BEA WebLogic Portal™

Performance Tuning Guide

Version 8.1 with Service Pack 6
WebLogic Portal 8.1 Performance Tuning Guide
Document Revised: July 2006

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

4. Tuning Your Portal Web Application

A. Performance Tuning Checklists

B. WebLogic Portal Cache Settings

CHAPTER 1

General Performance Tuning Guidelines

Application performance is affected by many factors. This chapter discusses a few of the initial aspects that can affect performance and provides links to documentation resources that can assist you.

- Understanding Performance Tuning and BEA WebLogic Portal
- Tuning Your WebLogic Server
- Tuning Your JVM
- Tuning Your Database
- Tuning Your Operating System
- Upgrading to the Latest Service Packs
- Other Resources

Understanding Performance Tuning and BEA WebLogic Portal

Performance tuning is a process which spans development, staging and deployment. During all phases, performance should be monitored and appropriate adjustments made. If you are new to performance testing, see Approaches to Performance Testing on BEA's dev2dev website.

BEA recommends that you establish an environment where you can performance test the installation for the following reasons:

- Testing your prototype under load will help you validate design decisions early in the development cycle that may significantly alter the performance of your application.
- Any configuration change can dramatically affect application performance (hardware, database, clustering environment, application tuning parameters, and so on). Load testing your application whenever design changes are made provides a way to narrow down performance problems to a particular area.
- Testing early and often increases the likelihood that your site implementation will be successful and scalable.

The recommended approach for performance testing is to start with the simplest aspect of the installation and then move into areas of increased complexity. If you observe slow behavior in any portion of this testing process, you should begin a more thorough investigation into its causes.

General Architecture

First, perform the following steps to identify performance issues with your network, database, or other software that is independent of WebLogic Portal.

1. Test your database (independent of any web components) to determine how well your schema and SQL work. Note any areas where the schema or SQL may not be optimized for performance. For more information about configuring database connection pools, see http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc_connection_pools.html#1106131
2. Test your network for sufficient bandwidth, and check that the TCP/IP parameters on the server's operating system can sufficiently handle the application load you expect. It is possible that the network is the slowest aspect of your deployment.
3. Test your web server, ensuring that it has sufficient capacity to serve static HTML pages when many concurrent threads are running.
4. Ensure that you have enough resources available to meet application requirements.Most large applications are clustered, but keep in mind that a clustering environment requires resources to perform load-balancing tasks. For more information, see Understanding Cluster Configuration and Application Deployment.
5. Test your servlet engine by running a load test against a trivial servlet such as a HelloWorld servlet. If this simple servlet does not perform and scale horizontally (meaning that as you add Java Virtual Machines, performance increases accordingly), the performance problems you encounter may be related to an infrastructure or resource issue.

WebLogic Portal

Now, perform the following steps to identify performance issues with WebLogic Portal:

1. Verify that your BEA WebLogic Server database configuration is optimal. WebLogic Portal makes extensive use of the database. Check that your connection pool is large enough, and verify that your database handles connection failures in an efficient manner. For example, you may want to increase the number of connections at start up, increase the wait time before requesting new connections, determine whether your pool can shrink, and so on.
2. Verify that each portlet is optimized for speed as follows:
   - Avoid using forms in a portlet that update the data within the portlet. This causes the entire portal to refresh its data, which can be very time consuming.
   - Place items that require heavy processing in an edit page or a maximized URL. If you do not, the portal must wait for the portlet to process, and this considerably slows down the eventual rendering of the portal.
   - Avoid large amounts of data retrieval that can take significant time to process.
3. Test your application's components, starting from the data access layer. Then proceed toward the GUI one step at a time. Pay attention to performance and scalability differences at each component and between each layer of your application. Finally, do end-to-end testing from a browser-based load-testing tool using a proxy server,
4. Test the behavior and performance of your application under simulated, real-world conditions. (Many tools are available to help you do this.) Be sure to use both anonymous and logged-in users simultaneously.

Tuning Your WebLogic Server

Because WebLogic Portal runs on WebLogic Server, factors impacting the performance of WebLogic Server will also impact the performance of WebLogic Portal.

For more information about tuning WebLogic Server, see http://edocs.bea.com/wls/docs81/perform/index.html.

Tuning Your JVM

Your Java Virtual Machine is key to running your Portal efficiently. For more information about tuning WebLogic JRocket, see Tuning WebLogic JRocket JVM.

Recommendations

When using JRockit, adjust the `-gcx:parallel` flag, as mentioned in Identify the Best JVM Settings as found in the WebLogic Server Performance and Tuning Guide. JRockit typically gives better performance with WebLogic Server than SunMicrosystem's HotSpot.

When using Sun Hotspot, adjust the `-server` and `-client` flags to see which offers the maximum throughput for your application.

Tuning Your Database

Keeping your database tuned is an important part of using WebLogic Portal. Portal uses the database to store content, rules, portal framework customizations, and user profile data.

See your vendor database documentation on how best to tune your database for your needs and production environment.

For more information about database tuning for WebLogic Portal see, the WebLogic Portal Database Administration Guide.

Tuning Your Operating System

Tune your operating system according to your operating system documentation. BEA certifies WebLogic Server on multiple operating systems on the Supported Configurations pages.

For Windows platforms, the default settings are usually sufficient. However, the Solaris and Linux platforms usually need to be tuned appropriately.

If you plan on using WSRP portlets, be sure to modify the file descriptor parameters in the `/etc/system/limits.conf` file, as noted in the WebLogic Server Performance and Tuning Guide.

Upgrading to the Latest Service Packs

Service packs almost always include improvements to some area of performance. Service packs are available individually for download to Contract Support Customers. Go to the http://support.bea.com to login to eSupport. Navigate to Product Download and Service packs in the left navigation bar. Choose the product of interest and follow links to the version and service pack you are interested in.

