	7-5
Writing a Listener Class	7-5
Templates for Listener Classes.....	7-6
Servlet Context Listener Example.....	7-6
HTTP Session Attribute Listener Example.....	7-7
Additional Resources.....	7-8

7. Filters

Overview of Filters	8-1
WebLogic Server 6.1 with J2EE 1.2 and J2EE 1.3 Functionality	8-2
WebLogic Server 6.1 with J2EE 1.2 Plus Additional J2EE 1.3 Features 8-2	
WebLogic Server 6.1 with J2EE 1.2 Certification.....	8-2
How Filters Work	8-3
Uses for Filters	8-3
Configuring Filters	8-4
Configuring a Filter	8-4
Configuring a Chain of Filters.....	8-6
Writing a Filter	8-6
Example of a Filter Class.....	8-8
Filtering the Servlet Response Object	8-9
Additional Resources.....	8-9

8. Writing Web Application Deployment Descriptors

Overview of Web Application Deployment Descriptors.....	9-2
Tools for Editing Deployment Descriptors	9-2
Writing the web.xml Deployment Descriptor	9-3
Main Steps to Create the web.xml File	9-3
Detailed Steps to Create the web.xml File	9-4
Sample web.xml	9-21
Writing the WebLogic-Specific Deployment Descriptor (weblogic.xml)	9-24
Main Steps to Create the weblogic.xml File	9-24
Detailed Steps to Create the weblogic.xml File	9-25

A. web.xml Deployment Descriptor Elements

icon Element	A-4
display-name Element	A-4
description Element	A-5
distributable Element	A-5
context-param Element.....	A-6
filter Element	A-6
filter-mapping Element.....	A-7

listener Element	A-8
servlet Element	A-8
icon Element.....	A-10
init-param Element	A-10
security-role-ref Element.....	A-11
servlet-mapping Element.....	A-12
session-config Element.....	A-13
mime-mapping Element	A-13
welcome-file-list Element	A-14
error-page Element	A-15
taglib Element.....	A-16
resource-ref Element	A-17
security-constraint Element.....	A-18
web-resource-collection Element.....	A-18
auth-constraint Element.....	A-19
user-data-constraint Element.....	A-20
login-config Element.....	A-21
form-login-config Element.....	A-22
security-role Element.....	A-22
env-entry Element	A-23
ejb-ref Element	A-24

B. weblogic.xml Deployment Descriptor Elements

description Element.....	B-2
weblogic-version Element	B-2
security-role-assignment Element	B-3
reference-descriptor Element.....	B-3
resource-description Element	B-4
ejb-reference-description Element	B-4
session-descriptor Element.....	B-4
Session Parameter Names and Values	B-5
jsp-descriptor Element.....	B-11
JSP Parameter Names and Values.....	B-11
container-descriptor Element.....	B-13
check-auth-on-forward Element.....	B-14

redirect-with-absolute-url	B-14
charset-params Element	B-14
input-charset Element	B-14
charset-mapping Element	B-15

Index

About This Document

This document describes how to assemble and configure J2EE Web Applications.

The document is organized as follows:

- Chapter 1, “Web Applications Basics,” provides an overview of using Web Applications in WebLogic Server.
- Chapter 2, “Deploying Web Applications,” describes how to deploy a Web Application on WebLogic Server.
- Chapter 3, “Configuring Web Application Components,” describes how to configure Web Application components.
- Chapter 4, “Using Sessions and Session Persistence in Web Applications,” describes how to use HTTP sessions and session persistence in a Web Application.
- Chapter 5, “Configuring Security in Web Applications,” describes how to configure authentication and authorization in a Web Application.
- Chapter 6, “Application Events and Listeners,” describes how to use J2EE event listeners in a Web Application.
- Chapter 7, “Filters,” describes how to use filters in a Web Application.
- Chapter 8, “Writing Web Application Deployment Descriptors,” describes how to manually write Web Application deployment descriptors.
- Appendix A, “web.xml Deployment Descriptor Elements,” provides a reference of deployment descriptor elements for the `web.xml` deployment descriptor.
- Appendix B, “weblogic.xml Deployment Descriptor Elements,” provides a reference of deployment descriptor elements for the `weblogic.xml` deployment descriptor.

Audience

This document is written for application developers who want to build e-commerce applications using the Java 2 Platform, Enterprise Edition (J2EE) from Sun Microsystems. It is assumed that readers know Web technologies, object-oriented programming techniques, and the Java programming language.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

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

A PDF version of this document is available on the WebLogic Server 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 Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at <http://www.adobe.com>.

Related Information

The BEA corporate Web site provides all documentation for WebLogic Server. The following WebLogic Server documents contain information that is relevant to creating WebLogic Server application components:

- **Programming WebLogic HTTP Servlets** at <http://e-docs.bea.com/wls/docs61/servlet/index.html>.
- **Programming WebLogic JSP** at <http://e-docs.bea.com/wls/docs61/jsp/index.html>.
- **Programming WebLogic Web Services** at <http://e-docs.bea.com/wls/docs61/webServices/index.html>.
- **Developing WebLogic Server Applications** at <http://e-docs.bea.com/wls/docs61/programming/index.html>.

For more information in general about Java application development, refer to the Sun Microsystems, Inc. Java 2, Enterprise Edition Web Site at <http://java.sun.com/products/j2ee/>.

Contact Us!

Your feedback on BEA 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 documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at <http://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	Usage
Ctrl+Tab	Keys you press simultaneously.
<i>italics</i>	Emphasis and book titles.
monospace text	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard. <i>Examples:</i> <pre>import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float</pre>
<i>monospace italic text</i>	Variables in code. <i>Example:</i> <pre>String CustomerName;</pre>

Convention	Usage
UPPERCASE TEXT	Device names, environment variables, and logical operators. <i>Examples:</i> LPT1 BEA_HOME OR
{ }	A set of choices in a syntax line.
[]	Optional items in a syntax line. <i>Example:</i> <code>java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]</code>
	Separates mutually exclusive choices in a syntax line. <i>Example:</i> <code>java weblogic.deploy [list deploy undeploy update] password {application} {source}</code>
...	Indicates one of the following in a command line: <ul style="list-style-type: none">■ An argument can be repeated several times in the command line.■ The statement omits additional optional arguments.■ You can enter additional parameters, values, or other information
.	Indicates the omission of items from a code example or from a syntax line.



1 Web Applications Basics

The following sections describe how to configure and deploy Web Applications:

- [“Overview of Web Applications” on page 1-1](#)
- [“Main Steps to Create a Web Application” on page 1-2](#)
- [“Directory Structure” on page 1-3](#)
- [“URLs and Web Applications” on page 1-4](#)
- [“Web Application Developer Tools” on page 1-5](#)

Overview of Web Applications

A Web Application contains an application’s resources, such as servlets, JavaServer Pages (JSP), JSP tag libraries, and static resources such as HTML pages and image files. A Web Application can also define links to resources outside of the application such as Enterprise JavaBeans (EJBs). Web Applications deployed on WebLogic Server use a standard J2EE deployment descriptor in conjunction with an optional WebLogic-specific deployment descriptor to define the resources and their operating parameters.

JSP pages and HTTP servlets can access all services and APIs available in WebLogic Server. These services include EJBs, database connections through Java Database Connectivity (JDBC), JavaMessaging Service (JMS), XML, and more.

Web Applications use a standard directory structure defined in the J2EE specification and can be deployed as a collection of files that use this directory structure (this type of deployment is called *exploded directory format*) or as an archived file called a `.war` file. Deploying a Web Application in exploded directory format is recommended primarily for use while developing your application. Deploying a Web Application as a `.war` file is recommended primarily for production environments.

The WAR file can be deployed alone or packaged in an Enterprise Archive (EAR file) with other application components. If deployed alone, the archive must end with a `.war` extension. If deployed in an EAR file, the archive must end with an `.ear` extension. (Note: If you are deploying an entire directory, do not name the directory `.ear`, `.war`, `.jar` and so on.)

Main Steps to Create a Web Application

The following steps summarize the procedure for creating a Web Application. You may want to use some of the provided developer tools for creating and configuring Web Applications. For more information, see [“Web Application Developer Tools” on page 1-5](#).

To create a Web Application:

1. Arrange the resources (servlets, JSPs, static files, and deployment descriptors) in the prescribed directory format. For more information, see [“Directory Structure” on page 1-3](#).
2. Write the Web Application deployment descriptor (`web.xml`). In this step you register servlets, define servlet initialization parameters, register JSP tag libraries, define security constraints, and define other Web Application parameters. (Information on the various components of Web Applications is included throughout this document.) Place the descriptor in the `WEB-INF` directory of the Web Application.

For detailed instructions, see [“Writing the web.xml Deployment Descriptor” on page 8-3](#).

You can also use the Administration Console to edit Web Application deployment descriptors. For more information, see [Web Application Deployment Descriptor Editor Help](#) at [at](#)

http://e-docs.bea.com/wls/docs61/ConsoleHelp/webservices_ddehelp.html.

3. Create the WebLogic-specific deployment descriptor (`weblogic.xml`). In this step you define JSP properties, JNDI mappings, security role mappings, and HTTP session parameters. If you do not need to define any attributes in this file, you do not need to create the file.

For detailed instructions on creating the WebLogic-specific deployment descriptor, see [“Writing the WebLogic-Specific Deployment Descriptor \(`weblogic.xml`\)” on page 8-24](#).

You can also use the Administration Console to edit Web Application deployment descriptors. For more information, see [Web Application Deployment Descriptor Editor Help](#) at http://e-docs.bea.com/wls/docs61/ConsoleHelp/webservices_ddehelp.html.

4. Archive the files in the above directory structure into `war` file. Only use archiving when the Web Application is ready for deployment in a production environment. (During development you may find it more convenient to update individual components of your Web Application by developing your application in exploded directory format.) To create a `.war` archive, use this command line from the root directory containing your Web Application:

```
jar cv0f myWebApp.war .
```

This command creates a Web Application archive file called `myWebApp.war`.

5. (Optional). Archive the Web Application into an Enterprise Application.
6. Deploy the Web Application on WebLogic Server. This final step configures your Web Application to service requests on WebLogic Server. For detailed instructions, see [“Deploying Web Applications” on page 2-1](#).

Directory Structure

You develop your Web Application within a specified directory structure so that it can be archived and deployed on WebLogic Server, or another J2EE compliant server. All servlets, classes, static files, and other resources belonging to a Web Application are organized under a directory hierarchy. The root of this hierarchy defines the *document*

root of your Web Application. All files under this root directory can be served to the client, except for files under the special directory `WEB-INF`, located under the root directory. The name of your Web Application is used to resolve requests for components of the Web Application.

Place private files in the `WEB-INF` directory, under the root directory. All files under `WEB-INF` are private, and are not served to a client.

`WebApplicationName/`

Place your static files, such as HTML files and JSP files in this directory (or a subdirectory). This directory is the document root of your Web Application.

`/WEB-INF/web.xml`

The Web Application deployment descriptor that configures the Web Application.

`/WEB-INF/weblogic.xml`

The WebLogic-specific deployment descriptor file that defines how named resources in the `web.xml` file are mapped to resources residing elsewhere in WebLogic Server. This file is also used to define JSP and HTTP session attributes.

`/WEB-INF/classes`

Contains server-side classes such as HTTP servlets and utility classes.

`/WEB-INF/lib`

Contains `.jar` files used by the Web Application, including JSP tag libraries.

URLs and Web Applications

You construct the URL used to access a Web Application from a client by using the following pattern:

`http://hoststring/ContextPath/servletPath/pathInfo`

Where

hoststring

is either a host name that is mapped to a virtual host or `hostname:portNumber`.

ContextPath

is the name of your Web Application.

servletPath

is a servlet that is mapped to the *servletPath*.

pathInfo

is the remaining portion of the URL, typically a file name.

If you are using virtual hosting, you can substitute the virtual host name for the *hoststring* portion of the URL.

For additional information, see [How WebLogic Server Resolves HTTP Requests at `http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req`](http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req).

Web Application Developer Tools

BEA provides several tools you can use to help you create and configure Web Applications. These tools are described in this section.

ANT Tasks to Create Skeleton Deployment Descriptors

You can use the WebLogic ANT utilities to create skeleton deployment descriptors. These utilities are Java classes shipped with your WebLogic Server distribution. The ANT task looks at a directory containing a Web Application and creates deployment descriptors based on the files it finds in the Web Application. Because the ANT utility does not have information about all of the desired configurations and mappings for your Web Application, the skeleton deployment descriptors the utility creates are incomplete. After the utility creates the skeleton deployment descriptors, you can use a text editor, an XML editor, or the Administration Console to edit the deployment descriptors and complete the configuration of your Web Application.

For more information on using ANT utilities to create deployment descriptors, see [Packaging Web Applications at `http://e-docs.bea.com/wls/docs61/programming/packaging.html#pack005`](http://e-docs.bea.com/wls/docs61/programming/packaging.html#pack005).

Web Application Deployment Descriptor Editor

The WebLogic Server Administration Console has an integrated deployment descriptor editor. You must create at least a skeleton `web.xml` deployment descriptor before using this integrated editor.

For more information, see [Web Application Deployment Descriptor Editor Help](http://e-docs.bea.com/wls/docs61/ConsoleHelp/webservices_ddehelp.html) at http://e-docs.bea.com/wls/docs61/ConsoleHelp/webservices_ddehelp.html.

XML Editor

BEA now provides a simple, user-friendly tool from Ensemble for creating and editing XML files. It can validate XML code according to a specified DTD or XML Schema. The XML editor can be used on Windows or Solaris machines and is downloadable from [BEA dev2dev](http://dev2dev.bea.com/resourcelibrary/utilitiestools/xml.jsp) at <http://dev2dev.bea.com/resourcelibrary/utilitiestools/xml.jsp>.

2 Deploying Web Applications

The following sections describe how to deploy Web Applications:

- [“Overview of Deploying Web Applications” on page 2-1](#)
- [“Deploying Web Applications Using Auto-Deployment” on page 2-3](#)
- [“Deploying Web Applications with Production Mode Enabled” on page 2-7](#)
- [“Redeploying a Web Application with Production Mode Enabled” on page 2-10](#)
- [“Refreshing Static Components Without Redeploying the Application” on page 2-13](#)
- [“Un-deploying Web Applications” on page 2-15](#)
- [“Deleting a Web Application” on page 2-16](#)
- [“Deploying Stand-Alone Web Applications” on page 2-18](#)

Overview of Deploying Web Applications

Deploying a Web Application is the final step that allows WebLogic Server to serve the components of a Web Application to clients. You can deploy a Web Application using one of several procedures, depending on your environment and whether or not

your Web Application is in production. You can use the WebLogic Server Administration Console, the `weblogic.deploy` utility, or you can use auto-deployment.

The instructions for deploying a Web Application presume that you have created a functional Web Application that uses the correct directory structure and contains the `web.xml` deployment descriptor and, if needed, the `weblogic.xml` deployment descriptor. For an overview of the steps required to create a Web Application, see [Main Steps to Create a Web Application at](#)

<http://e-docs.bea.com/wls/docs61/webapp/basics.html#100368>.

There are two modes you can use to run WebLogic Server, development mode and production mode. To enable development mode, start WebLogic Server with the following flag: `-Dweblogic.ProductionModeEnabled=false` (this is the default behavior). To enable production mode, start WebLogic Server with the following flag: `-Dweblogic.ProductionModeEnabled=true`. For information on setting these modes, see [Starting and Stopping WebLogic Servers at](#)

<http://e-docs.bea.com/wls/docs61/adminguide/startstop.html>.

When you specify development mode, you can use the auto-deploy feature to quickly deploy a Web Application by copying it into the `applications` directory of your Administration Server, located in the `config/mydomain` directory of the WebLogic Server installation (where *mydomain* is the name of a WebLogic Server domain). WebLogic Server automatically deploys any application that is copied into the `applications` directory. If a Web Application in `.war` format is modified, WebLogic Server automatically redeploys the Web Application. If the Web Application is in exploded format, see [“Redeploying a Web Application in Exploded Directory Format” on page 2-6](#). It is not advisable to store applications in the `config/mydomain/applications` directory. Applications already in this directory will deploy automatically even in production mode.

You can deploy a Web Application from your file system in exploded directory format or archived `.war` format. Exploded directory format means the Web Application components are arranged on your file system using the standard directory structure for Web Applications. A `.war` archive is created by using the Java `jar` utility to archive the contents of a Web Application in exploded directory format.

If you are using a cluster of managed WebLogic Servers, you can specify that your Web Application be deployed on one or more “targeted” Managed Servers. The Administration Server automatically copies the Web Application to the Managed Servers in the domain that have been targeted for the Web Application.

Deploying Web Applications Using Auto-Deployment

Auto deployment is a convenience feature designed for Web Application development and should not be used for production applications. When you move your application into production, use the manual deployment procedures described in “[Deploying Web Applications with Production Mode Enabled](#)” on page 2-7. You can only deploy a Web Application using auto-deployment on the Administration Server. If an auto-deployed Web Application is targeted to any Managed Servers, the Web Application is also deployed on the Managed Servers.

Before you can use auto-deployment, you must create a functional Web Application using the correct directory structure and containing the `web.xml` deployment descriptor and, if necessary, the `weblogic.xml` deployment descriptor. You can deploy your Web Application as an archived `.war` file or in exploded directory format (see “Directory Structure” on page 1-3).

How to Deploy a Web Application Using Auto-Deployment

To deploy a Web Application using auto-deployment:

1. Start your WebLogic Administration Server with the `-DProductionModeEnabled=false` flag. For more informations, see [Starting and Stopping WebLogic Servers](#) at <http://e-docs.bea.com/wls/docs61/adminguide/startstop.html>.
2. Copy the `.war` archive file or a directory containing your Web Application to the `applications` directory for your domain. The `applications` directory is located in the `config/mydomain` directory of the WebLogic Server installation (where `mydomain` is a WebLogic Server domain). The Web Application is automatically deployed by the Administration Server.
3. Configure the Web Application using the Administration Console.
 - a. Start the Administration Console in a Web browser.

- b. Expand the Deployments node in the left pane.
 - c. Expand the Web Applications node in the left pane. A list of Web Applications appears in the right pane, including the Web Application you just copied to the applications directory.
 - d. Click on the name of the Web Application.
 - e. Select the Targets tab and target servers, clusters, or virtual hosts by selecting the appropriate tab and moving a target from the Available to the Chosen column. An application is deployed to any Managed Servers when you target the Web Application to a server or cluster. Targeting a Web Application to a virtual host makes the Web Application available to clients through the virtual host.
 - f. Click Apply.
 - g. Configure other attributes under the Files and Other tabs. For more information, see [Web Applications / Configuration at `http://e-docs.bea.com/wls/docs61/ConsoleHelp/webappcomponent.html#webappcomponent_configuration`](http://e-docs.bea.com/wls/docs61/ConsoleHelp/webappcomponent.html#webappcomponent_configuration).
 - h. Click Apply.
4. Test your Web Application by accessing a resource through a Web browser. You access resources with a URL constructed as follows:

`http://myServer:myPort/myWebApp/resource`

Where:

- *myServer* is the name of the machine hosting WebLogic Server
- *myPort* is the port number where WebLogic Server is listening for requests
- *myWebApp* is the name of the Web Application archive file (*myWebApp.war*, for instance) or the name of a directory containing the Web Application.
- *resource* is the name of a resource such as a JSP, HTTP servlet, or HTML page.

Uploading a Web Application to the applications Directory

You can also use the Administration Console to upload a Web Application to the `applications` directory. This procedure can be useful if your WebLogic Server is located on a remote machine.

To upload a Web Application:

1. Start your WebLogic Administration Server with the `-DProductionModeEnabled=false` flag. For more informations, see [Starting and Stopping WebLogic Servers at `http://e-docs.bea.com/wls/docs61/adminguide/startstop.html`](http://e-docs.bea.com/wls/docs61/adminguide/startstop.html).
2. Select the Web Applications node in the left pane.
3. Click Install a New Web Application.
4. Browse to the location in your file system of the `.war` file.
5. Click Upload. The Web Application is copied to the `applications` directory on the Administration Server and deployed.

Redeploying a Web Application Using Auto-Deployment

When you modify static components of a Web Application (such as a JSPs and HTML pages) that are deployed in the `applications` directory, and you are using auto-deployment, the Web Application must be redeployed in order for the changes to become effective. The procedure is different for Web Applications deployed as `.war` archive files and Web Applications deployed in exploded directory format.

Redeploying a Web Application in a `.war` Archive

Modifying the archive file automatically triggers redeployment of the Web Application. If an auto-deployed Web Application is targeted to any Managed Servers, the Web Application is also redeployed on the Managed Servers.

Redeploying a Web Application in Exploded Directory Format

You can redeploy a Web Application deployed in exploded directory format when using auto-deployment by modifying a special file called `REDEPLOY`, or you can use the Administration Console, or you can cause a partial redeploy by copying a new version of a class file over an old in the `WEB-INF/classes` directory.

Touching the REDEPLOY File

To re-deploy a Web Application by modifying the `REDEPLOY` file:

1. Create an empty file called `REDEPLOY` and place it in the `WEB-INF` directory of your Web Application. (You may have to create this directory.)
2. Modify the `REDEPLOY` file by opening it, modifying the contents (adding a space character is the easiest way to do this), and then saving the file. Alternately, on UNIX machines, you can use the `touch` command on the `REDEPLOY` file. For example:

```
touch
user_domains/mydomain/applications/DefaultWebApp/WEB-INF/REDEPL
OY
```

As soon as the `REDEPLOY` file is modified, the Web Application is redeployed.

Redeploying with the Administration Console

To redeploy a Web Application using the Administration Console:

1. Expand the Deployments node in the left pane.
2. Select the Web Application node.
3. Select the Web Application you want to redeploy.
4. Deselect the Deployed box in the right-hand pane.
5. Click Apply.
6. Check the Deployed box in the right-hand pane.
7. Click Apply.

Hot-Deployment

Redeploy files in the WEB-INF/classes directory in the following way. If a class is deployed in WEB-INF/classes, then simply copying a new version of the file with a later time stamp will cause the Web Application to reload everything in the WEB-INF/classes folder with a new classloader.

The frequency in which WLS will look at the filesystem is governed through the console. In the Deployments-->Web Applications tab, select your Web Application. Go to the Configuration tab, and Files subtab, and enter a value in seconds for the Reload Period.

