Logical Structure of a Portal

The Manage Pages window allows you to add, delete, edit, and reorder pages in your portal. The portal uses a treelike hierarchical structure, where the root of the tree is called Content Root. Nodes in this hierarchy belong to one of the following types:

- Pages
- Labels
- URLs

Pages are the basic building block used for displaying content in the form of portlets. Page nodes may contain other nodes, including other pages. A label is a placeholder for pages. Labels may contain other nodes, but display no content, and are used to provide logical grouping of nodes. This logical grouping is used to define common attributes such as access permissions. URLs are used to access any resource that can be addressed with a URL. URLs obviously do not contain any other nodes. URLs are useful for addressing external Web sites or another part of the portal.

Note that there is no built-in notion of a "portal." You may define multiple portals inside WebSphere Portal by using the abstractions of pages or labels. For example, you can create different subhierarchies for different sets of users and consider each such subhierarchy as a portal.

Figure 7-1 shows the composition of the Content Root as per the default configuration, immediately following product installation. We have five different nodes under Content Root of which three are labels and two are pages.

My Portal is a label containing the default portal configuration shipped with the product. By default, users who do not have administrative privileges can only access this node. The Administration label is accessible by users with administrative privileges, and incidentally is the label through which you have reached the current page. The last label is Page Customizer, which contains several portlets used for configuring page layout and content. This label is not accessible directly, but is invoked through the button in the portal toolbar. The two pages under Content Root are Page Properties and Organize Favorites. As the names imply, the former shows a portlet used to edit some properties of pages, and the latter allows users to organize favorites through the My Favorites drop-down menu in the portal banner. Note that Page Properties is hidden from navigation, which means you cannot see a corresponding entry in the portal toolbar.

This is in fact the default behavior for entries that are directly under Content Root. Typically, one would add new content at lower levels, for example, under My Portal (in the next chapter you will see how to customize the portal toolbar allowing you to access from there entries that are directly

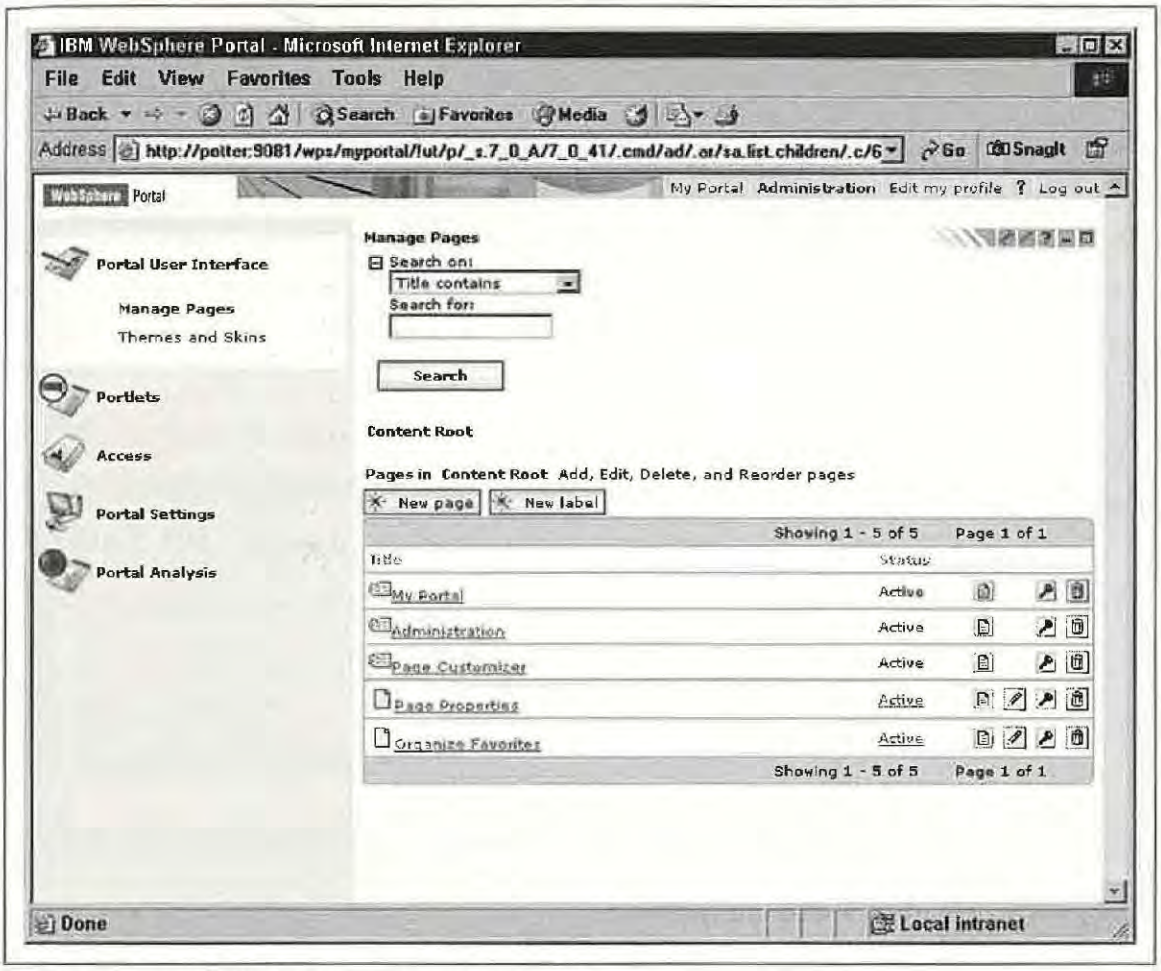


Figure 7-1 Default Content Root composition.

under Content Root). The page properties are accessed when you click the context-sensitive Properties icon on a window or under Manage Pages.

Browsing through the Content Hierarchy

The Manage Pages window allows you to browse through the content hierarchy. For example, clicking on My Portal would go to the next level down this hierarchy and show the nodes under My Portal, as shown in Figure 7-2. Note that the options at the top of the page allow you to search for a page by various criteria, and below that you see the current node you are viewing, as well as the path in the hierarchy leading up to it, starting with Content Root. Each of the parent node names is a clickable link, bringing you to that page. On the right-hand side of the current node name are some buttons (Figure 7-3 summarizes the buttons available under Manage Pages):

- Editing page properties
- Editing page layout (only for page nodes)

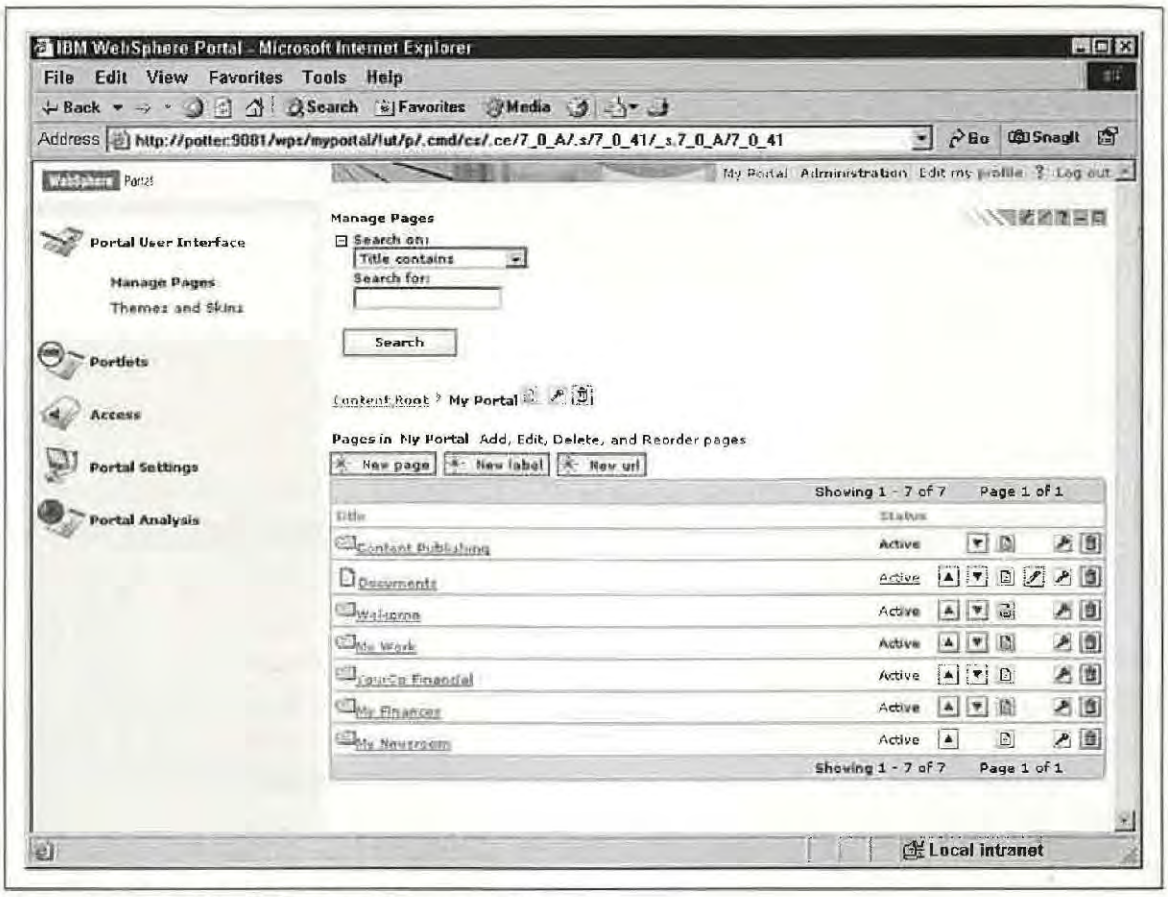


Figure 7-2 Default My Portal composition.

- Controlling access permissions
- Deleting the node

A table below the current node summarizes the children nodes for this node. In the case of My Portal in Figure 7-2, we have six groups and one page. Clicking on any of the child node names brings you to that node, allowing you to drill lower in the portal structure. Each one of the children pages has three or four buttons similar to the ones beside the current node name. In addition, each such children page has an up arrow, or down arrow, or both. These arrows allow you to change the ordering of children nodes. The ordering of children nodes controls their order in toolbars.

Button	Meaning
	Edit Page Layout
	Remove Page
	Move down in current node
	Move up in current node
	Edit Properties
	Set Page Permissions

Figure 7-3 Buttons available under Manage Pages.

Customizing the Portal Structure

WebSphere Portal provides several levels of customization. Here we will look at customizing the structure—that is, adding new pages and editing their attributes. In the next chapter we will look into customizing the Web design of the portal. It is important to remember that the most powerful way of customizing the portal is by adding new portlets, but we will get to this only in Part 3 of the book.

Adding New Nodes

To add a new node under My Portal, go to Manage Pages and navigate to the My Portal page (as shown in Figure 7-2).

1. Click the New Label button.
2. Enter **My Stuff** in the Title field.
3. Select OK.
4. Select OK in the confirmation window shown next.

The new group will be added under My Portal as shown in Figure 7-4. Now add a new page in this group.

1. Click My Stuff in the table of children nodes shown in Figure 7-4.
2. Click the New Page button.
3. Enter **Sample Page** in the Title field.
4. Click the Advanced Options button.
5. Click the two-column under A Content Page with These Properties (second option).
6. Select OK.
7. Select OK in the confirmation window shown next.

Finally, add a new URL under the same group. As mentioned above, we can use a URL node to point to any URL-addressable resource.

1. Click the New URL button.
2. Enter **Thinkpad** in the Title field.
3. Enter **http://www.ibm.com/thinkpad** in the HTML field.
4. Select OK.
5. Select OK in the confirmation window shown next.

Now you should see both new nodes under My Stuff, as shown in Figure 7-5.

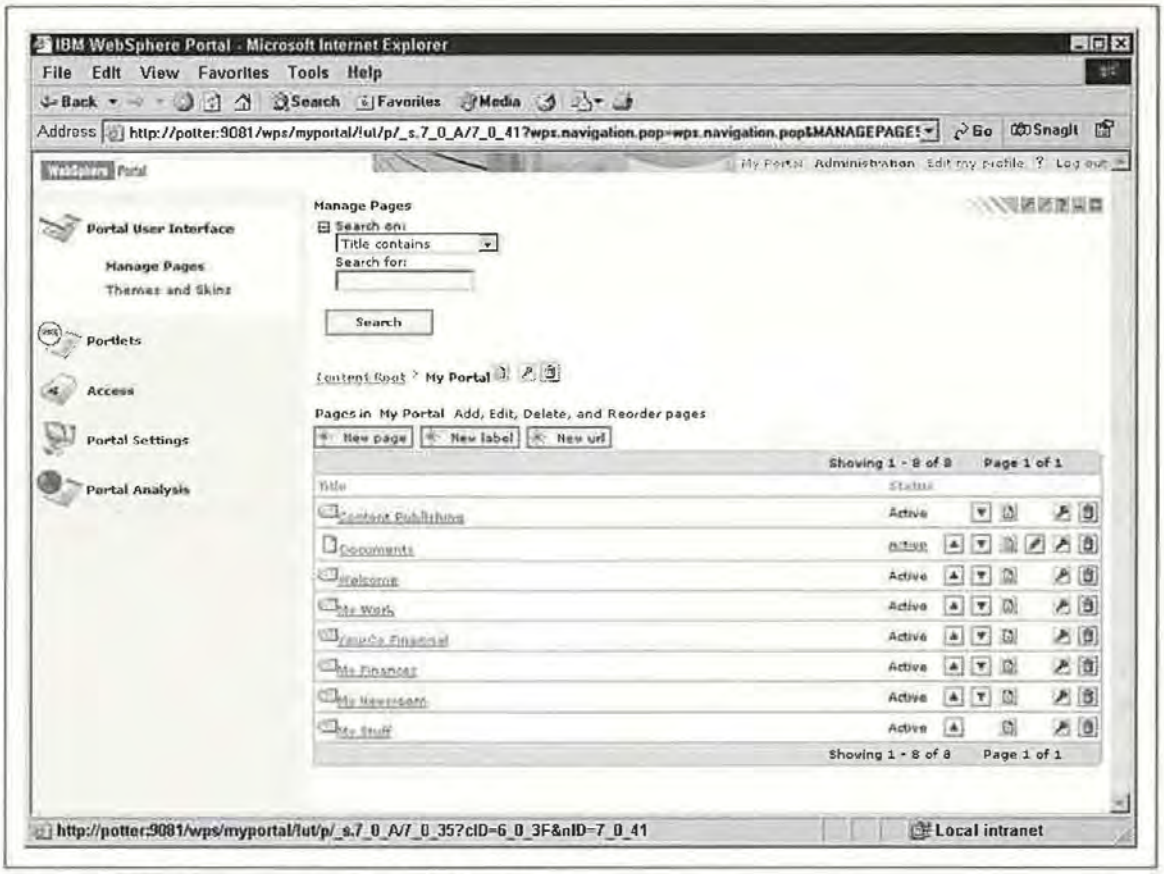


Figure 7-4 New group under My Portal.

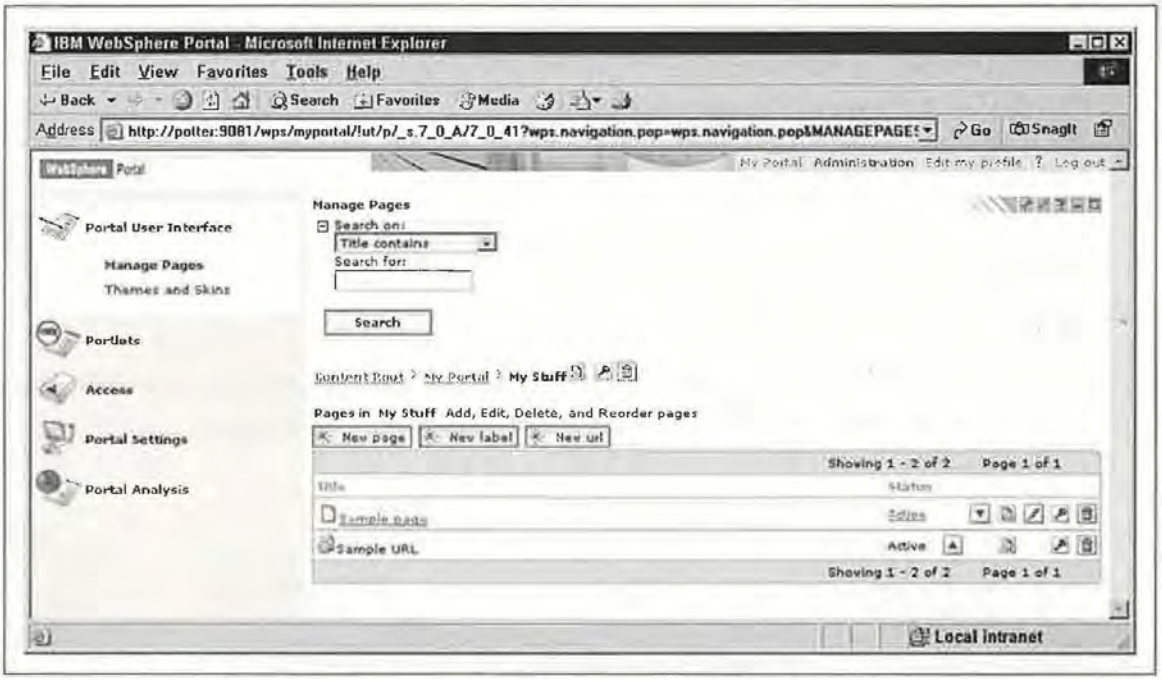


Figure 7-5 User view of new group My Stuff under My Portal.

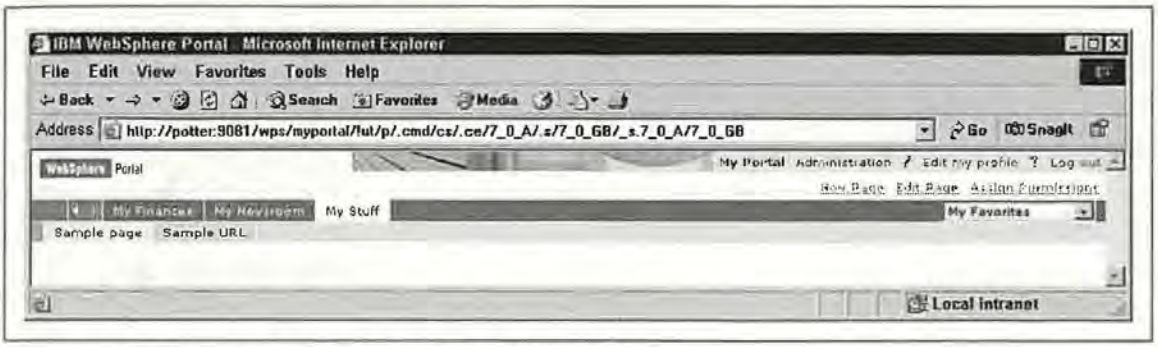


Figure 7-6 Viewing My Stuff through My Portal.

To see the new group from the user perspective, click My Portal on the portal toolbar, and then click My Stuff on the second layer toolbar. You should see the window shown in Figure 7-6. As you can see, the page Sample Page is empty, and clicking on Sample URL gets you to the IBM ThinkPad site.

The interesting thing to note at this point is how smoothly the new components were integrated into the portal as perceived by the user. If you have any experience with WebSphere Application Server (or any other application server), you will appreciate this seamless update of the configuration without the need to restart the server or any container.

Defining Page Content

In the previous section we added a new page but it remained empty. To place some content in the page, we can edit the page layout. To edit a page layout you need to go to the page and click the Edit Page link just below the portal toolbar. Another way is to go to Manage Pages, navigate to Sample page and click the Edit Page Layout button (denoted by a pencil icon).

The Edit Layout page (shown in Figure 7-7) allows you to choose between six layout types:

- One column
- Two column
- Three column
- T layout
- Reverse T layout
- I layout

Our page was defined as a two-column page, so it includes two adjacent containers. Each of these containers is a column container, which means elements within it will stack vertically.

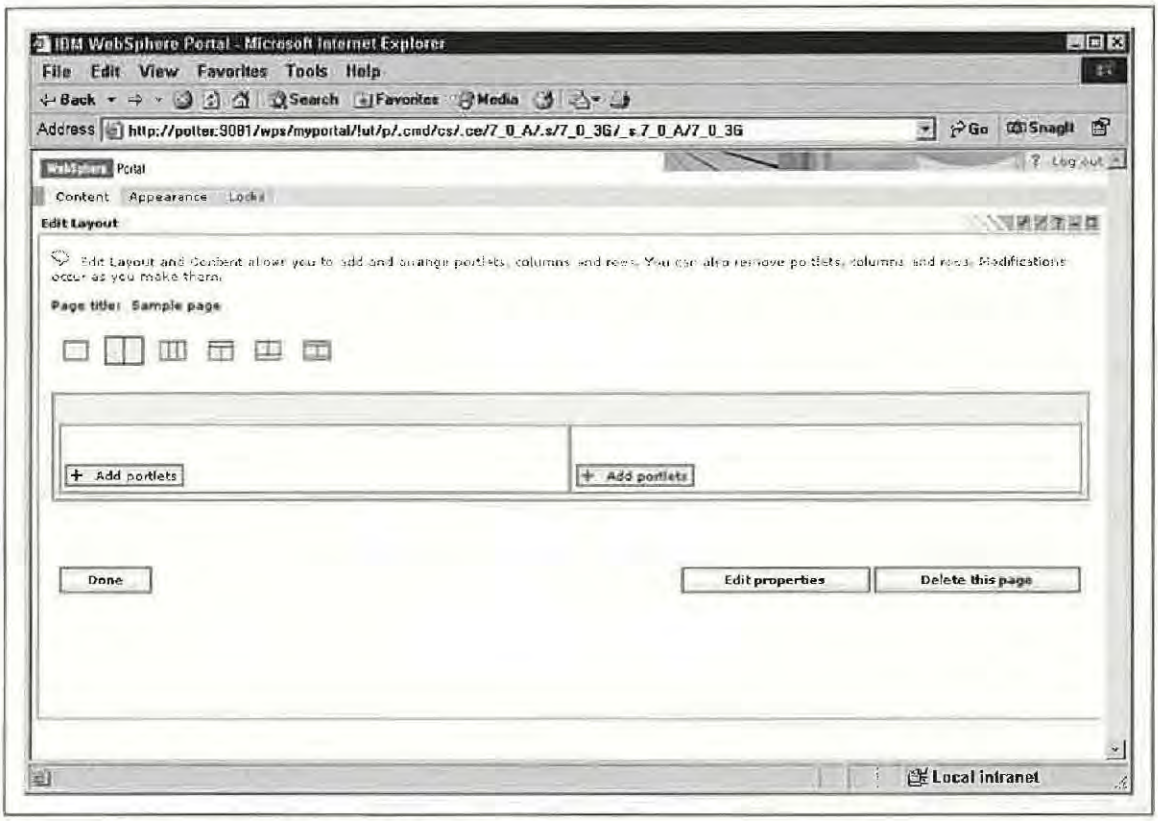


Figure 7-7 Page Customizer Content tab.

Below the layout type you can insert portlets into each of the containers. In our case we have two containers, and each can have several portlets. To add a portlet in the left column, simply click the Add Portlet button below that column.

A new window opens, and here you may select which portlet to add. The set of portlets available depends on whatever portlets were installed. Later in the book we will see how to develop our own portlets, but for now we will stick to the ones packaged into the default installation. For example, add the World Clock portlet (in version 5.0, it is the 41st entry shown on the 5th page). Once we add the portlet, it is visible inside the column container, with the following two buttons:

- Left arrow. Move to the container on the right
- Trashcan. Delete portlet

You may add a portlet on the right column as well, for example the Currency Calculator portlet. Note that this portlet will have a button for moving it to the container on the left. In general, a portlet will have a button for moving it in each of the four possible directions (left, right, up, and down) where there is a container (see Figure 7-8).

Button	Meaning
	Edit Page Layout
	Remove portlet
	Move portlet down
	Move portlet up
	Move portlet left
	Move portlet right

Figure 7-8 Page Customizer Portlet buttons.

Note that at this point you may also edit the page properties via the Edit Properties button and delete the page using Delete this Page button.

Once you click the Done button, the page is instantaneously updated, and you may view it going back to My Portal.

Editing Node Properties

Node properties are edited as part of addition of a new node, as we have seen earlier for both page and label nodes. When editing the properties of a label node we may modify the following properties (see Figure 7-9):

- Title. Text field controlling the title of the label
- Theme. Theme used for Web design (more on themes will be discussed in the next chapter)
- Advanced options.
  - This page supports: WML, cHTML, HTML. Markup language support. The default markup language for all pages is HTML, and a markup is only available for selection if supported for the parent node of the label.
  - I want to set titles and descriptions. Allows you to set the title and description for this label in each of the locales defined (24 of them by default).

When editing a page node, you may modify the following properties:

- Title. Text field controlling the page title
- Advanced options.
  - This page can be added to a user's My favorites list. A check box determining whether a user may bookmark (using the portal's My favorites drop menu) the page. This check box is checked by default.

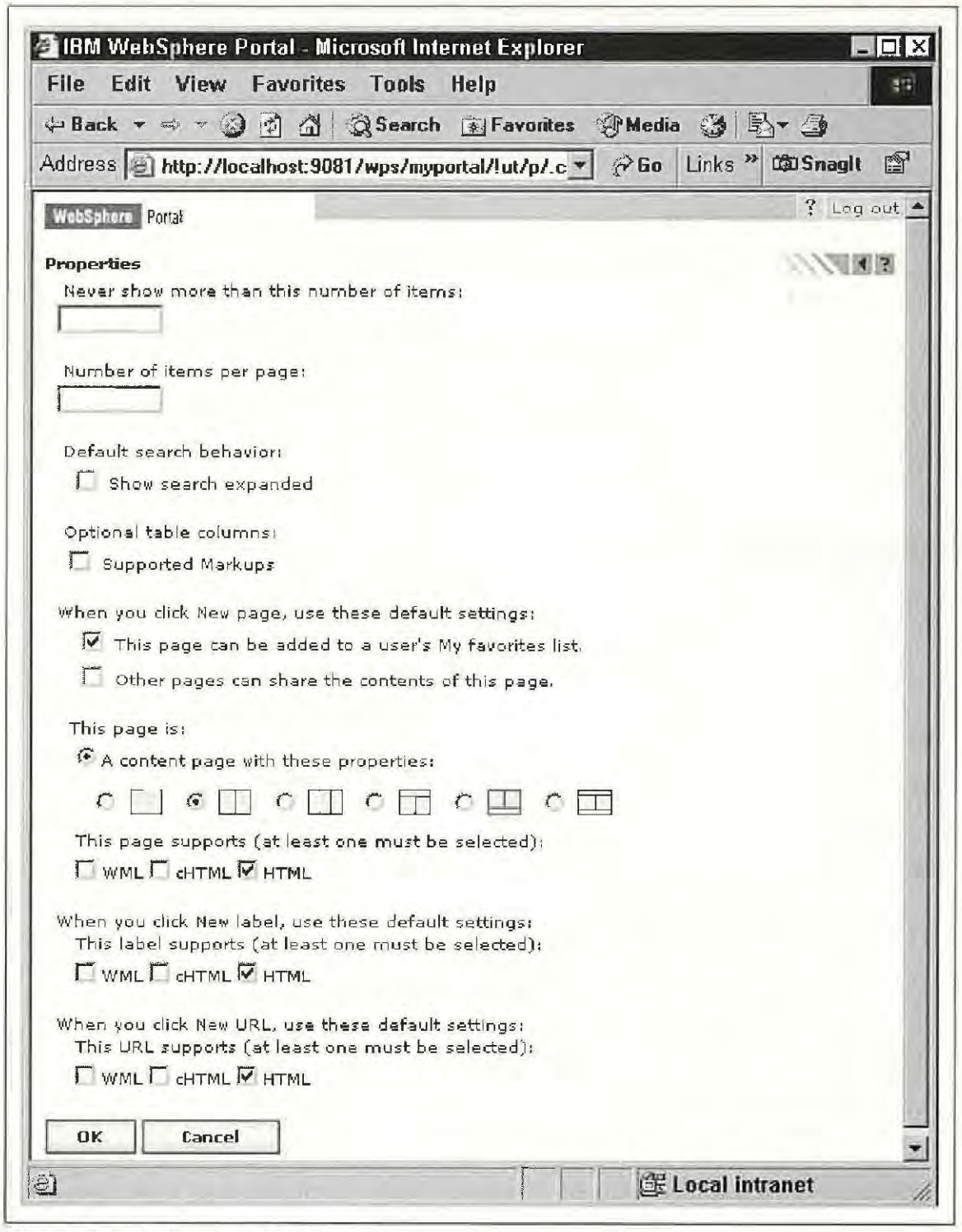


Figure 7-9 Editing node properties.

- Other pages can share the content of this page. Check box indicating whether the page can be shared (see discussion below). This check box is unchecked by default.
- This page supports: WML, cHTML, HTML. Markup language support. The default markup language for all pages is HTML, and

a markup is only available for selection if supported for the parent node of the page.

- This page has a list of allowed portlets. If this check box is clicked, the set of portals allowed within the page is limited. To set the list of allowed portlets, click the text beside the check box. This check box is unchecked by default.
- I want to set titles and descriptions. Allows you to set the title and description for this page in each of the locales defined (24 of them by default).

Page Inheritance with Shared Pages

When adding a new page, Advanced Options (shown in Figure 7-10) allows choosing the page content based on one of the six default layouts or based on another page. In the latter case, the other page must be one that was defined as a page that can be shared. When content of a new page is based on another page, the new page is essentially the aggregation of the existing page with a new specialized layer on top of the latter. This implies that a prerequisite for accessing a page using a shared page is that both the page and the shared page must be accessible by the user.

Shared pages essentially implement a form of inheritance. A page based on a shared page inherits the layout and content of the latter. An important characteristic of shared pages is that any changes made to the shared page are reflected on all the pages referencing it. For example, the addition or removal of a portlet from a shared page is immediately reflected in all referencing pages. This makes shared pages useful for implementing cascading portals.

Cascading portals are used in environments where the portal needs to meet the requirement of a central organization as well as those of units within the organization. For example, a portal for a certain department might share the structure of the basic portal of the whole company but add on department-specific content. In this case, one would like to share the page layout as well as visual elements reflecting the company identity.

WebSphere Portal provides a powerful paradigm for building cascading portals through the shared page mechanism. A shared page can reference another shared page, thus creating a chain of page references. From a practical standpoint, this means that top-level administrators may define a certain set of pages, and allow lower-level administrator to further refine and enhance the layout and content of pages as required to implement the needs of users within their respective subunit in the organization. This customization process can continue for a number of levels until end users are reached. End users may also customize pages, based on the permissions assigned to them by administrators.

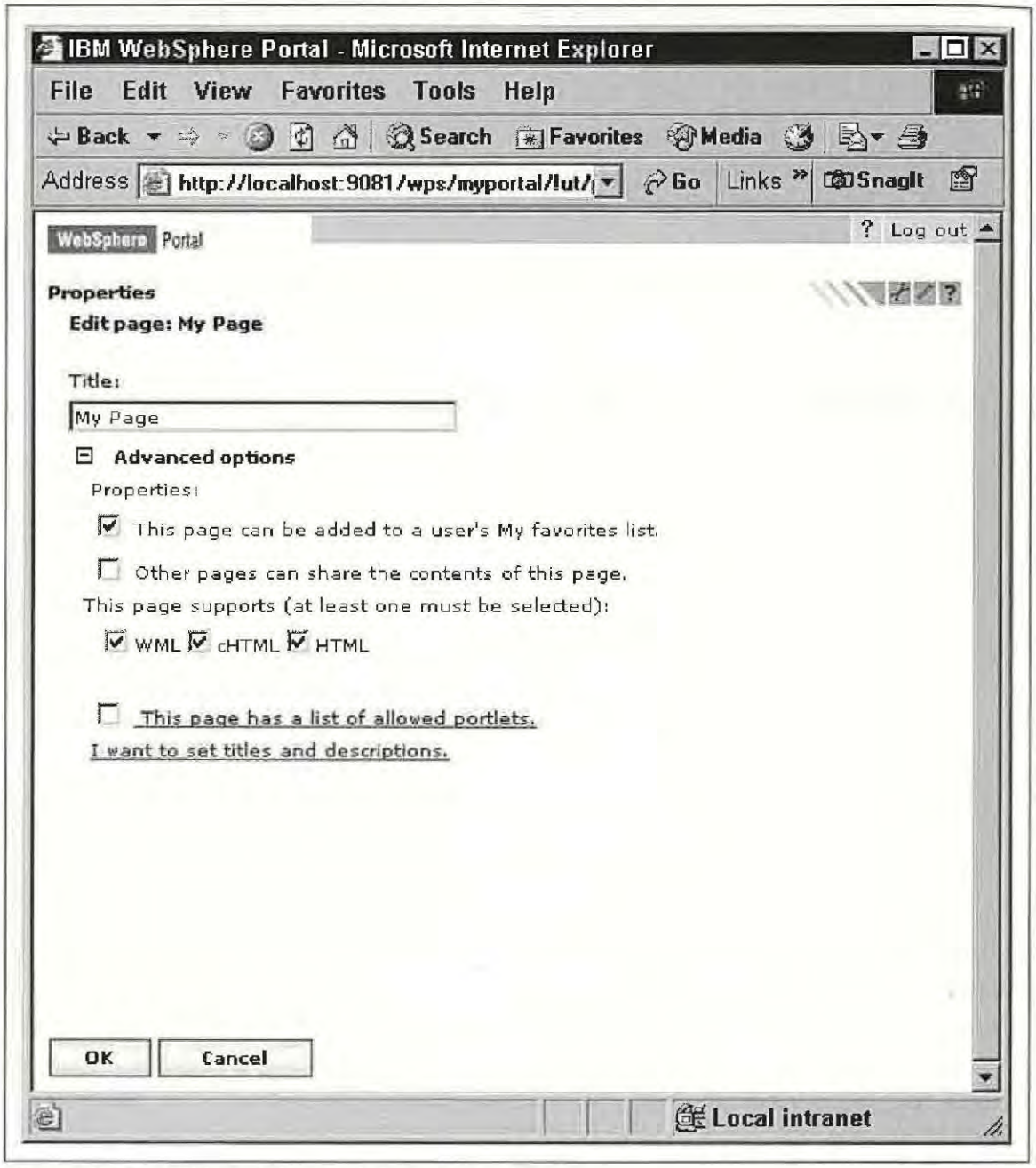


Figure 7-10 Editing Advanced Page properties.

Advanced Layout Editing

WebSphere Portal allows six different page basic layouts. While these might cover most simple cases, a real portal usually aggregates a variety of information sources, and fine-tuned control of layout is needed. To this end, WebSphere Portal provides *layout tools*. Layout tools are invisible by default. To enable them (for example, in our new page created earlier) follow these steps:

1. Go to Manage Pages under Administration.
2. Navigate to the Sample Page under My Stuff.

3. Click Edit Page Layout (second icon from the left following the page name).
4. Click the Configure button in the top-right corner of the window (that is, the leftmost button in the title bar).
5. Check the box reading Show Toggle Link for Show Layout Tools/Hide Layout Tools.
6. Click Done.

In the same configuration window you may also select whether users may modify the page layout, and if so what layouts (out of the six default ones) are allowed.

Once you have the Show Layout Tools link available in the layout editing window, you can click it. Once you do this, the six layout patterns are no longer visible and you may customize your layout by adding as many containers as you want. For each column container you may set its width, as well as add row containers within it. You may add column containers within a column container, and of course you may add portlets in both. This allows you to create a complex nested structure, which would provide the exact layout you are interested in.

Locks

In the page customization mode, where we edit the layout, there are two tabs in addition to the Content tab that opens by default. The Appearance tab can be used to customize appearance and will be discussed in the next chapter, and the Locks tab allows the administrator to limit the flexibility of a user in editing the page layout and content. For each container (row or column) you may lock or unlock the container and the content of the container. Locking the container prevents users from removing this container from a page, whereas locking the container content prevents users from adding or removing portlets and subcontainers from the container. You may also lock specific portlets, preventing the user from removing them.

Locks are applicable to shared pages, and allow top-level administrators in cascading portals to define content that is mandatory in a page, as lower-level administrators and end users cannot override the decisions of top-level administrators.

Removing Nodes

Removing nodes is a straightforward process. You just navigate in Manage Pages to a location you can see the node (to do this you navigate to the node or to its parent), and click the Delete button.

Note that removing a node implies removing all children nodes of that node, so a delete operation can have far-reaching consequences (just like deleting a folder in a file system). Furthermore, removing a shared page removes all reference pages.

Creating a New Portal

As previously mentioned, there is no built-in notion of portal within WebSphere Portal. Usually you can think of a portal within WebSphere Portal as a high-level label, available from the topmost navigation bar. In this section we will guide you through the process of addition of a new such label. This process is essentially identical to the process we went through earlier in this chapter, but it will allow us to define a structure we will use in the following chapters.

In order to define a new label, follow these steps:

1. Enter the Administration section.
2. Select Manage Pages.
3. Make sure you are viewing Content Root.
4. Click the New Label button.
5. Enter **New Portal** in the Title field.
6. Select OK.
7. Select OK in the confirmation window shown next.

To add a new label under this portal, follow these steps:

1. Select the New Portal entry.
2. Click the New Label button.
3. Enter **Company** in the Title field.
4. Select OK.

Now, follow the same steps to add two other labels named "Work" and "Personal." The following sequence of steps places some pages within these labels. We start with the Company page:

1. Select the New Portal entry.
2. Select the Company entry.
3. Click the New Page button.
4. Enter **Welcome** in the Title field.
5. Select OK.

Similarly, you add **Transactions** under Company; **Mail**, **Calendar**, and **Contacts** under Work, and **My Page** under Personal. Next we proceed to putting some content into these pages:

1. Select the new Welcome page.
2. Click the Edit Layout button.
3. Click the One Column button.
4. Click the Add Portlet button.
5. Select Interesting Links.
6. Select Breaking News.
7. Select Welcome to Your Co.
8. Select OK.
9. Click twice the Move Portlet Down button on Interesting Links.
10. Click the Move Portlet Up button on Welcome to Your Co.
11. Select OK.

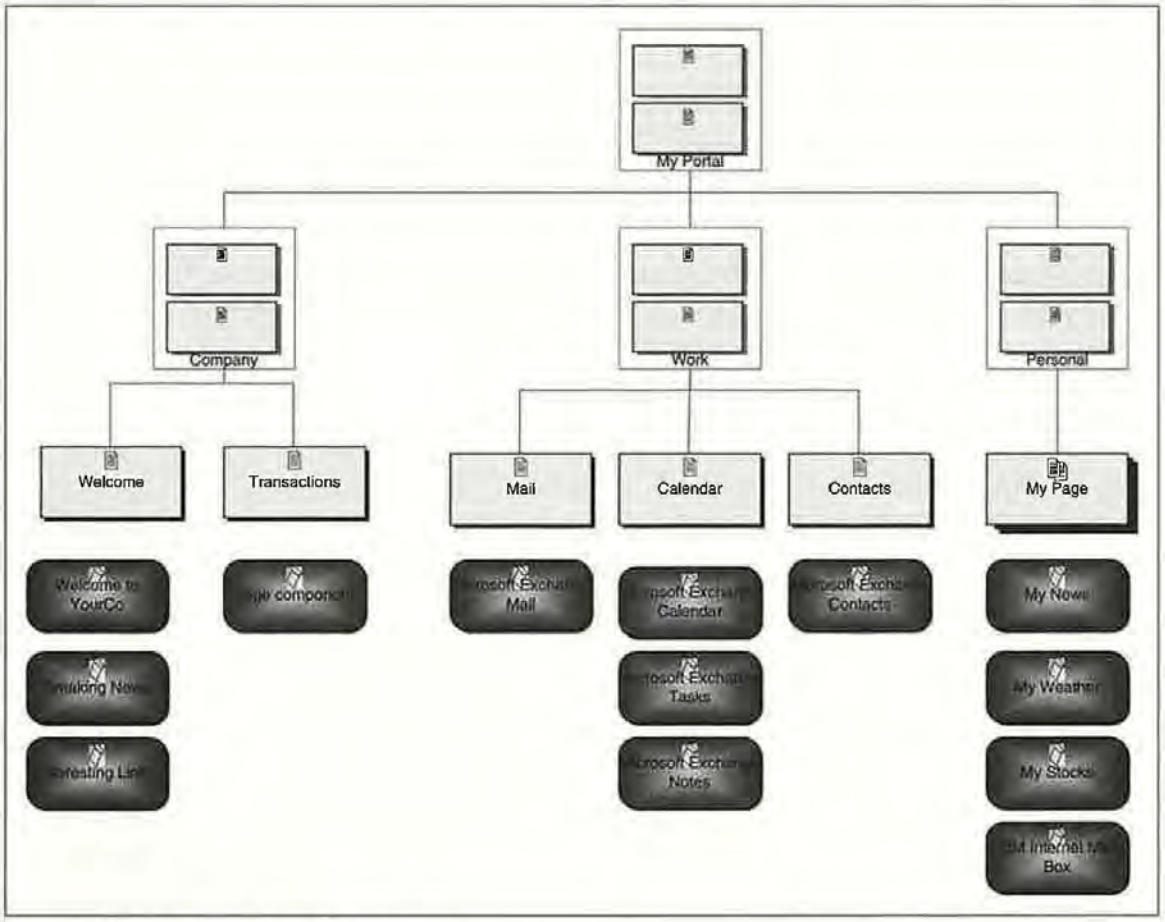


Figure 7-11 New Portal structure.

Note that after choosing the portlets to be placed within the page, they were ordered according to their ordering within the list of available portlets. In steps 9 and 10 we customize the ordering.

To complete the construction of our portal, we need to place content in all pages. To do this you need to follow the same sequence of steps choosing different portlets for each page. Figure 7-11 summarizes the structure and content of this portal.

Once you have finished the other five pages finished, you might want to see how this new portal looks like. Unfortunately, the default behavior of WebSphere Portal prevents users from accessing new label New Portal, as the set of top-level labels is fixed. In the next chapter we will see how to make this portal accessible, and we will continue the process of customizing it.

Summary

In this chapter we saw how the structure of a portal is defined and modified. We went through the process of modifying the default preinstalled portal, as well as defining our own new portal. This chapter provided a glimpse into the ease with which WebSphere Portal is configured and customized. At this point you should be able to define new portals and pages, and to place portlets within pages.

It goes without saying that controlling portal structure is merely the first step in customizing your portal. An important aspect of your portal design that has a strong impact on usability and user adoption is the look and feel of the portal. In the next chapter we will expand on the topic of portal customization, going into the visual (or Web design) aspects. Later in the book you will learn how to develop your own portlets.

CHAPTER

8

Tailoring the Portal Web Design

In the previous chapter we described the process of customizing the structure and content of WebSphere Portal. In this chapter we focus on visual aspects of the portal. The visual aspects of your portal have a significant impact on the quality of user experience, and thus indirectly on the success of your portal. You would like the look and feel of your portal to be appealing, and usually you would like it to be unique. In this chapter you will learn how to customize the visual appearance of your portal. For the most part, such customization requires at least basic JSP programming skills.

Themes and Skins

A WebSphere Portal theme is an abstraction that captures the look and feel of a Web site. The theme is used to determine the global visual appearance of a page. Thus the choice of theme affects the navigational structure, logos, banners, fonts, and colors. For example, the theme determines whether navigation is performed using a toolbar on top of the page or a menu on the side of the page. It also determines the colors and fonts used.