In addition to WebLogic Portal service packs, you should also check available improvements with WebLogic Server. For example, the WebLogic Server proxy plug-in for SP4 contains

several performance improvements. For more information about proxy plug-ins, see Using Web Server Plug-ins with WebLogic Server.

Table 1-1 lists the available services packs for WebLogic Portal 8.1 and the performance improvements found in each.

Table 1-1 WebLogic Portal Service Pack Information

WebLogic Portal 8.1 SP2	• Improved performance of 25%-35% over 8.1 GA • Faster control tree creation • Faster content repository operations in a cluster • Faster portal rendering when the first user logs in • More caching (for example, caching `web.xml`) • Faster data binding engine (NetUI) • Faster User/Group tree performance in the Administration Portal for large group hierarchies
WebLogic Portal 8.1 SP3	• Faster portal rendering by bypassing servlet filters for framework JSPs • Faster entitlement evaluation, especially with a large number of entitlements • Faster algorithm for portal creation in the Administration Portal • New feature to replace the group tree with a text box in the Administration Portal, useful when you have a very large number of groups • Optimized framework-related JSPs • Faster WSRP by caching registration handles in the producer
WebLogic Portal 8.1 SP4	• Improved performance of 15%-70% over previous service packs. • Optimized entitlement role calculations. • Optimized content searches against the BEA repository. The degree of improvement depends on the type of search and the complexity of the expression. • Control tree optimization that allows partial generation of the tree where appropriate; performance improvements are substantial with large portals. • More robust control tree state management that improves scalability and performance in a clustered environment. • Improved Portal Preferences and Portlet Event/Event Handlers using a new lightweight UIControlData object that reduces overhead • Reduced memory requirements for Portal customizations that reduce memory requirements. User customizations are now internally optimized. As a result, the memory requirements for customizations have been significantly reduced from SP3.

Other Resources

Remember that WebLogic Portal uses many components from WebLogic Platform. See the following documentation for more information about tuning WebLogic Portal.

- Designing Portals for Optimal Performance
- WebLogic Portal Capacity Planning Guide
- WebLogic Server Performance and Tuning Guide
- WebLogic Server Capacity Planning Guide
- Tuning WebLogic JRocket JVM
- dev2dev

CHAPTER 2

Tuning Your Portal Domain

Key aspects of portal performance are managed at the domain level. These include:

- Tuning Your Domain Configuration
- Removing Debugging Tools from Your Domain
- Tuning for Users and Groups

Tuning Your Domain Configuration

Optimally, when you deploy, you need to create a new domain that is configured for your production environment, including clusters, production configuration settings, and so on.

However, if you have deployed a development domain and want to use it for production, you must change your domain environment settings to optimize performance.

Note: It is not recommended to use a development domain for production, see http://edocs.bea.com/platform/docs81/confgwiz/newdom.html#1059076 for more information.

The domain settings are managed by the `setDomainEnv.cmd` (or `setDomainEnv.sh`) script which is found in your domain directory. By default, the script is found in: *`bea-home`*/user_projects/*`domain_name`*/`setDomainEnv.cmd/sh`.

To edit this file, open it in a text editor.

Table 2-1 lists the start script settings and their appropriate values for a production domain. Remember if you are using a domain that was created for production mode, you do not need to modify the configuration.

Table 2-1 setDomainEnv Settings

Flag Name	Production Mode Setting	Notes
`DOMAIN_PRODUCTION_MODE`	`true`	• Indicates whether you are in a production mode or a development mode. Default is false for domains created in development mode and true for domains created in production mode.
`iterativeDevFlag`	`false`	• Checks for updated files, and if found, rebuilds and redeploys the application. Disable this option to prevent checking for changed WebLogic Workshop files. Default is true for domains created in development mode and false for domains created in production mode.

Table 2-1 setDomainEnv Settings (Continued)

`debugFlag`	`false`	• Used in start scripts to set debugging options and indicate if the WebLogic Workshop Debugger should be started. When switched to false, you save the resource overhead used for debugging. • Default is `debugFlag=true` for domains created in development mode; `debugFlag=false` for domains created in production mode.
`testConsoleFlag`	`false`	• Enables the JWS test view. • Verify by checking the log for: `wlw.testConsole = false.` • Default is true for domains created in development mode; false for domains created in production mode.
`logErrorsToConsoleFlag`	`false`	• Controls logging functionality. • Verify by checking the log for: `wlw.logErrorsToConsole = false` • Saves you additional logging. The trade-off is that you may see exceptions more easily when this is set to true (without checking the log). • Default is true for domains created in development mode and false for domains created in production mode.

Table 2-1 setDomainEnv Settings (Continued)

`verboseLoggingFlag`	`false`	• If true, override the default `LOG4J_CONFIG_FILE` (`workshopLogCfg.xml`) with `workshopLogCfgVerbose.xml`. • Priority value in the default file is `warn`; in the verbose version it is `debug`. • Verify by checking the log for: `log4j.configuration = …workshopLogCfg.xml` instead of `workshopLogCfgVerbose.xml` • You can also start in verbose mode using `startWebLogic.cmd verbose`. • Saves you debugging overhead. • Default is `false` for both domains created in development mode and in production mode.
`pointbaseFlag=`	`false`	• Indicates whether Pointbase should be started. • Verify by checking for a running Pointbase process. • Saves you the resource overhead of starting Pointbase when it is not needed. • Default is `true` for domains created with Pointbase as the database.

Removing Debugging Tools from Your Domain

When deploying a domain, you should remove the `debug.properties` file from the domain directory. Although this file is helpful during development, debugging should not be done in production environments.

