that for every deviation an action and time factor are required to resolve the issue.

After you address your processes, and your organization, you look at the technological elements associated with WebSphere Portal. Specifically:

- Remove any single point of failure. Look at each component including routers, communication vendors, operating systems, middleware, application servers, applications, disk drives, network cards, servers, portlets, Web servers, databases, and LDAP servers. Single point of failures can be eliminated using hardware redundancy, multiple network cards attached to isolated networks supported by different communication vendors, load balancing, hot fail-over, and clustering. For LDAP servers, look at implementing a master–slave relationship, databases either hot fail-over or parallel databases with data replicated over multiple storages using mirrored disks. Disk redundancy can be implemented with two NAS (network attached storage), each attached to a separate isolated network with storage software that replicates data stored on the master NAS to a slave NAS. Upon failure of the master NAS, the router will redirect to the slave NAS. Keep redundant servers in different locations so they are not impacted by local environmental failures.
- Use only mature software and reliable and serviceable hardware. Most vendors are pressured by the market and their shareholders to release products quickly. Thorough testing is not always performed due to time and cost factors. Sometimes due to number of products that can interact with it, it is not always possible. In mature products, both design and function flaws have been found by the market and usually fixed (or a workaround provided) by the vendor.
- Implement security software to prevent unauthorized access and data corruption.
- Automate the system management as much as possible. Employ operating and server agents and agents for WebSphere Application System. These agents should monitor and perform recovery or performance tuning when a threshold occurs. Also automate the change management process. Integrate the system management process to your help desk.
- Set up a test environment that duplicates your production environment. Ensure that development, test, and production are isolated from each other.
- Reuse well-established configurations. Your staff does not need to learn a system and they are familiar with the issues. The configuration is tested and spare parts are usually available.

- Over-configure your hardware. Calculate what memory, DASD, and CPU speed you need and multiply by at least 4. It is cheaper to have inefficient software running on faster hardware than tuning the software for the hardware configuration. It is very easy to underestimate capacity needs and costlier to add on when the system is in production. No user ever complained that his or her system was too fast.
- And now, to drive you nuts, the last and most important point completely contradicts the first point. Consolidate your servers and keep your solution as simple as possible. Complex solutions introduce more failure points and make it more difficult to find the problem. While the time to failure may lengthen with a complex configuration, so also will the time to repair due to the increase in manageability issues.

Implementing a Highly Available WebSphere Portal Solution

So now after reading all this high-level advice on high availability, you must be saying, "Great stuff, but how do I implement a highly available WebSphere Portal?" This section will tell you how. In the paragraphs that follow you will examine two highly available models for WebSphere Portal based on vertical and horizontal scaling. We will discuss their advantages and disadvantages and show you how to implement WebSphere Portal clustering. We are not going to drill down further the process and organizational components because this will require a separate book.

Support for automatic fail-over and load balancing for WebSphere Portal is provided by WebSphere Application Server Network Deployment. It supports the concept of clusters, which enables a logical collection of application server processes to operate logically as a single application server process. If one server within the cluster fails, then the workload is picked up by the other servers.

WebSphere Application Server Network Deployment terminology for a hardware server is a node. A group of nodes under a single administrative node make up a cell. Single point administration for the cell is done by the Deployment Manager, which uses the cell master configuration repository to store the configuration for all nodes in the cell. Each node has an agent that communicates with the Deployment Manager and provides file transfer services, configuration synchronization, and performance monitoring.

You can use three different models for designing your highly available WebSphere Portal: WebSphere Portal using vertical scaling, WebSphere Portal using horizontal scaling, and lastly a hybrid using both horizontal and vertical scaling.

Vertical Scaling with a WebSphere Portal Cluster

Each WebSphere Portal instance has a dedicated JVM. Vertical scaling is basically the implementation of multiple instances of WebSphere Portal on a single node. Figure 21-3 shows a simple example of WebSphere Portal cluster configuration using vertical scaling. Requests first go to a reverse proxy server that sprays the request to an available Web server. The Web server via the plug-in forwards the request to the WebSphere Portal Web containers WebSphere_Portal_1 and WebSphere_Portal_2. Since they are part of a cluster, the node agent will monitor their performances and forward the information to the Deployment Manager and manage the workload if one of the instances goes offline or is busy. Authentication is via the LDAP server, which uses a master–slave relationship to maintain availability. Availability for the database is maintained by using a parallel database. All data is stored on two Network Access Storage systems that mirror each other using storage synchronization software. Each server has two network cards that access two separate networks that are isolated from each other.

Vertical scaling provides many benefits. It enables you to make better use of the CPU since multiple JVMs can more fully utilize the processing power than a single JVM due to JVM's concurrency limitations. It is also easier to maintain and cheaper since you are administrating fewer machines and session management is easier since you can use memory-to-memory session replication.

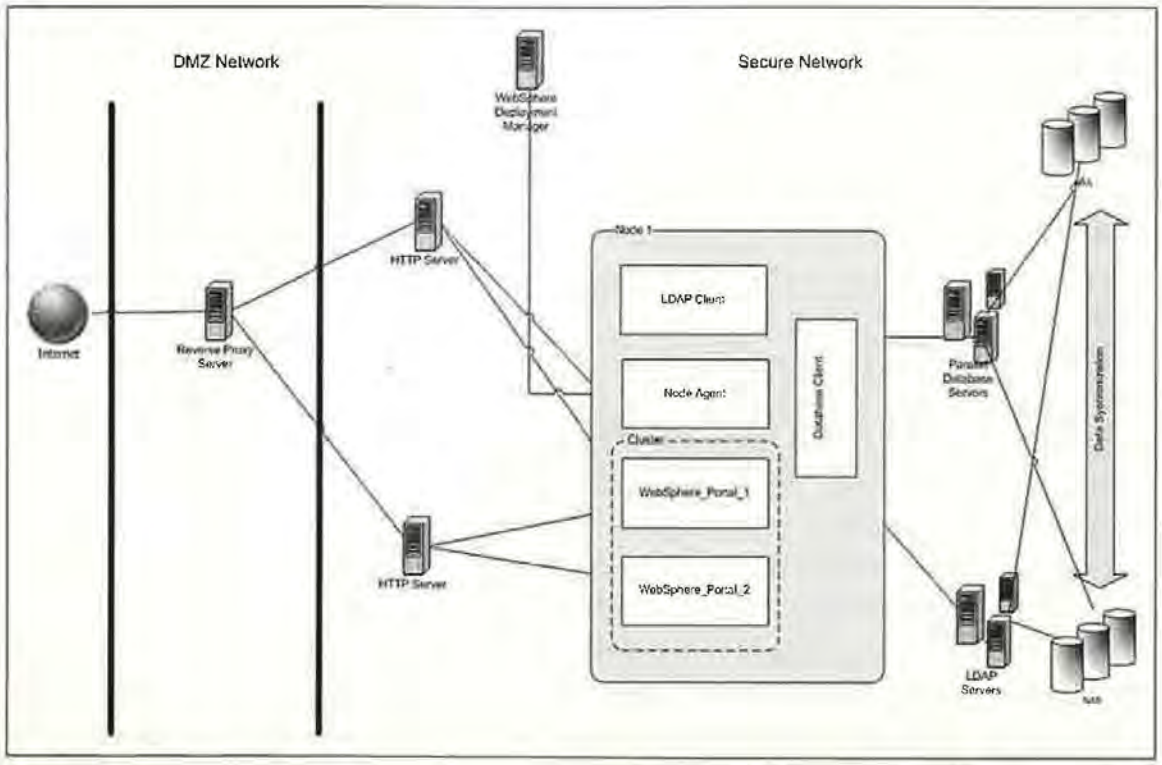


Figure 21-3 WebSphere Portal cluster using vertical scaling.

Of course, the vertical scaling model has certain limitations. It has more single point of failures (LDAP Client, Node Manager, Database Client, hardware, and so on), requires greater memory management, and value decreases significantly as more instances are added.

Horizontal Scaling with a WebSphere Portal Cluster

Horizontal scaling is putting on WebSphere Portal instance per node and then adding the nodes to the cluster. Figure 21-4 shows the previous example using horizontal scaling. As in the previous example, Node Manager monitors the performance of the WebSphere Portal instances; however, this time Deployment Manager receives information from multiple agents.

The benefits of horizontal scaling are numerous. You can isolate the system from various hardware or software failures. Each system can be in different locations, thus isolating you from environmental factors such as a disaster. By creating multiple cells, you can also perform maintenance or test a new version in a production system without interrupting service.

The disadvantages are that it is more costly (more machines, software license), more complicated and thus harder to administrate, and session management is slower and less reliable since it requires database persistence.

The last model combines both vertical and hardware scaling. Each machine runs multiple instances and all the instances are part of a cluster. This, of course, combines the advantages of both vertical and horizontal

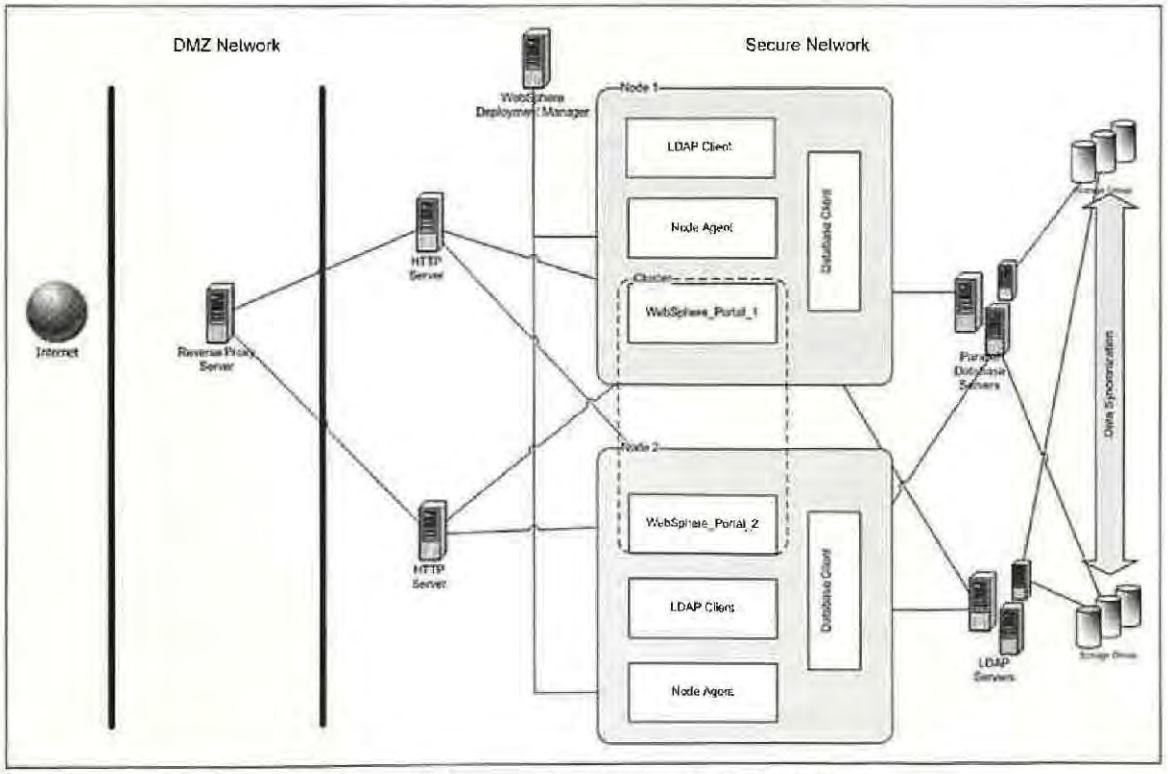


Figure 21-4 WebSphere Portal cluster using horizontal clustering.

clustering so long as you only use a few instances per server. However, it is very costly and complex to administer. This is not a preferred strategy for those who like to keep it simple.

Configuring WebSphere Portal in a Clustered Environment

Now that you understand the advantages and disadvantages of different availability models, the next step is to learn how to implement WebSphere Portal in a clustered environment. Specifically, you will find out how to implement WebSphere Portal using the example infrastructure detailed in Figure 21-5. The infrastructure, which is based on horizontal scaling, has a single IBM HTTP Web Server (sandbox1), which passes requests to two WebSphere Portal Server nodes (sandbox3, sandbox4) that are part of a WebSphere cell. Authentication is performed using IBM Directory Server (sandbox5) and the data is stored remotely on a DB2 server (sandbox6). Another server is used for Deployment Manager (sandbox2). For the purpose of this example, all servers are Windows 2000 Advance Servers; however, any of the WebSphere-supported platforms can be used.

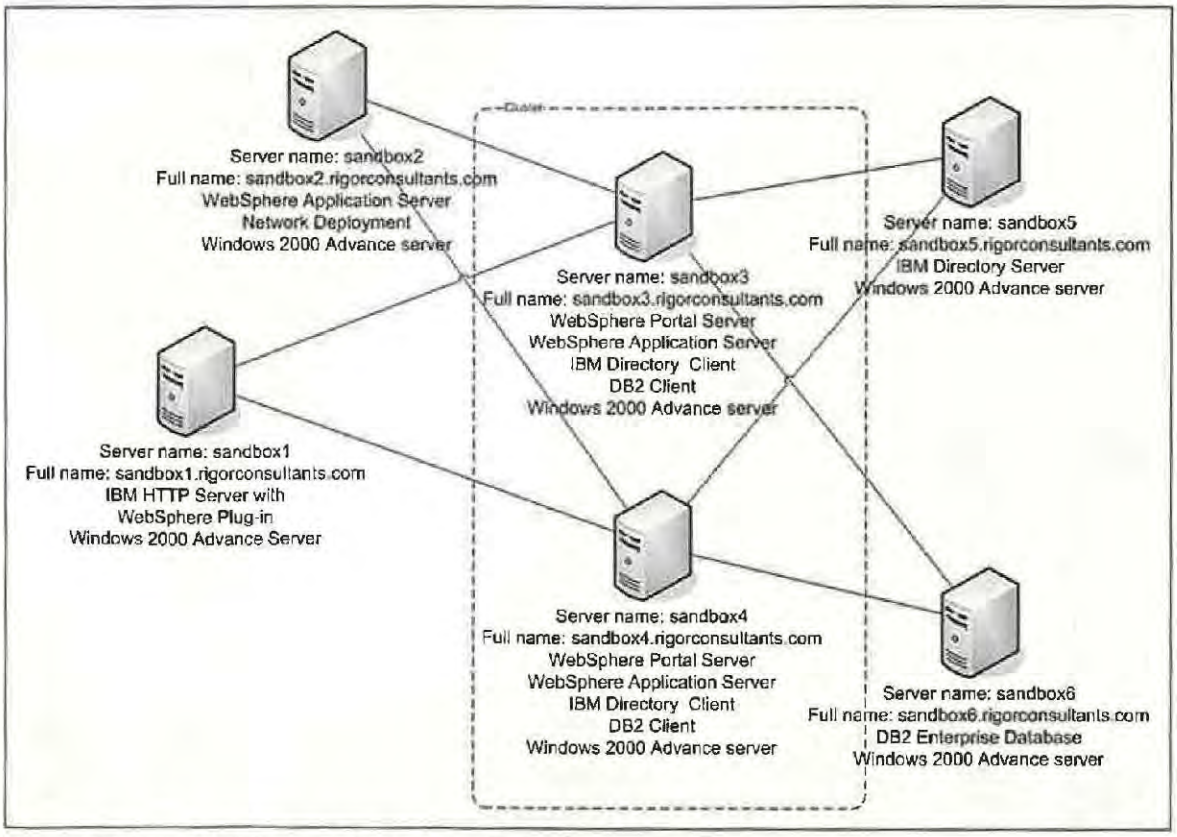


Figure 21-5 WebSphere Portal cluster.

For the purpose of this example, assume the following:

1. You have installed and configured WebSphere Network Deployment 5.0.1 and WebSphere Application server 5.0.1 on `sandbox2.rigorconsultants.com`
2. You have enabled security on Deployment Manager using single sign-on and the authentication server is IBM Directory Server sandbox5. Java 2 security needs to be disabled. Make sure that wpsadmin has been set to an Administrator role in the Deployment Manager Admin console; sandbox2.rigorconsultants.com:9090/admin.
3. IBM Directory Server 5.1 has been installed and configured on sandbox5 and IBM Directory Client is installed on sandbox3 and sandbox4. The latest fixpacks have been installed.
4. DB2 Enterprise 8.1 database has been installed and configured on sandbox6 and the DB2 client is installed on sandbox3 and sandbox4. The latest fixpacks have been installed.
5. IBM HTTP Server 1.3.6 is installed and configured on sandbox1.
6. For both sandbox3 and sandbox4 (which must be done after you set up sandbox1, sandbox2, sandbox5, and sandbox6), you have done the following:
   a. Performed a full installation of WebSphere Portal 5.0. You cannot install WebSphere Portal into an existing cluster environment. If the node is within a cell, you must first remove it, install WebSphere Portal, and then reinstall it.
   b. Configured both WebSphere Portal servers to use the remote Web server sandbox1.
   c. Configured the WP servers to access the remote IBM Directory Server sandbox5 and the remote DB2 server sandbox6.
   d. Configured the tables on the remote database server and exported the data for sandbox4 WebSphere Portal to the DB2 server.
   e. Set up wpsbind and wpsadmin in the LDAP server as administrative users using the definitions specified in the LDIF file.
7. You have validated that each component is working by separately testing each WebSphere Portal server, IBM Directory Server, the DB2 server, and the Network Deployment Manager server.
8. The date and time in Deployment Manager (sandbox2) and the nodes (sandbox3, sandbox4) must be the same or within 5 minutes and must be in the same time zone.

9. WebSphere Network Manager is running on sandbox2 and all the node names are unique in the cell.
10. Server1 and WebSphere Portal are running on sandbox3 and sandbox4.
11. Maximum heap size has been increased to 512 to `<was_root>\bin\addnode.bat` and `<was_root>\bin\removenode.bat` by adding **-Xmx512** to the Java command line.

Adding WebSphere Portal to a Cell

After you have set up your infrastructure, you need to add the WebSphere Portal nodes to the cells by doing the following:

1. Click Start ➪ Programs ➪ IBM WebSphere ➪ Application Server V5.0 ➪Start the Server.
2. Click Start ➪ Programs ➪ IBM WebSphere ➪ Portal Server V5.0 ➪Start the Server.
3. Open a command prompt and go to `<was_root>\bin` and execute addnode `sandbox2.rigorconsultants.com 8879` -username wpsbind -password wpsbind -includeapps. The deployment host name is `sandbox2.rigorconsultants.com`, soup port is 8879, and wpsbind is the Deployment Manager administrator username and password. The -includeapps option is used for the first cell only in order to install the applications into the Deployment Manager cell. Upon completion (which will take some time) you will see the message "node sandbox3 has been successfully federated."
4. Repeat the process for sandbox4 except do no use the -includeapps option in the addnode command.

Remember that when you add a node to a cell, all administration must be done through the Deployment Manager Administration console instead of the Portal Administration console. So you need to remove the Portal Administration console to avoid future problems with the plug-in configuration. Do the following to remove the Portal Administration console:

1. Open the administrative console `http://sandbox2.rigorconsultants.com:9090/admin`
2. Log in as the administrative user **wpsbind**.
3. Expand Applications on the Navigation menu.
4. Click Enterprise Applications.
5. Find WpsAdminconsole application and click Uninstall.
6. Save the changes to the master configuration.

Implementing the WebSphere Portal Cluster

Now that WebSphere Portal nodes have been added to the cell, you need to create the cluster. Perform the following tasks:

1. Log in into the Deployment Manager Administrative console `http://sandbox2.rigorconsultants.com:9090/admin` as **wpsbind**.
2. Expand Servers and select Clusters. On the right side is the Create a New Cluster window.
3. Enter the cluster name (we chose **Portal_Cluster**) and check Prefer Local Enabled and Create Replication Domain for this Cluster.
4. Assign a weight that determines how much workload is distributed to that node relative to the other node. We recommend keeping an equal weight or the default value.
5. Click Select an Existing Server to add to this cluster, choose the server WebSphere_Portal from the node sandbox3, and click Next.
6. Enter the name of the next cluster member. We called it **WebSphere_Portal_Sandbox4**.
7. Assign a weight that determines how much workload is distributed to that node relative to the other node. We recommend keeping an equal weight or the default value.
8. Select the node sandbox4 and check Generate Unique HTTP Ports and Create Replication Domain for this cluster. It is imperative that you click Generate Unique HTTP Ports, otherwise there will be conflicts between the original servers and other servers on sandbox4.
9. Click Apply and the cluster member WebSphere_Portal_sandbox4 will appear on the application server list on the bottom of the page.
10. Click Next and the summary window is displayed. Then click Finish.
11. Save the changes to the master configuration.
12. Log in remotely to sandbox3 and sandbox4 and change the application server name to the cluster name so that the portlet deployment can work. Open a command prompt and go to `<wp_root>\shared\app\config\services` and edit `DeploymentService.properties`. Change the value assigned to `wps.appserver.name` to Portal_Cluster, and save the file.
13. Go back to the Deployment Manager Administrative console, expand Servers, and click Clusters.

14. Select the cluster Portal_Cluster and click Start. Wait for some time.
15. Check if the cluster members have started by expanding Servers, selecting Application Servers, and refreshing the Status view.

The next few steps relate to updating the WebSphere plug-in configuration since the plug-in configuration now resides on the Deployment Manager (sandbox2) instead of the nodes (sandbox3, sandbox4). The HTTP Server host name and port must be added to the host alias list in order for a clustered portal application to be accessible. (Remember, you are still in the Deployment Administrative Console.)

16. Expand Environment and click Virtual Hosts.
17. Select default_host, and click Host Aliases on the Additional Properties table.
18. Click New and add `sandbox1.rigorconsultants.com` (our Web Server) for the Host Alias and 80 for the Port Number.
19. Save the configuration.
20. Regenerate the plug-in expand Environment, select Update Web Server Plug-in, and click OK.
21. Copy the `plugin-cfg.xml` file located on the `<nd_root>\config\cells` directory to `<was_root>\config\cells` on sandbox1 (our Web server). Edit `plugin-cfg.xml` and change `<nd_root>\etc\plugin-key.kdb` to `<was_root>\etc\plugin-key.kdb`. Also change `<nd_root>\etc\plugin-key.sth` to `<was_root>\etc\plugin-key.sth`. Save the files.

These last steps are to enable dynamic caching. Dynamic caching improves performance by caching the output of dynamic servlets and Java Server Pages. However, it is also required for clustered portal applications because all nodes in a cluster share the same cache information.

22. Expand Servers, click Application Servers, and select WebSphere_Portal.
23. Select Dynamic Cache in the Additional Properties section and select Enable service at server startup.
24. Select Enable cache replication and click on its link. Select the portal cluster name (Portal_Cluster) and the desired application server name as the replicator (WebSphere_Portal).
25. Ensure the runtime mode is Push only and click OK.
26. Save the changes to the master configuration.

27. Go to step 22 and repeat for application server Websphere_Portal_Sandbox4.

Testing Your WebSphere Portal Cluster

Before you perform the next steps, you need to ensure that WebSphere Portal cluster is working. You can do this by testing the fail-over functionality. This is done as follows:

1. Log in into the Deployment Manager Administrative console `http://sandbox2.rigorconsultants.com:9090/admin` as **wpsbind**.
2. Expand Servers and click Clusters.
3. Select the cluster Portal_Cluster and click Start. Wait for some time.
4. Check if the cluster members have started by expanding Servers, selecting Application Servers, and refreshing the Status view.
5. Log in into WebSphere Portal `http://sandbox1.rigorconsultants.com/wps/portal` using **wpsadmin**.
6. Select Administration page. If everything is fine, go back to your home page.
7. Pull the network cable out of a WebSphere Portal Node (sandbox3 or sandbox4).
8. Select the Administration page. It should show up without any problems. Repeat the test except plug-in the network cable and take out the cable from the other node.
9. Select the Administration page again. If it appears then your WebSphere Cluster has been successfully implemented.

Deploying Portlets into a WebSphere Portal Cluster

The next task you have to do to set up Deployment Manager to automatically deploy an installed Portlet to all WebSphere Portals application server in the cluster. Basically, what you do is install a portlet through WebSphere Portal, and Deployment Manager will then synchronize all the other cluster members. The following are the steps to do this:

1. Log into WebSphere Portal `http://sandbox1.rigorconsultants.com/wps/portal` using **wpsadmin**.
2. Go to Administration ➪Portlets ➪Install.
3. Browse for the portlet WAR file and Click Next.

4. The portlets included in the WAR file will be displayed. Click Next and then click Install.
5. You will then get a message stating that your portlets were installed but not activated.
6. Open the Deployment Manager Administrative console `http//sandbox2.rigorconsultants.com:9090/admin`. You need to manually synchronize all nodes. You can set the autosynchronization on; however, it is recommended for portlets to still manually resync them.
7. Expand System Administration and select Nodes.
8. Select sandox3 and sandbox4 and click Synchronize.
9. Expand Applications and select Enterprise Applications. Find your portlet and click Start.
10. Log in again into the WebSphere Portal `http://sandbox1.rigorconsultants.com/wps/portal` using **wpsadmin**.
11. Go to Administration and click on Portlets icon. Select Manage Applications.
12. To activate the portlet, select the portlet application and then the portlet name and click Activate/Deactivate.

Deploying Themes and Skins into a WebSphere Portal Cluster

Like portlets, you install new theme and skin or modify it, it has to be synchronized with the other Portal Servers. Specifically, you have to export the Portal EAR file from Deployment Manager, update the EAR file with the new or modified theme and skin directories, and import the EAR file back to the Deployment Manager cell. Using the example in Figure 21-6, you would do this by taking the following steps:

1. Log in to sandbox2, open a command prompt, and go to `<nd_root>\bin`.
2. Enter wasdmin -user wpsbind -password wpsbind where wpsbind is the administrator ID and password we set up.
3. Enter AdminApp export wps `c:\temp\wp_old.ear`.
4. Enter **Quit**. Deployment Manager WebSphere Portal EAR will be exported into the file `c:\temp\wp_old.ear`.
5. Create the directory `c:\temp\wp_expand` and expand the `wp_old.ear` file by typing the command **EARExpander-ear c:\temp\wp_ld.ear -operationDir c:\temp\wp_expand\ -operation expand**.

6. Copy the new themes to `c:\temp\wp_expand\wps.war\themes\<markup>\`.
7. Copy the new skins to `c:\temp\wp_expand\wps.war\skins\<markup>\`.
8. Collapse the files back to an EAR file by typing the following command: **EARExpander -ear c:\temp\wp_new.ear -operationDir c:\temp\wp_expand\ -operation collapse**
9. Now you will import the modified EAR back into the Deployment Manager. Enter **wasdmin -user wpsbind -password wpsbind**, where `wpsbind` is the administrator ID and password you set up.
10. Enter **AdminApp** and install wps `c:\temp\wp_new.ear`. If you are updating EAR then you add the options -update -appname wps. Wait for the message that Application wps has installed successfully.
11. Enter **AdminConfig save**, which saves the changes to the master configuration.
12. Enter Quit.
13. Next you have to add the new skin and theme to the Portal Administration page, by doing the following:
    a. Log on to `http://sandbox1.rigorconsultants.com/wps/portal` as **wpsadmin**.
    b. Select Administration ➪ portal User Interface ➪ Themes and Skins and click Add New Skin.
    c. Enter the Skin name and default locale title.
    d. Enter the skin directory name and click Add New Theme.
    e. Enter the theme name in Theme Name and default locale title and enter theme directory.
    f. Select the desired skins and click OK.

Summary

In this chapter we have touched on some of the challenges in implementing a highly available WebSphere Portal. We have discussed the various availability components that need to be looked at when implementing WebSphere Portal. We also showed how to implement WebSphere Portal in clustered environment.

