BEA WebLogic Portal

Designing Portals for Optimal Performance

BEA WebLogic Portal Version 8.1 Service Pack 6
Revised: July 2006

Copyright

Copyright © 2004-2006 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

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

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

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

Trademarks or Service Marks

BEA, BEA WebLogic Server, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic JRockit, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server Process Edition, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

About This Document

Product Documentation on the dev2dev Web Site. v
Contact Us v
Documentation Conventions vi

Designing Portals for Optimal Performance

Control Tree Design. 2
How the Control Tree Works 2
How the Control Tree Affects Performance 3
Using Multiple Desktops 4
Why this is a Good Idea 4
Design Decisions for Using Multiple Desktops 6
Optimizing the Control Tree 7
Enabling Control Tree Optimization. 7
Setting the Current Page 9
How Tree Optimization Works 10
Limitations to Using Tree Optimization 11
Disabling Tree Optimization. 13
Other Ways to Improve Performance 14
Use Entitlements Judiciously 14
How Entitlements Affect Performance 15
Recommendations for Using Entitlements 15

Limit User Customizations 16
Optimize Page Flow Session Footprint 16
Use File-based Portals for Simple Applications 17
Why Use a File-based Portal? 17
Limitations to Using File-based Portals 18
Create a Production Domain in Development 18
Use Remote Portlets 19
Do Remote Portlets Really Provide a Performance Boost? 19
Customize Portlets to Take Advantage of Optimization Features 20
Use Backing Files 22
Use Pagination When Displaying Large Amounts of Data in a Portlet 22
Thoughtful Design Decisions Ensure Optimal Performance 23

About This Document

This document investigates some of the reasons for inadequate portal performance and offers suggestions and best practices for avoiding those problems. It explores:

- How portal taxonomy affects performance and describes some of the latest features included in BEA WebLogic Portal 8.1 that leverage this taxonomy to improve performance.
- Other performance-enhancing portal concepts, such as the use of entitlements, streaming portals, remote portals, and optimizing page flow sessions.

Product Documentation on the dev2dev Web Site

BEA product documentation, along with other information about BEA software, is available from the BEA dev2dev Web site:

`http://dev2dev.bea.com`

To view the documentation for a particular product, select that product from the list on the dev2dev page; the home page for the specified product is displayed. From the menu on the left side of the screen, select Documentation for the appropriate release. The home page for the complete documentation set for the product and release you have selected is displayed.

Contact Us

Your feedback on the BEA BEA WebLogic Portal 8.1 documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be

reviewed directly by the BEA professionals who create and update the BEA WebLogic Portal 8.1 documentation.

In your e-mail message, please indicate that you are using the documentation for BEA WebLogic Portal 8.1, and include the product version.

If you have any questions about this version of BEA WebLogic Portal 8.1, or if you have problems installing and running BEA WebLogic Portal 8.1, contact BEA Customer Support at **http://support.bea.com**. You can also contact Customer Support by using the contact information provided on the quick reference sheet titled “BEA Customer Support,” which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

<table>
<tr><th>Convention</th><th>Item</th></tr>
<tr><td rowspan="2"><code>monospace text</code></td><td>Indicates user input, as shown in the following examples:
• Filenames: <code>config.xml</code>
• Pathnames: <code>BEAHOME/config/examples</code>
• Commands: <code>java -Dbea.home=BEA_HOME</code>
• Code: <code>public TextMsg createTextMsg(</code></td></tr>
<tr><td>Indicates computer output, such as error messages, as shown in the following example:
<code>Exception occurred during event dispatching:java.lang.ArrayIndexOutOfBoundsException: No such child: 0</code></td></tr>
<tr><td><code>monospace boldface text</code></td><td>Identifies significant words in code.
Example:
<code>void commit ( )</code></td></tr>
<tr><td><code>monospace italic text</code></td><td>Identifies variables in code.
Example:
<code>String expr</code></td></tr>
<tr><td><code>{ }</code></td><td>Indicates a set of choices in a syntax line. The braces themselves should never be typed.</td></tr>
<tr><td><code>[ ]</code></td><td>Indicates optional items in a syntax line. The brackets themselves should never be typed.
Example:
<code>java utils.MulticastTest -n name [-p portnumber]</code></td></tr>
<tr><td>|</td><td>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
Example:
<code>java weblogic.deploy [list|deploy|update]</code></td></tr>
</table>

Convention	Item
`. . .`	Indicates one of the following in a command line: • That an argument can be repeated several times in a command line • That the statement omits additional optional arguments • That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: `buildobjclient [-v] [-o name] [-f "file1.cpp file2.cpp file3.cpp . . ."`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

Designing Portals for Optimal Performance

Performance is always an issue in the acceptance of an enterprise-level software solution and WebLogic Portal is not immune to these issues. However, many performance problems are not the result of flaws in the product but rather the result of poor decisions made during the design phase of application development. Proper planning allows you to take advantage of the inherent strengths of WebLogic Portal to ensure optimal performance for your portals.

Portal performance is usually measured by the amount of time required to actually render that portal and all of its constituent parts once a visitor clicks an object on the screen (that is, sends a request to the portal servlet). Any number of reasons—all easily addressed and rectified by proper design—can negatively impact the anticipated system response, although foremost among them is portal layout and design.