Deploying Web Applications with Production Mode Enabled

To deploy a Web Application with Production Mode enabled, you must start WebLogic Admin Server with the `-DProductionModeEnabled=true` flag. WebLogic Admin Server can use this feature; managed servers cannot. For more information, see [Starting and Stopping WebLogic Servers](http://e-docs.bea.com/wls/docs61/adminguide/startstop.html) at <http://e-docs.bea.com/wls/docs61/adminguide/startstop.html>.

You can deploy a Web Application using either the Administration Console or the `weblogic.deploy` utility. When you deploy a Web Application it is automatically deployed to all of the targeted Managed Servers in the domain. (For more information on administering Managed Servers, see [Configuring WebLogic Servers and Clusters](http://e-docs.bea.com/wls/docs61/adminguide/config.html) at <http://e-docs.bea.com/wls/docs61/adminguide/config.html>.)

It is not advisable to store applications in the `config/mydomain/applications` directory. In production mode, WebLogic Server will not detect applications newly placed in the `config/mydomain/applications` directory, but applications that were detected in the directory before switching to production mode will continue to deploy automatically.

These deployment instructions presume that you have created a functional Web Application using the correct directory structure and containing the `web.xml` deployment descriptor and, if needed, the `weblogic.xml` deployment descriptor. You can deploy your Web Application as an archived `.war` file or in exploded directory format. For more information, see “[Directory Structure](#)” on page 1-3.

Deploying a Web Application Using the Administration Console

To deploy a Web Application using the Administration Console:

1. Start your WebLogic Server Administration Server.
2. Start the WebLogic Server Administration Console. For more information, see [Overview of WebLogic Server Management at `http://e-docs.bea.com/wls/docs61/adminguide/overview.html`](http://e-docs.bea.com/wls/docs61/adminguide/overview.html)
3. Expand the Deployments node in the left pane.
4. Right-click on the Web Applications node.
5. Select Configure a New WebAppComponent.
6. Enter the full path to the exploded directory or `.war` file in the Path URI field. For example:

(exploded directory format) `c:\myApps\myWebApp`

(archived format) `c:\myApps\myWebApp.war`
7. Click Create.
8. Select the new Web Application from the Web Applications node in the left pane.
9. Select the Targets tab in the right pane.
10. Select the Servers tab and move the appropriate servers to the Chosen box.
11. If you are using a cluster of WebLogic Servers, select the Clusters tab and move the appropriate clusters to the Chosen box.

12. If you are using virtual hosting, select the Virtual Hosts tab and target any virtual hosts that should apply to this Web Application. For more information, see [Configuring Virtual Hosting at `http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#virtual_hosting`](http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#virtual_hosting).
13. Select the configuration tab and configure applicable attributes on the General, Files, and Other tabs. You can click on the question mark icons to get more information about these attributes.
14. Test your Web Application by accessing a resource through a Web browser. You access resources with a URL constructed as follows:

`http://myServer:myPort/myWebApp/resource`

Where:

- *myServer* is the name of the machine hosting WebLogic Server
- *myPort* is the port number where WebLogic Server is listening for requests
- *myWebApp* is the name of the Web Application archive file (*myWebApp.war*, for instance) or the name of a directory containing the Web Application.
- *resource* is the name of a resource such as a JSP, HTTP servlet, or HTML page.

Deploying a Web Application Using the `weblogic.deploy` Utility

To deploy a Web Application using the `weblogic.deploy` utility:

1. Set up your local environment so that WebLogic Server classes are in your system CLASSPATH and the JDK is available. You can use the `setEnv` script located in the `config/mydomain` directory to set the CLASSPATH.
2. Enter the following command:

```
% java weblogic.deploy -port port_number -host host_name  
    -component application:target deploy password application  
source
```

Where:

- *host_name* is the name of the machine hosting WebLogic Server
- *port_number* is the port number where WebLogic Server is listening for requests
- *application* is the name you want to assign to this Web Application.
- *target* is the name of a server, cluster or virtual host to be targeted by this Web Application. You can enter multiple targets, separated by a comma.
- *password* is your system administration password
- *source* is the full pathname of the `.war` file you want to deploy, or the full pathname of a directory containing a Web Application in exploded directory format.

For example:

```
java weblogic.deploy -port 7001 -host myhost -component  
myWebApp:myserver deploy pswd1234 myWebApp d:\myWebApp.war
```

Redeploying a Web Application with Production Mode Enabled

To redeploy a Web Application with Production Mode enabled, you must start WebLogic Admin Server with the `-DProductionModeEnabled=true` flag. WebLogic Admin Server can use this feature; managed servers cannot. For more information, see [Starting and Stopping WebLogic Servers](http://e-docs.bea.com/wls/docs61/adminguide/startstop.html) at <http://e-docs.bea.com/wls/docs61/adminguide/startstop.html>.

When you modify a component (for instance, a servlet, JSP, or HTML page) of a Web Application on the Administration Server, you must take additional steps to refresh the modified component so that it is also deployed on any targeted Managed Servers. One way to refresh a component is to redeploy the entire Web Application. Redeploying the Web Application means that the entire Web Application (not just the modified component) is re-sent over the network to all of the Managed Servers targeted by that Web Application.

Note the following regarding redeployment of Web Applications:

- Depending on your environment, there may be performance implications due to increased network traffic when a Web Application is re-sent to the Managed Servers.
- If the Web Application is currently in production and in use, redeploying the Web Application causes WebLogic Server to lose all active HTTP sessions for current users of the Web Application.
- If you have updated any Java class files, you must redeploy the entire Web Application to refresh the class.
- If you change the deployment descriptors, you must redeploy the Web Application.

If you need to refresh static files such as JSPs, HTML files, image files, you can use the refresh option of the `weblogic.deploy` utility to refresh the static file. For more information, see [“Refreshing Static Components Without Redeploying the Application” on page 2-13](#).

Redeploying a Web Application Using the Administration Console

To redeploy a Web Application through the Administration Console:

1. Start the WebLogic Server Administration Console.
2. Expand the deployments node in the left pane.
3. Expand the Web Application node.
4. Select the Web Application you want to redeploy.
5. Deselect the Deployed box in the right-hand pane.
6. Click Apply.
7. Select the Deployed box in the right-hand pane.
8. Click Apply.

Note: When you redeploy a Web Application, any active HTTP sessions are lost.

Redeploying a Web Application Using the weblogic.deploy Utility

Enter the following command:

```
% java weblogic.deploy -port port_number -host host_name  
    update password application source
```

Where:

- *host_name* is the name of the machine hosting WebLogic Server.
- *port_number* is the port number where WebLogic Server is listening for requests.
- *password* is your system administration password.
- *application* is the name of the Web Application.
- *source* is the full pathname of the `.war` file you want to redeploy, or the full pathname of a directory containing a Web Application in exploded directory format.

Notes: When you redeploy a Web Application, any active HTTP sessions are lost.

Updating an application on any single server instance to which it is targeted causes it to be updated on all servers to which is targeted. For instance, if an application is targeted to a cluster, and you update it on one of the clustered servers instances, the application will be updated on all members of cluster. Similarly, if the application is targeted to a cluster and to a standalone server instance, updating it on the standalone server instance will result in its update on the cluster, and vice versa.

Partial Redeployment

Refreshing Static Components Without Redeploying the Application

To refresh static components of an application without having to redeploy the entire application, use the `weblogic.refresh` tool.

When you update a static file in a Web Application on the Administration Server, the file is not automatically copied to the Web Application on any Managed Servers, but you can use `weblogic.refresh` to refresh, add, or delete static files such as:

- JSPs
- XML files
- HTML files
- Image files, such as `.gif` and `.jpg`
- Text files

Refreshing JSPs does not require redeploying the Web Application because each JSP is loaded by its own classloader, of which the Web Application classloader is a parent.

Note the following limitations of this utility:

- You cannot use this utility to refresh Java class files.
- To use this utility, *the Web Application must be deployed in exploded directory format, and if it is contained in an ear file, the ear file must be in exploded format.* The utility does not work for components archived in `.war` files.

To refresh a static file:

1. Set up your development environment so that WebLogic Server classes are in your system CLASSPATH and the JDK is available. You can use the `setEnv` script located in the `config/mydomain` directory to set your environment.
2. Enter a command in the following form:

```
% java weblogic.refresh -url -username -password -application  
-component -files -delete -root
```

Where:

- *url* is the URL of your WebLogic Administration Server.
- *username* is the username for system administration.
- *password* is your system administration password.
- *application* is the name of an Enterprise Application that contains the Web Application being refreshed. If your Web Application is not part of an Enterprise Application, enter the name of the Web Application.
- *component* is the name of the Web Application being refreshed.
- *files* is a comma-separated list of files to be refreshed. The files can be specified using wildcards, e.g., *.jsp or *.gif. Note that to specify multiple files you cannot use both wildcards and multiple comma-separated files. Filenames must be relative filenames to the root of the Web Application, so if the file is ball.gif and is contained in a subdirectory \foo of the Web Application root directory, the filename is foo\ball.gif. If a file does not exist on the server, it will be created, along with any specified subdirectories.
- if *delete* is set, named files will be deleted.
- *root* is the directory from which files should be picked up for refresh. Default is the current directory.

For example, the following command refreshes the files HelloWorld.jsp and foo\ball.gif in the myWebApp Web Application:

```
java weblogic.refresh -url t3://localhost:7001 -username  
myUsername -password myPassword -application myApplication  
-component myWebApp -root c:\stagedir\myWebApp  
HelloWorld.jsp,foo\ball.gif
```

Un-deploying Web Applications

Un-deploying a Web Application leaves your configuration intact, but makes the Web Application unavailable to respond to client requests. To un-deploy a Web Application you can use the Administration Console or the `weblogic.deploy` utility:

Un-deploying a Web Application Using the Administration Console

To un-deploy a Web Application using the Administration Console:

1. Start the Administration Console.
2. Expand the Deployments node in the left pane.
3. Expand the Web Applications node.
4. Select the Web Application you want to un-deploy.
5. Select the Configuration tab in the right pane.
6. Deselect the Deployed box.
7. Click Apply.

Un-deploying a Web Application Using the `weblogic.deploy` Utility

To un-deploy a Web Application using the `weblogic.deploy` utility:

1. Set up your development environment so that WebLogic Server classes are in your system `CLASSPATH` and the JDK is available. You can use the `setEnv` script located in the `config/mydomain` directory to set the `CLASSPATH`.
2. Enter the following command:

```
% java weblogic.deploy -port port_number -host host_name  
undeploy password application source
```

Where:

- *host_name* is the name of the machine hosting WebLogic Server.
- *port_number* is the port number where WebLogic Server is listening for requests.
- *password* is your system administration password.
- *application* is the name of the Web Application.
- *source* is the full pathname of the `.war` file you want to redeploy, or the full pathname of a directory containing a Web Application in exploded directory format.

Deleting a Web Application

Deleting a Web Application removes all of the configuration information of your Web Application from your domain configuration and makes the Web Application unavailable to respond to client requests. You can delete a Web Application using the Administration Console or by using the `weblogic.deploy` utility.

Deleting a Web Application does *not* delete the Web Application from your file system.

Deleting a Web Application in the applications Directory

If you are running WebLogic Server in development mode (you started WebLogic Server with the `-DProductionModeEnabled=false` flag), to delete a Web Application in the `applications` directory, you must physically delete the directory or archive containing the Web Application from the `applications` directory.

Deleting a Web Application Using the Administration Console

To delete a Web Application using the Administration Console:

1. Start the WebLogic Server Administration Console. For more information, see [Overview of WebLogic Server Management](http://e-docs.bea.com/wls/docs61/adminguide/overview.html) at <http://e-docs.bea.com/wls/docs61/adminguide/overview.html>
2. Expand the Deployments node in the left pane.
3. Select the Web Applications node.
4. In the right pane, click the trash can icon next to the name of the Web Application you want to delete. You are prompted to confirm the deletion.
5. Click Yes.

Deleting a Web Application Using the `weblogic.deploy` Utility

To delete a Web Application using the `weblogic.deploy` utility:

1. Set up your development environment so that WebLogic Server classes are in your system `CLASSPATH` and the JDK is available. You can use the `setEnv` script located in the `config/mydomain` directory to set the `CLASSPATH`.
2. Enter the following command:

```
% java weblogic.deploy -port port_number -host host_name  
    delete password application
```

Where:

- `host_name` is the name of the machine hosting WebLogic Server
- `port_number` is the port number where WebLogic Server is listening for requests.
- `password` is your system administration password.
- `application` is the name of the Web Application.

Deploying Stand-Alone Web Applications

If your Web Application is stand-alone, you can specify its context root in the `weblogic.xml` file using the `context-root` element. If the Web Application is part of an EAR, specify the context root in the EAR's `application.xml` file. A `context-root` setting in `application.xml` takes precedence over `context-root` setting in `weblogic.xml`. See [context-root Element at `http://e-docs.bea.com/wls/docs61/webapp/weblogic_xml.html`](http://e-docs.bea.com/wls/docs61/webapp/weblogic_xml.html).

3 Configuring Web Application Components

The following sections describe how to configure Web Application components:

- [“Configuring Servlets” on page 3-2](#)
- [“Configuring JSP” on page 3-5](#)
- [“Configuring JSP Tag Libraries” on page 3-6](#)
- [“Configuring Welcome Pages” on page 3-7](#)
- [“Setting Up a Default Servlet” on page 3-8](#)
- [“Customizing HTTP Error Responses” on page 3-9](#)
- [“Using CGI with WebLogic Server” on page 3-9](#)
- [“Serving Resources from the CLASSPATH with the ClasspathServlet” on page 3-12](#)
- [“Configuring External Resources in a Web Application” on page 3-13](#)
- [“Referencing EJBs in a Web Application” on page 3-15](#)
- [“Determining the Encoding of an HTTP Request” on page 3-16](#)
- [“Mapping IANA Character Sets to Java Character Sets” on page 3-17](#)

Configuring Servlets

Servlets are registered and configured as a part of a Web Application. To register a servlet, you add several entries to the Web Application deployment descriptor. The first entry, under the `<servlet>` element, defines a name for the servlet and specifies the compiled class that executes the servlet. (Or, instead of specifying a servlet class, you can specify a JSP page.) This element also contains definitions for initialization parameters and security roles for the servlet. The second entry, under the `<servlet-mapping>` element, defines the URL pattern that calls this servlet.

For complete instructions on editing the Web Application deployment descriptor, see:

- [“Step 9: Deploy Servlets” on page 8-9](#)
- [“Step 10: Map a servlet to a URL” on page 8-12](#)

Servlet Mapping

Servlet mapping controls how you access a servlet. The following examples demonstrate how you can use servlet mapping in your Web Application. In the examples, a set of servlet configurations and mappings (from the `web.xml` deployment descriptor) is followed by a table (see [“url-patterns and Servlet Invocation” on page 3-3](#)) showing the URLs used to invoke these servlets.

Listing 3-1 Servlet Mapping Example

```
<servlet>
  <servlet-name>watermelon</servlet-name>
  <servlet-class>myservlets.watermelon</servlet-class>
</servlet>

<servlet>
  <servlet-name>garden</servlet-name>
  <servlet-class>myservlets.garden</servlet-class>
</servlet>

<servlet>
  <servlet-name>list</servlet-name>
```



```
<servlet-class>myservlets.list</servlet-class>
</servlet>

<servlet>
  <servlet-name>kiwi</servlet-name>
  <servlet-class>myservlets.kiwi</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>watermelon</servlet-name>
  <url-pattern>/fruit/summer/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>garden</servlet-name>
  <url-pattern>/seeds/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>list</servlet-name>
  <url-pattern>/seedlist</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>kiwi</servlet-name>
  <url-pattern>*.abc</url-pattern>
</servlet-mapping>
```

Table 3-1 url-patterns and Servlet Invocation

URL	Servlet Invoked
http://host:port/mywebapp/fruit/summer/index.html	watermelon
http://host:port/mywebapp/fruit/summer/index.abc	watermelon
http://host:port/mywebapp/seedlist	list

Table 3-1 url-patterns and Servlet Invocation

URL	Servlet Invoked
<code>http://host:port/mywebapp/seedlist/index.html</code>	The default servlet, if configured, or an HTTP 404 File Not Found error message. If the mapping for the <code>list</code> servlet had been <code>/seedlist*</code> , the <code>list</code> servlet would be invoked.
<code>http://host:port/mywebapp/seedlist/pear.abc</code>	<code>kiwi</code> If the mapping for the <code>list</code> servlet had been <code>/seedlist*</code> , the <code>list</code> servlet would be invoked.
<code>http://host:port/mywebapp/seeds</code>	<code>garden</code>
<code>http://host:port/mywebapp/seeds/index.html</code>	<code>garden</code>
<code>http://host:port/mywebapp/index.abc</code>	<code>kiwi</code>

Servlet Initialization Parameters

You define initialization parameters for servlets in the Web Application deployment descriptor, in the `<init-param>` element of the `<servlet>` element, using `<param-name>` and `<param-value>` tags. For example:

Listing 3-2 Example of Configuring Servlet Initialization Parameters

```
<servlet>
  <servlet-name>HelloWorld2</servlet-name>
  <servlet-class>examples.servlets.HelloWorld2</servlet-class>

  <init-param>
    <param-name>greeting</param-name>
    <param-value>Welcome</param-value>
  </init-param>

  <init-param>
    <param-name>person</param-name>
    <param-value>WebLogic Developer</param-value>
  </init-param>
</servlet>
```

For more information on editing the Web Application deployment descriptor, see [“Writing Web Application Deployment Descriptors” on page 8-1](#).

Configuring JSP

You deploy JavaServer Pages (JSP) files by placing them in the root (or in a subdirectory below the root) of a Web Application. Additional JSP configuration parameters are defined in the `<jsp-descriptor>` element of the WebLogic-specific deployment descriptor, `weblogic.xml`. These parameters define the following functionality:

- Options for the JSP compiler
- Debugging
- How often WebLogic Server checks for updated JSPs that need to be recompiled
- Character encoding

For a complete description of these parameters, see [“JSP Parameter Names and Values” on page B-11](#).

For instructions on editing the `weblogic.xml` file, see “Main Steps to Create the `weblogic.xml` File” on page 8-24.

You can also register a JSP as a servlet using the `<servlet>` tag. In this example a URL containing `/main` will invoke `myJSPfile.jsp`:

```
<servlet>
    <servlet-name>myFoo</servlet-name>
    <jsp-file>myJSPfile.jsp</jsp-file>
</servlet>

<servlet-mapping>
    <servlet-name>myFoo</servlet-name>
    <url-pattern>/main</url-pattern>
</servlet-mapping>
```

Registering a JSP in this manner allows you to specify the load order, initialization parameters, and security roles for a JSP, just as you would for a servlet.

Configuring JSP Tag Libraries

Weblogic Server provides the ability to create and use custom JSP tags. Custom JSP tags are Java classes that can be called from within a JSP page. To create custom JSP tags, you place them in a tag library and define their behavior in a tag library descriptor (TLD) file. This TLD must be made available to the Web Application containing the JSP by defining it in the Web Application deployment descriptor. It is a good idea to place the TLD file in the `WEB-INF` directory of your Web Application, because that directory is never available publicly.

In the Web Application deployment descriptor, you define a URI pattern for the tag library. This URI pattern must match the value in the `taglib` directive in your JSP pages. You also define the location of the TLD. For example, if the `taglib` directive in the JSP page is:

```
<%@ taglib uri="myTaglib" prefix="taglib" %>
```

and the TLD is located in the `WEB-INF` directory of your Web Application, you would create the following entry in the Web Application deployment descriptor:

```
<taglib>
    <taglib-uri>myTaglib</taglib-uri>
```

```
<taglib-location>WEB-INF/myTLD.tld</taglib-location>
</taglib>
```

You can also deploy a tag library as a . jar file. For more information, see [Deploying a JSP Tag Library as a JAR File](#) at

<http://e-docs.bea.com/wls/docs61/taglib/admin.html#jar>.

For more information on creating custom JSP tag libraries, see [Programming JSP Tag Extensions](#) at <http://e-docs.bea.com/wls/docs61/jsp/index.html>.

WebLogic Server also includes several custom JSP tags that you can use in your applications. These tags perform caching, facilitate query parameter-based flow control, and facilitate iterations over sets of objects. For more information, see:

- [Using Custom WebLogic JSP Tags](#) at <http://e-docs.bea.com/wls/docs61/taglib/customtags.html>.
- [Using WebLogic JSP Form Validation Tags](#) at http://e-docs.bea.com/wls/docs61/jsp/validation_tags.html.

Configuring Welcome Pages

WebLogic Server allows you to set a page that is served by default if the requested URL is a directory. This feature can make your site easier to use, because the user can type a URL without giving a specific filename.

Welcome pages are defined at the Web Application level. If your server is hosting multiple Web Applications, you need to define welcome pages separately for each Web Application.

To define Welcome pages, edit the Web Application deployment descriptor, `web.xml`. For more information, see [“Step 13: Define welcome pages” on page 8-14](#)

If you do not define Welcome Pages, WebLogic Server looks for the following files in the following order and serves the first one it finds:

1. `index.html`
2. `index.htm`
3. `index.jsp`

For more information, see [How WebLogic Server Resolves HTTP Requests at `http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req`](http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req).

Setting Up a Default Servlet

Each Web Application has a *default servlet*. This default servlet can be a servlet that you specify, or, if you do not specify a default servlet, WebLogic Server uses an internal servlet called the `FileServlet` as the default servlet. For more information on the `FileServlet`, see [How WebLogic Server Resolves HTTP Requests at `http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req`](http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req).

You can register any servlet as the default servlet. Writing your own default servlet allows you to use your own logic to decide how to handle a request that falls back to the default servlet.

Setting up a default servlet replaces the `FileServlet` and should be done carefully because the `FileServlet` is used to serve most files, such as text files, HTML file, image files, and more. If you expect your default servlet to serve such files, you will need to write that functionality into your default servlet.

To set up a user-defined default servlet:

1. Define your servlet as described in [Configuring Servlets on page 3-2](#).
2. Map your default servlet with a url-pattern of `"/`". This causes your default servlet to respond to all types of files except for those with extensions of `*.htm` or `*.html`, which are internally mapped to the `FileServlet`.