But if there is one message that you need to get out of this chapter, it is that high availability is very complex and costly. You need to have a good financial reason to put it in place and major commitment from all levels of the organization. In our simple example we identified 33 single point of

failures and this did not include organizational, process, or environmental factors. In a real-life organization there are thousands of elements that impact a major system and addressing every one is cost prohibitive. You need to focus on the components that have the largest impact upon failure and are the least costly to fix (otherwise known as the low hanging fruit).

As we showed in the beginning of this chapter, most of the problems that cause outages (or performance degradation that appears as outages) relate to software bugs or configuration errors. If you put a buggy portlet into a cluster, all that will happen is all the nodes will crash at the same time. If you misconfigure your servers, operating system or WebSphere Portal, the cluster will probably fail. Remember that putting WebSphere Portal cluster does not imply that you will get higher availability. It may actually get worse.

The complexity of WebSphere Network Deployment requires a very disciplined and well-run IT shop with defined processes that staff adhere to. If you implement clustering in a haphazard mode, the probability of your application having frequent outages is quite high. Before you implement high availability, you have to first ensure that quality and engineering processes are in place, metrics are collected, and that there is a high-level organizational visibility and accountability.

Do not perform a high-availability implementation if your organization is just giving it lip service or is not prepared to address cultural issues. Instead, just focus on reducing your failure points and providing better diagnostic and automated repair processes.

In the next chapter we will discuss a technology preview that was introduced in WebSphere 5.0.2, Web Service Remote Portlets or WSRP.

CHAPTER

22

WebSphere Portal Support for Web Services and Remote Portlets

Web services have become the preferred method for Internet-based integration and have been adopted by all vendors. This widespread adoption has made Web services a technology that any Web developer needs to understand and use—and portal developers are no exception.

In this chapter you will learn to use Web services with WP. You will see how to make Web service calls from portlets using the various wizards available within WebSphere Studio. We will show you how to discover existing Web services using UDDI registries and how to use them from within portlets you develop and deploy from within your portal. In addition (and perhaps more importantly) you will learn about the Web Services for Remote Portal (WSRP) standard and the concept of portal proxies. Using WSRP support in WP you will see how to publish your portlets for use by other portals and how you can use remote portlets from within your own WP instance. This integration metaphor allows you to bypass all development efforts and use existing portlets (including both the functional layer and the presentation layer) within your portal. You will learn how to enable your portal for WSRP, how to produce remote portlets, and how to use them within a portal.

We think it is important that you understand these two main methods for using Web services within WP. The first method uses core Web service functionality—for example, calls to remote functions that are described by WSDL and that are formatted using SOAP over HTTP. This is the most general-purpose use of Web services and one that can be utilized in every environment. The second, using WSRP, is more limited to remote portlets but allows you to integrate the function as well as the presentation with little or no development effort.

A Quick Review of Web Services and Remote Portlets

A brief review is in order for those who have not recently used Web services. The goal in this section is not to teach you all about Web services but rather to ensure that the concept is well understood by all readers of this chapter.

Web services are functional elements deployed on a node on the network and accessed over the Internet. This description is quite generic and doesn't say too much; what makes such a service endpoint a Web service is the "how," not the "what." Web services are based on a set of standards—specifically the Simple Object Access Protocol (SOAP), the Web Service Description Language (WSDL), and Universal Description Discovery and Integration (UDDI). SOAP is a protocol by which a remote client can invoke the functionality implemented by the Web service. Developers of Web services use WSDL to define the metadata describing the Web service, and remote clients use WSDL to learn what arguments are required for invoking the Web service (as well as other things required to make the remote call). Web service providers use UDDI to publish their Web services, and clients of Web services use UDDI to discover where these Web services, and the metadata describing them, reside. For an overview of SOAP, WSDL, and UDDI see *Executive's Guide to Web Services* by Eric Marks and Mark Werrell and for an in-depth discussion see *Java, XML, and Web Services Bible* by Mike Jasnowski.

SOAP, WSDL, and UDDI form the core of the Web services stack, and Web services are not specially made for portal deployment. Using Web services within the context of portlet development is no different than making use of Web services when developing servlets, JSPs, or any other Java programs. But Web services do not stop with SOAP, WSDL and UDDI—in fact, there are over 100 higher level protocols and definitions within the world of Web services. One of these "extensions" is very much specific to portals and portlets—remote portlets and the Web Services for Report Portlets (WSRP) standard.

WSRP is an OASIS standard that was created to simplify the integration of remote applications and content into portals. OASIS (`www.oasis-open.org`) is a not-for-profit global consortium that drives the development, convergence, and adoption of e-business standards and is responsible for much of the work in areas such as XML and Web services. The purpose of WSRP is to allow exposing content from third-party portlets in portals with zero programming effort. This is possible because WSRP defines an interface for presentation-oriented Web services rather than basic functional Web services.

WSRP allows a portlet—both its functionality and its presentation—to be packaged as a reusable component using Web service technologies. Web services allow you to take a function, describe and publish it in a standard way,

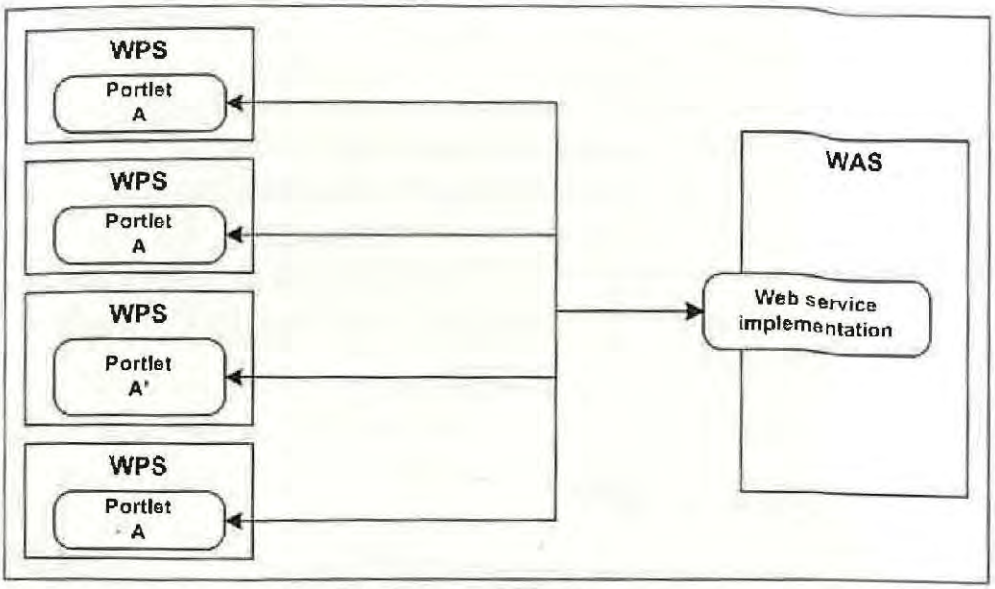


Figure 22-1 Multiple portlets calling a Web service.

and allow others to find and invoke it in a standard way. The inventors of the remote portlet concept have taken a viewpoint of a portlet as a "reusable function." Instead of requiring you to separate your application functionality from the presentation layer, it allows you to embed both function and presentation inside a single reusable component that uses Web service infrastructure such as SOAP and UDDI registries to perform the plumbing.

The difference between using Web services from within a portlet and the use of remote portlets is best exemplified by the implementations shown in Figures 22-1 and 22-2. In Figure 22-1 a portlet makes use of a Web

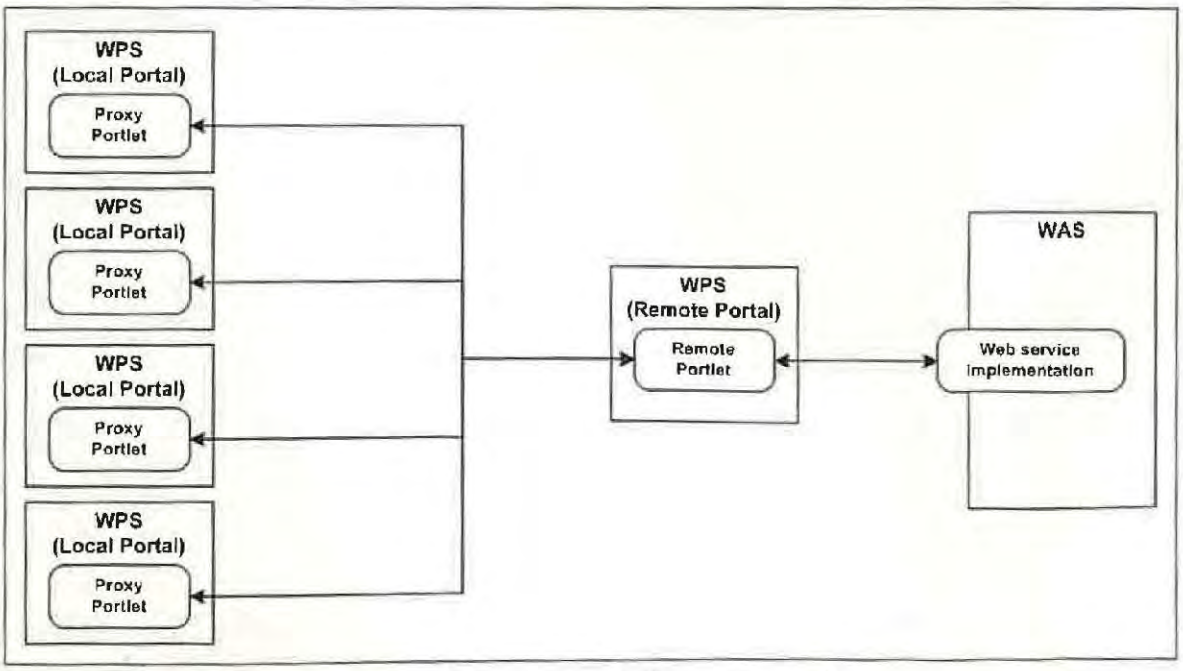


Figure 22-2 Multiple proxy portlets using a single report portlet.

A Quick Review of Web Services and Remote Portlets

A brief review is in order for those who have not recently used Web services. The goal in this section is not to teach you all about Web services but rather to ensure that the concept is well understood by all readers of this chapter.

Web services are functional elements deployed on a node on the network and accessed over the Internet. This description is quite generic and doesn't say too much; what makes such a service endpoint a Web service is the "how," not the "what." Web services are based on a set of standards—specifically the Simple Object Access Protocol (SOAP), the Web Service Description Language (WSDL), and Universal Description Discovery and Integration (UDDI). SOAP is a protocol by which a remote client can invoke the functionality implemented by the Web service. Developers of Web services use WSDL to define the metadata describing the Web service, and remote clients use WSDL to learn what arguments are required for invoking the Web service (as well as other things required to make the remote call). Web service providers use UDDI to publish their Web services, and clients of Web services use UDDI to discover where these Web services, and the metadata describing them, reside. For an overview of SOAP, WSDL, and UDDI see *Executive's Guide to Web Services* by Eric Marks and Mark Werrell and for an in-depth discussion see *Java, XML, and Web Services Bible* by Mike Jasnowski.

SOAP, WSDL, and UDDI form the core of the Web services stack, and Web services are not specially made for portal deployment. Using Web services within the context of portlet development is no different than making use of Web services when developing servlets, JSPs, or any other Java programs. But Web services do not stop with SOAP, WSDL and UDDI—in fact, there are over 100 higher level protocols and definitions within the world of Web services. One of these "extensions" is very much specific to portals and portlets—remote portlets and the Web Services for Report Portlets (WSRP) standard.

WSRP is an OASIS standard that was created to simplify the integration of remote applications and content into portals. OASIS (`www.oasis-open.org`) is a not-for-profit global consortium that drives the development, convergence, and adoption of e-business standards and is responsible for much of the work in areas such as XML and Web services. The purpose of WSRP is to allow exposing content from third-party portlets in portals with zero programming effort. This is possible because WSRP defines an interface for presentation-oriented Web services rather than basic functional Web services.

WSRP allows a portlet—both its functionality and its presentation—to be packaged as a reusable component using Web service technologies. Web services allow you to take a function, describe and publish it in a standard way,

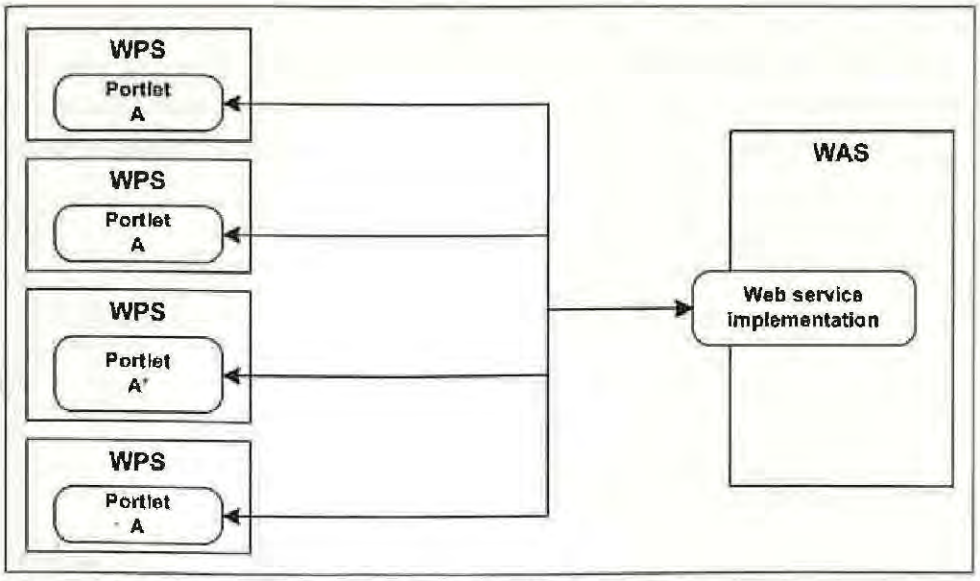


Figure 22-1 Multiple portlets calling a Web service.

and allow others to find and invoke it in a standard way. The inventors of the remote portlet concept have taken a viewpoint of a portlet as a "reusable function." Instead of requiring you to separate your application functionality from the presentation layer, it allows you to embed both function and presentation inside a single reusable component that uses Web service infrastructure such as SOAP and UDDI registries to perform the plumbing.

The difference between using Web services from within a portlet and the use of remote portlets is best exemplified by the implementations shown in Figures 22-1 and 22-2. In Figure 22-1 a portlet makes use of a Web

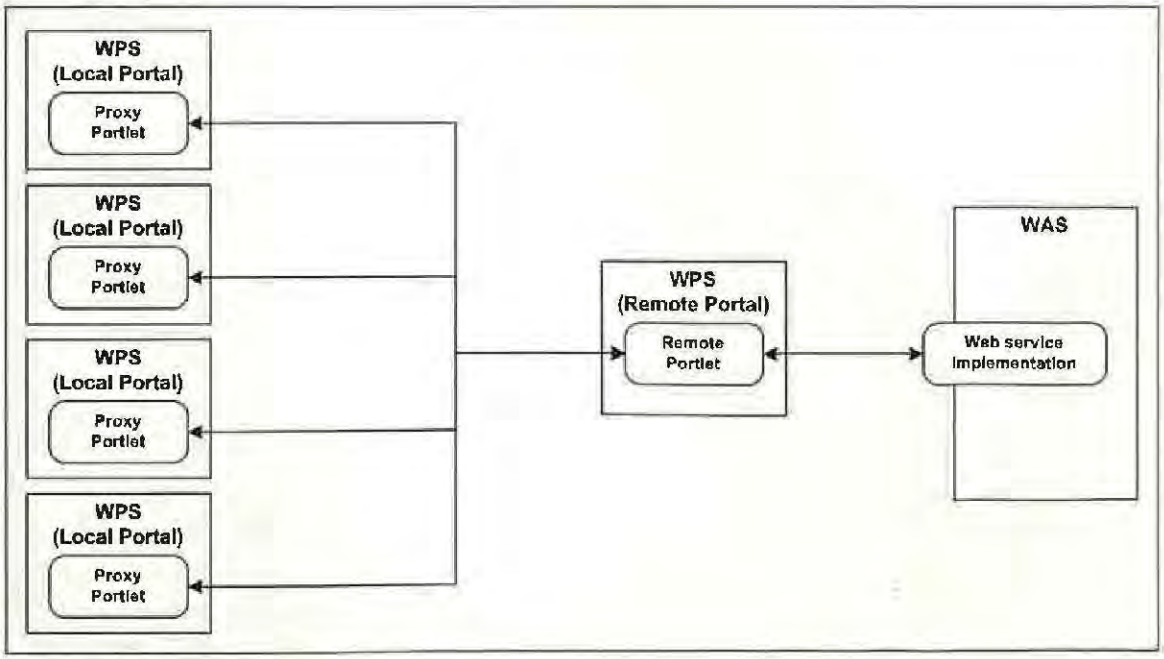


Figure 22-2 Multiple proxy portlets using a single report portlet.

service. While the same Web service implementation can be used by many portlet instances, there are still multiple portlets that can either be the same packaged portlet deployed on multiple portals or, as is more likely, involve separately developed portlets that deliver similar functionality into the portal. In Figure 22-2, a single portlet makes calls to the published Web service and multiple proxy portlets embed this portlet within multiple portals. The proxy portlets in this case are artifacts of WP that are automatically created and that do not need to be developed—saving you work in building portlets that do the same thing anyway.

Using Remote Portlets in WP

WSRP is a relatively new initiative support for WSRP within WP. Version 4x of WebSphere Portal supported Web services and remote portlets through a proprietary IBM implementation. Version 5 of WP still supports this proprietary method but we highly recommend that you do *not* use this method. This implementation is being phased out by IBM and is being replaced by the WSRP-conformant implementation. At the time of writing this chapter, WP did not yet fully support the WSRP specification for production environments; WSRP support within WP 5.0.2 was supplied as a Technology Preview by IBM. However, this implementation is fairly mature and it is likely that by the time you read this chapter and use Web services within your portal, WP will fully support WSRP. Therefore, we will cover only the WSRP support within WP and suggest that if you use remote portlets you use WSRP.

Figure 22-3 shows the full life cycle support provided by WP for remote portlets. Starting off with a portlet, which you wish to make available to other portal servers, the major steps in the life cycle are as follows:

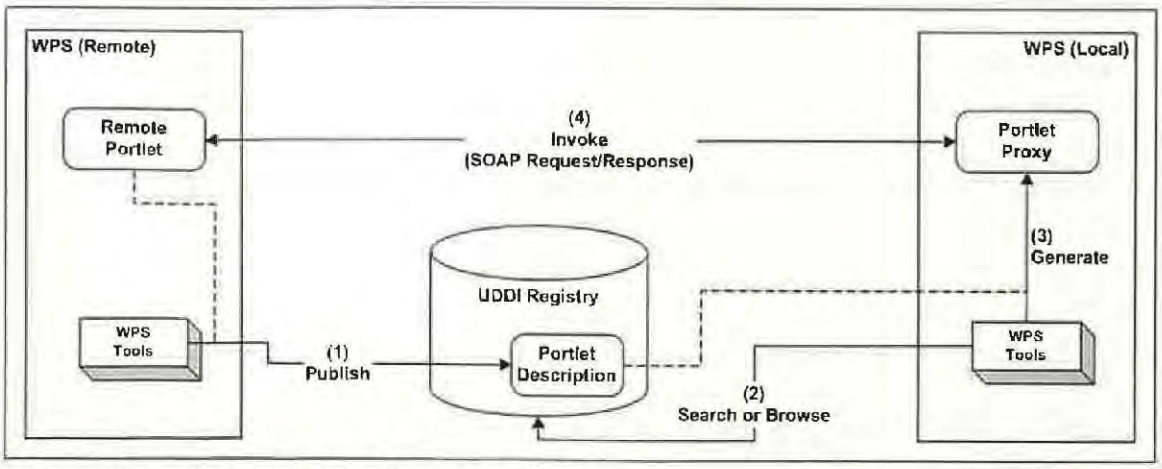


Figure 22-3 Report portlet life cycle with WP.

1. Using WP, you publish your portlet metadata to a UDDI repository. This can be either a private UDDI repository that you have set up or a public repository available on the Web. Throughout the chapter, we will mostly use a public test UDDI repository but we will also discuss about using a private repository.
2. As a portal administrator who wants to embed a certain function packaged as a portlet in your own portal, you use the UDDI browser to search the UDDI registry for available portlets.
3. Once you find the desired remote portlet, you use the Web services manager to create a portlet proxy based on the published portlet. You can then add the portlet to any page(s) on the portal.
4. When the portlet proxy is opened on a portal page, it communicates with the remote portal running on your portal to retrieve not just data but also the entire portlet content. The mechanics of these steps are interesting. The portal server calls the portlet proxy passing it the normal `PortletRequest` and `PortletResponse` objects—for the portal server the portlet proxy is just another portlet. The portlet proxy marshals all arguments into a SOAP request and invokes the remote portal. The remote portal makes use of the information in the request to find the remote portlet and then unmarshals all the information from the original `PortletRequest`. When the response is received by the remote portlet, it is marshaled into a SOAP response and returned to the portlet proxy that returns it to the local portal server.

We now turn to the details of using Web services within WP. While for most part, the chapter focuses on remote portlets, we will start with the most basic scenario—using a regular Web service from within a portlet.

Using Web Services as "Back-End" APIs

There is nothing special about using a Web service call from within a portlet. It merely involves two development tasks: developing the portlet and using the Web service. Because we have already discussed portlet development extensively, our focus in this section will be on how to make use of an existing Web service from your Java program (your portlet code).

Use of a Web service from Java code is supported within WebSphere Studio by the Web Services wizard. The Web Services wizard allows you to use the UDDI explorer to search for a UDDI registry for a Web service, find the service you need and the WSDL file describing the Web services, and then import the WSDL definitions. This import process creates a Java

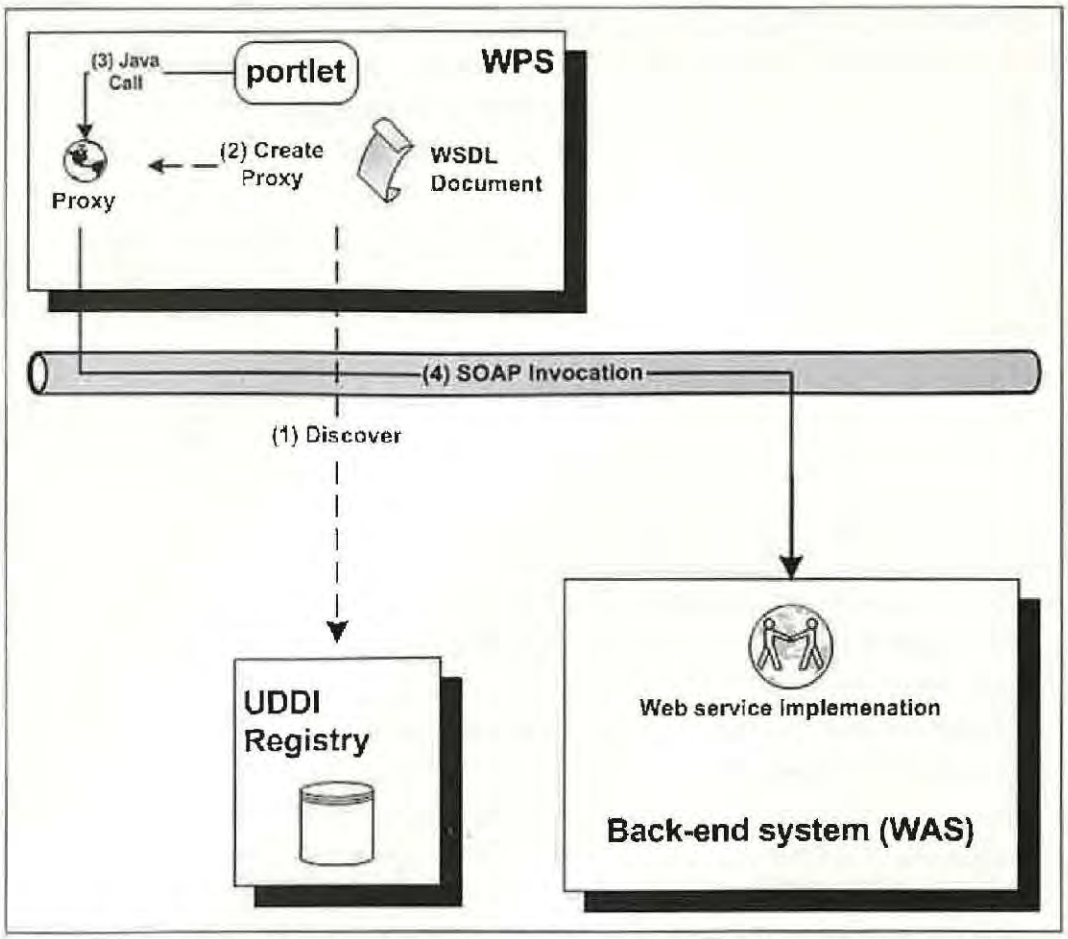


Figure 22-4 Generating and using the Web service proxy from a Java portlet.

proxy object that has the same interface as the object implementing the Web service, allowing you to make calls from within your portlet code to the Java proxy, which then communicates over SOAP to the implementation routine as shown in Figure 22-4.

In the following subsections we walk you through the process shown in Figure 22-4, including registering with a public registry, discovering the Web service, generating the proxy, and invoking the service using the proxy.

Using the IBM UDDI Business Test Registry

UDDI registries allow you to publish businesses and services. Once you publish your Web services to a UDDI registry, they can be discovered and used by potential customers. UDDI registries can be partitioned into private and public registries. A private UDDI registry is one that you can set up and manage yourself. It is usually used to set up a UDDI-based discovery scheme within a company or an organization—an example is the IBM WebSphere UDDI Registry that comes on the CD set of WP. Publishing your

Web service onto a private registry makes it visible only to those who have access to that registry. The IBM UDDI Business Test Registry is a public UDDI registry that can be used to register businesses and services to the public. It is most commonly used to test the publishing and discovering aspects of Web services in a development and test environment.

Before you can publish your Web services to this registry, you will need to set up a user account. Open your browser and navigate to `https://uddi.ibm.com/testregistry/registry.html`. Then click the Get an IBM User ID and Password link. Fill in the registration page; the most important values to remember are the ID and password. Make sure you have the spelling of your e-mail account correct because you will need to receive an activation e-mail.

Once you have a registered user in the IBM UDDI Test Registry, you can proceed to search for a Web service that you would like to use. We will show you how to do this based on a Web service that is already published to the IBM UDDI Business Test Registry as part of the IBM Speed-start Web services program and the Speed-start community. The service we will use is a simple one—merely providing the server clock time or the server on which the Web service is deployed.

Discovering a Web Service

The first thing you need to do is discover the service. Within WebSphere Studio you use the Web services explorer to search a UDDI registry and import Web service definitions. Bring up the Web services explorer by selecting File ⇨ Import from the menu bar. This brings up the Import wizard where you need to select Web Service and click the Next button. Check Launch the Web Services Explorer to find a Web service from a UDDI Registry and make sure that the IBM UDDI Test Registry is selected. Click the Finish button to bring up the Web services explorer in import mode. Once you are in the explorer, click the Find tool in the top-right pane. Then select Businesses from the drop-down list and SpeedStartExample as the name of the business you wish to find and press Go, as shown in Figure 22-5.

The results will include the SpeedStartExample registry entry and you can proceed to import WSDL into your WebSphere Studio environment. Click on the business that is retrieved by the search. Expand the query result entry in the UDDI Navigator pane and click SpeedStartExample. Click the Get Services tool in the pane's toolbar. You will see the details for the RemoteServerClock service as shown in Figure 22-6. Click the Get Service Interface tool and then the Import WSDL to Workbench tool. Select a project name and click Go. This imports the WSDL definition into a file within your project from which you can go ahead and generate the Java proxy that will enable you to make the remote call.