This document examines several concepts and offers tips and practices for creating high-performing portals. This document contains the following sections:

- Control Tree Design
- Using Multiple Desktops
- Optimizing the Control Tree
- Other Ways to Improve Performance
- Thoughtful Design Decisions Ensure Optimal Performance

Control Tree Design

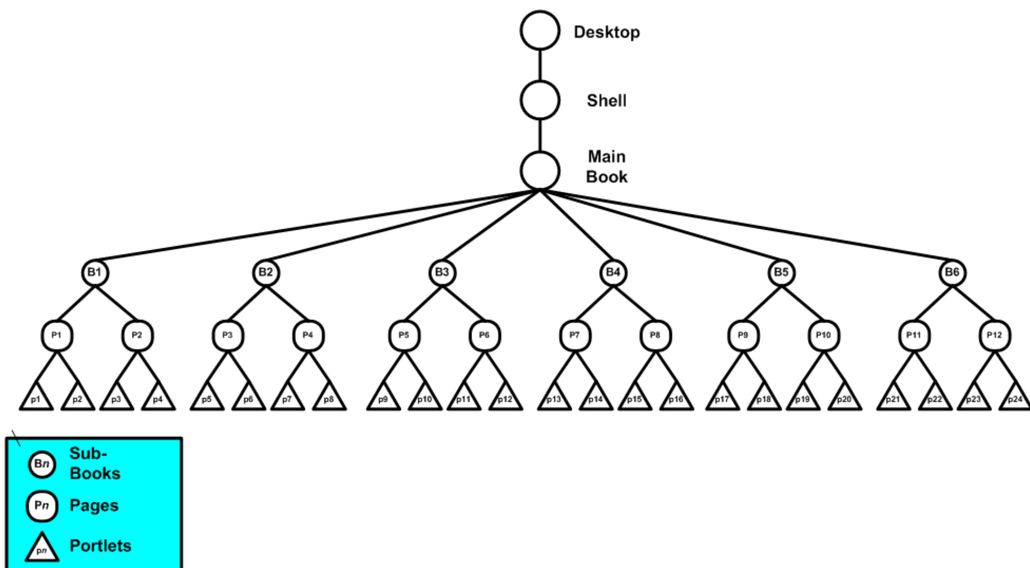
One of the most significant impediments to portal performance lies with the number of controls on a portal. The more portal controls (pages, portlets, buttons, etc.) you have, the larger your control tree. For a detailed description of the controls available for a portal, see Portal Controls at:

http://e-docs.bea.com/wlp/docs81/whitepapers/netix/body.html#1061553

How the Control Tree Works

When a portal is instantiated, it generates a taxonomy, or hierarchy of portal resources, such as desktops, books, pages, and portlets. Each resource is represented as a node on the control tree, as shown in Figure 1.

Figure 1 Simple Portal Schematic



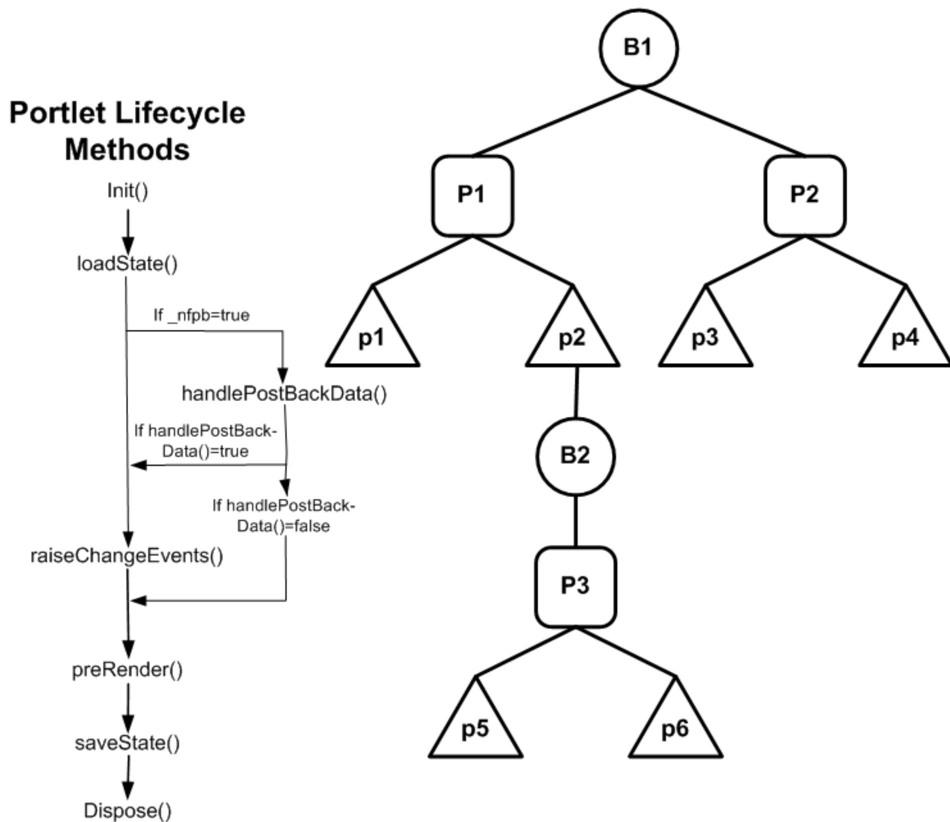
This example depicts a single portal with a main book containing six sub-books, which in turn contain two pages each, and each page contains two portlets each, for a minimum of 42 controls in the portal; the inclusion of buttons, windows, menus, and layouts will increase the number of controls on the portal significantly.

Note: This example is significantly oversimplified; enterprise portals might include thousands of controls.

How the Control Tree Affects Performance

Once the control tree is built and all the instance variables are set on the controls, the tree must run through the lifecycle for each control before the portal can be fully-rendered. The lifecycle methods are called in depth-first order. That is, all the `init()` methods for each control are called, followed by the `loadState()` method for each control, and so on in an order determined by the position of each control in the portal's taxonomy. For example, the control tree illustrated in Figure 2 depicts the taxonomy a simple portal comprised of a book (B1) containing two pages (P1 and P2), which each contain two portlets (p1-p4; note that p2 also contains its own subordinate book, page, and portlet hierarchy).

Figure 2 Control Tree with Lifecyle Methods



When this portal is rendered, the `init()` method (and `handlePostBackData()` if `_nfpb=true`) will be called first, for each control, in this order: B1, P1, p1, p2, B2, P3, p5, p6, P2, p3, and finally p4. Next, the `loadState()` method would be called in the same order, and so on for all lifecyle methods through `saveState()`.