If you also want your default servlet to respond to files ending in `*.htm` or `*.html`, then you must map those extensions to your default servlet, in addition to mapping `"/`". For instructions on mapping servlets, see [Configuring Servlets on page 3-2](#).

3. If you still want the `FileServlet` to serve files with other extensions:
 - a. Define a servlet and give it a `<servlet-name>`, for example `myFileServlet`.
 - b. Define the `<servlet-class>` as `weblogic.servlet.FileServlet`.

- a. Using the `<servlet-mapping>` element, map file extensions to the `myFileServlet` (in addition to the mappings for your default servlet). For example, if you want the `myFileServlet` to serve `gif` files, map `*.gif` to the `myFileServlet`.

Note: The `FileServlet` includes the `SERVLET_PATH` when determining the source filename if `docHome` is not specified. As a result, it is possible to explicitly server only files from specific directories by mapping the `FileServlet` to `/dir/*`, etc.

Customizing HTTP Error Responses

You can configure WebLogic Server to respond with your own custom Web pages or other HTTP resources when particular HTTP errors or Java exceptions occur, instead of responding with the standard WebLogic Server error response pages.

You define custom error pages in the `<error-page>` element of the Web Application deployment descriptor (`web.xml`). For more information on error pages, see [“error-page Element” on page A-15](#).

Using CGI with WebLogic Server

WebLogic Server provides functionality to support your legacy Common Gateway Interface (CGI) scripts. For new projects, we suggest you use HTTP servlets or JavaServer Pages.

WebLogic Server supports all CGI scripts through an internal WebLogic servlet called the `CGIServlet`. To use CGI, register the `CGIServlet` in the Web Application deployment descriptor (see [“Sample Web Application Deployment Descriptor Entries for Registering the CGIServlet” on page 3-11](#)). For more information, see [“Configuring Servlets” on page 3-2](#).

Note: It is important to note that a CGI script cannot be served by `CGIServlet` if the script filename does not have an extension. Filename extensions are required.

Configuring WebLogic Server to use CGI

To configure CGI in WebLogic Server:

1. Declare the `CGIServlet` in your Web Application by using the `<servlet>` and `<servlet-mapping>` elements. The class name for the `CGIServlet` is `weblogic.servlet.CGIServlet`. You do not need to package this class in your Web Application.
2. Register the following initialization parameters for the `CGIServlet` by defining the following `<init-param>` elements:

`cgiDir`

The path to the directory containing your CGI scripts. You can specify multiple directories, separated by a “;” (Windows) or a “:” (Unix). Specify `cgiDir` if you want your CGI directory to be relative to the app-root or located inside a WAR file. Note the following about `cgiDir`:

You must specify each and every directory that contains a CGI script so that WebLogic Server can extract it.

The `CGIServlet` does not extract subdirectories from `cgiDir` paths.

All CGI script names must be unique regardless of subdirectory location since they are all extracted into the same directory and/or they are searched for using the script name and not their full or relative path.

For example: `/foo/myscript.pl` and `/bar/myscript.pl` will be confused and it is undetermined which one will be invoked.

`useByteStream`

The alternate to using the default `Char` streams for data transfer, this parameter, which is case sensitive, will allow the use of images in the CGI servlet without distortion.

extension mapping

Maps a file extension to the interpreter or executable that runs the script. If the script does not require an executable, this initialization parameter may be omitted.

The `<param-name>` for extension mappings must begin with an asterisk followed by a dot, followed by the file extension, for example, `*.pl`.

The `<param-value>` contains the path to the interpreter or executable that runs the script. You can create multiple mappings by creating a separate `<init-param>` element for each mapping.

Listing 3-3 Sample Web Application Deployment Descriptor Entries for Registering the CGIServlet

```
<servlet>
  <servlet-name>CGIServlet</servlet-name>
  <servlet-class>weblogic.servlet.CGIServlet</servlet-class>
  <init-param>
    <param-name>cgiDir</param-name>
    <param-value>
      /bea/wlserver6.0/config/mydomain/applications/myWebApp/cgi-bin
    </param-value>
  </init-param>

  <init-param>
    <param-name>*.pl</param-name>
    <param-value>/bin/perl.exe</param-value>
  </init-param>
</servlet>

...

<servlet-mapping>
  <servlet-name>CGIServlet</servlet-name>
  <url-pattern>/cgi-bin/*</url-pattern>
</servlet-mapping>
```

Requesting a CGI Script

The URL used to request a perl script must follow the pattern:

`http://host:port/myWebApp/cgi-bin/myscript.pl`

Where

`host:port`

Is the host name and port number of WebLogic Server.

myWebApp

is the name of your Web Application.

cgi-bin

is the url-pattern name mapped to the CGIServlet.

myscript.pl

is the name of the Perl script that is located in the directory specified by the *cgiDir* initialization parameter.

Serving Resources from the CLASSPATH with the ClasspathServlet

If you need to serve classes or other resources from the system CLASSPATH, or from the WEB-INF/classes directory of a Web Application, you can use a special servlet called the ClasspathServlet. The ClasspathServlet is useful for applications that use applets or RMI clients and require access to server-side classes. The ClasspathServlet is implicitly registered and available from any application.

The general form for invoking the ClasspathServlet is:

```
http://server:port[/<context-path>]/classes[/<app>@[<webapp.>]]/  
<package>.<classname>.<class>
```

where

context-path

specifies the application for which ClasspathServlet is invoked.

app

is the name of the application that contains the classes (or contains the Web application that contains the classes) to be served.

webapp

is the name of the Web application that contains the classes.

There are two ways that you can use the ClasspathServlet:

- To serve a resource from the system CLASSPATH, call the resource with a URL such as:

```
http://server:port/classes/weblogic/myClass.class
```

- To serve a resource from the `WEB-INF/classes` directory of a Web Application, call the resource with a URL such as:

```
http://server:port/classes/myApp@myWebApp/examples/servlets/myClass.class
```

In this case, the resource is located in the following directory, relative to the root of the Web Application:

```
WEB-INF/classes/my/resource/myClass.class
```

`disableStrictCheck` relaxes the restriction of the download file type. When `disableStrictCheck` is set to `false`, `ClasspathServlet` only allows downloads of `.class` files, `.war` files, `.jar` files, `.ear` files, `.rar` files and `.mf` files.

To access an applet or another similar file, set `disableStrictCheck` to `true`.

Warning: Since the `ClasspathServlet` serves any resource located in the system `CLASSPATH`, do not place resources that should not be publicly available in the system `CLASSPATH`.

Configuring External Resources in a Web Application

When accessing external resources, such as a `DataSource` from a Web Application via JNDI, you can map the JNDI name you look up in your code to the actual JNDI name as bound in the JNDI tree. This mapping is made using both the `web.xml` and `weblogic.xml` deployment descriptors and allows you to change these resources without changing your application code. You provide a name that is used in your Java code, the name of the resource as bound in the JNDI tree, and the Java type of the resource, and you indicate whether security for the resource is handled programmatically by the servlet or from the credentials associated with the HTTP request.

To configure external resources:

1. Enter the resource name in the deployment descriptor as you use it in your code, the Java type, and the security authorization type. For instructions on making deployment descriptor entries, see [“Step 16: Reference external resources” on page 8-16](#).
2. Map the resource name to the JNDI name. For instructions on making deployment descriptor entries, see [“Step 3 Map resources to JNDI” on page 8-26](#).

This example assumes that you have defined a data source called `accountDataSource`. For more information, see [JDBC Data Sources at `http://e-docs.bea.com/wls/docs61/ConsoleHelp/jdbcdatasource.html`](http://e-docs.bea.com/wls/docs61/ConsoleHelp/jdbcdatasource.html).

Listing 3-4 Example of Using a DataSource

Servlet code:

```
javax.sql.DataSource ds = (javax.sql.DataSource) ctx.lookup  
    ( "myDataSource" );
```

web.xml entries:

```
<resource-ref>  
. . .  
    <res-ref-name>myDataSource</res-ref-name>  
    <res-type>javax.sql.DataSource</res-type>  
    <res-auth>CONTAINER</res-auth>  
. . .  
</resource-ref>
```

weblogic.xml entries:

```
<resource-description>  
    <res-ref-name>myDataSource</res-ref-name>  
    <jndi-name>accountDataSource</jndi-name>  
</resource-description>
```

Referencing EJBs in a Web Application

You can reference EJBs in a Web Application by giving them a name in the Web Application deployment descriptor that is mapped to the JNDI name for the EJB that is defined in the `weblogic-ejb-jar.xml` file deployment descriptor. This procedure provides a level of indirection between the Web Application and an EJB and is useful if you are using third-party EJBs or Web Applications and cannot modify the code to directly call an EJB. In most situations, you can call the EJB directly without using this indirection. For more information, see [Invoking Deployed EJBs at `http://e-docs.bea.com/wls/docs61/ejb/EJB_design.html#design_invoking`](http://e-docs.bea.com/wls/docs61/ejb/EJB_design.html#design_invoking).

To reference an EJB for use in a Web Application:

1. Enter the EJB reference name you use to look up the EJB in your code, the Java class name, and the class name of the home and remote interfaces of the EJB in the `<ejb-ref>` element of the Web Application deployment descriptor. For instructions on making deployment descriptor entries, see [“Step 21: Reference Enterprise JavaBean \(EJB\) resources” on page 8-21](#)
2. Map the reference name in `<ejb-reference-description>` element of the WebLogic-specific deployment descriptor, `weblogic.xml`, to the JNDI name defined in the `weblogic-ejb-jar.xml` file. For instructions on making deployment descriptor entries, see [“Step 3 Map resources to JNDI” on page 8-26](#).

If the Web Application is part of an Enterprise Application Archive (`.ear` file), you can reference an EJB by the name used in the `.ear` with the `<ejb-link>` element.

Determining the Encoding of an HTTP Request

WebLogic Server needs to convert character data contained in an HTTP request from its native encoding to the Unicode encoding used by the Java servlet API. In order to perform this conversion, WebLogic Server needs to know which codeset was used to encode the request.

There are two ways you can define the codeset:

- For a POST operation, you can set the encoding in the HTML `<form>` tag. For example, this form tag sets `SJIS` as the character set for the content:

```
<form action="http://some.host.com/myWebApp/foo/index.html">  
  <input type="application/x-www-form-urlencoded; charset=SJIS">  
</form>
```

When the form is read by WebLogic Server, it processes the data using the `SJIS` character set.

- Because all Web clients do not transmit the information after the semicolon in the above example, you can set the codeset to be used for requests by using the `<input-charset>` element in the WebLogic-specific deployment descriptor, `weblogic.xml`. The `<java-charset-name>` element defines the encoding used to convert data when the URL of the request contains the path specified with the `<resource-path>` element.

For example:

```
<input-charset>  
  <resource-path>/foo/*</resource-path>  
  <java-charset-name>SJIS</java-charset-name>  
</input-charset>
```

This method works for both GET and POST operations.

For more information on the Web Application deployment descriptor, see [“Writing the WebLogic-Specific Deployment Descriptor \(weblogic.xml\)” on page 8-24](#).

Mapping IANA Character Sets to Java Character Sets

The names assigned by the Internet Assigned Numbers Authority (IANA) to describe character sets are sometimes different than the names used by Java. Because all HTTP communication uses the IANA character set names and these names are not always the same, WebLogic Server internally maps IANA character set names to Java character set names and can usually determine the correct mapping. However, you can resolve any ambiguities by explicitly mapping an IANA character set to the name of a Java character set.

To map a IANA character set to a Java character set the character set names in the `<charset-mapping>` element of the WebLogic-specific deployment descriptor, `weblogic.xml`. Define the IANA character set name in the `<iana-charset-name>` element and the Java character set name in the `<java-charset-name>` element. For example:

```
<charset-mapping>
  <iana-charset-name>Shift-JIS</iana-charset-name>
  <java-charset-name>SJIS</java-charset-name>
</charset-mapping>
```


4 Using Sessions and Session Persistence in Web Applications

The following sections describe how to set up sessions and session persistence:

- [“Overview of HTTP Sessions” on page 4-1](#)
- [“Setting Up Session Management” on page 4-2](#)
- [“Configuring Session Persistence” on page 4-4](#)
- [“Using URL Rewriting” on page 4-11](#)

Overview of HTTP Sessions

Session tracking enables you to track a user's progress over multiple servlets or HTML pages, which, by nature, are stateless. A *session* is defined as a series of related browser requests that come from the same client during a certain time period. Session tracking ties together a series of browser requests—think of these requests as pages—that may have some meaning as a whole, such as a shopping cart application.

Setting Up Session Management

WebLogic Server is set up to handle session tracking by default. You need not set any of these properties to use session tracking. However, configuring how WebLogic Server manages sessions is a key part of tuning your application for best performance. Tuning depends upon factors such as:

- How many users you expect to hit the servlet
- How many concurrent users hit the servlet
- How long each session lasts
- How much data you expect to store for each user
- Heap size allocated to the WebLogic Server instance.

HTTP Session Properties

You configure WebLogic Server session tracking with properties in the WebLogic-specific deployment descriptor, `weblogic.xml`. For instructions on editing the WebLogic-specific deployment descriptor, see [“Step 4: Define session parameters” on page 8-28](#).

For a complete list of session attributes, see [“session-descriptor Element” on page B-4](#).

Session Timeout

You can specify an interval of time after which HTTP sessions expire. When a session expires, all data stored in the session is discarded. You can set the interval in one of two ways:

- Set the `TimeoutSecs` attribute in the [“session-descriptor Element” on page B-4](#) of the WebLogic-specific deployment descriptor, `weblogic.xml`. This value is set in seconds.

- Set the `<session-timeout>` (see [“session-config Element” on page A-13](#)) element in the Web Application deployment descriptor, `web.xml`.

Configuring Session Cookies

WebLogic Server uses cookies for session management when supported by the client browser.

The cookies that WebLogic Server uses to track sessions are set as transient by default and do not outlive the life of the browser. When a user quits the browser, the cookies are lost and the session lifetime is regarded as over. This behavior is in the spirit of session usage and it is recommended that you use sessions in this way.

It is possible to configure many aspects of the cookies used to track sessions with attributes that are defined in the WebLogic-specific deployment descriptor, `weblogic.xml`. A complete list of session and cookie-related attributes is available [“session-descriptor Element” on page B-4](#).

For instructions on editing the WebLogic-specific deployment descriptor, see [“Step 4: Define session parameters” on page 8-28](#).

Using Longer-Lived Cookies

For longer-lived client-side user data, your application should create and set its own cookies on the browser via the HTTP servlet API, and should not attempt to use the cookies associated with the HTTP session. Your application might use cookies to auto-login a user from a particular machine, in which case you would set a new cookie to last for a long time. Remember that the cookie can only be sent from that client machine. Your application should store data on the server if it must be accessed by the user from multiple locations.

You cannot directly connect the age of a browser cookie with the length of a session. If a cookie expires before its associated session, that session becomes orphaned. If a session expires before its associated cookie, the servlet is not able to find a session. At that point, a new session is assigned when the `request.getSession(true)` method is called. You should only make transient use of sessions.

You can set the maximum life of a cookie with the `CookieMaxAgeSecs` parameter in the session descriptor of the `weblogic.xml` deployment descriptor. For more information, see [“Step 4: Define session parameters” on page 8-28](#).

Logging Out and Ending a Session

User authentication information is stored both in the users's session data and in the context of a server or virtual host that is targeted by a Web Application. Using the `session.invalidate()` method, which is often used to log out a user, only invalidates the current session for a user—the user's authentication information still remains valid and is stored in the context of the server or virtual host. If the server or virtual host is hosting only one Web Application, the `session.invalidate()` method, in effect, logs out the user.

There are several Java methods and strategies you can use when using authentication with multiple Web Applications. For more information, see [Implementing Single Sign-On](#) in the *Programming WebLogic HTTP Servlets*, at <http://e-docs.bea.com/wls/docs61/servlet/progtasks.html#sso>.

Configuring Session Persistence

Session Persistence allows the data stored in an HTTP session object to be permanently stored to enable failover and load balancing across a cluster of WebLogic Servers.

There are five different implementations of session persistence:

- Memory (single-server, non-replicated)
- File system persistence
- JDBC persistence
- Cookie-based session persistence
- In-memory replication (across a cluster)

The first four are covered here; in-memory replication is covered in [Understanding HTTP Session State Replication](http://e-docs.bea.com/wls/docs61/cluster/servlet.html) (at <http://e-docs.bea.com/wls/docs61/cluster/servlet.html>).

For file, JDBC, Cookie-based, and in-memory replication, you need to set additional attributes, including `PersistentStoreType`. Each method has its own set of attributes as shown below.

When your applications stores data in an HTTP session object, the data must be serializable.

Common Properties

You can configure the number of sessions that are held in memory by setting the following properties in the `<session-descriptor>` element of the WebLogic-specific deployment descriptor, `weblogic.xml`. These properties are only applicable if you are using session persistence:

`CacheSize`

Limits the number of cached sessions that can be active in memory at any one time. If you are expecting high volumes of simultaneous active sessions, you do not want these sessions to soak up the RAM of your server since this may cause performance problems swapping to and from virtual memory. When the cache is full, the least recently used sessions are stored in the persistent store and recalled automatically when required. If you do not use persistence, this property is ignored, and there is no soft limit to the number of sessions allowed in main memory. By default, the number of cached sessions is 1024. The minimum is 16, and maximum is `Integer.MAX_VALUE`. An empty session uses less than 100 bytes, but grows as data is added to it.

`SwapIntervalSecs`

The interval the server waits between purging the least recently used sessions from the cache to the persistent store, when the `cacheEntries` limit has been reached.

If unset, this property defaults to 10 seconds; minimum is 1 second, and maximum is 604800 (1 week).

`InvalidationIntervalSecs`

Sets the time, in seconds, that WebLogic Server waits between doing house-cleaning checks for timed-out and invalid sessions, and deleting the

old sessions and freeing up memory. Set this parameter to a value less than the value set for the `<session-timeout>` element. Use this parameter to tune WebLogic Server for best performance on high traffic sites.

The minimum value is every second (1). The maximum value is once a week (604,800 seconds). If unset, the parameter defaults to 60 seconds.

You set `<session-timeout>` in the “[session-config Element](#)” on page A-13 of the Web Application deployment descriptor `web.xml`.

Using Memory-based, Single-server, Non-replicated Persistent Storage

To use memory-based, single-server, non-replicated persistent storage, set the `PersistentStoreType` property in the `<session-descriptor>` element of the WebLogic-specific deployment descriptor, `weblogic.xml` to `memory`. When you use memory-based storage, all session information is stored in memory and is lost when you stop and restart WebLogic Server.

Note: If you do not allocate sufficient heap size when running WebLogic Server, your server may run out of memory under heavy load.

Using File-based Persistent Storage

For file-based persistent storage for sessions:

1. Set the `PersistentStoreType` property in the `<session-descriptor>` element of the WebLogic-specific deployment descriptor, `weblogic.xml` to `file`.
2. Set the directory where WebLogic Server stores the sessions. For more information on setting this directory, see “[PersistentStoreDir](#)” on page B-8.

If you do not explicitly set a value for this attribute, a temporary directory is created for you by WebLogic Server.

To use file-based persistent storage in a clustered environment (in a WebLogic cluster or in multiple WebLogic instances not in a WebLogic cluster):

- You must explicitly set this attribute to a shared directory that is accessible to all the servers in a cluster. You must create this directory yourself.
- You also need a front-end proxy server to maintain session stickiness. For a given client, the requests should be served by the same server. Otherwise, a problem occurs when multiple servers try to write the same storage.
- You must also set the `InvalidationIntervalSecs` attribute to a very high value in `weblogic.xml`.

Using a Database for Persistent Storage (JDBC persistence)

JDBC persistence stores session data in a database table using a schema provided for this purpose. You can use any database for which you have a JDBC driver. You configure database access by using connection pools.

To Configure JDBC-based persistent storage for sessions:

1. Set JDBC as the persistent store method by setting the `PersistentStoreType` property in the `<session-descriptor>` element of the WebLogic-specific deployment descriptor, `weblogic.xml` to `jdbc`.
2. Set a JDBC connection pool to be used for persistence storage with the `PersistentStorePool` property of the WebLogic-specific deployment descriptor, `weblogic.xml`. Use the name of a connection pool that is defined in the WebLogic Server Administration Console.

For more details on setting up a database connection pool, see [Managing JDBC Connectivity at](#)

<http://e-docs.bea.com/wls/docs61/adminguide/jdbc.html>.

3. Set an ACL for the connection that corresponds to the users that have permission. For more details on setting up a database connection, see [Managing JDBC Connectivity at](#)
<http://e-docs.bea.com/wls/docs61/adminguide/jdbc.html>.

4. Set up a database table named `wl_servlet_sessions` for JDBC-based persistence. The connection pool that connects to the database needs to have read/write access for this table. The following table shows the Column names and data types you should use when creating this table.

Table 4-1 wl_servlet_sessions table for JDBC-based Persistence

Column name	Type
wl_id	Variable-width alphanumeric column, up to 100 characters; for example, Oracle VARCHAR2(100). <i>The primary key must be set as follows:</i> wl_id + wl_context_path.
wl_context_path	Variable-width alphanumeric column, up to 100 characters; for example, Oracle VARCHAR2(100). <i>This column is used as part of the primary key. (See the wl_id column description.)</i>
wl_is_new	Single char column; for example, Oracle CHAR(1)
wl_create_time	Numeric column, 20 digits; for example, Oracle NUMBER(20)
wl_is_valid	Single char column; for example, Oracle CHAR(1)
wl_session_values	Large binary column; for example, Oracle LONG RAW
wl_access_time	Numeric column, 20 digits; for example, NUMBER(20)
wl_max_inactive_interval	Integer column; for example, Oracle Integer. Number of seconds between client requests before the session is invalidated. A negative time value indicates that the session should never timeout.

If you are using an Oracle DBMS, you can use the following SQL statement to create the wl_servlet_sessions table:

```
create table wl_servlet_sessions
( wl_id VARCHAR2(100) NOT NULL,
  wl_context_path VARCHAR2(100) NOT NULL,
  wl_is_new CHAR(1),
  wl_create_time NUMBER(20),
  wl_is_valid CHAR(1),
  wl_session_values LONG RAW,
  wl_access_time NUMBER(20),
  wl_max_inactive_interval INTEGER,
```



```
PRIMARY KEY (wl_id, wl_context_path) );
```

You can modify the proceeding SQL statement for use with your DBMS.