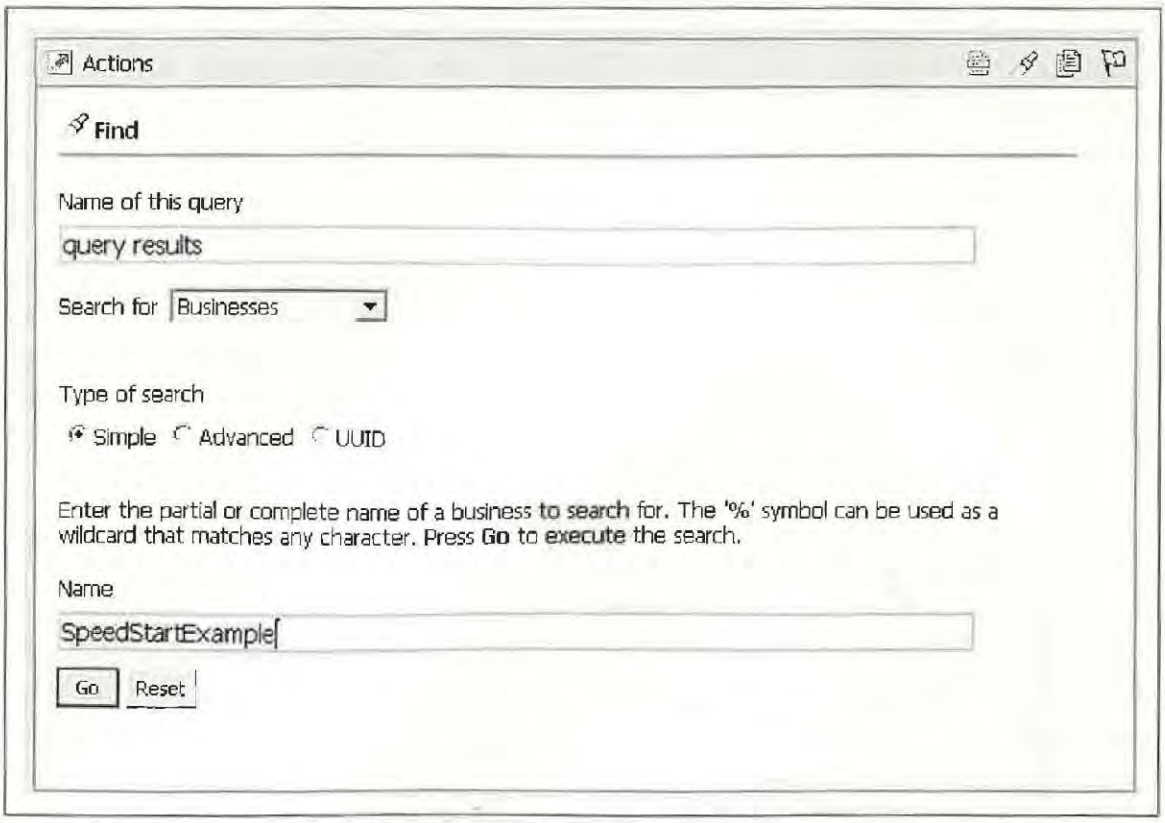


Figure 22-5 Searching for the Web service.

Generating a Java Proxy

To create the proxy, you also use the WebSphere Studio Web Services wizard. Follow these steps:

1. Select File ⇨ New ⇨ Other from the menu bar.
2. Select Web Services in the left pane and Web Service Client from the right pane.
3. Select Java Proxy from the drop-down menu and check Test the Generated Proxy. Click Next.
4. Enter the WSDL binding document.
5. Click Finish.

This will generate a proxy that you can use from your local programs to invoke the service remotely. The wizard also asks you if you want to generate a test application. The test application comprises a set of simple JSPs that provide a simple UI making calls to the proxy and the remote Web service—it is a convenience to allow you to test that everything is functioning correctly.

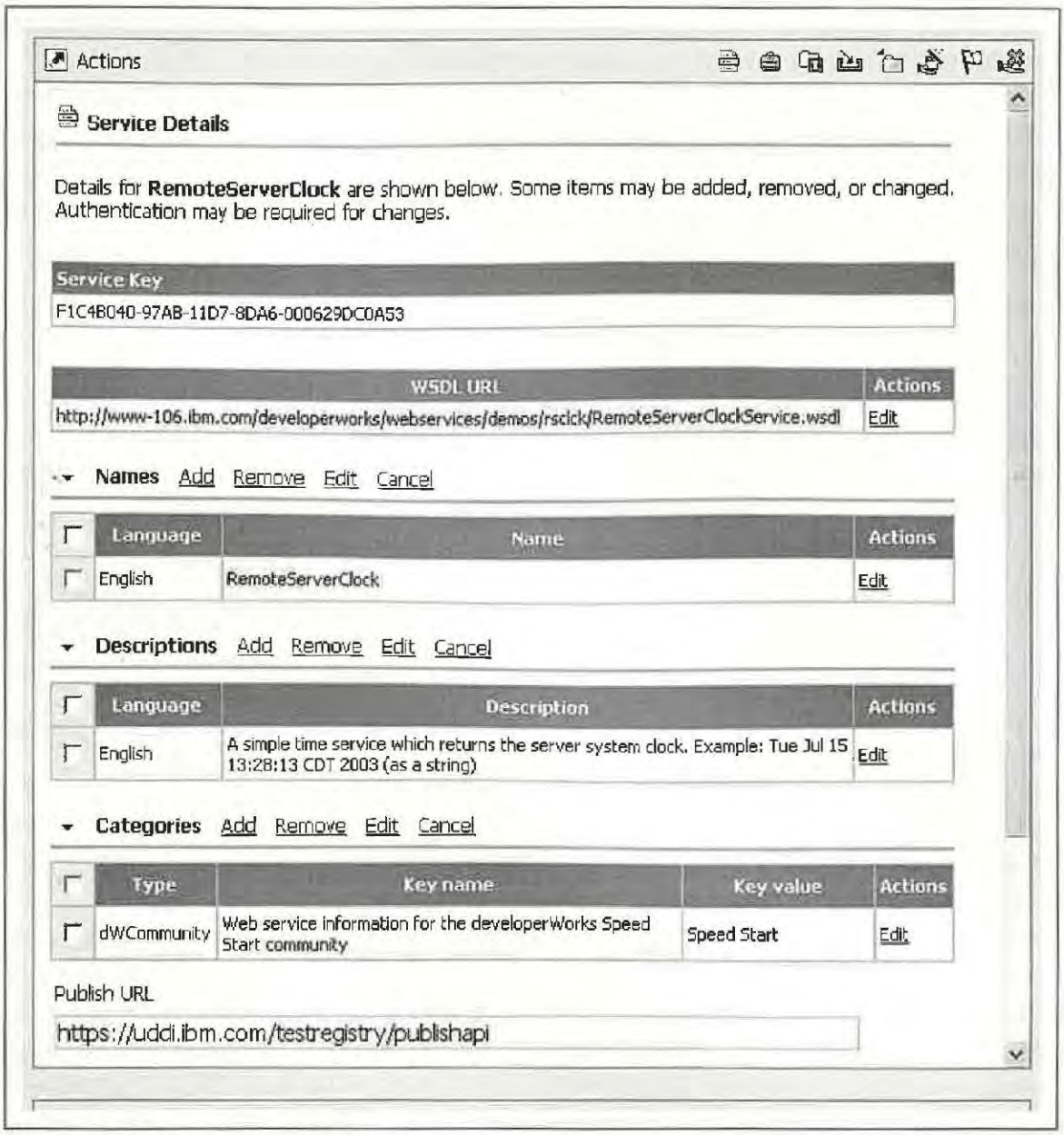


Figure 22-6 Importing the WSDL.

This simple process is all you need to follow in order to make Web service calls from your own portlets. While this is indeed quite simple, remote portlets and WSRP eliminate even this small amount of work.

Using WSRP within WP

WP support for WSRP services is based on producers of content and consumers of content. Producers are portals that implement WSRP services and that offer one or more proxy portlets. These services are used by remote

portlets—the consumers. A consumer is a portal that uses WSRP interfaces to incorporate and consume content supplied by producers. The consumer receives markup from the WSRP service and presents it as a native portlet.

In order for this to occur, producers need to provide enough information about the services they provide. This is done through a set of WSRP interfaces that are used by consumers; note that not all of these interfaces are mandated by the WSRP standard for remote portlets to be able to work:

- Service description. A producer *must* provide an interface through which a consumer can interrogate which portlets are available for use. WP supports the service description interface.
- Markup. A producer *must* provide an interface through which a consumer can request markup fragments forming the content of the remote portlet. This interface must support the various states a portlet may be in. WP supports the markup interface.
- Portlet management. A producer *may* allow consumers to manage state of the proxy portlet including its persistent state. WP supports the portlet management interface.
- Registration. A producer *may* provide registration capabilities for use by consumers. WP does not support the registration interface.

In the technology preview provided in version 5.0.2, configuration is done using the XML configuration interface. This is the only tested and supported method or configuration WSRP services at this time.

The XML configuration interface includes updates that are required for configuring WSRP, including a new portal resource type called `wsrp-producer` allowing you to browse and integrate remote portlets using the producer interface. You can also access WSDL documents describing the remote portlets by using another added element: the `wsdl-url` element within the `wsrp-producer` element. The WSDL description itself will be accessed at runtime by using the `http://<host>/WP_ROOT/wsdl/wsrp` URL where WP_ROOT is the root directory of your WP installation. Finally, a new configuration element called `preferences` has been added to the XML configuration interface allowing you to define which user attributes you want transferred in the WSRP communications with the producer. `Preferences` is an extension of the name-value element supporting multivalued names. This allows you to associate one or more child elements with portal resources such as `wsrp-producer`, which is important given that you will typically have many consumers per producer.

In addition to these added elements, some existing resource types have been changed in the XML configuration interface:

- `portlet`. WSRP-based portlets will not reside within your existing WAR files. Therefore, this XML configuration interface element has

been changed, allowing you to specify a `handle` attribute and specify that a portlet is being implemented as a remote portlet using a new flag called `provided`.

- `portlet-app`. Similarly, when a portlet application containing multiple portlets is provided from a remote implementation, the application needs to provide a `group-id` attribute corresponding to the `concrete-portlet-app` tag normally found in `portlets.xml`.

WP provides WSRP support for both producers and consumers. On the producer side, WP allows you to take an existing portlet and expose it as a WSRP producer, which can then be used by a remote consumer. On the other side, WP allows you to take existing WSRP services and include them as local portlets within your portal.

Configuring WP for WSRP

Regardless of whether you want to set up producers of WSRP services or consume WSRP services, you first need to enable WSRP within your portal. In order to support both of these functions, you need to perform the following settings in `WP_ROOT/shared/app/config/services/ConfigService.properties`:

1. Set `portal.enable.jsr168=true`.
2. Set `portal.enable.wsrp=true`.
3. Restart the portal instance.

In addition to these mandatory settings, there are a few additional configuration settings that may be appropriate for your environment. These include proxy settings and security settings.

In terms of proxy settings you may want to configure a proxy for WSRP. Unfortunately, you cannot specify a proxy for WSRP communication only—this will affect all communications. To do this you need to change the JVM arguments for the WP/WAS instance using `-DproxyHost` and `-DproxyPort`.

In terms of advanced security configuration, you can enable WSRP communications over SSL and you can implement additional security for URLs exposed through WSRP. To enable SSL for WSRP communications, you first need to enable SSL for your WP and WAS instances. Then you need to activate WSRP security by setting `wsrp.security.enabled=true` in `WP-ROOT/shared/app/config/services/ConfigService.properties`.

Controlling security for URLs exposed through WSRP is important because WSRP allows you to use content generated by a remote producer. In

such scenarios you have little control over the links provided in the markup and, therefore, where your users may be taken in terms of link navigation. You provide this control by implementing the `WSRPResourceURLFilterFactory` interface and register your class with WP. This registration is done through the `wsrp.resource.url.filter.factory` property in `WP_ROOT/shared/app/config/services/ConfigService.properties`, which should name the class you create. The interface to be implemented is shown below and is used by consumers as they process resource URLs:

```
package com.ibm.wps.wsrp.Consumer.security;

/**
 * This interface describes a factory that can be used
 * to provide a <code>java.io.Writer</code> and
 * <code>java.io.Reader</code> which are used to encode and decode
 * resource URLs used by the WSRP Consumer. The reader and writer
 * implementation should be symmetric.
 *
 * This provides a plug-point for custom resource URL filters
 * that can, for example, be used to encrypt and decrypt WSRP
 *resource URLs.
 **/
public interface WSRPResourceURLFilterFactory
{
    /**
     * Get the <code>java.io.Writer</code> used to filter the passed
     * Writer. WSRP Resource URLs are written to that Writer.
     * An implementation of this Writer could for example encrypt
written
     * resource URLs. It is important that writing to that Writer does
     * not generate characters that are reserved in URIs. See
     * <a href="http://www.ietf.org/rfc/rfc2396.txt">RFC 2396</a> for a
list
     * of reserved URI characters.
     *
     * @param out The underlying character stream
     * @return A <code>java.io.Writer</code> which can be used to modify
     * WSRP resource URLs written to it.
     **/
    public java.io.Writer getUrlFilterWriter(java.io.Writer out);
    /**
     * Get the Reader used to modify the characters passed to the
Reader.
     * Requested WSRP Resource URLs are read from that Reader.
     * An implementation of this Reader could, for example, decrypt
     * WSRP resource URLs which have been encrypted using the previous
Writer.
     *
     * @param out The underlying character input stream
```

```
     * @return A <code>java.io.Reader</code> which is used to modify
WSRP
     * resource URLs read from it.
     **/
    public java.io.Reader getUrlFilterReader(java.io.Reader in);
}
```

Setting Up WSRP Producers

WSRP producer support within WP includes the entire remote portlet life cycle including adding services, removing services, and exporting portlets. In order to add a WSRP portlet service, use the XML configuration with the following XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<request type="update" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="PortalConfig_1.2.xsd">
  <portal action="locate">
   <web-app action="locate" active="true" uid="portlet.war.webmod">
    <portlet-app action="update" uid="stdTestsuite.war">
     <portlet action="update" name="TestPortlet1"  provided="true" />
    </portlet-app>
   </web-app>
  </portal>
 </request>
```

In order to remove a service, use a similar XML command except for changing the `provided` attribute to `false`:

```
<?xml version="1.0" encoding="UTF-8" ?>
<request type="update" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="PortalConfig_1.2.xsd">
  <portal action="locate">
   <web-app action="locate" active="true" uid="portlet.war.webmod">
    <portlet-app action="update" uid="stdTestsuite.war">
     <portlet action="update" name="TestPortlet1"  provided="false" />
    </portlet-app>
   </web-app>
  </portal>
 </request>
```

Finally, to export the WSRP portlets use the XML file provided in `WP_ROOT/doc/xml-samples/ExportAllPortlets.xml` and search for WSRP producers or portlets that have the `provided` attribute set to `true`:

```
<?xml version="1.0" encoding="UTF-8"?>
<request
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="PortalConfig_1.2.xsd"
```

```
    type="export">
    <portal action="locate">
        <wsrp-producer objectid="*" action="export"/>
    </portal>
</request>

<?xml version="1.0" encoding="UTF-8"?>
<request
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="PortalConfig_1.2.xsd"
    type="export">
    <portal action="locate">
        <portlet provided="true" action="export"/>
    </portal>
</request>
```

Setting Up WSRP Consumers

To set up a consumer you first need to create a producer instance within WP. This may be a little confusing at first but WP needs to know about the service definitions or the producer's WSDL definitions in order for a consumer to be created. To create the producer specification using a WSDL definition, use the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<request type="update" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="PortalConfig_1.2.1.xsd" create-
oids="true">
  <portal action="locate">
    <wsrp-producer action="update" registration-required="false"
uniquename="wps.myProducer">
     <wsdl-url>
       http://hostname/wsdl/wsrp_service.wsdl
     </wsdl-url>
     <localedata locale="en">
       <title>My Producer</title>
     </localedata>
   </wsrp-producer>
  </portal>
</request>
```

Alternatively, you can create the producer using a service description URL and a markup URL. This is useful when you don't have direct access to the remote portlet and its WSDL when creating the producer and consumer. You will need the remote portlet to be available at runtime in order to use it, but you may not need it when you are setting up the consumer. In this case use the following XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<request type="update" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
```

```
instance" xsi:noNamespaceSchemaLocation="PortalConfig_1.2.1.xsd" create-
oids="true">
  <portal action="locate">
    <wsrp-producer action="update" uniquename="wps.myProducer">
      <service-description-url>
        http://hostname/wps/WSRPServiceDescriptionService
      </service-description-url>
      <markup-url>
        http://hostname/wps/WSRPBaseService
      </markup-url>
      <localedata locale="en">
        <title>My Producer</title>
      </localedata>
    </wsrp-producer>
  </portal>
</request>
```

Once you create the producer definition, you can go ahead and define the consumer using this producer. In order to do this you need to provide the `handle` and `groupID` for the remote portlet:

```
<?xml version="1.0" encoding="UTF-8"?>
<request type="update" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="PortalConfig_1.2.1.xsd">
  <portal action="locate">
    <wsrp-producer action="locate" uniquename="wps.myProducer">
      <portlet-app action="update" active="true"
groupid="_2_00KJL5N5NE0U0FIA_LT">
        <portlet action="update" active="true" defaultlocale="en"
              handle="_5_00KJL5N5NE0U0FIA_FN" name="WSRP">
          <localedata locale="en">
            <title>WSRP</title>
            <description>Sample WSRP</description>
          </localedata>
          <access-control externalized="false" owner="undefined"
private="false"/>
        </portlet>
      </portlet-app>
    </wsrp-producer>
  </portal>
</request>
```

Once you define the consumer you can use the normal portal administration to add the remote portlets; at this point there is no difference between these WSRP consumers and other local portlets.

By default WP caches producer service descriptions once they have been defined within a consumer. This allows WP to avoid multiple network requests when numerous services are being used. In some cases you might want to disable this caching while troubleshooting WSRP support. To do this set `cacheinstance.wsrp.cache.servicedescription.enabled`

= false in WP_ROOT/shared/app/config/services/CacheManagerService.properties.

Futures

WSRP support in WP 5.0.2 is a technology preview. Future versions of WP will provide more robustness and more functions than the current implementation. Among these added features you should keep in mind that future versions will provide the following support:

- Complete implementation of JSR 168
- More comprehensive implementation of the OASIS WSRP specification
- Use of UDDI for discovering WSRP services
- Better tools in addition to the XML interface including administration portlets for WSRP configuration

UDDI and tModelKeys

Now that you understand how Web services are used from within a portal in both a pure Web service scenario as well as in a WSRP scenario, it is time to merge all of these concepts. We mentioned that WSRP is a standard that is layered on top of the core WSRP layers (SOAP, WSDL, and UDDI). This means that all producers of WSRP services use the core Web service layers and that consumers create report portlets by using core UDDI and WSDL features. The key to all this is the tModel.

UDDI allows you to describe Web service not only in business and service terms but also in technical terms. The tModel structure provides the ability to describe the specification to which the Web service adheres to including a description of how the service behaves, what conventions it follows, and what standards the service is compliant with. The tModel represents a set of technical specifications including wire protocols, interchange formats, and sequencing rules.

The tModel is the part of the UDDI framework that allows decoupled software components that need to communicate with each other to adhere to a preagreed specification. Such specifications are not dependent on a particular Web service (in fact, it is the other way around). The designers of such specifications can establish a unique technical identity with a UDDI registry, which can then be used by multiple service publications. Each one is registered with the UDDI registry while referencing the appropriate tModel identifier (`tModelKey`).

This is also true for WSRP and report portlet support within WP. A special tModel defines the technical structure for any Web service that is used as a remote portlet, allowing WP to generate the appropriate portlet proxy based on this specification.

Summary

Web services are becoming the mainstream in terms of system integration. Given that portals are often the main integration platform, you need to be able to use Web services from within your portal. This includes both core Web service integration and WSRP-based integration. Initially, you are more likely to use low-level Web services given that WSRP is still in its infancy. However, WSRP has the benefit that it allows you to reuse a complete function including the presentation layer, as opposed to using just a remote function. While the usage of core of both types of Web services is the same, the mechanics within WP are different. In this chapter we showed you how to perform development and setup in both cases so that you are prepared for all types of Web service integrations.

In the next chapter we will tackle the broader issue of integration. We will show you the various methods and tools that you have at your disposal when integrating applications and content to be delivered through WP. Because portals are becoming the "uber-application" for most organizations, application integration is certainly one of the most important topics you will come across when using WP.

CHAPTER

23

Integrating External Applications with WebSphere Portal

In this chapter you will learn about the various options you have for integrating external applications using a WebSphere Portal infrastructure. We will describe a number of approaches to achieve such integration and for each one provide you with enough tools to start doing some of the integration work. We will try to keep the discussion at a generic level that is true to all such applications including integration of custom-built and homegrown applications as well as commercial off-the-shelf (COTS) application suites. Because custom applications will involve custom APIs and integration, our examples will focus on integrating packaged business applications such as SAP, PeopleSoft, Siebel, J.D. Edwards, Oracle, and so on.

Why Discuss Integration?

Portals are about integration. A portal is really just an integrator and an aggregator of information and applications. In an enterprise environment, portals are quickly becoming the new "corporate desktop" through which functions related to human resources applications, finance applications, and corporate communications are delivered to all employees. As another example, consumer portals try to be a one-stop-shop for all activities and offer links to many other applications, Web sites, and easy-to-use search engines.

Figure 23-1 is a conceptual depiction of what portals try to accomplish. Some people call this aggregation and others call this unification—but whatever your preference, it is really all about integration of other applications, systems, data stores, and so on. Without robust integration capabilities, a

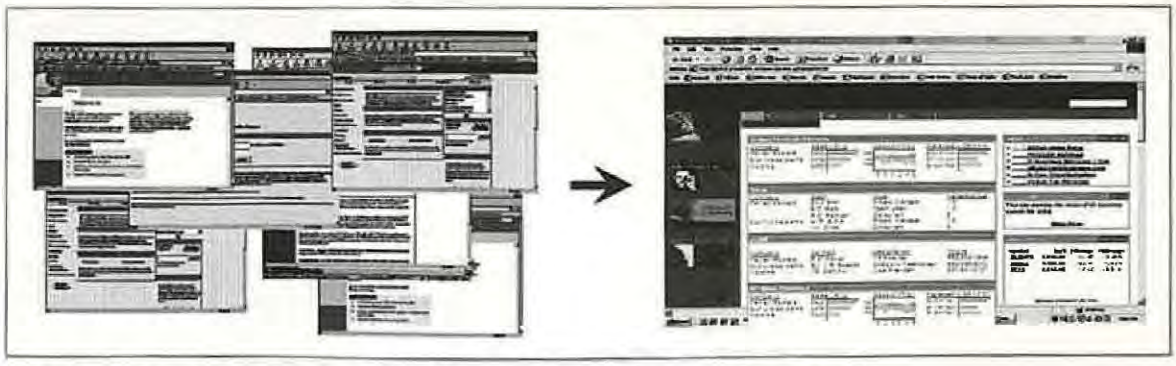


Figure 23-1 Integrated applications delivered through the portal.

portal is just an empty shell that allows you to manage a Web user interface. With many applications integrated into the portal infrastructure, the portal becomes *the* application delivery platform. To exemplify this further, the personalization features of WP would be meaningless without integrated environments. What exactly would you personalize?

Portals deliver Web-based content to a user. Applications generate content that needs to be used by users. Therefore, any portal platform needs to provide good integration tools for allowing you to expose application content through your portal. This is why integration is such a crucial topic for any portal platform and why we devote this chapter to an overview of WebSphere's (and specifically, WP's) integration capabilities. We should mention that integration, and WebSphere integration specifically, is a very broad topic—one on which numerous books can be (and have been) written (for an excellent reference see *Enterprise Integration: An Architecture for Enterprise Application and Systems Integration* by Fred Cummins). Therefore, we do not provide a comprehensive discussion on integration within WebSphere. Instead, we focus on what kind of integration approaches you can use specifically within WP and provide you with tools to help you decide which integration pattern is right for you and how to start such an integration project.

What Options Can You Choose from?

When looking to integrate your applications into the portal, the first thing you need to do is determine which of the many options is most appropriate for your specific case. We will therefore first outline the possibilities you can choose from and provide a simple classification that can help you understand the big picture before delving down into the details. Throughout the rest of this chapter we will provide more details on each of these integration patterns. At the end of the chapter we will summarize with some

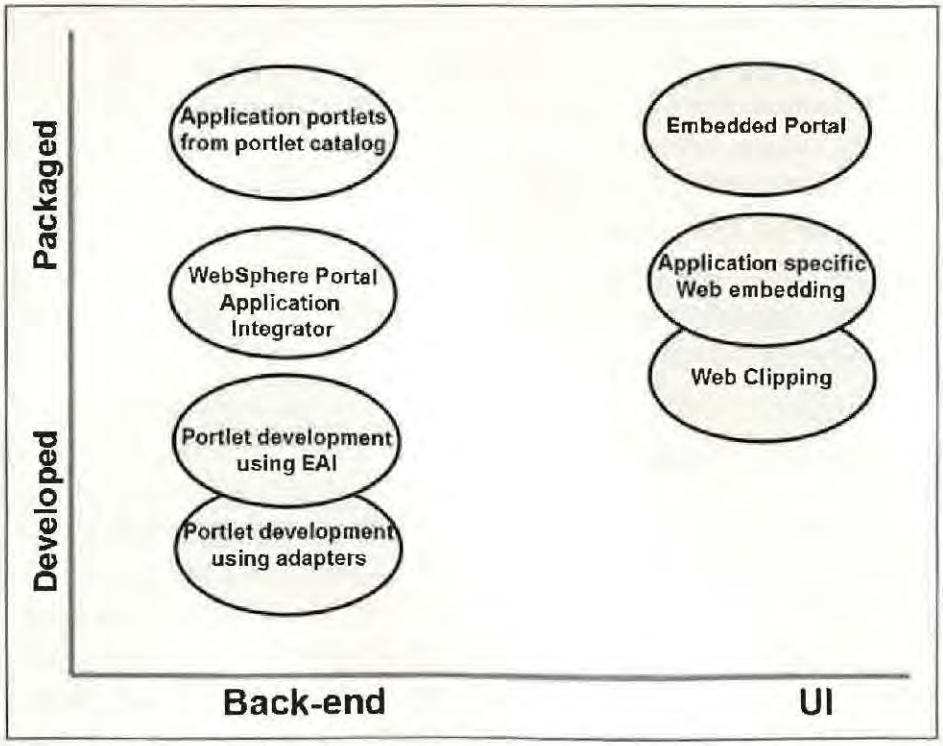


Figure 23-2 Classification of integration patterns within WP.

basic guidelines for choosing the right integration pattern based on some simple characteristics.

As Figure 23-2 shows, we chose to classify the major WP integration patterns based on two dimensions. The first dimension is developed versus packaged. Many of the tools we will show you in this chapter allow you to make calls into enterprise information systems and applications but require that you develop the code that makes these calls and the portlets that display the information returned by these calls. Other integration options include using prepackaged portlets created by IBM or by third parties—portlets that package not only the logic for interfacing with applications but also the presentation layer. It should be clear that such portlets can only exist for applications with a large installed based—greater chances are that there will be no packaged portlets for custom-built and proprietary applications used by your company.

Our classification makes use of a second dimension: back-end versus UI-based integration patterns. In back-end integration patterns, adapters are used by your code to make remote invocations or send messages to the application. This allows you to retrieve information and make changes to the data stored within the application. These patterns are called back-end integration because the application is front-ended by your code. In our context, the application sits in the back-end and is front-ended by portlets deployed on the portal. Because these patterns make use of application adapters, we will sometimes refer to these patterns as adapter-based integration patterns.

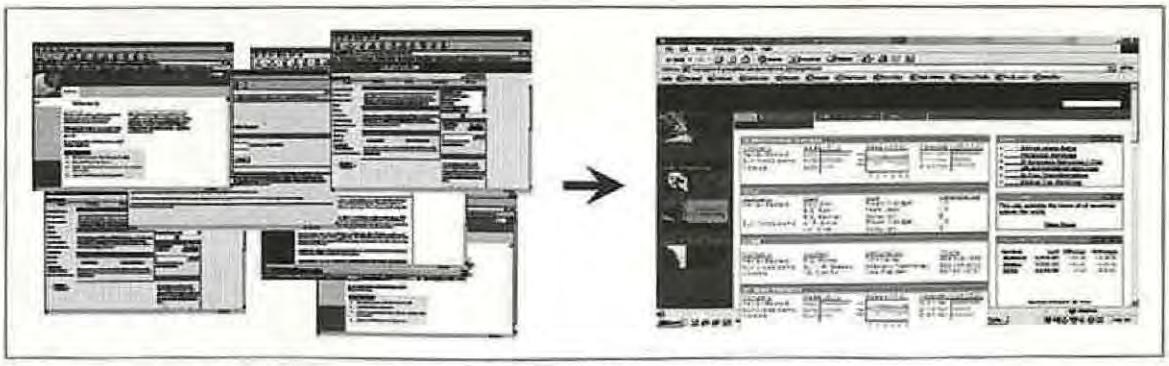


Figure 23-1 Integrated applications delivered through the portal.

portal is just an empty shell that allows you to manage a Web user interface. With many applications integrated into the portal infrastructure, the portal becomes *the* application delivery platform. To exemplify this further, the personalization features of WP would be meaningless without integrated environments. What exactly would you personalize?

Portals deliver Web-based content to a user. Applications generate content that needs to be used by users. Therefore, any portal platform needs to provide good integration tools for allowing you to expose application content through your portal. This is why integration is such a crucial topic for any portal platform and why we devote this chapter to an overview of WebSphere's (and specifically, WP's) integration capabilities. We should mention that integration, and WebSphere integration specifically, is a very broad topic—one on which numerous books can be (and have been) written (for an excellent reference see *Enterprise Integration: An Architecture for Enterprise Application and Systems Integration* by Fred Cummins). Therefore, we do not provide a comprehensive discussion on integration within WebSphere. Instead, we focus on what kind of integration approaches you can use specifically within WP and provide you with tools to help you decide which integration pattern is right for you and how to start such an integration project.

What Options Can You Choose from?

When looking to integrate your applications into the portal, the first thing you need to do is determine which of the many options is most appropriate for your specific case. We will therefore first outline the possibilities you can choose from and provide a simple classification that can help you understand the big picture before delving down into the details. Throughout the rest of this chapter we will provide more details on each of these integration patterns. At the end of the chapter we will summarize with some

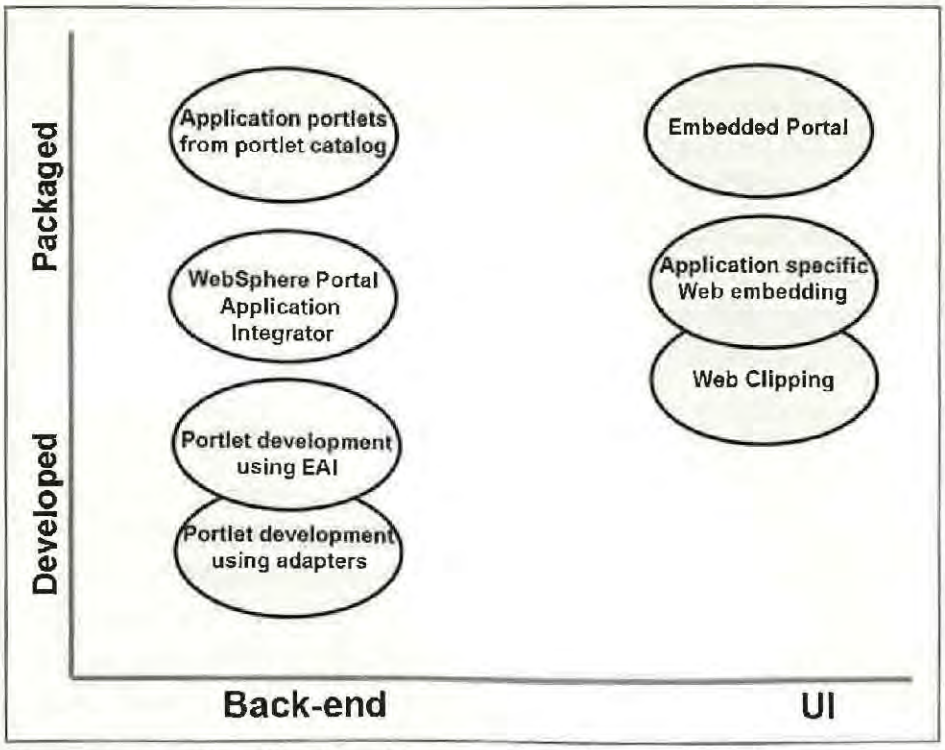


Figure 23-2 Classification of integration patterns within WP.

basic guidelines for choosing the right integration pattern based on some simple characteristics.

As Figure 23-2 shows, we chose to classify the major WP integration patterns based on two dimensions. The first dimension is developed versus packaged. Many of the tools we will show you in this chapter allow you to make calls into enterprise information systems and applications but require that you develop the code that makes these calls and the portlets that display the information returned by these calls. Other integration options include using prepackaged portlets created by IBM or by third parties—portlets that package not only the logic for interfacing with applications but also the presentation layer. It should be clear that such portlets can only exist for applications with a large installed based—greater chances are that there will be no packaged portlets for custom-built and proprietary applications used by your company.

Our classification makes use of a second dimension: back-end versus UI-based integration patterns. In back-end integration patterns, adapters are used by your code to make remote invocations or send messages to the application. This allows you to retrieve information and make changes to the data stored within the application. These patterns are called back-end integration because the application is front-ended by your code. In our context, the application sits in the back-end and is front-ended by portlets deployed on the portal. Because these patterns make use of application adapters, we will sometimes refer to these patterns as adapter-based integration patterns.

UI integration patterns are fairly recent and make use of the fact that Web-based applications use a consistent interaction model between the Web browser and the application server. All these systems make use of HTTP request/response pairs, URLs, and various markup languages such as HTML and WML. This commonality allows WP to introduce a new breed of integration patterns—patterns that make calls on the front-end of the system and script the UI rather than make use of APIs based in the back end. These patterns can often offer you very quick results in terms of exposing existing Web-based applications on your portal.

Which Tools Are Available to You?

The integration patterns described above can broadly be classified into portlet-based integration methods and adapter-based integration methods. Each of these methods is supported by a different set of tools within the WebSphere world. For portlet-based integration, the major tool you will use is the WebSphere Portlet Catalog available on the IBM developer domain (for example, at `www7b.software.ibm.com/wsdd/zones/portal/catalog`). The catalog includes many portlets that you can download and use—many of which include prepackaged integration options with major application providers such as SAP, PeopleSoft, and Siebel. In addition, a number of special portlets are really integration tools packaged in a way that assists you in exposing other applications as portlets. The most important such tool is the WeSphere Portal Application Integrator.

Adapter-based integration makes use of existing APIs exposed by application providers. Adapters are tools that make use of these APIs and allow you to easily call these APIs from your code. They handle issues such as session handling, calling procedures, and transaction handling so that you don't have to deal with each application provider and its proprietary integration methodology. This makes integration simpler and more consistent and if you master the use of adapters you will find that integrating with one package is not all that different than integrating with another.

Adapter-based integration is based on a different set of tools—a set of tools that is part of the broader family of WebSphere products and not specific to the portal server. An adapter can broadly be classified as Java Connector Architecture (JCA) based or messaging middleware adapters. JCA adapters are deployed on WebSphere Application Server (WAS), in your case on the server that forms the underlying platform for the portal server. The latter category includes adapters used within WebSphere MQ (formerly MQ Series) and adapters used within the CrossWorlds architecture. The actual adapter that you choose will depend on your overall environment and your familiarity with these platforms. In addition to these infrastructures,

you will find that WebSphere Studio Application Developer Integration Edition (WSADIE) is a useful tool for developing the integration code.

Developing Portlets Using Adapters

Integration of external applications using adapters appears in the lower left-hand quadrant of our taxonomy shown in Figure 23-2. This integration pattern involves quite a bit of development because it requires two distinct implementation/development tasks. The first involves writing code that makes use of an adapter to connect to the APIs exposed by the external application and the second involves writing the portlets that make use of this information. While both parts of this integration code are written in Java, they are distinct pieces of code and should be viewed in this manner. They could (and probably should) be developed by different individuals or groups and the development involves different tools. Specifically, portlet development can be done in any WebSphere Studio environment that has the portlet toolkit installed while the code interfacing with the external application—(sometimes called an enterprise information system (EIS))—is best done within WebSphere Studio Application Developer Integration Edition (WSADIE). Because we have already discussed portlet development and the portlet toolkit in Chapters 14 through 19, the focus in this section will be on the back-end integration aspects and using WSADIE.

JCA Resource Adapters

While there are many adapters for enterprise systems that can be used, J2EE defines a standard architecture for building adapters: the J2EE Connector Architecture (JCA). The JCA standard provides a mechanism to store and retrieve enterprise data in J2EE and is akin to the standardization provided by JDBC to access relational databases. JCA is relatively new in the J2EE landscape and has already been highly successful in making enterprise integration easier to accomplish.

In order to use JCA you must have a JCA resource adapter. A JCA adapter is specific to an EIS (for example, SAP or PeopleSoft) and is contained in a resource adapter archive (RAR) file composed of the jar files and native libraries necessary to deploy the resource adapter on a J2EE container. JCA adapters interact with a J2EE server and manage things such as connections, transactions, and security. Connection management includes establishing, pooling, and tearing down connections, allowing listeners registered with a connection to receive events, and so on. The underlying protocol an adapter uses to connect to an EIS is outside the scope of the JCA specification.

Transaction management includes support for both distributed transactions providing a mechanism to propagate transactions that originate from inside an application server to an EIS system as well as local transactions existing only on a particular EIS resource. Finally, security allows the application server to connect to an EIS system using security properties defined either at deployment time or at runtime.

JCA resource adapters are available for major packaged applications. As an example, we will continue the discussion of this integration pattern while using the WebSphere Adapter for mySAP.com to connect to a SAP R/3 system. At the time of writing this chapter, this adapter supported SAP R/3 release 3.1H and higher, 4.0B and higher, 4.5A and higher, and 4.6A, 4.6B, and 4.6C. In addition, you should download the WebSphere Adapter for mySAP.com from the IBM Web site; there is no charge involved.

The JCA adapter for SAP provides a standards-based connection architecture for interfacing with SAP systems—and specifically, access to Remote Function Call (RFCs) and Business Application Programming Interfaces (BAPIs). RFC is the protocol used by SAP for remote communication, that is, for communications between two independent SAP systems or for communications between an SAP system and a non-SAP system. Programs using RFCs can either make or receive RFC to or from an SAP system. SAP R/3 systems contain a set of business objects, which are an object-oriented representation of real-world business objects. Examples of business objects are a purchase requisition, a G/L account, or an employee. BAPIs provide a programming interface to such business objects and can be accessed from Java programs using the JCO Java library provided by SAP.

WebSphere Adapter for mySAP.com and WSADIE

WebSphere Studio Application Developer Integration Edition, as its name suggests, packages a large number of tools for helping you with integration tasks. Once you install the WebSphere Adapter for mySAP.com, you can browse BAPIs and RFCs from the SAP Business Object Repository and create Web Service Definition Language (WSDL) representations of the desired function modules. You can then use the WSDL representations to generate Java proxies for BAPIs/RFCs both as Java beans and stateless session EJBs. These beans can be used by your portlets.

Given that you have downloaded the WebSphere Adapter for mySAP.com from the IBM Web site and that you have a working version of an SAP R/3 system, the next thing to do is install the adapter into the appropriate WebSphere Studio environment—in this case into WSADIE. This is important because the JCA adapter not only provides a runtime environment for making the calls to SAP but also includes a tool plugin, which will make using the adapter much easier. Installing the mySAP.com adapter involves

running the installation file and then performing the following procedure within WSADIE:

1. Select Import from the File menu.
2. Select RAR file and click Next.
3. Specify the location of the WebSphere Adapter for mySAP.com RAR (`sap.rar`).
4. Check the Standalone connector project box and specify a project name.
5. Click Finish to import the resource adapter.
6. Open the Server Configuration perspective.
7. Create a server instance and a server configuration (EE Test Environment).
8. Expand Server Configurations in the Server Configuration view.
9. Double-click your server configuration.
10. Click the J2C tab and click Add beside the J2C Resource Adapters list in the server settings.
11. Select the name of your resource adapter (specified in step 6) from the Resource Adapter Name drop-down list. Click OK.
12. Click Add beside the J2C Connection Factories pane. On the Create Connection Factory window enter a name, a JNDI name for the connection factory. Click OK.
13. In the Resource Properties pane enter the SAP R/3 connection properties.
14. Close the editor and click Yes to save the changes.
15. Make sure to add the following entry to the PATH environment variable:

    ```
    <WSADIE ROOT>\WORKSPACE_NAME>\<CONNECTOR_PROJECT_NAME>
    \connectorModule\windows
    ```

    If the PATH environment did not include this segment then you will have to exit WSADIE and restart it for the changes to take effect.

Your server configuration should look similar to that shown in Figure 23-3. Having finished the installation, you can start using the JCA Tool Plugin to connect to SAP, find the proper BAPIs, build WSDL files to describe the APIs, and generate proxy beans for making the call from your portlets. The JCA Tool Plugin makes use of WSDL to describe an EIS-specific binding and to automatically generate a WSDL definition from the metadata

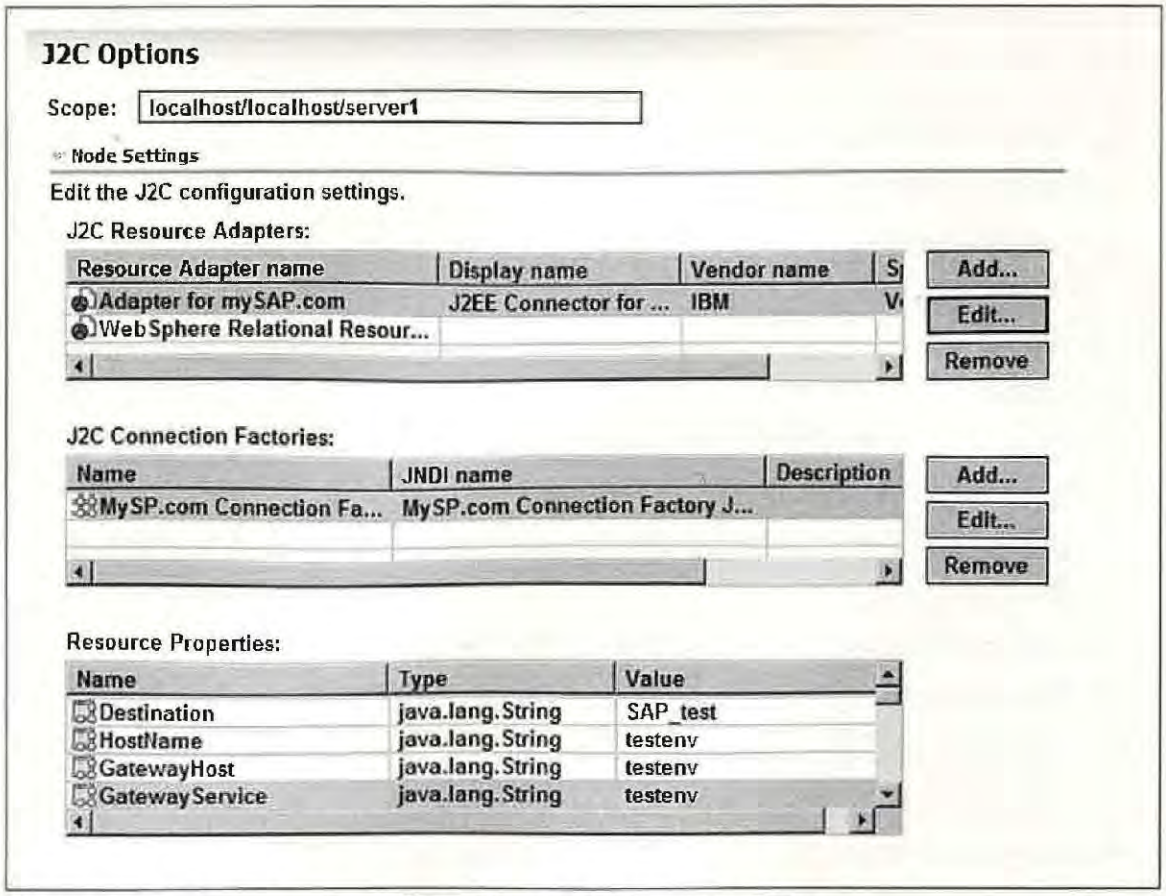


Figure 23-3 Server configuration within WSADIE—node settings.

exposed by the EIS as well as automatically generate a proxy based on this WSDL. This allows you to decouple yourself from the actual EIS and think of the APIs available through the adapter as being similar to Web services—at least in terms of the way they are described.

Describing the Interface Using WSDL

WSDL is perfectly suitable for describing any service (in fact, it is easier to think of it without the "W"—i.e. as Service Definition Language). The authors of the JCA Tool Plugin recognized this fact and made use of WSDL to describe which services are offered by a specific EIS instance and how to access them. Figure 23-4 shows the mapping between WSDL constructs and JCA constructs. The service created by the JCA Tool Plugin (described by the WSDL document) is called an SAP service.

Generating the SAP Service and Using an SAP Proxy

To generate an SAP service you need to follow the steps given below:

1. Open the Service Project wizard by selecting File ⇨ New ⇨ Project.

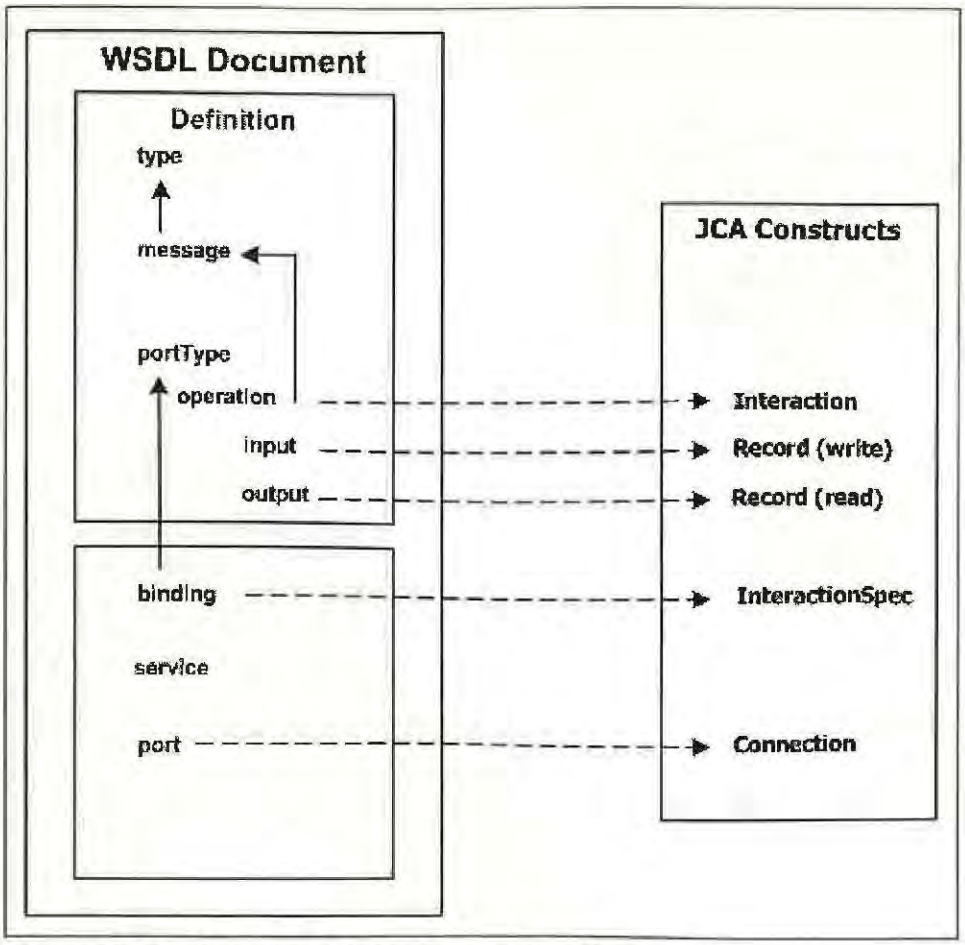


Figure 23-4 Mapping WSDL constructs to JCA constructs.

2. Select Business Integration from the left pane and Service Project from the right pane. Click Next.
3. Enter the project name and click Finish.
4. The Service Provider browser opens with a list of available service providers. Select SAP Services. The Connection Properties page opens.
5. In the "SAP Import" dialog, specify the necessary SAP R/3 connection properties for your SAP instance. For a system without load balancing, the required properties are: `hostName`, `systemNo`, `client`, `userId`, and `language`. Click Continue.
6. In the following page, specify a query string to list the desired BAPIs/RFCs or type in a * for a catch-all query. Click the List Port Types button to establish a connection with the SAL R/3 system and query the available BAPIs/RFCs that match the query string.
7. Select the BAPIs/RFCs for which you want to generate the service definitions by checking the BAPIs/RFCs you want to use. Then click the Add Service button.

8. Accept the default values and click the Finish button; this generates the WSDL representation.
9. Switch back to the Service Provider browser and repeat steps 7 and 8 for each BAPI and RFC you require. The wizard will create two WSDL files for each BAPI/RFC.

WSADIE creates a service interface file, a service binding file, and a schema definition file, all very similar to those generated for a Web service. (See `www-106.ibm.com/developerworks/views/webservices/tutorials.jsp` for a list of tutorials on Web services and WebSphere Studio.) In addition, WSADIE creates a `.ser` file with serialized information about the SAP R/3 instance. Note that while there is a service file and a binding file per BAPI/RFC, there is only a single schema file common to all.

Once you have the WSDL definitions, you can use the service binding file to generate a proxy bean called an SAP proxy by invoking the Service Proxy wizard:

1. Right-click on one of the service binding files and select New ⇨ Proxy.
2. Click Next on the first page of the wizard. On the second page of the wizard, select the Command bean option and check the BAPI/RFC. Click Finish to generate the proxy.
3. Repeat steps 1 and 2 for every BAPI/RFC for which you want to generate a proxy.

You can then use the proxy from your portlet code to get access to data from the SAP system. In a runtime environment (after packaging your code and deploying it on an instance of WP/WAS), this will look like Figure 23-5.

If you want to create a stateless session EJB as a wrapper to the BAPI/RFC, make sure you have already created an Enterprise Application project and within this project do the following:

1. Right-click the service binding file for the BAPI/RFC and select New ⇨ Deployment.
2. The WSDL, service, and port type will be filled in from the values stored in the service binding file. Make sure that the EAR, EJB, and Web project values are correct. If you wish to enable the proxy function as a Web service (perhaps to be used later by a remote portlet), select SOAP as the incoming binding type. Click Next.
3. Click Next on the next wizard page leaving the JNDI name as default. Specify the SOAP binding properties and click Next. Specify the SOAP port address and click Finish.
4. By following these steps you have created both an EJB binding file and a SOAP binding file. The deployment descriptor for the EJB

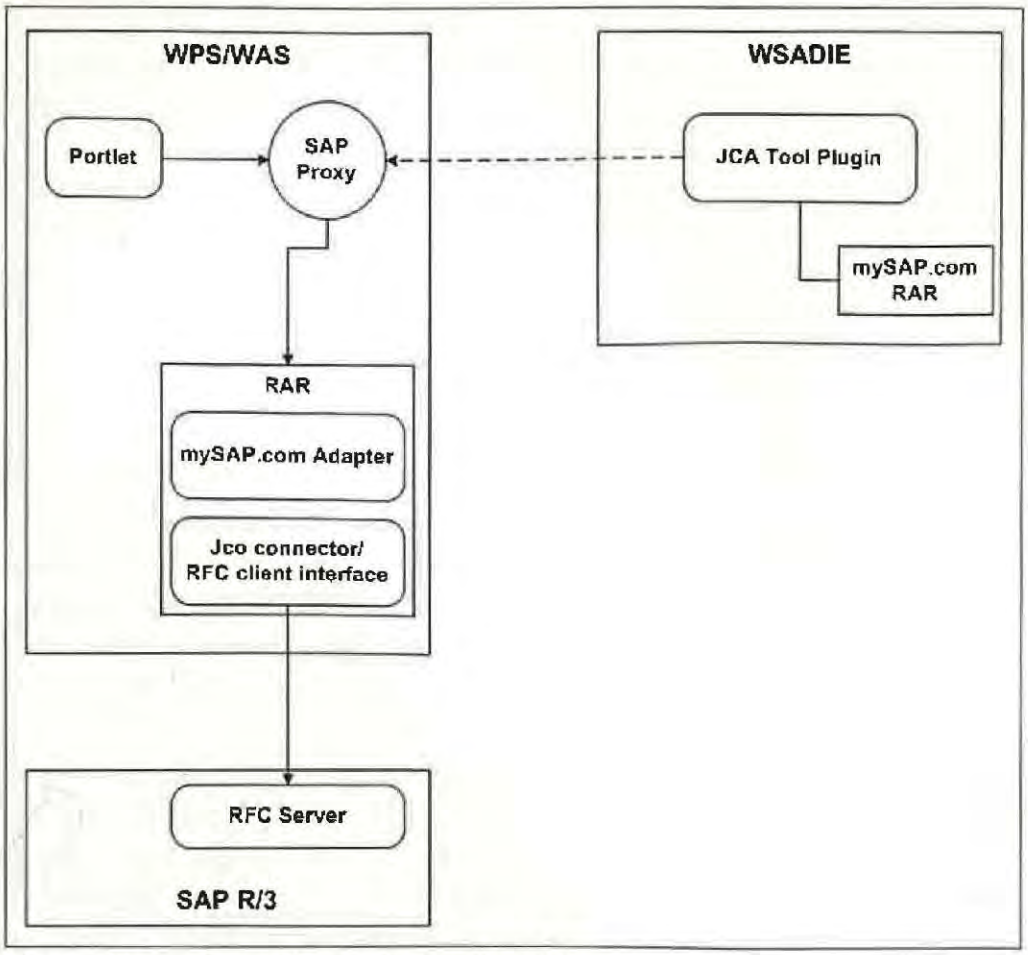


Figure 23-5 Generating and using the SAP proxy.

references the SAP connection factory. In order to bind this resource reference to a physical connection factory, click the J2EE View tab and expand EJB Modules, then right-click EJB and select Open with => EJB Extension Editor. Click the Bindings tab, expand the EJB entry and the service entry, and select ResourceRef.

5. Enter the JNDI name you selected when configuring your environment (for example, eis/SAP). Save your changes; you can now use not only the proxy from your portlet but also the session bean and a SOAP interface from local or remote portlets.

Developing Portlets Using EAI Connectors

Using EAI adapters to connect to external applications is similar to using JCA adapters in that you can view the integration project as comprising two development tasks: developing the code that makes the call using the adapters and developing the portlets that generate the user interface making use of the data and functions exposed through the adapters. The

difference between using JCA adapters and using EAI adapters is an architectural one. When you use the JCA adapter your deployment architecture includes WP running on WAS and the external application. (For example, you use the WebSphere Adapter for mySAP.com to create code running on your WAS instance accessing an SAP R/3 system.) When you use an EAI adapter your architecture involves another infrastructure component: the messaging middleware.

There are many EAI products available with adapters for WebSphere and adapters for packaged business applications and it is not within the scope of this chapter to even list all the options, much less explore them. Of these many options there are two options that are part of the WebSphere family of products: specifically, WebSphere MQ and IBM CrossWorlds. WebSphere MQ, or MQSeries as it was formerly called, has been a cornerstone of IBM's integration strategy and is very well integrated with WAS. CrossWorlds was acquired by IBM at the end of 2001 and is thus more loosely integrated; using CrossWorlds for your WP integration is, from a technical standpoint, not that different from using other middleware products such as WebMethods, Vitria, and TIBCO.

WebSphere MQ adapters exist for every major business application suite. These adapters typically transform data from application-specific formats to MQSeries messages (usually in XML format) and rely on the messaging broker—WebSphere MQ Integrator—for routing messages and for transforming between different application-specific message structures. Because WebSphere MQ can be directly called from within code running on WAS, you can use this infrastructure to connect to packaged applications from your integration code, which is called by your portlets.

As an example, the WebSphere MQ adapter for mySAP.com uses MQSeries Adapter offering technology to integrate with SAP systems using RFCs, BAPIs, or IDOCs. RFCs and BAPIs were already defined in the previous section; Intermediate Documents (IDOCs) are features of SAP's Application Linking Enablement and are used for message data interaction with SAP. Regardless of whether you use RFCs, BAPIs, or IDOCs, all messages sent to SAP through the WebSphere MQ adapter conform to the SAP XML message format or need to be converted to this format. The adapter includes a set of inbound and outbound templates used for receiving and generating SAP XML messages and a DTD generator used for producing DTD files that define the SAP XML needed to represent the various SAP interfaces involved in the business transactions. The integration architecture using this MQ component is shown in Figure 23-6.

Using EAI connectors as opposed to JCA adapters means that your architecture must include an additional messaging broker component. You may be wondering why you need this added complication and when you would prefer to use this integration pattern rather than using a JCA adapter.

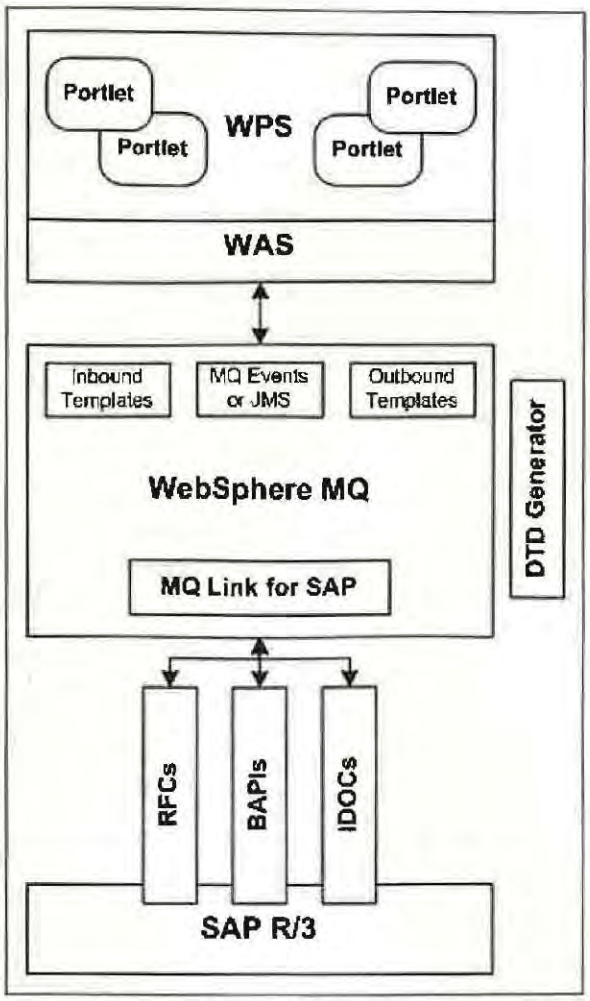


Figure 23-6 Using WebSphere MQ for integrating with SAP.

There are two cases in which you should consider using the EAI integration pattern:

1. You already have an EAI backbone or have decided to invest in an EAI backbone as a strategic decision. In this case you have already made an investment in EAI, you probably have a set of adapters between the applications you use and the EAI backbone, and there is no point in duplicating effort and creating a separate point-to-point integration between your WP infrastructure and your business applications.
2. You need to connect to applications and systems for which there are no mature JCA adapters. MQ Series has been out there for a very long time and there will be more MQ adapters than JCA adapters.

If neither of these cases applies you will find that JCA adapters are easier to set up, easier to develop, and easier to maintain. Keeping your architecture as simple as it can be is always a good idea.

Using the Portal Application Integrator

The WebSphere Portal Application Integrator (WPAI) allows you to create new portlets that use JCA adapters accessing back-end applications without any programming. You create these portlets from within the portal itself. By selecting business objects available from the Enterprise Application, you can specify which data fields should be displayed in your portlet. The WPAI is available from the IBM WebSphere Portlal Portlet Catalog and can be downloaded from `www-3.ibm.com/services/cwi/portal/_pagr/105/`. It supports the following Enterprise Applications:

- PeopleSoft 7.5 with PeopleTools 7.54
- PeopleSoft HRMS 8.30 with PeopleTools 8.18
- Siebel 7 or higher
- SAP R/3

Note that WPAI includes the RARs for Siebel and PeopleSoft but not for SAP; you will have to download the SAP libraries from `http:service.sap.com` if you want to integrate with an SAP R/3 system. The JCA adapters are prepackaged within the WPAI, which means that you do not have to do development to get access to the information and display it within your portlets. The flip side is that you have much less control on what you can do with the information and can typically only allow object-centric behavior—that is displaying data associated with an object, making modifications to a single object, and so on.

Use of the WPAI involves two stages: downloading from the portlet catalog and installing it on your portal and then setting up the configuration attributes for the specific enterprise system you want to use. For example, if you want to use WPAI to expose a PeopleSoft 8 HRMS system, you should follow these steps:

1. Extract the contents of the WPAI zip file and copy `psjoa.jar` and `pstools.properties` from your PeopleSoft server to a new directory.
2. Deploy the `psft.rar` JCA adapter from the WPAI zip file as a J2C Resource Adapter to your WP/WAS instance.
3. Define a connection factory for the PeopleSoft JCA adapter—right-click J2C Connection Factories in the resource adapter entry and create a new factory using a JNDI name of `eis/psft`. Then click the Connections tab and specify the `hostname`, `portNumber`, and `jarPath` to the PeopleSoft server.

4. Extract `psft8adapter.jar` from `psft.rar` and place it in the `lib` folder under WAS. Restart the server instance.
5. Extract the contents of `wps_ps_portlets.zip`.
6. Start PeopleSoft Application Designer. Select File ➪ Copy Project from File to create a new project from the extracted files. Call the project `WPS_PS_PORTLETS` and once saved, exit PeopleSoft Application Designer.
7. Log into the PeopleSoft Web application. Select PeopleTools ➪ Maintain Security ➪ Use ➪ Permission Lists to set the permissions for the component interface. Select the Component Interface tab on the permission list to display the panel for adding component interfaces and add all of the imported component interfaces to a permission list. You can use an existing permission list or create a new one.

Now you need to install WPAI into your WP environment. Log into your WP instance using the administrator password and click Administration. Then click Portlets, and then Install. Select the location of the .war file from the WPAI zip file. You will then see the object builders available for installation as shown in Figure 23-7.

You are now ready to use the WPAI within your WP instance to build portlets. As mentioned, WPAI provides an object-centric view to applications and you first have to select which object to use. You can either use existing portlets that come prepackaged with WPAI for the supported

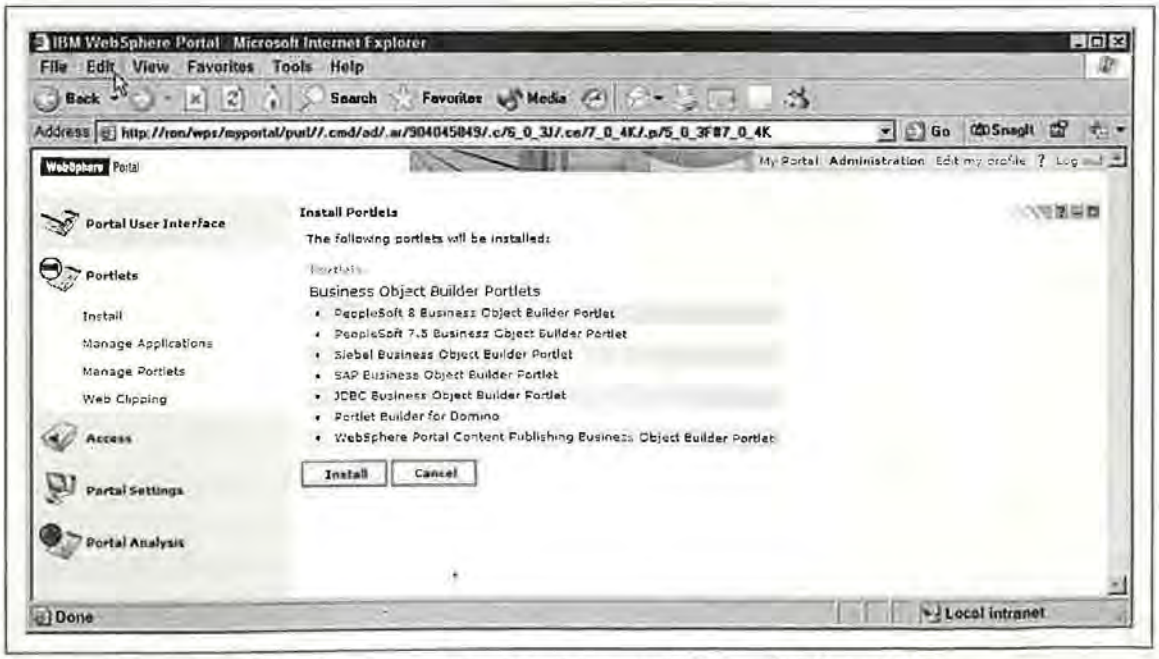


Figure 23-7 Installing business object builder portlets available in WPAI.

enterprise systems or select the Create New Portlet button. You will then be requested to name your portlet and enter the name of your PeopleSoft instance and user credentials. Click Next and then click Create New Business Object. Select the object you need to display within your portlet and define which actions you allow from within your portlet—an action can be search, update, create, and delete. Enter the titles to use when performing these actions within your portlet and click Next.

Once WPAI knows which object your portlet will use, it reads the object metadata using the adapter and presents you with a list of fields. For each field that you want displayed/edited within your portlet, check the appropriate check box, define a user-presentable name to be used as a label, define the field behavior (for example, read only versus editable), and select which field you want to appear on the finder allowing a user to search for objects matching some field criteria. Click Next to complete the creation of your portlet. You can now make this portlet available to your users.

Using the Credential Vault

In the example shown in the previous section, you built a portlet to access objects within PeopleSoft. Note that once you access this portlet at runtime (after adding it to your page), you will first have to enter the username and password to be able to log in to the PeopleSoft instance. Because WPAI front-ends your back-end systems, you need to pass user information to the back-end application. If you do not want this additional login phase, you need to implement back-end single sign-on using the Portal Server Credential Vault (CV). CV consists of vault segments and slots for maintaining the data required to access the back-end system. CV stores secrets such as user ID and password and CV service provides an API for portlets to be able to look up such values (per portal user or per shared secret).

The vault provides a bridge between a physical store and a logical API that is then available to portlets (your code). The API is provided by the `CredentialVaultService` interface, which uses adapters to access the resources themselves. The default adapter that comes packaged with WP uses the portal database to store the resources. You can also use a Tivoli Access Manager (AM) adapter for resources stored in AM and you can build your own adapter by extending the `VaultAdapter` class.

In Chapters 10 and 21 we already dealt with various single sign-on issues and certainly SSO is one of the more important functions supported by portals. Normally when you hear SSO, it is in the context of UI-oriented integration where a Web application is delivered through the portal and you do not want your users to have to log in to the application once they have

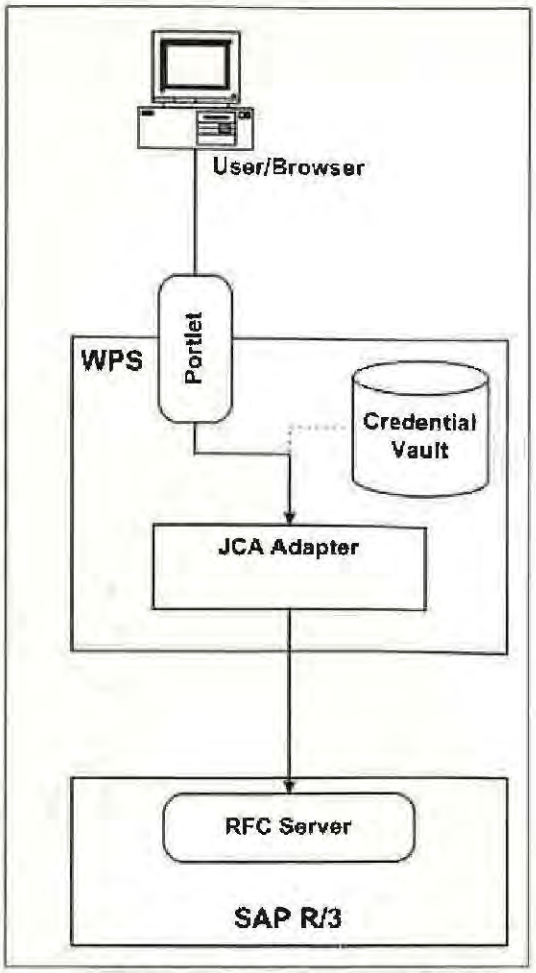


Figure 23-8 Back-end SSO using the CV.

logged into the portal. When using WPAI and adapter-based integration, you will also have to deal with SSO—in this case, back-end SSO.

Back-end SSO means that you will have to make sure that programs running on your WP instance (whether it is code you have developed or whether it is WPAI portlets) use a correct user credential when accessing the back-end system using a JCA adapter, as shown in Figure 23-8. Instead of hard-coding this into your code, you should use Credential Vault—a place to maintain credentials such as those required to access the back-end system.

Credential Vault Segments

CV is one of the built-in portal services within WP providing a mechanism for portlets to be able to map a user identity with stored credentials. The CV is partitioned into vault segments, which are themselves partitioned into vault slots. Slots can be specific to a back-end application (in which case they are shared among many users) or can be specific to a pairing of

back-end system and user. In addition, the CV can contain two types of segments: administrator managed and user managed.

Use the WebSphere Portal CV portlet to create a segment within CV as follows:

1. Log in as an administrator and click Administration.
2. Select the Access option and the CV suboption.
3. Click Add a Vault Segment.
4. Enter a vault segment name and click OK as shown in Figure 23-9.

Credential Vault Slots

CV segments contain one or more CV slots. You use slots to store user credentials and portlets access slots within segments to store and retrieve values. A single slot contains a single value and is used either as a single credential for a specific user or as a value that is shared among all users. Slots can be divided into three types:

- System credentials. Shared among all users and portlets and created in administrator-managed CV segments

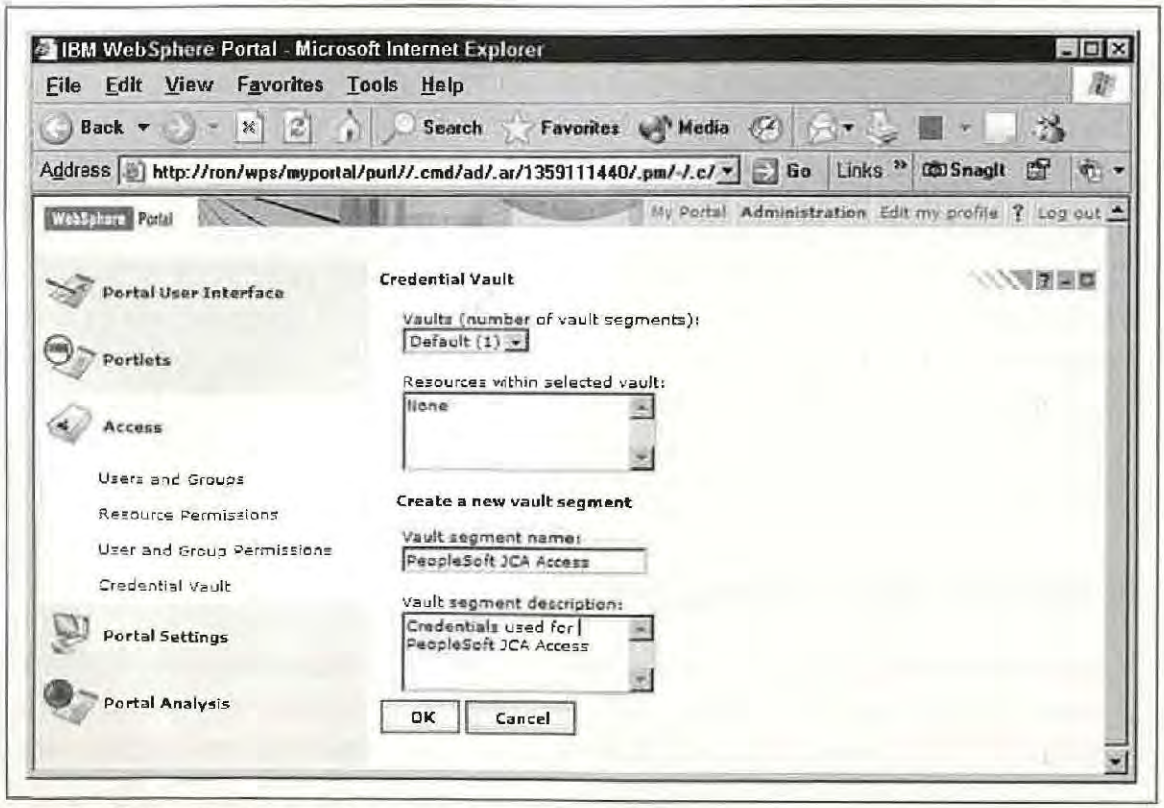


Figure 23-9 Creating a CV segment.

- Shared credentials. Shared among all portlets of a single user; also created in the administrator-managed CV segments
- Private credentials. Private to a combination of user and portlet

Slots are accessed through the CV service as instances of the `CredentialSlotConfig` class. This class stores the metadata associated with the slot as well as a link to the actual resource. Slots can be created programmatically in user-managed segments or using the CV portlet for administrator-managed segments. To create CV slot in an administrator-managed segment, log in as an administrator and follow these steps:

1. Click Administration ⇨ Access ⇨ Credential Vault.
2. Select Add a Vault Slot.
3. Select the vault and the CV segment in which this slot will be created.
4. Enter a unique name for the slot and select a resource to be created (or link the slot to an existing resource).
5. If you are creating a shared slot for all resources, check the Vault slot is shared check box and enter the user ID and password to be used; if there will be a distinct value for each user in WP leave this check box empty.
6. Finally, enter a description and click OK, as shown in Figure 23-10.

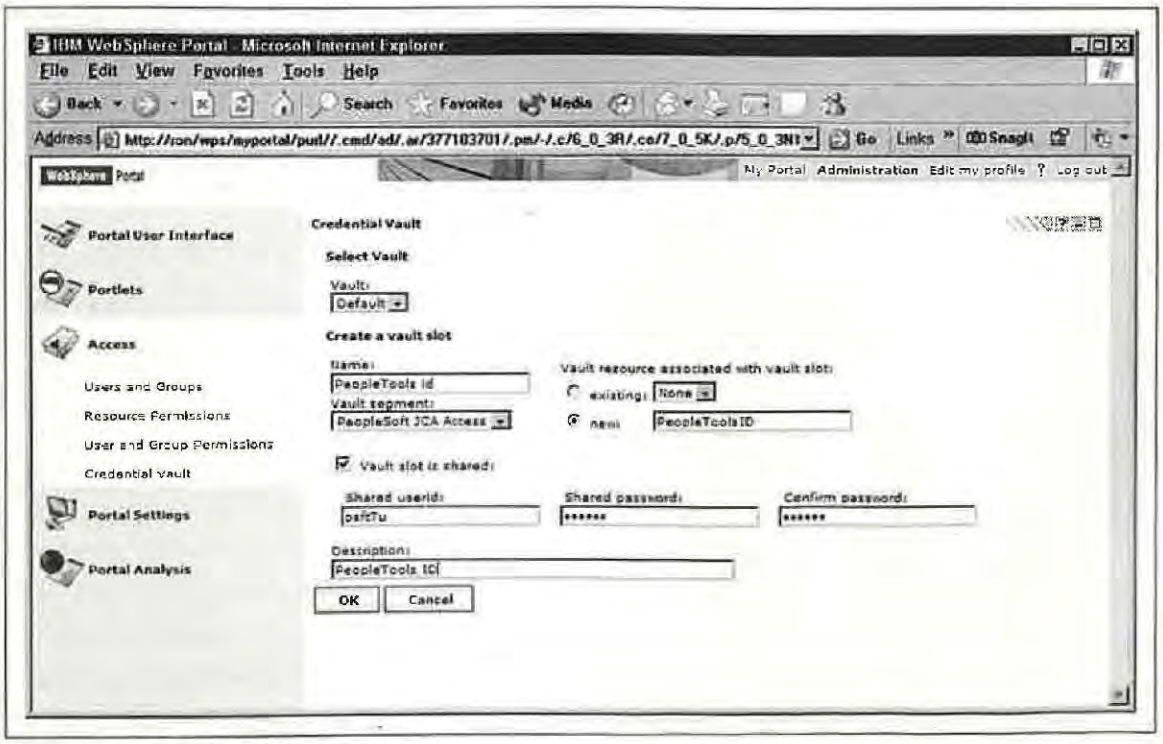


Figure 23-10 Adding a CV slot.

Using the Credential Vault Service API

The CV service provides an API for portlets to access the CV. The API is available from the `com.ibm.wps.portletservice.credentialvault.credentials` package and allows you to programmatically create slots, create or modify credentials, and access credentials stored in a slot.

In order to use the CV service, you will first need to initialize the `CredentialVaultService` object:

```
try {
    vaultService =
          (CredentialVaultService)getPortletConfig().getContext().
                        getService(CredentialVaultService.class);
} catch(Exception e){
     e.printStackTrace(System.out);
}
```

Once you have the vault service object, you can create a new private slot and store a value in it as follows:

```
try {
    PortletData data = portletRequest.getData();
    String slotId = "PeopleTools id";
    String resourceName = "PeopleToolsID";
    ObjectID segmentID = vaultService.getDefaultUserVaultSegmentId();
    Map description = new Hashtable();
    Map keywords = new Hashtable();
    int secretType =
        vaultService.SECRET_TYPE_USERID_STRING_PASSWORD_STRING;
    boolean active = true;
    boolean portletPrivate = false;

    CredentialSlotConfig slot =
        vaultService.createSlot(
           resourceName, segmentID, description,
           keywords, secretType, active, portletPrivate,
portletRequest);

     vaultService.setCredentialSecretUserPassword(
           slotId, userID,
           password.toCharArray(),portletRequest);
   } catch(Exception e){
     e.printStackTrace(System.out);
   }
```

To access credentials from a private slot (using portlet data):

```
Map toReturn = new Hashtable();
try {
    String slotId = (String) portletRequest.getData().
```

```
                        getAttribute("PeopleToos id");
    if(slotId == null) return ;

    UserPasswordPassiveCredential credential =
        (UserPasswordPassiveCredential)vaultService.getCredential
          (slotId, "UserPasswordPassive", new HashMap(),
portletRequest);
    toReturn.put("userid", credential.getUserId());
    toReturn.put("password", String.valueOf(credential.getPassword()));
} catch(Exception e){
    e.printStackTrace(System.out);