Skins control the look and feel of borders around components. Skins can be either independent or associated with themes. For example, you may have a skin for which portlets have no borders, or another skin where borders are rectangular with rounded corners.

It is important to realize that both themes and skins are not necessarily fixed throughout the portal. It is possible to have a page of a set of pages with one theme and other pages with another theme. Similarly, portlets even within a single page can have different skins.

Similar to other aspects of WebSphere Portal customization, theme and skin customization can take place at three different levels:

- Portal Administrator. Controls user access permissions, and defines the extent to which a user may customize pages in the portal
- Web Designer. Controls visual aspects usually going into the level of HTML
- End user. Controls content and appearance as much as is permitted by the administrator

Default Themes and Skins

One of the easiest ways of understanding how something is built is to look at some existing examples. The themes and skins available to WebSphere Portal are stored in the directory `<WAS ROOT>/installedApps/<HOSTNAME>/wps.ear/wps.war`. In this location you can find two directories named `themes` and `skins`. On a Windows machine called Potter and using the default installation location for WebSphere, you have these directories under `C:\Program Files\WebSphere\AppServer\installedApps\potter\wps.ear\wps.war`. Each of the two directories contains subdirectories for the different markup types supported. By default HTML, cHTML, and WML are supported, and each has a separate directory. However, in the default configuration WebSphere Portal supports themes only for HTML. This makes perfect sense, as cHTML and WML are used to deliver text-based content to cell phones. In the `HTML` directory, the JSP page `Default.jsp` is used as the entry point for the default theme. Additional themes have their own subdirectories under the directory `HTML`:

- Corporate
- Engineering
- Finance
- Science

Each of these directories has its own `Default.jsp` page, and in addition it contains CSS style sheets and images. The default style sheet in each theme is called `Styles.css`, and the default version of this file is directly under the `HTML` directory. Images are used as much as they would in the context of ordinary HTML content, and for example the file `Banner.jpg` holds the banner shown on top of each page. Figure 8-1 shows the default theme as well as the other four preinstalled themes. Note that the file `preview.gif` in the directory corresponding to each theme shows a downsized version of a page rendered with the theme.

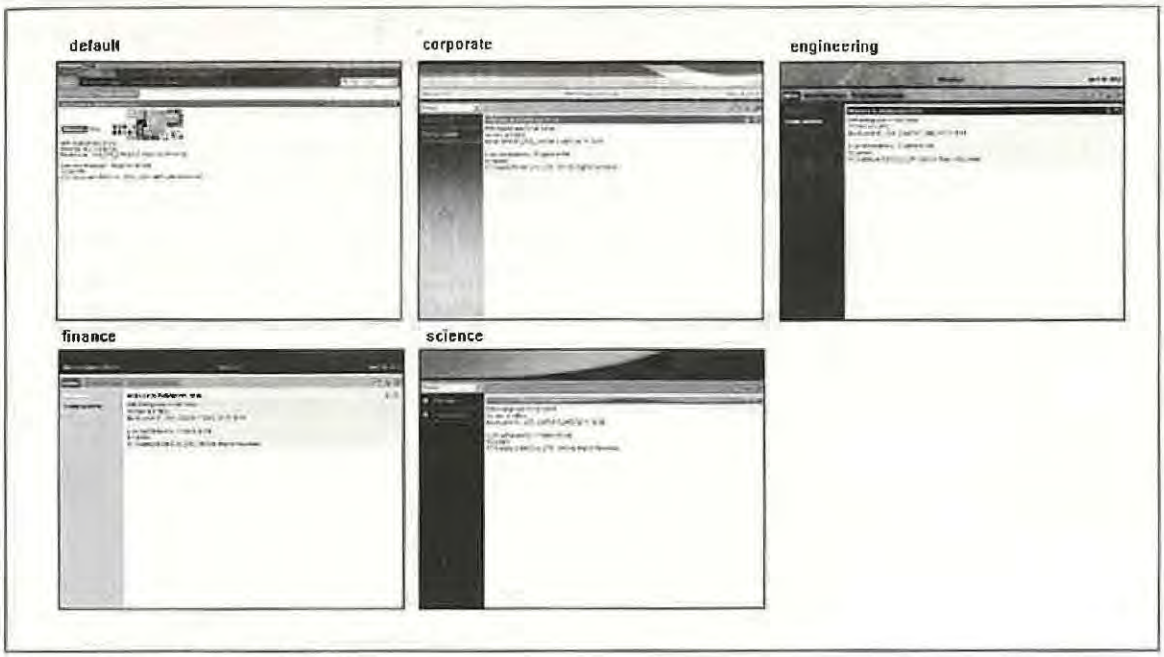


Figure 8-1 Preinstalled themes.

At this point you might wonder why WebSphere Portal organizes the portal resources in such a complicated way, where the default theme is defined in one directory and all other themes are defined in subdirectories. This organization stems from the fact that the portal provides fully dynamic aggregation of information, depending on locale, client, theme, and markup. In order words, whenever a certain resource is needed (be it a JSP, image, or CSS style sheet), the portal looks for it based on the directory structure. For example, when looking for the `Banner.jpg` file, the directory of the current theme is searched, and if nothing is found, the same file in the `HTML` directory is used. More generally, each research is sought for by traversing a set of directories, starting from the most specific location and moving higher in the hierarchy. The location is based not only on theme, but also on locale, client, and markup. For example, consider the file `Styles.css` when you are in the `Engineering` theme, using an Internet Explorer client using an `en` (English) locale. The search sequence would be as follows:

- `/themes/html/Engineering/ie/en/Styles.css`
- `/themes/html/Engineering/ie/Styles.css`
- `/themes/html/Engineering/en/Styles.css`
- `/themes/html/ie/en/Styles.css`
- `/themes/html/ie/Styles.css`
- `/themes/html/Styles.css`
- `/themes/Styles.css`

While this method of organizing resources might be confusing at first due to its complexity, it is quite powerful in allowing you to adapt style and content to different users. The search sequence starts with the most specific definition to the most general and stops when the needed resource is found. The way resources are sought allows you to minimize duplication of resources. By placing a resource at a high level of the hierarchy, you make it available for all locales and clients, and you can make customized copies of it when needed. It goes without saying that if a resource is nowhere to be found, the resulting HTML content might be erroneous. To this end, it is important to keep a default copy of important resources in the root directory.

Similar to themes, there is a directory `skins` under the `wps.war` directory. The default skins are:

- Album
- Clear
- Corner
- Diamonds
- Fade
- Hint
- Noborder
- Noskin
- Outline
- Pinstripe
- Shadow
- Wave

The visual effects of the various skins are shown in Figure 8-2. The differentiation between skins is through the look and feel of the title bar and window borders.

Choosing Themes and Skins

In the previous chapter, we saw how the Manage Pages window allows you to browse through the content hierarchy, and edit various properties of pages and labels. One such property is the theme, which may be edited through the Edit Page Properties button. An example of this is shown in Figure 8-3 where the theme for the My Portal label is set. The drop-down menu lists the available themes, as well as Portal Default Theme. For lower-level

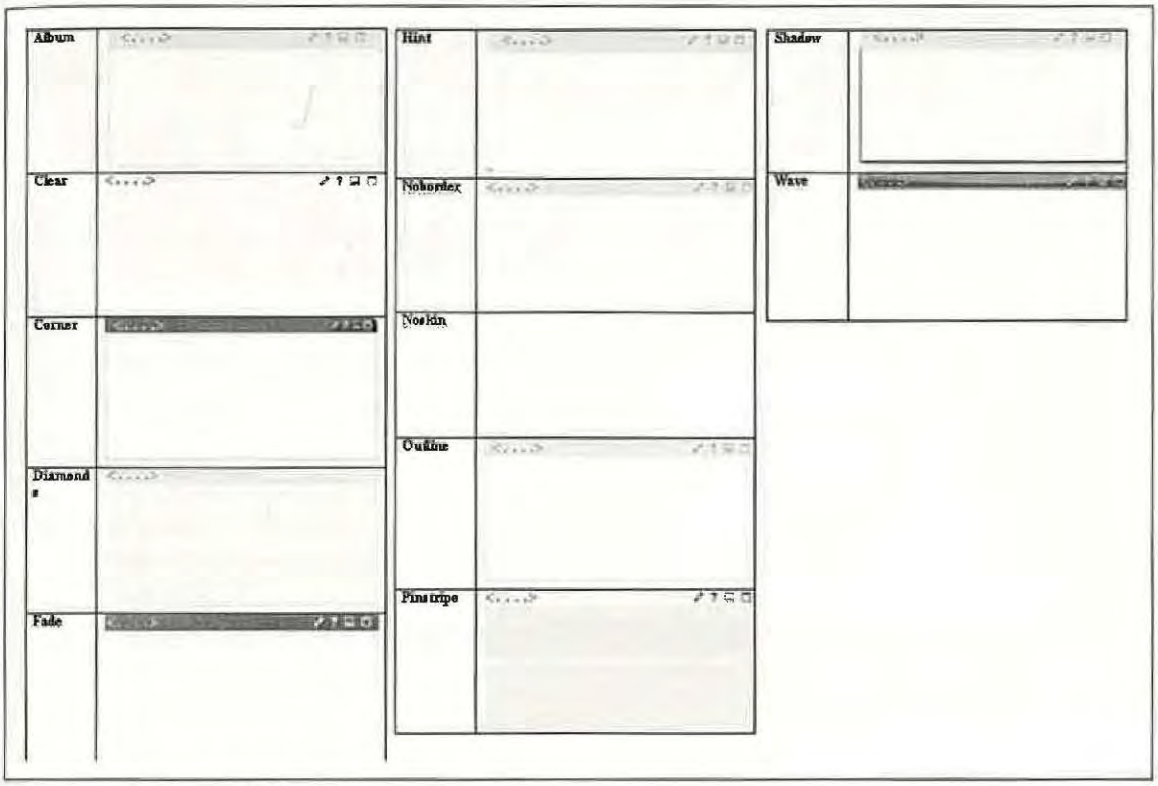


Figure 8-2 Preinstalled skins.

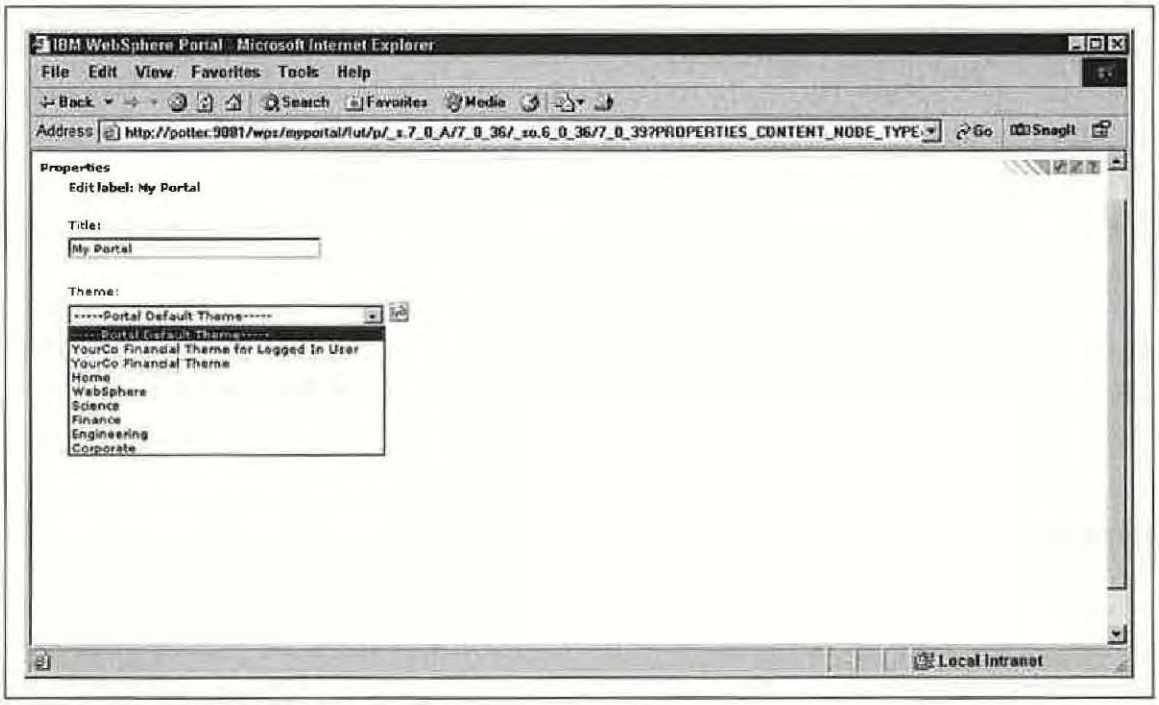


Figure 8-3 Choosing a theme.

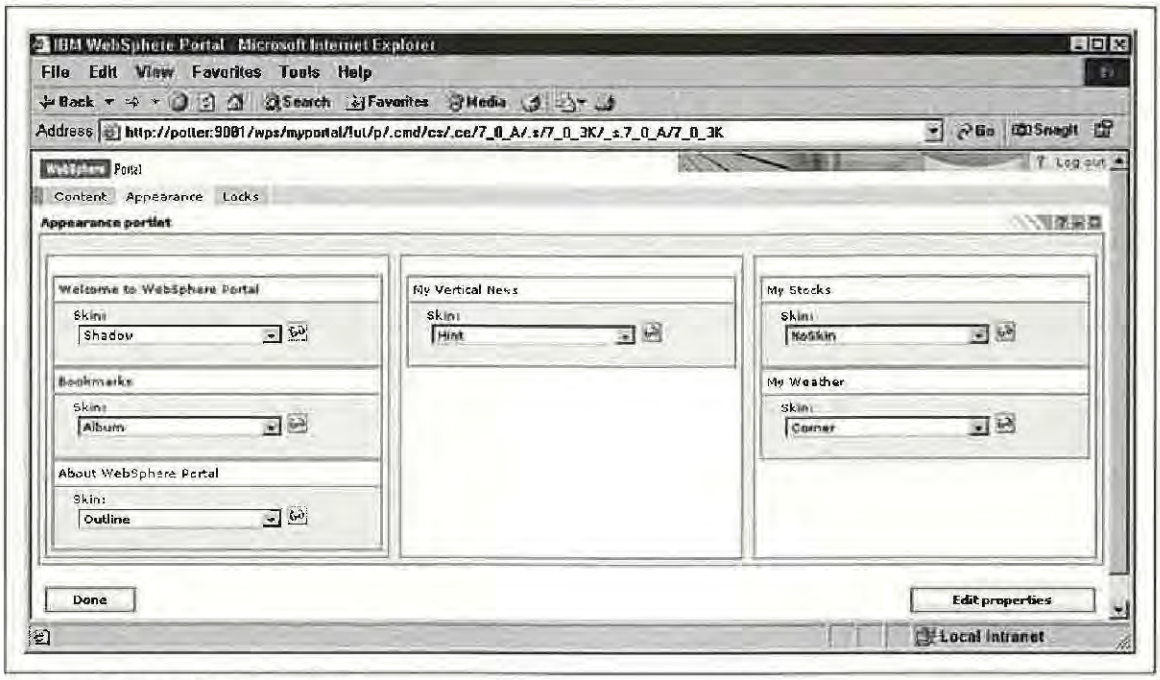


Figure 8-4 Choosing a skin.

nodes in the content hierarchy, the first entry (and the default one) is Inherit Parent Theme. This implements the simplest policy, which implies a uniform look and feel for the whole portal.

Clicking the Appearance bar brings the appearance portlet, which enables the selection of skins. Figure 8-4 shows this window for the default welcome window used when WebSphere Portal is first installed. Note that there are six different portlets, and each one may have its own skin.

End users can customize themes according to their own preference, as long as they have permission to do this. End-user editing is made through the page itself using the Edit Page link when available.

The easiest way to see the effects of different skins would be to choose six different skins for each of the portlets here, and going back to My Portal to see the visual effect. Figure 8-5 shows the Welcome portlet with a `Hint` skin.

Screens

In addition to themes and skins, WebSphere Portal defines a notion of screens. A screen is defined as a template for a page of portal content. Don't confuse this definition of a screen with that of a portal page, which is the end result of a process involving the screen. A portal page that is visible to the user is the product of the aggregation of several information sources, which include the portal page (which is part of the portal hierarchy as discussed in the previous chapter), page theme, skins, and the screen.



Figure 8-5 Welcome portlet with Hint skin.

The default screen is the Home screen, located in the file `Home.jsp`. This file is located in the directory `screens\html` under the aforementioned `wps.war` directory.

A portal page is only loaded when the tag `<wps:screenRender/>` is found in the page. This tag is a WebSphere Portal Server JSP tag, as implied by the `wps:` prefix. The Portal Server defines dozens of tags to support the customization of your portal, and we will cover most of them in the remainder of this chapter. Literally, the `screenRender` tag indicates the location where the current screen is to be rendered. This specific tag can only be used once within a portal JSP. We will see an example of its use in a short while.

Switching to a different screen is usually done through toolbar links. This is done using a special tag `wps:url`. This tag generates a URL pointing into a page in the portal. This tag can use the attributes `screen`, `home`, or `command`. These attributes specify respectively a screen name, the home page (using the value `public` for non-logged-in, and `protected` for logged-in users), or a specific command (using one of the attribute values `LoginUser`, `LogoutUser`, and `ShowTools`).

Defining Your Own Themes

Unless you are only exploring WebSphere Portal server or using it to build very small implementations, it is almost certain that you will want to create your own themes and possibly your own skins. While IBM made an effort

to include a variety of themes and skins, to allow you to use the portal out of the box, in reality you would need to conduct some further customization. As with any Web site or Web application, you need to fine-tune the visual appearance to make it appealing to users and to ensure that it follows the standards used throughout your organization.

Creating a New Theme

To create a new theme, you need to go back to the directory `<WAS ROOT>/installedApps/<HOSTNAME>/wps.ear/wps.war` where themes and skins are located. Here, you create a new directory with a new name corresponding to the theme name. For example, to add a theme called `NewTheme` you need to create a directory with this name.

Once the directory is created, you need to populate it with the various resources needed, for example, JSP, images, and CSS style sheets. One way to go about this is to just copy an existing theme, for example, the `Corporate` theme.

Inside the theme directory, the most important JSP is the previously mentioned `Default.jsp`, which is the entry point for the theme. We have also discussed the CSS style sheet `Styles.css`. A lot of small image files are used for the various buttons (for example, `go.gif`, `help.gif`) and you may customize each one. The banner is stored in `Banner.jpg`, and the file `navfade.jpg` provides the background for the navigation bar on the left side of a page. As starters, we will change the banner. To do this, you need to replace the file `Banner.jpg` with your own file. For example, we will use the banner shown in Figure 8-6.

At this point you have your `New Theme` ready to be used, and we need to make WebSphere Portal aware of it. To do this, you need to perform the following sequence of operations:

1. Click the Administration button on the main toolbar.
2. Click Portal User Interface.
3. Click Themes and Skins.
4. You will now see the window as shown in Figure 8-7; click Add New Theme.



Figure 8-6 New `Banner.jpg`.

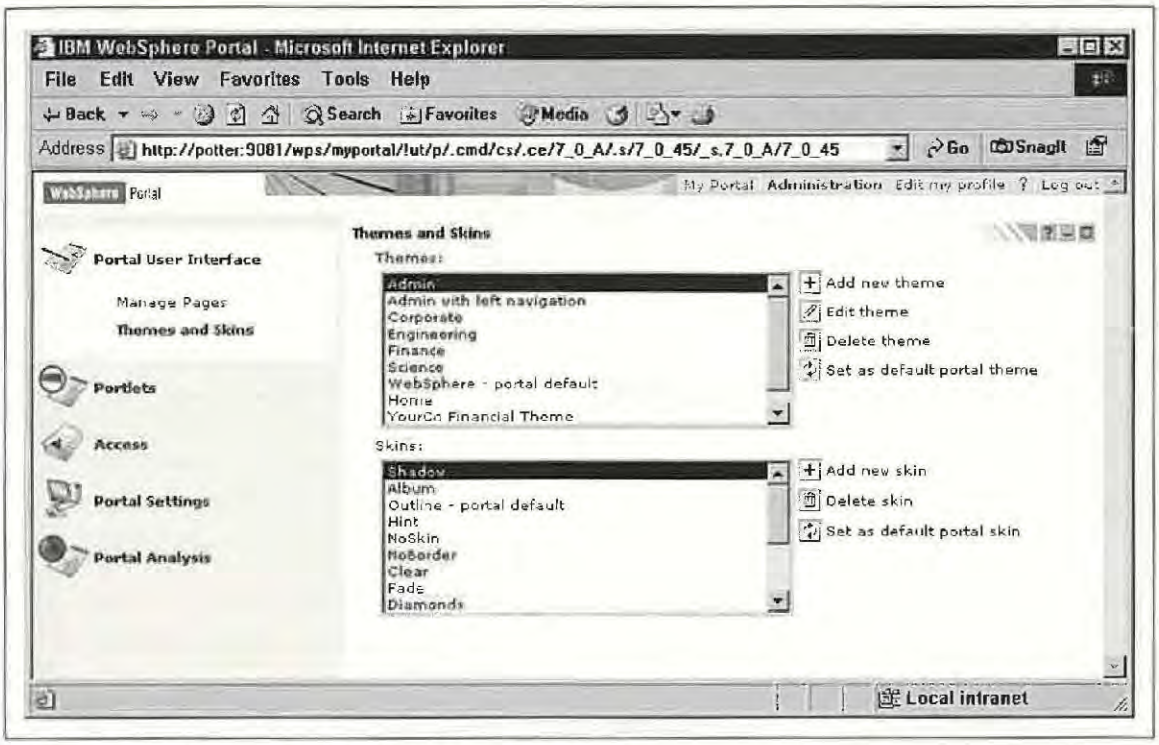


Figure 8-7 Themes and skins management.

A new window will be shown (see Figure 8-8). Here you need to follow these steps:

1. Enter the theme name, for example, **New Theme**.
2. Enter the theme directory name, in our case `New Theme` (note this is a relative path).
3. Choose which themes may be used in this theme.
4. You may click `Set locale-specific` titles to choose local-specific names for this theme.
5. Select OK.

Finally, you can see the new theme (and specifically the new banner) just by returning to `My Portal`.

In the window shown in Figure 8-7 you may also remove a theme or edit a theme. Removing the theme is important whenever you accidentally create an erroneous theme, as leaving it in the portal might allow users to apply it. Theme editing is quite limited, and allows you to control its local-specific naming and allowed skins.

It is usually a good idea to prepare a preview of your theme, and store it in the `preview.gif` file in the theme directory. This will allow users (or other administrators) to associate the name with the theme.

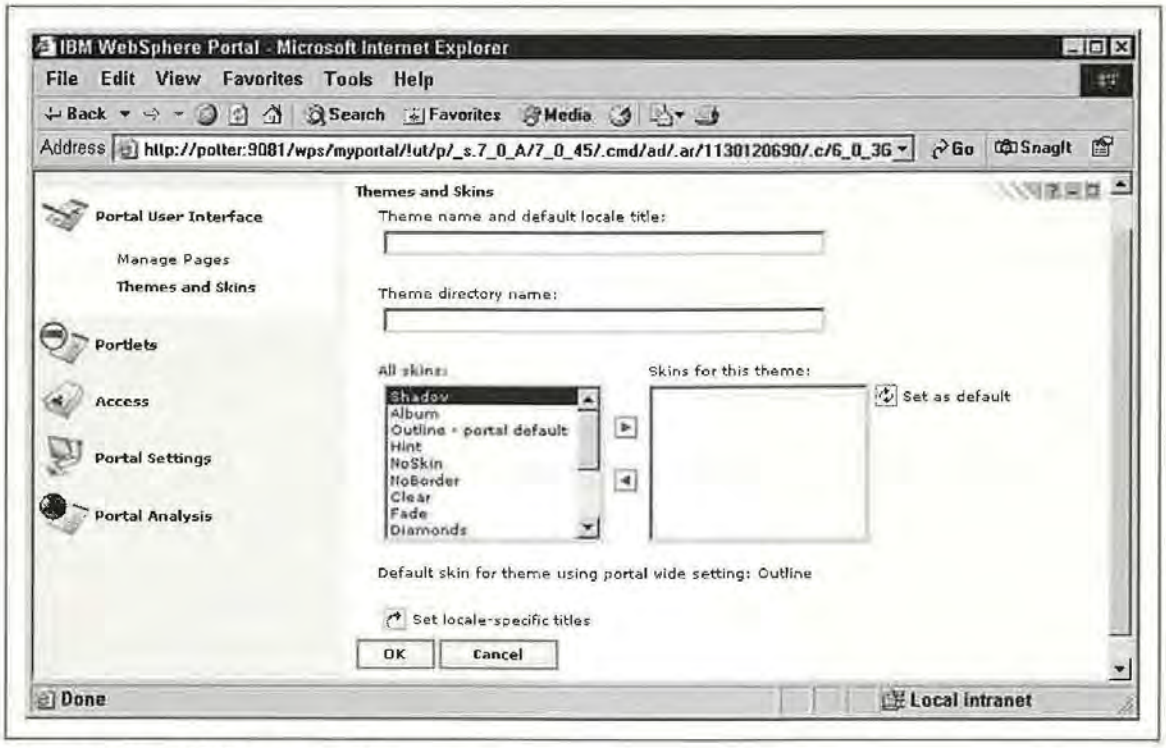


Figure 8-8 Adding a new theme.

Figure 8-9 shows the portal main page following the banner change. Notice that the Portal Server automatically concatenated several copies of the image provided to fill the full width of the page.

The Anatomy of a Theme

A theme in WebSphere Portal Server is implemented via a set of JSPs. We have mentioned the `Default.jsp` page, which serves as the entry point for the generation of content. This page includes other JSPs, which are part



Figure 8-9 New banner within portal page.

of the theme definition. If a theme does not include any of these files, the default version of the JSP is used.

The other JSPs are as follows:

- `AdminToolBarInclude.jsp`. Displays links used by the Administrator for customizing pages and creating new pages:
  - Edit Page
  - Assign Permissions
  - New Page
- `ToolBarInclude.jsp`. Displays the toolbar available on the top of the page, offering options such as Sign Up or Log in when the user is not logged in, and options such as My Portal or Administration when the user is logged in
- `Head.jsp`. Displays the text in the `<HEAD>` element of the generated HTML page
- `PlaceBarInclude.jsp`. Displays a second level navigation toolbar
- `PageBarInclude.jsp`. Displays a third level navigation toolbar

In addition to JSP file, a theme directory contains other files used as building blocks for JSPs, such as GIF image files, and CSS style sheets.

Customizing Themes

In the previous section we added a new theme, but in fact we did not change much in terms of layout or functionality.

One of the first things you would like to change is the company name and company logo used in the portal. There are two ways to go about doing this. One way is to go to a file called `engine.properties` located in `<WPS Root>/shared/app/nls`, and edit the resource called title. Alternatively, you can modify this in the file `Default.jsp` of your theme. In order to do this, you must have some basic acquaintance with JSPs, which we will assume you have (if not, you can skip to the next section).

Given below is the default content of the file `Default.jsp`. This is a relatively short, albeit not very readable JSP. This JSP determines the general layout of pages in the portal, using an HTML table construct. The most notable feature in this page is that it uses tags, which are specific to WebSphere Portal. As mentioned above, these tags are identified by the `wps:` prefix.

The file `Default.jsp` starts with some standard headers.

```
<%@ page session="false" buffer="none" %>
<%-- Must be Transitional DTD or Mozilla/Netscape 6 will not correctly
display tables used for visual layout. --%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<%-- @copyright jsp --%>
```

We then include the `engine.tld` taglib, with the prefix `wps`, as well as some additional tag libraries.

```
<%@ taglib uri="/WEB-INF/tld/engine.tld" prefix="wps" %>
<%@ include file="./extension/TagLibInclude.jsp" %>
<%-- This includes Extend/Express specific tag libraries. --%>
```

Next comes a `<wps:constants/>` tag. This tag makes the two constants `wpsBaseURL` and `wpsDocURL` available in the remainder of the page. Later this page uses the former, which indicates the base URL for WebSphere Portal, e.g. `http://localhost:9081/wps/`.

```
<wps:constants/>
```

The HTML page starts here, including the file `Head.jsp`. This file defines the HTML page header using a `<head>` tag. The `Head.jsp` page is of course customizable, and in fact it is taken from the theme (that is, you may define a different implementation for different user groups). By default, the header reads "IBM WebSphere Portal."

```
<html>
<%@ include file="./Head.jsp" %>
```

Next starts the page body, with some browser-specific directives.

```
<%-- Set margins to 0 here in addition to stylesheet to support old
versions of Netscape --%>
<body marginwidth="0" marginheight="0" <%=bidiDirAttr%>
onLoad="if ( document.forms['wpsPageGroupSelectionForm'] != null )
 document.forms['wpsPageGroupSelectionForm'].reset();" >
```

At this point, immediately after the `<body>` tag, the file `PageBeginInclude.jsp` is included. The content of this file is dependent on the version of WebSphere Portal Server used, but it is typically empty.

```
<%@ include file="./extension/PageBeginInclude.jsp" %>
 <%-- This includes Extend/Express specific page logic. --%>
```

Finally we get to add the table, which is used to ensure a uniform width for the components of the page. You can notice the use of the tag `wps:text`, for example `<wps:text key="title" bundle="nls.engine">`. This returns the text for the key `title` in the locale of the user. Replacing this with a hard-coded title will be equivalent to changing the `engine` file residing in the directory `shared\apps\nls` under the Portal Server root, or one of the locale-specific variants of it. Obviously, to have a locale-specific title it might be easier to stick to the latter solution.

```
<%-- This table is here so the banner is the same width as the portlets
--%>
<table border="0" cellpadding="0" cellspacing="0" width="100%"
 height="100%">
```

```
<tr> <!-- logo -->
<td valign="top" align="<%=bidiAlignLeft%>" nowrap>
<img align="absmiddle" alt='<wps:text key="title"
 bundle="nls.engine">Portal Title</wps:text>'
 title='<wps:text key="title" bundle="nls.engine"/>'
 src='<wps:urlFindInTheme file="logo.gif"/>'>
<a href="#wpsMainContent"><img width="1" height="1" border="0"
 src='<%= wpsBaseURL %>/images/dot.gif'
 alt='<wps:text key="link.skiptocontent" bundle="nls.engine"/>'
 title='<wps:text key="link.skiptocontent" bundle="nls.engine"/>'>
</a>
</td>
<td width="100%" height="1" valign="top" align="<%=bidiAlignRight%>">
```

We include the file `ToolbarInclude.jsp` that controls the top-level navigation, allowing you to switch between `My Portal` and `Administration`. This would be the page to modify if you want to include another visible node directly under the `ContentRoot`. Following it, another toolbar `AdminLinkBarInclude.jsp` is loaded based on the tag `<wps:if portletSolo="no">`. This is a standard condition statement and several conditions can be checked with its attributes. WebSphere Portal provides a multitude of attributes, which can be checked, such as the locale, whether a user is logged in or not, or client capabilities. Here only one condition is checked and that is `portletSolo` is set to no, i.e. the portlet is not running solo state. Portlets are running in one of the following four states: `normal`, `maximized`, `minimized`, and `solo`. There is another possible condition attribute for testing the portlet state, called `portletState`. Thus the condition `portletSolo="no"` is equivalent to `portletMode="solo"`. The `wps:if` below is used to enclose an include statement for adding the navigation JSP, which we will discussed later.

```
        <%@ include file="./ToolBarInclude.jsp" %>
          <%-- Don't show navigation in solo mode --%>
               <wps:if portletSolo="no">
                       <%@ include file="./AdminLinkBarInclude.jsp"%>
               </wps:if>

    </td>
<tr>
     <td colspan="2">
```

If navigation is to be used, include `PlaceBarInclude.jsp`, providing the next level of navigation below the top-level one (for example, within `My Portal`, between Welcome and the other options), and `PageBarInclude.jsp` provides the next level of navigation. Obviously, by removing the JSP include statements for either of these files, you can remove this level of navigation altogether from a specific theme.

These pages provide specific implementation of a general concept in WebSphere Portal, which is navigation. Using the <wps:navigation> tag you can define your own navigation menus, which are not necessarily limited to two or three levels down.

```
                <%-- Don't show navigation in solo mode --%>
                    <wps:if portletSolo="no">
                    <%@ include file="./PlaceBarInclude.jsp" %>
                    <%@ include file="./PageBarInclude.jsp" %>
                </wps:if>
        </td>
</tr>
<tr>
<td width="100%" height="100%" valign="top" colspan="2">
        <a name="wpsMainContent"></a>
        <wps:screenRender/>
</td>
</tr>
</table>
```

We conclude with a product-specific `PageEndInclude.jsp`.

```
<%@ include file="./extension/PageEndInclude.jsp" %>  <%-- This
includes Extend/Express specific page logic. --%>
</body>
</html>
```

An important caveat to note about included files is that if you change them, you need to make the Portal Server aware of this by modifying the `Default.jsp` command (for example, using the UNIX `touch` command).

Using the WPS tags, you may create intricate logic to customize your portal. WebSphere Portal provided dozens of other tags, which are available. These tags are described in the next section.

Portal JSP Tags

In the previous section you saw how the tags `wps:constants`, `wps:text`, `wps:screenRender`, `wps:navigation`, and `wps:if` are used.

The Portal Server provides many more JSP tags. The `taglib` file for these tags is called `engine.tld` and is located in the `<WPS ROOT>/shared\app\WEB-INF\tld` directory. The main purpose of these JSP tags is to allow the JSP designer to leverage some of the information and capabilities available inside the Portal Server engine.

The remainder of this section describes some of the more commonly used JSP tags provided by the Portal Server. A full list can be found in the IBM product documentation.

The most commonly used portal JSP tag is `wps:if`. In the `Default.jsp` listing above you saw how to use it with the `portletSolo` attribute, which checks whether the portlet is in solo state. There are a multitude of other conditions that can be checked:

- `capableof="capability"`. Tests whether the client has a specific capability (for example, HTML_4_0, HTML_FRAME).
- `locale="locale"`. Tests for a specific locale (for example, in_US). The `notLocale="locale"` attribute checks if the locale is different than the one specified. The `notLocale` form is required as the syntax of the `wps:if` tag does not allow for fully blown logical expressions.
- `loggedIn="yes|no"`. Tests whether the user is logged in. This condition is used extensively in `Toolbarinclude.jsp`, discussed in the next section.
- `navigationAvailable="yes|no"`. Tests the availability of navigation.
- `newWindow="yes|no"`. Tests whether portlet is running in a separate browser window.
- `screen="screen"`. Checks if the current screen is the one specified. There is also a `notScreen` version available.
- `pageAvailableNext="yes|no"`. Checks if there is a following page available. A similar tag `pageAvailablePrevious` checks for the availability of a previous page. Both of these are used in `Banner.jsp` to add a scroll icon allowing the user to access additional tabs if all of them are not visible concurrently.
- `pageCompletelyActive="yes|no"`. Tests if the current page and its parents are all active.
- `pageBookmarkable="true|false"`. Tests if a page can be bookmarked (as per the definitions in the administrator's `Manage Pages`). This condition can be used to enclose a bookmark link.
- `portletMaximed="yes|no"`. Tests if the portlet state is maximized.
- `portletMode="edit|view|configure|help"`. Tests what modes the portlet supports. This is important for the title bar display in customized skins.
- `showTools="yes|no"`. Tests whether additional controls should be placed in the portlet title bar (usually, clicking the wrench icon in the top-level toolbar sets this to **yes**).

- `resumeLevel="0|1|2"`. Tests for the setting of the level of session preservation as set by the administrator (0 means no persistent session state, 1 means portlet states and modes are stored, and 2 means all portlet settings are stored).
- `resumeOption="0|1"` . Tests whether users are being offered to restore their most recent session upon login. 0 means no, 1 means yes.

In addition to the `wps:if` tag, there is a negative `wps:unless` tag. With this tag, if any of the conditions is true, the following text is not written to the output page.

In order that you are able to access files and resources, the Portal Server provides the tags `wps:urlFind`, `wps:urlFindInSkin`, and `wps:urlFindInTheme`. All three have a `file="`*`filename`*`"` attribute. The `urlFind` also has `path` and `root` attributes to specify where to look for a file, and the other two just have an `id` attribute, which allows you to place the URL into a scripting variable instead of to the output stream. All forms of `urlFind` use the hierarchical search sequence described earlier, and thus depend on the locale and markup of the client.

While `urlFind` is sufficient for accessing static resources, you need to use `wps:urlGeneration` to generate URLs for portal pages. This tag places the generated URL into a variable `wpsURL` and has attributes called `root`, `navigationNode`, `contentNode`, and `compositionNode` used to identify the node where the page or portlet is located. Other attributes available with this tag are as follows:

- `portletWindowState="maximized|solo|normal"`. Indicates the state of the portlet window when it is displayed. This is only applicable to portlets.
- `pacCheck="CreatePage|EditLayout|DeletePage|AssignRoles|NoCheck"`. Indicates what kinds of permissions are required from the users and should be checked. If `NoCheck` is specified, no checks are made.
- `useReqID="true|false"`. This attribute determines whether a request ID is appended to the URL. By default this is true, thus allowing users access to a new browser window without the links getting invalid.
- `newWindow="true|false"`. Opens a new window or new iFrame.

If you need to pass parameters to a generated URL, `<wps:urlGeneration />` together with `<wps:urlParam/>` can be used. The latter tag has two attributes: `name` specifying the parameter name and `value` specifying the parameter value.

Some other tags are as follows:

- `wps:date`, d `wps:time`. These can be used to insert the date and time in various formats.
- `<wps:user attribute="userID|fullName|familyName">`. Returns an identification of the user based on the attribute, if the user is logged in.
- `<wps:captureContext contextKey="context">`. Captures the content of the tag (that is, whatever is placed between the tag beginning and ending markers) and places as a string attribute in `PageContext`. This is useful when you need to use data, which can only be generated by JSP tags from within JSP Java code.

Adding Top-Level Links

Recall that during our review of the `Default.jsp` page, the file `ToolbarInclude.jsp` was included to control the top-level navigation. In order to enable the "New Portal" entry defined at the end of the previous chapter, you need to understand how navigation works. To this end, let's take a look at `TooBarInclude.jsp`. For brevity, some of the comments were removed, and we only show the code for the first two navigation buttons. The code for each is denoted by a comment.

```
<table border="0" cellspacing="0" cellpadding="0" >
<tr>
<%@ page import="com.ibm.portal.*"%>
<%-- My Portal button --%>
<wps:if loggedIn="yes" portletSolo="no">
      <td class="wpsToolBar" nowrap><b>
      <wps:text key="link.my.portal" bundle="nls.engine"/></b></td>
</wps:if>
<%-- Administration button --%>
<wps:if loggedIn="yes" portletSolo="no">
     <wps:urlGeneration contentNode="wps.Administration"
     portletWindowState="Normal">
     <td class="wpsToolBar" valign="middle" align="<%=bidiAlignRight%>"
      nowrap>
     <a href="<% wpsURL.write(out); %>"
      class="wpsToolBarLink"><wps:text key="link.administration"
      bundle="nls.engine"/></a>
     </td>
     </wps:urlGeneration>
</wps:if>
```

What you can see is that if the user is logged in, and the portlet mode is not `solo`, the JSP provides button linking to My Portal. Notice this link

has no `href` tag, since it is only used within the portal and not from the administration pages.

The Administration button does sport an actual link using an `href` tag obtained by using the `generateURL` tag.

In order to provide buttons for top-level New Portal label, we may add the following and replace the code after the My Portal button with the following:

```
<%-- New Portal button --%>
<wps:if loggedIn="yes" portletSolo="no">
<wps:urlGeneration contentNode="wps.New Portal"
 portletWindowState="Normal">
<td class="wpsToolBar" nowrap>
<a href="<% wpsURL.write(out); %>" class="wpsToolBarLink">
New Portal</a>
</td>
</wps:urlGeneration>
```

Once you implement this change, you have a new button allowing you to navigate to the new top-level page. However, once users go there, they have no way to return. To tackle this difficulty, you must create a different theme for the New Portal label, and implement there `ToolBarInclude.jsp`. The implementation should be such that the New Portal button is boldfaced instead of the My Portal button, but the latter would have a link. Incidentally, this is exactly the reason for having an `Admin` theme preinstalled with the Portal Server. You can notice that when entering the `Admin` mode the Administration button becomes bold and the My Portal button becomes active. To have the New Portal button available from the administration mode, this theme needs to be modified as well.

In some cases, it is desired to use different screens, especially for top-level links. For example, in the preinstalled themes, a different screen is used for logging in or for users who have forgotten their password. The following excerpt from `ToolBarInclude.jsp` shows the code for the "forgot password" button. This code uses the `wps:url` tag with the `screen` attribute.

```
<%-- forgot password button --%>
<wps:if loggedIn="no" notScreen="ForgotPassword">
<td class="wpsToolBar" valign="middle" align="<%=bidiAlignRight%>"
nowrap>
<a class="wpsToolBarLink"
href='<wps:url screen="ForgotPassword" home="public"/>'><wps:text
key="link.password" bundle="nls.engine"/></a>
</td>
</wps:if>
```

Defining Your Own Skins

Skins are less powerful constructs than themes. For this reason they are much easier to generate, but on the flip side it might not be as important to you to change the skins available in WebSphere Portal.

Creating a New Skin

To create a new theme, we have to go back to the directory `<WAS ROOT>/installedApps/<HOSTNAME>/wps.ear/wps.war` where skins are located. As was the case with themes, you need to create a directory under the directory `skins`. In the new directory you need to place the JSP and image resources for your skin. The only mandatory file is `Control.jsp`, a JSP file defining the behavior of the skin. Essentially, this JSP uses several WPS tags to determine the portlet status (for example, minimized, maximized), and to generate the needed HTML appearance, using several images. Each of the preinstalled skins includes several images as needed, for the title bar buttons, the title bar background, and for window borders.

Once you have a new directory, say `New Skin`, ready with your `Control.jsp` file and any images needed, you may follow these steps to install it into WebSphere Portal:

1. Click the Administration button on the main toolbar.
2. Click Portal User Interface.
3. Click Themes and Skins.
4. You will now see the window as shown in Figure 8-6. Click Add New Skin.

A new window will be shown. Here you need to follow these steps:

1. Enter the skin name, for example, **New Skin**.
2. Enter the skin directory name, in our case `New Skin` (note this is a relative path).
3. You may click Set Locale-Specific Titles to choose local-specific names for this theme.
4. Select OK.

To use the skin, you need to add it to your portal default theme as discussed above.

Just like themes, you may edit skins (but only to change their names), and remove them. Also, you can provide a preview of skins in a file `preview.gif` in the skin directory.

Modifying Styles

CSS style sheets are an essential tool in designing Web pages. For this reason, the `Styles.css` file is part of each theme. This file defines a set of styles used in HTML files generated by the portal. You may change some of the font, color, and layout definitions given in the file to control the look and feel of portlets including the IBM administration portlets.

For example, the color in which active links and visited links are shown can be modified by changing the `color` definitions in the following snippet out of the `Styles.css` file:

```
a, .wpsLink, a:active, .wpsLink:active {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-size: x-small;
    color: #3366CC;
}
a:visited, .wpsLink:visited {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-size: x-small;
    color: #666699;
}
```

Summary

In this chapter we learned how to customize the portal Web design. The portal Web design can be customized with themes, skins, and styles. You learned how to create new themes and skins, and how to modify them. In the next chapter we will discuss personalization in WebSphere. Some level of personalization is provided in the Portal Server by allowing users to customize pages with themes and skins. IBM takes personalization to the next level with WebSphere personalization as described in the next chapter.