If you are using SqlServer2000, you can use the following SQL statement to create the `wl_servlet_sessions` table:

```
create table wl_servlet_sessions
( wl_id VARCHAR2(100) NOT NULL,
  wl_context_path VARCHAR2(100) NOT NULL,
  wl_is_new VARCHAR(1),
  wl_create_time DECIMAL,
  wl_is_valid VARCHAR(1),
  wl_session_values IMAGE,
  wl_access_time DECIMAL,
  wl_max_inactive_interval INTEGER,
  PRIMARY KEY (wl_id, wl_context_path) );
```

You can modify the proceeding SQL statement for use with your DBMS.

Note: You can configure a maximum duration that JDBC session persistence should wait for a JDBC connection from the connection pool before failing to load the session data with the `JDBCConnectionTimeoutSecs` attribute. For more information, see [“JDBCConnectionTimeoutSecs” on page B-10](#).

Using Cookie-Based Session Persistence

Cookie-based session persistence provides a stateless solution for session persistence by storing all session data in a cookie that is stored in the user’s browser. Cookie-based session persistence is most useful when you do not need to store large amounts of data in the session. Cookie-based session persistence can make managing your WebLogic Server installation easier because clustering failover logic is not required. Because the session is stored in the browser, not on the server. You can start and stop WebLogic Servers without losing sessions.

When using Cookie-based session persistence, requests are balanced on a per-request basis as opposed to a per-session basis. Balancing on a per-request basis is the correct behaviour, as cookie-based session persistence is completely stateless and it makes no difference as to which server handles the request (data is stored on the client side).

There are some limitations when using cookie-based session persistence:

- You can only store string attributes in the session. If you store any other type of object in the session, an `IllegalArgumentException` exception is thrown.
- You cannot flush the HTTP response (because the cookie must be written to the header data *before* the response is committed).
- If the content length of the response exceeds the buffer size, the response is automatically flushed and the session data cannot be updated in the cookie. (The buffer size is, by default, 8192 bytes. You can change the buffer size with the `javax.servlet.ServletResponse.setBufferSize()` method.
- You can only use basic (browser-based) authentication.
- Session data is sent to the browser in clear text.
- The user's browser must be configured to accept cookies.
- You cannot use commas (,) in a string when using cookie-based session persistence or an exception occurs.

To set up cookie-based session persistence:

1. Set the `PersistentStoreType` parameter in the `<session-descriptor>` element of the WebLogic-specific deployment descriptor, `weblogic.xml` to `cookie`.
2. If desired, set a name for the cookie using the `PersistentStoreCookieName` parameter. The default is `WLCOOKIE`.

Secure Cookies

As of Service Pack 3, BEA Systems introduced a new parameter that can be used to mark the `sessionCookie` as secure. If set, the client's browser will only send the cookie back over an HTTPS connection. This ensures that the cookie ID is secure and should only be used on websites that exclusively use HTTPS. Once this feature is enabled, session cookies over HTTP will no longer work; if your client is directed to a non-HTTPS location the session will not be sent. It is highly advisable to turn off URLRewriting if you use this feature; if your application attempts to encode URLs, the session ID will be shared over HTTP. To use this feature, add the following to your `weblogic.xml`:

```
<session-param>
<param-name>CookieSecure</param-name>
<param-value>true</param-value>
</session-param>
<session-param>
```

As of Service Pack 5, BEA Systems added a new capability to WebLogic Server that allows a user to securely access HTTPS resources in a session that was initiated using HTTP, without loss of session data. For more information on this new feature, see “Multiple Web Applications, Cookies, and Authentication” on page 5-3.

Using URL Rewriting

In some situations, a browser or wireless device may not accept cookies, which makes session tracking using cookies impossible. URL rewriting is a solution to this situation that can be substituted automatically when WebLogic Server detects that the browser does not accept cookies. URL rewriting involves encoding the session ID into the hyper-links on the Web pages that your servlet sends back to the browser. When the user subsequently clicks these links, WebLogic Server extracts the ID from the URL address and finds the appropriate `HttpSession` when your servlet calls the `getSession()` method.

To enable URL rewriting in WebLogic Server, set the “[URLRewritingEnabled](#)” on [page B-10](#) attribute to `true`. You set the `URLRewritingEnabled` attribute in the WebLogic-specific deployment descriptor, `weblogic.xml`, under the `<session-descriptor>` element. (The default value for this attribute is `true`.)

Coding Guidelines for URL Rewriting

There are some general guidelines for how your code should handle URLs in order to support URL rewriting.

- Avoid writing a URL straight to the output stream, as shown here:

```
out.println("<a href=\" /myshop/catalog.jsp\">catalog</a>");
```

Instead, use the `HttpServletResponse.encodeURL()` method, for example:

```
out.println("<a href=\""+
    + response.encodeURL("myshop/catalog.jsp")
    + "\">catalog</a>");
```

Calling the `encodeURL()` method determines if the URL needs to be rewritten, and if so, it rewrites it by including the session ID in the URL. The session ID is appended to the URL and begins with a semicolon.

- In addition to URLs that are returned as a response to WebLogic Server, also encode URLs that send redirects. For example:

```
if (session.isNew())
    response.sendRedirect
(response.encodeRedirectUrl(welcomeURL));
```

WebLogic Server uses URL rewriting when a session is new, even if the browser does accept cookies, because the server cannot tell whether a browser accepts cookies in the first visit of a session.

- Your servlet can determine whether a given session ID was received from a cookie by checking the Boolean returned from the `HttpServletRequest.isRequestedSessionIdFromCookie()` method. Your application may respond appropriately, or simply rely on URL rewriting by WebLogic Server.

URL Rewriting and Wireless Access Protocol (WAP)

If you are writing a WAP application, you must use URL rewriting because the WAP protocol does not support cookies. In addition, some WAP devices have a 128-character limit on the length of a URL (including parameters), which limits the amount of data that can be transmitted using URL rewriting. To allow more space for parameters, you can limit the size of the session ID that is randomly generated by WebLogic Server by specifying the number of bytes with the “[IDLength](#)” on page B-9 attribute.

You can save additional space by setting the WAP Enabled attribute, which prevents WebLogic Server from sending primary/secondary information with the URL. You set the WAP Enabled attribute by selecting the Server > Configuration > HTTP tabs of the Administration Console.

5 Configuring Security in Web Applications

The following sections describe how to configure security in Web Applications:

- “Overview of Configuring Security in Web Applications” on page 5-1
- “Setting Up Authentication for Web Applications” on page 5-2
- “Multiple Web Applications, Cookies, and Authentication” on page 5-3
- “Restricting Access to Resources in a Web Application” on page 5-5
- “Using Users and Roles Programmatically in Servlets” on page 5-6

Overview of Configuring Security in Web Applications

You can secure a Web Application by using authentication, by restricting access to certain resources in the Web Application, or by using security calls in your servlet code. Several types of security realms can be used. Security realms are discussed in the document [Security Fundamentals](http://e-docs.bea.com/wls/docs61/security/concepts.html) at

<http://e-docs.bea.com/wls/docs61/security/concepts.html>. Note that a security realm is shared across multiple virtual hosts.

Setting Up Authentication for Web Applications

To configure authentication for a Web Application, use the `<login-config>` element of the `web.xml` deployment descriptor. In this element you define the security realm containing the user credentials, the method of authentication, and the location of resources for authentication. For information on setting up a security realm, see [Security Fundamentals at `http://e-docs.bea.com/wls/docs61/security/concepts.html`](http://e-docs.bea.com/wls/docs61/security/concepts.html).

To set up authentication for Web Applications:

1. Open the `web.xml` deployment descriptor in a text editor or use the Administration Console. For more information, see “[Web Application Developer Tools](#)” on [page 1-5](#).
2. Specify the authentication method using the `<auth-method>` element. The available options are:

BASIC

Basic authentication uses the Web Browser to display a username/password dialog box. This username and password is authenticated against the realm.

FORM

Form-based authentication requires that you return an HTML form containing the username and password. The fields returned from the form elements must be: `j_username` and `j_password`, and the action attribute must be `j_security_check`. Here is an example of the HTML coding for using FORM authentication:

```
<form method="POST" action="j_security_check">
  <input type="text" name="j_username">
  <input type="password" name="j_password">
</form>
```

The resource used to generate the HTML form may be an HTML page, a JSP, or a servlet. You define this resource with the `<form-login-page>` element.

The HTTP session object is created when the login page is served. Therefore, the `session.isNew()` method returns `FALSE` when called from pages served after successful authentication.

CLIENT-CERT

Uses client certificates to authenticate the request. For more information, see [Configuring the SSL Protocol at `http://e-docs.bea.com/wls/docs61/adminguide/cnfgsec.html#cnfgsec015`](http://e-docs.bea.com/wls/docs61/adminguide/cnfgsec.html#cnfgsec015).

3. If you choose FORM authentication, also define the location of the resource used to generate the HTML page with the `<form-login-page>` element and the resource that responds to a failed authentication with the `<form-error-page>` element. For instructions on configuring form authentication see [“form-login-config Element” on page A-22](#).
4. Specify a realm for authentication using the `<realm-name>` element. If you do not specify a particular realm, the realm defined with the Auth Realm Name field on the Web Application→ Configuration→Other tab of the Administration Console is used. For more information, see [“form-login-config Element” on page A-22](#).
5. If you want to define a separate login for a Web Application, see [“Multiple Web Applications, Cookies, and Authentication” on page 5-3](#). Otherwise, all Web Applications that use the same cookie use a single sign-on for authentication.

Multiple Web Applications, Cookies, and Authentication

By default, WebLogic Server assigns the same cookie name (`JSESSIONID`) to all Web Applications. When you use any type of authentication, all Web Applications that use the same cookie name use a single sign-on for authentication. Once a user is authenticated, that authentication is valid for requests to any Web Application that uses the same cookie name. The user is not prompted again for authentication. For more information on single sign-on, see [“Single Sign-on Requirements” on page 5-4](#).

If you want to require separate authentication for a Web Application, you can specify a unique cookie name or cookie path for the Web Application. Specify the cookie name using the `CookieName` parameter and the cookie path with the `CookiePath` parameter, defined in the WebLogic-specific deployment descriptor `weblogic.xml`, in the `<session-descriptor>` element. For more information, see [“session-descriptor Element” on page B-4](#).

If you want to retain the cookie name and still require independent authentication for each Web Application, you can set the cookie path parameter (`CookiePath`) differently for each Web Application.

As of Service Pack 5, BEA Systems added a new capability to WebLogic Server that allows a user to securely access HTTPS resources in a session that was initiated using HTTP, without loss of session data. To enable this new feature, add `AuthCookieEnabled="true"` to the `WebServer` element in `config.xml`:

```
<WebServer Name="myserver" AuthCookieEnabled="true"/>
```

Setting `AuthCookieEnabled` to `true` causes the WebLogic Server instance to send a new secure cookie to the browser when authenticating via an HTTPS connection. Once the secure cookie is set, the session is allowed to access other security-constrained HTTPS resources only if the cookie is sent from the browser.

Note: If authenticating via plain HTTP, the secure cookie is not set or required for any HTTPS resources. When accessing a non-protected HTTPS resource, the cookie is not verified (since it will not have been sent from the browser). This allows the browser to access non-protected HTTPs resources without the user logging in.

Single Sign-on Requirements

The following are requirements that must be met in order to use single sign-on authentication with WebLogic Server.

- The cookie must be shared (the Name, Path, and Domain must match).
- You must have uniform deployments in all cluster members.
- Single-sign on is only supported across a cluster when it is front-ended by a proxy and when session stickiness is working.

Restricting Access to Resources in a Web Application

To restrict access to specified resources (servlets, JSPs, or HTML pages) in your Web Application you apply security constraints to those resources.

To configure a security constraint:

1. Open the `web.xml` and `weblogic.xml` deployment descriptors in a text editor or use the Administration Console. For more information, see [“Web Application Developer Tools” on page 1-5](#).
2. Define a role that is mapped to one or more principals in a security realm. You define roles with the [“security-role Element” on page A-22](#) in the Web Application deployment descriptor. You then map these roles to principals in your realm with the [“security-role-assignment Element” on page B-3](#) in the WebLogic-specific deployment descriptor, `weblogic.xml`.
3. Define which resources in the Web Application the security constraint applies to by using the `<url-pattern>` element that is nested inside the `<web-resource-collection>` element. The `<url-pattern>` can refer to a directory, filename, or a `<servlet-mapping>`.

Alternatively, apply the security constraint to the entire Web Application, use the following entry:

```
<url-pattern>/*</url-pattern>
```

4. Define the HTTP method(s) (GET or POST) that the security constraint applies to using the `<http-method>` element that is nested inside the `<web-resource-collection>` element. Use separate `<http-method>` elements for each HTTP method.
5. Define whether to use SSL for communication between client and server using the `<transport-guarantee>` element nested inside of the `<user-data-constraint>` method.

Listing 5-1 Sample Security Constraint:

```
web.xml entries:
<security-constraint>
  <web-resource-collection>
    <web-resource-name>SecureOrdersEast</web-resource-name>
    <description>
      Security constraint for
      resources in the orders/east directory
    </description>
    <url-pattern>/orders/east/*</url-pattern>
    <http-method>POST</http-method>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
    <description>
      constraint for east coast sales
    </description>
    <role-name>east</role-name>
    <role-name>manager</role-name>
  </auth-constraint>
  <user-data-constraint>
    <description>SSL not required</description>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>

...
```

Using Users and Roles Programmatically in Servlets

You can write your servlets to access users and roles programmatically in your servlet code using the method

```
javax.servlet.http.HttpServletRequest.isUserInRole(String role).
```

The string `role` is mapped to the name supplied in the `<role-name>` element nested inside the `<security-role-ref>` element of a `<servlet>` declaration in the Web

Application deployment descriptor. The `<role-link>` element maps to a `<role-name>` defined in the `<security-role>` element of the Web Application deployment descriptor.

The following listing provides an example.

Listing 5-2 Example of Security Role Mapping

Servlet code:

```
isUserInRole( "manager" );
```

web.xml entries:

```
<servlet>
  . . .
  <role-name>manager</role-name>
  <role-link>mgr</role-link>
  . . .
</servlet>

<security-role>
  <role-name>mgr</role-name>
</security-role>
```

weblogic.xml entries:

```
<security-role-assignment>
  <role-name>mgr</role-name>
  <principal-name>al</principal-name>
  <principal-name>george</principal-name>
  <principal-name>ralph</principal-name>
</security-role-ref>
```

6 Application Events and Listeners

This section discusses how to configure and use Web Application events and listeners:

- [“Overview of Application Events and Listeners” on page 6-1](#)
- [“Servlet Context Events” on page 6-3](#)
- [“HTTP Session Events” on page 6-4](#)
- [“Configuring an Event Listener” on page 6-5](#)
- [“Writing a Listener Class” on page 6-5](#)
- [“Templates for Listener Classes” on page 6-6](#)
- [“Additional Resources” on page 6-8](#)

Overview of Application Events and Listeners

Application events provide notifications of a change in state of the *servlet context* (each Web Application uses its own servlet context) or of an *HTTP session object*. You write event listener classes that respond to these changes in state and you configure and deploy Application event and listener classes in a Web Application.

For servlet context events, the event listener classes can receive notification when the Web Application is deployed or is being undeployed (or when WebLogic Server shuts down), and when attributes are added, removed, or replaced.

For HTTP Session events, the event listener classes can receive notification when an HTTP session is activated or is about to be passivated, and when an HTTP session attribute is added, removed, or replaced.

Use Web Application events to:

- Manage database connections when a Web Application is deployed or shuts down
- Create counters
- Monitor the state of HTTP sessions and their attributes

Note: Application events are a new feature in the Java servlet specification, version 2.3 from Sun Microsystems. Version 2.3 is a proposed final draft of the servlet specification. If you are planning to use application events in your application, note that the specification has not been finalized and could change in the future.

The Servlet 2.3 specification is part of the J2EE 1.3 specification. To use J2EE 1.3 features, please see [“WebLogic Server 6.1 with J2EE 1.2 and J2EE 1.3 Functionality” on page 6-2.](#)

WebLogic Server 6.1 with J2EE 1.2 and J2EE 1.3 Functionality

BEA WebLogic Server 6.1 is the first e-commerce transaction platform to implement advanced J2EE 1.3 features. To comply with the rules governing J2EE, BEA Systems provides two separate downloads: one with J2EE 1.3 features enabled, and one that is limited to J2EE 1.2 features only. Both downloads offer the same container and differ only in the APIs that are available.

WebLogic Server 6.1 with J2EE 1.2 Plus Additional J2EE 1.3 Features

With this download, WebLogic Server defaults to running with J2EE 1.3 features enabled. These features include EJB 2.0, JSP 1.2, Servlet 2.3, and J2EE Connector Architecture 1.0. When you run WebLogic Server 6.1 with J2EE 1.3 features enabled, J2EE 1.2 applications are still fully supported. The J2EE 1.3 feature implementations use non-final versions of the appropriate API specifications. Therefore, application code developed for BEA WebLogic Server 6.1 that uses the new features of J2EE 1.3 may be incompatible with the J2EE 1.3 platform supported in future releases of BEA WebLogic Server.

WebLogic Server 6.1 with J2EE 1.2 Certification

With this download, WebLogic Server defaults to running with J2EE 1.3 features disabled and is fully compliant with the J2EE 1.2 specification and regulations.

Servlet Context Events

The following table lists the types of Servlet context events, the interface your event listener class must implement to respond to the event, and the methods invoked when the event occurs.

Type of Event	Interface	Method
Servlet context is created.	<code>javax.servlet.ServletContextListener</code>	<code>contextInitialized()</code>
Servlet context is about to be shut down.	<code>javax.servlet.ServletContextListener</code>	<code>contextDestroyed()</code>
An attribute is added.	<code>javax.servlet. ServletContextAttributesListener</code>	<code>attributeAdded()</code>
An attribute is removed.	<code>javax.servlet. ServletContextAttributesListener</code>	<code>attributeRemoved()</code>

Type of Event	Interface	Method
An attribute is replaced.	<code>javax.servlet. ServletContextAttributesListener</code>	<code>attributeReplaced()</code>

HTTP Session Events

The following table lists the types of HTTP session events, the interface your event listener class must implement to respond to the event, and the methods invoked when the event occurs.

Type of Event	Interface	Method
An HTTP session is activated.	<code>javax.servlet.http. HttpSessionListener</code>	<code>sessionCreated()</code>
An HTTP session is about to be passivated.	<code>javax.servlet.http. HttpSessionListener</code>	<code>sessionDestroyed()</code>
An attribute is added.	<code>javax.servlet.http. HttpSessionAttributeListener</code>	<code>attributeAdded()</code>
An attribute is removed.	<code>javax.servlet.http. HttpSessionAttributeListener</code>	<code>attributeRemoved()</code>
An attribute is replaced.	<code>javax.servlet.http. HttpSessionAttributeListener</code>	<code>attributeReplaced()</code>

Note: The Servlet 2.3 specification also contains the `javax.servlet.http.HttpSessionBindingListener` and the `javax.servlet.http.HttpSessionActivationListener` interfaces. These interfaces are implemented by objects that are stored as session attributes and do not require registration of an event listener in `web.xml`. For more information, see the Javadocs for these interfaces.

Configuring an Event Listener

To configure an event listener:

1. Open the `web.xml` deployment descriptor of the Web Application for which you are creating an event listener in a text editor, or use the Web Application Deployment Descriptor Editor that is integrated into the Administration Console. (For more information, see [Web Application Deployment Descriptor Editor Help at `http://e-docs.bea.com/wls/docs61/ConsoleHelp/webservices_ddehelp.html`](http://e-docs.bea.com/wls/docs61/ConsoleHelp/webservices_ddehelp.html).) The `web.xml` file is located in the `WEB-INF` directory of your Web Application.
2. Add an event declaration using the `<listener>` element. The event declaration defines the listener class that is invoked when the event occurs. The `<listener>` element must directly follow the `<filter>` and `<filter-mapping>` elements and directly precede the `<servlet>` element. You can specify more than one listener class for each type of event. WebLogic Server invokes the event listeners in the order that they appear in the deployment descriptor (except for shutdown events, which are invoked in the reverse order). For example:

```
<listener>
  <listener-class>myApp.myContextListenerClass</listener-class>
</listener>

<listener>
  <listener-class>myApp.mySessionAttributeListenerClass</listener-class>
</listener>
```

3. Write and deploy the Listener class. See the next section, [Writing a Listener Class](#), for details.

Writing a Listener Class

To write a listener class:

1. Create a new class that implements the appropriate interface for the type of event your class responds to. For a list of these interfaces, see “[Servlet Context Events](#)” on page 6-3 or “[HTTP Session Events](#)” on page 6-4. See “[Templates for Listener Classes](#)” on page 6-6 for sample templates you can use to get started.
2. Create a public constructor that takes no arguments.
3. Implement the required methods of the interface. See the [J2EE API Reference \(Javadocs\)](#) at <http://java.sun.com/j2ee/tutorial/api/index.html> for more information.
4. Copy the compiled event listener classes into the `WEB-INF/classes` directory of the Web Application, or package them into a `jar` file and copy the `jar` file into the `WEB-INF/lib` directory of the Web Application.

The following useful classes are passed into the listener methods in a listener class:

```
javax.servlet.http.HttpSessionEvent
    provides access to the HTTP session object

javax.servlet.ServletContextEvent
    provides access to the servlet context object.

javax.servlet.ServletContextAttributeEvent
    provides access to servlet context and its attributes

javax.servlet.http.HttpSessionBindingEvent
    provides access to an HTTP session and its attributes
```

Templates for Listener Classes

The following examples provide some basic templates for listener classes.

