BEA Campaign Manager for WebLogic
BEA WebLogic Commerce Server
BEA WebLogic Personalization Server

Performance Tuning Guide

BEA Campaign Manager for WebLogic 1.1
BEA WebLogic Commerce Server 3.5
BEA WebLogic Personalization Server 3.5
Document Edition 3.5
April 2001

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Performance Tuning Guide

Document Edition	Date	Software Version
3.5	April 2001	BEA Campaign Manager for WebLogic 1.1 BEA WebLogic Commerce Server 3.5 BEA WebLogic Personalization Server 3.5

Contents

Performance Tuning Guide

When you first install BEA WebLogic Commerce Server™, BEA WebLogic Personalization Server™, and BEA Campaign Manager for Weblogic™, it is configured to support Web-site developers and administrators.

When you are ready to make your Web site available to customers, refer to the following topics in this document for information about tuning WebLogic Commerce Server and Personalization Server performance:

- Precompile JSPs
- Specifying a Different Java Compiler
- Adjust the Intervals for Checking JSP and Servlet Modifications
- Adjust Database Connections Available at Startup
- Adjust Caching
  - Adjust and Use the Session and Global Caches
  - Adjust Caching for Content Management
  - Enable Property Caching
  - Enable Group Caching
  - Adjust Caching for the Discount Service
  - Use CachedProfileBean to Get and Set User Properties from the API
- Increase the Size of the Display-Count Buffer
- Adjust Portal and Portlet Settings While Load Testing
- Display Metadata, Sort and Query Explicit Metadata
- Use LDAP for Authentication Only

- Use the DocumentManager EJB
- Use the HotSpot Virtual Machine

For information on tuning the Behavior Tracking service, refer to "Persisting Tracking Behavior Data" under "Event Properties in the weblogiccommerce.properties File" in the *Guide to Events and Behavior Tracking*.

Precompile JSPs

By default, Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server web applications deactivate the JavaServer Page (JSP) precompile option. With this option deactivated, the server starts quickly but must compile each new or modified JSP when you access it, causing a significant delay the first time you request a new or modified JSP.

When you activate the precompile option, the server startup process checks for new or modified JSPs in the web application and compiles them. Activating the precompile option can cause a significant delay in server startup if you have modified or added JSPs but avoids delays when you access a new or modified JSP for the first time.

To activate the precompile option for a web application that is deployed as an expanded directory hierarchy, do the following:

1. From the web application's `WEB-INF` directory, open the `web.xml` file in a text editor and find the following element:

   ```
   <context-param>
     <param-name>weblogic.jsp.precompile</param-name>
     <param-value>false</param-value>
   </context-param>
   ```

2. In the `<param-value>` element, replace `false` with `true`. For example, `<param-value>true</param-value>`

3. Save the file and restart the server.

   For information on shutting down and starting the server, refer to "Starting and Shutting Down the Server" in the *Deployment Guide*.

To activate the precompile option for a web application that is deployed as a `.war` file do the following:

1. Make a backup copy of the `.war` file.
2. Create a temporary directory and copy the `.war` file to the directory.
3. In the temporary directory, unjar the `.war` file by entering the following command:

   pathname`\jar -xf WarFileName`

   For example:

   ```
   c:\jdk1.3\bin\jar -xf tools.war
   ```

4. Under the temporary directory, open `WEB-INF\web.xml` in a text editor and find the following element:

   ```
   <context-param>
     <param-name>weblogic.jsp.precompile</param-name>
     <param-value>true</param-value>
   </context-param>
   ```

5. In the `<param-value>` element, replace `false` with `true`. For example, `<param-value>true</param-value>`
6. Save `web.xml`.
7. Under the temporary directory, if the `WEB-INF` directory contains a subdirectory named `_tmp_war`, delete the `_tmp_war` directory. This directory contains compiled JSPs and you must remove them before you re-jar the `.war` file to ensure that Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server recompile all JSPs the next time you start the server.
8. Remove the old `.war` file from the temporary directory.
9. Create a new `.war` file for the web application by entering the following command:

   pathname`\jar -cf WarFileName *.*`

   For example:

   ```
   c:\jdk1.3\bin\jar -cf tools.war *.*
   ```

10. Move the new `.war` file back to its original directory.
11. Remove any other files in the original directory that may have been left over from previous `.war` extractions. For example, there may be a `WEB-INF` directory remaining from the last time you ran the web application from the `.war` file.
12. Restart the server.

    For information on shutting down and starting the server, refer to "Starting and Shutting Down the Server" in the *Deployment Guide*.

The server console logs a message for each file it compiles. Ignore any `[JSP Enum] no match` messages. These are displayed for files that do not match the `.jsp` file extension.

Specifying a Different Java Compiler

The WebLogic Server Administration Console specifies a Java compiler for the components in a WebLogic Server domain. You can override this property for a web application by activating the following element in the web application's `weblogic.xml` file:

```
<--
<jsp-param>
      <param-name>compileCommand</param-name>
      <param-value>java-compiler</param-value>
</jsp-param>
-->
```

To activate the element, remove the `<--` and `-->` comment tags.