Tuning for Users and Groups

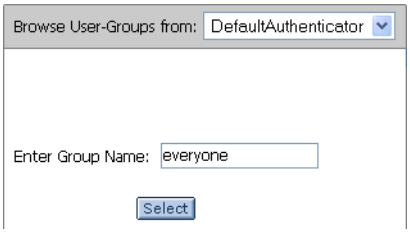
Note: This feature is available for WebLogic Portal 8.1 SP4 and higher.

If you are using an authentication provider and have a large number of users and groups, it can be beneficial to change the way the Portal Administration tools search for names.

If an authentication provider contains a few thousand groups, you may get better performance in the user management interface by not building a group hierarchy tree for the provider. In the place

of a hierarchy tree, you must type the name of a known group in a text box to select that group, as shown in Figure 2-2.

Figure 2-2 Defining a Group Name



Once a group is selected this way, you can add and edit users and set up delegated administration on the group. However, without a group hierarchy tree, you cannot create, delete, or rearrange groups.

Conversely, you can also modify how often the server checks for changes in the hierarchy tree by adjusting the cache sizes. See the online help topic entitled Authentication Hierarchy Service for more information.

Note: In your server startup script(s), you can disable all group hierarchy trees by adding the following to the `JAVA_OPTIONS` line:

```
-Dcom.bea.jsptools.disableGroupTree=true
```

CHAPTER 3

Tuning Your Portal Application

Key aspects of portal performance are managed at the portal application level. These include:

- Managing Caches
- Disabling Unused Services
- Tuning for Campaigns
- Tuning for Entitlements
- Tuning for Content Management

Managing Caches

WebLogic Portal provides a single framework for configuring, accessing, monitoring, and maintaining caches. If configured properly, the caches can vastly reduce the time needed to retrieve frequently used data. Keep in mind that caches are read-only and cluster-aware.

Many WebLogic Portal services use preconfigured caches that you can tune to meet your performance needs. Some services use internally configured caches that you cannot configure or access. If you extend or create additional services, you can use the cache framework to define and use your own set of caches.

Appendix B, "WebLogic Portal Cache Settings" lists caches that might be used by your portal application. Use the list to assist you in your tuning. Keep in mind the memory that is available to your system. When modifying the maximum cache sizes, also monitor the system memory to determine the effects.

Using the Portal Administration Tool to Configure Cache Settings

You can use the Service Administration tools within the Portal Administration tool to configure statically-defined caches. For a list of configurable caches, see Appendix B, "WebLogic Portal Cache Settings."

When you configure a cache, you modify its parameters to change its behavior or capability. Each cache has a Max Size setting and a Time To Live setting. For example, you can set up a cache to hold only the last 10,000 entries, and set the time they can remain in the cache. You can also flush the cache so that all new requests for information come directly from the database.

For instructions on how to configure cache settings, see Configuring Cache Settings.

Caching with JSP Tags

Some WebLogic Portal JSP tags support caching results at various scopes such as session or page. This allows for more control over the caching of individual content queries. Although this can be seen as an advantage, remember that when you control caches with coding, any cache change will require more maintenance, depending on the size (amount of code) of your application.

The following content management-related JSP tags include cache-related attributes:

- `<cm:search>`
- `<cm:getNode>`
- `<pz:contentSelector>`
- `<pz:contentQuery>`

For more information about these JSP tags and their attributes, see Portal JSP Tags in the WebLogic Workshop online help.

Disabling Unused Services

When you create a new portal application, WebLogic Portal enables most services such as commerce services, event listening and campaigns. If your portal application does not require these services, you can improve performance by turning them off.

You can disable services by using the Administration Portal via the Service Administration page; see Adding/Removing Configurable Items in the Administration Portal online help.

The following services should be disabled if your application does not use features that are dependent on them:

- Event services (used for behavior tracking)
- Behavior Tracking services

Tuning for Campaigns

Campaigns are powerful tools for personalization, letting you target users with specific web content, e-mails, and discounts based on fine-grained rules. The following tips allow you to tune your campaign settings to ensure better performance.

Referencing Events

Always make scenario rules dependent on a particular event. This allows optimizations based on the event types referenced in the scenario rules.

Avoiding Firing Extraneous Events

Whenever possible, avoid firing any extraneous events. The campaign services must listen to all events. Use events to signify important occurrences on the site.

Using Goal Checking for Campaigns

If you are using campaigns that take advantage of goal checking, set the goal checking appropriately. Goal checking is used to determine if a campaign's goals are met. When your developers create campaigns, they can set them to end on a specific date or use a set of goals (for example, number of views or clicks). You should set it according to the duration of your campaign. If a campaign's goal check mechanism is set too low, it will affect portal performance. The default is 300000 milliseconds (five minutes).

You can adjust the goal check time for campaigns using the Administration Portal.

For more information about how to adjust this setting, see Configuring a Campaign Service in the Administration Portal online help.

Using Ads During Campaigns

The Campaign service uses display counts to determine whether a campaign has met its end goals. Each time an ad placeholder finds an ad to display as a result of a scenario action, the Campaign service updates the display count.

By default, the Campaign service does not update the display count in the database until an ad placeholder has found 10 ads to display as a result of one or more scenario actions. For performance tuning you can change this default to decrease the database traffic needed to support a campaign.

For sites with high traffic, increase this number to a range of 50 to 100.

To configure the Ad Service cache, use the Administration Portal to perform the following steps:

1. From the Administration Portal, choose **Service Administration**.
2. In the Application Configuration Settings Resource tree, select **Ad Service Group**.
3. Adjust the **Display Flush Size** to a number appropriate for your portal needs. The default is 10.
4. Click **Update**.