CHAPTER

9

WebSphere Portal Personalization

The evolution of Web applications in general and Portals in particular places a lot of emphasis on the issue of personalization. Personalization is the process of producing and delivering individually tailored content based on previous information associated with a specific user.

Personalizing your portal in an appropriate way you can have the distinct advantage of making information and tools more accessible to users. As the quantity of Web-based content grows, the importance of filtering information and providing each user with the most relevant content becomes crucial. User loyalty and satisfaction tend to grow as content gets more personalized.

WebSphere Personalization

The need for customizing Web page content is a common thread in many applications. For example, in the e-commerce realm, it is important to provide customers with personalized recommendations and advice.

Generating personalized content from within a Web application is not very difficult. You can plug in a couple of "if" statements into your JSP to achieve some crude form of personalization. Personalization becomes a bit more difficult to tackle when you need to apply complex considerations in the customization of content. Alternatively, you might want to apply certain personalization processes across several applications, and to allow nonprogrammers to manage the personalization process.

WebSphere Portal Server provides some form of personalization through its out-of-the-box functionality as described in the previous chapter. The

portal may allow users to customize the layout and themes for portlets and pages, and different content can be delivered to different users by using access control.

Personalization is essentially a process of matching content to users, based on well-defined attributes of users and of content. IBM WebSphere Personalization provides a framework that facilitates content personalization based on previous user interactions with the system. To this end, it provides a way to track users and analyze their activities, as well as store content in a manner that facilitates matching it against user attributes.

The personalization support in WebSphere allows you to use two different technologies: *rules-based personalization* and *recommendation-based personalization*.

Rules-based personalization relies on having a person define a set of business rules that determine which content is displayed for a given user. Rules-based personalization is tightly integrated into the WebSphere Portal Server.

Recommendation-based personalization uses a technology called collaborative filtering, implemented in a recommendation engine. The recommendation engine uses complex statistical models and other techniques to learn the usage patterns of a Web application or Web site in order to provide personalization without the need to provide explicit rules.

Rules-Based Personalization

As the name suggests, rules-based personalization involved using rules to determine what content to present to a given user. To this end, rules-based personalization requires three kinds of resources: users, contents, and rules to map users to content.

The basis for personalization is collecting and maintaining information regarding users. The collection of all the pieces of information relating to one user is referred to as a user profile. User information can be collected either directly from the user (via a user-profile form) or from other sources, such as tracking the user's online activities. Users can be grouped into categories to facilitate personalization.

Web content consists of the full array of data delivered through the portal, such as JSP, and static HTML or XML pages. It is important to note that on a typical portal page only a portion of the page need be personalized. In other words, some parts of the page are static, and the remainder of the page consists of "spots" where personalized content is to be placed.

Personalization rules allow nonprogrammers to specify what content to show in a "content spot" based on a user profile. For example, consider the

following simplistic illustration of rules in a credit-card company customer portal:

```
When UserType is Platinum_member, Show Platinum News
When UserType is Gold_member, Show Gold News
```

In order for these rules to work, there must be a way to identify whether a member is a platinum card holder or a gold card holder. Furthermore, content should have an indication of whether it belongs to platinum news or gold news.

Resources

In WebSphere personalization, both users and content are considered personalization resources. Each resource has one or more fixed attributes defined by the schema for the resource. For instance, the user data schema may include the name, address, and phone number of a customer visiting your Web site. A schema for Web content includes attributes about the content, indicating to which Web visitors it is relevant. Typically the content itself is accessible via a URL, which will be part of the schema.

There are two ways to go about creating and managing resource in WebSphere personalization. One way is to use the WebSphere Portal user and content wizards in WebSphere Studio Application Developer. The other option is to provide Java classes implementing the resource class APIs.

Clearly, personalization requires having permanent storage of both user and content resources. Resources are stored in a content repository, and are loaded into the Portal Server by the Resource Engine. In WebSphere Portal Server 5.0, the Resource Engine is shared with WebSphere Portal Content Publishing (WPCP). Going forward, WPCP is taken out of the WebSphere Portal Server distribution, and it would be replaced with Aptrix/Lotus Workplace Web Content Management. In addition, it seems likely that at some point IBM will adopt the JSR 170 standard for content management, thus removing the need for the Resource Engine. Content Management in WebSphere Portal is discussed in detail in Chapter 11.

Content Spots

The result of personalization is the delivery of personalized content. This content is concentrated in content spots. Content spots usually do not take up the full space on a page, as some content need not be personalized (for example, a company logo would appear on every page regardless of the individual user). Technically speaking, a content spot is a Java wrapper bean that is placed inside your Web page. This bean is essentially a placeholder for

a rule, and can invoke the rule to display personalized content or perform data updates.

The following JSP code snippet illustrates the use of such beans within a JSP. In this example we assume a bean called HelloContentSpot provides a message, which is based on user classification. To retrieve the content derived from the rule, the method `getRuleContent()` is invoked.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
<TITLE> Sample Personalized Page </TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<jsp:useBean class="HelloContentSpot"
   id="helloContentSpot">
<% helloContentSpot.setRequest(request); %>
</jsp:useBean>
<%= helloContentSpot.getRuleContent()%>
</BODY>
</HTML>
```

Defining Rules

Rules define interactions between your Web site and users. Rules are structured as easy-to-read statements, in order to make rules accessible to non-programmers. WebSphere personalization introduces the concept of campaign to group together a set of rules, which are used to achieve a certain goal over a defined period.

To define new rules, you need to perform the following actions:

1. Go to the Personalization entry under the Content Publishing tab.
2. Select Rules on the left-hand panel.
3. Click the New Folder icon to create a new folder.
4. Name the new folder `Rule`.
5. In the new folder, click the New Rule button to create a new rule.

At this point, you reach the rule definition screen, as shown in Figure 9-1.

Editing Rules

The screen shown in Figure 9-1 allows you to specify the rule name and description, as well as choose the rule type and edit the rule itself. Rule editing is performed in the bottom part of the window. The rule editor provides HTML links to help you identify a variable, and items are color coded to help you recognize your options. For each such link, once you

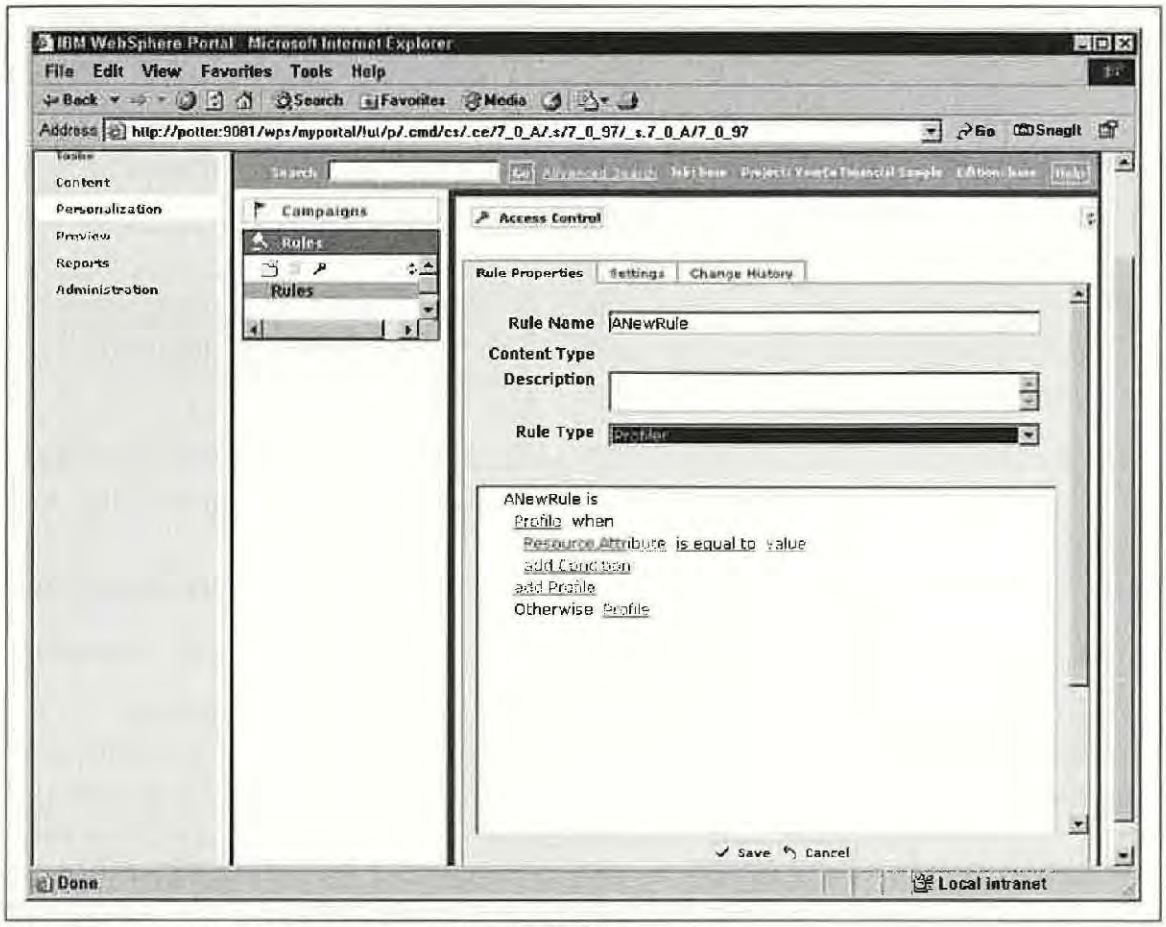


Figure 9-1 Rule definition screen for Profiler rule.

click it a new window opens. In this window you may choose a value out of a list of options. Once a selection is made, it is reflected in the rule text.

The color of links is as follows:

- Blue indicates an item that is completed. It may still be clicked for editing.
- Red indicates an item that must be edited to finish the rule.
- Magenta indicates an item that is optional. It is not required to complete the rule.

WebSphere personalization provides several types of rules, which are as follows:

- Profilers—allow you to define types of visitors based on current user properties and other object properties.
- Select Content—allows you to choose content to be displayed, and also allows to sort the order of returned results.

- Update—allows you to perform updates to data (for example, user profiles).
- E-mail—allows you to send an e-mail.
- Bindings—combine profilers with an action (select, update, or e-mail), so that specific actions can take place under defined conditions.
- Recommend Content—allows you to define rules for content recommendation. This type of rule will be discussed in the next section.

In order to complete editing a rule, you need to click all the blue links. In the example of a profiler rule, you need click the links for Profile, Resource.Attribute, and value.

To enter a new profiling rule, perform the following sequence of actions:

1. Click the Profile link.
2. In the window that opens, enter the desired profile name (see Figure 9-2).
3. Click the Resource.Attribute link.
4. In the window that opens, choose the resource and attribute. For example, Figure 9-3 shows the selection of the current date.
5. Click the value link.
6. In the value window, select the date, as shown in Figure 9-4.
7. Click Save.

Note that prior to saving you could add additional conditions or additional classifications of users (for example, using the else clause).

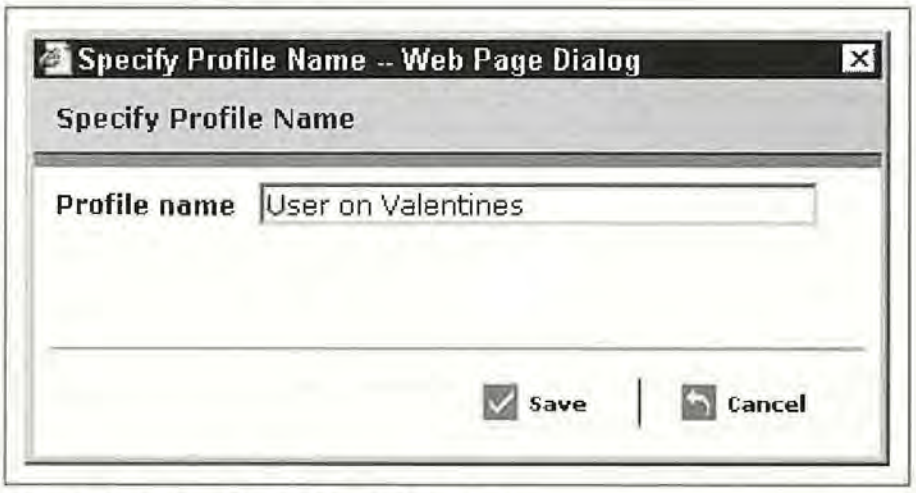


Figure 9-2 Specifying the profile name.

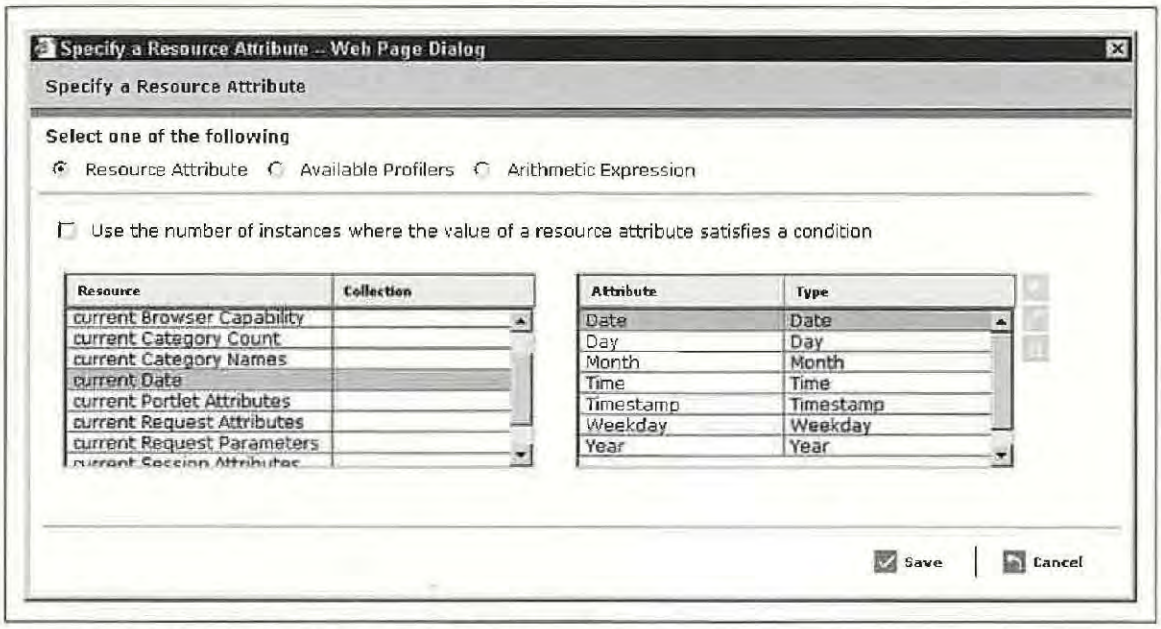


Figure 9-3 Specifying the resource attribute.

Furthermore, the Resource.Attribute link allows you to use other profilers (via the Available Profilers radio button) or an arithmetic expression combining several attributes (via the respective radio button). The `is equal to` link can be clicked to choose another type of comparison, out of the following options:

- includes
- includes any of
- is between
- is between but not equal to
- is equal to
- is greater than

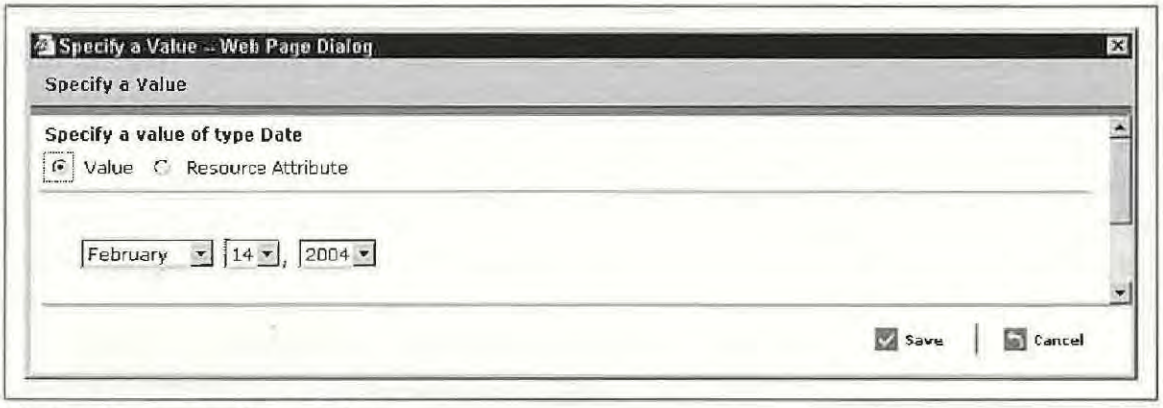


Figure 9-4 Specifying a date.

- is greater than or equal to
- is included in
- is less than or equal to
- is less than
- is not equal to

Other types of rules are edited in a similar fashion, only that the building blocks for these rules are different. Select content rules have the following structure:

```
Select content
Whose Resource.Attribute is equal to value
add Condition
order as is
show all items
```

The `whose` clause is similar to the one used in profiler rules. The two bottom links in this rule allow ordering of the results, and limiting their number. The ordering options are either "as is" (the default), in random ordering, or sorted by some attributes. `Show all items` may be replaced by putting a limit on the number of content elements to show. The limit may either be a fixed numeric value or extracted from a resource attribute.

Update rules follow a similar structure:

```
Update
Resource.Attribute set to value
```

Binder rules have a more complicated structure:

```
When Profiler is
     Profile do Action
    Otherwise do Action
    Always do Action
    Exclude do Action
order as is
show all items
```

The binder rule defines an action to take place for a certain profile. The action to perform can be defined to be any previously defined select content or update rule. The `Otherwise` clause allows you to define actions to run in case the profile is different, and the `Always` clause allows defining actions that are to run regardless of the profile. None of these actions are compulsory. The `Exclude` clause allows you to define an action whose result shall be removed from the result set of the rule. In other words, it allows you to remove items from the output of this rule. The ordering and item limit clauses are identical to those used in the select content rule.

The e-mail rule allows you to define an action by which e-mail is automatically sent. The e-mail rule has the following fields:

- Recipient
- Sender
- ccRecipient
- bccRecipient
- Subject
- bodyURI

These fields are extracted from select content actions.

Campaigns

Campaign Management is a component of Personalization that allows you to make rules to work. The portal personalization has a "default" set of rules, referred to as the *Normal View*. The Normal View defines the personalization of your Web site when there are no active campaigns. Normal View displays all of the content spots on your Web site, what content type they use, and their assigned rules.

Campaigns provide a platform for setting business goals and specifying a path for a Web site to achieve these goals. Campaigns are defined by three components: a set of rules, a specific period of time when the campaign is active, and content spots to be filled. Through the rules, campaigns may deliver personalized information via the Web or e-mail. They may also display forms to users where they can provide information to the site.

Campaigns are accessible via the Personalization tab in the portal, above the rules.

The Campaigns view shown in Figure 9-5 summarized all currently available campaigns. A campaign includes a collection of rules assigned to content spots on Web pages. These rules override the Normal View rules during the specified period. It is possible to prioritize campaigns to determine which one takes precedence in case there is an overlap in dates. It is possible to have two campaigns with the same priority. In that case, both are used based on the split percentage defined.

For example, Figure 9-6 shows a 30-70 split between two campaigns, which means the first will be shown for 30 percent of the time and the other in the remaining instances.

Within a campaign or within the Normal View, you may assign rules to each of the content spots, as shown in Figure 9-7.

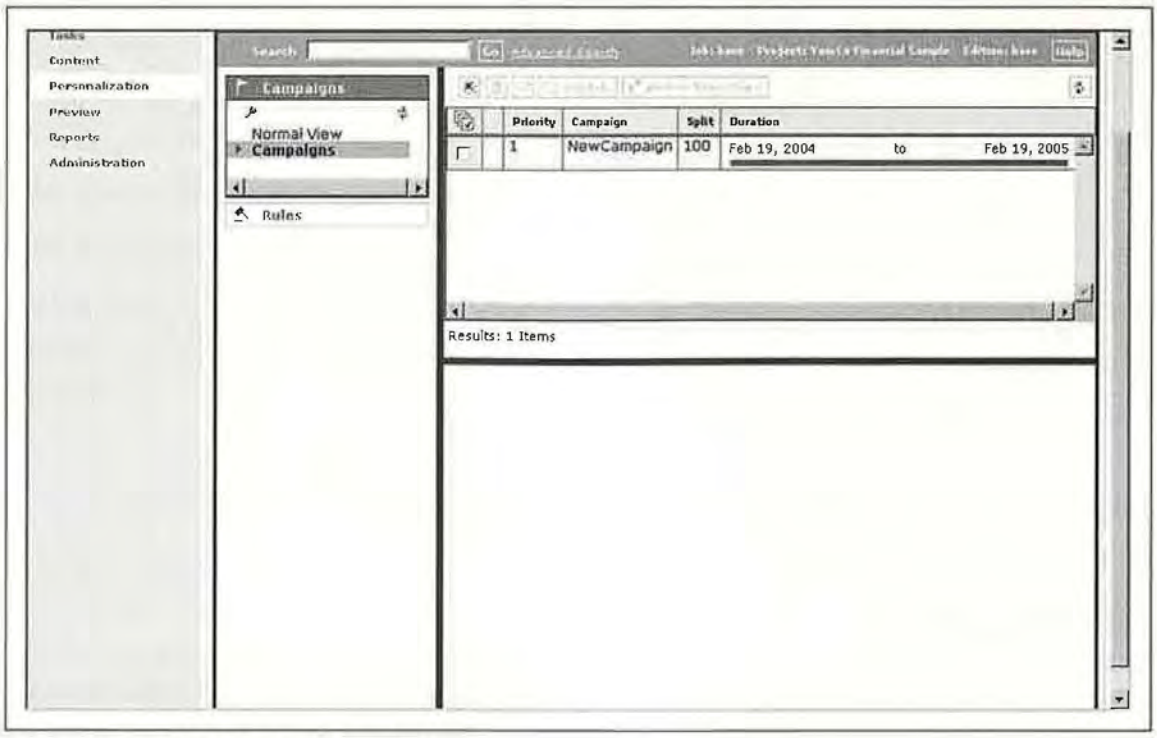


Figure 9-5 The Campaigns view.

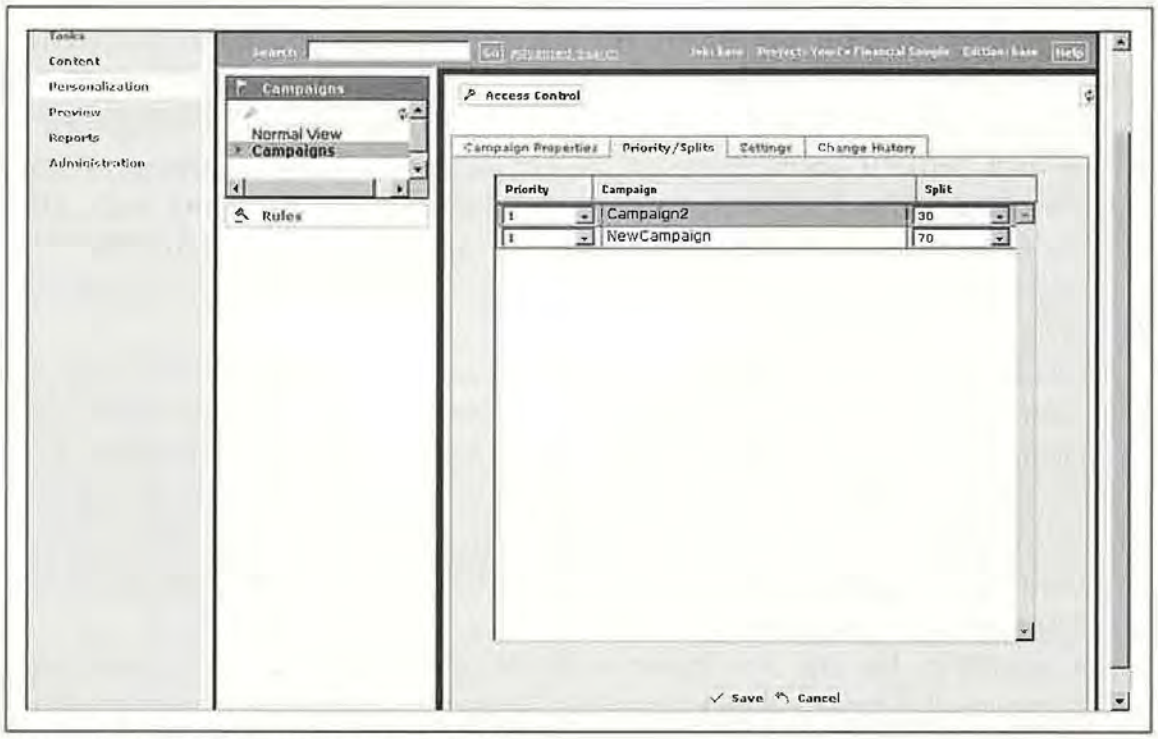


Figure 9-6 Defining Campaign priorities.

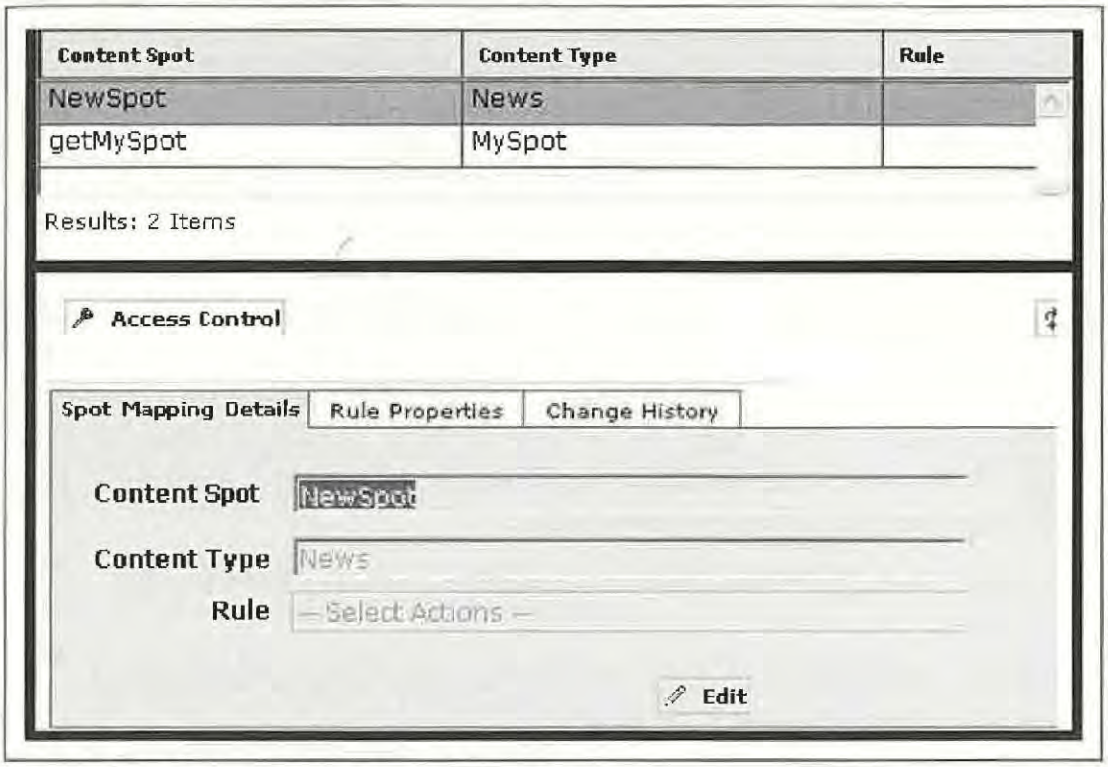


Figure 9-7 Assigning rules to content spots.

The Personalization APIs

The WebSphere Portal Personalization provides open APIs that allow the portal and WebSphere Studio Advanced Edition to access your own user and content data in customer data stores.

The WebSphere application server provides user-profile functionality via the packages:

```
com.ibm.websphere.userprofile.UserProfile
com.ibm.websphere.userprofile.UserProfileManager
com.ibm.websphere.userprofile.UserProfileExtender
```

WebSphere Personalization provides a more powerful mechanism with better support and larger flexibility in implementing resources (both user and content) for personalization. Flexibility is achieved via the support for data modeling. You can define user models and content models suitable for your own requirements. The APIs supplied by IBM to access resources in your own data store are as follows:

```
com.ibm.websphere.personalization.resources.Resource
com.ibm.websphere.personalization.resources.ResourceDomain3
com.ibm.websphere.personalization.resources.ResourceManager3
```

To provide your own implementations of the Personalization APIs for accessing resources in the customer data store, you can use wizards included

with Studio Advanced Edition or use an IDE, such as VisualAge for Java. Regardless of the method used, you need to be acquainted with the API. The following sections provide a brief summary of the API. For full details refer to the IBM product JavaDoc documentation.

The class files for your implementations of the resource access APIs must be placed in the Web module of your Enterprise Application.

Resource Interface

The interface com.ibm.websphere.personalization.resources.Resource allows you to map your user model or content model to data in your database. The resource essentially implements a property container. This interface requires the following methods:

- `String getId()`—returns an identifier (primary key) for the resource
- `Object get(String prop)`—returns the value of the named dynamic property in this resource
- `Enumeration keys()`—returns all property keys associates with the resource
- `void put(String name, Object value)`—sets the named dynamic property to a value
- `void remove(String name)`—removes the named dynamic property from the resource

In addition to these methods, you need to implement methods for setting and getting each fixed property defined in the data model for the resource. For example, for a fixed property `surname` defined for your user resource, the methods `getsurname()` and `setSurname()` must be implemented.

ResourceDomain3 Interface

The com.ibm.websphere.personalization.resources.ResourceDomain3 interface allows you to query and select resources based on fixed properties. The access is read-only mode. The methods `ResourceDomain3` use the class `ResourceContext`. The latter is an object that can be instantiated in your JSPs to pass data to your resource classes. This interface requires the following methods:

- `Resource findById(String name, ResourceContext context)`—returns the named resource
- `Enumeration findResourcesByQuery(Query query, ResourceContext context)`—returns all resources meeting the

query specification (query specification is similar to the WHERE clause in an SQL SELECT statement)

- `Enumeration findResourcesByPropertyQuery(String name, String value, ResourceContext context)`—returns all resources having the named property with the specified value

ResourceManager3 Interface

The com.ibm.websphere.personalization.resources.ResourceManager3 interface supports the process of dynamically updating, adding, and deleting resources (of type Resource) in the customer data store. This interface requires the following methods:

- `void add(Resource resource, ResourceContext context)`—adds a new resource
- `void delete(Resource resource, ResourceContext context)`—deletes the specified resource
- `Resource getForUpdate(String id, ResourceContext context)`—returns a resource having the named id (recall the ID is unique, thus there is only one)
- `void syncdelete(Resource resource, ResourceContext context)`—synchronizes any changes made (that is, making the changes permanent by storing them in the datastore)

Recommendation-Based Personalization

Rules-based personalization, as discussed above, requires a lot of work for defining the various criteria for delivering content.

Recommendation-based personalization relies on products external to the WebSphere application server to recommend content or to recommend content to visitors of your Web site. Recommendation-based personalization is powered by LikeMinds, which is not part of the WebSphere Portal Server Distribution.

Recommendation-based personalization complements and does not replace rules-based personalization. It uses collaborative filtering to capture subtle behavior patterns not captured in rules, and thus it adapts to changes without the need to create new rules. However, it can be used in the same page as rules-based personalization.

When defining new rules in the rule editor, the last option available under the rule type is `Recommend Content`. The rule structure for this rule type is shown in Figure 9-8. The content to be recommended is defined

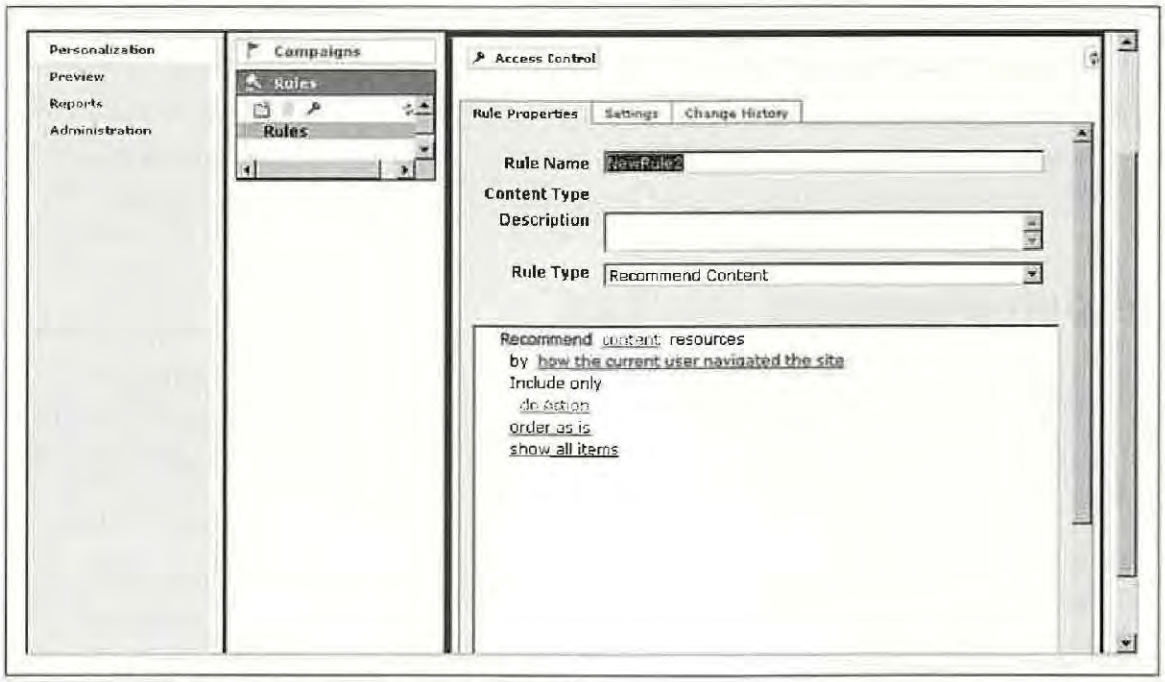


Figure 9-8 Defining a recommendation rule.

by a resource collection. The recommendation can be made by one of the following three options shown in Figure 9-9:

- `how the current user navigated the site`—this recommendation method is used to generate recommendation based on the user navigation sequence through the site. This method uses the LikeMinds ClickStream engine and is the default method. The

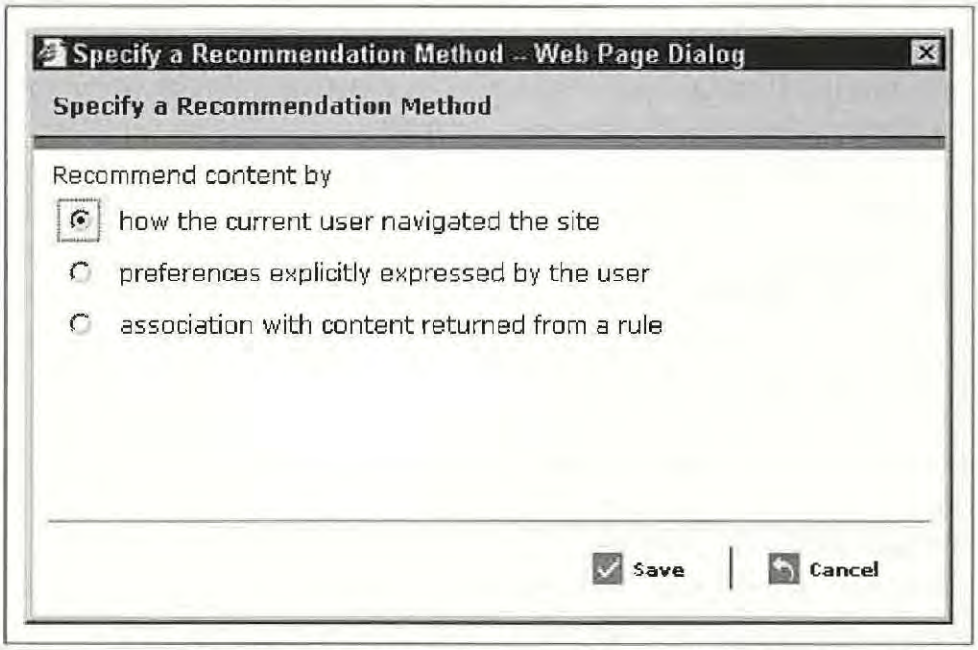


Figure 9-9 Recommendation methods available within a Recommendation rule.

ClickStream engine captures each navigation even within a bean called the Action bean.

- `preferences explicitly expressed by the user`—this recommendation method is used to generate recommendations based on the ratings users give to items. This method uses the LikeMinds Preference engine. Resources are rated using a Rating bean, and data is stored by LikeMinds for later use.
- `association with content returned from a rule`—this method generates recommendations based on market-based analysis. It associates resources that the user has interest in with items that other users have previously been interested in or have purchased. This method uses the LikeMinds Item Affinity engine. This engine makes use of transaction data being collected.

The action, sort, and limit output clauses of the rule are similar to other rules discussed above.

Summary

In this chapter we have reviewed the WebSphere Personalization solution. This solution relies heavily on content management solutions to store the actual content to be displayed based on personalization rules and recommendations. To implement personalization in your portal, you would need to get acquainted with the content management concept described in Chapter 11. In addition, content spots are deployed in the portal through portlets. You will learn to develop your own portlets in Part 3 of the book.

The next chapter describes the various facets of portal administration.

CHAPTER 10

Portal Administration

The administration component of WebSphere Portal is the "ship's bridge," the area that allows you to control the layout, portlets, server, users, groups, and resources. The Portal Administration consists of five sections: portal user interface, portlets, access, portal settings, and portal analysis. Portal User Interface was covered in Chapter 7 Defining Portals and Pages Installing portlets will be covered in Chapter 17 Portlet Interactive Debug and JSR 168 Example. In this chapter you will learn how to customize and configure your portal, use Web Clippings, administer your search engine, and perform simple portal analysis.

Your Portal Settings

To configure your portal, you first need to access the administration page. After logging in using a user ID with administrative rights (for example, wpsadmin), click Administration in the right-hand corner. This will bring you to the main administration page.

One of the first tasks you want to do after you have installed WebSphere Portal (WP) is to configure your Portal Settings. Within Portal Settings, your first stop is to click Global Settings.

Global Settings

The first setting you need to review is the default portal language. Each user can change the selected language during the enrollment process or by selecting Edit My Profile. WP comes with support for numerous languages

including Arabic, Czech, Danish, Dutch, English, Finnish, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Russian, Spanish, Chinese, and Turkish. But just in case you speak a dialect not covered (such as Swahili), WP provides the flexibility to add a new language.

You add a new language by adding it to the file and then copying and translating a resource bundle and JSP pages. Texts displayed in JSPs or stored in Java code are maintained in resource bundles. To add a resource bundle, copy and translate it, name it using the resource bundle naming convention, and then with the JDK Native-to-ASCII converter native2ascii, convert to Unicode. For any JSPs that have text directly imbedded in them, you have to translate them and store them in the appropriate location.

The next global setting you have to set is how the first page will be displayed. You have three options. Set it to the default page, the most recently visited page, or let the user make the choice at log-on. However, the latter two imply that you set the session preservation levels to 1 or 2.

The user's session state is defined by the session preservation level variable that is found in ConfigServices.properties. There are three values. If persistent.session.level = 0 then no persistent session information is stored in the database and no settings can be restored after login. If persistent.session level = 1, then the portlet states and portlet modes are stored but information regarding the last page is not. If persistent.session level = 2, then all session state information is stored.

The last two settings define portal-wide Find URL and enable transcoding of portlet content. Enter the search engine URL you want used when the user clicks Find and click Enable Transcoding of Portlet Content if you want this function to be operative.

URL Mappings

Another useful feature of WebSphere Portal is URL mapping. URL mapping allows you to create user-friendly URLs and map them to portal pages. For instance, using the default portal that is installed with WP, you want to directly reference My Finance page. The URL is `sandbox1.rigorconsultants.com/wps/myportal/!ut/p/.cmd/cs/.ce/7_0_A/.s/7_0_6S/_s.7_0_A/7_0_6S`, which you must admit is somewhat difficult to remember. With URL mapping, you can replace that URL with a URL little easier to remember such as `sandbox1.rigorconsultants.com/wps/myportal/finance`. URL mapping can also be used to map several different contexts to the same internal URL. You can externalize and provide different access rights for each new context.

URL mapping is quite simple. Under Portal Settings, click URL Mappings. In Figure 10-1, you can see that we have already added three new

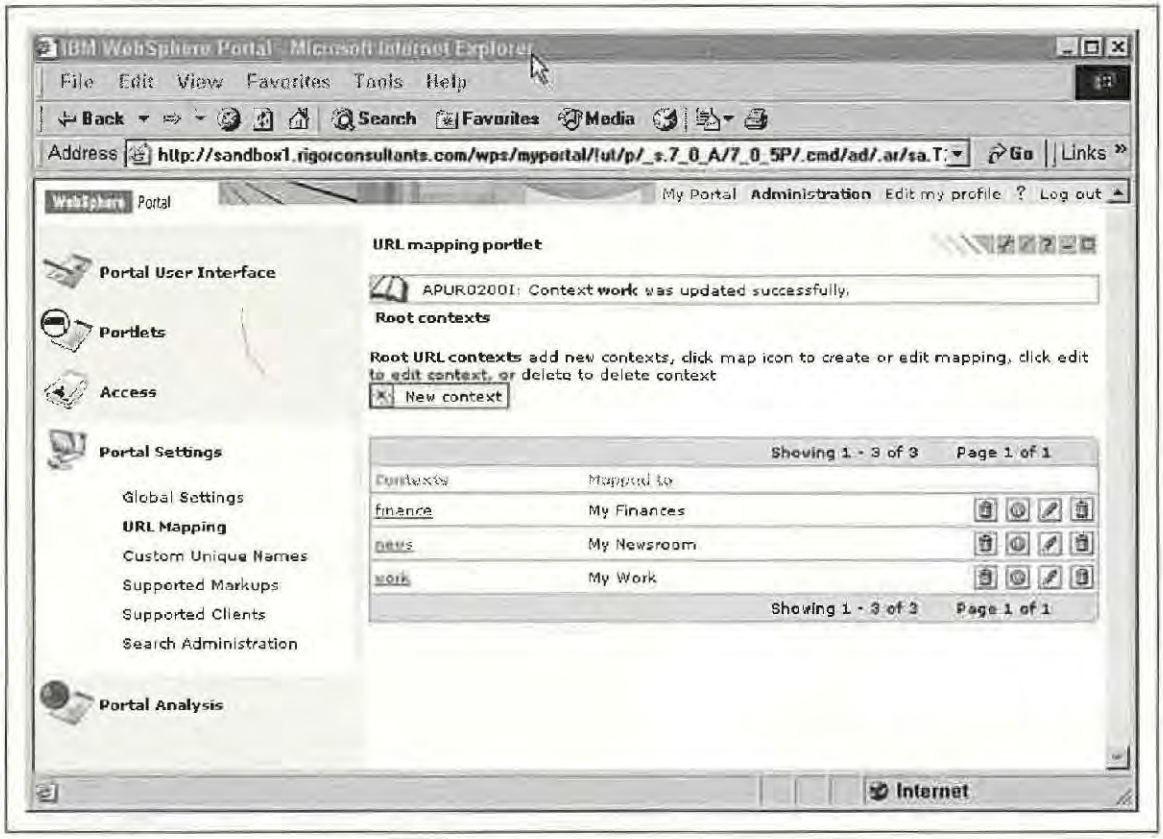


Figure 10-1 URL mapping administration.

contexts. To add a new context, click New Context and provide a label such as finance. If you wanted to add some more child contexts, you could then click Finance and create another context. However for simplicity, we are leaving it at one level. The label is appended to the portal prefix, which in this example is `http://sandbox1.rigorconsultants.com/wps/myportal/`. After you create the context, click the icon Edit Mapping and select the portal page that you want to be associated with the label and click OK. If there are too many pages, use the search capabilities. Choose the field you want to search (such as title) and then enter a value that will find the page you want to choose. If you need to change the label, click the icon Edit Label.

Custom Unique Names

Custom Unique Names allows you to associate a human readable name with the object IDs created by the portal. These object IDs are cryptic so that portal resources can be identified uniquely between different portals. Portal object IDs require human readable names since they are referenced when exporting or importing a portal configuration, linking to another portal resource, accessed by third-party vendor products, and so on.

To create or modify a custom unique name, click Custom Unique Names under Portal Settings. Select the resource type that you want to associate a custom unique name with. You can choose from the following:

- Pages
- Portlet Applications
- Portlets
- URL Mapping Contexts
- User Groups

Upon choosing the resource, a page will display the resource title, its unique identifier, and custom name. A search bar will also be displayed that will allow you to find the resource by title, description, keyword, last modified, unique name, or all available. If you click the Edit icon, you can enter a custom name for the unique identifier associated with the resource.

Supported Markups

Supported Markups is used to administer the markups associated with the portal. Upon clicking the Supported Markup portlet, you will be provided the options to add, edit, activate, show info, or delete a markup. When you add a markup, you need to give it a name, a MIME type (the Internet standard on how messages must be formatted), and default character set. More information on markups is provided in Chapter 24.

Supported Clients

This section is important if your portal supports unique devices. Supported Clients is where you define the types of devices that your portal supports. By default, most browsers and popular wireless devices are already supported. However, sometimes you want to add a unique device. To do so, you need to add support for the Sony Ericsson T310 device, which is done by performing the following steps:

1. Click Supported Clicks, which will display the Manage Clients page.
2. Click Add, and the Manage Clients page is displayed, with the Clients field.
3. In the User Agent field enter the name of the client surrounded with a period plus asterisk combination. This must match as close as possible to the user agent string the client sends in its request header. For T310, it is .*SonyEricssonT30/R201.*
4. Choose the markup from the drop-down box. These markups were defined in Supported Markups. For T310, it is wml.

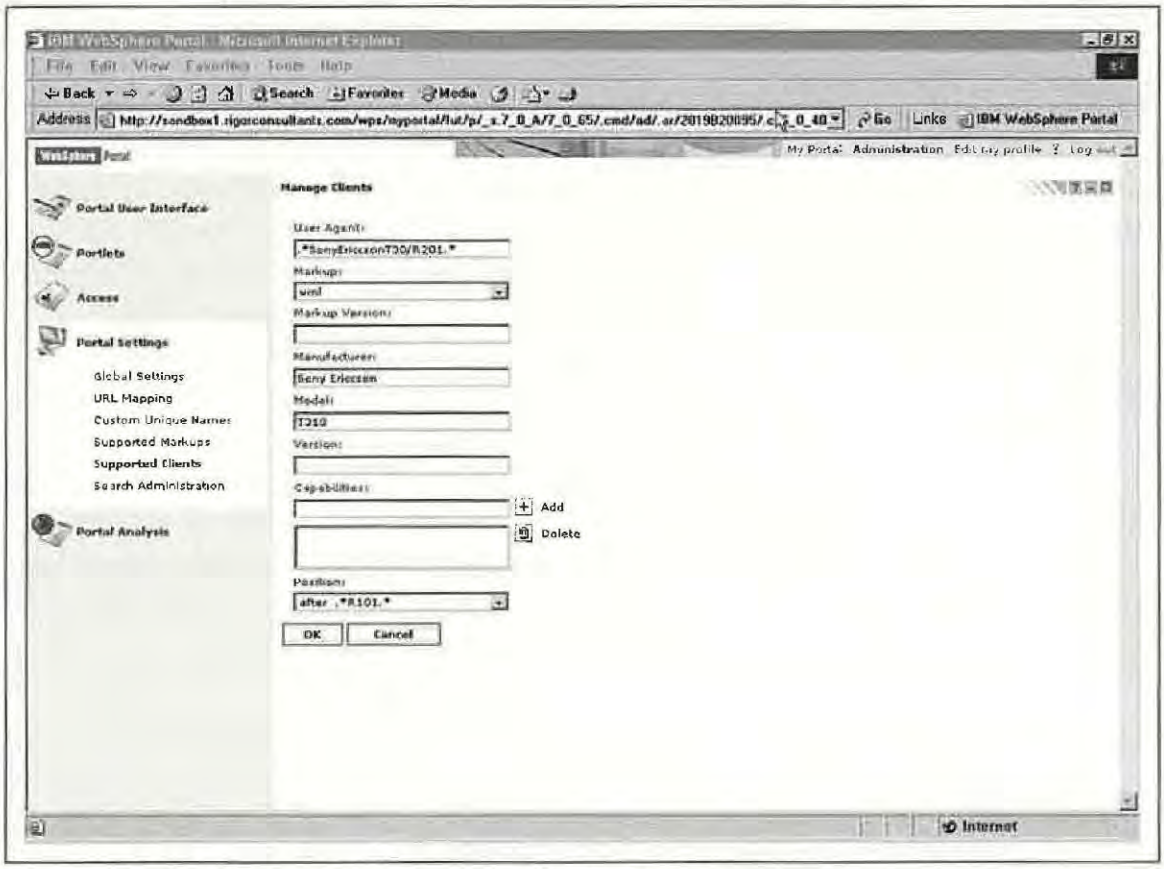


Figure 10-2 Adding a Sony Ericsson T30/R201 client.

5. Enter the markup version (optional). For this situation, leave this blank.
6. Enter the model or name of the client (optional). For T310 it is Sony Ericsson.
7. Enter the version associated with the model (optional).
8. Enter the capabilities of the client such as Frames or JavaScript. Click Add or Delete. Don't add anything since this is a dumb mobile device.
9. Choose the order from the drop-down box where the client will be entered in the client registry. The most specific user agent patterns should be placed close to the top. Add it before R201. Your screen should now look like the one shown in Figure 10-2.

Search Administration

WebSphere Portal provides a very sophisticated search and document collection capability. Basically you can create collections of various types of documents for other portal users to access. You create the collection using a very sophisticated search engine that supports a wide range of Internet

search operators. Each collection is a set of documents that can be summarized and/or categorized. Collections can be automatically categorized using a predefined static taxonomy or can be user defined. Documents can be anything from HTML to Excel spreadsheets and then can be stored on local or remote sites. The collections can be scheduled and filters can be applied to exclude or include any documents. In this section, we will discuss how WebSphere Portal search and document collection, including administration, at a high level. Chapter 12 will discuss this in greater detail.

Under Manage Documents Collection, you can define Web sites or documents, which will be prefetched by WebSphere Portal Web crawler. WebSphere Portal will create indexes that will map key words and terms to the documents. These indexes are referenced by setting the `IndexName` parameter in the Document Search portlet.

In learning how to use this feature, create a document collection from information stored at `http://www.rigorconsultants.com`. Refer to Figure 10-3.

1. Click Administration ⇨ Portal Settings ⇨ Search Administration.
2. Under Document Collections, click Create Collection.

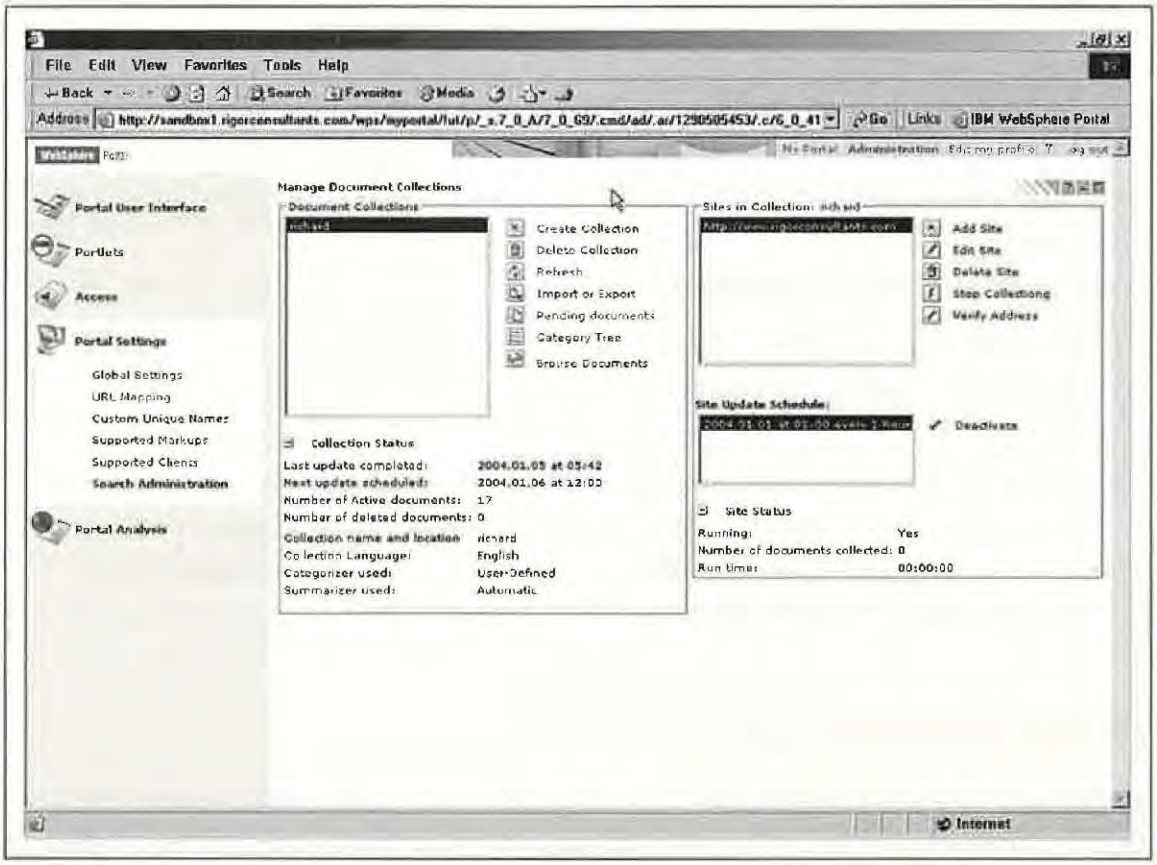


Figure 10-3 Creating a collection.

3. Specify directory path where you want the collection to be created. If you give it a name WP saves it as an appropriate directory under `<was_root>`. Call this collection Richard.
4. Specify Collection language; select English.
5. Select Categorizor. You have a choice of none, predefined, or user-defined. If you choose predefined, WP will automatically categorize your documents in over 2,300 subjects. Normally you will choose predefined; however, to show all the features you will choose user-defined. Note that when you choose user-defined, the Category Tree icon will appear on the Manage Documents Collection screen. It will not be there if you choose predefined.
6. Select Summarization. It can be either none or automatic. If you select automatic, WP will create a coherent narrative summary for each document, independent of language.
7. Check the Remove Common Words from queries box and then click OK.

Now, you have to create a custom categorization. Usually you set these up if you want to see all the documents associated with your product. In this case, you are going to create a category (surprise!) called WebSphere, which will find all Web pages that contain the word WebSphere. Refer to Figure 10-4 and perform the following steps:

1. Under Manage Document Collections, click Category Tree.
2. Under the root category, input **WebSphere Portal** as the sub-category name and click Create.
3. Expand the root tree on your left and highlight WebSphere Portal.
4. Under the Manage Category Rules Box in WebSphere Portal, click Create.
5. A new page will appear. In the Create Category Rule box, perform the following tasks:
   a. Input **WebSphere** as your rule name.
   b. Click Apply Rule to Content.
   c. Input **WebSphere** as the key word that you want to select documents with.
   d. Click Create.
6. Click the WebSphere Rule now appearing in the Manage Category Rules box.
7. Click Manage Rules.

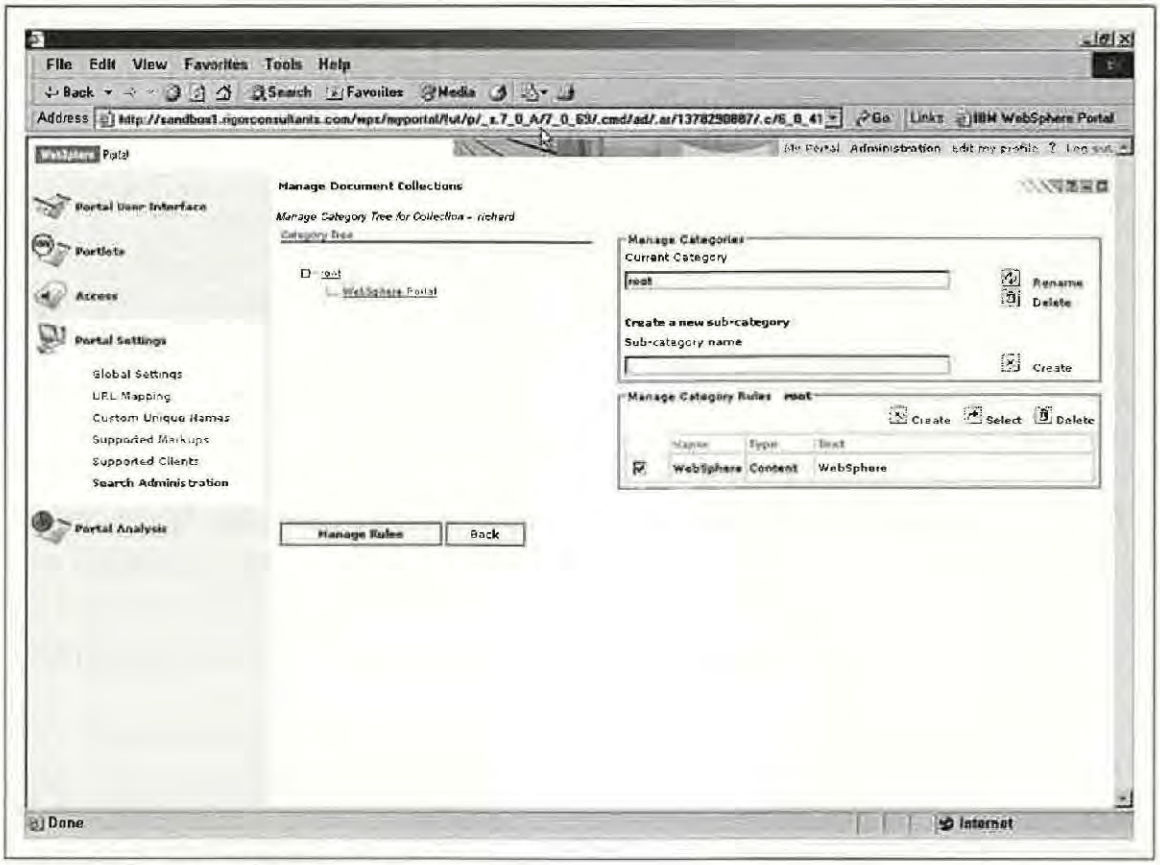


Figure 10-4 Creating a category tree.

Next, as seen in Figure 10-5, you need to specify from where documents are to be collected by following the steps given below:

1. Under Manage Documents Collection, highlight the collection.
2. Under Sites in Collection, click Create Site.
3. Enter the URL of the site. For this example, it is 5. Specify the level of linked documents to collect. This is the maximum number of levels the crawler will follow.
4. Choose number of linked documents to collect. This is the maximum number of documents that will be indexed.
5. Select the time to stop collection. The value of this field determines the maximum time the crawler will run in a single session.
6. Choose Stop Fetching a Document After (sec). This field defines the maximum time limit in seconds for completing the initial phase of the HTTP connection so that the crawler does not get stuck infinitely in a bad connection.
7. Select Link Expire After (days). Each document is time stamped. This field determines when documents will be removed from the collection.

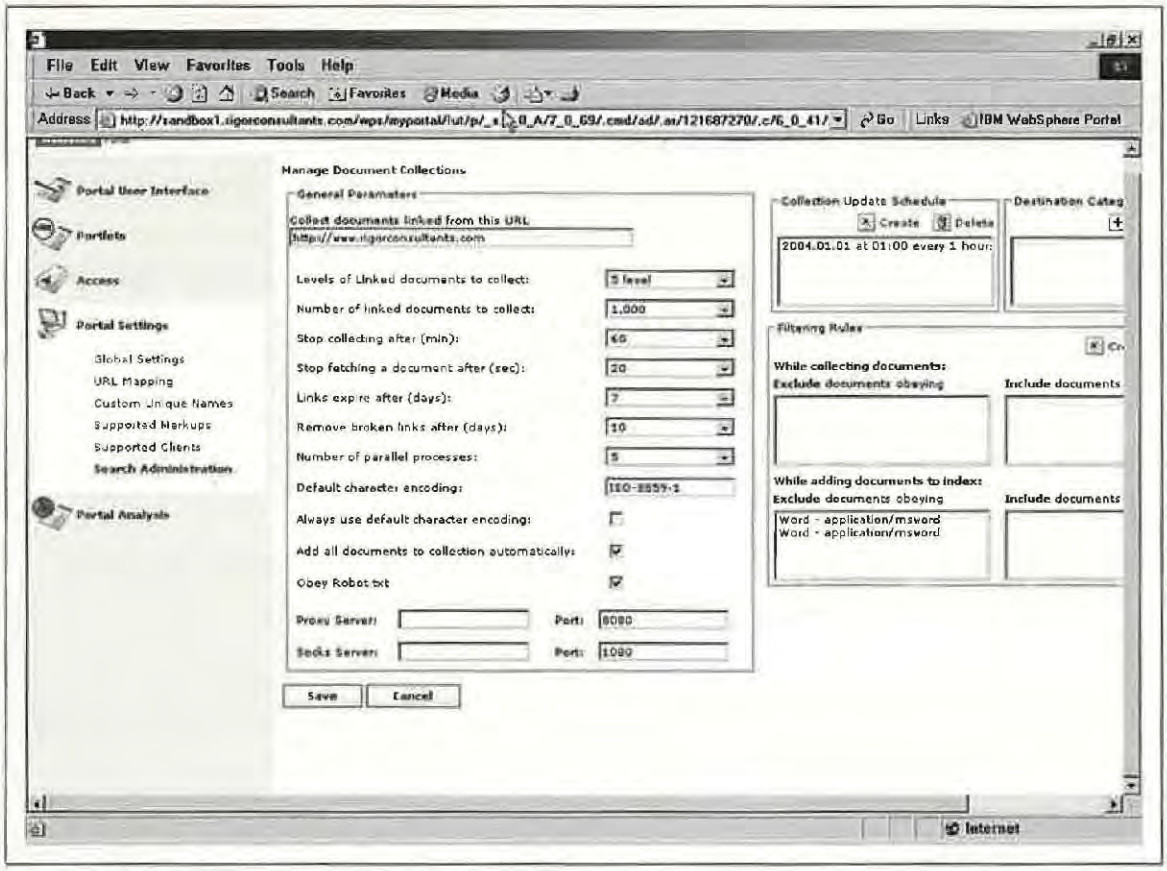


Figure 10-5 Manage Document collections.

8. Choose Remove Broken Links After (days). This field tells the crawler how many days to wait before removing documents that it found before but can no longer find because of invalid links.
9. Choose the Number of parallel processes. This is the number of threads that the crawler uses during a session.
10. Enter the default character encoding. If the crawler cannot determine the character set of a document, it uses this character set.
11. Click Always Use Default Character Encoding if you want the default character set at all times.
12. Click Add All Documents to Collection Automatically if you want the documents put in their destination folders and indexed, otherwise they are put in the pending documents folder until an administrator approves them.
13. Click off `Robot.txt` if you do not want the crawler to observe the restrictions specified in the `robot.txt` file.
14. Enter the proxy server and socks server if the crawlers uses on.
15. Click the Create button.

For this particular example, you used all the default values except Increase the Stop Fetching a Document to 20 seconds and you clicked Add all Documents to Collection Automatically. If you did not click Add all Documents to Collection Automatically, then the documents fetched would be held until an administrator released them by clicking Pending Documents (under Manage Document Collection). We also told it to obey `Robot.txt`, which is a file on Web sites that informs Web crawlers if a page should be excluded from the collection.

Next you need to create an update schedule, that is basically telling the robot how frequently to collect documents:

- Under Collection Update Schedule (as seen in Figure 10-5), click Create.
- Under Define Schedule choose the date, time, and frequency by clicking on the drop-down boxes and then click Create.

The schedule update will then appear in the right-hand side box. You can create multiple schedulers. In the schedule box you can see that we created a schedule to run the robot every hour starting at 1:00 P.M. on January 1, 2004.

After creating your scheduler, you might want to create some filters to include or exclude certain documents, Web sites, or files. In the example shown in Figure 10-6, we created a filter rule to exclude Word documents. To do this, perform the following steps:

1. Enter the rule name. Choose Word.
2. Choose your rule. Click Exclude.
3. Choose your file types. Choose Word files.
4. Click Create and click OK.

You may also want to add a destination category. If you do this all documents will be chosen based on your criteria and filters for this site will be associated with this category. To add a destination category do the following:

1. Under Destination Categories (as seen in Figure 10-5), click Add.
2. Expand the Category Tree, and click the sub-category.
3. Click Add to the List and click OK.

After you finish creating your site, click Save.

Now to verify whether it is working, under Manage Document Collections, click Start Collection, wait a few seconds, and click Browse Documents. You should see results similar to Figure 10-7.

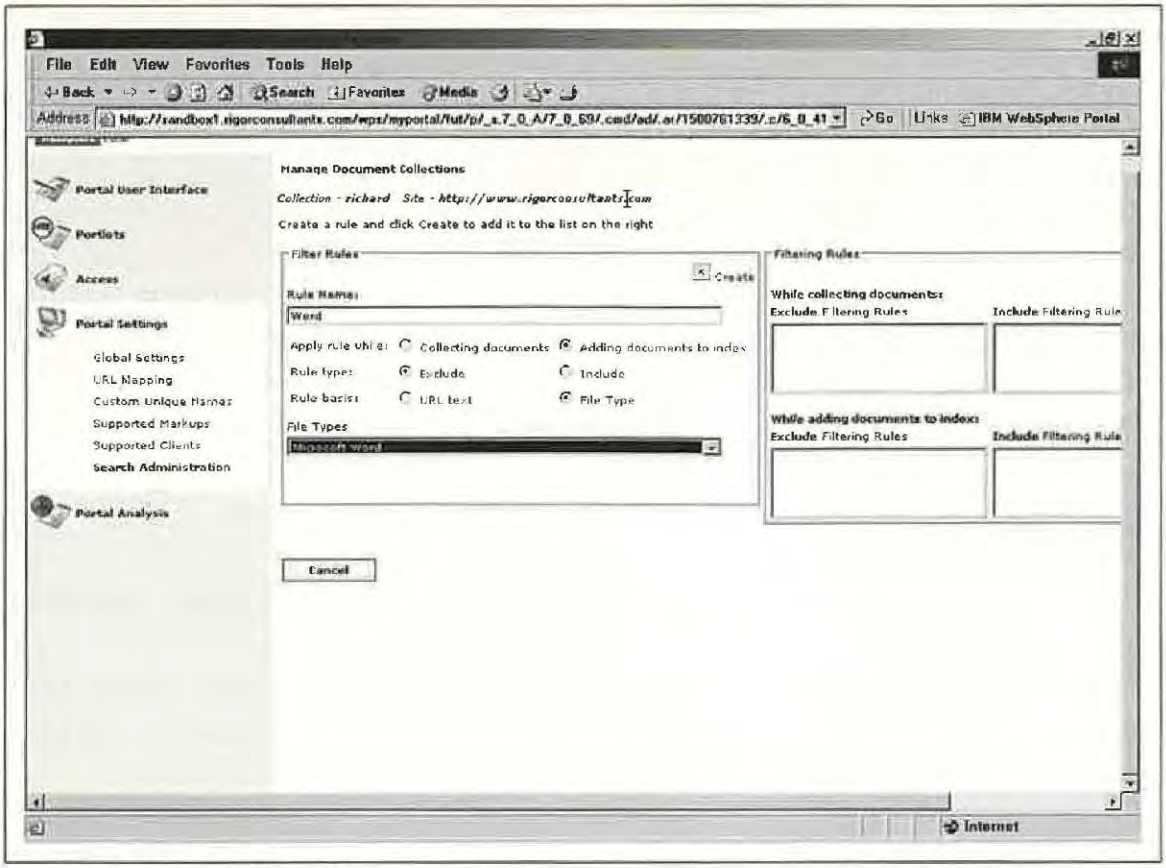


Figure 10-6 Creating a filter.

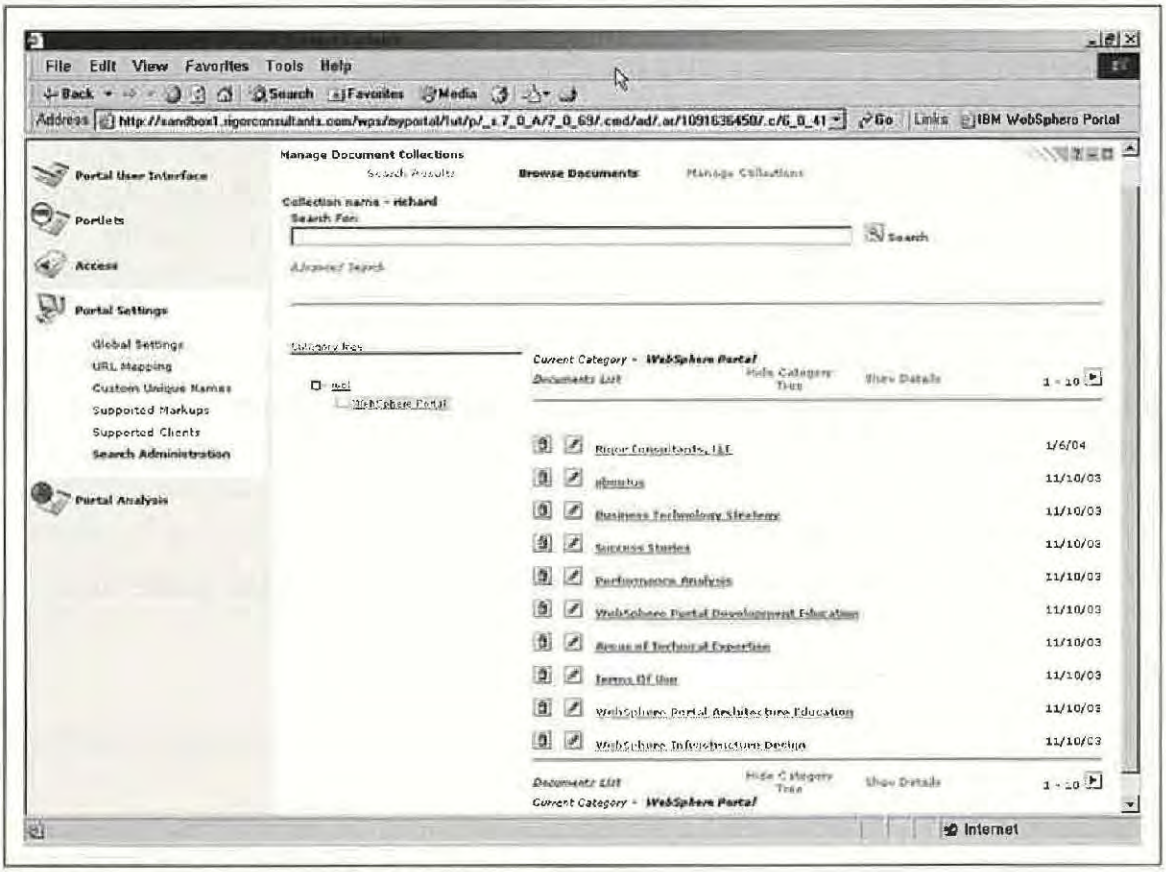


Figure 10-7 Browsing the Document Collection.

Portal Access Control

Once you have defined your default portal settings, the next step is to add some groups and users. Account information can be registered and managed by users themselves or an administrator. Under Administration, click User and Group Permissions. You first add the groups you want. Normally, you will have at least three groups: a group for administrators, managers, and users.

User and Groups

In WP V5, Manage User Groups and Manage Users have been combined into one portlet.

You have already defined the group for administrators, wpsadmin. This group was defined when you installed the program. Now define a group for general users called general as follows:

1. Click Users and Group.
2. Click the New Group button.
3. In the Name text field, enter a group name. We will call our group general. The group name can have blanks but like all IDs and password may contain only the characters a–z, A–Z, period (.), and underscore (_). The group name, like all IDs and passwords, can contain between 5 and 256 characters; however, the chance of your forgetting a 256-character password is quite high. If you are storing your names and passwords in an LDAP, the size will be greatly reduced because the schema fields are included.
4. Click the OK button.

Now click General and add a new user called Richard. Every field prefixed with an asterisk is mandatory.

1. Click the New User button.
2. Using the same rules mentioned above for group name, enter user ID.
3. Enter password.
4. Confirm password.
5. Enter first name.
6. Enter last name.
7. Enter the user's email (optional).

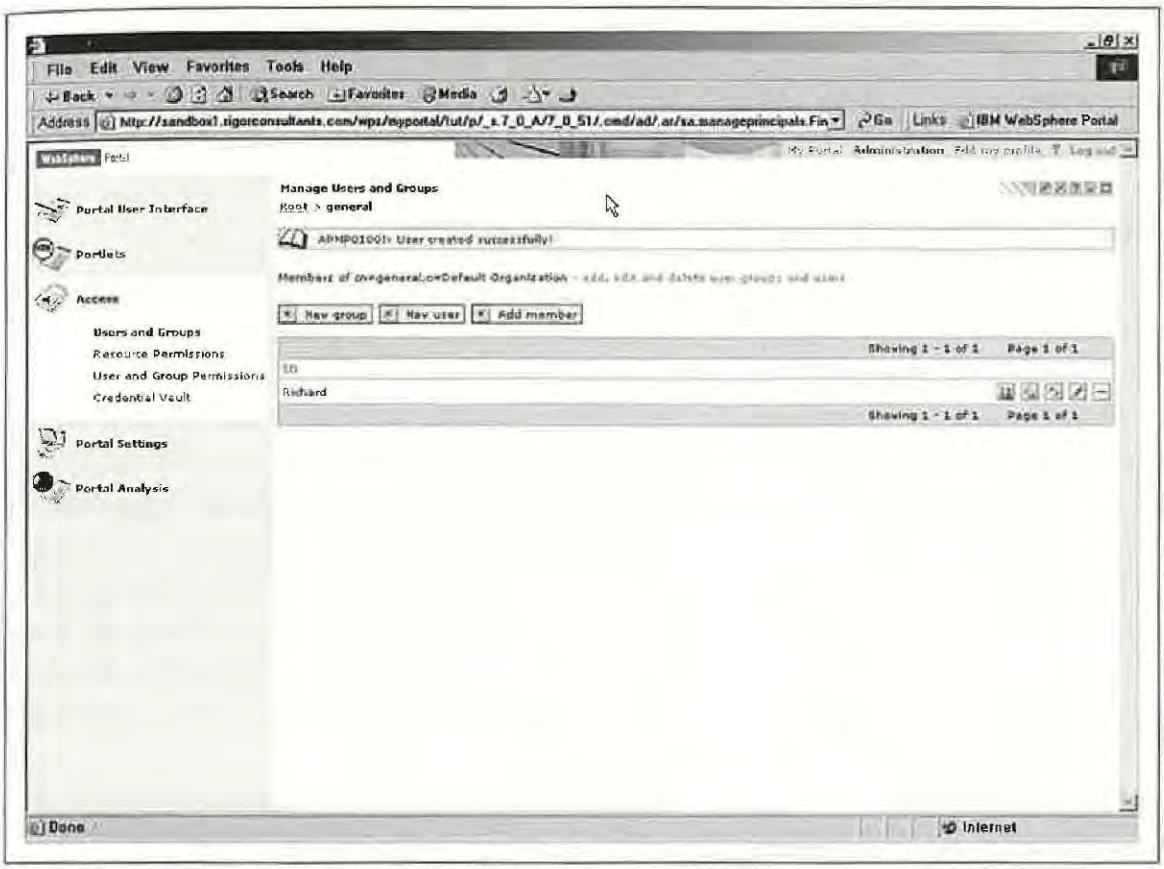


Figure 10-8 Creating and modifying users.

8. Enter his or her preferred language. This is the language that will appear on the portal pages after the user signs in.
9. Select OK.

In Figure 10-8, you will now see the user Richard with five icons associated with it. These will enable you to view the groups of which the user is a member, duplicate group or role assignments, edit a user, or remove a user. It is easy to understand the function of the icons for viewing and editing. The icons for duplicate group or role assignments allow you to choose another user or group and inherit its permissions. These are new for this version. Please note to remove the user from the group and do not delete it from the portal. To delete a user from the portal, find the user under all the authenticated user group, and click Delete.

Permissions

In WebSphere Portal, you can select, view, and modify only those resources for which you have access rights. Access rights are administered using the Resource Permission and the User and Group permission portlet found

on the Administration page. They are usually defined by groups but can be defined by users. The access rights are stored in the default database; however, a third-party external security manager can be used. This will be discussed in greater detail in Chapter 20.

Resource, User, and Group Permissions

To assign permissions to a resource, click Resource Permissions on the Administration page. To assign permissions to a user or group, click User and Group Permissions on the Administration page.

If you clicked User and Group Permissions, you will see a page with Users and Groups (depending on the permissions set for your user id). If you click on users, you will see a user called anonymous portal user. When you click Edit, you can define what resources an unsigned user can access.

If you click User Groups, you get a list of groups including the default administration group. You will also see two virtual resources called all portal user groups that define all nonvirtual user groups and all authenticated portals, which are all known user groups. When you click Edit, you can define what resources a group can access.

If you clicked Resource Permissions, you will see the resource types listed in Table 10-1 on the page.

Click on the resource that you want to assign permissions. When you get to the resource, click Assign Access. You will see a page with a list of roles.

Roles

In WebSphere 5.0, access control changed with the introduction of roles. A role is a defined set of permissions associated with a resource (such as a page or a portlet). They enable access to specific operations within the WP such as viewing a page or modifying a layout. Roles use the convention Role@Resources since roles are tied to a specific resource. For example Manager + My Portal Page is defined as Manager@My Portal, see Figure 10-9.

There are seven different Websphere role types as shown in Table 10-2. They are hierarchical in the sense that each role from the top down has less permissions. For instance the top role is the administrator, which changes, reads, updates, and deletes any resource. The bottom role (excluding no role) is the user that can only view the resource it is associated with.

Roles can be explicit or implicit. You can explicitly associate a resource, such as a user, with a role. Alternatively, you can associate a group with a role and any members of the group will implicitly inherit the role. Practically, you would only associate roles with groups since the administrative overhead of associating a role with each resource will be huge.

Descriptions of these roles are shown in Table 10-2.

Table 10-1 Resource Permissions

RESOURCE	DESCRIPTION
Pages	Set permissions for pages within the portal. If you set the permissions for a parent page, the child pages will inherit them unless you specifically block a page
Web Modules	Set permissions for Web modules within the portal
Portlet Applications	Set permissions for portlet applications within the portal
Portlets	Set permissions for portlets within the portal
User Groups	Set permissions for user groups within the portal
URL Mapping Contexts	Set permissions for URL mapping contexts within the portal
Virtual resources	Virtual resources are resources that have access control definitions but are not actually represented in the portal. Here you can set permissions for the various virtual resources such as markups, portal settings, event handlers, and so on
WPCP Projects, WPCP Editions, WPCP resource Collections, WPCP Directories, WPCP Resources	Set permissions for WPCP resources within the portal

Table 10-2 Role Definitions

ROLE	DESCRIPTION
Administrator	Super-user. Can do everything including creating, configuring, and deleting resources. Administrator can also change the access control configuration
Security Administrator	Can create and delete role assignments for roles tied to specific resources
Delegator	Can assign users or user groups to roles
Manager	Creates, configures, and deletes new or existing resources for use by one or more users
Editor	Creates, and configures new or existing resources for use by one or more users
Privileged User	Can create new private pages, view portal content, and personalize portlets/pages
User	Can view portal content
No Access	Can do nothing

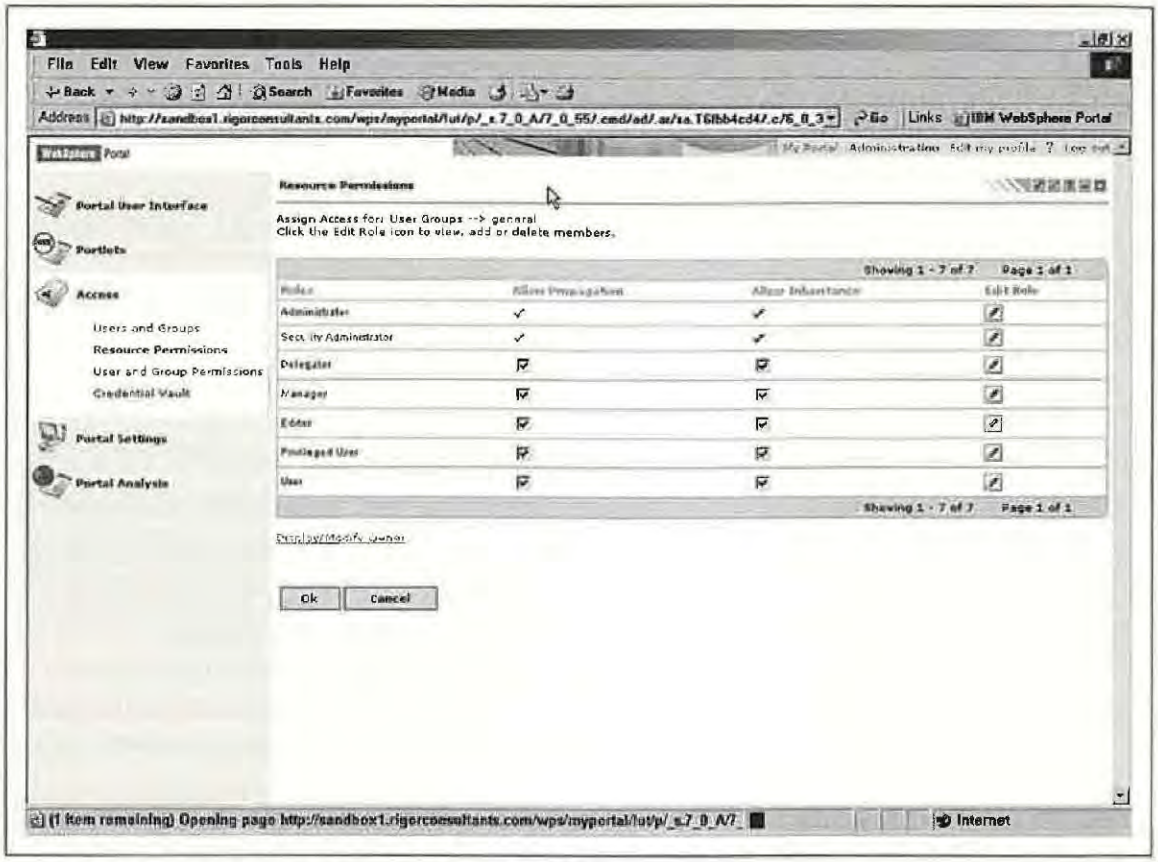


Figure 10-9 Assigning permissions and roles.

For a resource, you can determine which roles will allow propagation or inheritance with the exception of the administration and security administration roles, which always support both.

Inheritance and propagation are functions supported to reduce the overhead of administrating a large number of resources. Inheritance enables the role to inherit any permission from its parent while propagation allows the role to pass permissions to its children. If you uncheck the box, you are basically blocking the role from inheriting or propagating (based on which box you unchecked). However, the portal also provides traversal support. For Pages and URL Mappings resources, you can navigate to the resource even if you are blocked from the parent resources. You can view the navigation entities but you cannot see the contents.

When you click Edit Role when assigning a role to a resource, you can explicitly add or remove a user or user group from the role associated with the resource.

You can also change the ownership of a page by clicking Display/Modify Owner. When you create a resource, you automatically become the initial owner of that resource. As the owner you have permissions equivalent to the Manager role.

Another unique resource available to you is private page. Private page is created by Privileged User by personating a nonprivate page and optionally creating new private pages underneath it. Only the owner can access the private page.

Under User and Groups Permission, you can indicate whether a role is explicitly assigned or inherited. You click User if you want to define permissions for an anonymous user, since all other users are part of the group; authenticated user. To get a sense of how this can be done, do something controversial: allow any user to have a Manager role with the Web Clipping Portlet. To do so, perform the following steps:

1. Click User and Group Permissions under Administration page.
2. Click User.
3. Click Select Resource Type associated with anonymous portal user.
4. Click a resource type such as Portlets.
5. Click Assign Access associated with the resource such as the Web Clipping Editor Portlet.
6. Check the box associated with Manager.

Here is a more practical example. You are going to explicitly assign permission to everybody in the staff group to create, modify, or delete pages under My Portal.

1. Click User and Group Permission.
2. Click User Groups.
3. Besides staff, click Select Resource Type.
4. Click Pages.
5. Click Content Root.
6. Besides My Portal, click Assign Access.
7. Check the Manager Role box.

Credential Vault

If you are using a single-sign on package, sharing credentials amongst your portlets, or passing credentials to your back end, you will need to set up your Credential Vault. A Credential Vault is a repository that stores credentials such as private keys, user IDs, passwords, and certificates. Vaults are then segmented into two partitions: administrator managed and user managed. Vault slots are part of a vault segment and are used to store and retrieve credentials. For administrator managed, you access the Credential

Vault portlet on the Administrator page. With this portlet, you can do the following:

1. Add a vault segment.
2. Manage a vault segment.
3. Add a vault slot.
4. Manage system vault slots.

Chapter 20 will go into further detail on Credential Vaults.

Web Clipping

Under the portlets administration section, you will find a very useful feature called Web Clipping. Web Clipping allows you to quickly create a portlet that is composed of specific portions of Web site content. Of course, it can also be the content of a browser-based application so long as it can be accessed by a URL. The design concept of Web Clipping is to allow you to extract content/documents from sites that were designed to be displayed on desktops and allow the information to be displayed on other devices (such as PDAs) that have more display restrictions.

Web Clipping has certain limitations. Clipping decisions are based on the hierarchical format of the HTML tag structure so pages with JavaScript will cause irregular results. Also pages with Frames and double-byte character set support will not display correctly. Lastly old browsers such as Netscape Communicator and Navigator 4.7 or Internet Explorer 5.0 will also generate display irregularities. Even with these limitations, Web Clipping is very useful.

You also need to be aware of how the clipping portlets handle cookies. Normally clipping portlets preserve cookies set on the server side. If you need to preserve them on the client side, then go to Administration ➪ Portlets ➪ Manage Portlets ➪ Modify Parameters and specify the cookie on the configuration parameter. For instance, if you wanted to capture the cookie name and password on the client side then you would set the configuration parameter to:

```
ClientCookies = name, password
```

You should be aware of a few other items regarding Web Clippings, which the WebSphere Portal InfoCenter goes into greater detail. Refer to this document if you need to change the restart settings, override preconfigure settings, or specify a clipping portlet caching timeout.

Now it's time to learn how to create a Web Clipping, see Figure 10-10. For example, extract a portion of the Service page of the Rigor Consultants, LLC Web site into a portlet (a little bit of self-promotion never hurts):

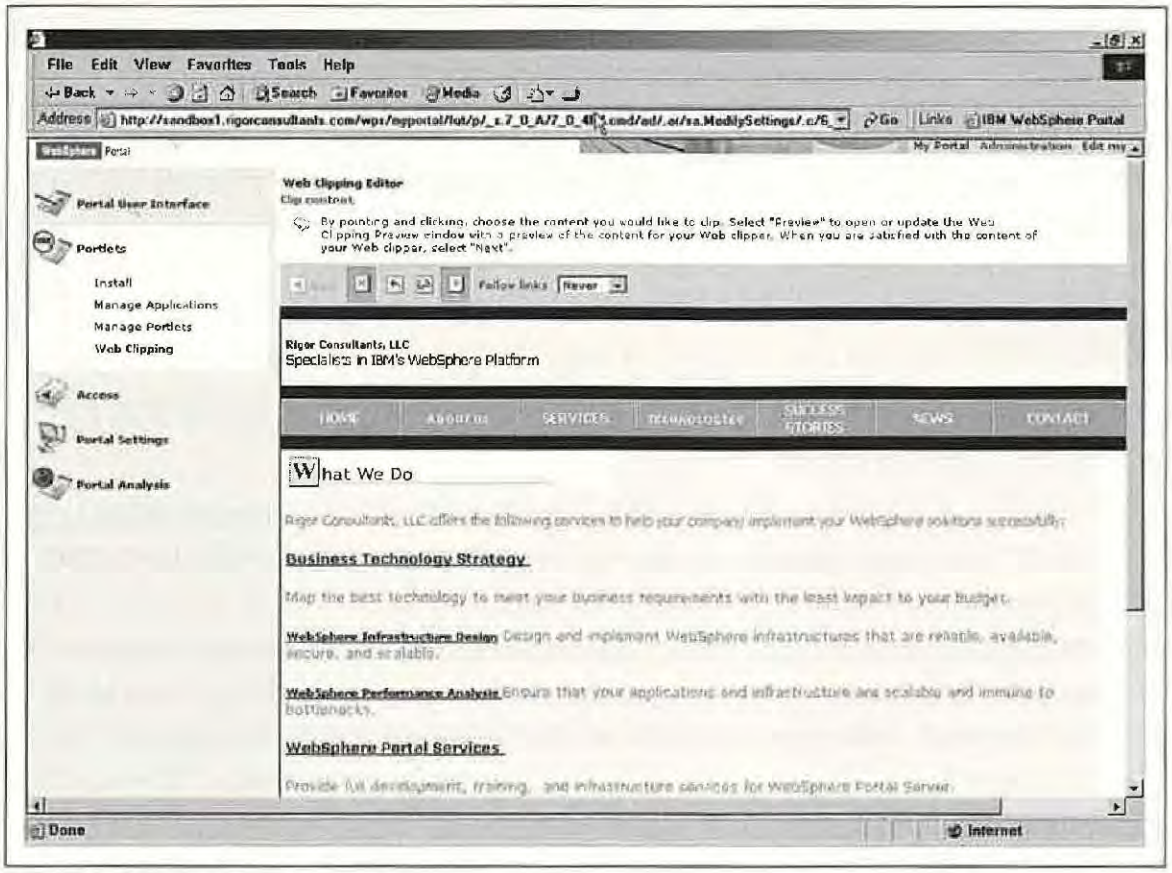


Figure 10-10 Web Clipping.

1. Click Portlets under Administration.
2. Click Web Clipping. You will see a page with a list of your Web Clippings where you can edit them or delete them.
3. Click New Portlet.

This brings you to the Add a Web Clipper function of Web Clipping Editor. Three fields need to be completed: the name of the portlet (rigor_clip), the fully qualified URL to clip (`http://www.rigorconsultants.com/services.htm`), and the description of the clip (Rigor Consultants, LLC Services). By selecting `I want to set titles and descriptions for other languages`, you can have the title displayed for different languages but this implies that you have enabled multilanguages.

If you want to select the entire page, then just select Next. However, if you need further control or security, click Advance Options. Advance Options allows you to do the following:

- Modify the clipping type. You can manually select the content type, keep all the content, or clip between specified texts. For this example, click HTML clipping and manually select the text.

- Configure for proxy server by clicking Modifying Firewall Options
- Modify authentication options or rules for URL rewriting. See Chapter 23 for further discussion.
- Remove JavaScript by clicking Modify Security Options and then clicking the check box Remove JavaScript from clipped content.

After you click OK in Advance Options, you will return to Add a Web Clipper page:

1. Click the Next button and the Clip content will be displayed in the Web Clipping Editor.
2. Click on the content for Rigor Consultant Services page and the content is highlighted in yellow. Click Preview and it will display in a separate window.
3. Click the Next button. If you don't see the Next, Back, or Cancel button, the content you clipped was too big and the page bled over the buttons. After you click Next, the Content Preview page is displayed.
4. Click the Finish button. A new portlet, clip_rigor, has been created, active, and ready to be added to a page layout.

Portal Analysis

WebSphere Portal provides a couple of portlets to help analyze and debug your portal. Clicking Portal Analysis ➪ Frequent Users enables you to determine how many users are currently logged on. The other portlet, Enable Tracing, allows you to enable or disable trace logging for individual classes and entire packages.

To trace a class or package, you enter a trace string in the Append these Trace Settings field and click Add. This trace string is added to the `shared/app/config/log.properties` file. The trace values are written to the log file, which is usually stored in a file with a name similar to `log/wps_2003.05.23-15.11.23.log` (name changes based on the time stamp).

The trace string must conform to the grammar:

```
TraceString := <ClassString>(:<ClassString>)*
ClassString := <ClassName>=<type>=<state>(,<type>=<state>)*
ClassName := a java String
state := [enabled| disabled]
type := [all| low| medium| high]
```

This is used to identify the class (or set of classes) to apply the specified change. Either this ClassName may be an exact name or it may identify a point in the class hierarchy class by using the "*" wild card character. State indicates whether you want trace enabled or disabled for this class. For instance if you want to trace the World Clock Portlet, you would enter **com.ibm.wps.portlets.worldclock.*=all=enabled**. Remember that any changes to the trace setting are only valid for the current running portal. The type is specified at the discretion of the programmer imbedding the trace as low, medium, or high. If you set type to all, the all trace types will be included.

Summary

This chapter reviewed all the stations of the WebSphere Portal "bridge" and discussed some of the powerful new features available in WebSphere Portal V5.0 from URL Mappings, custom names, role permission to Web Clippings. Also covered were the powerful new features available for searching and document collection.

In the next chapter we will explore some of the powerful information management features that are available in Websphere Portal. You will learn how to use Websphere Portal Web Content and Document Management components.

CHAPTER

11

Document and Content Management within WebSphere Portal

When you are implementing a portal (or any Web site), one of the key criteria is that content must be fresh, accurate, complete, and legitimate. To ensure this, you need a content management tool. WebSphere Portal provides IBM Lotus Workplace Web Content Management (ILWWCM). ILWWCM comes as a separate limited licensed product with WebSphere Portal 5.0.2.

WebSphere Portal also comes with Document Manager, a tool that enables you to create, edit, and share documents with other portal users.

This chapter examines the features of Document Manager and ILWWCM and shows you how to use these powerful WebSphere Portal tools.

Document Manager

Document Manager is new to WebSphere V5.0. It allows you to create and edit documents, store them, track changes and comments, and transfer them to an authorized person. A document is not just a text document but it can be a PDF, a word document, or a spreadsheet. Actually Document Manager supports importing over 100 document types. This tool is designed for business uses and entry level administrative uses. It is not a replacement for Microsoft Office or Lotus SmartSuite!

The first step in creating a Document Manager project is to log on as an administrator, click on the Document Manager tab, click Configure, and then click Create.

Enter Project Name and click OK. As shown in Figure 11-1, you will then see a number of fields. Do the following to each field:

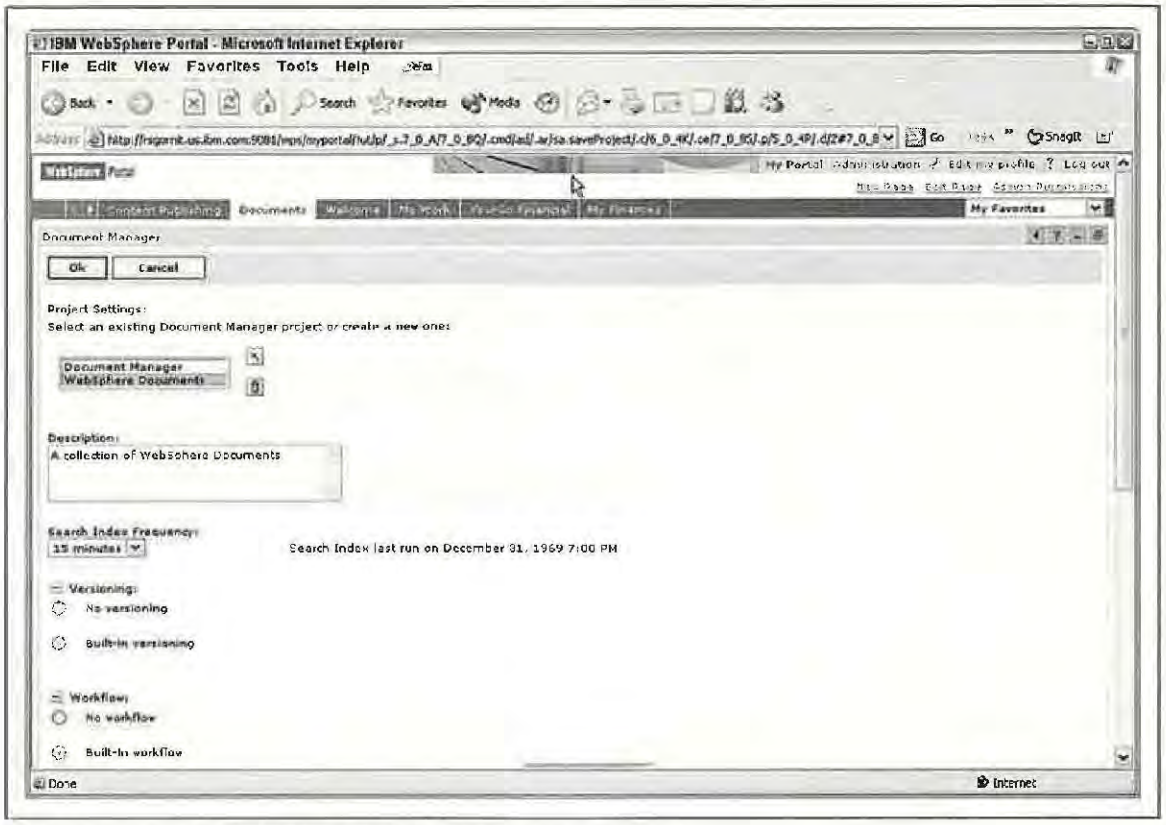


Figure 11-1 Defining a Document Manager project.

1. Highlight your Project Name.
2. Enter a description.
3. Indicate the search index frequency (the time to rebuild the search index).
4. Expand Versioning and select Built-in Versioning (if you want versioning).
5. Expand Workflow and select Built-in Workflow (if you want workflow) and select a default approver.
6. Expand Updates and select Enable Updates.
7. Expand Conversions and select Enable Document Conversions. This stores both the original document and the document converted to HTML. This is an important point, which will be discussed in more detail later.
8. Expand Editors and select Enable Portal Editors.
9. Select OK.

You, no doubt, now have some questions such as what workflow, versioning, and updates are.

Document Manager provides an optional simple workflow support. In the configuration, you assigned a group of reviewers. When an author creates or changes a document it appears in the task list of the assigned reviewer. Multiple users can have review authority but only one reviewer can approve it. If the document is approved then it becomes visible in the Document folder. If it is rejected then it goes back to the author task folder for modifications.

Document Manager also provides versioning. If versioning is activated, you are asked if you want to create a new version of the document you are saving. If you click OK, then a new version is created using automatic versioning numbering. If you want versioning, you must also check the versioning box when you create a document.

The Receive Updates option enables authorized users to see a document when it is created or updated. When a document is added or changed it shows up in your Updates folder.

After you create the Document Manager project, you need to assign access rights. In Administration ⇨ Access ⇨ Users and Groups, create two groups: wpsDocAuthor and wpsDocReviewer, and assign a user to wpsDocAuthor, and another user to wpsDocReviewer. You then click Access Control assigned to the Document Manager project (for you, it is WebSphere Documents) and associate wpsDocAuthor with editor role, wpsDocReviewer with manager role, and all authenticated portal users with the user role. Then you click Assign Access associated with Document Manager and add groups to the roles that they will be associated with. If you want some users to have read-only access to documents, then assign them the user role. The editor role is assigned to users who you want to have the ability to view and add new folders/documents or modify the properties of existing Productivity components within the current resource. The manager's role is like an editor who scans the above for all documents, deletes folders or documents within the current resource, and moves folders. An administrator can perform any operation.

Next log off as an administrator and log on as an author. Click Document Manager and then click New Folder. Give it a unique name, click OK, click the new folder, and then click Create New Document or Upload New Document.

As seen in Figure 11-2, if you click Create New Document, then you have the choice of three formats: Rich Text Editor file, Presentation Editor file, and Spreadsheet Editor file. Then choose whether you want versioning. The three productivity portlets provide you bare-bones Word Editor, Spreadsheet Editor, and Presentation Editor. However, you can also use a native editor. If you decide that you want to upload a file, Document Manager will convert the file into HTML for editing and displaying.

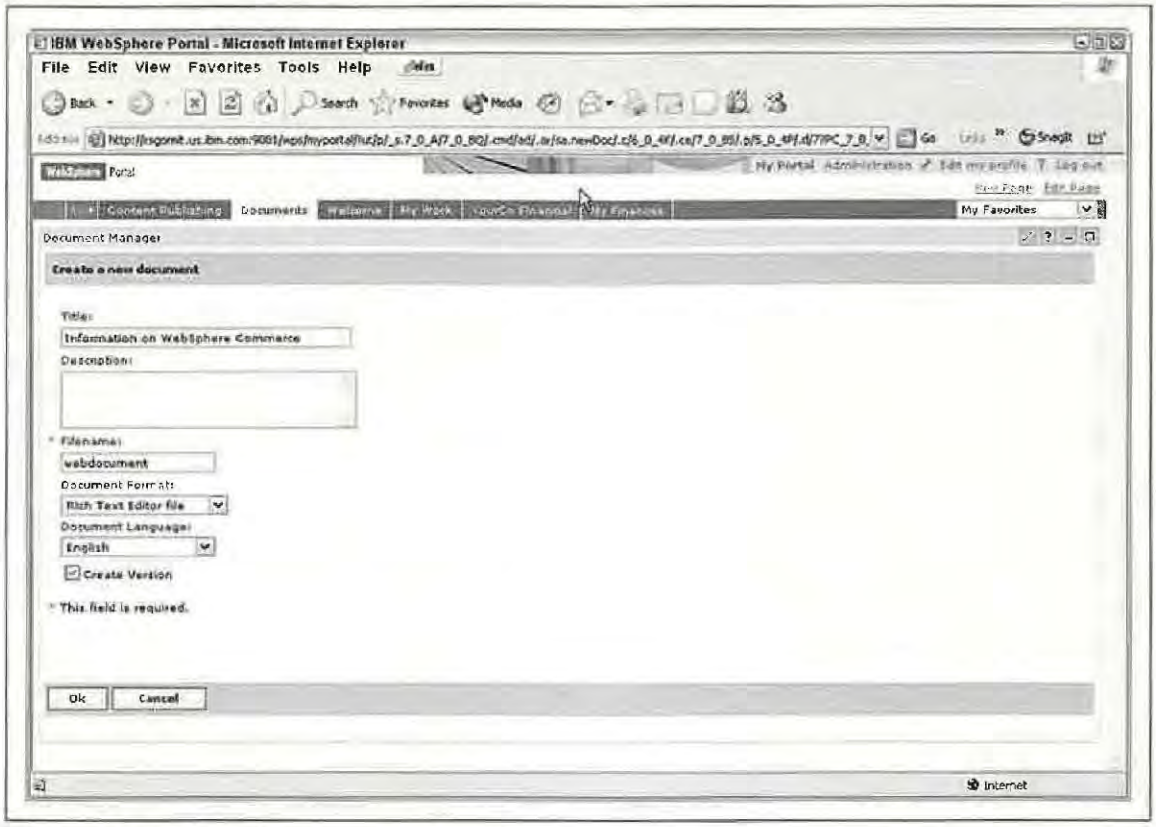


Figure 11-2 Creating a new document.

NOTE Because Document Manager converts the files into HTML, many of the original format features are not supported due to the limitations of HTML. Before using the productivity portlets, read the Document Manager InfoCenter section.

After you have modified your document, save and close it. Since Workflow is enabled, the document will be found under Tasks. All those who are authorized to see updates will also see it in their Update folder. Log off the portal as author and log back as a reviewer. When you click Documents, you will see under your Task folders documents to be approved. You can approve them by clicking Approve Change or reject them by clicking Reject Change.

Figure 11-3 shows the Documents information on WebSphere Commerce waiting to be approved under Tasks.

Now let's say you have documents in the Updates folder. Normally the document will be viewable for one day; however, if you click Edit on Document Manager you can choose how long to display the document (1 to 14 days) in the Update folder and which editors you can use.

If you want to edit and restore a specific version (obviously versioning is on), click Edit Document and click the Version tab. Click Activate Next for the version you want and it will be restored from archives.

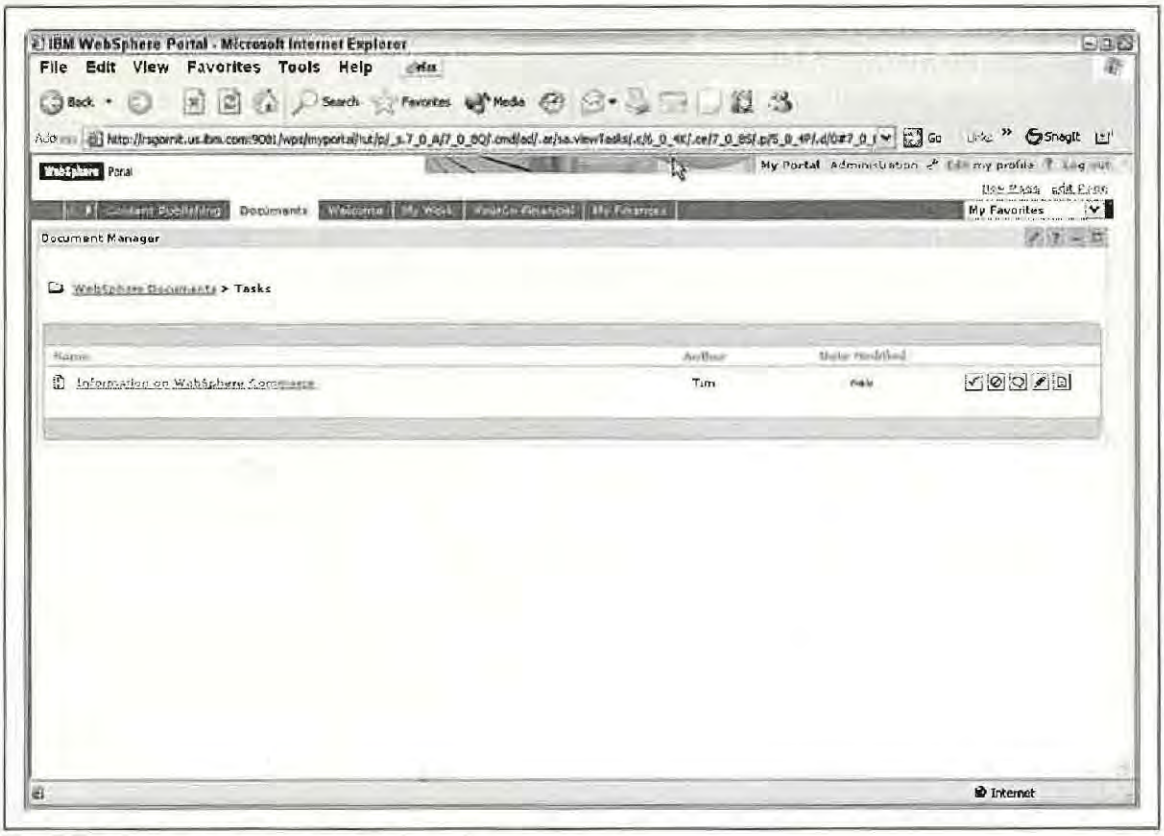


Figure 11-3 Document waiting to be approved.

WebSphere Content Management

ILWWCM provides full Web content management (WCM) services that facilitate integration of content that is accurate, authorized, easy to navigate, and complete. It is designed to replace WPCP. It provides a complete solution for creation, storage, management, and delivery of content. Services can be broken into three categories: Content Creation, Content Management, and Content Delivery. Don't get confused; ILWWCM is a Web content manager and not an enterprise content manager like DB2 Content Manager.

Content Creation enables you to develop and source content collaboratively using page design, distributed authoring, databases, and code. Content Management allows you to manage the integrity of the content and the process through staging, storage, workflow, versioning, and taxonomy. Lastly Content Delivery manages the delivery of content to the portal using features such as aggregation, syndication personalization, dynamic presentation, and caching.

The features of ILWWCM are best examined while performing a task, so, as an example, your task is to create and publish a sample site. You are encouraged to first review the material in Chapter 5, which discusses how

to install ILWWCM, as you will be using the parameters defined in that chapter. (For instance, the context root used is `wcm`.)

Creating a Site

In your portal, you are going to have a page that consists of two portlets. The first portlet will list WebSphere Books, and the second will display the details of the book that you clicked in the first portlet.

You will now learn how to create a site using content templates and menu/navigator components and then integrate it into WebSphere Portal. A workflow will be created to ensure that the published content has been approved. In our example, four people are working on the project: Jim, Scott, Tim, and Ron. Ron is Content Approver, Tim is Content Editor, Scott is Page Designer, and Jim is Site Administrator.

Next, assume that you have started WebSphere that includes server1. Then go into the WCM administrative console as `http://localhost:9080/wcm/connect` aptrix. Make sure you using Sun JVM 1.3.1_09.

Setting Up Security

The first thing that you have to define for ILWWCM is the user's roles. At this time ILWWCM does not integrate into portals' access control; however, this limitation will be resolved very shortly.

First log in as an administrator. You will see in ILWWCM on your left side a tree that contains all the functions necessary to create, manage, and publish one or more sites.

To add a user, expand Home, expand Security and Workflow and then highlight Users and Group, as seen in Figure 11-4. For each user, do the following:

1. Click New.
2. Under Type, select User from the drop-down box. A window will appear like Figure 11-5 with a series of tabs that appear in almost every window for WCM.
3. Click the ID tab and enter the name and description. For this step, enter **Ron**.
4. Click the User tab and enter the password and e-mail address. Just enter the password: **password**.
5. Click the Profile tab if you want any objects associated with this use to be automatically categorized (which we don't for this site).
6. You click Security if you want to assign permissions to users directly. You can enable Live, Read, Write, or Edit, or Delete Access. Live permission enables the user or object to read an object rendered in

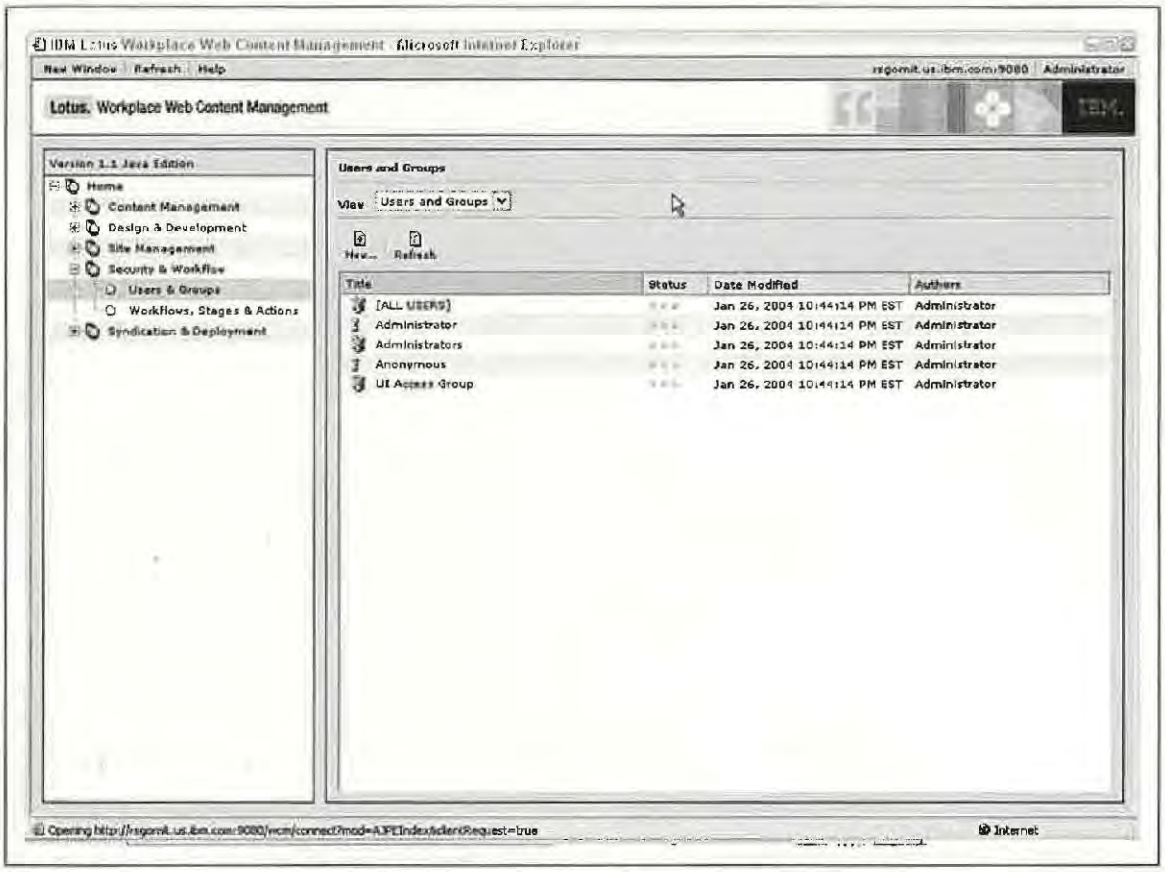


Figure 11-4 ILWWCM Security and Workflow Administration screen.

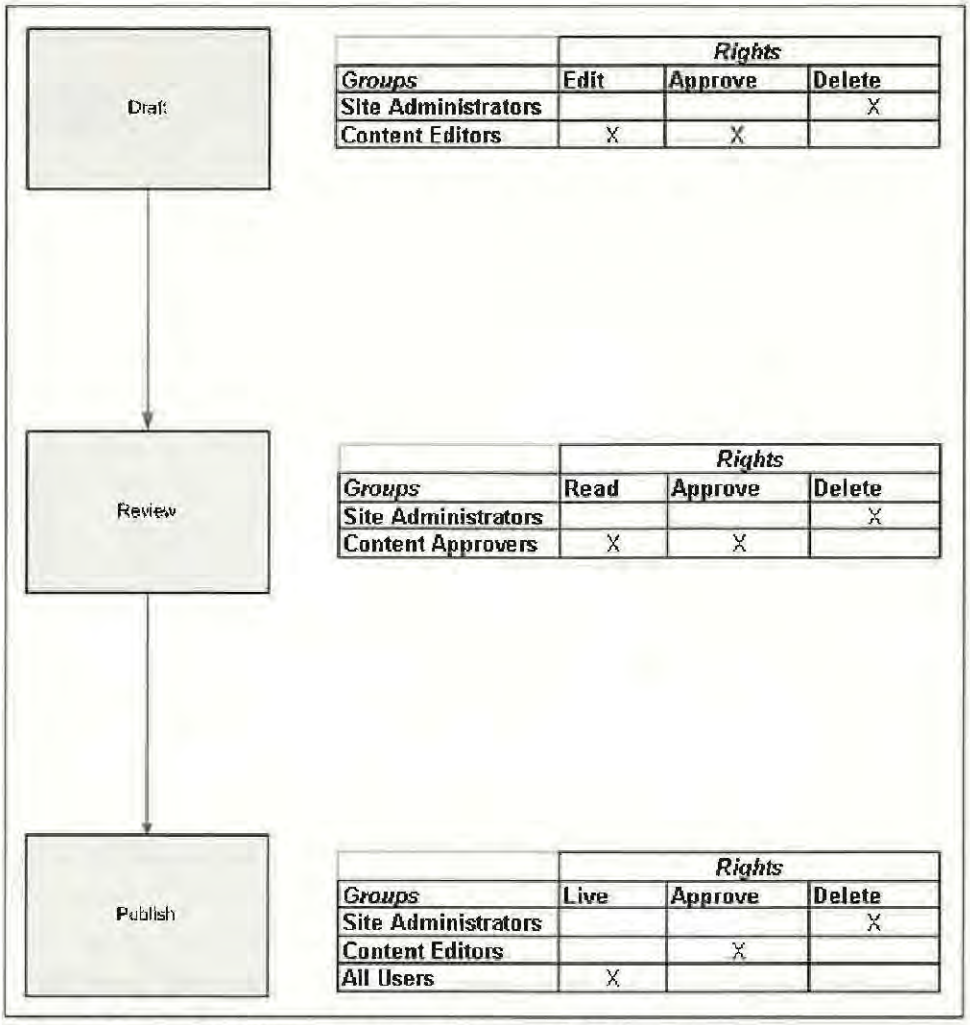
the Web browser. Selecting Delete Access automatically implies live, read, and write access. We don't recommend that you directly assign permissions to users; instead, define a group with the appropriate permissions and then assign the users to the group.

7. Lastly, you can click History, which tells you what was done to the object.
8. Click Close and then Save.

Repeat for user IDs Tim, Scott, and Jim. Next, create the groups to associate the user IDs with. Under Users and Groups, click New and choose Groups from the drop-down box and click OK. Then do the following:

1. Under ID, enter **Content Approvers**.
2. Under the Group tab, click Members and highlight Ron and click Move Right to add Ron to the group. Click OK.
3. Click Save, Close, and Refresh.

Repeat the above and create the groups Content Editors, Page Designers, and Site Administrators. Add Tim to Content Editors, Scott to Page Designers, and Jim to Site Administrators. Also add all the users and the groups to AprixUIAccessGroup so that they can log on to the Aptrix user interface.



	Rights		
Groups	Edit	Approve	Delete
Site Administrators			X
Content Editors	X	X	

	Rights		
Groups	Read	Approve	Delete
Site Administrators			X
Content Approvers	X	X	

	Rights		
Groups	Live	Approve	Delete
Site Administrators			X
Content Editors		X	
All Users	X		

Figure 11-5 Workflow process with associated access rights.

Creating a Workflow

The next step is to create a workflow. A workflow is a defined process made up of stages for the purpose of determining if content should be published. Each stage has a list of users who can approve the content going to the next stage.

Figure 11-5 shows the stages of our workflow and the rights assigned to the stage. The Approve right enables a person to allow the document to proceed to the next stage.

Do the following to create the workflow:

1. Go to Home ➪ Security and Workflow and highlight Workflows, Stages, and Actions.
2. Click New and click Stage from the drop-down box and then click OK.
3. Under Name, enter **Draft**.

4. Click Properties tab and then assign the rights, as shown in Figure 11-5.
5. Click Security tab and assign read access to AptrixUIAccessGroup and delete access to both Administrators and Site Administrators.
6. Click Close, Save, and Refresh.

Repeat for the Review and Publish stages. When defining the Publish stage, you must inform ILWWCM that it is the Publish stage by clicking on the Properties tab, clicking on Execute on Entering Stage, and selecting Publish (see Figure 11-6).

Next, you need to tell ILWWCM the order of the stages in the workflow. Under Security and Workflow, do the following:

1. Click Workflows, Stages, and Actions, and then click New.
2. Select Workflow from the drop-down box.
3. Name the Workflow **Workflow1** .
4. Click the Properties tab and then click Workflow stages and move them all to the right-hand window as seen in Figure 11-7. You define the order with the arrows on the right.



Figure 11-6 Defining the Publish stage.

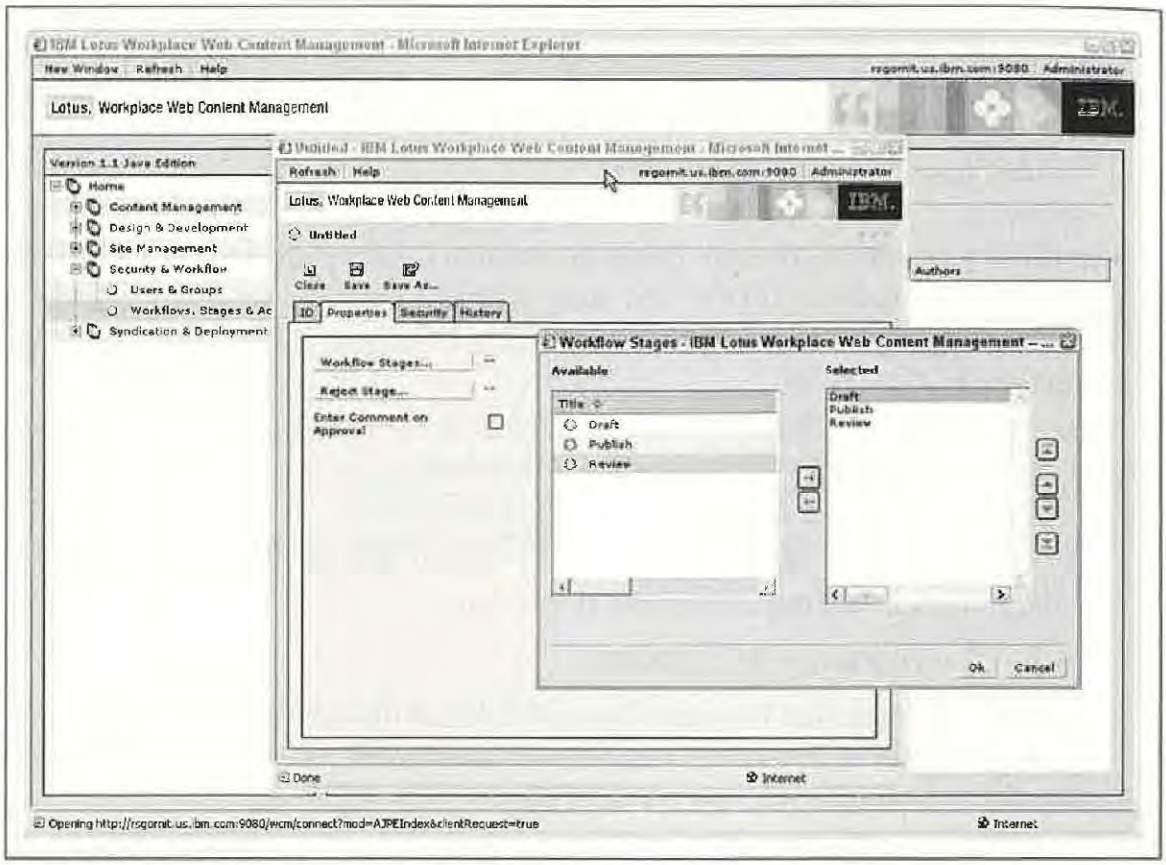


Figure 11-7 Defining Workflow order.

5. Click Security tab and assign read access to AptrixUIAccessGroup and delete access to both Administrators and Site Administrators.
6. Click Close, Save, and Refresh.

Creating Your Content Template

First, you need to create a content template that will be used by the content creators when they create new content. You can have as many as you want, each tailored for different types of content. For the particular Web site in this example, show the book title, a picture of the cover, some information on the book, and the book URL.

Do the following to create a content template WebSphere Books:

1. Expand Design and Development and Select Content Templates.
2. Click New and enter **WebSphere Books** as the name of content template.
3. Click Content Template tab.
4. In the Name field enter a description that will tell the author what information is needed such as [Enter Book Information].

5. Next click the Workflow button and select the workflow process associated with the template. For this particular workflow, it is **workflow1** .
6. Next click Component Manager and click Add. Enter BookTitle and select the type as text. Then go back and add BookCover, BookDescription, and BookLink as another component where picture is an image component, BookDescription is a Rich Text component, and BookLink is an HTML component.
7. Click Content Template and you will notice that fields for all the components have been added.
8. Click the Security tabs and give permissions to the appropriate people. Obviously, Page Designers will need delete access, Content Editors read access, and All Users will need live access.

At this point, you should be aware of the components that are used to build the template. You saw that ILWWCM supports not only text, but also various types of objects into a WCM object. Specifically, the following objects are supported:

- Text.
- Rich Text, which is Text with Web Content Management component tags. A Rich Text editor is included.
- HTML.
- File Resource—a file that is accessible from anywhere on the site or from syndication.
- Image file.
- Component-Reference—a reference to a library component.

Categorizing the Content

ILWWCM enables you to personalize content by defining content into categories. In ILWWCM you define a category tree that will enable you to display lists of content based on the category. For the example you have been using, you will define a very simplistic category tree with the taxonomy being WebSphere and the categories being Portal, Application Server, J2EE, and EJB. Do the following to create the category tree:

1. Expand Site Management, Click Category Management, and click New.
2. Select Taxonomy from the drop-down box and click OK.
3. Under the ID tab, give it a name and description. Call it **WebSphere**.

4. Under the Security tab, all security settings must be the same for the categories and the taxonomy. Give Site Administrators delete access, Page Designers and Content Editors read access, and All Users live access.
5. Click Close, Save, and Refresh.

Now under WebSphere, create the categories:

1. Expand Site Management, Click Category Management, and click New.
2. Select Categories from the drop-down box and click Next.
3. Highlight the Taxonomy with which the categories will be associated.
4. Under the ID tab, give it a name and description. Call it **Portal**.
5. Under the Security tab, all security settings must be the same for the categories and the taxonomy. Give Site Administrators delete access, Page Designers and Content Editors read access, and All Users live access.
6. Click Close, Save, and Refresh.

Repeat for the categories Application Server, J2EE, and EJB.

Creating Navigators and Menus

The next step in designing your site is to provide some sort of structure so that users can get around your site. ILWWCM provides navigators and menus. Navigators enable users to travel between areas of your site by showing the logical view of your site while menus are a list of related pages. A Navigator component consists of a start area, a child depth, a parent level, and a sibling value relative to the start area. ILWWCM is designed such that if you change your site using ILWWCM, your navigators and menus will reflect the changes. Navigators can also be used to create breadcrumbs or site maps. Breadcrumb shows a user where a page fits within his or her site while site map shows you the framework of your site.

Menus are usually just a list of hyperlinks on a page that, when clicked, will display the content. In ILWWCM, menus can search for content within matching site area, content templates, categories, key words, or any combination of the latter. Searches can also be based on category profiles. For the example in this chapter, you will create a menu that will list all the books that reference WebSphere in their title or description.

To get a sense of how WCM implements menu, create a menu that lists all the books on the site.

Do the following to create a Menu component:

1. Under Design and Development, go to Component Library
2. From the dropdown box, select Menu Component and click OK.
3. Enter ID Name. In this case, input **WebSphereList**.
4. Click the Menu Component tab.
5. Click Matching Content Templates and add select WebSphere Books.
6. Select Publish Date for the Results Primary Key and Name for the Results Secondary Sort Key.

You will then see a box for header, footer, and Component Design for Each Matching Content. The header and footer are for (surprise!) defining the header and footer. In these boxes, you place HTML for rendering the menus. The HTML consists of special ILWWCM tags used to display information about components.

In the header box, add `<h1>WebSphere Books</h1><br>`. This will provide a title to the menu page.

In Component Design for Each Matching Content, you enter your HTML code to display information about the Navigator component. In this case, enter:

```
<A HREF="<Placeholder tag="href"/>"><AptrixCmpnt context="autoFill"
key="BookTitle" type="content"/></A><br>
```

The placeholder tag can have multiple values. In this case, you instructed it to insert the URL of the matching content detail using Book Title as the link text.

Don't forget to set security; otherwise, nobody will be able to use these components. The Page Designer needs delete access, Content Editors need read access, and All Users need live access.

Page Design

The Page Design component accepts content and library components and enables them to be displayed in a formatted layout.

1. Under Design and Development, highlight Page Design and click New.
2. Under ID, assign a name. In this case, give it the name **WebSphere Page design**.
3. Click the Page Design tab and add the HTML code, as shown in Figure 11-8. Basically, all you need is to add straight HTML code with special ILWWCM that point to our content, site, or site area objects,

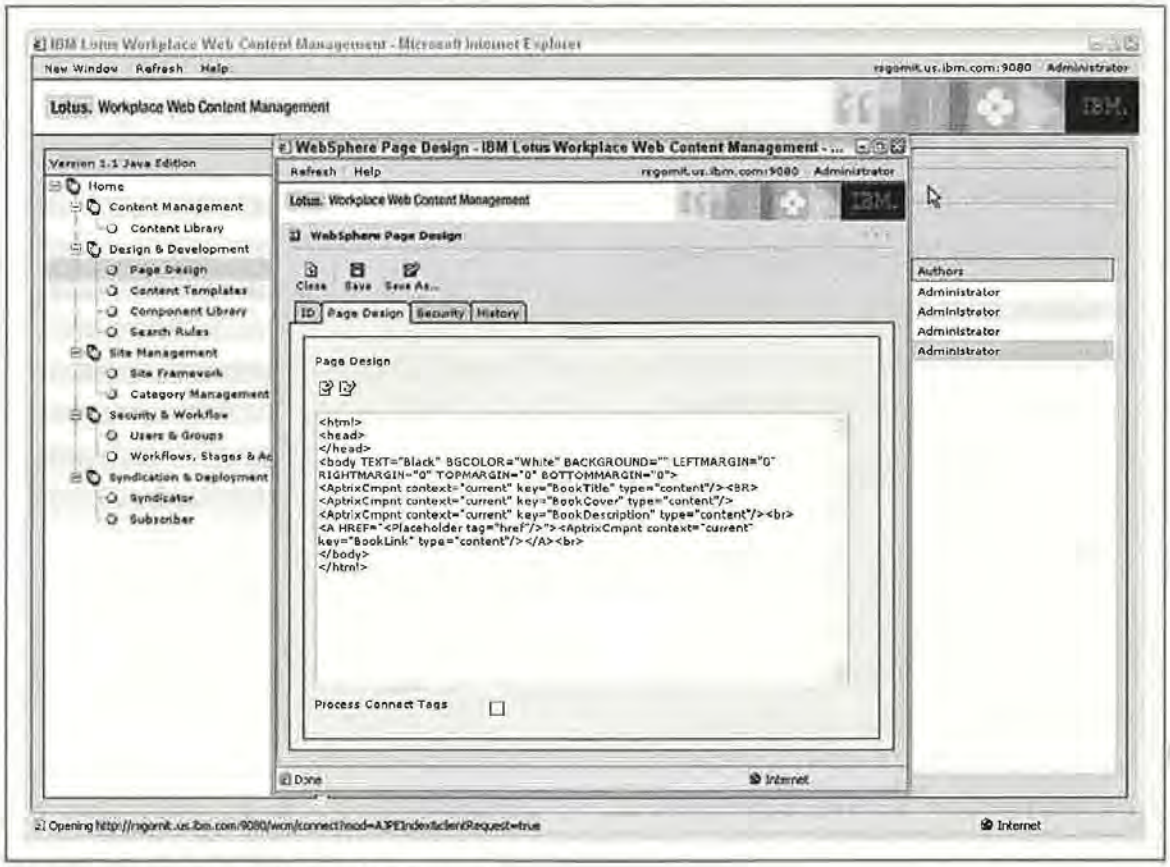


Figure 11-8 Page design HTML code.

as illustrated in Figure 11-8. The `<AprtrixCpmt>` tag is pointing to the components we defined from our content template. You can also click Import and bring in other saved HTML code. The contact parameter refers to whether the object is in the current site or whether it will be determined from the Name tag or the Menu, Navigator, or Taxonomy component.

4. Under Security, the Page Designer needs delete access, Content Editors need read access, and All Users need live access.

Building the Site

Now it's time to set up your site and define the content. Like categories, IL-WWCM enables you to define your site in a hierarchical manner by defining your site framework. The top node is your site, and the children are your site areas. For the example in this chapter, the WebSphere site will be very simple. It will consist of a WebSphere Area, your portlet. Do the following to define the site:

1. Expand Site and Framework and click New.
2. Select Site from the drop-down box and click OK.

3. Click ID tab and assign a name: WebSphereSite.
4. Select the Properties tab.
5. Click Content Template and highlight WebSphere Books. Click OK.
6. Choose Page Design and highlight WebSphere Page Design.
7. Under the Security tab, give Site Administrators delete access, Page Designers and Content Editors read access, and All Users live access.
8. Click Save, Close, and Refresh.
9. Click New and select Site Area from the drop-down box. Click Next.
10. Highlight the site area or site framework that is the parent and choose the link order, which specifies where the site area is below the parent.
11. Click the ID tab and assign a name. In this case, use the name **WebSphereArea.**
12. Select the Properties tab.
13. Choose Content Template and highlight WebSphere Books. Click OK.
14. Click Page Design and highlight WebSphere Page Design.
15. Under the Security tab, give Site Administrators delete access, Page Designers and Content Editors read access, and All Users live access.
16. Click Save, Close, and Refresh.

Adding Content

Up to this point, you have spent a significant amount of time defining the site, and the menus, security, workflow, and the content templates. Now, it's actually time to add content. Do this by performing the following steps:

1. Expand Content Management group and highlight Content Library.
2. Click New and select a content template. For this example, it is WebSphere Books.
3. Choose your site area and associate it with the content template. Expand WebSphereSite and move it to the selected area. Click OK.
4. Under the ID tab, enter a name. Call it **Portal Book 1.**
5. Click the Contents tab and enter the information in each of the fields. We added information on the WebSphere Portal book with a link back to Wiley's Web site. For our example, we added content about three WebSphere Books (including our own).
6. After you enter each document, click Save, Close, and Refresh.

You now need to view Content Library with Content by Workflow. You can now preview each document and edit it for correctness. If it is correct,

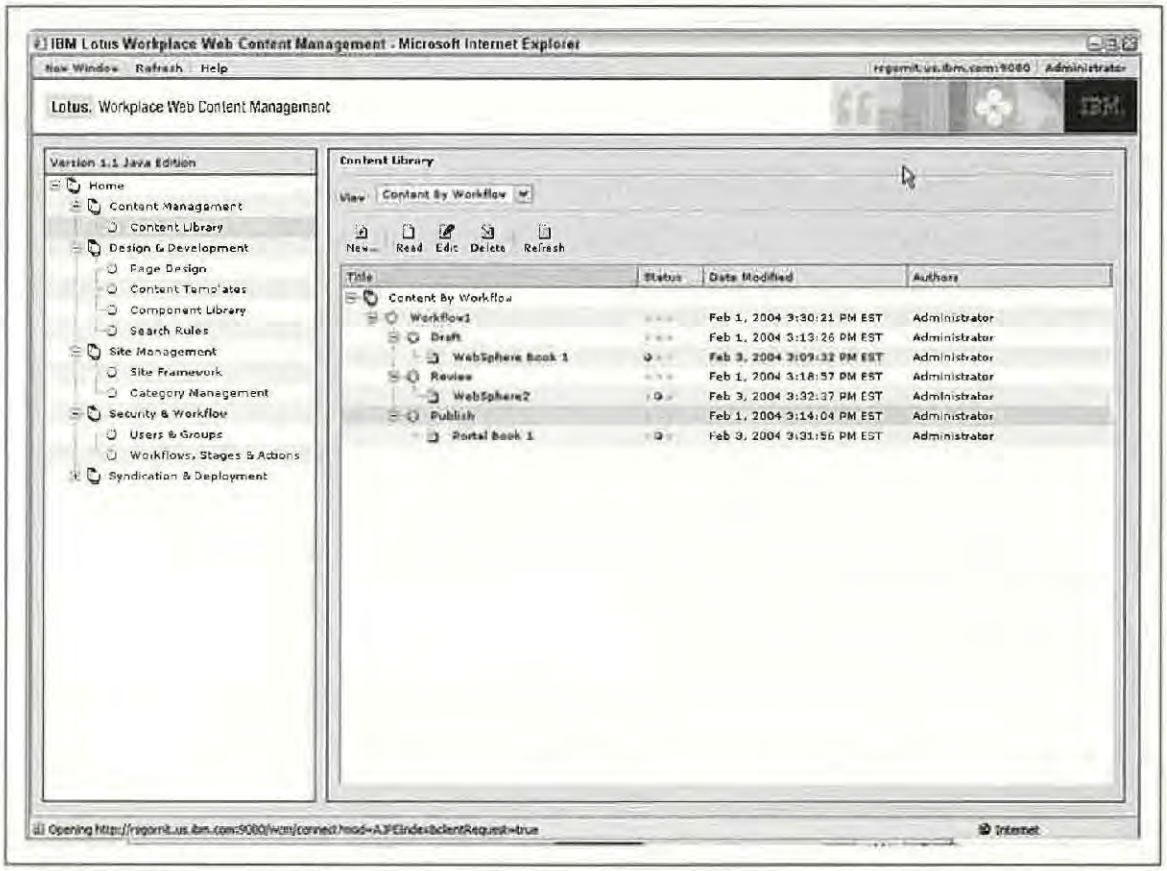


Figure 11-9 Content in various approval stages.

you click Read and then Next Stage. Content Editor will then review it and if he or she likes it will then authorize it to the next stage.

In Figure 11-9, you see that you have one document in the draft stage and one in the review stage and one ready for publishing.

Next, move all the documents into the Publish stage so that they can be accessible by WebSphere Portal.

Configuring WebSphere Portal

The next big question is, "How you see your content on WebSphere Portal?" Chapter 5 discussed installing ILWWCM; the Content Viewer portlet was installed. If you go to Administration ⇨ Portlets ⇨ Manage Portlets, you will see a portlet called Content Viewer, Java Edition. This portlet will enable you to see the content you are managing with ILWWCM.

Making a Copy of the Content Viewer, Java portlet

You will want to view content using two portlets. Since you are really using the same portlet twice on the same page, you need to make a copy; otherwise, the settings on one portlet will be the same as those on the other.

So to make a copy, under Administration ⇨ Portlets ⇨ Manage Portlets, highlight Content Viewer, Java Edition, click Copy, and highlight the copy version (it should stay inactive). Click Modify Parameters and then click the English button. Click Set Title for Selected Locale and change to something unique such as WebSphere Book Detail. Click Save, Close, and Activate. We repeated this for our other portlet WebSphere Book List.

Creating a Credential Vault

In order for the portlet to access the data in ILWWCM, it needs to authenticate against the ILWWCM server. This means that you need to set up a Credential Vault slot with the user ID and password that you use to log in to ILWWCM. You do not need to use the Credential Vault if you pass the portal user and it is defined in the ILWWCM or if you are using LDAP authentication and the user is mapped to the ILWWCM group.

As shown in Figure 11-10, create a vault slot called ILWWCM, set the vault slot to Shared, and enter your user ID and password.

Setting Up Your Portlets on a Page

The next step is to create a page and place your portlet on the page. Do so by performing the following steps:

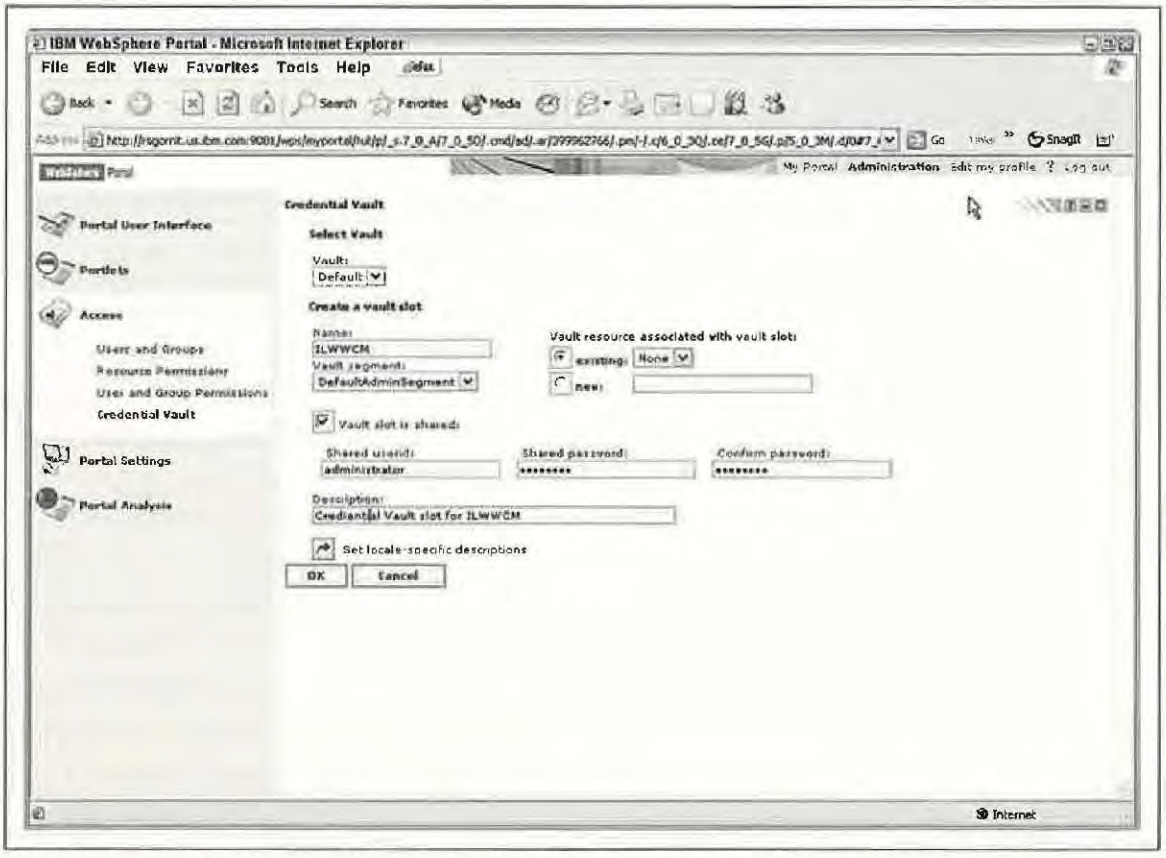


Figure 11-10 Setting up the Credential Vault for ILWWCM Viewer portlet.

1. Go to Administration ⇨ Portal User Interface ⇨ Manage Pages ⇨ My Portal and click New Label. Enter your Title. Call it **Books**.
2. Click the label Books and add a new page called WebSphere and click OK.
3. Click Edit Page Layout, add your portlets, and click Done. In this case, add WebSphere Book List to the left-hand column and WebSphere Book Detail to the right-hand column.
4. Navigate to the page, and you will see your two portlets with a message asking you to configure them.
5. To configure your portlets, click Configure.
   a. Enter the Portlet Display Title.
   b. Enter the ILWWCM server host name. For us it was http://192.212.131.100:9080. You need to use IP (Known issue). You need to use the IP address because if you use the same host name for both the portal and ILWWCM, then the session gets invalidated when you try to configure the WCM portlet. The workaround is to define multiple host names on client machines `<etc>\host` file.
   c. Enter the Web Content Management Server Path, which for us is `/wcm/connect`.
   d. Enter the Web Content Management Portlet Module, which is AJEPortletModule (see Figure 11-11).
   e. Click the Credential tab, click Override User Logon, and enter the Credential Vault Slot Name: **ILWWCM**
   f. Click the Links tab and choose Receive Links from Other Portlets and this Portlet so that the left portlet can send you an action.
   g. Click Content tab, select Content, and then click Edit under Content Component. The content site tree will appear. Choose the default content that you want to appear under the portlet. Choose Portal Book 1, as shown in Figure 11-12. If the Content tree 41is not displayed and you get an SHA1 error, then open the `wpsV5-j-ilwwcm-Cportlet.war` file using a Zip utility and extract `AptrixClientApplet.jar` to `C:\Program Files\WebSphere\PortalServer\installedApps\wpsV5-j-ilwwcm-Cportlet_PA_1_0_69.ear\wpsV5-j-ilwwcm-Cportlet.war\` (or the equivalent location in your Portal Server) overwriting the correct AptrixClientApplet in the applet directory. The error was caused by the applet signature getting corrupted.
   h. Click the Edit button under Alternative Page Design and choose a path. A path must be given even if you are not using it.

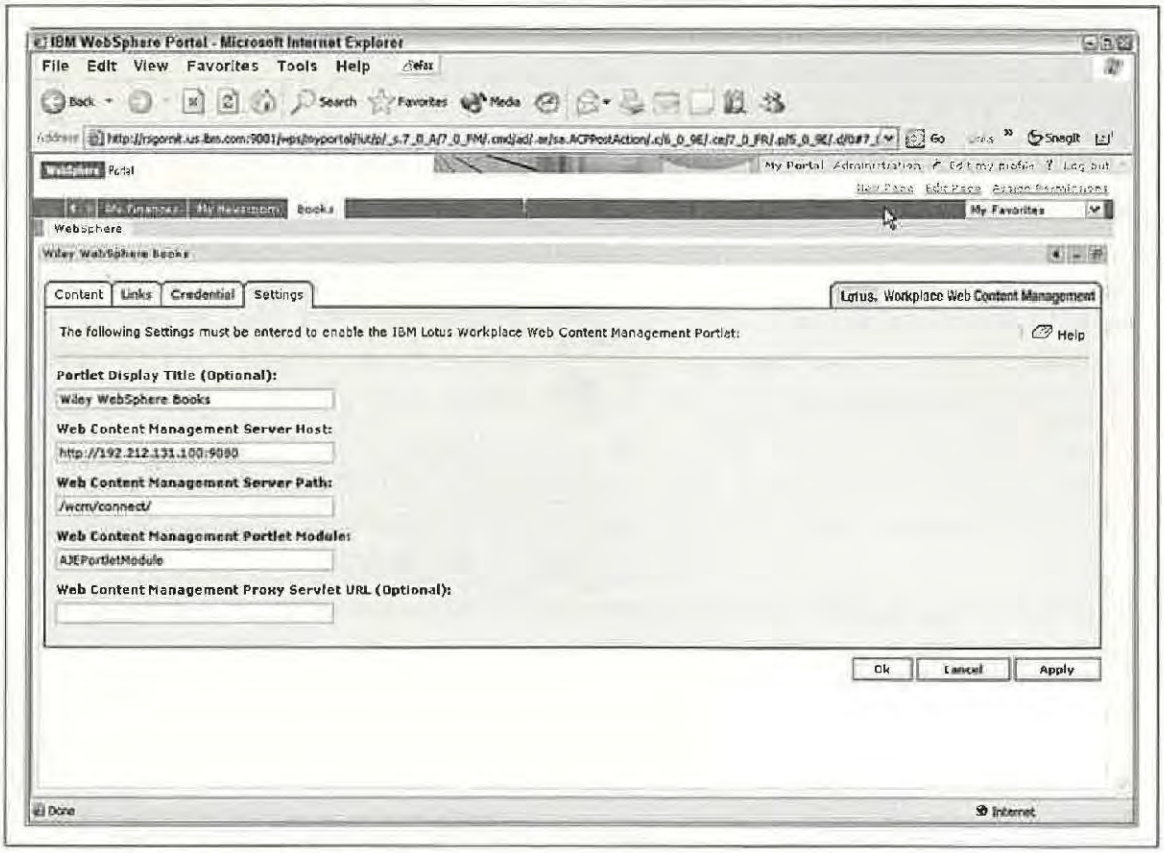


Figure 11-11 Portlet settings.

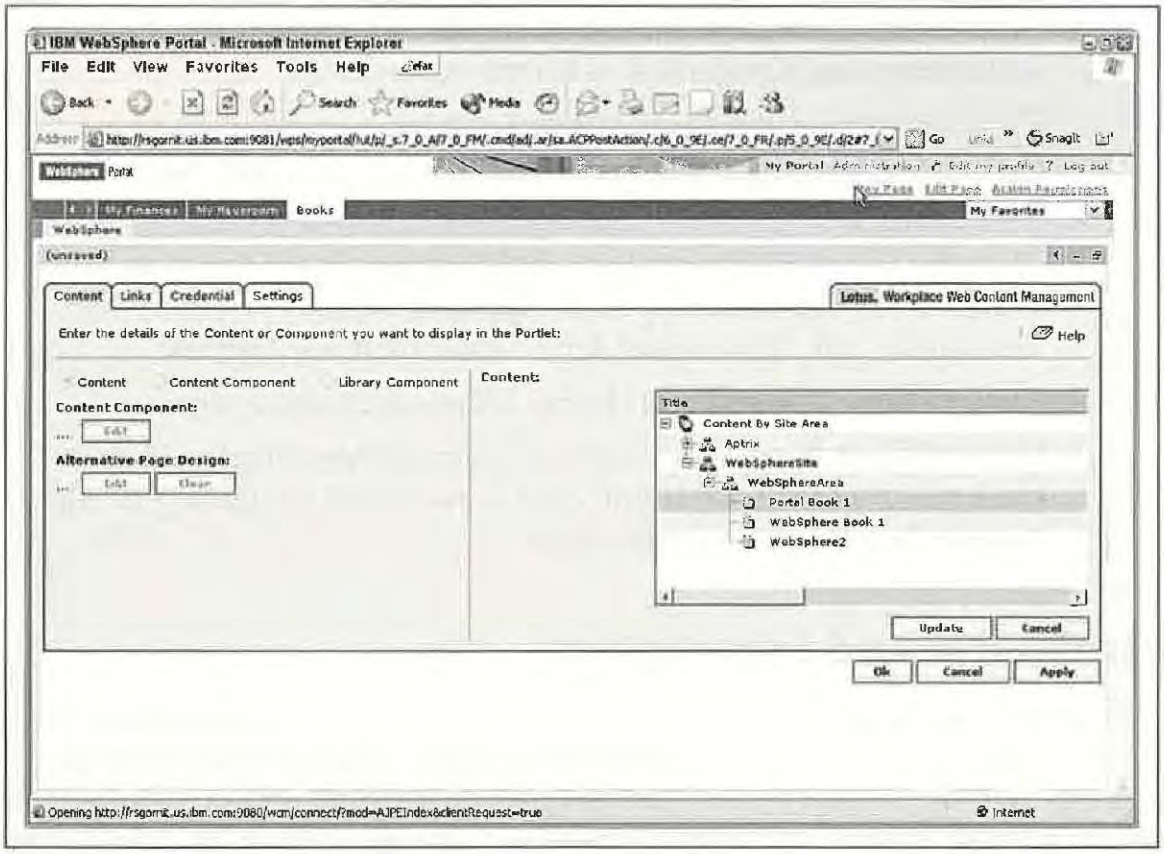


Figure 11-12 Content settings for ILWWCM Portlet Viewer.

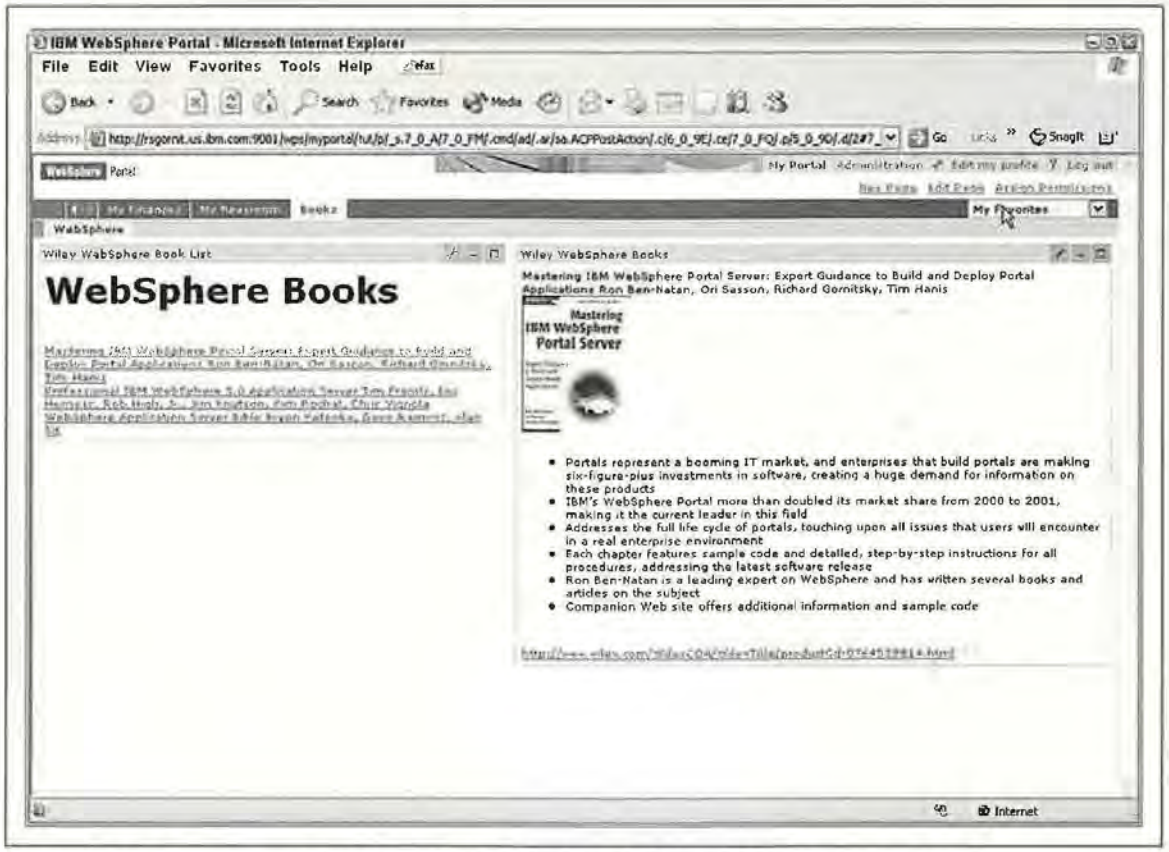


Figure 11-13 Content from ILWWCM displayed through WebSphere Portal.

i. Click Configure on the portlet on the left side. Use the same values for Settings and Credential as for the right portal.

j. Click the Links tab and click Broadcast Links to This Page, so a click the left portlet will cause a broadcast to the right portlet on the same page when an action is taken.

k. Click the Content tab and then click Library Component and Edit. When the Component Type tree appears, expand until you see WebSphereList. This is the Menu component we defined.

l. Click Default Context and choose a default context page.

m. Click OK and then you should see your content displayed as shown in Figure 11-13. If you click a menu link on the left portlet, the detail will appear on the right.

Syndication and Subscription

ILWWCM supports syndication and subscription. Syndication enables the WCM engine to replicate data from one engine to another so that distributed editing and separation of the development and production servers can be supported. Syndicator is the server making the objects available and

Subscriber is the server subscribing to the objects. WCM uses a syndication engine based on the Information and Content Exchange standard.

Summary

This chapter examined the WebSphere Portal Document Manager and ILWWCM, which will eventually replace WPCP. You saw that Document Manager provides some features for creating and managing documents; however, its productivity portlets have limited functionality for most users. With the advent of ILWWCM came a powerful, easy-to-use Web Content Manager that integrates into WebSphere Portal. Only a few of its features are examined in these paragraphs, which should give you a taste of its power. You are heartily encouraged to explore more.

In the next chapter, we will further explore how WebSphere Portal aids employee to employer facilitation by exploring WebSphere Portal's comprehensive collaboration and search components.

CHAPTER

12

Adding Collaboration and Search Components to Your Portals

A portal is by definition a Web site that gives users a single point of access to a variety of resources and tools. In this chapter we focus on two specific types of tools that are of interest to users: collaborative features and search.

The need for collaborative features in a portal is obvious: Any portal, be it a B2B, B2C, or an intranet portal internal to an organization, can benefit from some sort of person-to-person interaction. Furthermore, collaboration is a key ingredient in promoting portal adoption within any organization, as it shifts existing stand-alone applications into the portal boundaries.

The explosive growth of the Internet has given rise to the notion of search engines. One of the main offerings of first generation portals such as Yahoo!® was (and still is) their search capability.

In this chapter you will learn how to make use of the collaborative features and the search capabilities that are provided with the WebSphere Portal Server.

Collaborative Features

The motivation for the collaborative features is to provide physically distributed teams with ways to work in a virtual office environment, through tools such as instant messaging, e-mail, calendar sharing, and information sharing in general. In a more general setting, such tools can be used to develop and support user communities, such as customers or business partners.

The WebSphere Portal Server collaborative components are shipped as part of the WebSphere Portal Extend offering. Namely these are Lotus Sametime 4.0, Lotus QuickPlace 3.0, and Lotus Collaborative Components.

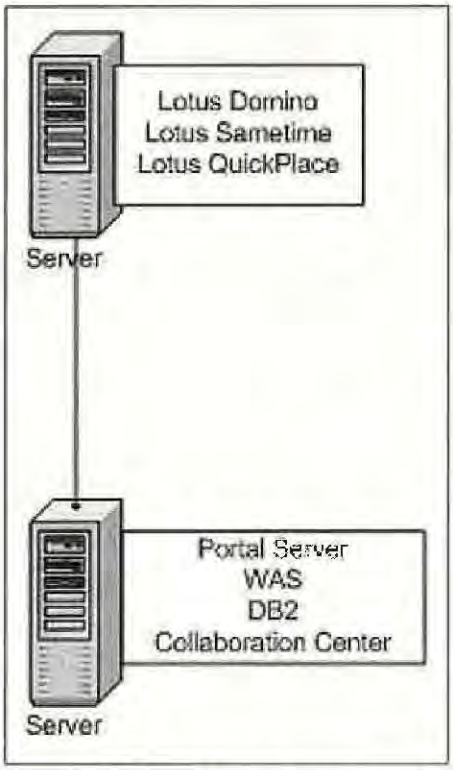


Figure 12-1 Minimal configuration for portal with collaboration.

Lotus Sametime provides a platform for instant messaging and application sharing. It is seamlessly integrated into the WebSphere Portal Server. Lotus QuickPlace facilitates shared virtual workspaces for sharing and organizing data, be it ideas, documents, or schedules. Lotus Collaborative Components provide a Java API to allow developers to implement customized portlets making use of features of Lotus Domino, Lotus QuickPlace, and Lotus Sametime.

The minimal configuration for the collaborative features is shown in Figure 12-1. It is advisable to place the Lotus Domino server and the Sametime and QuickPlace product on a separate machine, to avoid conflicts between the Lotus Domino server and the IBM HTTP server used with WAS.

In a production environment with large quantities of users, it would be more practical to place each product on a separate server (that is, use three different machines for the Lotus Domino, Sametime, and QuickPlace products).

Lotus Notes Integration

With Lotus Notes integration, WebSphere Portal Server provides access to mail, appointments, contacts, and tasks in Lotus. Since these are functions used extensively by nearly every user, placing them inside the portal increases the use and acceptance of the portal.

WebSphere Portal Server provides two different ways for Lotus Notes integration. On one hand you have the Domino Web Access (iNotes) portlet, and on the other hand the Notes and Domino portlet.

The former portlet uses Domino's own iNotes Web access, and consequently it has two distinct disadvantages: It requires browser support from iFrames, and it does not apply the portal's theme to the generated pages.

The Lotus and Domino portlet replaces five different portlets in earlier versions of the WebSphere Portal Server (for example, version 4.2). This single portlet can be configured to provide access to the following types of Notes databases:

- NotesMail—To access a Note Mail database
- NotesView—To access any view of any Notes database
- MyNotesMail—Access to a specific Notes Inbox, based on the login identity
- MyNotesCalendar—Access to a specific Notes Calendar, based on the login identity
- MyNotesToDo—Access to a specific task list, based on the login identity
- NotesTeamRoom—Access views of a Notes TeamRoom
- NotesDiscussion—Access to Notes Discussions

The Lotus and Domino portlet supports single sign-on (SSO) to the Domino server and the Notes database, with an LTPA token. This portlet has the following configuration parameters (accessible through the Manage Portlet entry in portlet administration), as shown in Figure 12-2.

- `NotesServer`—Host name of the Domino server where the Notes database resides
- `NotesDatabase`—Filename of the Notes database used as an information source for the portlet
- `VAULT_SLOT`—Name of the slot in the WebSphere Portal Credential Vault used for this portlet
- `NotesView`—Name of view to use, default is ($All)
- `PortletType`—One of the values NotesView, NotesMail, MyInbox, MyCalendar, MyTodo, NotesDiscussion, NotesTeamRoom.

Note that the default portlet type is NotesView, which is the most general one. While being the most flexible and powerful type, it also requires the most tweaking by users.

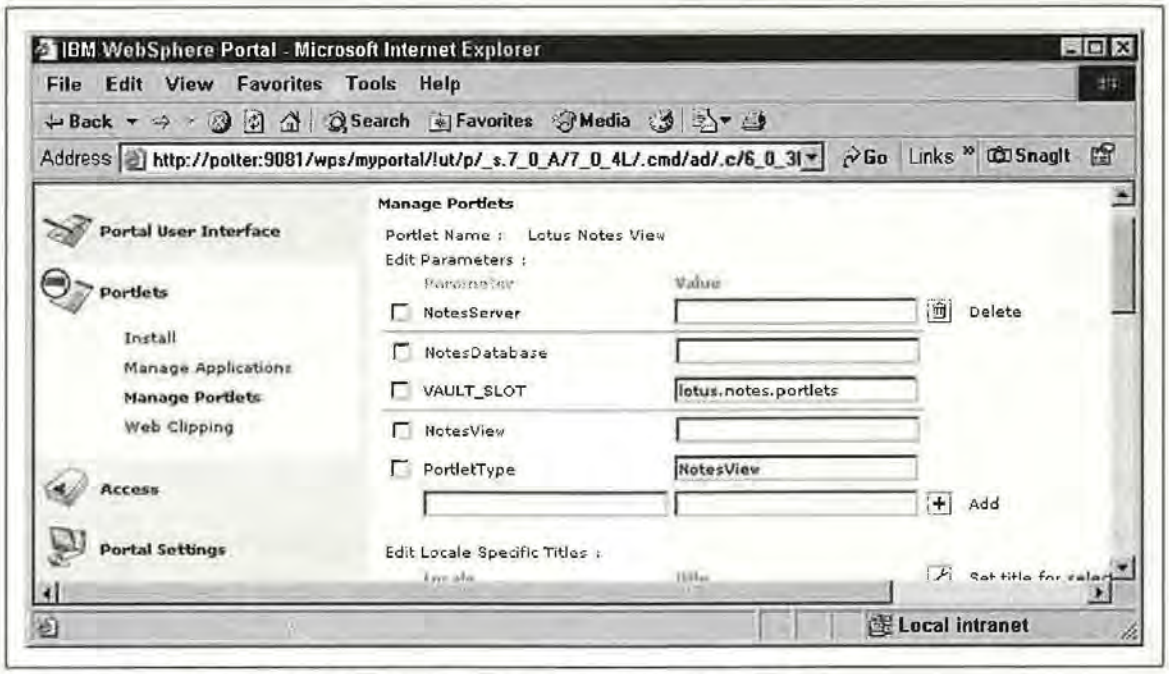


Figure 12-2 Configuration parameters for NotesView portlet.

Microsoft Exchange Integration

Understandably, WebSphere Portal Server has tight integration with Lotus products. Even so, it provides means for integration with Microsoft Exchange. Five different portlets are provided for Microsoft Exchange Integration:

- Microsoft Exchange Tasks
- Microsoft Exchange Notes
- Microsoft Exchange Contacts
- Microsoft Exchange Calendar
- Microsoft Exchange Mail

Each of these portlets provides a respective view into the Microsoft Exchange server.

Before using any of these portlets, users need to customize the portlet, by choosing a credential slot in which their user ID and password shall be stored, and by entering the details of the exchange server as shown in Figure 12-3.

Lotus Sametime Integration

Lotus Sametime allows teams to collaborate through instant messaging and online meetings. WebSphere Portal Server provides a Sametime portlet which allows users to launch the Sametime connect applet.

The Sametime portlet is shown in Figure 12-4. There are not a lot of configurable parameters for this portlet, just the host and port number. Once

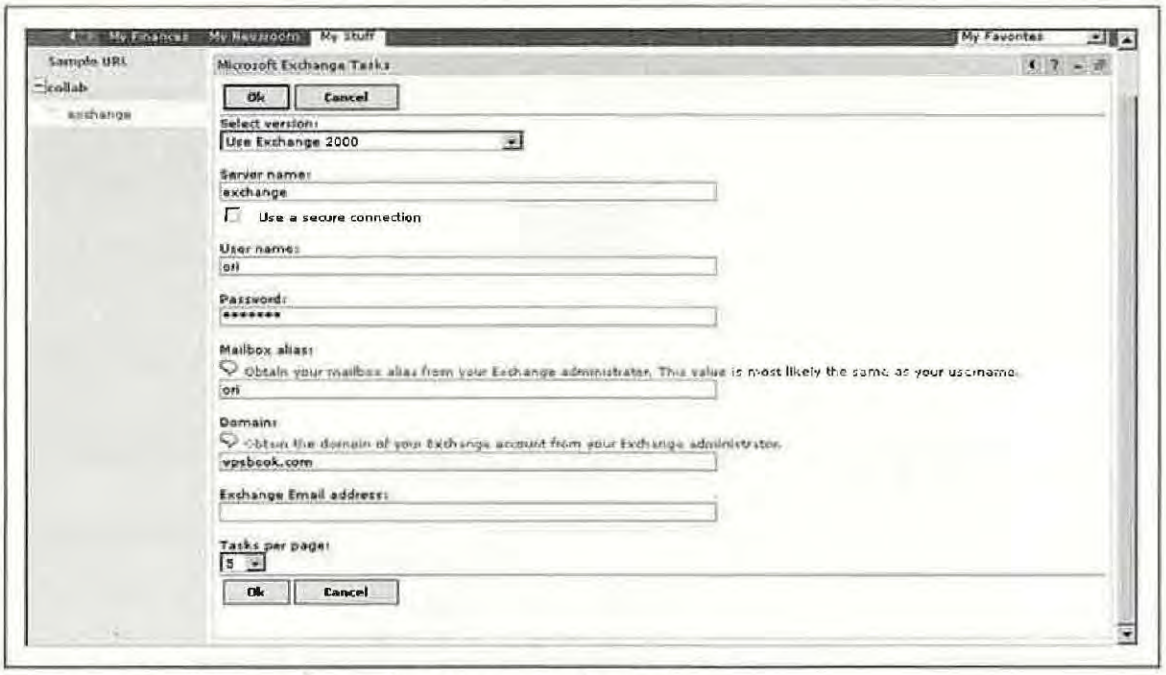


Figure 12-3 Editing Microsoft Exchange Tasks portlet.

you launch the Sametime connect applet, you need to log in to Sametime (as shown in Figure 12-5).

An important advantage of using Sametime through the portal is single sign-on. In other words, users need not log in to Sametime; instead their session with the portal is extended to Sametime, using the same username. In order for SSO to take place, you need to configure Sametime, by importing an LTPA token exported by the WebSphere Application Server.

Once logged into Sametime, users can send instant messages and start electronic meetings. The Sametime user interface is straightforward, and resembles most other instant messaging products. Each Sametime user has a contact list. The contact list typically contains individuals with whom the

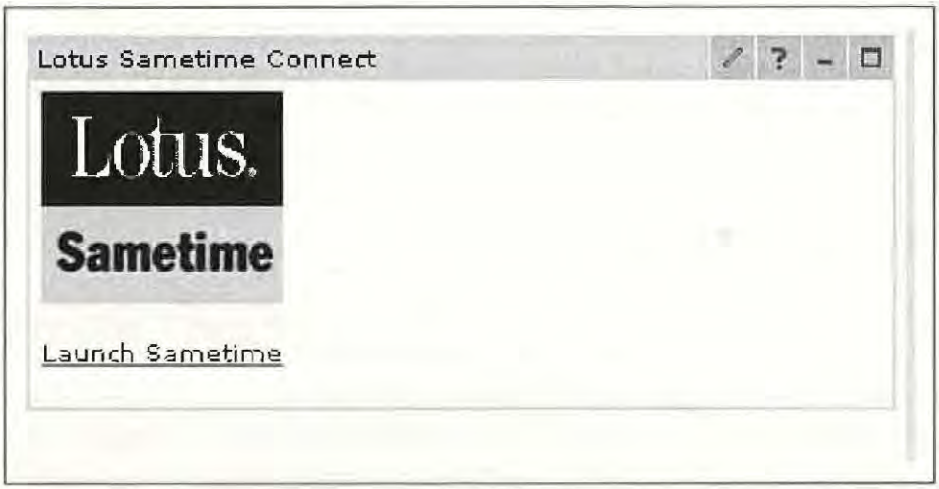


Figure 12-4 Lotus Sametime portlet.

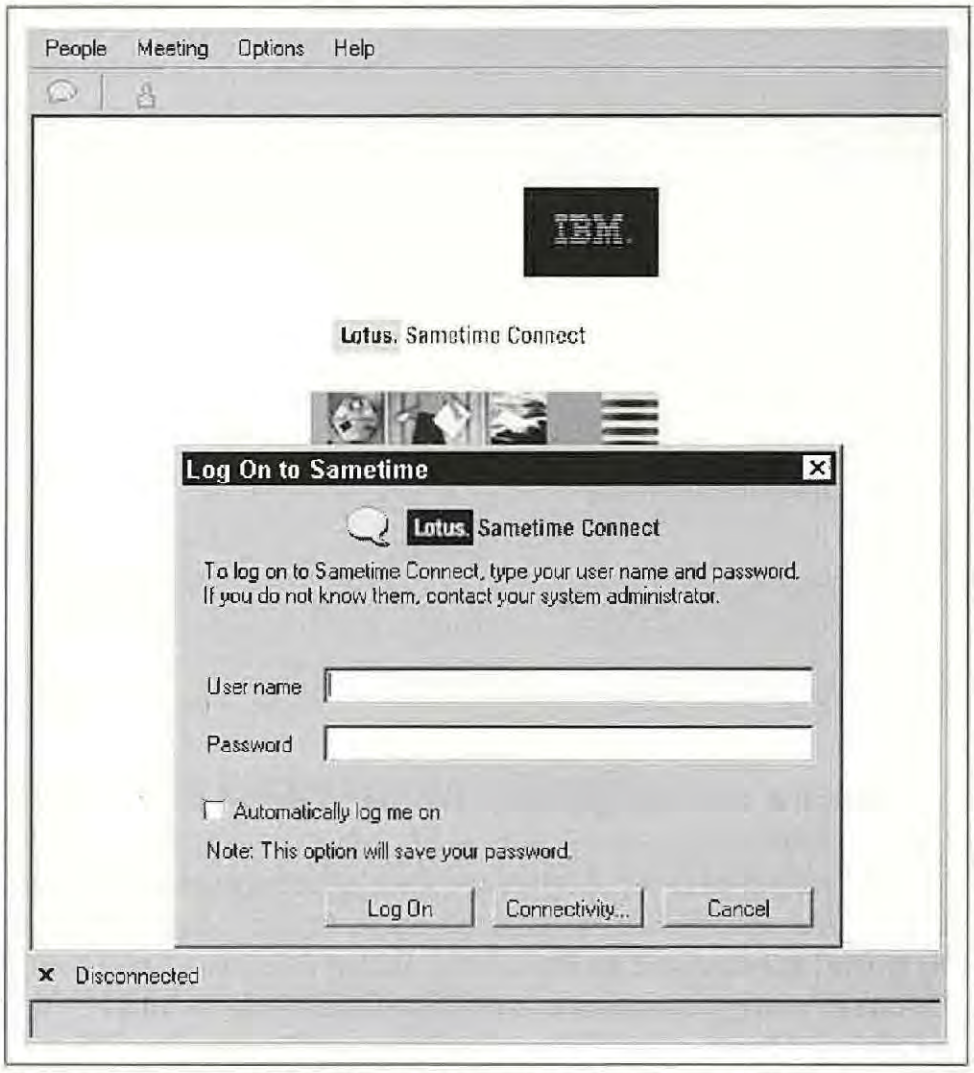


Figure 12-5 Lotus Sametime Connect, launched from portlet.

user intends to interact with frequently. Users are able to set their online status, to indicate whether they are available. Through the Meeting menu it is possible to initiate a variety of communication modes: text-based chat, audio, video, whiteboard, or application sharing. Once a meeting is initiated, other people (either from the contact list or outside of it) may be invited to join in.

Lotus QuickPlace Integration

Lotus QuickPlace (now known as IBM Lotus Team Workplace) allows teams to establish a team workplace to facilitate collaboration. Workplaces can store documents, discussions, plans, tasks, and resources.

QuickPlace provides tools for asynchronous collaboration, in contrast to the online and synchronous collaboration tools of Sametime.



Figure 12-6 Lotus QuickPlace portlet.

WebSphere Portal Server is shipped with two QuickPlace portlets. The QuickPlace portlet shown in Figure 12-6 is used to direct users to one or more QuickPlace places. Users may customize the portlet in the edit mode by specifying up to six different places. For each place the name and URI need to be specified. The inline QuickPlace portlet is associated with a specific place, which launches immediately within the same page.

When new places are created, users sharing the place are specified along with their respective access rights.

Microsoft NetMeeting Integration

WebSphere Portal Server includes a portlet for launching a Microsoft NetMeeting client. The client is invoked through the NetMeeting Active-X component, which can be embedded into a portal page.

The NetMeeting portlet is not installed by default in the portal, and it is located in the `netmeeting.war` file. This portlet does not have any configuration parameters, as the user profile is entered through the `NetMeeting` interface.

People Finder

A prerequisite for any sort of collaboration is finding the people with whom you want to collaborate with. While that is a nonissue in a small to medium-sized organization, it is much more difficult within a larger organization.

WebSphere Portal Server is shipped with the People Finder portlet. It is not installed by default, but is available in the `PeopleFinder.war` Web module.

In order for the People Finder portlet to work, you need to have the Directory Connector Web application (LDAPConnector.ear) to be installed. This is automatically done by the Collaboration Center installation program. The Directory Connectors bridges between the portal server and your LDAP directory. The Directory Connector uses the data store of the underlying WAS (be it the default CloudScape, DB2, or Oracle) to store data.

The Directory Connector uses an XML configuration file to define the LDAP server to which the People Finder portlet connects to. The Collaboration Center installs a sample file `PFSampleConnection.xml` under the directory `CollabCenter` under the portal installation root. You may direct the Directory Connector to use another XML file by opening it at

```
http:// yourHostName/PFDirectoryConnector0
```

Here you can load new XML files, validate them (that is, make sure that they are well formed), and activate them. The XML file defines the connection settings, as well as configuration data relating to the data stored in the LDAP server.

The connection settings include the following elements, which must be set correctly for the People Finder portlet to work:

- hostURI—Fully qualified LDAP URI (that is, `ldap://ldapServer:ldapPort`).
- baseDN—The starting point for any searches (that is, the topmost node from which a search will start).
- bindDN—Name of user used to access the LDAP server. If empty, anonymous bind is made.
- privateCredentials—Password, if applicable
- securityProtocol—`none` if anonymous binding, or `simple` if using a password.

The XML file indicates what LDAP attributes are exposed to the People Finder portlet. For each exposed attribute a meaningful name can be given. For example, the first attribute in the sample file is