Then change the `<param-value>` to specify the pathname of the Java compiler that you want to use for the web application.

The `weblogic.xml` file is located in a web application's `WEB-INF` directory. To deploy any modifications to this file, you must restart the server.

Adjust the Intervals for Checking JSP and Servlet Modifications

By default, each time a Web browser requests a JSP, Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server checks for any modifications to the JSP source file. Likewise, each time the server sends a request to a servlet, it checks for any modifications to the servlet class files.

For your production Web site, you can decrease the amount of time in which Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server serves JSPs and processes requests to servlets by increasing the intervals at which the server checks for modifications.

Although the server performs faster with higher values for the modification-check intervals, the higher values reduce sensitivity to changes in your source files. For example, you can set the server to check for JSP modifications every 10 minutes. After you change a JSP, it will take up to 10 minutes for the server to see the modifications.

This section includes the following topics:

- About the Page-Check Intervals Properties
- To Adjust the Intervals

About the Page-Check Intervals Properties

The `pageCheckSeconds` property determines the interval at which WebLogic Server checks to see if JSP files in a web application have changed and need recompiling. Each web application defines this property separately in its `WEB-INF\weblogic.xml` file. For example, the e-commerce sample web application defines this property in the following file:
`WL_COMMERCE_HOME\config\wlcsDomain\applications\wlcsApp\wlcs\WEB-INF\weblogic.xml`

The following excerpt from the e-commerce sample web application shows the `weblogic.jsp.pageCheckSeconds` context parameter in boldface text with the default value:

```
<jsp-param>
     <param-name>weblogic.jsp.pageCheckSeconds</param-name>
     <param-value>300</param-value>
</jsp-param>
```

Note: The page-check interval does not determine the frequency with which Commerce Server checks for updated content that is stored in the database and in a content management system. Instead, the `ttl` (time-to-live) settings for various caches determine the refresh rate for content. For example, if you set the page-check intervals to once a second, and you set the `ttl` for the content cache to 10 minutes, it can take up to 10 minutes for the server to see the new content, even though it is checking for new JSP source code every second. For information on setting `ttl` properties for caches, refer to “Adjust Caching” on page 10.

To Adjust the Intervals

To determine the optimal page-check and reload-servlet intervals for your production Web site do the following:

1. Establish performance baselines by testing Commerce Server performance with the interval set to `-1` (which specifies that the server never checks for modifications).
2. Test the performance with the interval set to various numbers of seconds. For example, set the interval to `600` seconds (10 minutes) and test the performance. Then set the interval to `900` seconds and test the performance.
3. Choose an interval that provides the best performance while checking for modifications to JSP files and servlet classes at a satisfactory rate.

Adjust Database Connections Available at Startup

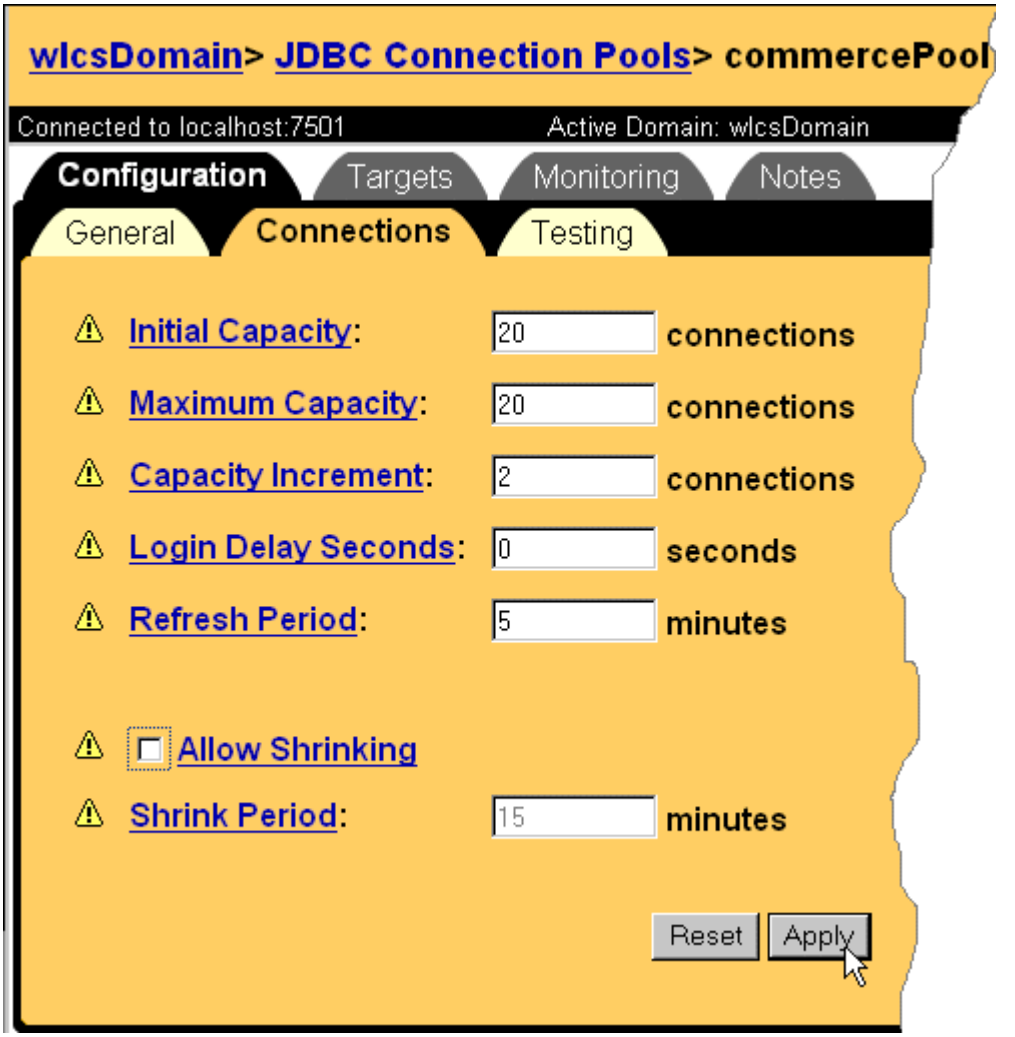
To optimize the database pool performance for your production Web site, do the following:

1. Start the WebLogic Server Administration Console for your domain.

   For information on starting the Administration Console, refer to "Viewing and Modifying Properties in the WebLogic Server Administration Console" under "The Server Configuration" in the *Deployment Guide*.
2. On the home page, under JDBC, click Connection Pools.
3. On the JDBC Connection Pools page, in the Name column, click commercePool.

4. Do the following:
    a. Increase the value in Initial Capacity to match the value in Maximum Capacity
    b. Change Login Delay Seconds to `0`
    c. Clear the Allow Shrinking check box
    d. Click Apply (see Figure 1)
5. Click the Testing tab and clear the Test Reserve Connections check box.
6. Click Apply.
7. Restart the server.

Figure 1 Change Values on the commercePool Tab



For More Information

For more information on database connection pools, refer to the Administration Console online help and to "Setting Up Connection Pools" under "Creating and Managing Content" in the *Guide to Building Personalized Applications*.

Adjust Caching

To adjust caching for production Web site, complete the following tasks:

- Adjust and Use the Session and Global Caches
- Adjust Caching for Content Management
- Enable Property Caching
- Enable Group Caching
- Use CachedProfileBean to Get and Set User Properties from the API

Adjust and Use the Session and Global Caches

In a clustered environment, you can improve scalability and performance by minimizing the use of `HttpSession` objects. (`HttpSession` is part of the JDK session-tracking mechanism, which servlets use to maintain state about a series of requests from the same user.)

To minimize using `HttpSession`, each server in the WebLogic Commerce Server and Personalization Server cluster provides the following caches:

- `session` cache, which stores data in memory about each session. The function of the session cache is the same as `HttpSession`, however, unlike `HttpSession`, it is not replicated across the cluster.
- `global` cache, which stores data in memory that multiple sessions can use. For example, sessions for anonymous users can access data from the global cache. Like the session cache, it is not replicated across the cluster.

This section discusses the following topics:

- Enabling the Caches
- JSP Tags for Accessing HttpSession and the Session and Global Caches
- An API for Accessing HttpSession and the Session and Global Caches
- Guidelines for Placing Data in HttpSession, Session Cache, or Global Cache

For More Information

For more information about how WebLogic Commerce Server and Personalization Server process HTTP requests, refer to "Foundation Classes and Utilities" in the *Guide to Building Personalized Applications*. For more information about `HttpSession`, see http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/http/HttpSession.html.

Enabling the Caches

To enable the session and global caches, add the following properties to `$WL_COMMERCE_HOME/weblogiccommerce.properties`:

```
_sessionCache.ttl=900000
_sessionCache.capacity=10000
_sessionCache.enabled=true


_globalCache.ttl=600000
_globalCache.capacity=1000
_globalCache.enabled=true
```

The `ttl` (time-to-live) property determines the number of milliseconds that the server maintains the cache. The `capacity` property determines the maximum number of objects in the cache. (Both `session` and `global` are in-memory caches.) The `enabled` property determines whether the cache is activated. A `false` value deactivates the cache and obviates the `ttl` and `capacity` properties; `true` activates it.

You can increase or decrease values for `ttl` and `capacity` based on the amount of available memory and the level of performance you desire.

Note: Each server in a cluster maintains its own set of caches, each of which must be configured separately by modifying the server's `weblogiccommerce.properties` file. Because the session and global caches are not replicated across servers in the cluster, if a server fails, the data in its caches is inaccessible. For guidelines about which types of data to place in the session and global caches, see "Guidelines for Placing Data in HttpSession, Session Cache, or Global Cache" on page 13.