Servlet Context Listener Example

```
package myApp;
import javax.servlet.*;
```

```
public final class myContextListenerClass implements
ServletContextListener {
    public void contextInitialized(ServletContextEvent event) {

        /* This method is called when the servlet context is
           initialized(when the Web Application is deployed).
           You can initialize servlet context related data here.
        */

    }

    public void contextDestroyed(ServletContextEvent event) {

        /* This method is invoked when the Servlet Context
           (the Web Application) is undeployed or
           WebLogic Server shuts down.
        */

    }
}
```

HTTP Session Attribute Listener Example

```
package myApp;
import javax.servlet.*;

public final class mySessionAttributeListenerClass implements
HttpSessionAttributeListener {

    public void attributeAdded(HttpSessionBindingEvent sbe) {
        /* This method is called when an attribute
           is added to a session.
        */
    }

    public void attributeRemoved(HttpSessionBindingEvent sbe) {
        /* This method is called when an attribute
           is removed from a session.
        */
    }

    public void attributeReplaced(HttpSessionBindingEvent sbe) {
        /* This method is invoked when an attribute
           is replaced in a session.
        */
    }
}
```

```
}  
}
```

Additional Resources

- [Assembling and Configuring Web Applications](http://e-docs.bea.com/wls/docs61/webapp/index.html) at <http://e-docs.bea.com/wls/docs61/webapp/index.html>
- [Writing Web Application Deployment Descriptors](http://e-docs.bea.com/wls/docs61/webapp/webappdeployment.html) at <http://e-docs.bea.com/wls/docs61/webapp/webappdeployment.html>
- [Servlet 2.3 Specification](http://java.sun.com/aboutJava/communityprocess/first/jsr053/index.html) from Sun Microsystems at <http://java.sun.com/aboutJava/communityprocess/first/jsr053/index.html>
- [J2EE API Reference \(Javadocs\)](http://java.sun.com/j2ee/tutorial/api/index.html) at <http://java.sun.com/j2ee/tutorial/api/index.html>
- [The J2EE Tutorial](http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html) from Sun Microsystems: at http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html

7 Filters

The following sections provide information about using filters in a Web Application:

- [“Overview of Filters” on page 7-1](#)
- [“Configuring Filters” on page 7-4](#)
- [“Writing a Filter” on page 7-6](#)
- [“Example of a Filter Class” on page 7-8](#)
- [“Filtering the Servlet Response Object” on page 7-9](#)
- [“Additional Resources” on page 7-9](#)

Overview of Filters

A filter is a Java class that is invoked in response to a request for a resource in a Web Application. Resources include Java Servlets, JavaServer pages (JSP), and static resources such as HTML pages or images. A filter intercepts the request and can examine and modify the response and request objects or execute other tasks.

Filters are an advanced J2EE feature primarily intended for situations where the developer cannot change the coding of an existing resource and needs to modify the behavior of that resource. Generally, it is more efficient to modify the code to change the behavior of the resource itself rather than using filters to modify the resource. In some situations, using filters can add unnecessary complexity to an application and degrade performance.

Note: Filters are a new feature in the Java servlet specification, version 2.3 from Sun Microsystems. Version 2.3 is a proposed final draft of the servlet specification. If you are planning to use filters in your application, note that the specification has not been finalized and could change in the future.

The Servlet 2.3 specification is part of the J2EE 1.3 specification. To use J2EE 1.3 features, please see [“WebLogic Server 6.1 with J2EE 1.2 and J2EE 1.3 Functionality”](#) on page 7-2.

WebLogic Server 6.1 with J2EE 1.2 and J2EE 1.3 Functionality

BEA WebLogic Server 6.1 is the first e-commerce transaction platform to implement advanced J2EE 1.3 features. To comply with the rules governing J2EE, BEA Systems provides two separate downloads: one with J2EE 1.3 features enabled, and one that is limited to J2EE 1.2 features only. Both downloads offer the same container and differ only in the APIs that are available.

WebLogic Server 6.1 with J2EE 1.2 Plus Additional J2EE 1.3 Features

With this download, WebLogic Server defaults to running with J2EE 1.3 features enabled. These features include EJB 2.0, JSP 1.2, Servlet 2.3, and J2EE Connector Architecture 1.0. When you run WebLogic Server 6.1 with J2EE 1.3 features enabled, J2EE 1.2 applications are still fully supported. The J2EE 1.3 feature implementations use non-final versions of the appropriate API specifications. Therefore, application code developed for BEA WebLogic Server 6.1 that uses the new features of J2EE 1.3 may be incompatible with the J2EE 1.3 platform supported in future releases of BEA WebLogic Server.

WebLogic Server 6.1 with J2EE 1.2 Certification

With this download, WebLogic Server defaults to running with J2EE 1.3 features disabled and is fully compliant with the J2EE 1.2 specification and regulations.

How Filters Work

You define filters in the context of a Web Application. A filter intercepts a request for a specific named resource or a group of resources (based on a URL pattern) and executes the code in the filter. For each resource or group of resources, you can specify a single filter or multiple filters that are invoked in a specific order, called a *chain*.

When a filter intercepts a request, it has access to the `javax.servlet.HttpServletRequest` and `javax.servlet.HttpServletResponse` objects that provide access to the HTTP request and response, and a `javax.servlet.FilterChain` object. The `FilterChain` object contains a list of filters that can be invoked sequentially. When a filter has completed its work, the filter can either call the next filter in the chain, block the request, throw an exception, or invoke the originally requested resource.

After the original resource is invoked, control is passed back to the filter at the bottom of the list in the chain. This filter can then examine and modify the response headers and data, block the request, throw an exception, or invoke the next filter up from the bottom of the chain. This process continues in reverse order up through the chain of filters.

Uses for Filters

Filters can be useful for the following functions:

- Implementing a logging function
- Implementing user-written security functionality
- Debugging
- Encryption
- Data compression
- Modifying the response sent to the client. (Although post processing the response can degrade the performance of your application.)

Configuring Filters

You configure filters as part of a Web Application, using the application's `web.xml` deployment descriptor. In the deployment descriptor, you declare the filter and then map the filter to a URL pattern or to a specific servlet in the Web Application. You can declare any number of filters.

Configuring a Filter

To configure a filter:

1. Open the `web.xml` deployment descriptor in a text editor or use the Administration Console. For more information, see [“Web Application Developer Tools” on page 1-5](#). The `web.xml` file is located in the `WEB-INF` directory of your Web Application.
2. Add a filter declaration. The `<filter>` element declares a filter, defines a name for the filter, and specifies the Java class that executes the filter. The `<filter>` element must directly follow the `<context-param>` element and directly precede the `<listener>` and `<servlet>` elements. For example:

```
<filter>
  <icon>
    <small-icon>MySmallIcon.gif</small-icon>
    <large-icon>MyLargeIcon.gif</large-icon>
  </icon>
  <filter-name>myFilter1</filter-name>
  <display-name>filter 1</display-name>
  <description>This is my filter</description>
  <filter-class>examples.myFilterClass</filter-class>
</filter>
```

The `icon`, `description`, and `display-name` elements are optional.

3. Specify one or more initialization parameters inside a `<filter>` element. For example:

```
<filter>
  <icon>
    <small-icon>MySmallIcon.gif</small-icon>
    <large-icon>MyLargeIcon.gif</large-icon>
```



```
</icon>
<filter-name>myFilter1</filter-name>
<display-name>filter 1</display-name>
<description>This is my filter</description>
<filter-class>examples.myFilterClass</filter-class>
<init-param>
  <param-name>myInitParam</param-name>
  <param-value>myInitParamValue</param-value>
</init-param>
</filter>
```

Your Filter class can read the initialization parameters using the `FilterConfig.getInitParameter()` or `FilterConfig.getInitParameters()` methods.

4. Add filter mappings. The `<filter-mapping>` element specifies which filter to execute based on a URL pattern or servlet name. The `<filter-mapping>` element must immediately follow the `<filter>` element(s).
 - To create a filter mapping using a URL pattern, specify the name of the filter and a URL pattern. URL pattern matching is performed according to the rules specified in the [Servlet 2.3 Specification from Sun Microsystems at `http://java.sun.com/aboutJava/communityprocess/first/jsr053/index.html`](http://java.sun.com/aboutJava/communityprocess/first/jsr053/index.html), in section 11.1. For example, the following filter-mapping maps `myFilter` to requests that contain `/myPattern/`.

```
<filter-mapping>
  <filter-name>myFilter</filter-name>
  <url-pattern>/myPattern/*</url-pattern>
</filter-mapping>
```
 - To create a filter mapping for a specific servlet, map the filter to the name of a servlet that is registered in the Web Application. For example, the following code maps the `myFilter` filter to a servlet called `myServlet`:

```
<filter-mapping>
  <filter-name>myFilter</filter-name>
  <servlet-name>myServlet</servlet-name>
</filter-mapping>
```
5. To create a chain of filters, specify multiple filter mappings. For more information, see [“Configuring a Chain of Filters” on page 7-6](#).

Configuring a Chain of Filters

WebLogic Server creates a *chain* of filters by creating a list of all the filter mappings that match an incoming HTTP request. The ordering of the list is determined by the following sequence:

1. Filters where the `filter-mapping` contains a `url-pattern` that matches the request are added to the chain in the order they appear in the `web.xml` deployment descriptor.
2. Filters where the `filter-mapping` contains a `servlet-name` that matches the request are added to the chain *after* the filters that match a URL pattern.
3. The last item in the chain is always the originally requested resource.

In your filter class, use the `FilterChain.doFilter()` method to invoke the next item in the chain.

Writing a Filter

To write a filter class, implement the `javax.servlet.Filter` interface (see <http://java.sun.com/j2ee/tutorial/api/javax/servlet/Filter.html>). You must implement the following methods of this interface:

`doFilter()`

Use this method to examine and modify the request and response objects, perform other tasks such as logging, invoke the next filter in the chain, or block further processing.

`getFilterConfig()`

Use this method to gain access to the `javax.servlet.FilterConfig` (see <http://java.sun.com/j2ee/tutorial/api/javax/servlet/FilterConfig.html>) object.

`setFilterConfig()`

Use this method to set the `javax.servlet.FilterConfig` (see <http://java.sun.com/j2ee/tutorial/api/javax/servlet/FilterConfig.html>) object.

Several other methods are available on the `FilterConfig` object for accessing the name of the filter, the `ServletContext` and the filter's initialization parameters. For more information see the [J2EE javadocs](#) from Sun Microsystems for

`javax.servlet.FilterConfig`. Javadocs are available at <http://java.sun.com/j2ee/tutorial/api/index.html>.

To access the next item in the chain (either another filter or the original resource, if that is the next item in the chain), call the `FilterChain.doFilter()` method.

Example of a Filter Class

The following code example demonstrates the basic structure of a `Filter` class.

Listing 7-1 Filter Class Example

```
import javax.servlet.*;
public class Filter1Impl implements Filter
{
    private FilterConfig filterConfig;

    public void doFilter(ServletRequest req,
        ServletResponse res, FilterChain fc)
        throws java.io.IOException, javax.servlet.ServletException
    {
        // Execute a task such as logging.
        //...

        fc.doFilter(req,res); // invoke next item in the chain --
                               // either another filter or the
                               // originally requested resource.

    }

    public FilterConfig getFilterConfig()
    {
        // Execute tasks
        return filterConfig;
    }

    public void setFilterConfig(FilterConfig cfg)
    {
        // Execute tasks
        filterConfig = cfg;
    }
}
```

Filtering the Servlet Response Object

You can use filters to post-process the output of a servlet by appending data to the output generated by the servlet. However, in order to capture the output of the servlet, you must create a wrapper for the response. (You cannot use the original response object, because the output buffer of the servlet is automatically flushed and sent to the client when the servlet completes executing and *before* control is returned to the last filter in the chain.) When you create such a wrapper, WebLogic Server must manipulate an additional copy of the output in memory, which can degrade performance.

For more information on wrapping the response or request objects, see the [J2EE javadocs](#) from Sun Microsystems for `javax.servlet.http.HttpServletResponseWrapper` and `javax.servlet.http.HttpServletRequestWrapper`. **Javadocs are available at** <http://java.sun.com/j2ee/tutorial/api/index.html>.

Additional Resources

- [Assembling and Configuring Web Applications](#) at <http://e-docs.bea.com/wls/docs61/webapp/index.html>
- [Writing Web Application Deployment Descriptors](#) at <http://e-docs.bea.com/wls/docs61/webapp/webappdeployment.html>
- [Servlet 2.3 Specification](#) from Sun Microsystems at <http://java.sun.com/aboutJava/communityprocess/first/jsr053/index.html>
- [J2EE API Reference \(Javadocs\)](#) at <http://java.sun.com/j2ee/tutorial/api/index.html>
- [The J2EE Tutorial](#) from Sun Microsystems: at http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html

8 Writing Web Application Deployment Descriptors

The following sections describe how to write Web Application deployment descriptors:

- [Overview of Web Application Deployment Descriptors on page 8-2](#)
- [Tools for Editing Deployment Descriptors on page 8-2](#)
- [Writing the web.xml Deployment Descriptor on page 8-3](#)
- [Sample web.xml on page 8-21](#)
- [Writing the WebLogic-Specific Deployment Descriptor \(weblogic.xml\) on page 8-24](#)

Overview of Web Application Deployment Descriptors

WebLogic Server uses the standard J2EE `web.xml` deployment descriptor for defining a Web Application. Some applications also require the WebLogic-specific deployment descriptor, `weblogic.xml`. You use these deployment descriptors to define components and operating parameters for a Web Application. Deployment descriptors are standard text files, formatted using XML notation and you package them in the Web Application. For more information on Web Applications, see [Web Applications Basics on page 1-1](#).

The deployment descriptor `web.xml` is defined by the servlet 2.3 specification from Sun Microsystems. This deployment descriptor can be used to deploy a Web Application on any J2EE-compliant application server.

The deployment descriptor `weblogic.xml` defines deployment properties that are specific to a Web Application running on WebLogic Server. `weblogic.xml` is not required for all Web Applications.

Tools for Editing Deployment Descriptors

To edit a deployment descriptor, you can use one of several tools:

- Use the deployment descriptor editor that is integrated into the WebLogic Server Administration Console. For more information, see [Web Application Deployment Descriptor Editor Help at `http://e-docs.bea.com/wls/docs61/ConsoleHelp/webservices_ddehelp.html`](#).
- Use any plain text editor, for example Windows Notepad, `emacs`, `vi`, or your favorite IDE.
- BEA WebLogic now provides a simple, user-friendly tool from Ensemble for creating and editing XML files. It can validate XML code according to a specified DTD or XML Schema. The XML editor can be used on Windows or

Solaris machines and is downloadable from the [BEA dev2dev](http://dev2dev.bea.com/resourcelibrary/utilitiestools/xml.jsp) at <http://dev2dev.bea.com/resourcelibrary/utilitiestools/xml.jsp>.

- You can use the ANT utilities to create skeleton deployment descriptors. The ANT task looks at a directory containing a Web Application and creates deployment descriptors based on the files it finds in the Web Application. Since the ANT tasks do not know all of the desired configurations, mappings, and other information, the skeleton deployment descriptors it creates are incomplete. You can then use a text editor, an XML editor, or the Administration Console to complete configuration of your Web Application using the deployment descriptors.

For more information, see [Packaging Web Applications](http://e-docs.bea.com/wls/docs61/programming/packaging.html#pack005) at <http://e-docs.bea.com/wls/docs61/programming/packaging.html#pack005>.

Writing the web.xml Deployment Descriptor

This section describes the steps to create the `web.xml` deployment descriptor. Depending on the components in your Web application, you may not need to include all of the elements listed here to configure and deploy your Web application.

The elements in the `web.xml` file must be entered in the order they are presented in this document.

Main Steps to Create the web.xml File

- Step 1: Create a deployment descriptor file on page 8-4
- Step 2: Create the DOCTYPE Statement on page 8-5
- Step 3: Create the main body of the web.xml file on page 8-5
- Step 4: Define deployment-time attributes on page 8-6
- Step 5: Define context parameters on page 8-6
- Step 6: Configure Filters (Servlet 2.3 specification only) on page 8-7

- Step 7: Define Filter Mappings (Servlet 2.3 specification only) on page 8-8
- Step 8: Configure Application Listeners (Servlet 2.3 specification only) on page 8-9
- Step 9: Deploy Servlets on page 8-9
- Step 10: Map a servlet to a URL on page 8-12
- Step 11: Define the session timeout value on page 8-13
- Step 12: Define MIME mapping on page 8-13
- Step 13: Define welcome pages on page 8-14
- Step 14: Define error pages on page 8-14
- Step 15: Define a JSP tag library descriptor on page 8-15
- Step 16: Reference external resources on page 8-16
- Step 17: Set up security constraints on page 8-16
- Step 18: Set up login authentication on page 8-18
- Step 19: Define security roles on page 8-19
- Step 20: Set environment entries on page 8-20
- Step 21: Reference Enterprise JavaBean (EJB) resources on page 8-21

If you have installed the WebLogic Server samples and examples, you can look at the `web.xml` and `weblogic.xml` files in the Pet Store sample to see a working example of Web application deployment descriptors. These files are located in the `/samples/PetStore/source/dd/war/WEB-INF` directory of your WebLogic Server distribution.

Detailed Steps to Create the `web.xml` File

Step 1: Create a deployment descriptor file

Name the file `web.xml` and place it under the `WEB-INF` directory of the Web application. Use any text editor.

Step 2: Create the DOCTYPE Statement

The DOCTYPE statement refers to the location and version of the Document Type Definition (DTD) file for the deployment descriptor. Although this header references an external URL at `java.sun.com`, WebLogic Server contains its own copy of the DTD file, so your host server need not have access to the Internet. However, you must still include this `<!DOCTYPE . . . >` element in your `web.xml` file, and have it reference the external URL because the version of the DTD contained in this element is used to identify the version of this deployment descriptor.

Use one of the following DOCTYPE statements:

- If you are using any of the features of the Servlet 2.3 Specification, such as Filters or Application Events, use the following DOCTYPE statement:

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
```

Note: The implementation of version 2.3 of the Servlet Specification is based on the *Proposed Final Draft 1* of the servlet specification and is subject to change. If you are planning to use features introduced in version 2.3, note that the specification has not been finalized and could change in the future. Features added with *Proposed Final Draft 2* are not supported.

- If you do not need to use features of the Servlet 2.3 specification, use the following DOCTYPE statement:

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//
DTD WebApplication 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
```

Step 3: Create the main body of the web.xml file

Wrap all of your entries within a pair of opening and closing `<web-app>` tags.

```
<web-app>
```

All elements describing this Web application
go within the `<web-app>` element.

```
</web-app>
```

In XML, properties are defined by surrounding a property name or value with opening and closing tags as shown above. The opening tag, the body (the property name or value), and the closing tag are collectively called an *element*. Some elements do not use the surrounding tags, but instead use a single tag that contains attributes called an *empty-tag*. Elements contained within other elements are indented in this text for clarity. Indenting is not necessary in an XML file.

The body of the `<web-app>` element itself contains additional elements that determine how the Web application will run on WebLogic Server. The order of the tag elements within the file *must follow the order reflected in this document*. This ordering is defined in the Document Type Definition (DTD) file.

Step 4: Define deployment-time attributes

These tags provide information for the deployment tools or the application server resource management tools. These values are not used by WebLogic Server in this release.

<code><icon></code>	(Optional)
<code><small-icon></code> <code>iconfile.gif(jpg)</code> <code></small-icon></code>	(Optional)
<code><large-icon></code> <code>iconfile.gif(jpg)</code> <code></large-icon></code>	(Optional)
<code></icon></code>	
<code><display-name></code> <code>application-name</code> <code></display-name></code>	(Optional)
<code><description></code> <code>descriptive-text</code> <code></description></code>	(Optional)
<code><distributable/></code>	(Optional)

Step 5: Define context parameters

The `context-param` element declares servlet context initialization parameters for a Web Application. These are parameters that you define that are available throughout your Web application. You set each `context-param` within a single

context-param element, using <param-name> and <param-value> elements. You can access these parameters in your code using the

```
javax.servlet.ServletContext.getInitParameter() and  
javax.servlet.ServletContext.getInitParameterNames() methods.
```

The weblogic.http.clientCertProxy context parameter specifies to trust client certificates in the WL-Proxy-Client-Cert header. In previous releases of WebLogic Server, the client certificates in the WL-Proxy-Client-Cert header were trusted by default.

<context-param>	For more information, see context-param Element on page A-6 .
<param-name> user-defined param name </param-name>	(Required)
<param-value> user-defined value </param-value>	(Required)
<description> text description </description>	(Optional)
</context-param>	

Step 6: Configure Filters (Servlet 2.3 specification only)

Each filter has a name and a filter class. (For more information on filters, see [Configuring Filters on page 7-4](#). A filter can also use initialization parameters. The following elements define a filter:

<filter>	For more information, see filter Element on page A-6 .
-----------------------	--

<code><icon></code>	<i>(Optional)</i>
<code> <small-icon></code> iconfile <code> </small-icon></code>	
<code> <large-icon></code> iconfile <code> </large-icon></code>	
<code></icon></code>	
<code><filter-name></code> Filter name <code></filter-name></code>	<i>(Required)</i>
<code><display-name></code> Filter Display Name <code></display-name></code>	<i>(Optional)</i>
<code><description></code> ...text... <code></description></code>	<i>(Optional)</i>
<code><filter-class></code> package.name.MyFilterClass <code></filter-class></code>	<i>(Required)</i>
<code><init-param></code>	<i>(Optional).</i>
<code> <param-name></code> name <code> </param-name></code>	<i>(Required)</i>
<code> <param-value></code> value <code> </param-value></code>	<i>(Required)</i>
<code></init-param></code>	<i>(Optional)</i>
<code></filter></code>	

Step 7: Define Filter Mappings (Servlet 2.3 specification only)

After you have declared a filter, map each filter to a URL pattern:

<code><filter-mapping></code>	For more information, see filter-mapping Element on page A-7.
-------------------------------------	---

<code><filter-name></code>	<i>(Required)</i>
name	
<code></filter-name></code>	
<code><url-pattern></code>	<i>(Required)</i>
pattern	
<code></url-pattern></code>	
<code></filter-mapping></code>	

Step 8: Configure Application Listeners (Servlet 2.3 specification only)

Configure Web Application event listeners using a separate `<listener>` element for each listener class:

For more information, see [Application Events and Listeners on page 6-1](#).

<code><listener></code>	For more information, see listener Element on page A-8 .
<code><listener-class></code>	<i>(Required)</i>
my.foo.listener	
<code></listener-class></code>	
<code></listener></code>	

Step 9: Deploy Servlets

To deploy a servlet, give the servlet a name, specify the class file or JSP used to implement its behavior, and set other servlet-specific properties. List each servlet in your Web application within separate `<servlet>...</servlet>` elements. After you create entries for all your servlets, you must include elements that map the servlet to a URL pattern. These mapping elements are described in “[Step 10: Map a servlet to a URL](#)” on page 8-12.