Tuning for Entitlements

If your portal uses entitlements, you will need to configure your application to recognize them. You can do this by editing the `netuix-config.xml` file.

The `netuix-config.xml` file resides in the portal web application directory. For example, if you are using the sample portal web application, the corresponding `netuix-config.xml` file is located at:

```
//weblogic81/samples/portal/portalApp/sampleportal/WEB-INF/netuix-config.xml
```

After making any changes, you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see Preparing The EAR File for Deployment in the Production Operations User Guide.

For more information about portal framework performance issues and the `netuix-config.xml` file see http://e-docs.bea.com/wlp/docs81/whitepapers/netix/appendix.html#1052773

1. Edit the `netuix-config.xml` file to include the following text:

```
<entitlements control-resource-cache-size="200">
    <enable>true</enable>
</entitlements>
```

2. If your portal uses a large number of entitlements (more than 5000), review the WebLogic Server documentation, Best Practices: Configure Entitlements Caching When Using WebLogic Providers.

3. After completing the changes, you need to redeploy your portal application.

Using Role Caching When Using Entitlements

Starting with WebLogic Portal 8.1 SP4, role values are cached automatically. However, if you decide to define roles with expressions whose evaluation changes within the course of processing a request, you may need to disable this setting.

To disable role caching, you need to edit the `web.xml` file for the respective application.

Note: After making any changes, you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see Preparing The EAR File for Deployment in the Production Operations User Guide.

1. Navigate to the respective `web.xml` file. It is located in the `WEB-INF` subdirectory of your portal application directory. For example, if you are using the sample portal application, the corresponding `web.xml` file is located at:

`<BEA home>/weblogic81/samples/portal/portalApp/sampleportal/WEB-INF/web.xml.`

2. Open the `web.xml` file in a text editor.
3. Add the following lines

```
<env-entry>
      <env-entry-name>p13n.entitlements.disableRoleCache</env-entry-name>
      <env-entry-value>Y</env-entry-value>
      <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

4. Save the new `web.xml` file.
5. Redeploy your web application.

Tuning for Content Management

When you use a BEA repository for your content management, you can tune the cache settings according to the needs of your portal applications.

Repository cache settings are accessed when you edit a repository.

You can adjust cache settings for nodes (content folders) or binaries (a content item) according to how often your content is accessed and how much content you want to remain in the cache. Keep in mind that your server must have enough memory to handle the cache settings you assign.

Table 3-1 Node Cache

Cache Setting	Usage Notes
Maximum Entries	Determines the maximum number of entries (folders) that can be cached.
Time To Live	Determines how long the entries will be cached.
Enable	Enables the cache. Mark this checkbox to enable this cache. To disable this cache, unmark the checkbox.

Table 3-2 Binary Cache

Cache Setting	Usage Notes
Maximum Entries	Determines the maximum number of entries (content items) that can be cached.
Time To Live	Determines how long the entries will be cached.
Cache Size/Item	Sets the maximum size of a single entry (content item) stored in the cache. The default is 1024 bytes (1K). If your content items average a larger size than this, you should consider changing this cache.
Enable	Enables the cache. Mark this checkbox to enable this cache. To disable this cache, unmark the checkbox.

CHAPTER 4

Tuning Your Portal Web Application

One of the key things you can do to ensure good performance for your web application is to design appropriately, see Designing Portals for Optimal Performance for hints and tips that will increase performance.

This chapter covers a few configuration settings and key areas that can be optimized according to your needs and includes the following sections:

- Optimizing Your Portal Control Tree
- Modifying Your Portal Web Application Parameters
- Tuning Guidelines for WSRP

Optimizing Your Portal Control Tree

Portal web applications use a control tree to cache and access different functionality. For example, portals use controls to access desktops, windows, books, pages, portlets, and menus. With WebLogic Portal 8.1 SP4 and higher, users creating complex portals that require a large number of controls, tree optimization is the easiest way to ensure optimal portal performance. Controls that are not active in the current portal instance are not built, saving considerable time and overhead.

For more information about how control trees work see http://e-docs.bea.com/wlp/docs81/whitepapers/netix/body.html#1056016

For more information about when to optimize your control tree, see Designing Portals for Optimal Performance.

Modifying Your Portal Web Application Parameters

Your portal application uses a configuration files to store application settings. Some default settings may not be applicable to your particular portal application.

Each portal application uses unique configuration files to customize parameters that can affect performance. Three configuration files that are key to portal performance include:

- `netuix.config.xml` (portal framework)
- `web.xml` (web application settings)
- `weblogic.xml` (server settings)

For most settings, you can adjust them using either the WebLogic Server Console or the Administration Portal. However, many of the settings discussed in this section must be manually entered in the configuration file.

Modifying Portal Framework Settings

The `netuix-config.xml` file resides in the portal web application directory. For example, if you are using the sample portal web application, the corresponding `netuix-config.xml` file is located at:

```
//weblogic81/samples/portal/portalApp/sampleportal/WEB-INF/netuix-config.xml
```

After making any changes, you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see Preparing The EAR File for Deployment in the Production Operations User Guide.

For more information about Portal Framework performance issues and the `netuix-config.xml` file see http://e-docs.bea.com/wlp/docs81/whitepapers/netix/appendix.html#1052773.

Table 4-1 lists key performance tuning elements within the `netui-config.xml` fle.

Table 4-1 netui-config.xml

Element	Usage Notes
`<customization>`	A switch to indicate if a portal is customizable or not. If a portal is served from a .portal file (rather than from a database) and users are not allowed to customize it then customization can be disabled by setting `enable` element's value to `false`. If a portal supports customizations then customization should be enabled.

<pageflow>	A switch to enable or disable page flows usage in a portal. Disable it if a portal is not using any page flows.
<validation>	A switch for validating portal related files such as .pinc, .portlet, and .portal files. Disable validation when running portal server in production.
<entitlements>	The <entitlements> element is a switch to indicate that a portal is setup to use entitlement policies (users to portal resources such as desktop, books, pages, portlets, and so on). Disable entitlements if a portal is not using any security policies. If a portal is using security policies, enable it and set the value for <control-resource-cache-size> attribute using number of desktops + number of books + number of pages + number of portlets + number of buttons (max, min, help, edit) used in a portal. The default value could be used if memory is a concern. For more information, see "Tuning for Entitlements" on page 3-5.
<localization>	A switch to indicate that a portal supports multiple locales. This could be disabled if a portal supports only one locale.

Modifying Web Application Settings

The web.xml file configures your web application. After making any changes, you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see Preparing The EAR File for Deployment in the Production Operations User Guide.

The web.xml file is located in the WEB-INF subdirectory of your portal web application directory. For example, if you are using the sample portal web application, the corresponding web.xml file is located at:

<BEA home>/weblogic81/samples/portal/portalApp/sampleportal/WEB-INF/web.xml.

Table 4-2 lists key elements of the web.xml file.

Table 4-2 web.xml

Parameter	Usage Notes
<createAnonymousProfile>	Set this to false if your portal does not store or use user profile information.
<enableTrackedAnonymous>	Set this to false unless you are tracking anonymous users. When this is set to false, only users who login to the portal are tracked.

Table 4-2 web.xml (Continued)

`<fireSessionLoginEvent>`	Set this to false unless using campaigns or behavior tracking. If this is set to true, session login events are generated
`<trackedAnonymousVisitDuration>`	Set to determine the when during a session you start tracking anonymous users. Ignored unless you are tracking anonymous users. This setting allows you to determine when to start tracking anonymous users. The longer you wait during a session to start tracking anonymous users, the greater the performance.
`<skipRequestPattern>`	Set to determine which request patterns to skip. Each page displayed in a web application may have many separate requests, several of which are irrelevant to TAU. For example, the tutorial portal sends requests for images, java script, and CSS files. Ignoring these requests for PortalServletFilter processing increases performance and guarantees that tracking anonymous users will behave as expected.

Modifying WebLogic Server Settings

You can modify the `weblogic.xml` file via the WebLogic Server Console. For more information on how to modify these settings see Viewing and Updating Run-Time Deployment Descriptors in the WebLogic Server Console online help.

The following parameters can be adjusted for performance. For more information about the `weblogic.xml` file, see http://e-docs.bea.com/wls/docs81/webapp/weblogic_xml.html#1037041 for a complete list of the elements configured in the `weblogic.xml` file.

Table 4-3 lists key performance tuning elements in the `weblogic.xml` file.

Table 4-3 weblogic.xml

Parameter	Usage Notes
`<jspPageCheckSeconds>`	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. This default is preset for a development environment. If set to -1, page checking and recompiling is disabled. In a production environment where changes to a JSP are rare, change the value of `pageCheckSeconds` to 60 or greater, according to your tuning requirements, or to -1 to disable page checking and recompiling.
`<PersistentStoreType>`	Must be edited manually. Sets the persistent store method to one of the following options: • `memory`—Disables persistent session storage. • `file`—Uses file-based persistence. • `jdbc`—Uses a database to store persistent sessions. • `replicated`—Same as memory, but session data is replicated across the clustered servers. • `cookie`—All session data is stored in a cookie in the user's browser. • `replicated_if_clustered`—If the web application is deployed on a clustered server, the in-effect PersistentStoreType will be replicated. Otherwise, memory is the default.
`<Timeout Secs>`	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. Minimum value is 1, default is 3600, and maximum value is integer `MAX_VALUE`. 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. This attribute can be overridden by the session-timeout element (defined in minutes) in `web.xml`.

Tuning Guidelines for WSRP

For more information about performance guidelines for Web Services Remote Portlets, see Best Practices for Implementing WSRP.

Enabling Caches for WSRP

If you are using WSRP portlets, adjust your caches accordingly. For specific information about WSRP caches, see "WSRP Caches" on page B-5.

Tuning the Server for WSRP

When using WSRP portlets from a server that is running on a Unix machine, be sure to modify the parameters in the `/etc/system` file to match the minimum requirements, as noted in the WebLogic Server Performance and Tuning Guide.

It is also recommended that you tune the Server Execute Queue using the instructions mentioned Tuning the Default Execute Threads, in the WebLogic Server Performance and Tuning Guide.

Other recommendations include:

- Set the Server Execute Queue to a maximum of 25 threads. Additional threads do not provide higher performance.
- Match the number of JDBC connection pools to the number of threads set in the Server Execute Queue, see Configuring JDBC Connection Pools in the "Creating WebLogic Configurations Using the Configuration Wizard" guide.
- Match the number of threads set in the portalRenderQueue to the threads set for the Server Execute Queue.

Caching Portlet Categories

Portlet category information is automatically cached, which enhances performance. If for any reason you do not want to cache portlet categories, you can turn off this cache by setting the following system property:

```
-enable.portlet.category.caches=false
```

APPENDIX A

Performance Tuning Checklists

Portal Framework Guidelines

Table A-1 Portal Framework Guidelines

Guideline Question	How to Verify
Is the `portalControlTreeCache MaxSize` set to the correct size for your portal?	See "portletControlTreeCache" on page B-2.
Are entitlements enabled? If yes, is `control-resource-cache` size is set correctly?	See "Tuning for Entitlements" on page 3-5.
If you do not need to support multiple locales, is localization disabled?	See "Disabling Unused Services" on page 3-2.
Is `jspPageCheckSecs` in `weblogic.xml` is set to -1?	See "Modifying WebLogic Server Settings" on page 4-4.
Is `servletReloadCheckSecs` in `config.xml` is set to -1?	See "Modifying WebLogic Server Settings" on page 4-4.
Is validation turned off?	See "Modifying Portal Framework Settings" on page 4-2.

Administration Portal Guidelines

You can improve the performance of the Administration Portal by making adjustments. You can decrease the time it takes to create a new desktop and also decrease the amount of time to search for existing desktops using the Administration Portal.

Decreasing the Amount of Time It Takes to Create a Desktop

When you create a new desktop in the Administration Portal, a list of `.portal` files is used to populate the templates drop-down list. If all .portal files reside under the same directory under the web application directory, this drop-down list can be created quickly.

To take advantage of higher performance in building the drop-down list, you must define the `portalFileDirectory` in the web application's `web.xml` file.

Note: After making any changes, you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see Preparing The EAR File for Deployment in the Production Operations User Guide.

1. Navigate to the respective `web.xml` file. It is located in the `WEB-INF` subdirectory of your portal application directory. For example, if you are using the sample portal web application, the corresponding `web.xml` file is located at:

```
<BEA home>/weblogic81/samples/portal/portalApp/sampleportal/WEB-INF/web.xml.
```

2. Open the `web.xml` file in a text editor.
3. Add the following lines

```
<context-param>
   <param-name>portalFileDirectory</param-name>
   <param-value>/</param-value>
</context-param>
```

4. Save the new `web.xml` file.
5. Redeploy your web application.

Increasing Performance When Browsing Portal Resources

When browsing portal resources within the Administration Tools, you can increase the performance by using a system property.

To increase the performance of browsing the Administration Tools, set the following system property at runtime:

```
-Dwlp.enable.portal.tree.perf=true
```

After you set this property, the Administration Portal allows you to browse portal resources more quickly. However, when you add or delete a page or portlet, you need to close and re-open the parent node to reflect the change. This is because the portal resources tree is kept in memory and needs to be refreshed when you add or delete items.

Note: If this property is set to true, searching for desktops behaves differently. When you search for desktops within the Administration Portal you cannot navigate within the Administration Portal until the search is complete.

Content Management Guidelines

Review "Tuning for Content Management" on page 3-7.

WebLogic Portal Cache Settings

This appendix lists the available caches for WebLogic Portal that can be managed within the Portal Administration tool.

Portal Framework Caches

Table B-1 portalContentUriCache

Cache	portalContentUriCache
Use	Used to store portal content URIs for a combination of webapp, portal, locale and optional user name.
Key	Key is equal to portal path + name of web application.

Table B-1 portalContentUriCache (Continued)

Value	Portal content URI
Notes	Set this cache according the number of portals that have associated content URIs. The default values are recommended. Default values: MaxEntries=500; TimeToLive=-1

Table B-2 portalLocalizationLocaleCache

Cache	portalLocalizationLocaleCache
Use	Used to store collection of LocalizationLocale objects. Localization locale specifies language, character encoding, country and variant.
Key	The key is private static final String called portalLocalizationLocaleCachekey.
Value	A set of LocalizationLocale objects.
Notes	Default TTL value should be okay. Max Entries could be set to a number based on the number of rows in the L10N_LOCALE table, i.e. number of supported locales. Default values: MaxEntries=500; TimeToLive=-1

Table B-3 portletControlTreeCache

Cache	portletControlTreeCache
Use	Used to store portlet control trees for floating portlets.
Key	The combination portletInstanceId and locale.

Table B-3 portletControlTreeCache (Continued)

Value	A portlet control tree.
Notes	Default TTL value should be okay, MaxEntries could be set to a number based on number of floatable portlet instances in a portal (including user customized portlets) and number of supported locales. It is recommended that the TTL be left at -1 because the cached default desktop needs to be kept in the cache indefinitely and the cached item for a logged in user is removed when they log out so there is no need to expire a user's cached items. To avoid having the LRU mechanism kick the cached default desktop out of the cache, the MaxEntries should be set to at least (max # of concurrent logged in users + 1) X (# of locales supported). If the cache is too small then LRU will kick out the cached default desktop and the memory saving advantage of this approach will be lost. Default values: MaxEntries=500; TimeToLive=-1

Table B-4 portletPreferencesCache

Cache	portletPreferencesCache
Use	Used to store portlet preferences.
Key	An instance of PortletPreferenceId.
Value	A map of preferences.
Notes	Default TTL and Max Entries values could be set to a value depending on amount of available memory and total number of preferences (at the application level). Defaults: MaxEntries = 500, TimeToLive=60000 (one minute)

Table B-5 portalLocalizationResourceCache

Cache	portalLocalizationResourceCache
Use	Used to store localization resources.
Key	The localizationIntersection.

Table B-5 portalLocalizationResourceCache (Continued)

Value	A LocalizationResource.
Notes	Default TTL and MaxEntries values could be set to a value based on total number of localization resources in the system, which is a combination of non-customized and customized localization resources, and the amount of available memory. Default values: MaxEntries=500; TimeToLive=-1

Table B-6 portalControlTreeCache

Cache	portalControlTreeCache
Use	Used to store portal control trees. Only used for streaming portals.
Key	The combination of webapp, portal, desktop, locale and optional user name.
Value	A portal control tree.
Notes	Default TTL value should be okay. This cache will contain one entry for the default portal, plus one entry for each user who has customized his or her portal. Max Entries could be set to a number based on number of users and available memory. If there are any changes to portal this cache will be flushed. Default values: MaxEntries=500; TimeToLive=-1

Table B-7 portalMarkupDefinitionCache

Cache	portalMarkupDefinitionCache
Use	Used to store MarkupDefinition objects.
Key	A MarkupDefintionID.

Table B-7 portalMarkupDefinitionCache (Continued)

Value	A MarkupDefinition.
Notes	Set this according to the number of rows in the PF_MARKUP_Definition. Markup is the blueprint for a portal library resource (desktop, book, page, portlet, placeholder, menu, Look And Feel, layout, shell or theme). Default values: MaxEntries=500; TimeToLive=60000 (one minute).

WSRP Caches

Table B-8 remoteProducerInfoCache

Cache	remoteProducerInfoCache
Use	Caches the metadata for producers added to a consumer application.
Key	Name of the consumer web application.
Value	A java.util.HashMap containing producer metadata. This map is keyed with the producerHandle of each producer.
Notes	This cache is used to look for producer metadata when a user or administrator is trying to interact with a remote portlet or a producer. Default values: MaxEntries=500; TimeToLive=-1

Table B-9 registrationHandleCache

Cache	registrationHandleCache
Use	Used to store registrationHandles of all registered consumers, for all producers.
Key	The registrationHandle of the consumer.

Table B-9 registrationHandleCache (Continued)

Value	A java.lang.boolean object with a value of true/false.
Notes	This cache is used to cache whether or not a particular registrationHandle is valid. Default values: MaxEntries=500;TimeToLive=-1.

Content and Ad Caches

Table B-10 binaryCache.*repository_name*

Cache	binaryCache.*repository_name*
Use	Used to store binary property values for a repository node.
Key	String (node ID + Property ID)
Value	A byte array associated with the binary property.
Notes	Set this according to the number and size of binary property values. Default values: MaxEntries: 10; TimeToLive:60000 (one minute)

Table B-11 adServiceCache

Cache	adServiceCache
Use	Used to store the results of searches for content rendered in a placeholder (ads). Used by the AdHelper to increase the speed of ad queries.
Key	The ad query (java.lang.String)
Value	A Content []
Notes	Set this according to the number of ad queries and the amount of content expected to be retrieved. Consider basing the maximum size on the total number of ad queries. If the ads returned from a particular query do not change, consider increasing the TTL. Default values: MaxEntries=32; TimeToLive=300000 (five minutes)

Table B-12 nodePathCache.*repository_name*

Cache	nodePathCache.*repository_name*
Use	Used to store a list of nodes for a repository based on a path.
Key	A String (NodeID).
Value	A Node.
Notes	Set according to the number of nodes in a repository. Default values: MaxEntries=50; TimeToLive=60000 (one minute)

Table B-13 searchCache

Cache	searchCache
Use	Used to store an array of IDs for nodes that satisfy a content search.
Key	A Search, which contain parameters for a query.
Value	An ID array of nodes that satisfy a query.
Notes	There is only one search cache used for all repositories. Default values: MaxEntries=20; TimeToLive-=60000 (one minute) Set the MaxEntries according to the amount of content expected to be retrieved. Set Time To Live according to how fresh the content should be. If your content

Table B-14 nodeCache.*repository_name*

Cache	nodeCache.repository_name>
Use	Used to store repository nodes. Each repository has its own cache setting.
Key	A String representing the node ID.

Table B-14 nodeCache.*repository_name* (Continued)

Value	A node.
Notes	Set this according to the number of nodes in a repository. Default values: MaxEntries=50; TimeToLive=6000 (one minute)

User Management Caches

Table B-15 entityIdCache

Cache	entityIdCache
Use	Caches the IDfor an entity (user or group ID)
Key	A com.bea.p13n.property.PropertyLocator. PropertyLocator is based on a user or group name (ENTITY.ENTITY_NAME) and entity type (ENTITY.ENTITY_TYPE).
Value	The entity ID (java.lang.Long).
Notes	Use the ENTITY table as a guide for the maximum size. The object being stored is a Long, which is fairly small. Therefore, it might be possible to set this cache's maximum size to the number of entries in the ENTITY table. Consider how often the ENTITY table might change when setting the TTL. Default values: MaxEntries=500;TimeToLive=600000