JSP Tags for Accessing HttpSession and the Session and Global Caches

Use the following JSP tags from the FlowManager tag library to place, retrieve, and remove data from `HttpSession` as well as the session and global caches:

- `<fm:getCachedAttribute>`
- `<fm:setCachedAttribute>`
- `<fm:removeCachedAttribute>`
- `<fm:getSessionAttribute>`
- `<fm:setSessionAttribute>`
- `<fm:removeSessionAttribute>`

For information about these tags, refer to "JSP Tag Library Reference" in the *Guide to Building Personalized Applications*.

An API for Accessing HttpSession and the Session and Global Caches

Use the following methods of the `com.beasys.commerce.foundation.flow.helper.FlowManagerHelper` API to place, retrieve, and remove data from `HttpSession` and the session and global caches:

- getCachedValue
- setCachedValue
- removeCachedValue
- getGlobalCachedValue
- setGlobalCachedValue
- removeGlobalCachedValue
- getSessionAttribute
- setSessionAttribute
- removeSessionAttribute

For information about these methods, refer to the documentation for `com.beasys.commerce.foundation.flow.helper.FlowManagerHelper` in the WebLogic Personalization Server Javadoc.

Guidelines for Placing Data in HttpSession, Session Cache, or Global Cache

In general, place only the following in `HttpSession`:

- Items that are required for replication across the cluster.
- Any keys that are required to look up information. When you enable session replication for WebLogic Server, `HttpSession` is replicated on all machines in a cluster. Placing information in `HttpSession` while session replication is enabled provides a backup for data lookups. For example, you place query parameters for a search in `HttpSession` and the search results in the `session` cache. While returning the search results the server fails. Another server can recreate the search by referring to the parameters that are stored in the `HttpSession` replica.

Place any information that multiple users require (either within the same application or across multiple applications) in the global cache.

Place all other session-related information in the session cache.

Adjust Caching for Content Management

To optimize content-management performance for your production Web site, configure WebLogic Personalization Server as follows:

- For the `cm:select`, `cm:selectById`, `pz:contentQuery`, and `pz:contentSelector` JSP tags, use the `useCache` attribute whenever possible. Doing so avoids a call to DocumentManager and, in the case of `pz:ContentSelector`, to the Rules Manager.

  For information on using the `useCache` attribute, refer to "JSP Tag Library Reference" in the *Guide to Building Personalized Applications.*

  To clear cached content when user and/or document attributes change, use the `remove` method of `com.beasys.commerce.content.ContentCache`. For more information, see the Javadoc for `com.beasys.commerce.content.ContentCache`.

  For an example of a JSP file that uses the `remove` method, see

  `WL_COMMERCE_HOME/server/public_html/examples/content/cache-control.jsp`

- For the `cm:select`, `cm:selectById`, `pz:contentQuery`, and `pz:contentSelector` JSP tags, set the `cacheScope` attribute to `application` whenever possible. For example:

  ```
  <cm:select id="myDocs" query="riskFactor = 'Low'"
  useCache="true" cacheId="myDocs"
  cacheScope="application"
  max="10" cacheTimeout="300000" />
  ```

  The application cache type is global instead of per-user and should speed up queries by avoiding a call to the DocumentManager EJB.

Note: For `pz:contentSelector`, set the `cacheScope` attribute to **`application`** only when you want to select **shared** content. For example, in exampleportal, the Acme Promotion portlet uses an application-scoped cache to select content for non-authenticated users. Because it uses the application scope, all non-authenticated users see the same content. For authenticated users, Acme Promotion provides personalized content by switching to a session scoped cache.

- Whenever you can predict the next document that users will view based on the document that they are currently viewing, load the next document into the cache before users request it. This “forward caching” will greatly improve the speed at which Personalization Server responds to user requests (assuming that your prediction is correct; forward caching a document that no one requests will only degrade performance and scalability).

  The following JSP fragment is an example of forward caching a document:

```
<%-- Get the first set of content --%>

<cm:select id="myDocs" query="riskFactor = 'Low'"
useCache="true" cacheId="myDocs"
cacheScope="application"
max="10" cacheTimeout="300000" />

<%-- Generate a query from each content's relatedDocId --%>

<% String query = null; %>
<es:forEachInArray array="<%=myDocs%>" id="myDoc"
type="com.beasys.commerce.axiom.content.Content">

<% String relId = (String)myDoc.getProperty("relatedDocId",
null); %>
<es:notNull item="<%=relId%>">

<%
if (query != null)
query += " || ";
else
query = "";
query += "identifier = '" +
ExpressionHelper.toStringLiteral(relId) + "'";

%>

</es:notNull>
</es:forEachInArray>

<%-- Load the related content into the cache via cm:select
--%>

<es:notNull item="<%=query%>">

<cm:select query="<%=query%>" id="foo" useCache="true"
cacheId="relatedDocs"
cacheScope="session" max="10" cacheTimeout="300000" />

</es:notNull>
```

For More Information

For more information about content management, see "Creating and Managing Content" in the *Guide to Building Personalized Applications.*

For more information about JSP tags for content management, see "JSP Tag Library Reference" in the *Guide to Building Personalized Applications.*

Enable Property Caching

The WebLogic Server Configurable Entity and Entity Property Manager provide several in-memory caches that you can enable for WebLogic Commerce Server and Personalization Server. The caches decrease the amount of time needed to access user, group, and other properties, but introduce the possibility of stale data.