For more information, see [Configuring Servlets on page 3-2](#)

Use the following elements to declare a servlet:

<code><servlet></code>	For more information, see “ servlet Element ” on page A-8.
------------------------------	--

<code><servlet-name></code> name <code></servlet-name></code>	(Required)
<code><servlet-class></code> package.name.MyClass <code></servlet-class></code>	(Required)
-or-	
<code><jsp-file></code> /foo/bar/myFile.jsp <code></jsp-file></code>	
<code><init-param></code>	(Optional) For more information, see “init-param Element” on page A-10.
<code><param-name></code> name <code></param-name></code>	(Required)
<code><param-value></code> value <code></param-value></code>	(Required)
<code><description></code> ...text... <code></description></code>	(Optional)
<code></init-param></code>	
<code><load-on-startup></code> loadOrder <code></load-on-startup></code>	(Optional)
<code><security-role-ref></code>	(Optional). For more information, see “security-role-ref Element” on page A-11.
<code><description></code> ...text... <code></description></code>	(Optional)
<code><role-name></code> rolename <code></role-name></code>	(Required)

<code><role-link></code> rolelink <code></role-link></code>	<i>(Required)</i>
<code></security-role-ref></code>	
<code><small-icon></code> iconfile <code></small-icon></code>	<i>(Optional)</i>
<code><large-icon></code> iconfile <code></large-icon></code>	<i>(Optional)</i>
<code><display-name></code> Servlet Name <code></display-name></code>	<i>(Optional)</i>
<code><description></code> ...text... <code></description></code>	<i>(Optional)</i>
<code></servlet></code>	

Here is an example of a servlet element that includes an initialization parameter.

```
<servlet>
  ...
  <init-param>
    <param-name>feedbackEmail</param-name>
    <param-value>feedback123@beasys.com</param-value>
    <description>
      The email for web-site feedback.
    </description>
  </init-param>
  ...
</servlet>
```

Step 10: Map a servlet to a URL

Once you declare your servlet or JSP using a `<servlet>` element, map it to one or more URL patterns to make it a public HTTP resource. The use of URL patterns is defined in the Servlet 2.3 specification from Sun Microsystems. For each mapping, use a `<servlet-mapping>` element.

<code><servlet-mapping></code>	For more information, see servlet-mapping Element on page A-12 .
<code><servlet-name></code> name <code></servlet-name></code>	(Required)
<code><url-pattern></code> pattern <code></url-pattern></code>	(Required)
<code></servlet-mapping></code>	

Here is an example of a `<servlet-mapping>` for the `<servlet>` declaration example used earlier:

```
<servlet-mapping>
  <servlet-name>LoginServlet</servlet-name>
  <url-pattern>/login</url-pattern>
</servlet-mapping>
```

Step 11: Define the session timeout value

<session-config>	(Optional)
<session-timeout> minutes </session-timeout>	For more information, see “session-config Element” on page A-13.
</session-config>	

Step 12: Define MIME mapping

To create a MIME mapping, you map a file extension to a MIME type.

<mime-mapping>	(Optional)
	Define MIME types.
	For more information, see “mime-mapping Element” on page A-13.
<extension> ext </extension>	
<mime-type> mime type </mime-type>	
</mime-mapping>	

Step 13: Define welcome pages

For more information, see [Configuring Welcome Pages on page 3-7](#)

<code><welcome-file-list></code>	(Welcome pages are Optional.) For more information, see “ welcome-file-list Element ” on page A-14.
<code><welcome-file></code> myWelcomeFile.jsp <code></welcome-file></code>	See also Configuring Welcome Pages on page 3-7 and How WebLogic Server Resolves HTTP Requests at http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req .
<code><welcome-file></code> myWelcomeFile.html <code></welcome-file></code>	
<code></welcome-file-list></code>	

Step 14: Define error pages

For more information, see [Customizing HTTP Error Responses on page 3-9](#)

<code><error-page></code>	(Optional) Define a customized page to respond to errors. For more information, see “ error-page Element ” on page A-15 and How WebLogic Server Resolves HTTP Requests at http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req .
---------------------------------	--

```

<error-code>
    HTTP error code
</error-code>

-or-

<exception-type>
    Java exception class
</exception-type>

<location>
    URL
</location>
</error-page>

```

Step 15: Define a JSP tag library descriptor

For more information, see [Configuring JSP Tag Libraries on page 3-6](#).

<pre> <taglib> </pre>	<p><i>(Optional)</i> Identify JSP tag libraries.</p> <p>For more information, see “taglib Element” on page A-16.</p>
<pre> <taglib-uri> string_pattern </taglib-uri> </pre>	<p><i>(Required)</i></p>
<pre> <taglib-location> filename </taglib-location> </taglib> </pre>	<p><i>(Required)</i></p>

The following is an example of a taglib directive used in a JSP:

```
<%@ taglib uri="string_pattern" prefix="taglib" %>
```

For more details, see [Programming WebLogic JSP Tag Extensions](http://e-docs.bea.com/wls/docs61/taglib/index.html) at <http://e-docs.bea.com/wls/docs61/taglib/index.html>.

Step 16: Reference external resources

For more information, see [Configuring External Resources in a Web Application on page 3-13](#).

<resource-ref>	(Optional) For more information, see “ resource-ref Element ” on page A-17.
<res-ref-name> name </res-ref-name>	(Required)
<res-type> Java class </res-type>	(Required)
<res-auth> CONTAINER SERVLET </res-auth>	(Required)
<res-sharing-scope> Sharable Unsharable </res-sharing-scope>	(Optional)
</resource-ref>	(Required)

Step 17: Set up security constraints

A Web application that uses security requires the user to log in in order to access its resources. The user’s credentials are verified against a security realm, and once authorized, the user will have access only to specified resources within the Web application.

Security in a Web application is configured using three elements:

- The `<login-config>` element specifies how the user is prompted to log in and the location of the security realm. If this element is present, the user must be authenticated in order to access any resource that is constrained by a `<security-constraint>` defined in the Web application.
- A `<security-constraint>` is used to define the access privileges to a collection of resources via their URL mapping.

- A `<security-role>` element represents a group or principal in the realm. This security role name is used in the `<security-constraint>` element and can be linked to an alternative role name used in servlet code via the `<security-role-ref>` element.

For more information, see [Restricting Access to Resources in a Web Application on page 5-5](#).

<code><security-constraint></code>	(Optional) For more information, see “security-constraint Element” on page A-18 .
<code><web-resource-collection></code>	(Required) For more information, see “web-resource-collection Element” on page A-18 .
<code><web-resource-name></code> name <code></web-resource-name></code>	(Required)
<code><description></code> ...text... <code></description></code>	(Optional)
<code><url-pattern></code> pattern <code></url-pattern></code>	(Optional)
<code><http-method></code> GET POST <code></http-method></code>	(Optional)
<code></web-resource-collection></code>	
<code><auth-constraint></code>	(Optional) For more information, see “auth-constraint Element” on page A-19 .
<code><role-name></code> group principal <code></role-name></code>	(Optional)

</auth-constraint>	
<user-data-constraint>	(Optional) For more information, see “user-data-constraint Element” on page A-20.
<description> ...text... </description>	(Optional)
<transport-guarantee> NONE INTEGRAL CONFIDENTIAL </transport-guarantee>	(Required)
</user-data-constraint>	
</security-constraint>	

Step 18: Set up login authentication

For more information, see [Setting Up Authentication for Web Applications on page 5-2.](#)

<login-config>	(Optional) For more information, see “login-config Element” on page A-21.
<auth-method> BASIC FORM CLIENT-CERT </auth-method>	(Optional) Specifies the method used to authenticate the user.

<pre> <realm-name> realmname </realm-name> </pre>	<p>(Optional) For more information, see Specifying a Security Realm at http://e-docs.bea.com/wls/docs61/adminguide/cnfgsec.html#cnfgsec004.</p>
<pre> <form-login-config> </pre>	<p>(Optional)</p> <p>For more information, see “form-login-config Element” on page A-22.</p> <p>Use this element if you configure the <code><auth-method></code> to FORM.</p>
<pre> <form-login-page> URI </form-login-page> </pre>	<p>(Required)</p>
<pre> <form-error-page> URI </form-error-page> </form-login-config> </pre>	<p>(Required)</p>
<pre> </login-config> </pre>	

Step 19: Define security roles

For more information, see [Configuring Security in Web Applications on page 5-1](#).

<pre> <security-role> </pre>	<p>(Optional)</p> <p>For more information, see “security-role Element” on page A-22.</p>
------------------------------------	--

<code><description></code> ...text...	(Optional)
<code></description></code>	
<code><role-name></code> rolename	(Required)
<code></role-name></code>	
<code></security-role></code>	

Step 20: Set environment entries

For more information, see [Configuring External Resources in a Web Application on page 3-13](#).

<code><env-entry></code>	(Optional) For more information, see “ env-entry Element ” on page A-23 .
<code><description></code> ...text...	(Optional)
<code></description></code>	
<code><env-entry-name></code> name	(Required)
<code></env-entry-name></code>	
<code><env-entry-value></code> value	(Required)
<code></env-entry-value></code>	
<code><env-entry-type></code> type	(Required)
<code></env-entry-type></code>	
<code></env-entry></code>	

Step 21: Reference Enterprise JavaBean (EJB) resources

For more information, see [Referencing EJBs in a Web Application](#) on page 3-15.

<code><ejb-ref></code>	<i>Optional)</i> For more information, see “ ejb-ref Element ” on page A-24.
<code><description></code> ...text... <code></description></code>	<i>(Optional)</i>
<code><ejb-ref-name></code> name <code></ejb-ref-name></code>	<i>(Required)</i>
<code><ejb-ref-type></code> Java type <code></ejb-ref-type></code>	<i>(Required)</i>
<code><home></code> mycom.ejb.AccountHome <code></home></code>	<i>(Required)</i>
<code><remote></code> mycom.ejb.Account <code></remote></code>	<i>(Required)</i>
<code><ejb-link></code> ejb.name <code></ejb-link></code>	<i>(Optional)</i>
<code><run-as></code> security role <code></run-as></code>	<i>(Optional)</i>
<code></ejb-ref></code>	<i>(Required)</i>

Sample web.xml

Listing 8-1 Sample web.xml with Servlet Mapping, Welcome file, and Error

Page

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//
DTD Web Application 1.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

<!-- The following servlet element defines a servlet called
servletA.
The Java class of this servlet is servlets.servletA -->
  <servlet>
    <servlet-name>servletA</servlet-name>
    <servlet-class>servlets.servletA</servlet-class>
  </servlet>

<!-- The following servlet element defines another servlet called
servletB. The Java class of this servlet is servlets.servletB -->
  <servlet>
    <servlet-name>servletB</servlet-name>
    <servlet-class>servlets.servletB</servlet-class>
  </servlet>

<!-- The following servlet-mapping maps the servlet called servletA
(see the servlet element) to a url-pattern of "blue".
The url-pattern is used when requesting this servlet, for example:
http://host:port/myWebApp/blue. -->
  <servlet-mapping>
    <servlet-name>servletA</servlet-name>
    <url-pattern>blue</url-pattern>
  </servlet-mapping>

<!-- The following servlet-mapping maps the servlet called servletB
(see the servlet element) to a url-pattern of "yellow".
The url-pattern is used when requesting this servlet, for example:
http://host:port/myWebApp/yellow. -->
  <servlet-mapping>
    <servlet-name>servletB</servlet-name>
    <url-pattern>yellow</url-pattern>
  </servlet-mapping>

<!--The following welcome-file-list specifies a welcome-file.
Welcome files are discussed elsewhere in this document-->
  <welcome-file-list>
    <welcome-file>hello.html</welcome-file>
  </welcome-file-list>

<!--The following error-page element specifies a page that is served
in place of the standard HTTP error response pages, in this case
```

```
HTTP error 404.-->
  <error-page>
    <error-code>404</error-code>
    <location>/error.jsp</location>
  </error-page>
</web-app>
```

Writing the WebLogic-Specific Deployment Descriptor (weblogic.xml)

The `weblogic.xml` file contains WebLogic-specific attributes for a Web application. You define the following attributes in this file: HTTP session parameters, HTTP cookie parameters, JSP parameters, resource references, security role assignments, character set mappings, and container attributes.

If you define external resources such as DataSources, EJBs, or a Security realm in the `web.xml` deployment descriptor, you can use any descriptive name to define the resource. To access the resource, you then map this resource name to the actual name of the resource in the JNDI tree using a file called `weblogic.xml`. Place this file in the `WEB-INF` directory of your Web application.

If you have installed the WebLogic Server samples and examples, you can look at the `web.xml` and `weblogic.xml` files in the Pet Store sample to see a working example of Web application deployment descriptors. These files are located in the `/samples/PetStore/source/dd/war/WEB-INF` directory of your WebLogic Server distribution.

The ordering of the tag elements within the `weblogic.xml` file must follow the ordering specified in this document.

Main Steps to Create the weblogic.xml File

- [Step 1: Begin the weblogic.xml file with a DOCTYPE header on page 8-25](#)
- [Step 2: Map security role names to a security realm on page 8-26](#)
- [Step 3 Map resources to JNDI on page 8-26](#)
- [Step 4: Define session parameters on page 8-28](#)
- [Step 5: Define JSP parameters on page 8-28](#)
- [Step 6: Define container parameters on page 8-29](#)
- [Step 7: Define character set parameters on page 8-30](#)

- [Step 8: Complete the descriptor file. on page 8-30](#)

Detailed Steps to Create the weblogic.xml File

Step 1: Begin the weblogic.xml file with a DOCTYPE header

This header refers to the location and version of the DTD file for the deployment descriptor. Although this header references an external URL at `www.bea.com`, WebLogic Server has its own copy of the DTD file, so your host server need not have access to the Internet. However, you must still include this `DOCTYPE` element in your `weblogic.xml` file, and have it reference the external URL since the version of the DTD is used to identify the version of this deployment descriptor.

```
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA  
Systems, Inc.//DTD Web Application 6.1//EN"  
"http://www.bea.com/servers/wls610/dtd/  
weblogic-web-jar.dtd">
```

```
<weblogic-web-app>
```

```
  <description>
```

```
    Text description of the Web App
```

```
  </description>
```

```
  <weblogic-version>
```

```
  </weblogic-version>
```

This element is not
used by WebLogic
Server.

Step 2: Map security role names to a security realm

<security-role-assignment>		
<role-name>		<i>(Required)</i>
name		
</role-name>		For more information, see “security-role-assignment Element” on page B-3.
<principal-name>		<i>(Required)</i>
name		
</principal-name>		
</security-role-assignment>		

If you need to define multiple roles, define each additional pair of `<role-name>` and `<principal-name>` tags within separate `<security-role-assignment>` elements.

Step 3 Map resources to JNDI

In this step you map resources used in your Web application to the JNDI tree. When you define an `<ejb-ref-name>` or a `<res-ref-name>` in the `web.xml` deployment descriptor, you also reference those names in `weblogic.xml` and map them to an actual JNDI name that is available in WebLogic Server. In the following example, a Data Source is referenced in a servlet with the name `myDataSource`. `myDataSource` is then referenced in `web.xml` and its data type defined. Finally, in the `weblogic.xml` file, `myDataSource` is mapped to the JNDI name `accountDataSource`, which is available in the JNDI tree. The JNDI name must match the name of an object bound in the JNDI tree. Objects can be bound to the JNDI tree programmatically or by configuring them in the Administration Console. For more information, see [Programming WebLogic JNDI](http://e-docs.bea.com/wls/docs61/jndi/index.html) at <http://e-docs.bea.com/wls/docs61/jndi/index.html>.

Servlet code:

```
javax.sql.DataSource ds = (javax.sql.DataSource) ctx.lookup
    ( "myDataSource" );
```

web.xml entries:

```
<resource-ref>
. . .
<res-ref-name>myDataSource</res-ref-name>
```



```

        <res-type>javax.sql.DataSource</res-type>
        <res-auth>CONTAINER</res-auth>
        . . .
    </resource-ref>

```

weblogic.xml entries:

```

<resource-description>
    <res-ref-name>myDataSource</res-ref-name>
    <jndi-name>accountDataSource</jndi-name>
</security-role-ref>

```

A similar pattern is used to map EJBs to the JNDI tree, but uses the `<ejb-ref-name>` element of the `<ejb-reference-description>` element in place of the `<res-ref-name>` element of the `<resource-description>` element.

<reference-descriptor>	For more information, see “reference-descriptor Element” on page B-3.
<resource-description>	For more information, see “resource-description Element” on page B-4.
<res-ref-name> name </res-ref-name>	(Required)
<jndi-name> JNDI name of resource </jndi-name>	(Required)
</resource-description>	
<ejb-reference-description>	
<ejb-ref-name> name </ejb-ref-name>	(Required) For more information, see “ejb-reference-description Element” on page B-4.

<code><jndi-name></code>	(Required)
JNDI name of EJB	
<code></jndi-name></code>	
<code></ejb-reference-description></code>	
<code></reference-descriptor></code>	

Step 4: Define session parameters

You define HTTP session parameters for this Web application inside of `<session-param>` tags, which are nested inside `<session-descriptor>` tags. For each `<session-param>` you need to supply a `<param-name>...</param-name>` element that names the parameter being defined and a `<param-value>...</param-value>` element that provides the value of the parameter. For a list of HTTP session parameters and details on setting them, see [“session-descriptor Element” on page B-4](#) .

<code><session-descriptor></code>	For more information, see “session-descriptor Element” on page B-4 .
<code><session-param></code>	
<code><param-name></code>	session param name
<code></param-name></code>	
<code><param-value></code>	my value
<code></param-value></code>	
<code></session-param></code>	
<code></session-descriptor></code>	

Step 5: Define JSP parameters

You define JSP configuration parameters for this Web application inside of `<jsp-param>` tags, which are nested in side `<jsp-descriptor>` tags. For each `<jsp-param>` you need to supply a `<param-name>...</param-name>` element that

names the parameter being defined and a `<param-value>...</param-value>` element that provides the value of the parameter. For a list of JSP parameters and details on setting them, see [“jsp-descriptor Element” on page B-11](#).

<code><jsp-descriptor></code>	For more information, see “jsp-descriptor Element” on page B-11 .
<code><jsp-param></code>	
<code><param-name></code> jsp param name <code></param-name></code>	
<code><param-value></code> my value <code></param-value></code>	
<code></jsp-param></code>	
<code></jsp-descriptor></code>	

Step 6: Define container parameters

There is one valid, optional element you can enter in the `<container-descriptor>` element, the `<check-auth-on-forward>` element.

<code><container-descriptor></code>	For more information, see “container-descriptor Element” on page B-13 .
<code><check-auth-on-forward/></code>	
<code><redirect-with-absolute-url></code> true false <code></redirect-with-absolute-url></code>	
<code></container-descriptor></code>	

Step 7: Define character set parameters

The optional `<charset-params>` element is used to define character set mappings.

<code><charset-params></code>	For more information, see “charset-params Element” on page B-14.
<code><input-charset></code>	
<code><resource-path></code> path to match <code></resource-path></code>	
<code><java-charset-name></code> name of Java character set <code></java-charset-name></code>	
<code></input-charset></code>	
<code><charset-mapping></code>	
<code><iana-charset-name></code> name of IANA character set <code></iana-charset-name></code>	
<code><java-charset-name></code> name of Java character set <code></java-charset-name></code>	
<code></charset-mapping></code>	
<code></charset-params></code>	

Step 8: Complete the descriptor file.

Close the descriptor file with the following tag:

```
</weblogic-web-app>
```

A web.xml Deployment Descriptor Elements

The following sections describe the deployment descriptor elements defined in the `web.xml` file. The root element for `web.xml` is `<web-app>`. The following elements are defined within the `<web-app>` element:

- “icon Element” on page A-4
 - `<small-icon>`
 - `<large-icon>`
- “display-name Element” on page A-4
- “description Element” on page A-5
- “distributable Element” on page A-5
- “context-param Element” on page A-6
 - `<param-name>`
 - `<param-value>`
 - `<description>`
- “filter Element” on page A-6
 - `<icon>`
 - `<filter-name>`
 - `<display-name>`
 - `<description>`
 - `<filter-class>`
 - `<init-param>`
- “filter-mapping Element” on page A-7
 - `<filter-name>`
 - `<url-pattern>`

<servlet>

- “listener Element” on page A-8
 - <listener-class>
- “servlet Element” on page A-8
 - <icon>
 - <small-icon>
 - <large-icon>
 - <servlet-name>
 - <display-name>
 - <description>
 - <servlet-class>
 - <jsp-file>
 - <init-param>
 - <param-name>
 - <param-value>
 - <description>
 - <load-on-startup>
 - <security-role ref>
 - <description>
 - <role-name>
 - <role-link>
- “servlet-mapping Element” on page A-12
 - <servlet-name>
 - <url-pattern>
- “session-config Element” on page A-13
 - <session-timeout>
- “mime-mapping Element” on page A-13
 - <extension>
 - <mime-type>
- “welcome-file-list Element” on page A-14
 - <welcome-file>
- “error-page Element” on page A-15
 - <error-code>
 - <exception-type>
 - <location>

-
- “taglib Element” on page A-16
 - <taglib-location>
 - <taglib-uri>
 - “resource-ref Element” on page A-17
 - <description>
 - <res-ref-name>
 - <res-type>
 - <res-auth>
 - <res-sharing-scope>
 - “security-constraint Element” on page A-18
 - <web-resource- collection>
 - <web-resource- name>
 - <description>
 - <url-pattern>
 - <http-method>
 - <auth-constraint>
 - <description>
 - <role-name>
 - <user-data- constraint>
 - <description>
 - <transport- guarantee>
 - “login-config Element” on page A-21
 - <auth-method>
 - <realm-name>
 - <form-login- config>
 - <form-login-page>
 - <form-error-page>
 - “security-role Element” on page A-22
 - <description>
 - <role-name>
 - “env-entry Element” on page A-23
 - <description>
 - <env-entry-name>
 - <env-entry-value>
 - <env-entry-type>
 - “ejb-ref Element” on page A-24
 - <description>