Note: Control lifecycle methods `preRender()`, `render()`, and `dispose()` are called only on *visible* controls.

Running each control through its lifecycle requires some overhead processing time, which, when you consider that a portal might have thousands of controls, can grow exponentially. Thus, you can see that larger the portal's control tree the greater the performance hit.

Using Multiple Desktops

With SP3 and earlier versions of WebLogic Portal, the simplest way to limit the size of the control tree without limiting the flexibility of the portal is to split the portal into multiple desktops. In portal taxanomics, a desktop is nothing more than a portal embedded into another portal. It maintains the ability to leverage all of the features inherent in any portal and, within itself, can contain additional desktops.

Why this is a Good Idea

When you split a complex portal into multiple desktops, you spread the controls among those desktops. Since the control tree is scoped to the individual portal and since a desktop behaves much like a portal, each desktop has its own tree and the controls on that tree are built only when that desktop is opened. Thus, by splitting a complex portal with a large control tree into multiple desktops, you reduce the number of controls on the tree to just that number necessary for the active desktop. As you might guess, this will reduce the amount of time required to render the portal as a single desktop and increase portal performance.

When a portal is rendered, about 15% of the processing time is dedicated to constructing the control tree, 70% to running the lifecycle methods, and 15% in garbage collection (clearing dead objects from the heap, thus releasing that space for new objects). While construction and garbage collection are always performed, running the lifecycle methods is only necessary for visible controls (that is, those on the exposed desktop). This results in considerable overhead savings and improved system performance.

For example, the sample control tree depicted in Figure 1 shows a single portal with 42 controls. Were we to split this portal up into multiple desktops, as in Figure 3, while we would increase the number of control trees in the entire portal, each tree would be nearly two thirds smaller, and thus

be processed in roughly two-thirds the time, significantly reducing the time required to render the portal.

Figure 3 Simple Portal Split into Multiple Desktops

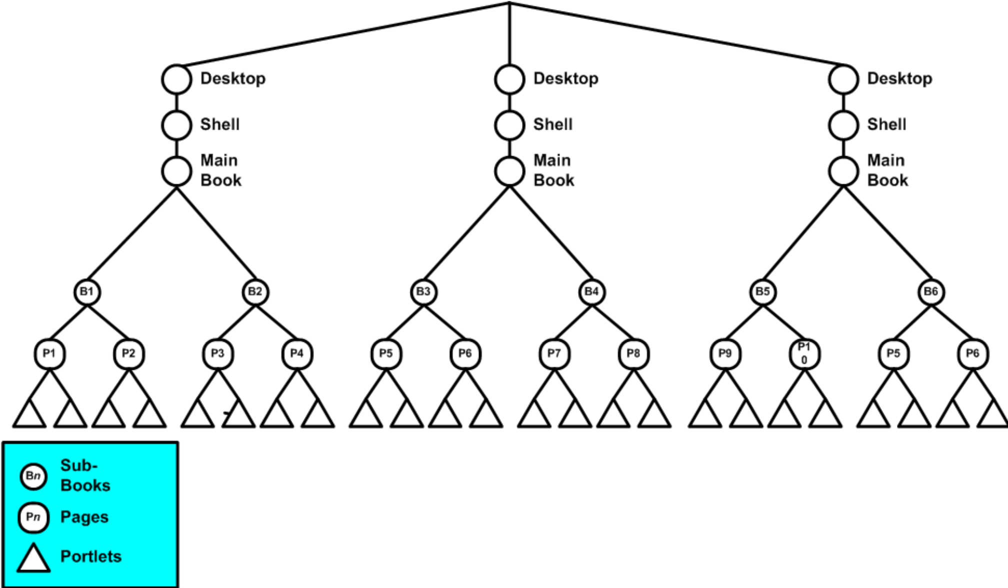
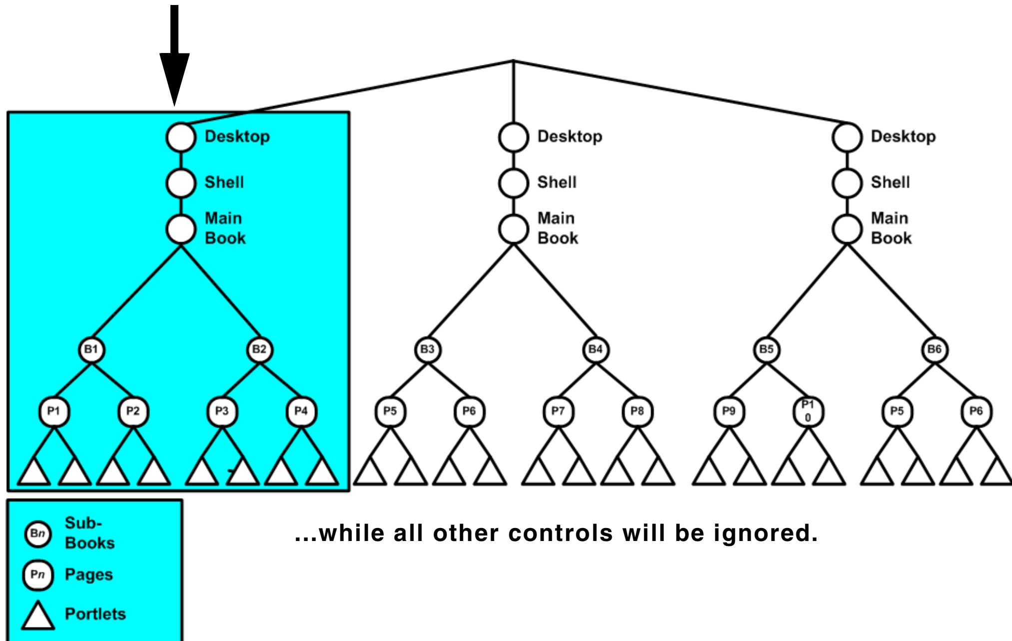


Figure 4 shows how the example in Figure 3 might be rendered once opened.