This section discusses the following topics:

- Property Caching in a Clustered Environment
- To Enable Property Caching

Property Caching in a Clustered Environment

With property caching enabled in a clustered environment, each server in a cluster maintains its own cache; the cache is not replicated on other servers. In this environment, when properties that are stored in the `defaultPropertyCache`, `entityPropertyCache`, `directPropertyManager`, or `ldapPropertyCache` change on one server, they may not change on another server in a timely fashion.

In most cases, immediate or quick access to properties on another server is not necessary: user sessions are pinned to a single server, and even with caching enabled, users immediately see changes they make to their own settings on the server.

However, if a server fails and loses the data in its caches, modifications to properties may be lost, depending on the longevity of the property cache. In addition, if an administrator changes a user's properties, the user may not see the changes during the current session if the user and the administrator are pinned to different servers in the cluster.

You can mitigate these situations by specifying a small `ttl` (time-to-live) setting when you enable the caches. The small `ttl` setting provides performance gains by caching data, but the short-lived caches increase the rate at which property changes are replicated across servers.

If you require multiple servers in a cluster to have immediate access to modified properties, disable property caching by adding the entries described in "To Enable Property Caching" and specifying `false` for the `unifiedProfileTypeCache.enabled` value.

To Enable Property Caching

To enable property caching, add the following entries to `WL_COMMERCE_HOME\weblogiccommerce.properties`, adjusting the values based on the number of properties in your property sets and the frequency with which you want the data updated:

Note: These entries enable in-memory caching. Caches that grow exceedingly large may degrade performance.

- To create a cache of unified profile types that lives for 1 hour and contains 100 entries, add:

```
unifiedProfileTypeCache.ttl=3600000
unifiedProfileTypeCache.capacity=100
unifiedProfileTypeCache.enabled=true
```

- To create a cache of default schema properties that lives for 10 minutes and contains 500 entries, add:

```
defaultPropertyCache.ttl=600000
defaultPropertyCache.capacity=500
defaultPropertyCache.enabled=true
```

- To create a cache of entity properties that lives for 10 minutes and contains 500 entries, add:

```
entityPropertyCache.ttl=600000
entityPropertyCache.capacity=500
entityPropertyCache.enabled=true
```

- To create a cache of LDAP entity properties that lives for 10 minutes and contains 500 entries, add:

```
ldapEntityPropertyCache.ttl=600000
ldapEntityPropertyCache.capacity=500
ldapEntityPropertyCache.enabled=true
```

- To create a cache of entity ids that lives for 1 hour and contains 500 entries, add:

```
entityIdCache.ttl=3600000
entityIdCache.capacity=500
entityIdCache.enabled=true
```

- To create a cache of explicit properties that lives for 10 minutes and contains 100 entries, add:

```
directPropertyManager.ttl=600000
directPropertyManager.capacity=100
directPropertyManager.enabled=true
```

- To create a cache of ConfigurableEntity methods that lives for 1 hour and contains 100 entries, add:

```
ConfigurableEntityMethodCache.ttl=3600000
ConfigurableEntityMethodCache.capacity=100
ConfigurableEntityMethodCache.enabled=true
```

For More Information

For more information about property sets, see "Creating and Managing Property Sets" in the *Guide to Building Personalized Applications*.

For more information about JSP tags for managing property sets, see "JSP Tag Library Reference" in the *Guide to Building Personalized Applications.*

Enable Group Caching

In systems with a deep group hierarchies, you can improve performance using group caching, which precalculates group membership information and stores the calculation results in a new database table, WLCS_USER_GROUP_CACHE. Any queries that are submitted while group caching is recalculating data return the old, previously committed data.

With group caching, you exchange faster performance for the risk of stale or inconsistent data. To balance performance with data consistency, you can configure the interval at which the caching mechanism recalculates and updates the table.

This section contains the following topics:

- Group Caching in a Clustered Environment
- To Set Up the Group-Cache Table
- To Enable and Configure the Group Cache
- To Access Data in the Group Cache Table

Group Caching in a Clustered Environment

To improve performance of group caching in a cluster, you can establish one cache as the master. The server with the master cache periodically updates its WLCS_USER_GROUP_CACHE table. All other servers in the cluster read this master table; they do not update the table or maintain their own copy. For information on setting up a master cache, refer to “To Enable and Configure the Group Cache” on page 20.