```
<exposedAttribute label="DisplayName">
     <description/>
     <mappedAttrbiute>displayName</mappedAttribute>
     <type>String</type>
</exposedAttribute>
```

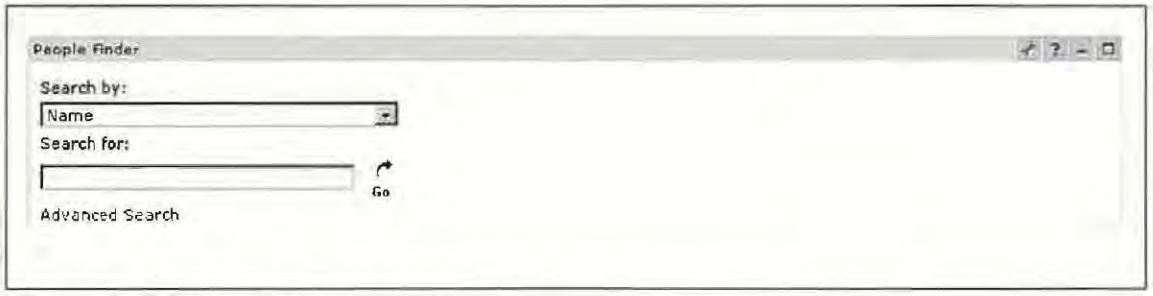


Figure 12-7 People Finder portlet.

The label controls the naming of the attribute in the People Finder portlet, and the `mappedAttribute` indicates the respective LDAP attribute.

The XML file can also specify exposed filters, custom queries, and recursive queries. Filters are used to limit the results given. For example, the default XML file includes a filter `objectclass=person` to ensure that only person records are retrieved. Custom queries are used to define queries that look at more than one attribute, for example searching in several name fields. Recursive queries allow traversing the directory recursively. In the sample XML file there is a query `ReportsTo` which recursively searches for the managers of a specific person.

Once the LDAP connection is configured, it is possible to use the People Finder portlet, by adding it to any page, as shown in Figure 12-7.

The People Finder portlet is customizable in several ways. To configure it click the wrench button on the portlet title bar. The configuration window is shown in Figure 12-8.

In `Configuration Basics` you can connect the People Finder portlet to connect the portlet to the LDAP directory by selecting one of the loaded XML configuration files, and to invoke the Directory Connector application. You can also choose what fields to display and for each such field its display format (for example, as a mailto link for mail addresses or as a Web page link for home pages).

In `Shared Elements` you can customize the business card display used by the People Finder portlet, as well as define sections where information shall be placed. Sections apply to both the Person Record and Advanced Search. Default sections are Contact Information, Current Job, and Background, and of course you can add, rename, or remove sections.

`Person Record` allows you to customize the Person Record display and the Organizational View. You can rearrange the fields, and place fields in the various sections defined.

The `Quick Search` and `Advanced Search` entries allow you to customize the search options, by choosing the fields available and the output fields.

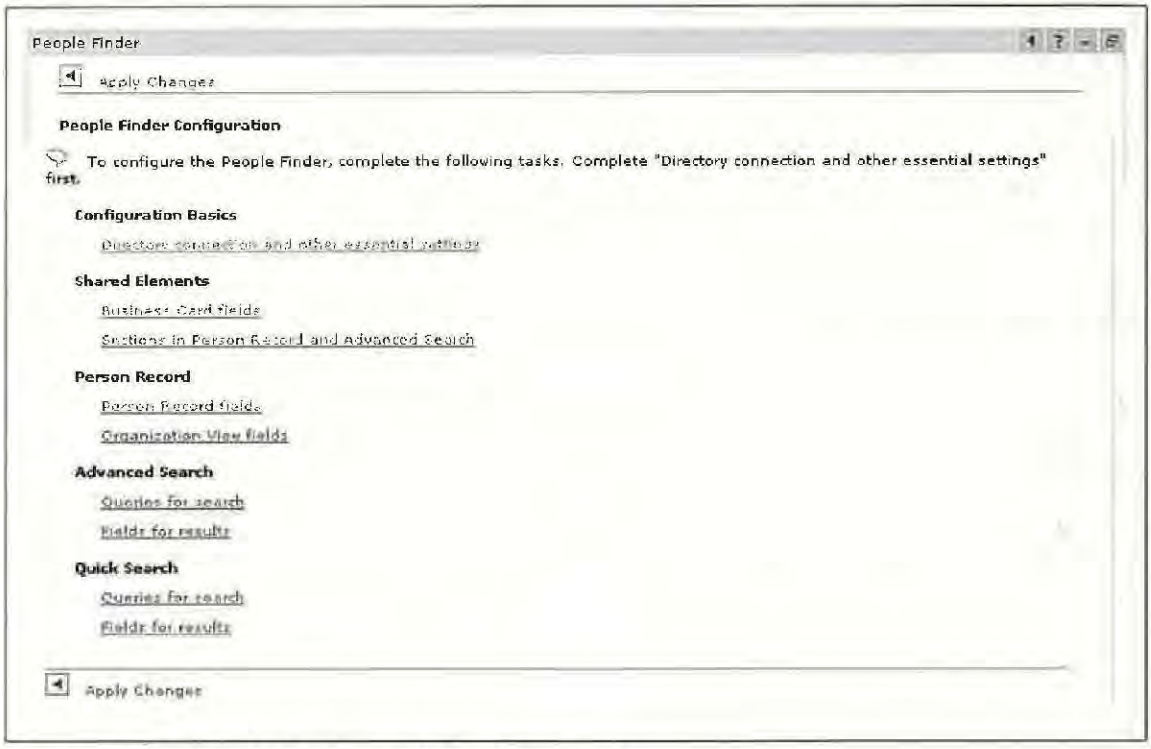


Figure 12-8 People Finder portlet configuration.

Collaborative Components

In addition to the out-of-the-box collaboration provided by the portlets described here, WebSphere Portal Server includes the Lotus Collaboration Components. These components are a set of Java APIs and taglibs for integrating the functionality of Lotus products into portlets. The Lotus products supported are Lotus Domino, Lotus Sametime, Lotus QuickPlace, and Lotus Discovery Server.

In contrast to the previous sections, the current one requires some basic understanding of JSP. In addition, to actually make use of the Collaborative Components, you need to develop and deploy portlets, which you will learn later in this book.

The Collaborative Components Java APIs are packaged in the file `cs.jar`, whereas the taglibs available are `people.tld` for people awareness and `menu.tld` for contextual menus.

The entry point to the Lotus Collaborative Components is the class `CSEnvironment`. To start using the Collaborative Components from within a JSP scriptlet, you need the following two lines of code.

```
CSEnvironment env = CSEnvironment.getEnvironment(request);
CSCredentials cred = env.getCredentials(request);
```

The first line initializes the Collaborative Components and sets the credentials for the currently logged-in user. The second line retrieves these

credentials. The credentials are used to instantiate service objects. The following are the service objects available:

- Domino Service—Provides access to Domino databases
- DiscoverServer Service—Retrieves data from the Lotus Discovery Server
- PeopleService—Provides access to information about a person (aggregates information from Domino Service and DiscoveryServer service)
- Calendar Service
- QP Service—Provides access to QuickPlace places

Service objects are instantiated using the object `CSFactory`. For example, the following line of code instantiates a Domino Service.

```
DominoService ds = (DominoService)
                CSFactory.getService(cred, CSFactory.DOMINO_SERVICE);
```

In this case the Domino Directory server is specified in the `CSEnvironment.properties` file, located in the `Appserver\lib\config` directory under your Application Server installation root. It is also possible (though not advised) to use the `getService` method with a hard-coded server name and protocol.

Given a `DominoService` instance you can access Domino databases through views. For example, the following line of code retrieves a full view of a Lotus Domino address book called `names.nsf`.

```
ViewInfo vi = ds.getViewInfo("names.nsf", "($all)");
```

With the `ViewInfo` at hand, we can generate a list of names in the view. A typical use of names would be to provide people awareness. In other words, allow portal users to see whether other users are online and to interact with such users. To that end, the Collaborative Components provide the `people.tld` taglib. This taglib introduces the `peopleservice` tag. When you use this tag, a link menu is generated, allowing users to interact with the specific user.

The following piece of code combines the information extracted from the `ViewInfo` with the `peopleservice` tag to generate a list of names that can be clicked to initiate contact with the respective person. For simplicity, we retrieve only the first 10 names out of the address book.

```
<%
RowInfo[] ri = vi.getRowInfo("1", 10);
int i;
String values[];
String name;
for (i = 0; i < ri.length; i++) {
     values = ri[i].getValues();
```

```
        name = values[0];
%>
<br><peopleservice:person><%=name%></peopleservice:person>
<%
}
%>
```

Note that the `peopleservice` tag can be used directly without any use of the DominoService. For example, the following JSP shown within a portlet will provide a list of the authors of this book, with a menu for communicating with them. The actual entries shall be available only if the individuals can be located through the definitions of the `CSEnvironment` properties.

```
<%@ page language="java" import="com.lotus.cs.*"^>
<%@ taglib uri="/WEB-INF/tld/people.tld" prefix="peopleservice" %>
<%@ taglib uri="/WEB-INF/tld/menu.tld" prefix="menu" %>

<TABLE>
<TR><TD><peopleservice:person>Ron Ben-Natan</peopleservice:person>
</TD></TR><TR><TD>
<peopleservice:person>Richard Gornitsky</peopleservice:person>
</TD></TR><TR><TD>
<peopleservice:person>Tim Hanis>/peopleservice:person>
</TD></TR><TR><TD>
<peopleservice:person>Ori Sasson</peopleservice:person></TD></TR>
</TABLE>
```

If Lotus Sametime is enabled, each of these lines will show a status icon for the respective person, indicating his online status (for example, Active, Away, Offline). Clicking the person link will show a menu. The menu options depend on the system configuration. With Sametime enabled, it is possible to initiate instant messaging with the user, or add him to the Sametime contact list. If Lotus Discovery server is enabled, it is possible to send e-mail to the specified user, display the user profile, and find documents authored by him. If the People Finder portlet is available within the page, it is also possible to invoke it to show the person record or his location in the organizational view.

The `menu.tld` taglib is used by the `people.tld` taglib to implement pop-up menus. You may use it for any purpose other than collaboration, by using the `menu` tag and implementing JavaScript callback functions within your portlet.

Search Capabilities

With the huge amount of information stored both within the corporate intranet and the Internet, efficient searching is an important ingredient of the capabilities you expect to have in a portal.

IBM offers search capabilities in all versions of the WebSphere Portal Server. The Portal Server has a built-in search engine that supports full-text searching. The search engine allows users to specify the + and – query operators to indicate words that must be in the document or words that must not be in the sought document, respectively.

In order to keep down the search time, some preprocessing is required. To that end, the search engine building indexed document collections, which can then be searched.

To enable search in your portal, you need to prepare a document collection. The document collection is essentially the database of documents to be searched. The procedure for creating a document collection was covered in Chapter 10.

Searching a Collection

Once you have prepared a document collection, you are ready to provide search from within your portal. The Portal Server comes with a preinstalled search portlet called `Document Search`. This portlet is configured to use a specific document collection. Therefore, you need to create a copy of this portlet for each collection you want to use. You can avoid this only if you have a single collection, in which case you can configure the preinstalled instance of the portlet to use that collection.

To configure the portlet, follow these steps:

1. Log in to the portal as **Administrator**.
2. Enter the portal Administration section.
3. Go to Manage Portlets under Portlets.
4. In the Portlets list select the portlet Document Search.
5. Click Copy (on the right of the list).
6. Click Modify Parameters (on the right of the list).
7. In the IndexName field, type the path of the document collection, for example **C:\tmpindex**.
8. Click Save.
9. Click Activate (right of the portlet list).

Now you have a new instance of the Document Search portlet configured to use the index created in the previous section. To actually use it, insert this portlet into any page. Note that typically it would be at the end of the portlet list, since it was just created.

Once the portlet is placed within a container, it is ready to use. To search it, users need to specify text in the Search For field, and click the Search button. The portlet has an Advanced Search mode where users can limit the search

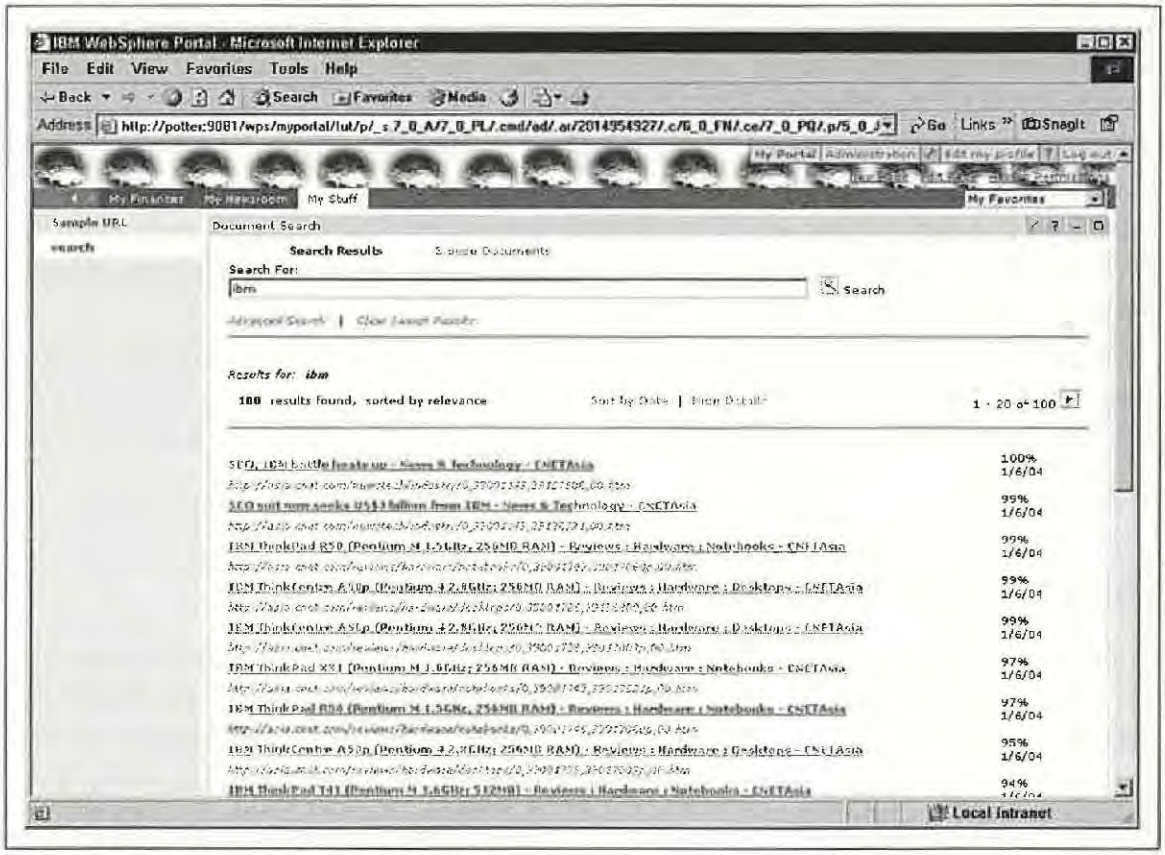


Figure 12-9 Sample Search Results.

to a specific part of the document (for example, Title, Keyword, Subject), and to limit document types. Figure 12-9 shows the result of searching for **ibm** within and index created from the URL `http//asia.cnet.com`.

Managing Taxonomy and Categorizing

In order to facilitate more efficient searching, the portal search supports categorization of documents. When documents are collected, they may be categorized. The basis for categorization is either a predefined static taxonomy or a dynamic user-defined taxonomy.

The predefined static taxonomy covers over 2,000 subjects in a variety of domains. You may customize this static taxonomy based on the needs of your organization. To that end, a Taxonomy Manager portlet is provided, which allows you to add synonyms to existing nodes, or to insert new nodes. New nodes may either be interior nodes, reflecting concepts and abstractions, or product nodes (reflecting specific products or terms relevant to your organization).

User-defined taxonomies have the advantage of being completely flexible, but the obvious downside is that you need to spend significant effort in

defining the various categories. You will also need to define the associated rules.

For each document collection, you can define whether it uses a categorizer (see the document collection parameters as shown in Figure 12-9), and if it does whether it is predefined or user-defined.

When a categorizer is used, it eases the task of browsing through the documents, as well as the Advanced Search which allows focusing on specific categories.

Extended Search

The WebSphere Portal Extend ships with the IBM Lotus Extended Search product. It provides a search engine that is more powerful than the one integrated into the Portal Server. Extended Search supports a variety of data sources, ranging from text documents, and document repositories such as Lotus Notes, to fully blown RDBMS such as DB2 and Oracle.

Extended Search is installed separately from the WebSphere Portal, and once installed, you are able to define data sources. For each data source, Extended Search supports field mapping to allow common naming for fields that have different names in different subsystems.

IBM offers a portlet for using Extended Search from within the portal. This portlet allows the user to query using Web-style querying (using + and −), using GQL (Generalized Query Language, which is a language used by the Extend Search internally), or using native syntax when searching in databases (for example, SQL query syntax when searching an RDBMS).

Summary

In this chapter you learned how to use WebSphere collaboration and search capabilities within the Portal Server. These capabilities allow you to seamlessly integrate existing collaboration tools such as Lotus Notes or Microsoft Exchange into your portal. In addition, Lotus QuickPlace and Lotus Sametime, which are shipped with the Portal Extend offering, can be used to facilitate people awareness, instant messaging, and common workplaces. This chapter concludes this part of the book, and by now you should be familiar with the Portal Server basics. In the next part of the book you will learn how to develop new portlets. New portlets are required whenever you want to implement functionality that is not available within an existing portlet.

PART

III

Portlet Development in WebSphere Portal

CHAPTER 13

Extending Portal Functionality: Portlets

In this chapter you will learn about portlets, what they are, how they work, and how they are managed by WebSphere Portal. We will discuss the portal environment in which the portlets operate and some key portlet objects and services that portal provides. In addition, we will discuss the portlet life cycle, relevant aspects of the portal container, page aggregation and the differences between portlet applications and portlets, abstract and concrete portlets, and portlet instances. We will conclude the chapter with a discussion about the portlet event model and two-phase portlet processing.

What Is a Portlet?

Portlets are distinct, functional components of a portal. They are implemented in Java and are based on a well-defined portlet API. They are deployed through portal, packaged as a Web application, and are managed by portal through a portlet container. They process HTTP requests forwarded through the container and generate dynamic content in response.

From a user's perspective, portlets are functional entities represented on a portal page. A portal page typically contains several portlets that are arranged on the portal page in a grid of rectangles of varying size. Common examples of portlets are portal applications that show the local weather forecast, the stock ticker, or a user's email.

A portlet rendered on a page contains a title bar showing the title of the portlet, portlet navigation and behavior icons, and the dynamic content generated by the portlet. The content generated by the portlet is a portion (fragment) of the total markup that makes up the portal page. All portlets

on the page contribute these markup fragments. The portlet navigation and behavior icons allow the user to minimize or maximize the portlet and to change portlet modes. We will discuss portlet modes and states in more detail later in this chapter. In addition to portlet content, the portal page consists of headers, banners, and a navigation area.

Portlets tend to implement very specific tasks with less page navigation and less complexity than a typical Web-based application may have. From a development perspective portlets are very similar to servlets. A servlet is an application within a Web application server. A portlet is an application within a portal.

From a deployment perspective portlets are packaged as J2EE WAR files and deployed into the portal environment using portal administrative utilities or administrative portlets.

Portlets benefit from functions and services of the portal, which provides a common user interface (UI) framework, a common user interaction model, single authentication, and common administration. Developers benefit by gaining access to supporting portal infrastructure for personalization, customization, and administrative functions such as security, access control, UI skins and themes support, and page layout.

Figure 13-1 shows the portal architecture. The portal receives the client HTTP request for a portlet page. Using the portal user and access control information the portlets on that page are determined. Each portlet is invoked to generate its content fragment in context of the requesting device type. Those fragments are aggregated in context of the device type and the

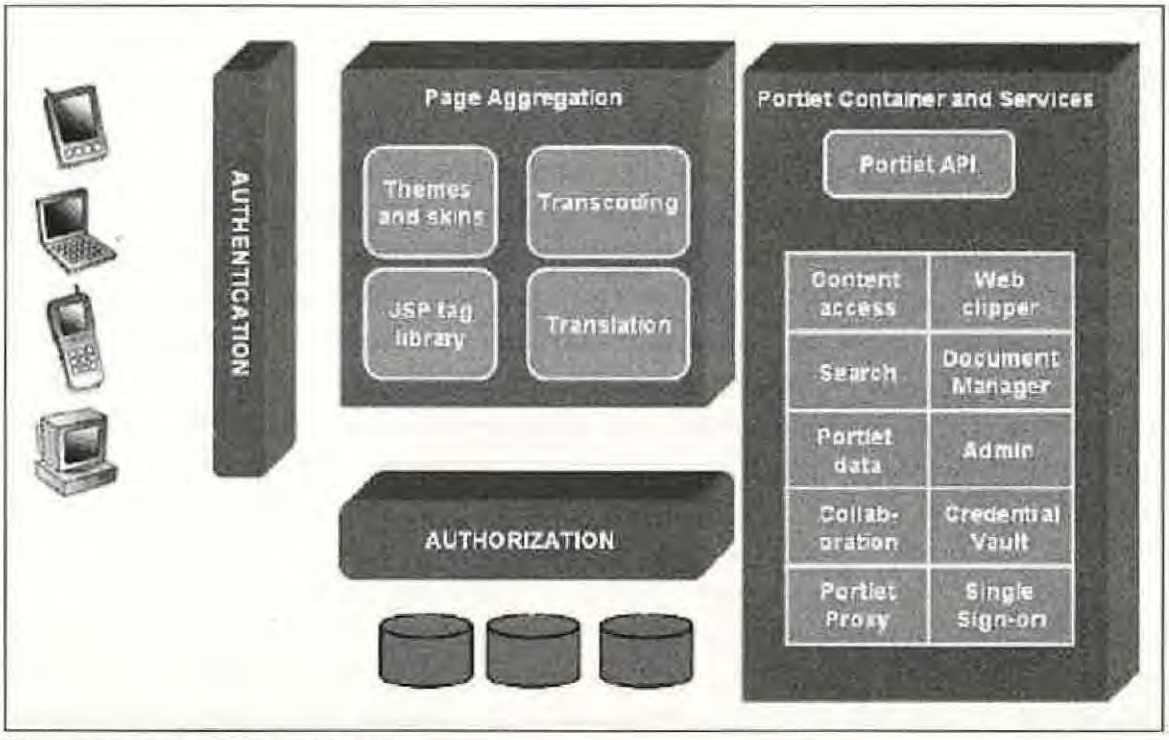


Figure 13-1 Portal architecture.

theme and skin definitions. The portlet container is the portion of the portal server that provides the runtime environment for the portlet.

The portal environment supports multiple device types and multiple markup languages. So, well written portlets that could be invoked from a variety of device types will respond with the appropriate markup and the portal will use markup specific aggregators to generate the correct markup response. These could be HTML, cHTML, or WML.

Portlet Container

Portlets run in the portlet container. The portlet container implements the portlet API and provides a runtime environment where portlets are instantiated, invoked, and destroyed. Through the API, portlets can access the logged-in users' profile, respond to events, communicate with other portlets, gain access to portal services such as the Credential Vault and content access service, persist application data, access portlet setting information, and access the portlet session object, the portlet request object, and the portlet response object.

WebSphere Portal provides two portlet containers that support two different portlet APIs. There is support for the `javax.portlet` API which is defined by JSR 168 and also support for the proprietary portlet API as it was defined by WebSphere Portal 4.1. All portlets written prior to Portal version 5 used only this API.

As shown in Figure 13-2, the portlet containers are components of the portlet environment. Portal itself runs as a Web application in WebSphere Application Server. The portal engine in that Web application talks to the portlet environment through the portlet invoker API and the information provider SPI. As portlets get invoked by the portal engine, they can communicate back to requesting portal information through the portlet API. The implementation of that API may need to gain access to the portal engine through the information provider SPI.

A portlet request will get directed to the correct portlet container depending on the implemented API of the portlet. The portlet then responds back with its markup fragment, which is passed back through the portlet container to the portal engine for aggregation.

Page Aggregation

One of the key differences between a portlet and a servlet is page aggregation. In a servlet the generated markup is solely the responsibility of the servlet to define and generate. In a portlet the portal aggregator is responsible for generating the resulting markup with assistance from the

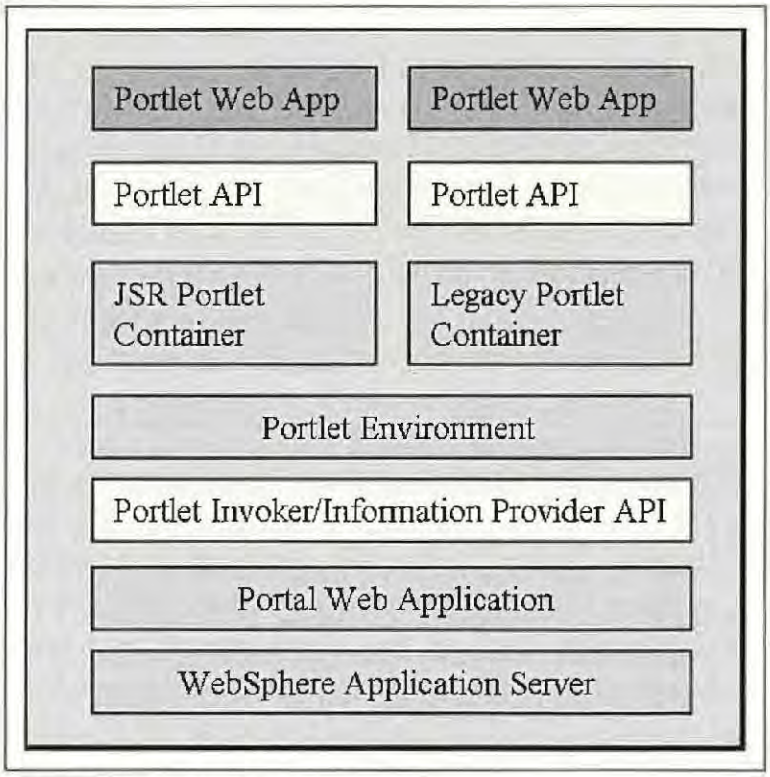


Figure 13-2 Portlet container.

portlets that are on the page to be rendered. Portlets have to be cautious that any generated markup is going to be in context of the markup generated by the aggregator and in context of the markup of other portlets on the page. For example, JavaScript that is generated by a portlet may conflict with JavaScript generated by other portlets contributing to the same portal page. Care should be taken to minimize the risk of name collisions.

Portlet Modes and States

Portlets inherit common UI behavior from the portal. One of these is the ability for users to select a portlet to be maximized or minimized. When maximized the portlet takes over the entire portal page. When minimized the portlet is reduced to just the title bar. This is known as the *state* of the portlet. The states that are allowed for a portlet are defined in the portlet deployment descriptor as a property of the abstract portlet.

The portal also provides infrastructure for common portlet behavior. Portlets often have requirements for runtime configuration and customization. Consider a portlet that shows the local weather. Typically we would expect that a portal administrator would deploy the portlet and provide portlet configuration. In this case, the portlet would probably be configured to specify the source of the weather information. The portal user would then need to customize the portlet to specify their location, probably in the form

of a zip code or city and state. Then the deployed portlet can successfully execute for all portal users customized for each user but configured by an administrator for common settings such as the weather source provider. We will see later where to store this kind of information so that it is shared in one case but local to each user in another.

So, given this scenario and given that it is common behavior for many portlets, we would like the portlet container and portlet API to provide the common infrastructure to remove some of that work from the portlet developer and also to make sure that we have a consistent user interaction model.

The first commonality is the need for the portlet API to implement portlet modes. The defined modes are view, configure, edit, and help, as shown in the sample portlet control icons in Figure 13-3. All portlets support view

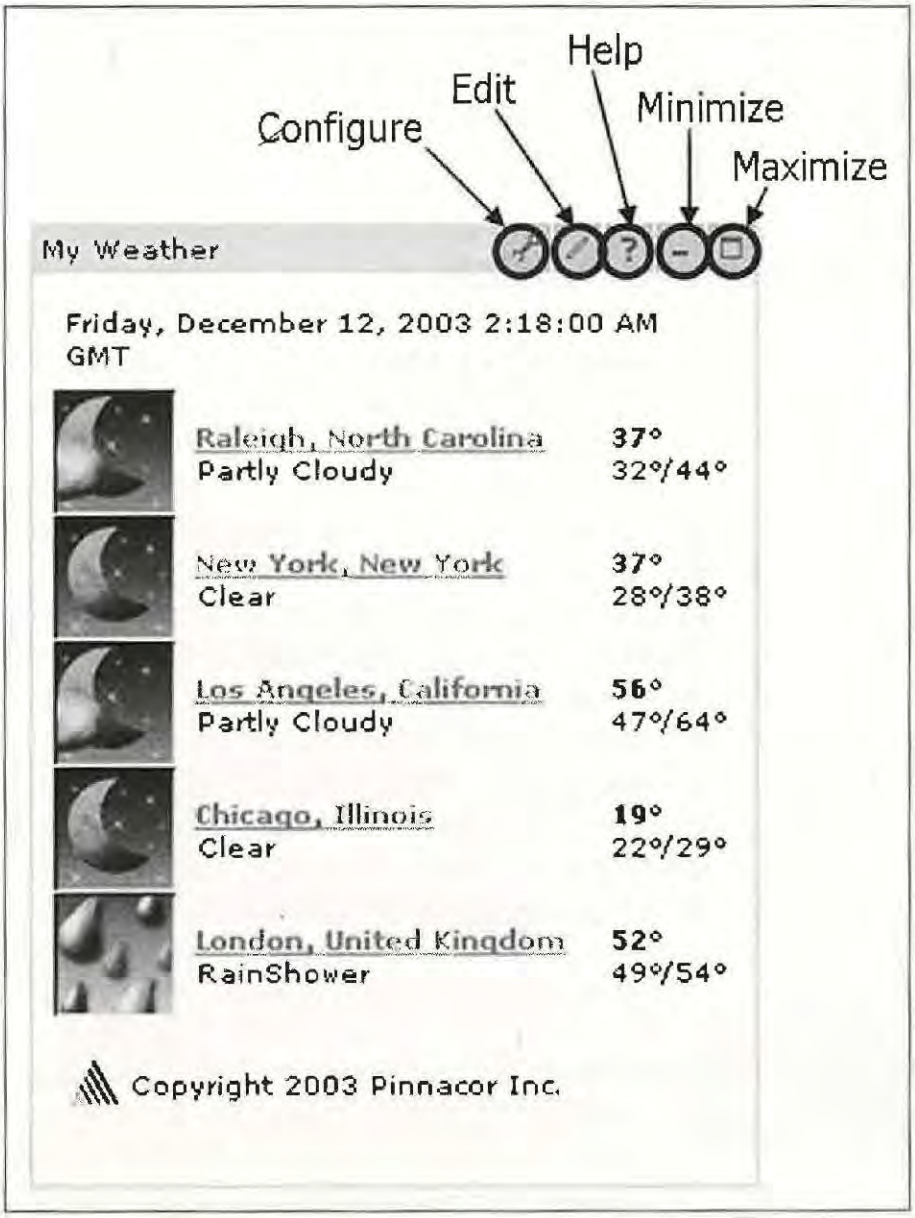


Figure 13-3 Sample portlet.

mode and this is the initial mode when the portlet is displayed. View mode is the main mode of the portlet. Configure mode is for shared portlet configurations as we discussed. Edit mode is for user-specific portlet customizations. Help mode is naturally for displaying portlet help.

The other common function requirement here is the need to make a distinction on the user's access rights not just at the portlet level but for the portlet and for modes within the portlet. We want to allow all users access to the portlet's view, edit, and help modes but give only portal administrators the ability to execute the configure mode of the portlet.

Keep in mind that it is up to the writer of the portlet to determine what, if any, configuration and customization processing needs to be accomplished, and implement it accordingly. Portal provides the infrastructure to associate access control permissions with portlet modes, to provide the Portlet API with mode support, and provides access to portlet settings, portlet application configuration parameters, and servlet initialization parameters. It provides access to a persistence store for user-specific application data. It is up to the portlet writer to determine which values should be shared and which should be user-specific.

Portlet Applications and Portlets

Portals are defined within the context of a portlet application. A portlet application may contain one or more portlets. Portlet applications are used for administrative management of a set of portlets.

Portlets are defined in the portlet deployment descriptor, an XML file that gets consumed by portal during the process of installing or updating a portal application. As we said before a portlet is deployed in a Web application so we also have a Web deployment descriptor for our portlet application. The Web application deployment descriptor defines one or more servlets and each of those servlet definitions is tied to a portlet definition in the portlet descriptor (via a mapping of the servlet `id` to the portlet `href`).

The portlet deployment descriptor defines portlet properties such as portlet display name, caching preferences, supported portlet states, modes, and markups, and portlet configuration parameters.

Portlet application properties are minimal and include the display name and an application unique identifier. A portlet application will have one or more concrete portlet applications defined in the portal deployment descriptor.

A concrete portlet application is differentiated by a set of Context Parameters that are shared across its concrete portlets. One or more concrete portlets are defined in the portlet deployment descriptor. They are mapped to the abstract (nonconcrete) portlets by the `href` attribute of the concrete portlet

mapping to the id attribute of the abstract portlet. Concrete portlets are differentiated by a set of additional properties that include the defined language locales (and default), the portlet-language-specific portlet title, short title, description and keyword properties, and defined setting parameters.

Although the portlets and servlets have many similarities, there are some key differences. Portlets cannot issue a redirect or forward a request. Since portlets only contribute to the response markup output stream rather than write the entire response, they have some limitations in what markup they can generate. In general, portlets should generate only markup that could be contained within a table. A portal provides dynamic administration of portlets, and dynamic updates can be made to change the portlet settings, to copy portlets, and to delete portlets. Portlet applications are installed into a running portal.

Figure 13-4 shows the relationship of portlet application to Web application and the attributes associated with portlet applications, concrete portlet applications, abstract portlets, and concrete portlets.

While administrative tasks can target portlet application and concrete portlet applications, only concrete portlets are put on pages to be used by the portal user.

The portlet class that implements the required portlet API and provides the function of the portlet is defined as the servlet class in Web application deployment descriptor. Portlet application and portlet configuration information is linked from the Web descriptor and the implementation of the portlet is invoked from the Web descriptor servlet class.

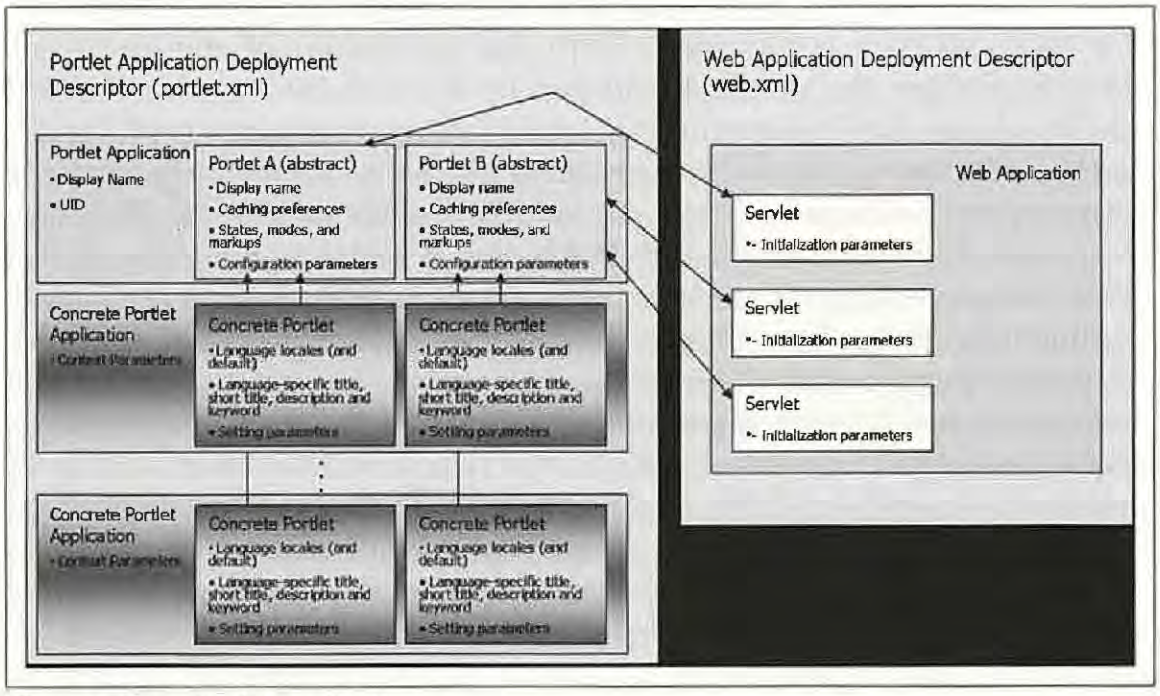


Figure 13-4 Portlet application.

To deploy a portlet to a portal there is a bit more work to do. We don't simply add a portlet to the portal. We define an abstract portlet in a portlet application and a concrete portlet in a concrete portlet application in the portlet deployment descriptor. We create a Web application and a servlet definition in a Web deployment descriptor and map the servlet to the abstract portlet. We package the deployment descriptors with the portlet class files and any associate resource files into a WAR file. Then we use the administrative tools in portal to deploy the WAR into portal. Under the covers, portal generates an Enterprise Application for the Web application and deploys the Enterprise Application to the Portal Application Server.

Portlet Life Cycle

Portlets have a life cycle similar to servlets. The portlet container is responsible to manage the portlet life cycle. As we would expect, portlets move through the life cycle by virtue of being deployed, being placed on a user's page, or having that page invoked by the user. They also progress as a result of configuration data that is associated with them.

As we see in Figure 13-5, an abstract portlet is loaded when the portal is initialized. Legacy portlets extend the Portlet API implementation class `org.apache.jetspeed.portlets.AbstractPortlet`, which in turn extends `org.apache.jetspeed.portlets.PortletAdapter`. `PortletAdapter` provides the default implementations for the methods referenced in Figure 13-5.

Prior to being first used the settings defined for the portlet are held in a `PortletSettings` object. With the association of the `PortletSettings` object the portlet is referred to as a concrete portlet. We can use the administrative function of portal to create many copies of the abstract portlet, each with its own portlet settings and each referred to as a concrete portlet. Portlet settings are read and write properties. They can be dynamically modified either through portlet function using the portlet API or through administrative services. Modification of these settings is required to be performed while the portlet is in configure mode if using the portlet API. Portlet settings are persisted by portal. They are initialized with the concrete portlet setting parameters defined in the portal deployment descriptor. We can then have multiple concrete portlets defined with their own setting information allowing a single portlet deployment to provide tailored behavior.

Like many representations of the portlet in the life cycle described here, there is no `ConcretePortlet` object. Rather the concrete portlet is a representation of the abstract portlet when associated with a specific `PortletSettings` object.

Portlet Representation	Action	Portlet API invoked	Associated Portal Objects	Result
	During portal initialization	ExamplePortlet init() method		ExamplePortlet class loaded and represented as an abstract portlet
Abstract portlet	Prior to first portlet access	ExamplePortlet initConcrete() method	PortletSettings	PortletSettings object created and associated to form a new concrete portlet
Concrete portlet	User or Administrator places ExamplePortlet on a page.		PortletData	PortletData object created and associated to form a concrete portlet instance
Concrete portlet instance	User access a page with ExamplePortlet	ExamplePortlet service() method	PortletSession	PortletSession object is associated to form a user portlet instance
Concrete portlet	Administrator deletes portlet	ExamplePortlet destroyConcrete() method		
Concrete portlet	Portal is stopping	ExamplePortlet destroy() method		

Figure 13-5 Portlet life cycle.

When the portlet is placed on a portal page, a `PortletData` object is created for the concrete portlet. This `PortletData` object is associated with the concrete portlet to generate a reference to a concrete portlet instance. There can be multiple concrete portlet instances for each concrete portlet. The `PortletData` object is available for portlet writers to persist application data. `PortletData` must be written when the portlet is in Edit mode but can be read from all modes. `PortletData` can be scoped to either a user or a group of users depending on the scope of the page on which the portlet was added. If an administrator puts a portlet on a shared page then the users that have view access to that page will share the portlet data. Users that have edit access to the page will get user-specific portlet data. Likewise, if users put a portlet on their own page, they will have private access to the portlet data.

Finally, when a user accesses a portal page, user portlet instances for the portlets on that page are created. User portlet instances are the association of the user `PortetSession` object with the concrete portlet instance. Of

course, there can be many user portlet instances for each concrete portlet instance. A `PortletSession` object is created for each user portlet when the user logs in to the portal. As we have discussed, for a new portlet deployment the deployment descriptors contain the information by which one or more abstract portlets and one or more concrete portlets are defined. When the portlet application is installed (deployed) to portal, the abstract portlet(s) are created, and when associated with their portlet settings their concrete portlet representations are also available. In this case, the portlet's `init()` and `initConcrete()` methods are invoked. Similarly, when the administrative function is used to copy a portlet a new `PortletSettings` object is created and associated with the new concrete portlet representation. This is performed by the `initConcrete()` method.

Portlet Configuration Objects

In the previous section on portlet life cycle we saw that changes in a portlet representation was marked by the association of a configuration object. They were `PortletSettings`, `PortletData`, and `PortletSession`. In addition, there was configuration defined in the portlet deployment descriptors that get encapsulated in other configuration objects including `PortletApplicationSettings` and `PortletConfig`. In this section we will take a closer look at these objects.

PortletConfig

In the Web deployment descriptor we can define a number of servlet context parameters. These configuration parameters provide the initial configuration for the abstract portlet and are available to all of its concrete portlets. This information is read-only and cannot be modified by the portlet. These parameters are encapsulated in a `PortletConfig` object and that object is passed to the abstract portlet on the `init()` method. Parameters are accessible using the `getInitParameters()` method on the `PortletConfig` object.

Figure 13-6 shows an example of a Web deployment descriptor with a configuration parameter named `persist_to_DB` defined.

PortletApplicationSettings

In the portlet deployment descriptor for each concrete portlet application we can define context parameters. Figure 13-7 shows an example portlet deployment descriptor with a portlet application setting parameter highlighted. There can be multiple parameters defined although only one is

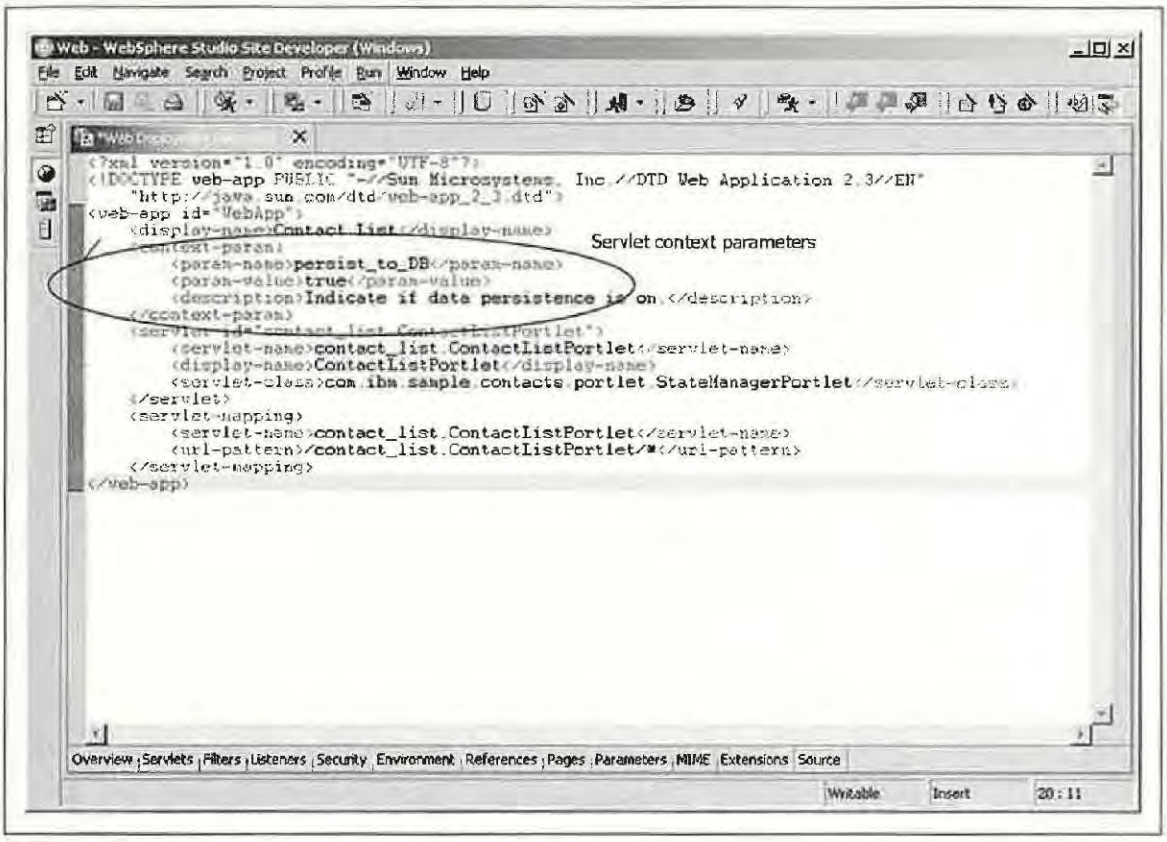


Figure 13-6 web.xml—context parameter.

shown here. When deployed these context parameters are encapsulated in a `PortletApplicationsSettings` object and are available to the portlet. This information is shared with all concrete portlets included in the portlet application. The `PortletApplicationsSettings` object can be read by the portlet in any mode but can only be updated in configure mode. It can be retrieved from the `PortletSettings` object using the `getApplicationSettings()` method.

PortletSettings

Also in the portlet deployment descriptor, for each concrete portlet we can define a number of configuration parameters. Figure 13-8 shows the example portlet deployment descriptor with a portlet setting highlighted. There can be multiple parameters defined although only one is shown here. When deployed these configuration parameters are encapsulated in a `PortletSettings` object and are available to the portlet. This information is shared with all concrete portlet instances. The `PortletSettings` object can be read by the portlet in any mode but can only be updated in configure mode. It can be retrieved from the `PortletRequest` object using the `getPortletSettings()` method.

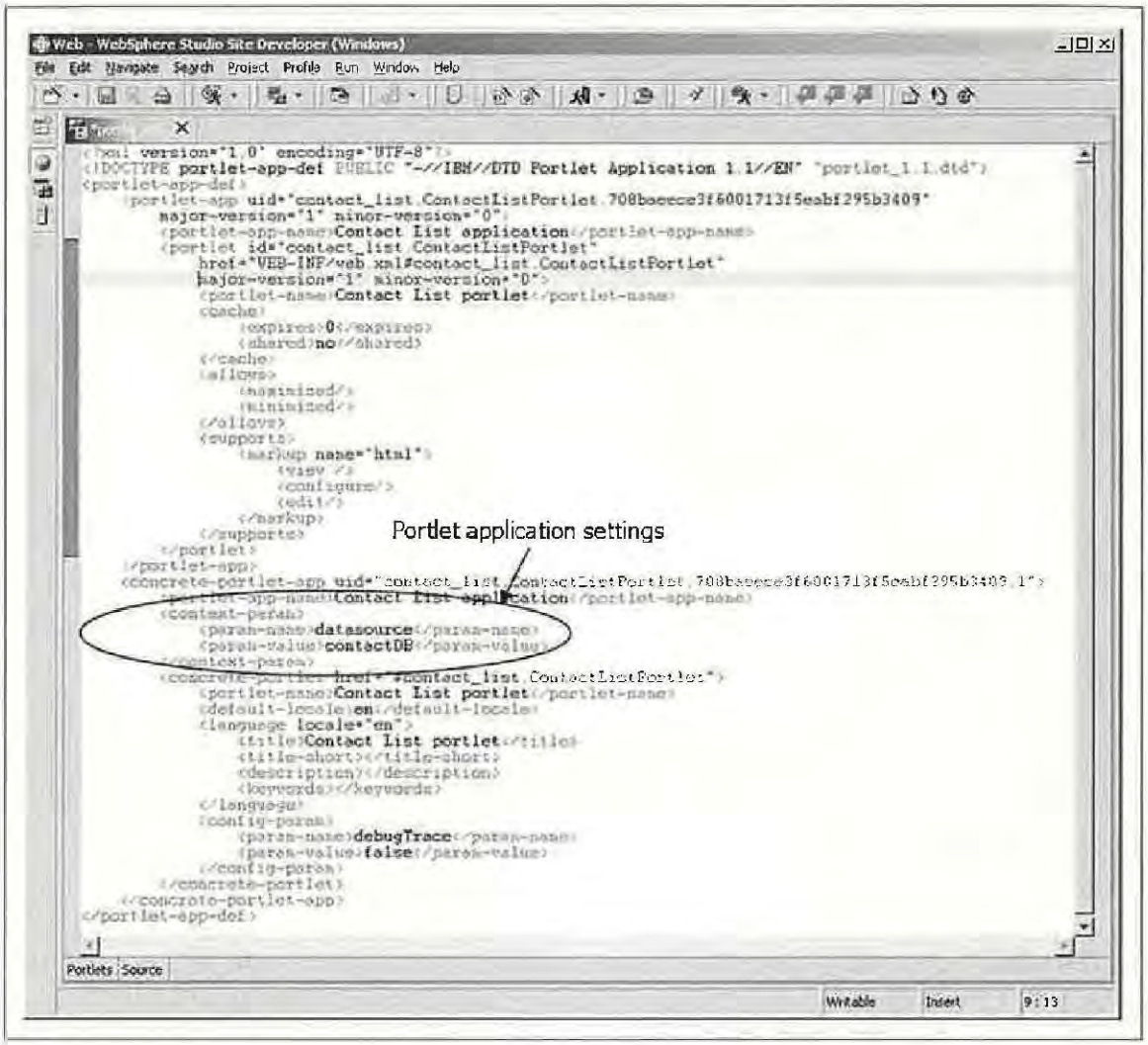


Figure 13-7 portlet.xml—portlet application settings.

The `PortletData` object manages persistent data for a concrete portlet instance. As we discussed in the section on portlet life cycle, when a portlet is placed on a portal page a `PortletData` object is created for the concrete portlet and then a concrete portlet instance is generated. The `PortletData` instance associated with the portlet can be retrieved using the `getData()` method of the `PortletRequest` object. Data is stored as name and value pairs, with the name represented as a string and the value as a serializable object. The `store()` method is used to persist the values set using the `setData()` method and as we discussed this can only be performed while the portlet is in edit mode.

`PortletData` can be scoped to either a user or a group of users, depending on the scope of the page on which the portlet was added. If an administrator puts a portlet on a shared page then the users that have view access to that page will share the portlet data. Users that have edit access to the page will get user-specific portlet data. Likewise, if users put a portlet on their own page, they will have private access to the portlet data.

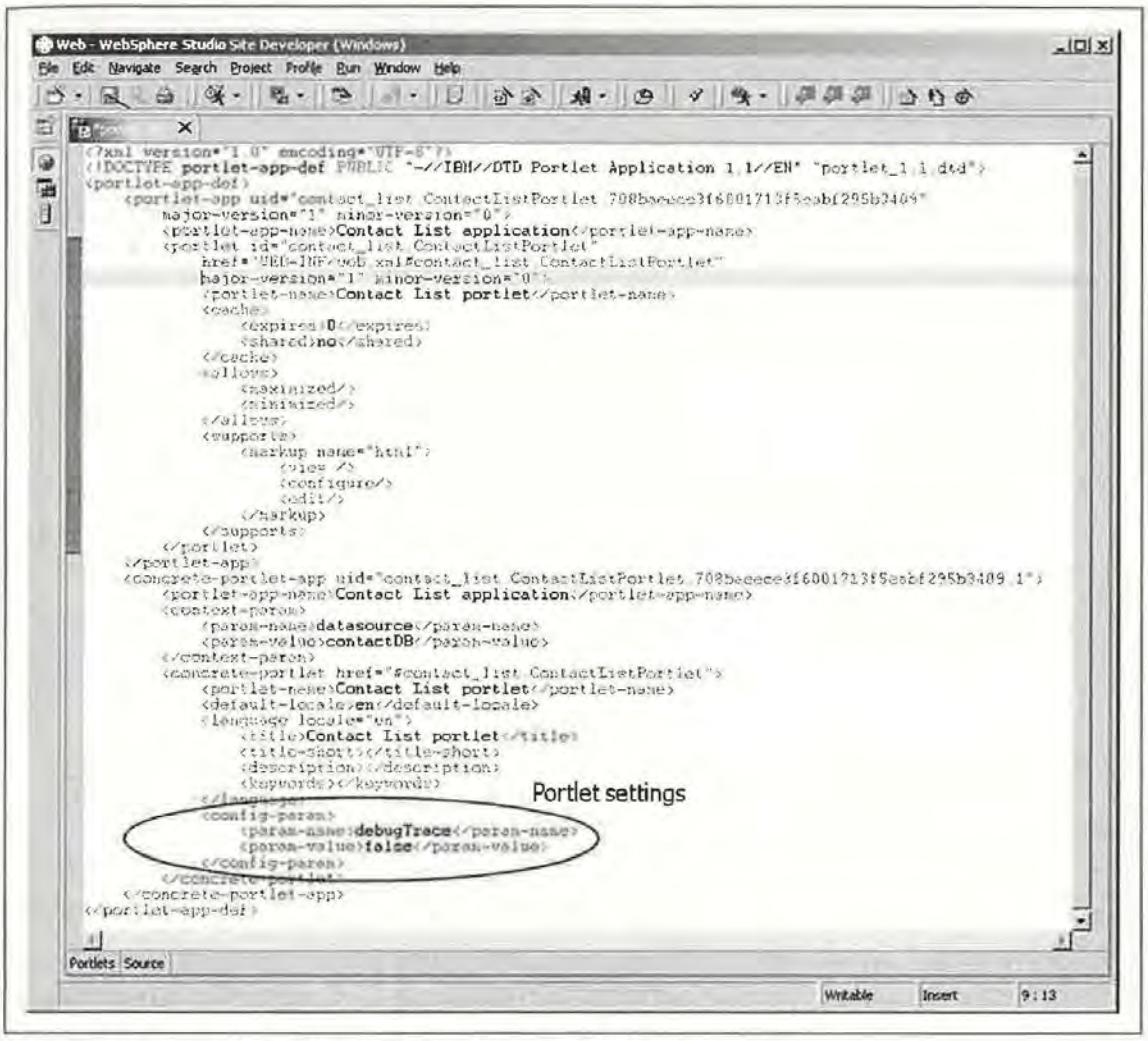


Figure 13-8 portlet.xml—portlet settings.PortletData.

PortletSession

A `PortletSession` is very similar to an `HttpSession`. Its function is to provide a facility to maintain transient user application data across multiple HTTP requests. A `PortletSession` object is created for all user portlets on authenticated pages when the user logs into the portal. Portlet user instances are differentiated from portlet instances (or other portlet user instances) by their `PortletSession` objects.

The `PortletSession` object is accessible from the request object using the `getPortletSession()` method.

A portlet placed on an unauthenticated page on the other hand will not have a `PortletSession` object assigned automatically. The portlet write can ask to have one created using the `getPortletSession()` or `getPortletSession(true)` method. Portlets should verify that they have a valid session object by testing the results of the `getPortletSession(false)` method.

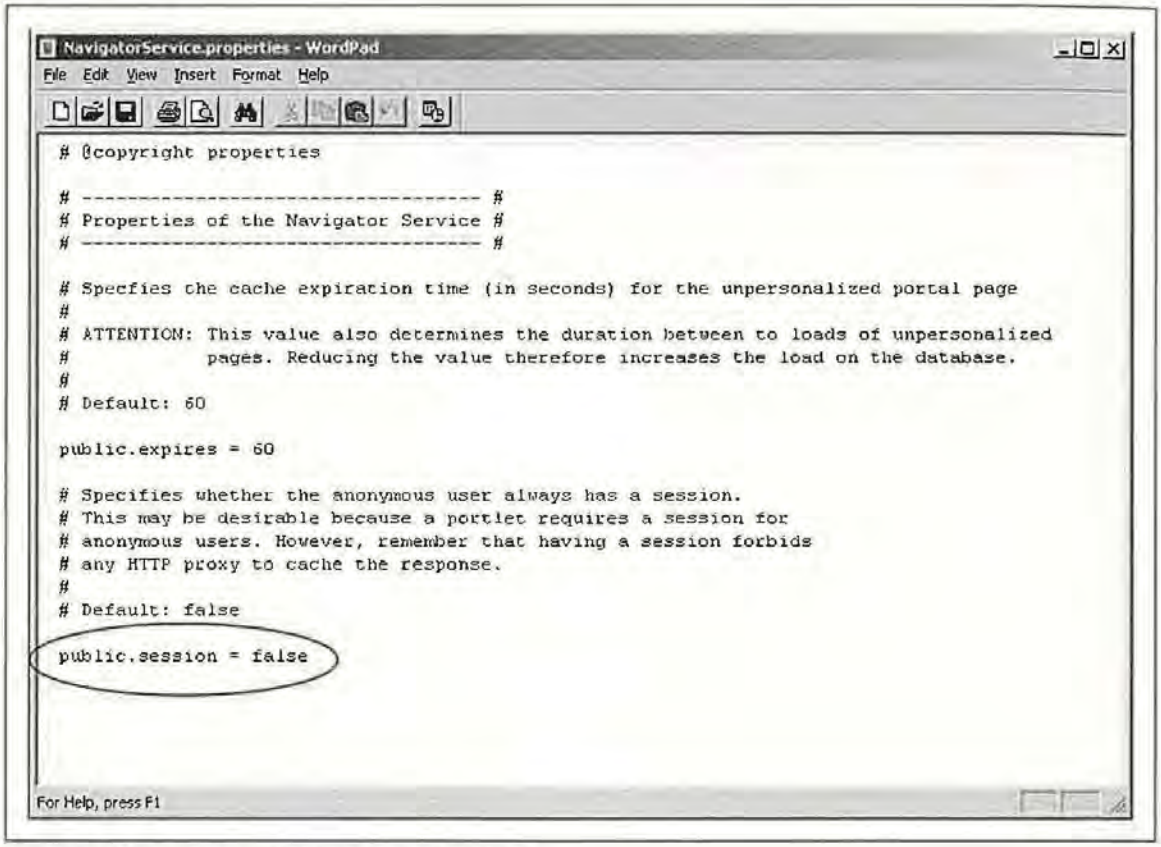


Figure 13-9 NavigatorService.properties.

A portal configuration change can be made to request a `PortletSession` object also by default for portlets on unauthenticated pages. Keep in mind that the session object will be available when the portlet is executed but will be transient. That is, a new `PortletSession` object will be created by the portlet container for each request to that portlet. Change the `public.session` property to true in the `NavigatorService.properties` (see Figure 13-9) file to have the `PortletSession` object created for unauthenticated page portlets. The `NavigatorService.properties` file can be found in `<wp_root>/shared/app/config/services`.

As with `HttpSession` objects, good programming practice is to be careful and limit how much data is put on the session. Application memory requirements and database overhead if the session data is persisted can grow quickly and limit performance and scalability.

This is especially true in a portal environment as there are as many "application sessions" active as there are portlets on pages that the user is visiting during the current logged-on session.

Portlet Objects

There are a number of portlet objects that are defined by the portlet API. The following section describes the `PortletRequest`, `PortletResponse`, `PortletContext`, `PortletLog`, and `User` objects. These represent the

key portlet objects in addition to the objects already defined in the section on portlet configuration (`PortletConfig`, `PortletSettings`, `PortletApplicationSettings`, `PortletData`, and `PortletSession`).

PortletRequest

The `PortletRequest` object provides the portlet access to HTTP request specific information and to portlet configuration information as well. The `PortletRequest` object is passed to the portlet as a parameter on the service method.

Through the `PortletRequest` the portlet can get and set attributes, can get parameters sent in the URI query string, and get the user agent information to determine client support information such as the markup name and the MIME types. As we have mentioned before, through the `PortletRequest` object, we can also retrieve the `PortletData`, `PortletSession`, and `PortletSettings` objects. We can also get the current mode and state of the portlet.

PortletResponse

Like the `PortletRequest` object the `PortletResponse` object is passed to the portlet as a parameter on the service method call. Typically, the `PortletResponse` object is used to return output to the client using a `PrintWriter`. The `PortletResponse` is also used to generate encoded URI strings that can be used to reference the portlet, and associated portlet actions, in context of the portal.

PortletContext

The `PortletContext` provides the portlet access to some context information in the portlet container including providing access to portlet services. Application scoped resources, attributes, and initialization parameters are all available through the `PortletContext`. Portlet-to-portlet messaging is also possible through the `PortletContext` as well as access to portlet services and the portlet log.

PortletLog

The `PortletLog` object is available from the `PortletContext` using the `getPortletLog()` method. It provides common trace logging services for portlet writers. Messages can be informational, warning, or errors. Logging at various levels can be switched on/off by changing a properties file setting or, dynamically, through portal administrative function.

One of the important benefits to applications running in a portal is the availability of common services and function. Some of these we have already mentioned, such as the common user interaction model and common UI facades through portal managed themes and skins. Another common service is user authentication and authorization (access control). Both of these services are provided by the underlying portal services, and portlet developers can focus on the application requirements and use the portlet API to determine user information.

The `User` object represents the profile of the user logged in to portal. The `User` object can be retrieved from the `PortletRequest` object using the `getUser()` method. While the `PortletSession` `getUser()` method is available as well, it has been deprecated in the current version of Portal.

Portlet Event Model

Unlike a servlet, which does all request processing in the `service()` method, a portlet has two processing phases. The first phase is the event phase and the second is the rendering phase.

Various types of events can be handled in the event phase. These can be action events, message events, portlet settings attributes events, and portlet application settings attributes events.

The `Event` interface is the basis for all events that are managed within the legacy portlet container. Portlets that implement the appropriate listener interface for these events will get notified during the event processing phase.

The portlet container is responsible to signal all event listeners before any rendering phase processing begins. Events can be generated by other event listeners during their event phase processing. In this case the portlet container will deliver that, and any subsequent events, before event processing completes. In addition, no events will be delivered after the event phase completes.

The listener interface defines the methods that will get invoked during event processing. For example, action events will get handled by the `actionPerformed()` method and message events by the `messageReceived()` method.

The rendering phase is sometimes referred to as the content generation phase. The portlet `service()` method is invoked to handle this request. Portlets typically inherit from `PortletAdapter` and the `service()` method implemented in that class performs the following:

```
public class Sample extends PortletAdapter {

  public void service(PortletRequest request, PortletResponse response)
       throws PortletException, IOException {
    if(request.getMode() == Portlet.Mode.VIEW)
```

```
            doView(request, response);
        else
        if(request.getMode() == Portlet.Mode.EDIT)
            doEdit(request, response);
        else
        if(request.getMode() == Portlet.Mode.HELP)
            doHelp(request, response);
        else
        if(request.getMode() == Portlet.Mode.CONFIGURE)
            doConfigure(request, response);
    }

}
```

So, since our portlet typically would want to make a distinction in processing depending on the mode the portlet is currently in, typically the portlet writer will simply override `doView()`, `doEdit()`, `doConfigure()`, and `doHelp()` instead of `service()`.

All portlet state changes are handled during event processing. The `service()` method is called after the portlets event handling listener is invoked for portlets that are direct targets of the event. In addition, the `service()` method of a portlet is called when the portal page is refreshed. So, when a user is interacting with a portlet on a page, that portlet could have processing in both phases. Other portlets on the same page will have their `service()` method called for the rendering phase to generate response markup fragments for the entire page to be generated.

Typically, the portlet class is used as the event listener class as well. The event handling method has access to the `PortletRequest` object and can add attributes to the `PortletRequest` to be referenced during the rendering phase, as the `PortletRequest` object is the same between both processing phases.

Action Events

Action events are generated when an HTTP request is received by the portlet container for which an action has previously been associated with that clickable link on the page. So, when we create content for a portlet and want to generate HTML with a clickable link we will use the portlet API to associate with the URI for that link, an encoding that can later be interpreted as a specific action event. We will look at this API more closely in the following chapter.

Message Events

Message events are generated when a portlet using the portlet API to send a message to another portlet where both portlets are within the same

portlet application and are on the same portal page. Also, a `DefaultPortletMessage` can be sent to portlets not in the same portlet application to all portlets on the portal page. The portlet receiving the message event must implement the `MessageListener` interface. This API allows for portlet-to-portlet communication and the capability to create cooperative portlets. We will see in later chapters a more advanced framework for generating cooperative portlets.

Portlet Settings Attributes Events

Portlets that implement the `PortletSettingsAttributesListener` interface will be notified when changes to the attribute list on the portlet settings of a concrete portlet occur. Depending on the change that occurred, the method `attributeAdded()`, `attributeRemoved()`, or `attributeReplaced()` will get invoked.

Portlet Application Settings Attributes Events

Portlets that implement the `PortletApplicationSettingsAttributesListener` interface will be notified when changes to the attribute list on the settings of a concrete portlet application occur. Depending on the change that occurred, the method `attributeAdded()`, `attributeRemoved()`, or `attributeReplaced()` will get invoked.

Summary

In this chapter we discussed what a portlet is. We discussed portlets in context of a high-level view of the portal architecture and how portlets are managed by the portlet container. We started the discussion about portlet container and portlet interface differences between the legacy portlet API and the API provided that implements JSR 168.

We learned about portlet rendering and page aggregation and how portlets provide markup fragments that the portlet container uses to generate a complete markup page response. We looked at how portlets are defined within a J2EE servlet deployment and how a portlet deployment descriptor maps portlet-specific information to a servlet definition in a Web application deployment descriptor that is needed for portlet deployment. We looked that the portlet life cycle and saw how portlets take on different contexts as configuration objects are associated with the portlet.

Finally we reviewed the portlet event model and looked at how a two-phase processing design handles portal and user interaction events prior to the portlet rendering phase.

In the next chapter we will look at the portlet API in more detail.

CHAPTER
14

Portlet Programming Model and API

In this chapter you will learn about the portlet Application Programming Interfaces (APIs) supported in WebSphere Portal. We will discuss in detail the API that exists for portlet development, starting with the existing API, which is based on Portal 5.0 and is referred to as the IBM Portlet API. We will discuss the Java Community Process portlet specification identified as Java Specification Request (JSR) 168.

JSR 168 is described as follows: "To enable interoperability between Portlets and Portals, this specification will define a set of APIs for Portal computing addressing the areas of aggregation, personalization, presentation and security."

We will review this specification in context of WebSphere Portal (version 5.02 or later) and compare this API to the IBM Portlet API.

The JSR 168 API

In the previous chapter we discussed portlets and the portal context under which they operate. Part of that discussion naturally relied on an understanding of the portlet API. The discussion in that chapter was based on the IBM Portlet API. In this chapter we will discuss the API based on JSR 168 and compare it with the IBM Portlet API.

Java Portlet Specification JSR 168 is a Java Community Process specification to provide a standard for interoperability between portlets and portals. The specification was approved in October 2003.

Portlet API in WebSphere Portal

As of version 5.0.2, WebSphere Portal provided support for the JSR 168 portlet API. However, at the time of this writing, there are functions and services provided in the IBM Portlet API that have not been defined in the JSR 168 specification. Therefore, the early implementation of JSR 168 API in WebSphere Portal does not contain all of the portlet functions available with the IBM Portlet.

As we mentioned in the previous chapter, WebSphere Portal provides two portlet containers to support the IBM Portlet API and the JSR 168 API. Portlets from both containers can reside on the same portal page. There is no difference to the portlet user as to which API is used as the administration and deployment of the portlets are the same. There are however functional differences in the API, which will be explored in this chapter.

Portlet API Comparison

In order to show an organized comparison between the two portlet APIs in total, we will review the API by functional area. The areas of the portlet API we will consider are:

- Portlet deployment descriptors
- Portlet processing model
- Portlet life cycle and configuration objects
- Portlet services and objects
- Portlet URI addressability
- Portlet window

Portlet Deployment Descriptors

In both portlet APIs, there is one portlet class instance for each portlet configuration in the Web deployment descriptor and one or more portlets are defined in a portlet application in a WAR file. And in both cases the portlet deployment descriptor (`portlet.xml`) defines the portlet application and portlets, while the Web deployment descriptor (`web.xml`) defines the associated Web application.

There are a few main differences with the JSR 168 portlet deployment descriptor. A change with the JSR 168 API is that a portlet is no longer a servlet. The portlet deployment descriptor is created in the format of an XML schema and it does not contain a reference to a corresponding

servlet ID in the Web descriptor. As with the IBM Portlet API we can associate context parameters in the Web descriptor and retrieve them using the `PortletContext getInitParameter()` method.

In order to uniquely identify the portlet application, a unique identifier (UID) is added to the `<portlet-app/>` element. In the case of the IBM Portlet this was the UID attribute, for the JSR 168 descriptor this is the ID attribute. The default ID value, if not specified, is the WAR filename.

Within the portlet element definition the `<allows>` maximized and minimized do not need to be explicitly defined in the JSR 168 descriptor. The portlet must support the default window states. Custom state however will need to be defined here. Figure 14-1 shows this element set in the IBM Portlet descriptor.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application 1.1//EN"
    portlet_1.1.dtd">
<portlet-app-def>
    <portlet-app uid="com.ibm.sample.HelloWorldPortlet.30ef486bec"
            major-version="1" minor-version="0">
        <portlet-app-name>Hello World application</portlet-app-name>
        <portlet id="com.ibm.sample.HelloWorldPortlet"
            href="WEB-INF/web.xml#com.ibm.sample.HelloWorldPortlet"
            major-version="1" minor-version="0">
            <portlet-name>Mastering WP Sample portlet</portlet-name>
            <cache>
                <expires>0</expires>
                <shared>no</shared>
            </cache>
            <allows>
                <maximized/>
                <minimized/>
            </allows>
            <supports>
                <markup name="html">
                    <view />
                </markup>
            </supports>
        </portlet>
    </portlet-app>
    <concrete-portlet-app
        uid="com.ibm.sample.HelloWorldPortlet.30ef486bec.1">
        <portlet-app-name>Hello World application</portlet-app-name>
        <concrete-portlet href="#com.ibm.sample.HelloWorldPortlet">
            <portlet-name>Hello World portlet</portlet-name>
            <default-locale>en</default-locale>
            <language locale="en">
                <title>Hello World</title>
                <title-short>Hello World</title-short>
                <description>Say hello</description>
                <keywords>Portlet, World</keywords>
            </language>
            <language locale="de">
                <title>Hello Welt</title>
                <title-short>Hallo Welt</title-short>
                <description>Sagen Sie Hallo</description>
                <keywords>Portlet, Welt</keywords>
            </language>
        </concrete-portlet>
    </concrete-portlet-app>
</portlet-app-def>
```

Figure 14-1 IBM Portlet API descriptor.

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
    version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
                        http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
    id="com.ibm.sample.jsr.api.HelloWorld.30ef406bec">
    <portlet id="Portlet.30ef406bec">
      <description xml:lang="EN">This is a Hello World portlet</description>
      <description xml:lang="DE">Dies ist ein Hallo Welt portlet</description>
      <portlet-name>Hello World portlet</portlet-name>
      <display-name xml:lang="EN">Hello World Portlet display name</display-name>
      <display-name xml:lang="DE">Hallo Welt Portlet Ausstellung Name</display-name>
        <portlet-class>com.ibm.sample.jsr.api.HelloWorld</portlet-class>
      <expiration-cache>0</expiration-cache>
      <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
      </supports>
      <supported-locale>en</supported-locale>
      <resource-bundle>nls.HelloWorld</resource-bundle>
    </portlet>
</portlet-app>
```

Figure 14-2 JSR 168 Portlet descriptor.

Localization

The IBM Portlet API allows you to specify localized strings in the portlet deployment descriptor for the portlet title, the portlet short title, the portlet description, and the searchable keywords. You can see in Figure 14-2 in the example deployment descriptor that the concrete portlet is defined to default to English (en) but includes translated text to support English and German.

The JSR 168 API provides two mechanisms in the deployment descriptor to support localization. The first is the ability to specify localized strings in the descriptor for administrative value like the portlet display name and the portlet description. These elements allow you to specify an `xml:lang` attribute to allow you to specify multiple element definitions using different language strings.

You can see in Figure 14-2 the elements defining the description and display name in both English and German. Also you can see in the example that the portlet also defines the locales it supports using the `<supported-locale>` element.

The second mechanism allows for the localization of strings like the portlet title, the portlet short title, and the portlet search keywords in a resource bundle. The resource bundle is specified in the deployment descriptor using the `<resource-bundle>` tag. The resource bundle then defines the localized strings for these properties. The portlet has access through the `PortletConfig` object to the resource bundle at runtime.

The following shows the `HelloWorld.properties` file for this portlet.

```
#
# HelloWorld Resource Bundle
#
```

```
# English Resource Bundle
#
javax.portlet.title=Hello World Portlet
javax.portlet.short-title=Hello World
javax.portlet.keywords=Hello

#
# German Resourse Bundle
#
javax.portlet.title=Hallo Welt Portlet
javax.portlet.short-title=Hallo Welt
javax.portlet.keywords=Hallo
```

Portlet Content Types

The JSR 168 API allows the portlet container to define to the portlets the content types that the container supports. It also allows the portlet deployment descriptor to specify the content types that the portlet supports. Figure 14-2 shows a portlet descriptor where the supported content type is defined in the `<supports>` section.

Portlets must explicitly set the content type of their markup response. The `RenderResponse setContentType()` method is used. The content type specified must match one of the content types supported by the portlet, otherwise an `IllegalArgumentException` is thrown and the `setContentType()` method must be called before the response writer is committed. The `getResponseContentTypes()` method of the `RenderResponse` object returns the list of supported content types. The method will only return content types that the portlet supports as defined in the deployment descriptor.

Portlet Class Definition

The `Portlet` interface defines the JSR 168 portlet class. Portlet can implement this interface directly or extend `GenericPortlet`, which implements Portlet and provides default method implementations.

By comparison, portlet classes using the IBM Portlet API typically extend `AbstractPortlet` and may also implement listener interfaces to participate in event model notifications.

Portlet Processing Model

Both APIs adhere to a two-phase processing model. A key difference in the JSR model is that the request and response objects are different between the two phases of processing. With the IBM Portlet API you could set attributes

Table 14-1 IBM Portlet API Events and Listeners

EVENT	LISTENER
`ActionEvent`	`ActionListener`
`MessageEvent`	`MessageListener`
`WindowEvent`	`WindowListener` (depreciated in WebSphere Portal 5.0)
`PortletSettingsAttributeEvent`	`PortletSettingsAttributeListener`
`PortletApplicationSettingsAttributeEvent`	`PortletApplicationSettingsAttributeListener`
Render phase life cycle	`PortletPageListener`
Session life cycle	`PortletSessionListener`
Event phase life cycle	`EventPhaseListener`

on the `ActionRequest` object in the event processing phase and retrieve them from the `RenderRequest` object in the rendering phase. Using the JSR 168 API you cannot do this. Passing user instance objects from the first phase to the second needs to be done on the session object. However, you can set string parameters on the `ActionRequest` in the event phase and have them added to the `RenderRequest` object for the render phase.

Instead of the `service()` method being invoked as in the IBM Portlet API, the JSR 168 API defines a `render()` method. The `actionPerformed()` event listener method is replaced with the `processAction()` method. Also, during the action phase, the JSR 168 API allows portlets to redirect requests to other Web resources, a servlet for example.

The IBM Portlet API provided an event model to signal processing events such as a user action or a portlet message. As we mentioned previously, the action event listener method is replaced by a method on the portlet that gets invoked from the portlet container. The IBM Portlet API supports the events and associated listeners shown in Table 14-1. The JSR 168 API does not support events or listeners.

Portlet Modes

The view, edit, and help modes are the same between both APIs and can be invoked through the corresponding `doView()`, `doEdit()`, and `doHelp()` methods. The JSR 168 API provides the capability for portlets to define custom modes where configure mode can be implemented. Other suggested custom modes include about, preview, print, and edit defaults.

JSP Invocation and Tag Libraries

To invoke a JSP from the portlet render phase, the IBM Portlet API provided an `include()` method on the `PortletContext` object. The JSR 168 API provides a `PortletRequestDispatcher include()` method. We will define the `PortletRequestDispatcher` object later in this chapter. The following code example shows the invocation of a JSP using the JSR 168 API.

```
public void doView(RenderRequest request, RenderResponse response)
    throws PortletException, IOException {

  PortletContext context = getPortletConfig().getPortletContext();
  response.setContentType("text/html");
  String jspFileName = "/jsp/html/main_view.jsp";
  context.getRequestDispatcher(jspFileName).include(request,response);
}
```

In the JSP, the portlet tag library provides a `<portletAPI:init/>` tag that will initialize local variables to reference the `PortletRequest`, `PortletResponse`, and `PortletConfig` objects. Using the tag library with JSR 168 API the `<portlet:defineObjects/>` tag initializes variables for `RenderRequest`, `RenderResponse`, and `PortletConfig` objects.

Other taglib differences for creating portlet URI references and to encode namespace references are `<portletAPI:createURI/>` and `<portletAPI:encodeNamespace/>` in IBM Portlet API and `<portlet:actionURL/>` and `<portlet:renderURL/>` and `<portlet:namespace/>` in the JSR 168 API.

Render Parameters

Render parameters allow a portlet to store its navigational state. The `setRenderParameter()` and `setRenderParameters()` methods of the `ActionResponse` object allows portlets to set render parameters during an action request. The parameters will then be used for all render requests until the next action occurs.

The portlet should be written to store any information that is needed to successfully redisplay and use those parameters in the render phase. When the portlet encounters a new action, the render parameters are set again.

Portlet Life Cycle and Configuration Objects

In the IBM Portal API the `PortletSettings` object encapsulates the portlet configuration setting initially defined in the portlet deployment descriptor. It is through a `PortletSettings` object that the abstract portlet was

Table 14-2 JSR 168 Portlet

PORTLET REPRESENTATION	ACTION	PORTLET API INVOKED	ASSOCIATED PORTAL OBJECTS	RESULT
	Portlet container starts the portlet application			Portlet class loaded and instantiated
Portlet	Prior to first portlet access	`init()` method	`PortletConfig`	Portlet is initialized
Portlet	User or Administrator places the portlet on a page		`PortletPreferences`	Portlet window

configured as a concrete portlet. There could be multiple concrete portlets for each abstract portlet. Similarly, an association of a `PortletData` object with a concrete portlet defines a concrete portlet instance.

When users edit the settings in a portlet according to their preferences, the settings are persisted using the `PortletData` object. There can be many `PortletData` objects associated with the same concrete portlet. Each `PortletData` object along with the concrete portlet comprises a concrete portlet instance.

The JSR 168 API does not define a concrete portlet or a concrete portlet application. Portlets are defined by a `PortletPreferences` object that encapsulates parameters in the portlet deployment descriptor. These parameters can be identified as read-only and in that case will be analogous to the IBM Portlet API `PortletSettings` object in that they can be modified only when the portlet is in configure mode.

Also since there are no concrete portlets, there are no life cycle methods such as `initConcrete()` or `destroyConcrete()` in the JSR 168 API. Table 14-2 shows this portion of the JSR 168 portlet life cycle.

The `PortletConfig` object in the IBM Portlet API provides access to the initialization parameters defined in the Web deployment descriptor. In the JSR 168 API, the `PortletConfig` object provides access to the initialization parameters in the portlet deployment descriptor.

Portlet Session

The JSR 168 API defines two session scopes, `APPLICATION_SCOPE` and `PORTLET_SCOPE`. Using APPLICATION_SCOPE allows the sharing of

information between portlets. Any object that is set to the `PortletSession` object using APPLICATION_SCOPE can be retrieved by any other portlet that is part of the same portlet application and that handles a request that is part of the same session. This sharing is among all components of the Web application so that you can share data between portlets or between a portlet and a servlet.

The code example below shows the JSR 168 API to set an attribute in the portlet session using APPLICATION_SCOPE.

```
PortletSession portletSession = portletRequest.getPortletSession(true);
MyAppBean myAppBean = new MyAppBean(portletRequest);
portletSession.setAttribute("app_bean", myAppBean,
  PortletSession.APPLICATION_SCOPE);
```

Portal Context

JSR 168 API supports a `PortalContext` that can be retrieved from the `PortletRequest` object. The `PortalContext` provides portal information to the portlet. This allows the portal vendor, portal version, and a set of portal properties to be defined by the portal and available to the portlet. The portlet can interrogate the portal information in the `PortalContext` and adapt accordingly.

Portlet Services and Objects

This section will discuss additional portlet services and portlet object differences between the two APIs.

Portal User Profile

In the IBM Portlet API the user profile information is accessed in the form of the `User` object, retrieved from the `PortletRequest` object. In an LDAP configuration the `User` object held attributes that mapped to an LDAP object, typically the `inetOrgPerson` LDAP object class.

In the JSR 168 API the user profile information (such as name, address, and userid) is stored in a map in the request and can be accessed through the Platform for Privacy Preferences (P3P) keys. In the JSR 168 API the deployment descriptor can define the user profile information that is to be made accessible. The recommendation from the JSR 168 specification is to use a list of standard P3P attributes but the portlet is not limited to that definition. The portal will map the requested user profile attributes to the supported profile attributes. The portlet can then determine which profile attributes are available by retrieving the map of attributes using the `getAttribute()` method of the request object for the attribute named `PortletRequest.USER_INFO`.

Portlet Caching

The two APIs use different mechanisms to implement portlet caching. The IBM Portlet API uses an invalidation-based caching mechanism where the portlet can explicitly invalidate the portlet cache using the `invalidateCache()` method on the `PortletRequest` object.

In addition, with the IBM Portlet API the portal queries a portlet to determine if the cache is still valid. The portlet deployment descriptor sets a cache time value for the portlet application. The value determines the length of time the cache is valid. The portal queries the portlet using the `getLastModified()` method to determine the time that the cache was last modified. By overriding this method the portlet can manage the portlet cache.

In Figure 14-1 you saw the cache value specifications in the portlet deployment descriptor for the portlet application. For this example, this entry looks like the following: The *expires* value of 0 indicates that the cache always expires. The *shared* value of "No" indicates that the portlet cache should not be shared between portlet instances. A value of −1 for *expires* indicates that the cache never expires and any other positive number indicate the number of seconds that the cache is to be valid.

The following xml is the cache definition from the portlet deployment descriptor:

```
<cache>
  <expires>0</expires>
  <shared>NO</shared>
</cache>
```

Caching is somewhat different in the JSR 168 as the portlet can attach an expiration time to the portlet definition. The JSR 168 API defines an expiration-based cache mechanism. As you saw in Figure 14-2 the `<expiration-cache>` timer value is set in the portlet deployment descriptor. It can be dynamically changed by the portlet using the `setAttribute()` method of the `RenderResponse` object referencing the `EXPIRATION_CACHE` attribute name.

The following xml shows the expiration cache timer value from the portlet deployment descriptor.

```
<expiration-cache>0</expiration-cache>
```

NOTE Sharing the cache entry across users is only possible in the IBM Portlet API. Dynamic fragment caching is not implemented for JSR 168 portlets in WebSphere Portal V5.0.2. Placing JSR 168 portlets on a page with other portlets that support caching results in incorrect page output. If you use dynamic fragment

cache, make sure that there are no JSR 168 portlets on pages where portlets are cachable.

Property Broker

The property broker provided with WebSphere Portal and the IBM Portlet API is used to allow the dynamic integration of portlets. Portlets can be developed and deployed independently and then by using the property broker service, portlets can register for and receive notification of property changes.

Cooperative portlets declare, publish, and share data (properties) using the property broker, not with a specific portlet or portlets. Portlets that need to respond based on changed property values will associate actions with properties. This action processing in the portlet does not differentiate between a user initiated action event or an action that is the result of a property value change. In this way, portlets can behave in a coordinated fashion without depending on a close development–time association.

This service is not available with the initial implementation of the JSR 168 API.

Portlet Services

The IBM Portlet API provides a `PortletService` interface. This interface allows vendors and developers to make pluggable portlet services available to portlet developers. `PortletService` writers follow a few simple steps to create their service:

1. Define the service (class extends `PortletService`).
2. Implement the service (class implements `PortletServiceProvider`).
3. Optionally create a new service factory (if not using the predefined factories).
4. Register the service in the `PortletServiceRegistryService.properties` file in the `<wp_root>/shared/app/config/services` directory.

Portlet developers can use the `PortletContext getService()` method to get the `PortletService` for the class of the given name. Two portlet services provided with this API are the `ContentAccessService` and the `CredentialVaultService`.

The `ContentAccessService` allows portlets to access remote systems or content from remote URLs, including URLs located on the other side of a proxy server.

The `CredentialVaultService` has two associated components. The credential service assists the portlet to handle basic authentication, LTPA Token authentication, and simple form-based userid/password login challenges from back-end systems being accessed on a user's behalf. The credential vault service is used to retrieve credentials from a secure, persisted store, a Credential Vault. This service can also pass Tivoli Access Manager or Netegrity SiteMinder single sign-on tokens to the back-end application in the appropriate HTTP headers.

Portlet services are not available with the initial implementation of the JSR 168 API.

Portlet Menus

The portlet menu service allows the portlet to add to the portal navigation menu, providing users easier navigation through the portal. This service is not available with the initial implementation of the JSR 168 API.

Portlet URI Addressability

References to a specific portlet, portal page, or portal resource must be encoded in the portal Web application URI. The IBM Portlet API creates a `PortletURI` object for this encoding, while the JSR 168 API creates a `PortletURL` object. In both cases you can set the portlet state or add parameters using these objects.

To encode the URI to reference portlet deployed resources, both APIs use the `encodeURL()` method on the `PortletResponse` object.

IBM Portlet API Addressability

The IBM Portlet API uses the `createURI()` or `createReturnURI()` method on the `PortletResponse` object to create a `PortletURI` object referencing the portlet. The `createURI()` method creates a `PortletURI` object pointing to the calling portlet with the current mode. The `createReturnURI()` method creates a `PortletURI` object pointing to the calling portlet with the previous mode. The `PortletURI toSring()` method will return a URI string that can be used in the markup for a link (such as an action tag or href tag).

In order to associate a portlet action event with that URI, the `PortletURI` can be modified to add an action event trigger. This is done by using the `addAction(String actionName)` method on the `PortletURI` object.

For example, you may want to have two buttons on a form input page on your portlet. One button is to "save" the form data and one to "cancel"