Figure 4 How Multiple Desktops Reduces Control Tree Size



Design Decisions for Using Multiple Desktops

As these examples demonstrate, splitting a complex portal into multiple desktops can be very rewarding in terms of improved performance; however, not all portals will benefit from the extra effort required to split them into multiple desktops. Before implementing a portal using multiple desktops, you need to consider some important design decisions. For example:

- **How many controls does your portal use?** If the portal is small (about ten pages or less) or uses a limited number of controls, the extra effort necessary to create multiple desktops might not be necessary.
- **Can your portal be logically divided into multiple desktops?** While splitting a complex portal into multiple desktops might save rendering time, arbitrarily assigning portlets to those desktops, with no thought to their interrelationships, can be dangerous.

Visitors might have a negative experience with the application if related information is not easily located, particularly if it is on a desktop separate from where it might logically go.

- **What sort of administrative overhead will be required once the multiple desktops are deployed into production?** For example, if you have 20 different potential desktops, a big consideration is how common they will be and that will help determine the strategy. If they are more alike than different, then fewer desktops is better because of smaller administrative tasks to perform.
- **Are there customization concerns?** Each desktop will have to be customized separately, which can add significant additional effort for portal developers and administrators. However, note that portal administrators can make changes in the library that will affect all desktops in the portal.
- **Can you afford to lose some functionality in your portal?** For example, if your application relies on interportlet communication, either through Page Flows or backing files, you might be better off not splitting-up the portal, as listeners and handlers on one desktop cannot communicate with their counterparts on other desktops. For portlets to communicate with each other, they will need to be on the same desktop and you portal design must take this requirement into consideration.

For more information on creating desktops, please refer to Create a Desktop in the WebLogic Portal Administration Portal online help system at:

`http://e-docs.bea.com/wlp/docs81/adminportal/help/PM_DesktopCreate.html`

Optimizing the Control Tree

WebLogic Portal 8.1 SP4 introduced the concept of control tree optimization. Tree optimization means that control tree rendering is done in a way that creates the least amount of system overhead while providing the user with as complete a set of portal controls as that user will need to successfully use the portal instance.

Enabling Control Tree Optimization

You enable control tree optimization by setting the `treeOptimizationEnabled` flag in the `.portal` file to true, as shown in Listing 1.

Listing 1 Enabling Tree Optimization in .portal

```
<desktop> element:
<netuix:desktop definitionLabel="defaultDesktopLabel"
```

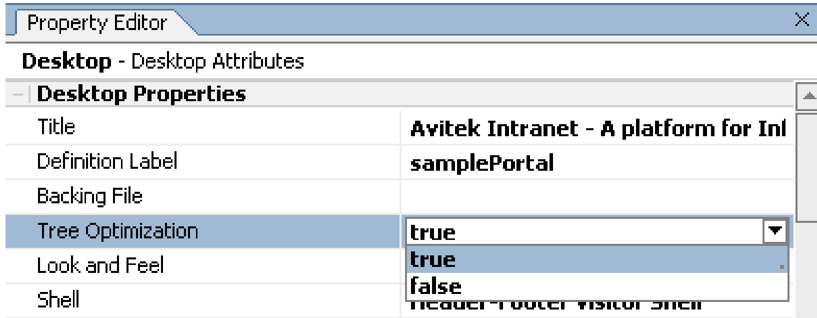
```
   markupName="desktop"  treeOptimizationEnabled="true"
   markupType="Desktop" title="SimplePortal"><netuix:lookAndFeel
   definitionLabel="defaultLookAndFeel">
<netuix:desktop/>
```

Note: If `treeOptimizationEnabled=` is not included in the `.portal` file, the portal defaults to `treeOptimizationEnabled=false`.

When this flag set to `"true"`, the portal framework will generate a partial control tree instead of the full control tree, basing this tree on just the controls that will be visible and active. Thus, with fewer controls needing to be rendered, processing time and expense can be significantly reduced.

- For portals, you can enable this flag by setting Tree Optimization to true in the WebLogic Workshop Properties Editor, as shown in Figure 5

Figure 5 Enabling Tree Optimization in WebLogic Workshop



- For desktops, you can set the flag from the Administration Portal, as shown in Figure 6.

Figure 6 Enabling Tree Optimization from the Administration Portal

Note: For new desktops, `treeOptimizationEnabled="true"` is the default value, so you really don't need to set anything in that circumstance.

Setting the Current Page

Before the flag can actually work, the file `url-template-config.xml` (in `<PORTAL_HOME>/`*`webAppName`*`/WEB-INF`) must have {url:`currentPage`} set in the `<url-template>` element, as shown in Listing 2.

Note: When you create a new project in WebLogic Workshop, `currentPage` is added automatically; however, if you are migrating from an earlier version of WebLogic Portal, you will need to manually update `url-template-config.xml`.

Listing 2 url-template-config.xml URL Templates Component

```
<!-- URL templates -->

<url-template name="default">
   {url:scheme}://{url:domain}:{url:port}/{url:path}?{url:queryString}
      {url:currentPage}
</url-template>
<url-template name="proxyurl">
   {url:scheme}://{url:domain}:{url:port}/{url:prefix}/{url:path}?
      {url:queryString}{url:currentPage}
</url-template>
<url-template name="finurl">
   https://fin.domain.com:7004/{url:prefix}/{url:path}?{url:queryString}
      {url:currentPage}&amp;dept=finance
```

```
</url-template>
<url-template name="default-complete">
    {url:scheme}://{url:domain}:{url:port}/{url:prefix}/{url:path}?
        {url:queryString}{url:currentPage}
</url-template>
<url-template name="jpf-default">
    http://{url:domain}:{url:port}/{url:path}?{url:queryString}
        {url:currentPage}
</url-template>
<url-template name="jpf-action">
    http://{url:domain}:{url:port}/{url:path}?{url:queryString}
        {url:currentPage}
</url-template>
```