To Set Up the Group-Cache Table

To set up the table for group caching, issue the following SQL commands:

```
CREATE TABLE WLCS_USER_GROUP_CACHE ( USER_NAME VARCHAR2(100) NOT NULL,
GROUP_NAME VARCHAR2(100) NOT NULL );
ALTER TABLE WLCS_USER_GROUP_CACHE
ADD CONSTRAINT WLCS_USER_GROUP_CACHE_INDEX PRIMARY KEY ( USER_NAME,
GROUP_NAME );
```

To Enable and Configure the Group Cache

To enable the group cache, do the following:

1. Start the WebLogic Server Administration Console for your domain.

   For information on starting the Administration Console, refer to "Viewing and Modifying Properties in the WebLogic Server Administration Console" under "The Server Configuration" in the *Deployment Guide*.

2. In the left pane of the Administration Console, click wlcsDomain → Deployments → Startup & Shutdown.

   When you click Startup & Shutdown, the Administration Console displays startup and shutdown properties in the right pane. (See Figure 2.)

Figure 2 Startup & Shutdown in the Left Pane

3. In the right pane, click Create a New Startup Class.
4. On the wlcsDomain → Startup Classes → create page click the Configuration tab.
5. On the Configuration tab, enter the following:
   - In the Name box, enter GroupCache

     In a clustered environment, enter MasterGroupCache or some other similar name.
   - In the ClassName box, enter com.beasys.commerce.axiom.contact.security.GroupCache
   - In the Arguments box, enter updateDb=true
6. Click Create. (See Figure 3.)

   The Administration Console creates the group cache.

Figure 3 Create the Group Cache

7. To deploy the group cache, click the Targets tab.

8. On the Servers subtab, select wlcsServer from the Available Servers list. Then move it to the Chosen Servers list.

   In a clustered environment, deploy the MasterGroupCache startup class only on the server that you want to contain the master cache.

9. Click Apply.

10. In a clustered environment, you can configure the remaining servers to use the master cache that you deployed in step 9. For more information, refer to “Configure Servers to Use the Master Cache” on page 22.

11. To configure the number of seconds that the server waits before calculating and updating the table, do the following:

    a. From the left pane of the Administration Console, click wlcsDomain → Security → CachingRealms → wlcsCachingRealm.

    b. In the right pane, on the wlcsDomain → Security → Caching Realms → wlcsCachingRealm page, click the Groups subtab.

    c. Modify the value in the Group Cache TTL Positive box.

12. Restart the server.

    For information on shutting down and starting the server, refer to “Starting and Shutting Down the Server" in the *Deployment Guide*.

Note: You do not need to specify the size of the group cache. The depth of the group hierarchies determines the size of the group cache table.

Configure Servers to Use the Master Cache

After you create a master group cache as described in “To Enable and Configure the Group Cache” on page 20, configure other servers to use the master cache by doing the following:

1. In the left pane of the Administration Console, click Startup & Shutdown.

2. On the wlcsDomain → Startup & Shutdown page, click Create a New Startup Class.

3. On the wlcsDomain → Startup Classes → Create page click the Configuration tab.

4. On the Configuration tab, enter the following:
   - In the Name box, enter GroupCache
   - In the ClassName box, enter com.beasys.commerce.axiom.contact.security.GroupCache
   - In the Arguments box, enter updateDb=false
5. Click Create.
6. Click the Targets tab.
7. On the Servers subtab, from the Available Servers list, select the remaining servers in the cluster. (That is, select all servers except the server on which you deployed the MasterGroupCache startup class.) Then move them to the Chosen Servers list.
8. Click Apply.

To Access Data in the Group Cache Table

To access data in the group cache table, use any of the following:

- The new `UserManager` method of `getCachedGroupNamesForUser`
- The `static` method of the `GroupCache` object

For more information about these methods, refer to *WebLogic Personalization Server Javadoc*.

Adjust Caching for the Discount Service

To reduce the amount of time the Order and Shopping Cart services need to calculate order and price information that include discounts, the Discount Service caches information about discounts that are available to customers.

When a customer adds an item to the shopping cart, removes an item from the shopping cart, checks out, or confirms an order, the Pricing Service is responsible for determining the price of the items in the cart.

To calculate the effect of discounts on the shopping cart, the Pricing Service requests the Discount Service to retrieve information about all global discounts and about any campaign discounts that apply to the current customer.

Global discounts apply to all customers, regardless of customer properties or customer segments. Campaign discounts are targeted to specific customers or customer segments, and are available only if you use BEA Campaign Manager for WebLogic.

The first request for information about discounts requires a separate call to the database for each discount that applies. For example, if you have defined one global discount and if a customer is eligible for two campaign-related discounts, the Discount Service makes three calls to the database.

To decrease the response time for any subsequent requests, the Discount Service places the information about each discount in one of the following caches:

- A cache for campaign discounts
- A cache for global discounts

You can use the properties in Listing 1 to adjust the default cache settings.

This section contains the following subsections:

- Adjusting the Campaign-Discount Cache
- Adjusting the Global-Discount Cache
- How the Discount-Service Cache Behaves in a Clustered Environment

Adjusting the Campaign-Discount Cache

The `discountCache.capacity` (illustrated in Listing 1) property determines how many campaign discounts the Discount Service caches. For maximum performance, set the capacity to the number of campaign discounts that are currently deployed. A larger capacity will potentially use more memory than a smaller capacity.