}
```

Using the WebSphere Portlet Catalog

The next category of integration is the portlet catalog, allowing you to make use of ready-made packages that provide both the portlet and the back-end integration component. This means that you have no development activities, and very little setup is required—making this pattern the easiest to use. The limitation of this pattern is that it can only provide you with the functionality that was implemented and packaged by the provider and this may not be precisely the functionality you need nor will the user interface always be consistent with your look and feel.

Business application portlets are available from the WebSphere Portlet Catalog at `www-3.ibm.com/services/cwi/portal/_pagr/105/`—this site is maintained by IBM. The catalog lists and describes offerings available from IBM and IBM Business Partners. The catalog enables you to search for information that meets specific criteria such as:

- Portlets and portlet builders
- Tools and other applications, which integrate with WP
- Solutions, such as services and value-added offerings

The catalog is organized based on a set of categories and each catalog entry contains information on the function as well as pointers to additional information about the offering. You can either browse the catalog using content lists such as the one shown in Figure 23-11 or search by category. The following are the available categories listed on the catalog (excluding categories that are based on the company creating the portal, language, markup, or market category):

- Collaboration
  - Calendar

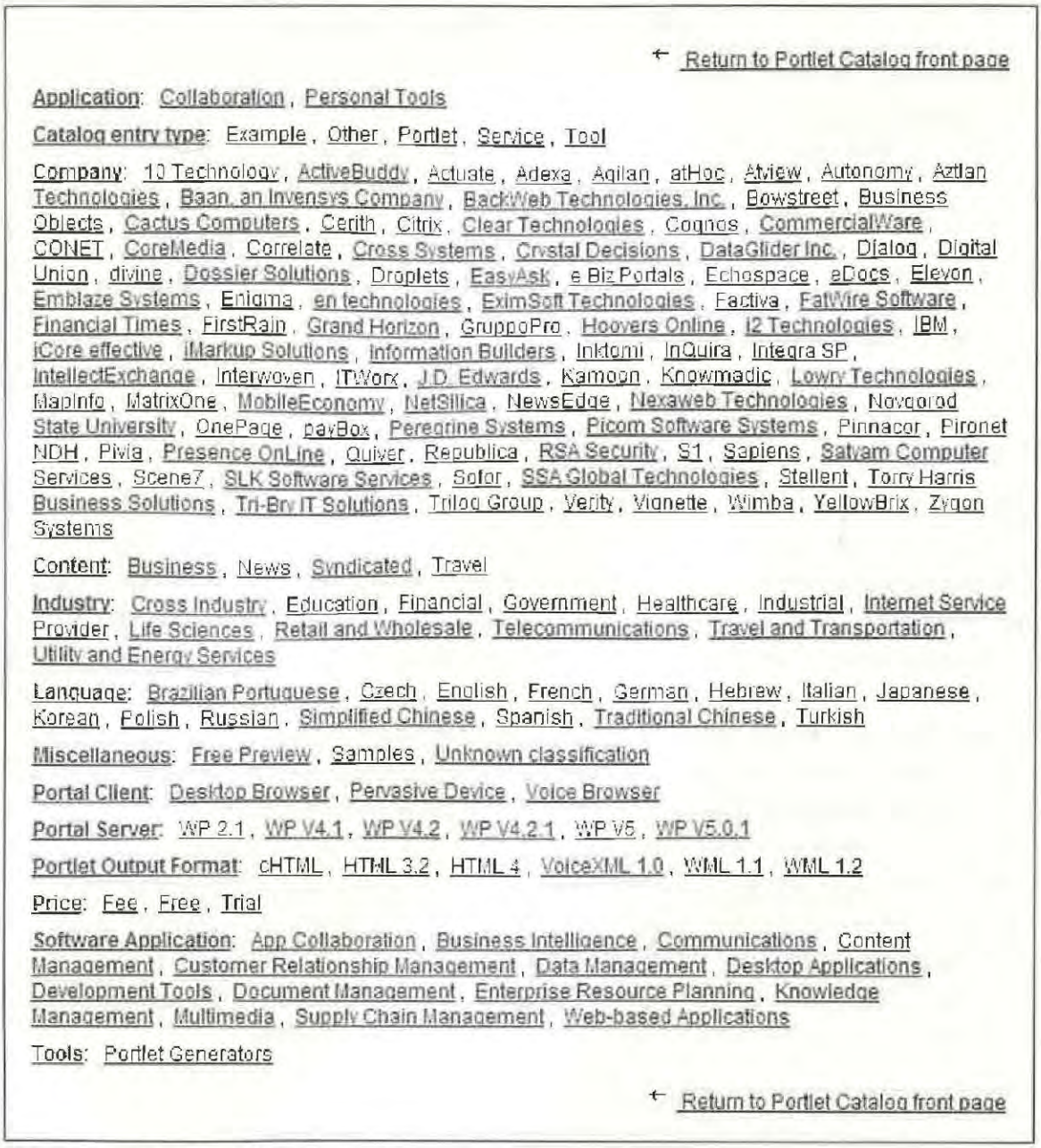


Figure 23-11 Browsing for portlets.

- Contacts
- E-mail
- Messaging
- Tasks
- Work areas

- Personal tools
  - Personal bookmarks
  - Personal calculators
  - Personal calendars

- Personal clocks
- Personal contacts
- Personal search
- Personal tasks

- Content
  - Business
  - News
  - Syndicated
  - Travel

- Miscellaneous
  - Samples
  - Free preview
  - Unknown classification

- Tools—Portlet generators

- Software application
  - Business Intelligence
  - App Collaboration
  - Lotus iNotes
  - Lotus Notes
  - Microsoft Exchange
  - Communications
  - Content Management
  - Customer Relationship Management
  - Siebel
  - Data Management
  - DB2
  - Oracle
  - Desktop Applications
  - Development Tools
  - Document Management
  - Enterprise Resource Planning
  - PeopleSoft
  - SAP
  - Knowledge Management
  - Multimedia
  - Supply Chain Management
  - Web-based Applications

The importance of the portlet catalog should not be underestimated. The more comprehensive the catalog is, the easier it will be for you to set up

meaningful content within your portal. The goal is therefore to have the catalog constantly evolving and have more and more prepackaged portlets offered through the catalog. Because it is supply and demand that drive companies to develop and offer applications, you should make it a point to always look for portlets using the catalog before you opt to develop your own.

While many of the portlets on the catalog are offered by IBM, some of them package integration with business applications. As an example, one of the most popular portlets offered on the catalog is the SAP Enterprise Integration portlets. These portlets were built by IBM using WPAI and support functions in SAP HR, Finance, Material Management, and more. Because the portlets are built using WPAI, they take a very object-centric (rather than process-centric or function-centric) view and support the following views (by category):

- Finance
  - List customer balances
  - List vendor balances
  - Show credit information of customer
  - List open items of debtor
  - List statements of vendor
  - List payment requests
- Human Resource
  - List employee by cost center
  - List and create appraisals
  - Show detail and rate appraisals
  - List qualifications for person
  - List requirements for position
- Sales Orders:
  - Show details of customer
  - List sales orders
  - Check sales order status
- Material Management
  - Material data
  - Material detail
  - Availability of material
- Cost Center
  - List cost center by controlling area

- Cost center general details
- Cost center activities

To install these portlets within your WP instance, you will first have to install the JCO library for SAP. You then need to create a CV slot named `bofactory.slot` associated with a resource named `sap`. You then need to bring up the portlet in edit mode and enter the SAP username, password, hostname, system number, client, language and connections per user pool.

Web Clipping

Referring back to Figure 23-2, we have dealt with the left-hand side of the chart and have covered the patterns termed as back-end integration patterns. All of these patterns have one thing in common: Under the covers portlets use some form of adapter or connector to access the back-end system. In this section we deal with a different kind of integration—integration that is at a user interface level only.

When you use portlets in the context of front-end integration patterns, the portlets do not interact with the application at an API level. Instead, it makes use of Web screen scraping techniques: activating the application through a set of HTTP requests (URLs) and using the responses in HTML (or another markup language) to display information through the portal—either by embedding the pages as is or 'clipping' parts of the HTML and creating a new page from a subset of the original information. Because all Web applications have an identical request/response interaction paradigm, this integration method is very generic.

In Chapter 9 we showed you how Web Clipping helps you build portlets that consume Web content and filter it to present a partial page of relevant portions of an original page. You do this by using the Web Clipping portlet, which helps you build such Web clippers by specifying the URL of the content to retrieve various attributes that affect how the content is clipped. The portlets created by the Web Clipping portlet are called Web clippers or cliplets. Web Clipping supports advanced features such as security and cookies, to allow you to clip not only simple Internet sites but also advanced Web-based applications.

The advanced features of WP cliplets make clipping a useful tool for application integration within WP. Web clipper options include selecting clipping types, modifying firewall options, setting authentication options, modifying rules for URL rewriting, and setting specific security policies. Of

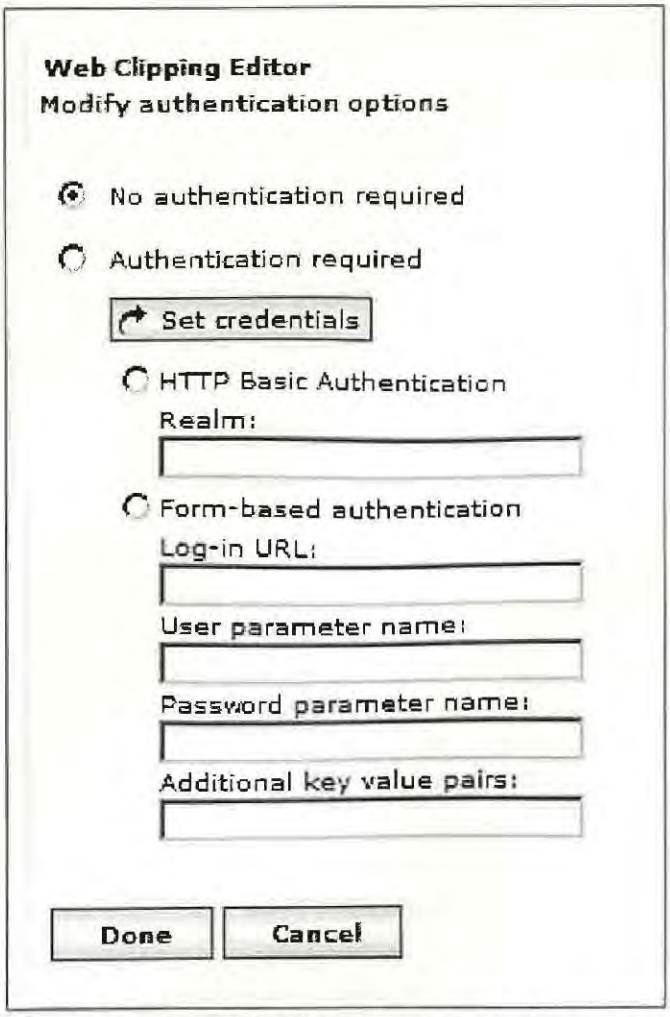


Figure 23-12 Web Clipping authentication options.

these, the ones that you use most often when integrating with a back-end application are the authentication option and the URL rewriting option.

Web Clipping Authentication Options

In using cliplets you will still have to handle issues of single sign-on. Application content is presented by cliplets because WP opens a connection to the Web or application server and makes an HTTP request. The application needs some way to authenticate this request, and you need to set up the authentication method with which such access will be done. When you click Modify Authentication Options for your cliplet, you can select among various authentication options as shown in Figure 23-12.

Select the Authentication Required option and select HTTP Basic Authentication or Form-Based Authentication. If you select Basic Authentication you need only enter the realm name. Form-Based Authentication is dependent on the form used to log into the application and requires you to

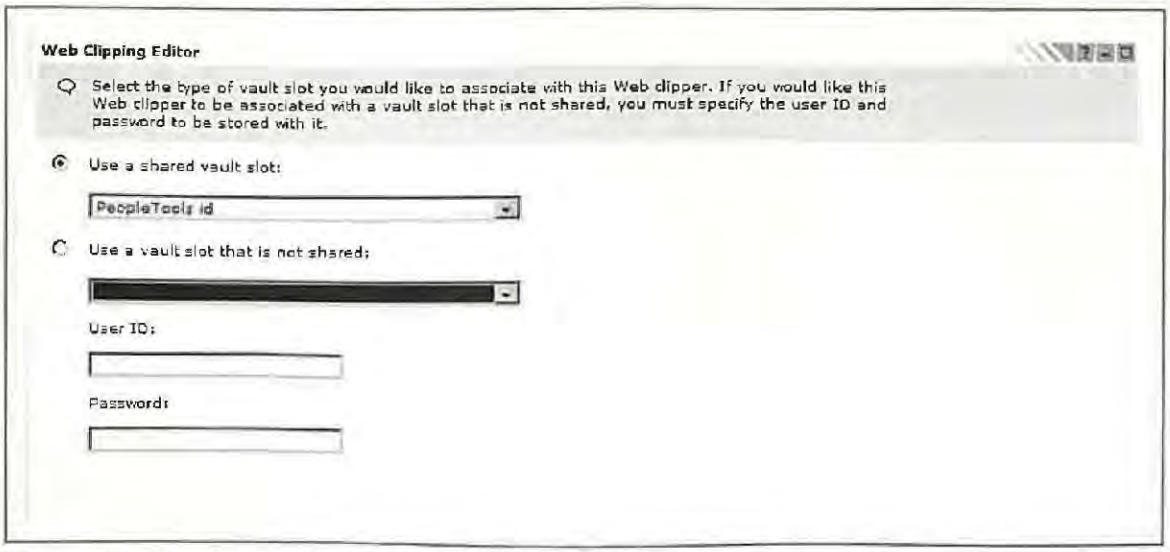


Figure 23-13 Setting the Credentials for Web Clipping authentication.

enter form-specific information—including the URL for the login form and the parameter names within that form where the username and password are entered. You can get these by navigating to the application login form using your browser and viewing the HTML source, looking for the login form and the parameter names. Finally, you need to set the credentials used to perform the login using a CV slot as shown in Figure 23-13. You can use either a shared CV slot or a CV slot that is not shared, in which case you should enter the username and password to be used.

Web Clipping Options for URL Rewriting

You can define the behavior of cliplets in terms of HTML links by using the Web Clipping URL rewriting options. These options control how references are modified when they are part of the cliplet page. Links often need to be modified in order to preserve the portal experience, and the default behavior of URL rewriting involves modifying the links to point at the portal server rather than the original host and ensuring that the page you have navigated to appears within the content area of the portlet.

If you do not want to use the default behavior, you can create a set of rules that affect how URLs are rewritten. When logged in as an administrator, click Administration, then Portlets, and then Web Clipping. Once you have located your cliplet, click Advanced Options and Modify Rules for URL Rewriting. Then check the Use Rules to Exclude URLs from Rewriting Radio button as shown in Figure 23-14. You can then define two sets of URL rules: URLs that will not be modified and, when clicked, will take up the entire browser window. The second set of rules define URLs that will not be

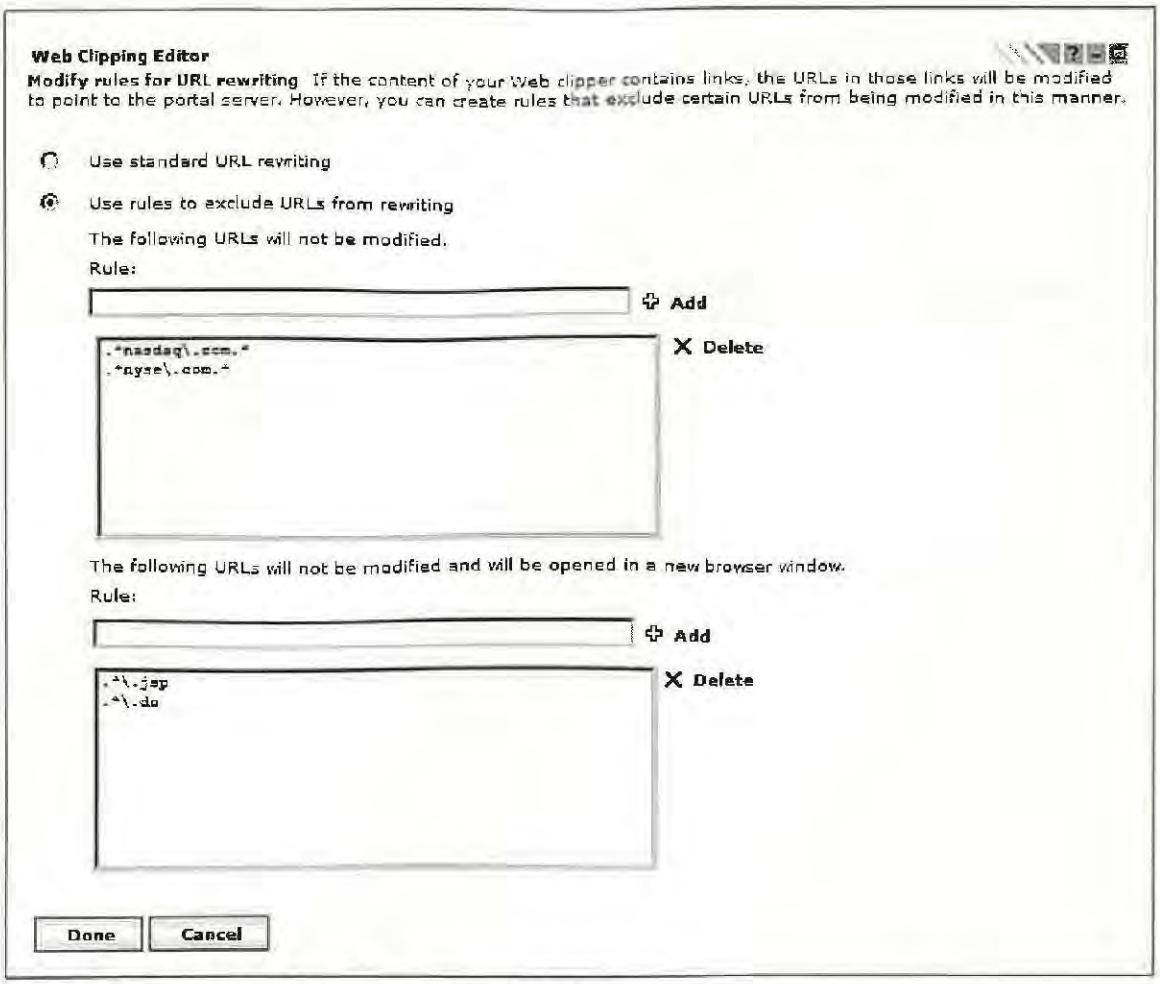


Figure 23-14 Modifying URL rewriting rules.

modified and will be opened in a new window when clicked. In both cases a rule is a Perl expression that defines a pattern. When the cliplet is rendered, the HTML is modified by inspecting each hyperlink and modifying the HREF attribute value based on these rules. If a hyperlink matches a pattern then the rule applies, the URL is not rewritten, and the correct windowing behavior will apply. In Figure 23-14 for example any URL with any host on the `nasdaq.com` or `nyse.com` domain will navigate to take up the browser windows, and any page ending with `.jsp` or `.do` will be opened in a new window.

Application-Specific Web Embedding

Application-specific Web embedding is another type of integration pattern that makes use of HTTP/HTML screen scraping and allows you to deploy existing Web screens directly on your portal without integrating to the

back-end system at an API level. What is special about application-specific Web embedding is that because the portlets that perform the integration task know that they only need to interact with a certain application they can make all sorts of assumptions on the application architecture and the way it behaves. Because these portlets do not need to be completely generic, they can do a better job in integrating with the specific application suite involved and they can contain parameters that are specific to the application suite, allowing for better integration with the application.

Two examples of application-specific Web embedding involve the SAPGUI for HTML portlet and the SAP Business Warehouse (BW) integration portlets, both available through the portlet catalog. The SAPGUI for HTML portlet allows you to embed any HTML page created by SAP's Internet Transaction Server (ITS) either as an iframe or as a pop-up window. The SAP BW portlet allows you to embed views from existing SAP BW systems and expose them within your portlet—including functions from opportunity management, personnel administration, and controlling activity-based costing. In this section we focus on SAPGUI for HTML because it is a good example of application-specific Web embedding.

The SAPGUI for HTML portlet is built to use SAP's ITS architecture. As Figure 23-15 shows, a mySAP.com system includes the SAP R/3 application and the SAP workplace middleware. Within SAP workplace, the ITS component serves to create and deliver HTML pages to the browser and serves as the broker between an R/3 system and the Internet. The ITS is placed between the Web server and the R/3 system and comprises two main components: the WGate and the AGate. The WGate resides on the Web server and is the plugin connecting to the application server—the AGate. The AGate controls the communication with the R/3 system and is responsible for session management, mapping R/3 screens or function modules to HTML, connection management, and most importantly the generation of HTML pages.

The SAPGUI for HTML portlet only supports Internet Explorer and has been tested with SAP R/2 version 4.6C. Setting up the portlet merely involves downloading it from the portlet catalog and adding it to your WP environment. You then need to set the following parameters:

- SAPGUIBrowserHost. Host name of the SAP ITS Server. If you do not use port 80 for the connection, you must also specify the port number.
- SAPGUIBrowserPathExtension. URL path where the application is provided by the SAP ITS Server.
- SAPGUIBrowserClient. Client number for the SAP system.
- APGUIBrowserLang. Language settings for SAP ITS server.

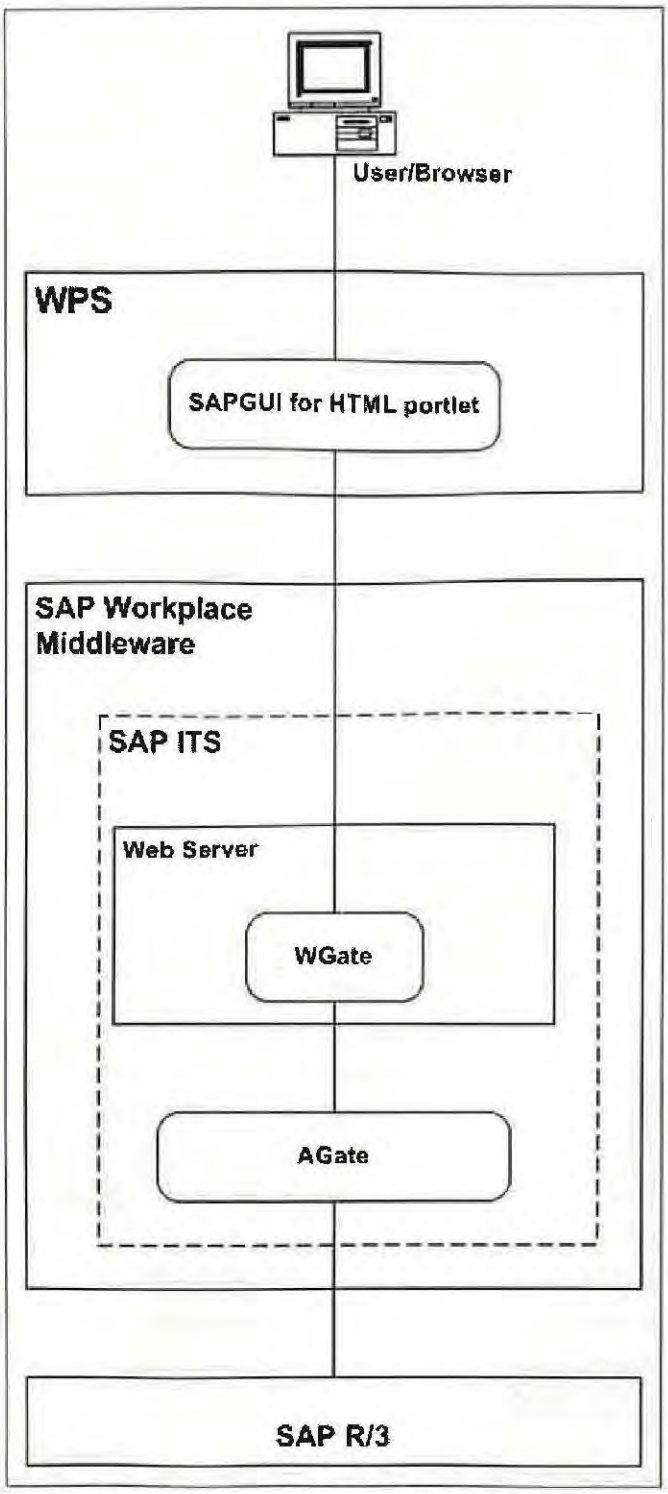


Figure 23-15 Using the SAPGUI for HTML portlet to connect to a SAP ITS system.

- SAPGUIBrowserTxCode. Transaction code for SAP ITS Server.
- SAPGUIBrowserIFrameWidth. Width of the iframe where the HTML page will be shown.
- SAPGUIBrowserIFrameHeight. Height of the iframe where the HTML page will be shown.

- SAPGUIBrowserDetach. Set to false if you want to view the SAP ITS server's content within a portlet iframe or set to true if you want to pop up a separate window.

Embedding Functional Portals

Functional portals are portals that focus on a certain business domain, set of functions used within the organization, or both. These portals are very common in many organizations and can be managed by a certain business function (for example, an HR portal or a finance portal) or by another organizational entity (for example, a business unit or a geographical unit within the organization). Functional portals can often be deployed quickly, sometimes due to the fact that they are limited in terms of the functions that need to be implemented and sometimes because a single organization is often more effective in making decisions quickly and moving the project further. In addition, many application suite vendors provide a portal that is already integrated with the back-end business function with the intent of making a functional portal very quick to deploy. Examples include SAP portals and the PeopleSoft portal.

Functional portals are popular because they allow an organization to put up their portal quickly. Unfortunately, functional portals often work against the main goal of having an enterprise portal. Referring back to Figure 23-1, many functional portals can create a reality that is not too dissimilar from the left-hand side of that figure. A common enterprise portal (the right-hand side of Figure 23-1) is where most enterprises want to get to.

The last integration pattern addresses a scenario in which an enterprise wants to integrate a number of already existing functional portals. In such a case it is often not desirable to build additional integration directly to the back-end application (that is, circumventing the functional portal). Instead, you can expose the functional portal and the functions deployed on it directly from within your WebSphere Portal.

Embedding functional portals is relatively simple when it comes to integration—and we only mention this pattern here for completeness. The functional portal is accessed by pointing directly to starting-point URLs that reside on the functional portal. In addition, there are two main issues you need to deal with in order to use this integration pattern. The first and most important issue is single sign-on. By this we don't mean using the CV—we mean SSO as presented and described in Chapter 21. In fact, the SSO tool used is really the tool performing the integration. The second issue you will need to deal with involves consistency of the user interface. Because each functional portal may have its own set of styles, skins, and themes (not to

mention navigation methods and operational standards), you may need to do quite a bit of look-and-feel standardization.

Summary

In this chapter we walked you through a number of integration patterns. We showed you the abundance of possibilities that are at your disposal and explained what each one entails. We showed you both front-end integration patterns and back-end integration patterns. Front-end integration patterns make use of the fact that the vast majority of applications you will want to integrate into the portal are Web enabled and have an Internet architecture. In these cases you can expose these screens directly within your portal or use sections from the HTML pages and create subsets of information to be exposed through the portal. This allows you to make use of the existing application suites with very little development and with less-than-perfect understanding of the underlying data model and functional semantics forming the core of the application. This type of integration is also useful when you already have a functional portal (for example, HR portal with HR self-service) and are building a larger enterprise portal that needs to provide functional elements as well. In this case you can get integration done very quickly with very little effort. The disadvantages are that you are limited to the functions already exposed through the functional portal and/or the Web-enabled applications, which you have to deal with having a possibly inconsistent user interface look and feel, that you need to handle single sign-on with the external application/portal outside of WP (refer back to Chapter 21 for more on single sign-on) and that you cannot have custom navigation among the existing functions. Even with all these disadvantages, it has been our experience that front-end integration is very effective, relatively easy to perform, and provides good return on investment.

The other category of integration patterns we covered was back-end integration. In these patterns the application suite is accessed through an adapter or connector that retrieves data from the back-end application and uses APIs to perform transactions. Portlets make use of classes that use the adapter to perform all this. Back-end integration provides you with much more control because most application suites have very comprehensive API sets and because (at least in some of these patterns) you are developing your own portlets and can control the user interface and the navigation.

Back-end integration tends to be more complex than front-end integration and tends to require more expertise. On the other hand, it is more stable—APIs and adapters change much less often than user interface and if you use some kind of screen-scraping approach you may be surprised when

the back-end application team upgrades their software. Back-end integration also usually means less operational cost because if all users access the application through the portal then you only need to have an API server running and do not require a full Web server environment for the application. Finally, many of the front-end integration portlets have only been tested for specific application versions and are sometimes no more than a proof-of-concept type of solution—be careful to check all the "small print" before you opt to bypass the more comprehensive back-end integration.

In either case, prepackaged solutions mean much less work with a disadvantage of providing you with less control. While back-end integration usually involves more work than front-end integration, prepackaged portlets that may be available on the portlet catalog require by far the least amount of work. Therefore, regardless of whether you choose a back-end or a front-end approach, our suggestion is that you try to use an existing integration solution before you delve into development of new portlets and code and stick with it if you can get most of the functionality you need.

Now that you know about the various methods that are available for integrating applications that may be used by the large user base using your portal, let's turn to the subject of how to support ALL users—regardless of where they access the portal from and what device they use. In the next chapter we address the topic of mobile portals supporting users on-the-go and users using devices other than conventional PCs.

CHAPTER

24

Supporting Mobile Users

In this chapter you will learn how to use WebSphere Portal to support mobile users. You will see how WP (and products that extend WP) can be used to deploy portlets that provide access using various browsers and microbrowsers commonly used on devices such as Personal Digital Assistants (PDAs) and mobile phones. You will learn how to develop portlets that generate content for such devices and how to deploy Transcoding Technology (TT) that can use existing portlets originally targeting standard browsers to be used from these more limited platforms. You will also learn how to address issues that come up due to the fact that mobile users do not always have a continuous connection back to the server and sometimes need to be able to continue using some portlets in an offline environment.

WebSphere's mobile strategy is centered on the WebSphere Everyplace product family. Parts of this chapter discuss elements of this product family, and predominantly WebSphere Everyplace Access (WEA). WEA is actually a product that includes WP as an embedded component and from a server-centric viewpoint is an extension of WP. We will introduce you to the features directly implemented in WP for allowing you to serve mobile users as well as show you what WEA has to offer as an extension to WP. We will also introduce you to TT allowing you to write portlets that generate content once and have external rules that generate multiple markup types (suited to multiple browsers and devices) from this single portlet presentation. This eliminates the need for multiple portlets that implement the same function but serve different device types. Both WP as well as WEA includes elements of TT and we will review the most important features in TT that can help you deploy multidevice support quickly.

Mobile Users

Most people think of portals in terms of Web pages that are accessed by workstations or laptops used in an office environment and/or from the home. Portals help organize the various applications and data that can be accessed and provide a way to manage the Web desktop.

The reason most people have this large-computer-centric view of portals is that portals, as any computing technology, were initially used primarily by users with a high degree of sophistication when it comes to information technology. However, two trends are making this PC and workstation viewpoint incomplete.

More and more portals serve very large communities—often serving communities that are not necessarily information technology professionals nor have a strong affinity to computers. The second is that today there are a very large number of devices that can access the Internet that are not computers in the classic sense (all these devices have some kind of processor and are technically computers too). These devices include Internet-enabled phones, PDAs, and handhelds. In fact, there are already more such devices being used today than there are PCs and the number of such devices is constantly growing—both in the consumer world and in the business world. This is especially relevant for portal developers. As an example, when large companies decide to build and deploy a large employee portal that will cover all Human Resource functions, the effect can only be achieved through very high utilization levels by employees. This sometimes means that multiple access channels need to be provided to the functionality deployed on the portal.

The direct result of these two trends is the fact that developers of portals and portlets need to address a wide variety of devices that can be used to access content and applications. Different devices have different form factors and different data-entry interfaces. A portlet that creates content that looks good on a laptop may look awful on a mobile phone—in fact, it will probably be unusable. In addition, different devices have data-entry mechanisms. Phones should be used in a way that users are asked to select among various options but should not be forced to key in large amounts of information. Another example includes Symbol Technologies' PDAs that have a barcode reader—in which case data can be entered without any keying in or graffiti-type typing (see Figure 24-1). Symbol has two such product lines—one based on the Palm OS and one on the Pocket PC operating system by Microsoft—both having a barcode scanner. If you have a job that includes delivering packages, you can simply scan a barcode on a label that is attached to the package rather than key in the number, thereby cutting the delivery time as well as errors that will undoubtedly occur if you have to type in the package number.

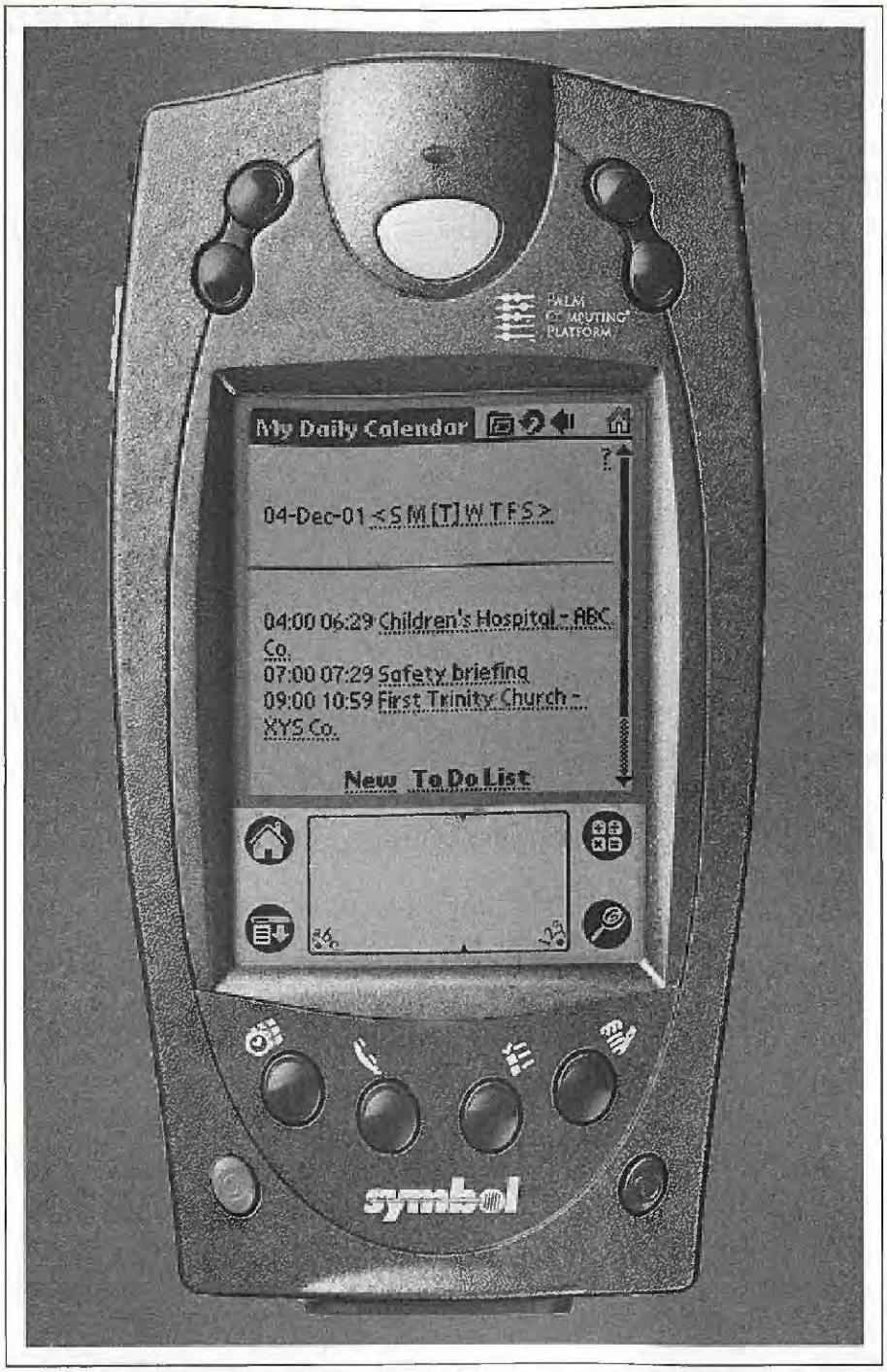


Figure 24-1 Palm OS based Symbol PDA with bar code scanner.

Addressing many devices means carefully thinking not only about the user interface, but also about some technical challenges including the fact that different platforms have different browsers and microbrowsers supporting different languages. For example, both Internet Explorer (IE) on a laptop and Pocket IE on a PDA support HTML but most microbrowsers on mobile phones support the wireless markup language (WML) or an older (and quickly fading) markup language called HDML. In Japan, mobile

phones (called i-mode phones) support yet another language called cHTML (compact HTML).

Support for different languages, screen sizes, and input mechanisms are not all you need to worry about. People who use workstations and PCs are normally in an office environment and connected through a high-speed network to the Internet. People who are in the field (such as service technicians) will often need to have access to the portal using a wireless environment—and we don't just mean hot spots in Starbucks shops. Such workers need to have access over true wireless networks that are usually provided by large cellular companies or through private radio networks. These networks tend to be slower and in many parts of the world (for example, the United States) provide spotty service. In order for these workers to be effective, they need to be able to continue to use their data and applications even when the network is down or when coverage is nonexistent (regardless of whether the reason is that there are simply no cell towers in the area or work needs to be performed in a basement of a large building).

To recap, the issues that you should keep in mind when building portals and portlets serving large and mobile communities (and that we will address in this chapter) are as follows:

- Support for multiple presentations for different devices
- Support for both online and offline access
- Support for multiple access networks

Supporting Multiple Markups

Most devices that could be used to access portal functionality have some kind of browser. Such browsers are based on a markup language, examples being HTML, WML, and cHTML. The first thing you need to ensure, therefore, is that your portlets have the right setup for supporting multiple markups.

If you are working within WebSphere Studio Site Developer (WSSD) and have the portlet toolkit installed, you can specify the additional markup support when creating a new portlet application project. Specifically, when you use the wizard for creating such a project click the Next buttons rather than the Finish button; the last wizard panel will allow you to specify additional markup types as shown in Figure 24-2.

Alternatively, if you already have a portlet application project and you want to add additional markup support, double-click the `portlet.xml` file. This will open the portlet configuration editor. The following steps allow you to add or remove markup supported by your portlet:

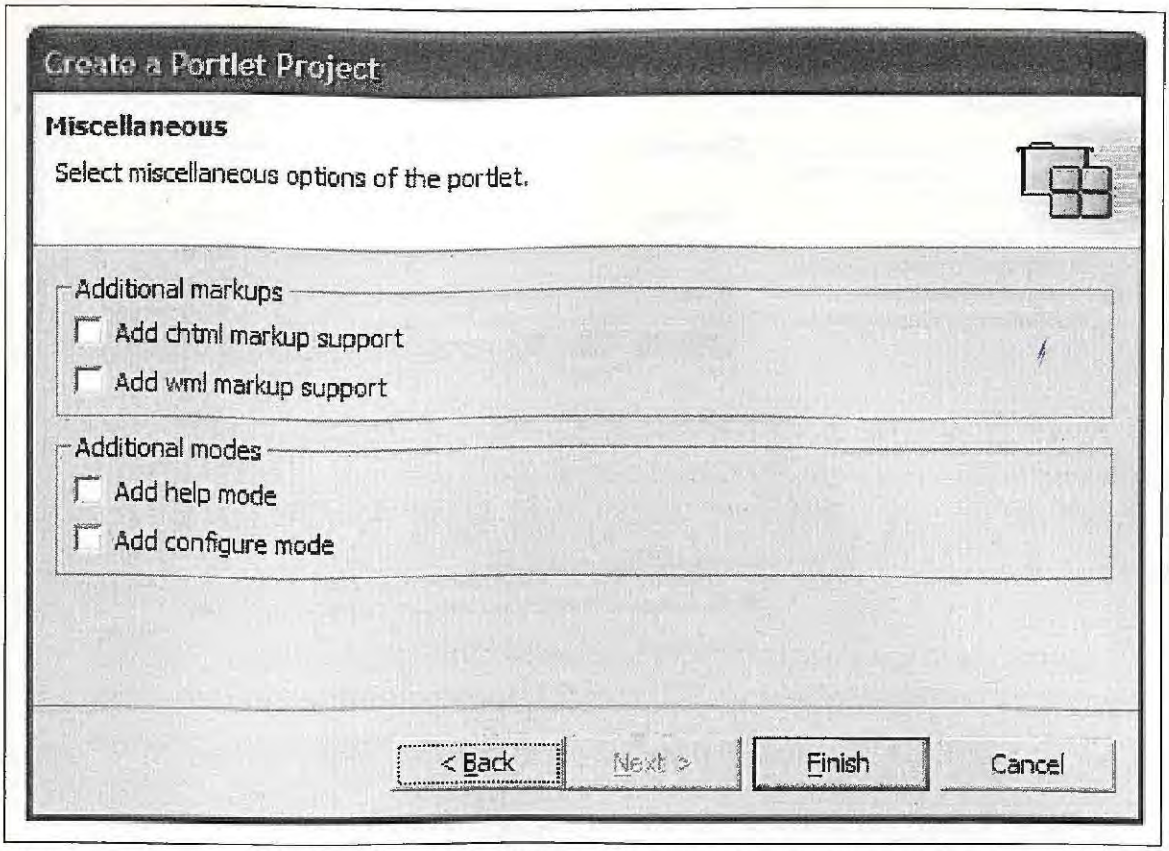


Figure 24-2 Specifying additional markup within the WSSD portlet application project wizard.

1. Select your portlet from within the PortletApplication selection.
2. Scroll down—the fourth section on the right-hand side allows you to add additional markup support. By default you only have HTML support.
3. Click Add and select the markup type as shown in Figure 24-3. Click OK.

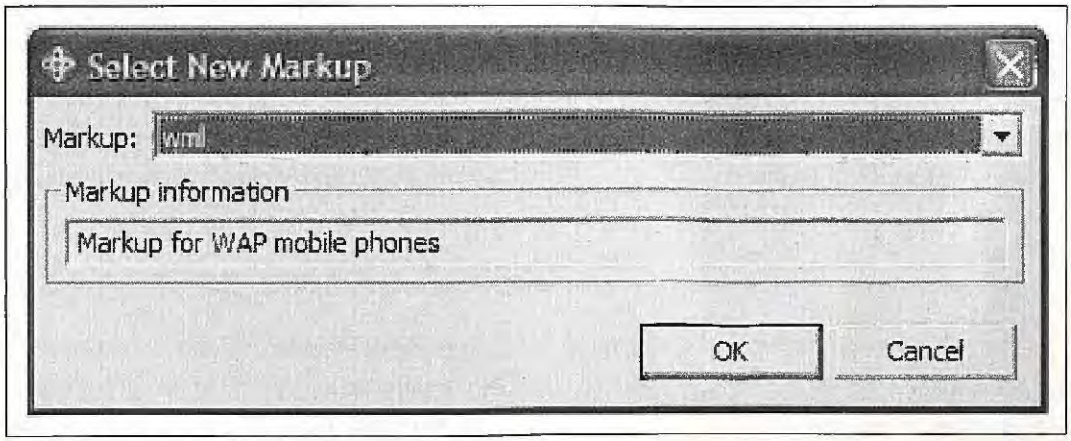


Figure 24-3 Adding markup within WSSD.

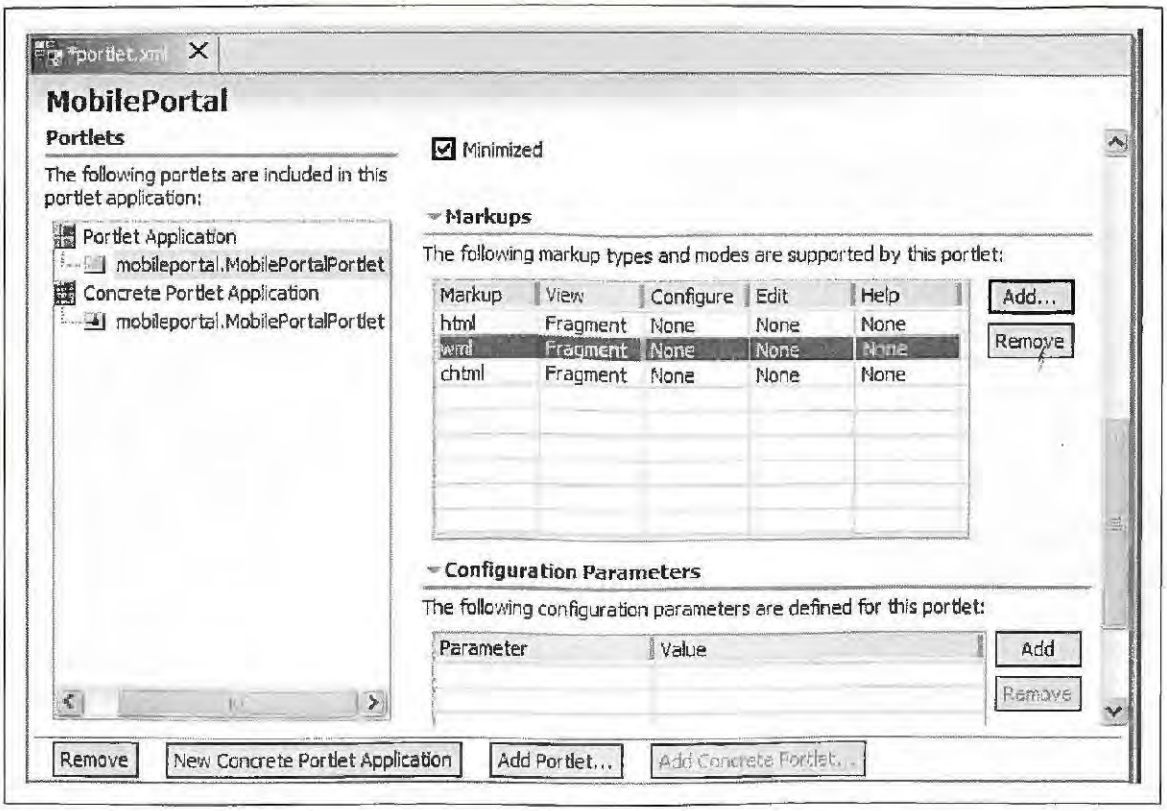


Figure 24-4 Multiple markup support for a portlet application project.

Once you have finished adding the various markup types (for example, as shown in Figure 24-4), save the portlet configuration file.

And finally, if you prefer to edit the portlet.xml file directly, markup is specified within the `supports` tag shown below:

```
<portlet-app
uid="mobileportal.MobilePortalPortlet.50a1bb94df0e00181166c78cbc0d3e57"
major-version="1" minor-version="0">
  <portlet-app-name>MobilePortal application</portlet-app-name>
  <portlet id="mobileportal.MobilePortalPortlet" href="WEB-
INF/web.xml#mobileportal.MobilePortalPortlet" major-version="1" minor-
version="0">
    <portlet-name>MobilePortal portlet</portlet-name>
    <cache>
      <expires>0</expires>
      <shared>no</shared>
    </cache>
    <allows>
      <maximized/>
      <minimized/>
    </allows>
    <supports>
      <markup name="html">
        <view />
```

```
        </markup>
        <markup name="wml">
          <view />
        </markup>
        <markup name="chtml">
          <view />
        </markup>
      </supports>
    </portlet>
</portlet-app>
```

HTML, WML, and cHTML are all supported by default with every installation of WP. There are additional markup types that are supported only through the use of WEA, and within WSSD only when you install the Everyplace Toolkit. In this case additional markups that are provided include the following:

- PDA—a subset of HTML useful for PDA browsing. This markup is also used for offline access with the WEA client.
- VXML—VoiceXML is a markup language that is used to develop and deploy voice-enabled portlets. VXML is supported by many commercial voice servers, and by using VXML markup you can deploy functionality that can be used by making a voice phone call, selecting menu options, and making data-entry commands—all using a normal voice call and pressing your phone keys.

WebSphere Everyplace Access

WEA is a part of the WebSphere Everyplace product line and allows you to deliver content to mobile devices. Interestingly enough, WEA includes an embedded instance of WP. In fact, the main goal of WEA is to extend portal functionality to mobile devices. WEA includes a set of tools as well as portlets for mobile use. From a tool perspective, it includes enhanced transcoding capabilities, synchronization functionality with Microsoft Exchange and Lotus Notes, and support for offline access. From a packaged application perspective, it includes a set of productivity and personal information management portlets.

Supporting Offline Access

In many companies with mobile workforce, there are situations where users have less-than-perfect coverage. If you have a cell phone you must be used to the problems of incomplete coverage. Even if you have good coverage, wireless connections are often considerably slower than landline

connections and conventional Internet connections. All this means that supporting a good experience for mobile users often means synchronizing content onto a mobile device, ensuring that the mobile unit is fully functional although it is not connected, and allowing the user to upload actions that were performed on the mobile unit and resynchronize content at a later point in time.

WEA and the WEA client (which needs to be installed on the mobile device) allow you to view, browse, and interact with portal content offline. One of the pages that is created for you by WEA is the offline page. All you need to do per portlet that you want to use offline is make sure that it supports the pda markup and add it to this offline page (incidentally—this means that only HTML-generating portlets can work in an offline mode). The WEA client will then synchronize with WEA and access a special servlet called WebCache that traverses links on the offline page and rewrites the links and pages referenced by the offline page. The rewriting process is done based on a depth parameter that can be configured within WEA and controls how many links will be traversed. As an example, if you have a Web page structure as shown in Figure 24-5 and provide a depth of 4, then the highlighted pages will be offloaded to the mobile unit. To specify the link depth, log in to WEA as an administrator and modify the depth parameter of the offline browsing administration portlet. The default link depth is defined on the WEA Home ⇨ Administration page. A user can also define a user-specific link depth value by navigating to WEA Home ⇨ Configure.

Once you've defined the depth level, you use the WEA client to view the offline page. You need to start the client, configure the access to the WEA server, and synchronize the offline content. As an example, if you are using a PocketPC unit follow these steps:

1. Select Start ⇨ Everyplace Client.
2. Enter your portal name and password.
3. Select My Settings from the drop-down list.
4. Select Network Profiles.
5. Enter the fully qualified name to your WEA server.
6. Select Home from the drop-down list and click the Synchronize icon. This will get the offline portlet content onto your unit.

The offline page supports a perception that you have direct access to the portal at all times. You can navigate between pages and view data. The only difference between offline operation and online access is that when you press Submit in a form, the WEA client will inform you that the submission has been deferred until you resynchronize with the main server. Specifically, you will get the following message:

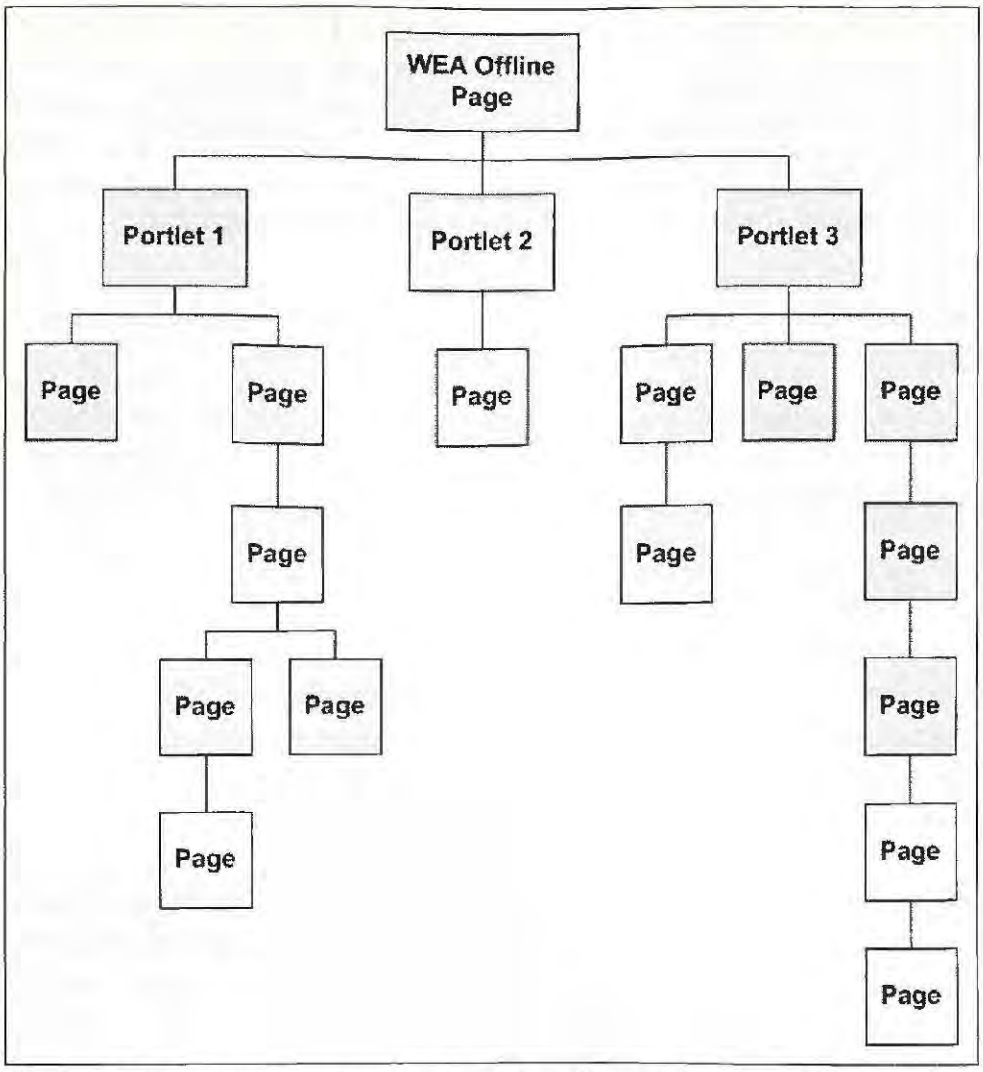


Figure 24-5 Offline synchronization of portlets with a depth of 4 for traversal of links.

Your request is deferred

Due to no active connection now, your request is stored in the database successfully and will be posted by invoking a sync application later.

In addition to the depth parameter, there is one additional WEA configuration step; you need to turn on URI addressability by changing the value of use.requestId to false in