```
<ejb-ref-name>
<ejb-ref-type>
<home>
<remote>
<ejb-link>
<run-as>
```

icon Element

The `icon` element specifies the location within the Web Application for a small and large image used to represent the Web Application in a GUI tool. (The `servlet` element also has an element called the `icon` element, used to supply an icon to represent a servlet in a GUI tool.)

This element is not currently used by WebLogic Server.

The following table describes the elements you can define within an `icon` element.

Element	Required/ Optional	Description
<code><small-icon></code>	Optional	Location for a small (16x16 pixel) .gif or .jpg image used to represent the Web Application in a GUI tool. Currently, this is not used by WebLogic Server.
<code><large-icon></code>	Optional	Location for a large (32x32 pixel) .gif or .jpg image used to represent the Web Application in a GUI tool. Currently, this element is not used by WebLogic Server.

display-name Element

The optional `display-name` element specifies the Web Application display name, a short name that can be displayed by GUI tools.

Element	Required/ Optional	Description
<display-name>	Optional	Currently, this element is not used by WebLogic Server.

description Element

The optional description element provides descriptive text about the Web Application.

Element	Required/ Optional	Description
<description>	Optional	Currently, this element is not used by WebLogic Server.

distributable Element

The distributable element is not used by WebLogic Server.

Element	Required/ Optional	Description
<distributable>	Optional	Currently, this element is not used by WebLogic Server.

context-param Element

The optional `context-param` element declares a Web Application's servlet context initialization parameters. You set each context-param within a single context-param element, using `<param-name>` and `<param-value>` elements. You can access these parameters in your code using the

`javax.servlet.ServletContext.getInitParameter()` and
`javax.servlet.ServletContext.getInitParameterNames()` methods.

The following table describes the elements you can define within a `context-param` element.

Element	Required/ Optional	Description
<code><param-name></code>	Required	The name of a parameter.
<code><param-value></code>	Required	The value of a parameter.
<code><description></code>	Optional	A text description of a parameter.

filter Element

The filter element defines a filter class and its initialization parameters. For more information on filters, see [“Configuring Filters” on page 7-4](#).

The following table describes the elements you can define within a `servlet` element.

Element	Required/ Optional	Description
<code><icon></code>	Optional	Specifies the location within the Web Application for a small and large image used to represent the filter in a GUI tool. Contains a small-icon and large-icon element. Currently, this element is not used by WebLogic Server.

Element	Required/ Optional	Description
<code><filter-name></code>	Required	Defines the name of the filter, used to reference the filter definition elsewhere in the deployment descriptor.
<code><display-name></code>	Optional	A short name intended to be displayed by GUI tools.
<code><description></code>	Optional	A text description of the filter.
<code><filter-class></code>	Required	The fully-qualified class name of the filter.
<code><init-param></code>	Optional	Contains a name/value pair as an initialization parameter of the filter. Use a separate set of <code><init-param></code> tags for each parameter.

filter-mapping Element

The following table describes the elements you can define within a `filter-mapping` element.

Element	Required/ Optional	Description
<code><filter-name></code>	Required	The name of the filter to which you are mapping a URL pattern or servlet. This name corresponds to the name assigned in the <code><filter></code> element with the <code><filter-name></code> element.
<code><url-pattern></code>	Required - or map by <code><servlet></code>	<p>Describes a pattern used to resolve URLs. The portion of the URL after the <code>http://host:port + ContextPath</code> is compared to the <code><url-pattern></code> by WebLogic Server. If the patterns match, the filter mapped in this element is called.</p> <p>Example patterns:</p> <pre>/soda/grape/* /foo/* /contents *.foo</pre> <p>The URL must follow the rules specified in Section 10 of the Servlet 2.2 Specification.</p>

Element	Required/ Optional	Description
<code><servlet></code>	Required - or map by <code><url-pattern></code>	The name of a servlet which, if called, causes this filter to execute.

listener Element

Define an application listener using the listener element.

Element	Required/ Optional	Description
<code><listener-class></code>	Optional	Name of the class that responds to a Web Application event.

For more information, see [“Configuring an Event Listener” on page 6-5](#).

servlet Element

The `servlet` element contains the declarative data of a servlet.

If a `jsp-file` is specified and the `<load-on-startup>` element is present, then the JSP is precompiled and loaded when WebLogic Server starts.

The following table describes the elements you can define within a `servlet` element.

Element	Required/ Optional	Description
<code><icon></code>	Optional	Location within the Web Application for a small and large image used to represent the servlet in a GUI tool. Contains a small-icon and large-icon element. Currently, this element is not used by WebLogic Server.
<code><servlet-name></code>	Required	Defines the canonical name of the servlet, used to reference the servlet definition elsewhere in the deployment descriptor.
<code><display-name></code>	Optional	A short name intended to be displayed by GUI tools.
<code><description></code>	Optional	A text description of the servlet.
<code><servlet-class></code>	Required (or use <code><jsp-file></code>)	The fully-qualified class name of the servlet. Use only one of either the <code><servlet-class></code> tags or <code><jsp-file></code> tags in your servlet body.
<code><jsp-file></code>	Required (or use <code><servlet-class></code>)	The full path to a JSP file within the Web Application, relative to the Web Application root directory. Use only one of either the <code><servlet-class></code> tags or <code><jsp-file></code> tags in your servlet body.
<code><init-param></code>	Optional	Contains a name/value pair as an initialization parameter of the servlet. Use a separate set of <code><init-param></code> tags for each parameter.
<code><load-on-startup></code>	Require	WebLogic Server initializes this servlet when WebLogic Server starts up. The content of this element must be a positive integer indicating the order in which the servlet should be loaded. Lower integers are loaded before higher integers. If no value is specified, or if the value specified is not a positive integer, WebLogic Server can load the servlet in any order in the startup sequence.
<code><security-role-ref></code>	Optional	Used to link a security role name defined by <code><security-role></code> to an alternative role name that is hard coded in the servlet logic. This extra layer of abstraction allows the servlet to be configured at deployment without changing servlet code.

icon Element

This is an element within the “servlet Element” on page A-8.

The `icon` element specifies the location within the Web Application for small and large images used to represent the servlet in a GUI tool.

The following table describes the elements you can define within an `icon` element.

Element	Required/ Optional	Description
<code><small-icon></code>	Optional	Specifies the location within the Web Application for a small (16x16 pixel) <code>.gif</code> or <code>.jpg</code> image used to represent the servlet in a GUI tool. Currently, this element is not used by WebLogic Server.
<code><large-icon></code>	Optional	Specifies the location within the Web Application for a small (32x32 pixel) <code>.gif</code> or <code>.jpg</code> image used to represent the servlet in a GUI tool. Currently, this element is not used by WebLogic Server.

init-param Element

This is an element within the “servlet Element” on page A-8.

The optional `init-param` element contains a name/value pair as an initialization parameter of the servlet. Use a separate set of `init-param` tags for each parameter.

You can access these parameters with the `javax.servlet.ServletConfig.getInitParameter()` method.

The following table describes the elements you can define within a `init-param` element.

Element	Required/ Optional	Description
<code><param-name></code>	Required	Defines the name of this parameter.
<code><param-value></code>	Required	Defines a <code>String</code> value for this parameter.

Element	Required/ Optional	Description
<description>	Optional	Text description of the initialization parameter.

WebLogic Server recognizes the special initialization parameter, `wl-dispatch-policy`, to assign a servlet or JSP to an available execute queue. For example, the following example assigns a servlet to use the execute threads available in an execute queue named `CriticalWebApp`:

```
<servlet>
...
  <init-param>
    <param-name>wl-dispatch-policy</param-name>
    <param-value>CriticalWebApp</param-value>
  </init-param>
</servlet>
```

If the `CriticalWebApp` queue is not available, the servlet will use execute threads available in the default WebLogic Server execute queue. See [Setting Thread Count](#) for more information about configuring execute threads in WebLogic Server. See [Using Execute Queues to Control Thread Usage](#) for more information about creating and using queues.

security-role-ref Element

This is an element within the “servlet Element” on page A-8.

The `security-role-ref` element links a security role name defined by `<security-role>` to an alternative role name that is hard-coded in the servlet logic. This extra layer of abstraction allows the servlet to be configured at deployment without changing servlet code.

The following table describes the elements you can define within a `security-role-ref` element.

Element	Required/ Optional	Description
<description>	Optional	Text description of the role.

Element	Required/ Optional	Description
<code><role-name></code>	Required	Defines the name of the security role or principal that is used in the servlet code.
<code><role-link></code>	Required	Defines the name of the security role that is defined in a <code><security-role></code> element later in the deployment descriptor.

servlet-mapping Element

The `servlet-mapping` element defines a mapping between a servlet and a URL pattern.

The following table describes the elements you can define within a `servlet-mapping` element.

Element	Required/ Optional	Description
<code><servlet-name></code>	Required	The name of the servlet to which you are mapping a URL pattern. This name corresponds to the name you assigned a servlet in a <code><servlet></code> declaration tag.
<code><url-pattern></code>	Required	<p>Describes a pattern used to resolve URLs. The portion of the URL after the <code>http://host:port + WebAppName</code> is compared to the <code><url-pattern></code> by WebLogic Server. If the patterns match, the servlet mapped in this element will be called.</p> <p>Example patterns:</p> <pre>/soda/grape/* /foo/* /contents *.foo</pre> <p>The URL must follow the rules specified in Section 10 of the Servlet 2.2 Specification.</p> <p>For additional examples of servlet mapping, see “Servlet Mapping” on page 3-2.</p>

session-config Element

The `session-config` element defines the session parameters for this Web Application.

The following table describes the element you can define within a `session-config` element.

Element	Required/ Optional	Description
<code><session-timeout></code>	Optional	<p>The number of minutes after which sessions in this Web Application expire. The value set in this element overrides the value set in the <code>TimeoutSecs</code> parameter of the <code><session-descriptor></code> element in the WebLogic-specific deployment descriptor <code>weblogic.xml</code>, unless one of the special values listed here is entered.</p> <p>Default value: -2</p> <p>Maximum value: <code>Integer.MAX_VALUE ÷ 60</code></p> <p>Special values:</p> <ul style="list-style-type: none">■ -2 = Use the value set by <code>TimeoutSecs</code> in <code><session-descriptor></code> element of <code>weblogic.xml</code>■ -1 = Sessions do not timeout. The value set in <code><session-descriptor></code> element of <code>weblogic.xml</code> is ignored. <p>For more information, see “session-descriptor Element” on page B-4.</p>

mime-mapping Element

The `mime-mapping` element defines a mapping between an extension and a mime type.

The following table describes the elements you can define within a mime-mapping element.

Element	Required/ Optional	Description
<extension>	Required	A string describing an extension, for example: <code>txt</code> .
<mime-type>	Required	<p>A string describing the defined mime type, for example: <code>text/plain</code>. The following default mime types are provided with WebLogic Server (you can override these):</p> <pre>setIfNone(mimeTypesMap, "html", "text/html"); setIfNone(mimeTypesMap, "htm", "text/html"); setIfNone(mimeTypesMap, "gif", "image/gif"); setIfNone(mimeTypesMap, "jpeg", "image/jpeg"); setIfNone(mimeTypesMap, "jpg", "image/jpeg"); setIfNone(mimeTypesMap, "pdf", "application/pdf"); setIfNone(mimeTypesMap, "zip", "application/zip"); setIfNone(mimeTypesMap, "class", "application/x-java-vm"); setIfNone(mimeTypesMap, "jar", "application/x-java-archive"); setIfNone(mimeTypesMap, "ser", "application/x-java-serialized-object"); setIfNone(mimeTypesMap, "exe", "application/octet-stream"); setIfNone(mimeTypesMap, "txt", "text/plain"); setIfNone(mimeTypesMap, "java", "text/plain"); // This is for JavaWebStart out of the box work setIfNone(mimeTypesMap, "jnlp", "application/x-java-jnlp-file");</pre>

welcome-file-list Element

The optional `welcome-file-list` element contains an ordered list of `welcome-file` elements.

When the URL request is a directory name, WebLogic Server serves the first file specified in this element. If that file is not found, the server then tries the next file in the list.

For more information, see “[Configuring Welcome Pages](#)” on page 3-7 and [How WebLogic Server Resolves HTTP Requests](#) at http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req.

The following table describes the element you can define within a `welcome-file-list` element.

Element	Required/ Optional	Description
<code><welcome-file></code>	Optional	File name to use as a default welcome file, such as <code>index.html</code>

error-page Element

The optional `error-page` element specifies a mapping between an error code or exception type to the path of a resource in the Web Application.

When an error occurs—while WebLogic Server is responding to an HTTP request, or as a result of a Java exception—WebLogic Server returns an HTML page that displays either the HTTP error code or a page containing the Java error message. You can define your own HTML page to be displayed in place of these default error pages or in response to a Java exception.

For more information, see “[Customizing HTTP Error Responses](#)” on page 3-9 and [How WebLogic Server Resolves HTTP Requests](#) at http://e-docs.bea.com/wls/docs61/adminguide/web_server.html#resolve_http_req.

The following table describes the elements you can define within an `error-page` element.

Note: Define either an `<error-code>` or an `<exception-type>` but not both.

Element	Required/ Optional	Description
<code><error-code></code>	Optional	A valid HTTP error code, for example, <code>404</code> .

Element	Required/ Optional	Description
<code><exception-type></code>	Optional	A fully-qualified class name of a Java exception type, for example, <code>java.lang.string</code>
<code><location></code>	Required	The location of the resource to display in response to the error. For example, <code>/myErrorPg.html</code> .

taglib Element

The optional `taglib` element describes a JSP tag library.

This element associates the location of a JSP Tag Library Descriptor (TLD) with a URI pattern. Although you can specify a TLD in your JSP that is relative to the `WEB-INF` directory, you can also use the `<taglib>` tag to configure the TLD when deploying your Web Application. Use a separate element for each TLD.

The following table describes the elements you can define within a `taglib` element.

Element	Required/ Optional	Description
<code><taglib-location></code>	Required	Gives the file name of the tag library descriptor relative to the root of the Web Application. It is a good idea to store the tag library descriptor file under the <code>WEB-INF</code> directory so it is not publicly available over an HTTP request.
<code><taglib-uri></code>	Required	Describes a URI, relative to the location of the <code>web.xml</code> document, identifying a Tag Library used in the Web Application. If the URI matches the URI string used in the <code>taglib</code> directive on the JSP page, this <code>taglib</code> is used.

resource-ref Element

The optional `resource-ref` element defines a reference lookup name to an external resource. This allows the servlet code to look up a resource by a “virtual” name that is mapped to the actual location at deployment time.

Use a separate `<resource-ref>` element to define each external resource name. The external resource name is mapped to the actual location name of the resource at deployment time in the WebLogic-specific deployment descriptor `weblogic.xml`.

The following table describes the elements you can define within a `resource-ref` element.

Element	Required/ Optional	Description
<code><description></code>	Optional	A text description.
<code><res-ref-name></code>	Required	The name of the resource used in the JNDI tree. Servlets in the Web Application use this name to look up a reference to the resource.
<code><res-type></code>	Required	The Java type of the resource that corresponds to the reference name. Use the full package name of the Java type.
<code><res-auth></code>	Required	Used to control the resource sign on for security. If set to <code>APPLICATION</code> , indicates that the application component code performs resource sign on programmatically. If set to <code>CONTAINER</code> WebLogic Server uses the security context established with the <code>login-config</code> element. See “login-config Element” on page A-21 .
<code><res-sharing-scope></code>	Optional	Specifies whether connections obtained through the given resource manager connection factory reference can be shared. Valid values: <ul style="list-style-type: none">■ Shareable■ Unshareable

security-constraint Element

The `security-constraint` element defines the access privileges to a collection of resources defined by the `<web-resource-collection>` element.

For more information, see [“Configuring Security in Web Applications” on page 5-1](#).

The following table describes the elements you can define within a `security-constraint` element.

Element	Required/ Optional	Description
<code><web-resource-collection></code>	Required	Defines the components of the Web Application to which this security constraint is applied.
<code><auth-constraint></code>	Optional	Defines which groups or principals have access to the collection of web resources defined in this security constraint. See also “auth-constraint Element” on page A-19 .
<code><user-data-constraint></code>	Optional	Defines how the client should communicate with the server. See also “user-data-constraint Element” on page A-20 .

web-resource-collection Element

Each `<security-constraint>` element must have one or more `<web-resource-collection>` elements. These define the area of the Web Application to which this security constraint is applied.

This is an element within the “security-constraint Element” on page A-18.

The following table describes the elements you can define within a `web-resource-collection` element.

Element	Required/ Optional	Description
<code><web-resource-name></code>	Required	The name of this Web resource collection.
<code><description></code>	Optional	A text description of this security constraint.
<code><url-pattern></code>	Optional	Use one or more of the <code><url-pattern></code> elements to declare to which URL patterns this security constraint applies. If you do not use at least one of these elements, this <code><web-resource-collection></code> is ignored by WebLogic Server.
<code><http-method></code>	Optional	Use one or more of the <code><http-method></code> elements to declare which HTTP methods (usually, GET or POST) are subject to the authorization constraint. If you omit the <code><http-method></code> element, the default behavior is to apply the security constraint to all HTTP methods.

auth-constraint Element

This is an element within the “security-constraint Element” on page A-18.

The optional `auth-constraint` element defines which groups or principals have access to the collection of Web resources defined in this security constraint.

The following table describes the elements you can define within an `auth-constraint` element.

Element	Required/ Optional	Description
<code><description></code>	Optional	A text description of this security constraint.
<code><role-name></code>	Optional	Defines which security roles can access resources defined in this security-constraint. Security role names are mapped to principals using the <code>security-role-ref</code> Element. See “security-role-ref Element” on page A-11 .

user-data-constraint Element

This is an element within the “security-constraint Element” on page A-18.

The `user-data-constraint` element defines how the client should communicate with the server.

The following table describes the elements you may define within a `user-data-constraint` element.

Element	Required/ Optional	Description
<code><description></code>	Optional	A text description.
<code><transport-guarantee></code>	Required	<p>Specifies that the communication between client and server. WebLogic Server establishes a Secure Sockets Layer (SSL) connection when the user is authenticated using the <code>INTEGRAL</code> or <code>CONFIDENTIAL</code> transport guarantee.</p> <p>Range of values:</p> <ul style="list-style-type: none">■ <code>NONE</code>—The application does not require any transport guarantees.■ <code>INTEGRAL</code>—The application requires that the data be sent between the client and server in such a way that it cannot be changed in transit.■ <code>CONFIDENTIAL</code>—The application requires that data be transmitted so as to prevent other entities from observing the contents of the transmission.

login-config Element

Use the optional `login-config` element to configure how the user is authenticated; the realm name that should be used for this application; and the attributes that are needed by the form login mechanism.

If this element is present, the user must be authenticated in order to access any resource that is constrained by a `<security-constraint>` defined in the Web Application. Once authenticated, the user can be authorized to access other resources with access privileges.

The following table describes the elements you can define within a `login-config` element.

Element	Required/ Optional	Description
<code><auth-method></code>	Optional	Specifies the method used to authenticate the user. Possible values: BASIC - uses browser authentication FORM - uses a user-written HTML form CLIENT-CERT
<code><realm-name></code>	Optional	<p>The name of the realm that is referenced to authenticate the user credentials. If omitted, the realm defined with the Auth Realm Name field on the Web Application → Configuration → Other tab of the Administration Console is used by default. For more information, see Specifying a Security Realm at <code>http://e-docs.bea.com/wls/docs61/adminguide/cnfgsec.html#cnfgsec004</code>.</p> <p>Note: The <code><realm-name></code> element does not refer to security realms within WebLogic Server. This element defines the realm name to use in HTTP Basic authorization.</p> <p>Note: The system security realm is a collection of security information that is checked when certain operations are performed in the server. The servlet security realm is a different collection of security information that is checked when a page is accessed and basic authentication is used.</p>

Element	Required/ Optional	Description
<code><form-login-config></code>	Optional	Use this element if you configure the <code><auth-method></code> to FORM. See “form-login-config Element” on page A-22 .

form-login-config Element

This is an element within the “login-config Element” on page A-21.

Use the `<form-login-config>` element if you configure the `<auth-method>` to FORM.

.

Element	Required/ Optional	Description
<code><form-login-page></code>	Required	The URI of a Web resource relative to the document root, used to authenticate the user. This can be an HTML page, JSP, or HTTP servlet, and must return an HTML page containing a FORM that conforms to a specific naming convention. For more information, see “Setting Up Authentication for Web Applications” on page 5-2 .
<code><form-error-page></code>	Required	The URI of a Web resource relative to the document root, sent to the user in response to a failed authentication login.

security-role Element

The following table describes the elements you can define within a `security-role` element.

Element	Required/ Optional	Description
<code><description></code>	Optional	A text description of this security role.

Element	Required/ Optional	Description
<code><role-name></code>	Required	The role name. The name you use here must have a corresponding entry in the WebLogic-specific deployment descriptor, <code>weblogic.xml</code> , which maps roles to principals in the security realm. For more information, see “security-role-assignment Element” on page B-3 .

env-entry Element

The optional `env-entry` element declares an environment entry for an application. Use a separate element for each environment entry.

The following table describes the elements you can define within an `env-entry` element.

Element	Required/ Optional	Description
<code><description></code>	Optional	A textual description.
<code><env-entry-name></code>	Required	The name of the environment entry.
<code><env-entry-value></code>	Required	The value of the environment entry.
<code><env-entry-type></code>	Required	The type of the environment entry. Can be set to one of the following Java types: <code>java.lang.Boolean</code> <code>java.lang.String</code> <code>java.lang.Integer</code> <code>java.lang.Double</code> <code>java.lang.Float</code>

ejb-ref Element

The optional `ejb-ref` element defines a reference to an EJB resource. This reference is mapped to the actual location of the EJB at deployment time by defining the mapping in the WebLogic-specific deployment descriptor file, `weblogic.xml`. Use a separate `<ejb-ref>` element to define each reference EJB name.

The following table describes the elements you can define within an `ejb-ref` element.

Element	Required/ Optional	Description
<code><description></code>	Optional	A text description of the reference.
<code><ejb-ref-name></code>	Required	The name of the EJB used in the Web Application. This name is mapped to the JNDI tree in the WebLogic-specific deployment descriptor <code>weblogic.xml</code> . For more information, see “ejb-reference-description Element” on page B-4 .
<code><ejb-ref-type></code>	Required	The expected Java class type of the referenced EJB.
<code><home></code>	Required	The fully qualified class name of the EJB home interface.
<code><remote></code>	Required	The fully qualified class name of the EJB remote interface.
<code><ejb-link></code>	Optional	The <code><ejb-name></code> of an EJB in an encompassing J2EE application package.
<code><run-as></code>	Optional	A security role whose security context is applied to the referenced EJB. Must be a security role defined with the <code><security-role></code> element.

B weblogic.xml

Deployment Descriptor Elements

This following sections describe the deployment descriptor elements that you define in the `weblogic.xml` file. The root element for `weblogic.xml` is `<weblogic-web-app>`. The following elements are defined within the `<weblogic-web-app>` element:

- “description Element” on page B-2
- “weblogic-version Element” on page B-2
- “security-role-assignment Element” on page B-3
 - `<role-name>`
 - `<principal-name>`
- “reference-descriptor Element” on page B-3
 - resource-description Element
 - `<res-ref-name>`
 - `<jndi-name>`
 - ejb-reference-description Element
 - `<ejb-ref-name>`
 - `<jndi-name>`
- “session-descriptor Element” on page B-4
 - Session Parameter Names and Values
- “jsp-descriptor Element” on page B-11
 - JSP Parameter Names and Values

- “container-descriptor Element” on page B-13
check-auth-on-forward Element
- “charset-params Element” on page B-14
input-charset Element
 - <resource-path>
 - <java-charset-name>charset-mapping Element
 - <iana-charset-name>
 - <java-charset-name>

The DOCTYPE header for `weblogic.xml` is as follows:

```
<!DOCTYPE weblogic-web-app PUBLIC
"-//BEA Systems, Inc.//DTD Web Application 6.1//EN"
"http://www.bea.com/servers/wls610/dtd/weblogic-web-jar.dtd">
```

You can also access the Document Type Descriptor (DTD) for `weblogic.xml` at <http://www.bea.com/servers/wls610/dtd/weblogic-web-jar.dtd>.

description Element

The description element is a text description of the Web Application.

weblogic-version Element

The `weblogic-version` element indicates the version of WebLogic Server on which this Web Application is intended to be deployed. This element is informational only and is not used by WebLogic Server.

security-role-assignment Element

The `security-role-assignment` element declares a mapping between a security role and one or more principals in the realm, as shown in the following example.