The `discountCache.ttl` (time-to-live property) determines the number of milliseconds that the Discount Service keeps the information in the cache. After the cache value times out, the next request for the value requires the Discount Service to call the database to retrieve the information and then cache the value. A longer TTL decreases the number of database calls made over time when requesting cached objects. In a clustered environment, the TTL is the maximum time required to guarantee that any changes to global discounts are available on all servers.

The `discountCache.enabled=true` property enables the cache. To disable the cache, set the value to `false`.

Adjusting the Global-Discount Cache

If you enable the global-discount cache, the Discount Service caches all global discounts. The `globalDiscountCache.capacity` property does not need to be modified.

The `globalDiscountCache.ttl` (time-to-live property) determines the number of milliseconds that the Discount Service keeps information in the global-discount cache. After the time-to-live (TTL) expires, the next request for global discount information requires the Discount Service to call the database to retrieve the information and then cache the value. A longer TTL decreases the number of database calls made over time when requesting cached objects. In a clustered environment, the TTL is the maximum time required to guarantee that any changes to campaign discounts are available on all servers.

Listing 1 Discount Service Cache Settings

```
# Discount Service cache entries
# cache of all discounts, 5 minutes, 100 entries

discountCache.ttl=300000
discountCache.capacity=100
discountCache.enabled=true

# the global discount cache, 5 minutes, 10 entries

globalDiscountCache.ttl=300000
globalDiscountCache.capacity=10
globalDiscountCache.enabled=true
```

How the Discount-Service Cache Behaves in a Clustered Environment

In either environment (clustered or non-clustered), when you change a discount priority, end date, or its active/inactive state, WebLogic Commerce Server flushes the discount from the appropriate cache. Changes to a campaign discount flush only the specific discount from the campaign-discount cache. Changes to a global discount flush all discounts from the global-discount cache.

For example, you log in to a WebLogic Commerce Server host named `bread` and deactivate a campaign discount named `CampaignDiscount1`. WebLogic Commerce Server flushes the `CampaignDiscount1` from the campaign-discount cache on `bread`.

In a clustered environment, other machines in the cluster continue to use their cached copy of the discount until the TTL for that discount expires.

Use CachedProfileBean to Get and Set User Properties from the API

To get and set user properties in a servlet via the API, use CachedProfileBean instead of entity beans like User, Group, and UnifiedUser.

For getting user properties, the first invocation of `com.beasys.commerce.user.jsp.beans.CachedProfileBean.getProperty()` is significantly faster than invoking `com.beasys.commerce.axiom.contact.User.getProperty()`. Subsequent invocations of `CachedProfileBean.getProperty()` are orders of magnitude faster.

For setting user properties, `CachedProfileBean.setProperty()` is slower than `User.setProperty()` but it reduces the possibility of deadlocks when setting properties in highly shared entity beans.

`CachedProfileBean` minimizes contact with the User entity bean by using a session cache. When it does need to contact the User entity bean to update a property, it must first get a reference, then update the property. For this reason, `CachedProfileBean.setProperty()` can be slower than `User.setProperty()` for setting property values. However, using `CachedProfileBean` reduces the risk of deadlocks, especially with highly shared entity beans like Group. As the number of references to an entity bean grows, the risk of a deadlock increases; since `CachedProfileBean` doesn't store a reference to the entity bean, the risk of deadlock is lowered.

If you have a large number of `set` statements in a row, you can improve performance by wrapping the statements in a transaction, such as `javax.transaction.UserTransaction`. In addition to reducing database connections (thereby improving performance), wrapping the statements in a transaction causes all of the `set` statements to succeed or fail as a unit.

Do not set properties with `User.setProperty()` and get them with `CachedProfileBean.getProperty()`. `User.setProperty()` updates the database but not the cache. `CachedProfileBean.getProperty()` retrieves data from the cache. Depending on the cache's last update, it might not contain properties that `User.setProperty()` placed in the database.

Other notes on `CachedProfileBean`:

- We recommend that you instantiate a separate `CachedProfileBean` for each user whose profile is to be manipulated.
- To use `CachedProfileBean` with a Unified User, use the default no-argument constructor, then set the profile key and profile type. The profile type should match the Unified Profile Type you defined using the Administration Tool web application. It is important to set the profile key before setting anything else, because this call causes all other state to be cleared. For example:

  ```
  CachedProfileBean user = new CachedProfileBean();
  user.setProfileKey("unifieduser_bob");
  user.setProfileType("Unified Profile Example");
  ```

  ```
  String points = user.getPropertyAsString(null, "userPoints",
  true);
  ```

  ```
  String favChar = user.getPropertyAsString("exampleportal",
  "FavoriteCharacter", true);
  ```

- To use the `CachedProfileBean` in a clustered environment, maintain an instance of `CachedProfileBean` in each user's session. For example, to set a user property, do the following:

  a. Retrieve the `CachedProfileBean` from the user's `HttpSession`

  b. Use `setProperty()`

  c. Return the `CachedProfileBean` to the session

  When you get a property, use the existing `CachedProfileBean` from the user's session instead of instantiating a new one.