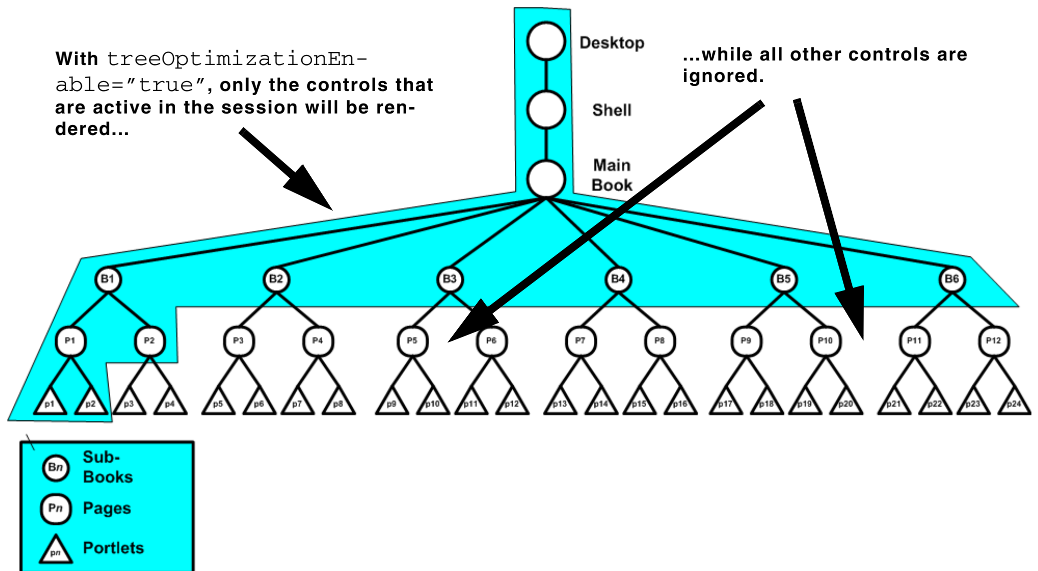
How Tree Optimization Works

When the portal servlet receives a request (that is, a mouse-click) it will read the cache to determine if a control tree factory exists. If one doesn't, it calls `controlTreeFactoryBuilder`, passing it the XML from the `.portal` file. This class returns a control tree factory to the servlet, which passes the request to the `CreateUIControlTree` class.

Assuming `_pageLabel` and `treeOptimizationEnabled="true"`, `CreateUIControlTreeFactory` calls the `PartialUIControlTreeCreator()` method, which returns a control tree comprised of just the control identified by the page label and the set of active page and book labels; this is a *partial* control tree.

For example, if tree optimizations were enabled for the portal depicted in Figure 1, when you submit a request (that is, a mouse click), only the active controls would be rendered, as illustrated in Figure 7.

Figure 7 How Tree Optimization Reduces Control Tree Size



The set of active page and book labels for that session stored during the `saveState()` lifecycle method execution tell `PartialUIControlTreeCreator`() which controls to build. Only these controls will be built; all others in the portal will be ignored. As you can see, a significant amount of processing overhead is eliminated when the control tree is optimized—since far fewer controls need to be built—resulting in greatly improved performance.

Limitations to Using Tree Optimization

For WebLogic Portal 8.1 with SP4 users creating complex portals that require a large number of controls, tree optimization is the easiest way to ensure optimal portal performance. Controls that aren't active in the current portal instance aren't built, saving considerable time and overhead. Nonetheless, you need to be aware that tree optimization slightly changes a portal's behavior and some portal implementations will not completely benefit from using it; for example:

- **The backing file lifecycle methods init() and handlePostBackData(),** which are called when the backing file is executed—even for non-visible controls—are not called when tree optimization is enabled, unless they appear on visible controls.
- **If your portal uses backing files on any of their controls**, some backing context APIs are limited in their functionality. On `DesktopBackingContext`,

`BookBackingContext`, and `PageBackingContext`, the following methods will return null if they are trying to access a page, book, or portlet that is not in the partial tree

- `public BookBackingContext getBookBackingContextRecursive(String definitionLabel)`
- `public PageBackingContext getPageBackingContextRecursive(String definitionLabel)`
- `public PortletBackingContext getPortletBackingContextRecursive(String instanceLabel)`
- `public PortletBackingContext[] getPortletsBackingContextRecursive(String definitionLabel)`

You might experience the same behavior—or lack thereof—on `DesktopPresentationContext`, `BookPresentationContext`, and `PagePresentationContext` with the presentation versions of these methods:

- `public BookPresentationContext getBookPresentationContextRecursive(String definitionLabel)`
- `public PagePresentationContext getPagePresentationContextRecursive(String definitionLabel)`
- `public PortletPresentationContext getPortletPresentationContextRecursive(String instanceLabel)`
- `public PortletPresentationContext[] getPortletsPresentationContextRecursive(String definitionLabel)`

- **If your portal uses multi-level menus** you need to decide if the benefit of multilevel menus outweigh any performance hit. If the menu is on an active book, every control accessible from that menu must be created before the portal is completely rendered, thus more overhead and a greater performance hit. On the other hand, because a multilevel menu results in the creation of a skeletal control tree, it can reduce the number of request cycles required to navigate to your desired destination, reducing the total overhead required to accomplish a navigation.
- **If your portal uses Programmatic Page Change Events** called from a backing file and the page to which the change is being directed is not within the partial control tree, it does not exist in the instance and the page change will not occur. You can work around this problem by doing one of the following (this is the preferred order):

  a. Use a link to perform the page change.

  b. Use the new declarative interportlet communications model.

  c. Implement a redirect from within the backing file.

d. Set `_nfto=false"` in the invoking link. This will cause the full control tree to be created for that single request.

e. Turn off tree optimization altogether on the portal.