```
<security-role-assignment>
  <role-name>PayrollAdmin</role-name>
  <principal-name>Tanya</principal-name>
  <principal-name>Fred</principal-name>
  <principal-name>system</principal-name>
</security-role-assignment>
```

The following table describes the elements you can define within a `security-role-assignment` element.

Element	Required Optional	Description
<code><role-name></code>	Required	Specifies the name of a security role.
<code><principal-name></code>	Required	Specifies the name of a principal that is defined in the security realm. You can use multiple <code><principal-name></code> elements to map principals to a role. For more information on security realms, see the <i>Programming WebLogic Security</i> at http://e-docs.bea.com/wls/docs61/security/index.html .

reference-descriptor Element

The `reference-descriptor` element maps a name used in the Web Application to the JNDI name of a server resource. The `reference-description` element contains two elements: The `resource-description` element maps a resource, for example, a `DataSource`, to its JNDI name. The `ejb-reference` element maps an EJB to its JNDI name.

resource-description Element

The following table describes the elements you can define within a `resource-description` element.

Element	Required/ Optional	Description
<code><res-ref-name></code>	Required	Specifies the name of a resource reference.
<code><jndi-name></code>	Required	Specifies a JNDI name for the resource.

ejb-reference-description Element

The following table describes the elements you can define within a `ejb-reference-description` element.

Element	Required/ Optional	Description
<code><ejb-ref-name></code>	Required	Specifies the name of an EJB reference used in your Web Application.
<code><jndi-name></code>	Required	Specifies a JNDI name for the reference.

session-descriptor Element

The `session-descriptor` element defines parameters for HTTP sessions, as shown in the following example:

```
<session-descriptor>
  <session-param>
    <param-name>
      CookieDomain
    </param-name>
    <param-value>
      myCookieDomain
    </param-value>
  </session-param>
</session-descriptor>
```



```

        </param-value>
    </session-param>
</session-descriptor>

```

Session Parameter Names and Values

The following table describes the valid session parameter names and values you can define within a `session-param` element:

Parameter Name	Default Value	Parameter Value
CookieDomain	Null	<p>Specifies the domain for which the cookie is valid. For example, setting <code>CookieDomain</code> to <code>.mydomain.com</code> returns cookies to any server in the <code>*.mydomain.com</code> domain.</p> <p>The domain name must have at least two components. Setting a name to <code>*.com</code> or <code>*.net</code> is not valid.</p> <p>If unset, this parameter defaults to the server that issued the cookie.</p> <p>For more information, see <code>Cookie.setDomain()</code> in the Servlet specification from Sun Microsystems.</p>
CookieComment	Weblogic Server Session Tracking Cookie	<p>Specifies the comment that identifies the session tracking cookie in the cookie file.</p> <p>If unset, this parameter defaults to <code>WebLogic Session Tracking Cookie</code>. You may provide a more specific name for your application.</p>

B *weblogic.xml Deployment Descriptor Elements*

Parameter Name	Default Value	Parameter Value
CookieMaxAgeSecs	-1	<p>Sets the life span of the session cookie, in seconds, after which it expires on the client.</p> <p>If the value is 0, the cookie expires immediately.</p> <p>The maximum value is <code>Integer.MAX_VALUE</code>, where the cookie lasts forever.</p> <p>If set to -1, the cookie expires when the user exits the browser.</p> <p>For more information about cookies, see “Using Sessions and Session Persistence in Web Applications” on page 4-1.</p>
CookieName	JSESSIONID	<p>Defines the session cookie name. Defaults to <code>JSESSIONID</code> if unset. You may set this to a more specific name for your application. When using <code>ProxyByExtension</code>, you may use either the <code>jsessionid</code> identifier or the <code>?jsessionid</code> identifier to pass rewritten URLs.</p>
CookiePath	Null	<p>Specifies the pathname to which the browser sends cookies.</p> <p>If unset, this parameter defaults to <code>/</code> (slash), where the browser sends cookies to all URLs served by WebLogic Server. You may set the path to a narrower mapping, to limit the request URLs to which the browser sends cookies.</p>
CookiesEnabled	true	<p>Use of session cookies is enabled by default and is recommended, but you can disable them by setting this property to <code>false</code>. You might turn this option off to test Using URL Rewriting.</p>
CookieSecure	false	<p>If set, the client's browser will only send the cookie back over an HTTPS connection. This ensures that the cookie ID is secure and should only be used on websites that exclusively use HTTPS. Once this feature is enabled, session cookies over HTTP will no longer work; if your client is directed to a non-HTTPS location the session will not be sent.</p>

Parameter Name	Default Value	Parameter Value
EncodeSessionIdInQueryParams	false	<p>By default, when you use the <code>HttpServletResponse.encodeURL(URL)</code> method to encode a URL in the HTTP response, the session identifier is added to the URL as a path parameter after the <code>;</code> character in the URL. This behavior is defined by the Servlet 2.3 J2EE specification, implemented as of Version 6.1 of WebLogic Server.</p> <p>In Versions 6.0 and previous of WebLogic Server, however, the default behavior was to add the session identifier as a query parameter after the <code>?</code> character in the URL. To enable this old behavior, set this session parameter to <code>true</code>.</p> <p>Note: You typically use this parameter when WebLogic Server interacts with Web Servers that do not completely comply with the Servlet 2.3 specification.</p>
InvalidationIntervalSecs	60	<p>Sets the time, in seconds, that WebLogic Server waits between doing house-cleaning checks for timed-out and invalid sessions, and deleting the old sessions and freeing up memory. Use this parameter to tune WebLogic Server for best performance on high traffic sites.</p> <p>The minimum value is every second (1). The maximum value is once a week (604,800 seconds). If unset, the parameter defaults to 60 seconds.</p>

B *weblogic.xml Deployment Descriptor Elements*

Parameter Name	Default Value	Parameter Value
PersistentStoreDir	session_db	<p>If you have set PersistentStoreType to file, this parameter sets the directory path where WebLogic Server will store the sessions. The directory path is either relative to the temp directory or an absolute path. The temp directory is either a generated directory under the WEB-INF directory of the Web Application, or a directory specified by the context-param <code>javax.servlet.context.tmpdir</code>.</p> <p>Ensure that you have enough disk space to store the <i>number of valid sessions</i> multiplied by the <i>size of each session</i>. You can find the size of a session by looking at the files created in the PersistentStoreDir. Note that the size of each session can vary as the size of serialized session data changes.</p> <p>You can make file-persistent sessions clusterable by making this directory a shared directory among different servers.</p> <p>You must create this directory manually.</p>
PersistentStoreTable	wl_servlet_sessions	<p>Applies only when PersistentStoreType is set to jdbc. This is used when you choose a database table name other than the default.</p>
PersistentStorePool	None	<p>Specifies the name of a JDBC connection pool to be used for persistence storage.</p> <p>For more information on setting up a database connection pool, see Managing JDBC Connectivity at http://e-docs.bea.com/wls/docs61/adminguide/jdbc.html.</p>

Parameter Name	Default Value	Parameter Value
PersistentStoreType	memory	<p>Sets the persistent store method to one of the following options:</p> <ul style="list-style-type: none">■ <code>memory</code>—Disables persistent session storage.■ <code>file</code>—Uses file-based persistence (See also <code>PersistentStoreDir</code>, above).■ <code>jdbc</code>—Uses a database to store persistent sessions. (see also <code>PersistentStorePool</code>, above).■ <code>replicated</code>—Same as <code>memory</code>, but session data is replicated across the clustered servers.■ <code>cookie</code>—All session data is stored in a cookie in the user’s browser.■ <code>replicated_if_clustered</code>—If the Web application is deployed on a clustered server, the in-effect <code>PersistentStoreType</code> will be replicated. Otherwise, <code>memory</code> is the default.
PersistentStoreCookieName	WLCookie	<p>Sets the name of the cookie used for cookie-based persistence. For more information, see “Using Cookie-Based Session Persistence” on page 4-9.</p>
IDLength	52	<p>Sets the size of the session ID.</p> <p>The minimum value is 8 bytes and the maximum value is <code>Integer.MAX_VALUE</code>.</p> <p>If you are writing a WAP application, you must use URL rewriting because the WAP protocol does not support cookies. Also, some WAP devices have a 128-character limit on URL length (including parameters), which limits the amount of data that can be transmitted using URL re-writing. To allow more space for parameters, use this parameter to limit the size of the session ID that is randomly generated by WebLogic Server.</p>

B *weblogic.xml Deployment Descriptor Elements*

Parameter Name	Default Value	Parameter Value
TimeoutSecs	3600	<p>Sets the time, in seconds, that WebLogic Server waits before timing out a session, where x is the number of seconds between a session's activity.</p> <p>Minimum value is 1, default is 3600, and maximum value is integer MAX_VALUE.</p> <p>On busy sites, you can tune your application by adjusting the timeout of sessions. While you want to give a browser client every opportunity to finish a session, you do not want to tie up the server needlessly if the user has left the site or otherwise abandoned the session.</p> <p>This parameter can be overridden by the <code>session-timeout</code> element (defined in minutes) in <code>web.xml</code>. For more information, see “session-config Element” on page A-13.</p>
JDBCConnectionTimeoutSecs	120	<p>Sets the time, in seconds, that WebLogic Server waits before timing out a JDBC connection, where x is the number of seconds between.</p>
URLRewritingEnabled	true	<p>Enables URL rewriting, which encodes the session ID into the URL and provides session tracking if cookies are disabled in the browser.</p>
ConsoleMainAttribute		<p>If you enable Session Monitoring in the WebLogic Server Administration Console, set this parameter to the name of the session parameter you will use to identify each session that is monitored. For more information, see Monitoring a WebLogic Domain at http://e-docs.bea.com/wls/docs61/adminguide/monitoring.html.</p>

jsp-descriptor Element

The `jsp-descriptor` element defines parameter names and values for JSPs. You define the parameters as name/value pairs. The following example shows how to configure the `compileCommand` parameter. Enter all of the JSP configurations using the pattern demonstrated in this example:

```
<jsp-descriptor>
  <jsp-param>
    <param-name>
      compileCommand
    </param-name>
    <param-value>
      sj
    </param-value>
  </jsp-param>
</jsp-descriptor>
```

JSP Parameter Names and Values

The following table describes the parameter names and values you can define within a `<jsp-param>` element.

Parameter Name	Default Value	Parameter Value
<code>compileCommand</code>	<code>javac</code> , or the Java compiler defined for a server under the configuration /tuning tab of the WebLogic Server Administration Console	Specifies the full pathname of the standard Java compiler used to compile the generated JSP servlets. For example, to use the standard Java compiler, specify its location on your system as shown below: <pre><param-value> /jdk130/bin/javac.exe </param-value></pre> For faster performance, specify a different compiler, such as IBM Jikes or Symantec <code>sj</code> .

B *weblogic.xml Deployment Descriptor Elements*

Parameter Name	Default Value	Parameter Value
compileFlags	None	Passes one or more command-line flags to the compiler. Separate multiple flags with a space. For example: <pre><jsp-param> <param-name>compileflags</param-name> <param-value>-g -v</param-value> </jsp-param></pre>
compilerclass	None	Name of a Java compiler that is executed in WebLogic Servers's virtual machine. (Used in place of an executable compiler such as <code>javac</code> or <code>sj</code> .) If this parameter is set, the <code>compileCommand</code> parameter is ignored.
encoding	Default encoding of your platform	Specifies the default character set used in the JSP page. Use standard Java character set names (see http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.htm). If unset, this parameter defaults to the encoding for your platform. A JSP page directive (included in the JSP code) overrides this setting. For example: <pre><%@ page contentType="text/html; charset=custom-encoding"%></pre>
compilerSupports Encoding	true	When set to true, the JSP compiler uses the encoding specified with the <code>contentType</code> attribute contained in the <code>page</code> directive on the JSP page, or, if a <code>contentType</code> is not specified, the encoding defined with the <code>encoding</code> parameter in the <code>jsp-descriptor</code> . When set to false, the JSP compiler uses the default encoding for the JVM when creating the intermediate <code>.java</code> file.
keepgenerated	false	Saves the Java files that are generated as an intermediary step in the JSP compilation process. Unless this parameter is set to true, the intermediate Java files are deleted after they are compiled.
noTryBlocks	false	If a JSP file has numerous or deeply nested custom JSP tags and you receive a <code>java.lang.VerifyError</code> exception when compiling, use this flag to allow the JSPs to compile correctly.
packagePrefix	jsp_servlet	Specifies the package into which all JSP pages are compiled.

Parameter Name	Default Value	Parameter Value
pageCheckSeconds	1	Sets the interval, in seconds, at which WebLogic Server checks to see if JSP files have changed and need recompiling. Dependencies are also checked and recursively reloaded if changed. If set to 0, pages are checked on every request. If set to -1, the page is not checked until server restart. Any classes used by the JSP page that live in the servlet classpath are also re-loaded.
precompile	false	When set to true, WebLogic Server automatically precompiles all modified JSPs when the Web Application is deployed or re-deployed or when starting WebLogic Server.
verbose	true	When set to true, debugging information is printed out to the browser, the command prompt, and WebLogic Server log file.
workingDir	internally generated directory	The name of a directory where WebLogic Server saves the generated Java and compiled class files for a JSP.
compiler	javac	Sets the JSP compiler for use with this instance of WebLogic Server.
superclass	weblogic.servlet.jsp.Base	Provides a means to override the default superclass for JSPs. The JSPs are compiled as servlet classes extending from this base class.
precompileContinue	false	When set to true, WebLogic Server continues precompiling all modified JSPs even if some of those JSPs fail during compilation. Only takes effect when precompile is set to true.

container-descriptor Element

The `<container-descriptor>` element defines general parameters for Web Applications.

check-auth-on-forward Element

Add the `<check-auth-on-forward/>` element when you want to require authentication of forwarded requests from a servlet or JSP. Omit the tag if you do not want to require re-authentication. For example:

```
<container-descriptor>
  <check-auth-on-forward/>
</container-descriptor>
```

Note that the default behavior has changed with the release of the Servlet 2.3 specification, which states that authentication is not required for forwarded requests.

redirect-with-absolute-url

The `<redirect-with-absolute-url>` element controls whether the `javax.servlet.http.HttpServletResponse.SendRedirect()` method redirects using a relative or absolute URL. Set this element to `false` if you are using a proxy HTTP server and do not want the URL converted to a non-relative link.

The default behavior is to convert the URL to a non-relative link.

charset-params Element

The `<charset-params>` Element is used to define codeset behavior for non-unicode operations.

input-charset Element

Use the `<input-charset>` element to define which character set is used to read `GET` and `POST` data. For example:

```
<input-charset>
  <resource-path>/foo</resource-path>
```

```
<java-charset-name>SJIS</java-charset-name>
</input-charset>
```

For more information, see [“Determining the Encoding of an HTTP Request” on page 3-16](#).

The following table describes the elements you can define within a `<input-charset>` element.

Element	Required/ Optional	Description
<code><resource-path></code>	Required	A path which, if included in the URL of a request, signals WebLogic Server to use the Java character set specified by <code><java-charset-name></code> .
<code><java-charset-name></code>	Required	Specifies the Java characters set to use.

charset-mapping Element

Use the `<charset-mapping>` element to map an IANA character set name to a Java character set name. For example:

```
<charset-mapping>
  <iana-charset-name>Shift-JIS</iana-charset-name>
  <java-charset-name>SJIS</java-charset-name>
</charset-mapping>
```

For more information, see [“Mapping IANA Character Sets to Java Character Sets” on page 3-17](#).

The following table describes the elements you can define within a `<charset-mapping>` element.

Element	Required/ Optional	Description
<code><iana-charset-name></code>	Required	Specifies the IANA character set name that is to be mapped to the Java character set specified by the <code><java-charset-name></code> element.
<code><java-charset-name></code>	Required	Specifies the Java characters set to use.

Index

A

- administration console
 - using to deploy 2-8
- application events 7-1
- applications directory 2-2
- application event listeners 7-1
- AuthCookieEnabled 5-4
- authentication
 - and multiple web applications, and cookies 5-3
 - basic 5-2
 - client certificates 5-3
 - form-based 5-2
- auto-deployment 2-2, 2-3

C

- CGI 3-9
- chaining filters 8-6
- config.xml 5-4
- Configuration
 - JSP 3-5
 - JSP tag libraries 3-6
 - servlets 3-2
- cookies 4-3
 - authentication 5-3
 - URL rewriting 4-11
- customer support contact information xi

D

- default servlet 3-8
- deleting a Web Application 2-16
 - using applications directory 2-16
 - using the administration console 2-17
 - using weblogic.deploy 2-17
- deploying
 - Web Application 1-2
- deployment
 - applications directory 2-2
 - auto-deployment 2-3
 - overview 2-1
 - upload 2-5
 - using weblogic.deploy 2-9
 - with production mode enabled 2-7
- deployment descriptor
 - editing using Administration Console 1-6
 - re-deployment 2-11
- development mode 2-2
- directory structure 1-3
- document root 1-3
- documentation, where to find it x
- doFilter() 8-6

E

- EJB
 - and Web Applications 3-15
- error pages 3-9
- event listener

- declaration 7-5
- event listeners
 - configuring 7-5
- events
 - declaration 7-5
- exploded directory format
 - re-deployment 2-6

F

- filter class 8-6
- filter mapping 8-5
 - to a servlet 8-5
 - URL pattern 8-5
- filters
 - and Web Applications 8-3
 - chaining 8-6
 - configuring 8-4
 - declaration 8-4
 - mappings 8-5
 - overview 8-1
 - uses 8-3
 - writing a filter class 8-6

G

- GetFilterConfig() 8-6

H

- HTTP session events 7-4
- HTTP sessions 4-2
 - and redeployment 2-11
- HTTPS
 - accessing resources securely 5-4
- HTTPS
 - accessing resources securely 4-11

I

- init params 3-4
- in-memory replication 4-4

I-2 Assembling and Configuring Web Applications

J

- jar command
 - Web Applications 1-3
- JSP
 - configuration 3-5
 - modifying 2-10
 - refreshing 2-10
 - tag libraries 3-6

L

- listener
 - writing a listener class 7-5
- listener class 7-5
- listeners 7-1
 - configuring 7-5
 - HTTP session events 7-4
 - servlet context events 7-3

M

- mapping
 - filters 8-5
- modifying components 2-10
- modifying JSP 2-10

P

- persistence for sessions 4-4
- printing product documentation x
- production mode 2-2, 2-7
 - deployment using administration
 - console 2-8
- ProductionModeEnabled 2-3

R

- REDEPLOY file 2-6
- re-deployment 2-5
 - .war archive 2-5
 - and HTTP sessions 2-11

- exploded directory format 2-6
- of Java classes 2-11
- using administration console 2-6, 2-11
- using REDEPLOY file 2-6
- using weblogic.deploy 2-12
- when using auto-deployment 2-5

refreshing

- JSP 2-10

refreshing JSPs 2-13

refreshing static components 2-13

response 8-1

S

security

- applying programatically in servlet 5-6
- authentication 5-2
- client certificates 5-3
- constraints 5-5
- Web Applications 5-1

servlet

- configuration 3-2
- default servlet 3-8
- initialization parameters 3-4
- mapping 3-2
- url-pattern 3-2

servlet context events 7-3

session persistence

- file-based 4-6
- JDBC (database) 4-7
- single server 4-6

Session Timeout 4-2

sessions 4-2

- cookies 4-3
- persistence 4-4
- Session Timeout attribute 4-2
- setting up 4-2
- URL rewriting 4-11
- URL rewriting and WAP 4-12

setFilterConfig 8-6

support

technical xi

U

un-deployment 2-15

- using administration console 2-15
- using weblogic.deploy 2-15

upload 2-5

URL rewriting 4-11

W

WAP 4-12

Web Application

- configuring EJB 3-15
- configuring external resources 3-13
- default servlet 3-8
- deploying 1-2
- directory structure 1-3
- document root 1-3
- error page 3-9
- jar file 1-3
- security 5-1
- security constraint 5-5
- URI 1-4
- war file 1-3

WEB-INF directory 1-4

weblogic.deploy

- re-deployment 2-12
- using to delete a Web Application 2-17
- using to refreshing components 2-13

weblogic.deploy utility

- deploying with 2-9

weblogic.xml

- context-root 2-18

weblogic-deploy

- using to un-deploy 2-15

welcome pages 3-7