- If you use the `CachedProfileBean` on a `t3` client or in a servlet that is deployed on a server other than Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server, you must set the Java server's `commerce.properties` environment property to `weblogiccommerce.properties`. The `CachedProfileBean` uses `weblogiccommerce.properties` for JNDI lookups. You can set this property on the command line when you start your `t3` client (or server) as follows:
  `java -Dcommerce.properties=weblogiccommerce.properties` *`MyT3Client`*

Increase the Size of the Display-Count Buffer

If you use campaigns (available only with BEA Campaign Manager for WebLogic), the `adservicebean.display.flush.size` property in `weblogiccommerce.properties` determines the number of display counts that are stored in memory before updating the database. (See Listing 2.)

The Campaign Service uses display counts to determine whether a campaign has met its end goals. Each time an ad placeholder displays an ad as a result of a scenario action, the Campaign Service updates the display count. With the default setting of 10, the Campaign Service does not update the display count in the database until 10 ads have displayed as a result of one or more scenario actions placing queries in ad placeholders.

If the server crashes before the Campaign Server flushes this display-count buffer to the database, you can lose display-count updates, up to the value of the `adservicebean.display.flush.size` property.

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

Listing 2 Display-Count Property in weblogiccommerce.properties

```
##############################################

# AdTargetTag Properties
adtargettag.rendering=com.bea.commerce.platform.ad.AdClickThruServlet

...

# This value determines how many display update counts are cached before
# we flush them to the database

adservicebean.display.flush.size=10
```

Adjust Portal and Portlet Settings While Load Testing

If you are testing the performance of the portal framework, do the following:

- Enable session and global caches as described in "Adjust and Use the Session and Global Caches" on page 10. (You do not need to add JSP tags or API methods that access the caches when testing the portal framework; the framework includes them by default.)

- Because slow portlets can severely slow a portal's performance, remove all of the portlets from the portal except for Dictionary, Search, and Quote. These portlets do not invoke external activities such as database connections.
- Modify the framework's Application Initialization Property Set as follows:
  - For `refreshWorkingDir`, increase the default number of seconds to prevent Personalization Sever from refreshing the working directory every five minutes (300 seconds) during a long load test.

    The working directory is the root of the portal pages and WebLogic Personalization Server pages hierarchy. You define the working directory in a JSP, and you can change it as needed without restarting the server. The `refreshWorkingDir` property determines how frequently the server checks to see if you have changed the working directory.

    The Application Initialization Property Set for the exampleportal defines the `refreshWorkingDir` property. If you base your portal on the exampleportal, it too will define the `refreshWorkingDir` property.
  - For `ttl`, increase the default number of milliseconds to prevent Personalization Sever from reloading properties every five minutes (300000 milliseconds) during a long load test.

For More Information

For more information, see the *Guide to Creating Portals and Portlets*.

Display Metadata, Sort and Query Explicit Metadata

If you used the BulkLoader to load document metadata into the reference implementation document database, you can improve document management performance when retrieving documents by doing the following:

- Display a document's metadata instead of the full document.
- Sort on explicit (system-defined) metadata attributes instead of implicit (user-defined) metadata attributes.
- Query on explicit metadata attributes instead of implicit metadata attributes.

For More Information

For more information about content management, see "Creating and Managing Content" in the *Guide to Building Personalized Applications*.

Use LDAP for Authentication Only

For improved performance, use LDAP for authentication only; do not use it to retrieve user and group properties. Instead of retrieving properties from LDAP servers, configure your system to use properties stored in the RDBMS by minimizing the number of properties registered for retrieval from LDAP in the user management tools.

For More Information

For more information about changing LDAP settings, see "Using Other Realms" under "Creating and Managing Users" in the *Guide to Building Personalized Applications*.

Use the DocumentManager EJB

Always use a DocumentManager EJB instead of a Document EJB. Document EJBs are deprecated.

Use the HotSpot Virtual Machine

Hot Spot enhances JDK 1.3 performance by using a just-in-time compiler (JIT) and other features. It provides two implementations: a client VM and a server VM.

On **Windows**, Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server support the **client VM**; they do not support the server VM.

On **UNIX**, Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server support the **server VM**; they do not support the client VM.

The default `StartCommerce` startup script activates the HotSpot VM that is appropriate for each platform type.

Deactivating HotSpot for Debugging

Because HotSpot uses a JIT, you cannot access thread dumps while it is active. If you require thread dumps while developing and debugging your application, do the following:

1. Open the `$WL_COMMERCE_HOME/StartCommerce` script and remove the value from the `JAVA_VM` variable.

   If you use your own startup script, on Windows, remove the `-hotspot` parameter from the Java command that starts the server. On UNIX, remove the `-server` parameter.

2. Restart the server.

   For information on shutting down and starting the server, refer to "Starting and Shutting Down the Server" in the *Deployment Guide*.