- **If your portal uses "cookie" or "url" state locations**, the partial control tree *will not* work.
- **If your portal uses non-visible portlets, the onDeactivation portlet events for non-visible portlets may not work with portal tree optimization turned on.** When the "tree optimization" flag in a .portal file is turned on, not all non-visible portlets for a given request are processed. (A non-visible portlet is one that lives on a page that is not displayed for the given request.) This can be a problem if you are trying to catch an `onDeactivation` event for a portlet—once the portlet has been deactivated, it is no longer visible, and so the system doesn't process it to fire its deactivation event. The recommended workaround is to set tree optimization to false for the portal in question. However, there is a trick you can play if you need the tree optimization. For each portlet that you want to catch deactivation events for, define a dummy event handler (for example, create a custom event handler with event = "[some nonsense string]" and set the property "Only If Displayed" to false. This will force the system to process the portlet whether visible or not.

Mindful of these conditions, *you should never* set `treeOptimizationEnabled` to `true` without first doing a complete regression test on the portal. If any of the above-listed problems occur, you might want to rethink your portal design or disable tree optimization completely.

Disabling Tree Optimization

As discussed above, although control tree optimization can benefit almost any portal, behavioral limitations might require that you disable it. When you disable optimization, the portal will create a full control tree upon every request. Be aware that this could significantly impede the performance of very large portal. You need to decide whether the anticipated performance hit is offset by the improvement in functionality.

To disable tree optimization, do one of the following:

- Set `treeOptimizationEnabled= "false"` in the `.portal` file or on the desktop.
- Include `nfto=false"` in the request parameter of just that instance for which you want to disable tree optimization. The parameter needs to be added to URL programmatically as the URLs are generated using framework classes `GenericURL` and `PostbackURL`; for more information on these classes, see the WebLogic Portal Javadoc.

The following code shows one way to adding this parameter:

```
PostbackURL url = PostbackURL.createPostbackURL(request, response);
   url.addParameter(GenericURL.TRE_OPTIMIZATION_PARAM, "false");
```

- Use one of the tags in the `render` tag libraries.
- Delete the `_pageLabel` parameter from the request.

Other Ways to Improve Performance

In addition to managing the taxonomy of your portal through effective use of the control tree, WebLogic Portal offers other ways to improve performance. These solutions can all be used in concert with multiple desktops and control tree optimization, ensuring superior portal performance. This section describes the most effective performance-enhancing solutions available with WebLogic Portal.

- Use Entitlements Judiciously
- Limit User Customizations
- Optimize Page Flow Session Footprint
- Use File-based Portals for Simple Applications
- Create a Production Domain in Development
- Use Remote Portlets
- Customize Portlets to Take Advantage of Optimization Features
- Use Backing Files
- Use Pagination When Displaying Large Amounts of Data in a Portlet

Use Entitlements Judiciously

Entitlements determine who may access the resources in a portal application and what they may do with those resources. This access is based on the role assigned to an application visitor, allowing for flexible management of the resources. For example, if you have an Employee Review portlet, you can assign the "Managers" visitor entitlement role you created to that portlet, letting only logged in users who belong in that role view the portlet.

Users visiting an application are assigned roles based on an expression that can include their name, the group that they are in, the time of day, or characteristics from their profile. For

example, the “gold member” role could be assigned to a user because they are part of the frequent flyer program and have flown more than 50,000 miles in the previous year. This role is dynamically assigned to the user when they log into the site.

How Entitlements Affect Performance

To ensure optimal portal performance, you should use entitlements judiciously. Too many entitlements can significantly impact performance. This happens because the entitlement engine is called during the render phase of an operation and is required to check system overhead and rules. Because this checking represents additional system overhead, if it is required too often on a portal, performance will suffer. In addition, the entitlements engine is also responsible for managing administrative tasks, which increases that overhead, again causing degrading performance.

With the release of WebLogic Portal 8.1 with Service Pack 4 and later, performance related to entitlements has been improved by storing the entitlements in the database as opposed to LDAP. Nonetheless, you should always be aware that too many entitlements will impede performance.

Recommendations for Using Entitlements

Here are some simple recommendation for using entitlements judiciously:

- **Avoid the temptation to create a role for every node on an organizational chart.** In large organizations, granting entitlements would then become a serious burden on the system. If you want to focus the user experience to a more granular level than that provided by the role assigned a user, consider employing the personalization capabilities available with WebLogic Portal 8.1.
- **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 the `<control-resource-cache-size=`*`nn`*`>` attribute to equal the number of desktops + number of books + number of pages + number of portlets + number of buttons (max, min, help, edit) used in a portal. Use the default value if you are concerned about available memory.
- **Limit your entitlement request to only one resource at a time.** Bundling a larger number of resources (portlets, pages, books) with one entitlement request can cause an unwanted performance hit.
- **If your portal uses more than 5000 entitlements, customize the cache settings for WebLogic Entitlements Engine.** For details, see Tuning for Entitlements in the WebLogic Portal *Performance Tuning Guide*, at:

http://edocs.bea.com/wlp/docs81/perftune/4PortalApplication.html#1073678

Limit User Customizations

BEA recommends that you allow users (visitors) to modify only one page or a small set of pages, and require that administrators control the remainder of pages.

When users customize a page, they get their own instance of that page. All other pages that have not been customized point back to the original library instance. When an administrator makes a change to a page, that change must iterate for each user who customized the page. If many users customized that page, propagating the change might take a long time because of the required database processing.

Optimize Page Flow Session Footprint

If your portal uses Page Flows portlets in a replicated clustering environment, you might experience a performance issue because the request attributes you add to these portlets might be persisted to the session as a component of a Page Flow portlet's state. As more request attributes are added, the session grows, often to sizes that can severely restrict performance.

Page Flow portlets are hosted within the Portal framework by the Scoped Servlet environment. This environment effectively acts as a servlet container, but its behavior causes the request attributes to be scoped to the session within a container class used to persist Page Flow state. This can be particularly unwelcome in clustered environments, when large amounts of data—including these Page Flow portlet request attributes—might be replicated across the cluster.