Table B-16 jndiNameCache

Cache	jndiNameCache
Use	Stores the JNDI names of entity property managers and UUP managers.
Key	An entity ID.
Value	The home name, which is a string value.
Notes	Set this according the combination of the number of entity property managers and the number of UUP managers. Default values: MaxEntries=500;TimeToLive=600000

Table B-17 entityPropertyCache

Cache	entityPropertyCache
Use	Caches property values for users and groups.
Key	A com.bea.p13n.property.PropertyLocator. PropertyLocator is based on the user or group name (ENTITY.ENTITY_NAME), entity type (ENTITY.ENTITY_TYPE, user or group) and property set type (PROPERTY_KEY.PROPERTY_SET_TYPE, usually USER).
Value	A com.bea.p13n.property.EntityPropertyCache object. This object contains a Map that stores property values keyed off the property set name and property name.
Notes	The larger you can afford to make this cache, the better. Use the ENTITY table as a guide for maximum size. The number of entries in this table should be the maximum number of cache entries that would ever be created. In most cases, there will be more entries here than you would want for a maximum cache size. So consider the average number of users you expect to be using your application at the same time. Consider a TTL based on how often new properties will be added to the property sets. If they are not being modified often, then a higher TTL might be appropriate. Default values: MaxEntries=500;TimeToLive=600000

Table B-18 profileTypeCache

Cache	profileTypeCache
Use	Caches user profile types that are used to look up the appropriate user manager profile manager when retrieving a user profile.
Key	A String (the user name).
Value	A String (the profile type).
Notes	This should be set based on the number of concurrent users. Set the TimeToLive never to expire Default values: MaxEntries=100;TimeToLive=3600000

Table B-19 propertyKeyIdCache

Cache	propertyKeyIdCache
Use	Caches the unique ID associated with a property set type, property set and property name combination (primary key in the PROPERTY_KEY database table).
Key	Based on a property set type, property set, and property name combination (inner class called PropertyKeyLocator).
Value	The ID (java.lang.Long)
Notes	Maximum size should be set with an eye towards the maximum number of properties in the application (use the PROPERTY_KEY table as an indicator). Consider a TTL based on how often these unique ID combinations are likely to change. Default value: MaxEntries=500;TimeToLive=600000

Campaign and Discount Caches

Table B-20 globalDiscountCache

Cache	globalDiscountCache
Use	Stores computed global discount definitions. This is the set of global discounts that is applicable to all users.
Key	The globalDiscountSet name (java.lang.String)
Value	The java.util.Set of qualificationDiscountDef objects.
Notes	Set this to the number of global discounts in your application. The frequency of changes to the global discounts should determine TTL. Default values: MaxEntries=10; TimeToLive-=300000 (five minutes)

Table B-21 discountCache

Cache	discountCache
Use	Used to store computed discount definitions (applicable to individual customers or to customer segments).
Key	A QualificationDiscountId. This is essentially a wrapping around a java.lang.Integer that represents the ID of a discount.
Value	The java.util.Set of qualificationDiscountDef objects
Notes	Set this to the number of discounts in your application. Frequency of changes to the global discounts should determine TTL. Default values: MaxEntries=100; TimeToLive-=300000 (five minutes)

Commerce Caches

Table B-22 globalDiscountAssocCache

Cache	globalDiscountAssocCache
Use	Stores computed global discount associations. This is the set of discount associations that is applicable to all users.
Key	CustomerPk, which is a unique identifier for a customer (java.lang.String).

Table B-22 globalDiscountAssocCache (Continued)

Value	A DiscountAssociation object. A discount association is the mapping of a Customer to a Discount. It is used to track and limit how many times the discount is used by a particular customer.
Notes	Default values: MaxEntries=100; TimeToLive-=3600000 (one hour). Set MaxEntries to the number of global discount associations in your application. The frequency of changes to the global discount associations should determine TTL. To use this cache, you must start the Weblogic server using the command line option: `-Denable.discount.assoc.caches=true` This enables caching of discount associations and global discount associations. The default is `false`, which means a separate read is performed for every pricing calculation for every line item in every shopping cart. Enabling the cache reduces database load by storing associations in a cache after the first read. You can use the Service Administration tools within the Portal Administration Tool to manage this cache. First you must add the cache so that it is visible in the Portal Administration Tool, as described in the online help for the Portal Administration Tool. You can also manage the cache using the p13n cache API.

Table B-23 discountAssocCache

Cache	discountAssocCache
Use	Stores computed discount associations (applicable to individual customers or to customer segments).
Key	CustomerPk, which is a unique identifier for a customer (java.lang.String).

Table B-23 discountAssocCache (Continued)

Value	A DiscountAssociation object. A discount association is the mapping of a Customer to a Discount. It is used to track and limit how many times the discount is used by a particular customer.
Notes	Default values: MaxEntries=100; TimeToLive-=3600000 (one hour). Set MaxEntries to the number of discount associations in your application. The frequency of changes to the discount associations should determine TTL. To use this cache, you must start the Weblogic server using the command line option: `-Denable.discount.assoc.caches=true` This enables caching of discount associations and global discount associations. The default is `false`, which means a separate read is performed for every pricing calculation for every line item in every shopping cart. Enabling the cache reduces database load by storing associations in a cache after the first read. You can use the Service Administration tools within the Portal Administration Tool to manage this cache. First you must add the cache so that it is visible in the Portal Administration Tool, as described in the online help for the Portal Administration Tool. You can also manage the cache using the p13n cache API.

Table B-24 CategoryCache

Cache	categoryCache
Use	Stores the root com.beasys.commerce.ebusiness.catalog.Category, the total number of categories in the product catalog (java.lang.Integer) and the CategoryInfo for each category. CategoryManagerImpl gets the cache name from the ejb-jar.xml in commerce.jar
Key	The key for the root Category is a static final String variable in the CategoryManagerImpl class. The key for the total number of categories is also a static final String variable in the CategoryManagerImpl class. The key for a given CategoryInfo object is a com.beasys.commerce.ebusiness.catalog.CategoryKey.

Table B-24 CategoryCache (Continued)

Value	The value for the root Category is com.beasys.commerce.ebusiness.catalog.Category. The value for the total number of categories is a java.lang.Integer. The value for the category info objects is a com.beasys.commerce.ebusiness.catalog.service.category.CategoryInfo.
Notes	The root Category and the total number of categories occupy two slots in the cache and the remaining slots are occupied by the CategoryInfo objects, so consider the total number of categories in the product catalog plus 2 when setting the maximum cache size. Consider how often these categories will change when setting TTL. Default values: MaxEntries:1000;TimeToLive: 8640000

Table B-25 ProductItemCache

Cache	ProductItemCache (ProductItemManagerImpl gets the cache name from the ejb-jar.xml in commerce.jar.)
Use	Stores the total number of product items in the catalog as well as the product items
Key	The key for the total number of product items is a static final String variable in ProductItemManagerImpl. The key for the product items is a com.beasys.commerce.ebusiness.catalog.ProductItemKey.
Value	The value for the total number of product items is a java.lang.Integer. The value for the product item is a com.beasys.commerce.ebusiness.catalog.ProductItem.
Notes	Consider the total number of product items when setting the maximum cache size. Consider how often these product items will change when setting the TTL. Default values:MaxEntries=1000;TimeToLive=21600000