To improve performance in these circumstances, WebLogic Portal 8.1 with Service Pack 4 and later provide the `requestAttrPersistence` attribute for Page Flow portlets. `requestAttrPersistence` is included in the `.portlet` file and can be set by from the Properties Editor in WebLogic Workshop.

`requestAttrPersistence` has these values:

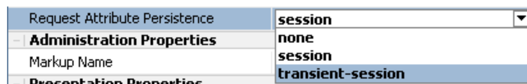
- **`session`**: this is the existing behavior (this is the default). All existing Page Flow portlets should not require changes by default.
- **`transient-session`**: places a non-serializable wrapper class around a persisted Page Flow state object into the session. These portlets work just as the existing portlets, except in failover cases, where the persisted request attributes will disappear on the failed-over-to server. In these cases you will need to write the forward JSPs to gracefully handle this contingency by, at minimum, not expecting any particular request attribute to be populated and, ideally, by having a mechanism to either repopulate the request attributes

automatically or present the user with a link to re-run the last action to repopulate the request attributes. For non-failover cases, request attributes will be persisted, providing a performance advantage for non-postback portlets identical to default `session` persistence portlets. While session memory will still be consumed in this case, there will be no additional cluster replication costs for the persisted request attributes.

- **`none`**: performs no persistence operation. Since these portlets never have request attributes available on refresh requests, you must write the forward JSPs to assume the request attributes will not be available. This option is helpful when you want to remove completely the framework-induced session memory loading for persisted request attributes.

To set the request attribute persistence attribute for a page flow portlet, open the Request Attribute Persistence drop-down under the Page Flow Content group in the WebLogic Workshop Properties Editor and select the desired value, as shown in Figure 8.

Figure 8 Selecting Request Attribute Persistence Attribute



Use File-based Portals for Simple Applications

Portals come in two flavors: file-based and streaming. As the name implies, a file-based portal—also called a “light portal”—obtains all of its resources from the user’s file system. Streaming portals, on the other hand, derive their resources from one to many databases. Although file-based portals are intended for development purposes, if you are creating a static portal (that is, a portal that doesn’t require customization customized by the end user or administrator) or very simple files, you might encounter some degree of performance improvement in a production environment.

Why Use a File-based Portal?

For simple, static portals, deriving resources from the file system can result in improved performance and bring these benefits:

- Source code control is easily manageable.
- Propagation to other environments is easy.
- They are easy to create in WebLogic Workshop.

Limitations to Using File-based Portals

While file-based portals might show some performance improvement over streaming portals, their functionality is limited; for example, because no database is involved, you can't take advantage of things such as user customization or entitlements. Other features that are missing from a file-based portal include:

- Delegated Administration
- Visitor Tools
- Preferences at the portal instance level and at the definition level.

Moreover, in the majority of cases, the performance improvement gained by using a file-based portal is not so significant as to outweigh these limitations.

Create a Production Domain in Development

While this tip doesn't directly improve performance at runtime, it nonetheless allows you to see how your application will perform before you propagate it to production. By creating a production domain in development, you can simulate and then evaluate how the portal will perform in production. You can then make the necessary adjustments before actually deploying the portal. If problems occur or performance is not optimal, you can rectify these situations before the end user ever sees them.

To create a production domain, you will need to update the startup script settings by setting the `WLS_PRODUCTION_MODE=` flag to `true` and setting to false these flags:

- `iterativeDevFlag`
- `debugFlag`
- `testConsoleFlag`
- `logErrorsToConsoleFlag`
- `pontbaseFlag`
- `verboseLoggingFlag`

Additionally, you will need to set default values for the threadcount and the JDBCConnectionPool sizes. If you are threading portlets (that is, using `forkable=true`) ensure that you configure a `portalRenderQueue` and/or `portalPreRenderQueue` in your `config.xml` file so that the forked portlets use their own thread pools and not the WebLogic thread pool. The following code sample describes how to set the thread count appropriately:

```
<ExecuteQueue Name="default" ThreadCount="15"/>
<ExecuteQueue Name="portalRenderQueue" ThreadCount="5"/>
<ExecuteQueue Name="portalPreRenderQueue" ThreadCount="5"/>
```

Use Remote Portlets

Implementing proxy, or "remote," portlets might result in some performance improvement, although this method also comes with its limitations. Remote portlets are made possible by Web Services for Remote Portlets (WSRP), which was first integrated into WebLogic Portal 8.1 with Service Pack 3. WSRP is a web services standard that allows you to "plug-and-play" visual, user-facing web services with portals or other intermediary web applications. It allows you to consume applications from WSRP-compliant Producers, even those far removed from your enterprise and surface them in your portal.

For more information on using WSRP, please refer to *Using WSRP with WebLogic Portal* at:

http://edocs.bea.com/wlp/docs81/wsrp/index.html

Do Remote Portlets Really Provide a Performance Boost?

While you can reduce development time by using a remote portlet—because you don't have to actually develop the contents of the portlet, just its container—the major performance benefit is that any controls within the application (portlet) you are retrieving are rendered by the producer and not by your portal. The expense of calling the control lifecycle methods is borne by resources not associated with your portal.

Although using remote portlets can improve portal performance, it is not without its drawbacks. For example:

- Fetching data from the producer can be expensive. You need to decide if that expense is within reason given the resources locally available.
- If you are using WebLogic Portal 8.1 with Service Pack 3 or earlier, you cannot establish interportlet communications with remote portlets. Event listeners in your local portlets cannot react to events on the remote portlet. Fortunately, this limitation has been overcome with WebLogic Portal 8.1 with Service Pack 4 and later. If interportlet communications is necessary to your application, you might either want to avoid using remote portlets or upgrade your version of WebLogic Portal 8.1 to Service Pack 4 or later.
- Some features, such as customizations, are unavailable to the remote portlet.

If the expense of portal rendering sufficiently offsets the expense of transport and the other limitations described above are inconsequential to your application, using remote portlets can provide some performance boost to your portal.

Customize Portlets to Take Advantage of Optimization Features

Customizing your portlet settings can help you avoid performance problems. Specifically you can do the following:

- Cache portlets. Caching portlets allows you to cache the portlet within a session instead of retrieving each time it recurs during a session (on different pages, and so on).
- Set process-expensive portlets to pre-render and/or render in a multi-threaded (forkable) environment. If a portlet has been designed as forkable (multi-threaded) you have the option of adjusting that setting when building your portal. This can increase performance of portlets whose processing can be time-extensive, such as RSS feeds.

  Note that if you are using page flows, the initial begin actions and refresh actions run during the pre-render phase.
- Use backing files to handle preprocessing of portlet functionality.

Table 1, "Performance-related Portlet Properties," on page -20 lists performance-related portlet properties that are available when creating/modifying portlets in WebLogic Workshop. If portlets are designed to allow portal administrators to adjust cache settings and rendering options, you can modify those properties in the Administration Portal.

Table 1 Performance-related Portlet Properties

Portlet Property	Use
Render Cacheable	Optional. To enhance performance, set to "true" to cache the portlet. For example, portlets that call Web services perform frequent, expensive processing. Caching Web service portlets greatly enhances performance. Do not set this to "true" if you are doing your own caching. This property must be set to false if you want to allow portal administrators to modify the cache setting in the Administration Portal.
Cache Expires (seconds)	Optional. When the "Render Cacheable" property is set to "true," enter the number of seconds in which the portlet cache expires.

Table 1 Performance-related Portlet Properties

Portlet Property	Use
Fork Render	Intended for use by a portal administrator when configuring or tuning a portal. Setting to "true" tells the framework that it should attempt to multithread render the portlet. This property can be set to true only if the "Forkable" property is set to "true."
Fork Render Timeout (seconds)	Allows you to set a timeout value for rendering. If Fork Render is set to "true," you can set a timeout attribute to indicate that the portal framework should wait only as long as the timeout value for each fork render portlet. The default value is -1 (no timeout). When a portlet rendering times out, an error is logged, but no markup is inserted into the response for the timed-out portlet.
forkPreRender	Enables forking in the preRender lifecycle phase (see How the Control Tree Affects Performance for more information about the control tree lifecycle). preRender forking is supported by these portlet types: • JSP • Page Flow • JSR168 • WSRP (Consumer portlets, only) Setting forkPreRender to true indicates that the portlet's preRender phase should be forked. If you are using page flows, the initial begin actions and refresh actions run during the preRender phase. At this time, Fork PreRender is not supported in the WebLogic Workshop interface. To implement it, please refer to the instructions in the topic "How Do I: Enable the forkPreRender Attribute?" in the WebLogic Workshop online help system.

Table 1 Performance-related Portlet Properties

Portlet Property	Use
forkPreRender Timeout	Takes an integer value and sets the timeout value, in seconds, for the forked preRender phase. This attribute is applied only if forkPreRender="true". If the time to execute the forked preRender phase exceeds the timeout value, the portlet itself times out (that is, the rest of the lifecycle phases for this portlet are aborted; see How the Control Tree Affects Performance for more information about the control tree lifecycle), the portlet is removed from the page where it was to be displayed, and an error level message is logged that looks similar to the following example. <May 26, 2005 2:04:05 PM MDT> <Error> <netuix> <BEA-423350> <Forked render timed out for portlet with id [t_portlet_1_1_1]. Portlet will not be included in response.> At this time, forkPreRenderTimeout is not supported in the WebLogic Workshop interface. To implement it, please refer to the instructions in the topic "How Do I: Enable the forkPreRenderTimeout Attribute?" in the WebLogic Workshop online help system.
Forkable	Optional. Lets a portlet developer determine whether or not the portlet is allowed to be multithread rendered or pre-rendered. When set to "true," a portal administrator can use the "Fork Render" or "Fork PreRender" property within the Administration Portal.

Use Backing Files

Backing files allow you to programmatically add functionality to a portlet by implementing (or extending) a Java class, which enables preprocessing (for example, authentication) prior to rendering the portal controls. Backing files can be attached to portals either by using WebLogic Workshop or coding them directly into a .portlet file.

For more information see the Interportlet Communication Guide.

Use Pagination When Displaying Large Amounts of Data in a Portlet

If you have a very large amount of data to render in a portlet, consider using pagination, especially if the data resides in tables. For example, Internet Explorer renders tables in such a way as to decrease performance when handling a lot of data. If you want to make sure rendering performance is adequate using Internet Explorer as well as other browsers, and you want to use

tables to display the data, you can use pagination so that the portlet does not have to handle all the data at once. Otherwise, consider creating a custom Look & Feel that does not use tables.

Thoughtful Design Decisions Ensure Optimal Performance

As we have discussed, a great many performance problems you might encounter with WebLogic Portal can be attributed to poor design decisions. This paper has demonstrated that some of the most serious performance issues—those dealing with portal rendering—can be resolved and significant performance improvement can be realized by making just a few critical design decisions:

- If you are using a pre-Service Pack 4 version of WebLogic Portal, consider breaking up complex portals into multiple desktops.
- If you are using Service Pack 4 or later, enable control tree optimization.
- Use entitlements judiciously; too many can impact performance. Avoid the temptation of granting a different role to every user. Instead, use WebLogic Portal's personalization capabilities to focus the user experience.
- If your portal is small or relies only on static resources, you might experience some performance boost by using a file-based portal rather than a streaming portal.
- If you are using Page Flows in your portal, ensure their session footprint is optimized to deliver the best performance.
- Develop your portal in a production domain. This way, you can actually see what issues might arise once the portal is propagated to production.
- Remote portlets might provide improved performance, but be aware that they come with some limitations.