```
<WAS HOME>/lib/app/config/services/ConfigService.properties.
```

Writing portlets that can work offline is nontrivial and requires a different discipline to that you may be used to. When building portlets that will be used in an offline mode you should adhere to the following rules:

1. Always support pda markup.
2. Have only one form per page (this is true for both portlet pages and pages that can be reached from the portlets).

3. Use only POST in your forms; do not use GET.
4. Write code that maintains session in a way that is resilient to duplicate submissions of the form. Synchronization does not guarantee that submissions will occur only once.
5. You can use simple Javascript validations but avoid complex Javascript code.
6. Do not include links within Javascript and do not rely on Javascript navigation.
7. Make sure your pages are well formed. You should verify that all tags are closed correctly as per the XML well-formed-ness rules. You can use tools such as **jtidy** available at `http://lempinen.net/sami/jtidy`.
8. Do not use PortletActions.
9. Avoid dynamic behavior on screen that affects submission parameters (for example, pages that dynamically add to HTML tables for inclusion in a collection; in this case put all options in the table with check boxes used for defining which entries should be added to the collection. Always avoid cascading forms).

Before completing this topic we would like to point out that there are additional packages other than WEA that provide offline portlet capabilities. Probably the best known is AvantGo by Sybase, which can also be used to provide offline portal capabilities. In this case you will be working directly with WP but will need to install an AvantGo server that will be doing the synchronization and caching work.

Supporting Multiple Wireless Environments

When building portals that will be used by mobile users not only do you need to worry about devices, markup, and offline capabilities, but you also address connectivity issues. Wireless networks can be highly complex and diverse and this is a topic that we cannot fully address in this book. Our goal is merely to point out that this is an infrastructure issue you should plan for.

The Internet is an IP-based network and is ubiquitously available from homes and offices throughout the world. This means you do not have to deal with connectivity issues when deploying your portlets and you do not need to think about how users will get access to your portal. For mobile users this is not necessarily true. Different geographies have different networks, some areas have multiple connection types, and others have none. In addition, wireless networks are not always IP-based and some involve highly

proprietary radio frequency protocols. The connectivity options include Hot Spots and 802.11b networks, CDPD, GPRS, Mobitex, GSM, CDMA, TDMA, satellite networks, iDEN, TETRA, EDACS, and so on. Each of these networks potentially has its own complexities and peculiarities. Without getting into too much detail, you should distinguish between the following types of wireless networks.

1. IP-based networks that provide you with an "always on" connection. This is the simplest case for you to support because in this case you don't have to do anything special—each mobile unit is perceived to be a node on the Internet and can access your portal in a normal way.
2. Dial-up networks. In these networks you use a phone or another device that acts as a modem. Once setting up a connection through another protocol (for example, PPP or over GSM) you can proceed as though you were connected to the Internet. In this case you will need to use a connection manager that is suited to the network and the devices you are using.
3. All other options have proprietary protocol suites. In these options you will need to either develop a set of adapters using SDKs of the network providers or use a wireless gateway. A wireless gateway abstracts the differences in the networks and provides you with a unified way to access the server regardless of the networks your users will be using. Different wireless gateways have different abstractions. For example, the WebSphere Wireless Gateway (another component within WebSphere Everyplace) abstracts all the networks to an IP network resolving all the complexities. A different example is the Broadbeam Wireless Gateway that creates a queue-based abstraction that allows your client-side applications to place messages that need to be delivered to the portal and have messages come back. The Broadbeam software then takes care of delivering these messages regardless of the underlying networks.

While adding an additional server to the mix certainly complicates your environment (see Figure 24-6), we strongly suggest that if you need to deploy your portlets over non-IP networks you use a gateway architecture. The alternative integration to multiple-carrier SDKs are far more complex.

Transcoding Technology

TT is an enabling technology that allows you to easily deploy portlets that can support different devices. TT implements three methods that transform content based on device information associated with the request.

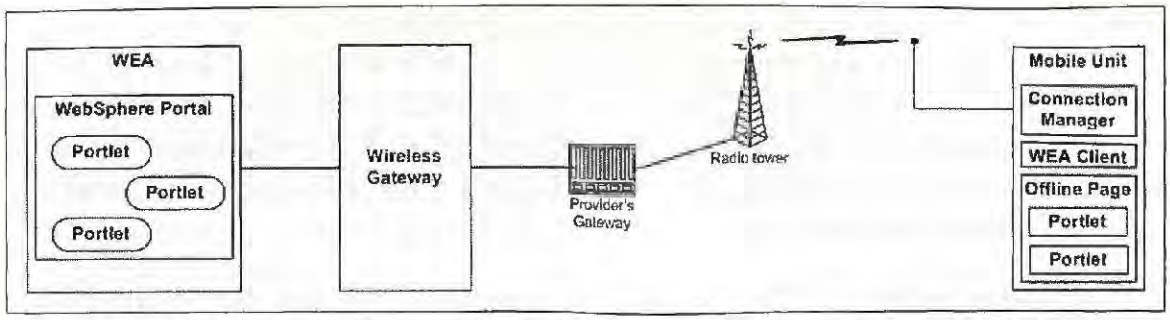


Figure 24-6 High-level architecture view including a wireless gateway and WEA.

TT inspects user agent information that is part of the headers packaged with the request and allows you to provide specific instructions for transforming the original content to a form that is suitable for the device that initiated the request. Elements of TT are available in WP but the full package is only available in WEA and most of the methods detailed in this section relate to the more advanced TT available in WEA.

In order to enable TT in WEA, log in to the portal as an administrator and follow these steps:

1. Select the Portal Administration tab.
2. Select Global Settings.
3. Check the `Enable trancoding of portlet content` check box.
4. Save and exit.

TT offers three ways to transform content, which are as follows:

1. XML style sheets—this method uses style sheets expressed in XSL to define mappings that should be applied before the content is delivered to the user's device.
2. Annotators—this method allows you to associate a set of annotation instructions with generated content. This instruction set allows you to specify (either within the generated content or using an external set of annotators) how the original portlet content should be modified before it is delivered to the user's device.
3. Transcoding plug-ins—a set of generic content modifiers that address common tasks such as conversion from normal HTML to WML.

In addition to the transcoding plug-ins TT has a built-in fragmentation transcoder. Fragmentation is important because many mobile phones have severe limits on storage capacity. As an example, some i-mode phones and gateways have a limit of 2048 bytes per document and some WML phones and gateways have a limit of 2880 bytes. Large documents must be broken

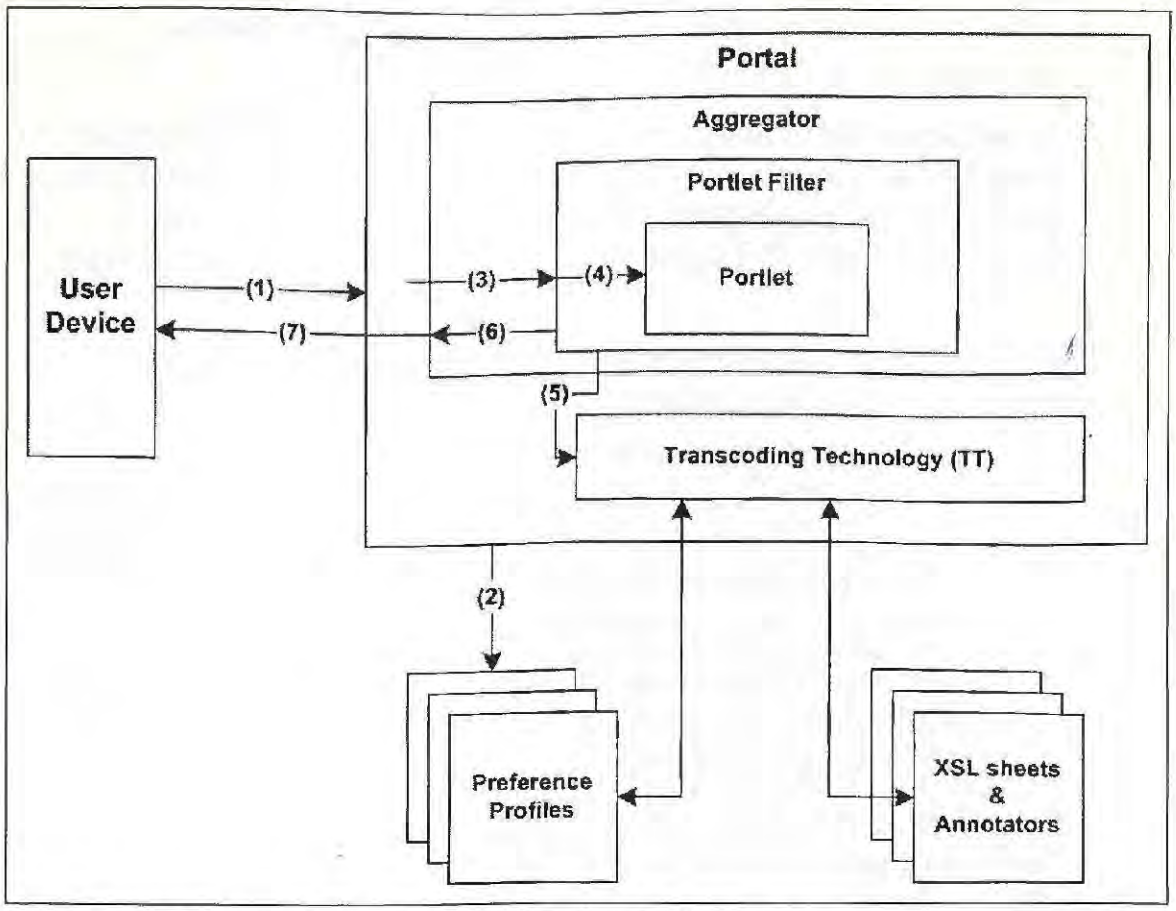


Figure 24-7 Request flow using TT as handled by the portal.

down into multiple documents linked together before they can be delivered to the user device. This function is carried out by the fragmentation transcoder.

Figure 24-7 shows the request handling flow that makes use of TT to adapt generated content. The flow shown in Figure 24-7 includes the following steps:

1. When a request comes in, the portal determines what the user agent is. This value captures which device the user is using.
2. The portal matches the user agent with the preference profiles installed. Preference profiles allow you to specify how to treat documents that will be delivered to different end user devices. You should place your preference files as `.prop` property files in the `<TT_ROOT>/etc/preferences/device` directory and name the files based on the user agent to which they correspond. Selection of the appropriate preference profile is based on the following sequence:
   a. The HTTP header is inspected to see whether it includes a value for `X-IBM-PVC-Device-Type`.

b. The `User-agent` field is checked in the HTTP header.

c. If neither exists, TT uses the default device profile.

d. In addition, you can specify a custom field in the HTTP header that will be used for selecting a user-specific profile (rather than a profile that is based on the device only). To do this, set the `httpUserIdField` value in `<TT_ROOT>/etc/localConfig.prop`.

Commonly used parameters that you can control through the preference profile files include the following:

a. `deviceRule`—defines which user agent this profile will be matched up with. You should include wild cards to cover multiple browsers and microbrowsers rather than have a specific preference profile for every different browser since many browsers and microbrowsers behave similarly.

b. `javaAppletsSupported`—a boolean value specifying whether applets should be stripped out of the content for devices that cannot render Java applets.

c. `javaScriptSupported`—a boolean value specifying whether Javascript code should be stripped out of the content for devices that cannot interpret and run Javascript.

d. `framesSupported`—a boolean values specifying whether the device can support HTML frames.

e. `desiredContentTypes`—a MIME type specifying the content type sent back to the user device.

f. A value for the maximal deck size as used by the fragmentation transcoder. This value defines the maximal size per fragment. As an example, the parameter for WML fragmentation is called `WMLMaximumDeckSize`.

3. The portal calls the aggregator that in turn calls the appropriate portlets. Each such portlet has a portlet filter that is invoked by the aggregator. An appropriate portlet is one that is configured to use TT. In order to configure a TT-enabled portlet you need to set two configuration parameters within your `portlet.xml` file as follows:

```
...
<portlet-app-def>
  ...
  <concrete-portlet-app ... >
    ...
    <concrete-portlet ... >
      ...
```

```
        <config-param>
          <param-name>DoTranscoding</param-name>
          <param-value>true</param-value>
        </config-param>
        <config-param>
          <param-name>FilterChain</param-name>
          <param-value>Transcoding</param-value>
        </config-param>
      </concrete-portlet>
    </concrete-portlet-app>
</portlet-app-def>
```

Alternatively, log in to the portal as an administrator, select Administration ⇨ Portlets ⇨ Manage Portlets, select your portlet, and click Edit Parameters. Then add values for the two parameters as shown in Figure 24-8.

4. The portlet filter calls the portlet to retrieve the content (in the source markup—for example, HTML).
5. The portlet filter then calls the TT runtime sending it the portlet's content. TT uses the information in the profiles as well as style sheets and annotator files to create the new content.
6. The portlet filter returns the content generated by TT as if this is the content generated by the portlet.

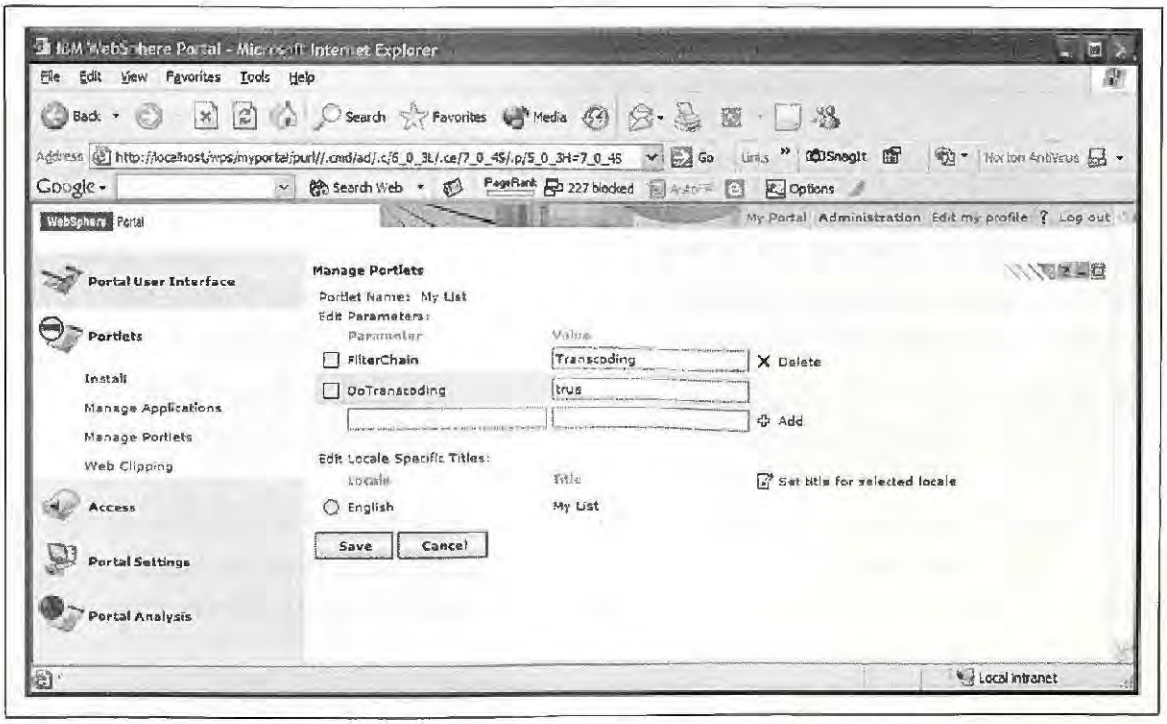


Figure 24-8 Enabling TT for your portlet—adding appropriate parameters.

7. The aggregator returns the aggregated content post-fragmentation (if relevant) and the portlet returns the content to the end user device.

Using XSL for Adapting Content

So long as all content generated by portlets is well-formed XML (and this is a nontrivial assumption that you should pay special attention to), XSL style sheets are a very natural and architecturally elegant solution to generating multiple presentations from a single portlet. In addition, XSL is an industry standard and the use of XSL for mapping one content to another is a common practice, which means that many people will be familiar with this method. There are numerous tools that can help you in doing this right—including the various versions of WebSphere Studio that include an XML editor, an XSL editor, and an XML-to-XML mapping editor.

To use style sheet transcoding you first need to configure the portal for XML processing, which can be done as follows:

1. Edit `<WAS_ROOT>/lib/app/config/services/PortletFilterService.properties` and add entries to allow for transcoding to the various markup types that your portal should support. Look for the last entry in the file starting with `Transcoding.transcodeMarkup` and make sure you use the next available sequence. As an example, for transcoding to WML and cHTML add the following entries:

```
Transcoding.transcodeMarkup.4 = xml->wml
Transcoding.transcodeMarkup.5 = xml->chtml
```

2. Log in to the portal as an administrator and select Administration ➪ PortalSettings ➪ Supported Markups. Click Add New Markup and enter the values as shown in Figure 24-9. Click OK.
3. The new markup type will be added as inactive. Select the xml markup and click Activate/Deactivate Selected Markup.
4. Validate your markup types by clicking Show Info; your display should include an active XML markup as shown in Figure 24-10.

Once the portal has been set up for XML transcoding, you need to tell it which style sheet should be used for transcoding each portlet. Specifying a style sheet per portlet can be done in one of the following three ways:

1. Add a `config-param` element within the `concrete-portlet` element in `portlet.xml` and put the style sheet in the WAR file.

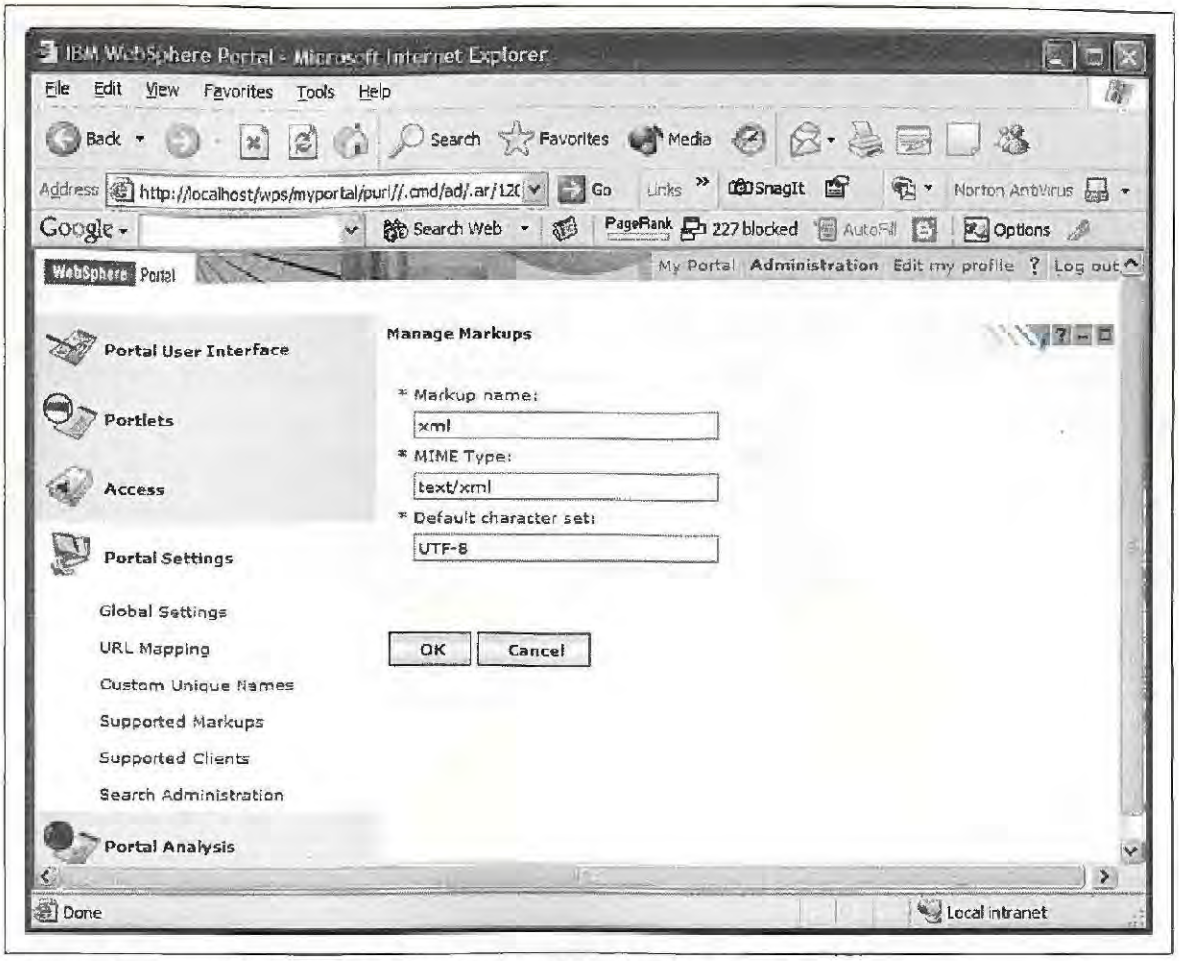


Figure 24-9 Adding support for XML markup for XSL transcoding.

2. Select Administration ➪Portlets ➪Manage Portlets, select your portlet and click Edit Parameters. Then add a parameter called `StylesheetFile` with the file name as the value.
3. If you need to transcode XML content into multiple other markup languages and want to separate the XSL transformation instructions into separate style sheets, your portlet can produce XML output with specific transcoding processing instructions. Each such directive includes a condition that defines the location of the style sheet and a category of user agents for which the style sheet should be applied. As an example, to apply one style sheet for all Nokia phones, which is different from a style sheet used for conventional Web browsers, add the following two directives to the code producing your XML content:

```
<xml version="1.0" encoding="ISO-8859-1"?>
<wtp-condition stylesheet="/browsers.xsl" condition="(user-
agent=*Mozilla*)"?>
<?wtp-condition stylesheet="/nokiaPhones.xsl" condition="(user-
agent=*Nokia*)"?>
... The rest of your XML content ...
```

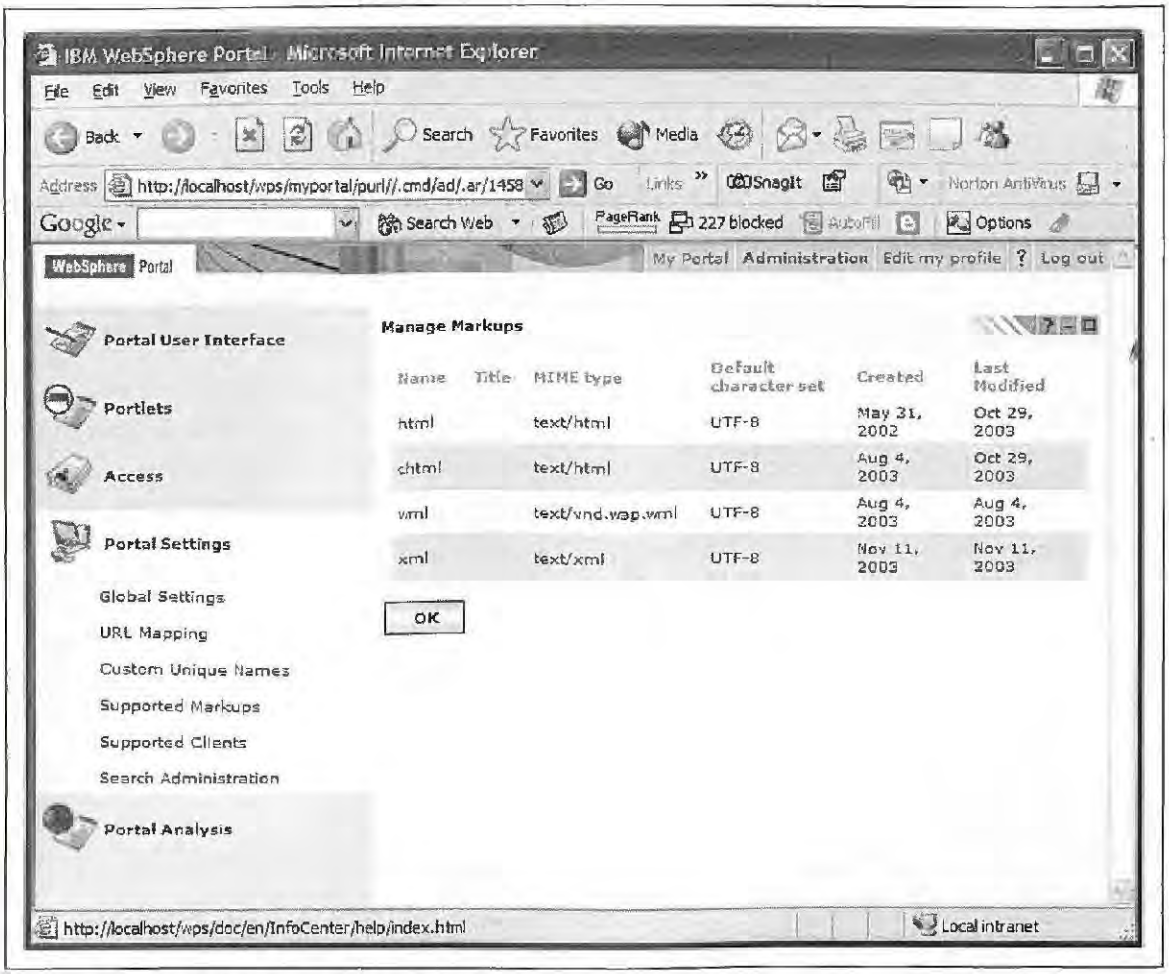


Figure 24-10 Validating active XML markup.

Using Annotators for Adapting Content

The second method to adapt content involves annotators that are TT-specific constructs that can help you transform HTML generated by a portlet. Annotators are important even though they are not a standard technology because XSL sheets are often hard to write while annotators follow a very simple instruction-based mapping metaphor. It is therefore often easier to build an annotator than an XSL sheet. In addition, annotators transform HTML content, which is probably what most of your portlets produce—making them easy to incorporate with your portlet JSPs or classes. You should therefore have both options in your arsenal of techniques.

Annotation instructions come in two flavors: external annotations and internal annotations. External annotations are instructions that are placed in an annotation file (a file with a .ann extension) and comprise two parts—a location and an action. The location is an XPath expression that defines the location at which the action should be applied. The action is an activity such as add or remove and is applied to the fragment of HTML pointed to by the location. Internal annotations have the same functional

behavior but are placed directly within your HTML document as small XML fragments within HTML comments. Annotators are applied by TT after your portlet returns its HTML content. At this point TT traverses your document and applies each annotation instruction. This happens before any other TT processing is performed, for example, transcoding to other markup languages. As an example, if you need to create WML based on a subset of your HTML page, you can add annotators that strip out less important information and then use a generic HTML-to-WML transcoder.

External annotations are placed in a `.ann` file. This is an XML file that has an `<annot version="2.0">` element as the root XML element. The file then follows with a series of `description` elements—each one including a location specified by a `target` attribute and a `take-effect` attribute, and an action as an embedded element. For example, to replace a text fragment in the first paragraph in the body of your HTML document, use the following annotation:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annot version="2.0">
 <description take-effect="before" target="/HTML[1]/BODY[1]
/P[1]/text">
  <replace>
    <text>Replacement Text</text>
  </replace>
 </description>
 ...
</annot>
```

Internal annotations can be added using any text editor. In addition, internal annotations are supported by the Page Designer Classic tool within WebSphere Studio Application Developer (which you have to install as an add-on). When using Page Designer Classic, point to the HTML fragment to which you want to apply the annotation, and right-click and select Annotation along with the action you want to perform. When adding or modifying a fragment, enter the new fragment in the dialog that opens. To make modifications to a TABLE, select the entry and choose Edit ➪Attributes from the menu bar. The last tab on the dialog is the Annotation tab.

Each such annotation changes your HTML code; as an example, if you add a `replace` annotation, your HTML code will now include a fragment similar to (we added line breaks to make it clearer) what follows:

```
<!--METADATA type="Annotation" startspan
<?xml version="1.0"?>
  <annot version="2.0">
    <replace>
      <text>Replacement Text</text>
    </replace>
  </annot>-->
```

```
<P>Original Text</P>
<!--METADATA type="Annotation" endspan-->
```

Finally, if you would like an annotation to apply only to certain devices, use a user-agent-based condition. As an example, if you want replacement text to apply only to Nokia phones you can modify the internal annotation as follows:

```
<!--METADATA type="Annotation" startspan
<?xml version="1.0"?>
  <annot version="2.0" condition="!(user-agent=*Nokia*)">
    <replace>
      <text>Replacement Text</text?>
    </replace>
  </annot>-->
<P>Original Text</P>
<!--METADATA type="Annotation" endspan-->
```

The same set of annotators is available regardless of whether you are using internal or external annotations. Annotators supported by TT include the following:

- `column`—removes a TABLE column located by the XPath expression
- `field`—modifies a field within the HTML form located by the XPath expression
- `insertattribute`—inserts an HTML attribute into the element located by the XPath expression
- `keep`—keeps the HTML fragment located by the XPath expression
- `inserthtml`—inserts content before the HTML located by the XPath expression
- `remove`—removes the HTML fragment located by the XPath expression
- `replace`—replaces the element located by the XPath expression with the specified fragment
- `replacewithhtml`—similar to replace but the content is always an HTML fragment
- `row`—removes a TABLE column located by the XPath expression
- `splitpoint`—specifies a location at which fragmentation should occur. This allows you to control how the fragmentation transcoder (mentioned in the section titled "Transcoding Technology") will break up long documents
- `table`—used with other annotators such as `column` and `row` and affects the heading of the TABLE located by the XPath expression

Transcoding Plug-ins

In addition to the use of TT for generating content for mobile and offline users, WEA provides a set of transcoding plug-ins that can help you prepare portlet content for delivery to mobile units. Plug-ins are generic and handle elements that exist in almost any portlet and hence can often do a lot of work that you would otherwise have to do yourself. Transcoding plug-ins include the following:

- Image transcoding plug-in: this plug-in converts images ensuring that you don't download large images to devices with lower resolution and overslow networks.
- Text transcoding plug-in: this plug-in can convert text fragments (including HTML and XML) from one format to another and can be used to simplify output. For example, it can be used to convert a complex multilevel table to a simple ordered list.
- HTML-to-WML transcoding plug-in: this plug-in converts HTML to WML and can be used to convert existing HTML pages to WML pages without additional custom work on your behalf.
- HTML-to-cHTML transcoding plug-in: similar to cHTML.

Enhanced Portlets

WEA provides not only tools but also packaged applications. These enhanced portlets allow you to support mobile users quickly and with no additional development work. You can support mobile users with out-of-the-box functions on the following devices:

1. Devices with Pocket IE compliant with HTML 3.2 and Javascript support as defined in the Windows CE JS3.0 specification
2. Palm OS devices with Eudora 2.1, the AU Mobile Internet browser and Palm Web Pro Browser 1.0 and higher
3. WML microbrowsers

In terms of the functionality provided, the following portlets are available for use by mobile users:

1. Lotus Notes portlets:
   a. E-mail
   b. Calendar
   c. To-do lists
   d. Contacts

   e. Journal
2. Microsoft Exchange portlets:
   a. E-mail
   b. Tasks
   c. Contacts
   d. Notes
3. Reminder portlet
4. World Clock portlet
5. QuickLinks portlet
6. Internet Mailbox portlet
7. Banner AD portlet
8. Image Viewer portlet

The Application Integrator Portlet

In Chapter 23 we discussed the Application Integrator portlet that allows you to portalize functions implemented within back-end application suites from vendors such as SAP, PeopleSoft, and Siebel. One of the useful features of Application Integrator is that it can also work with pocket-PC-based PDAs. More specifically, you can use Application Integrator to deliver applications to Pocket IE running on a Pocket PC 2000 or a Pocket PC 2002 environment—but you will need WEA version 4.3 or higher. If you also want these functions to be available in an offline mode, you will need to install the WEA client version 4.2.1 or higher.

When using Application Integrator there are some additional settings that you need to specify in order for the back-end functionality to be available on mobile devices. More specifically, remember the following:

1. Choose the markup type from the Select Markup to Configure pull-down list in the Object Builder page.
2. Install the WEA client on all PDAs and RIM devices—Application Integrator supports offline usage for applications once the client is installed.
3. Make sure the clipping portlet is installed—Application Integrator makes use of clippers to create the offline content and deliver it to the mobile unit.

Summary

People are mobile—they have always been mobile. For developers, viewing a user as someone using a full-blown PC connected through a fast connection to the Internet has always been a convenience. But this viewpoint is quickly becoming obsolete—and more so for portal and portlet developers. If you are part of a large-scale WP rollout, there is a good chance that sooner or later you will have to service people on the move using mobile phones, PDA, wireless-enabled laptops, or Blackberry devices. If so, remember that WP and WEA allow you to provide comprehensive support to your users.

From a rollout perspective you can handle many issues relating to multiple devices and markup within WP—and if this covers all of your requirements then your environment can be kept much simpler. If on the other hand you need complex transcoding then you need to introduce WEA into the mix and use its advanced TT capabilities. If you can stay with simple transcoding but need offline operations, then you can either choose WEA or another caching solution such as AvantGo by Sybase or MCP by ViryaNet. Finally, if the networks through which your functionality will be accessed include non-IP networks, then you need to introduce a gateway such as WebSphere Wireless Gateway or a third-party gateway such as the Broadbeam gateway or integrate proprietary SDKs that are usually provided by the network provider.

Index

SYMBOLS

* asterisk, 190
comparison rules, 169
  includes any of, 169
  includes, 169
  is between but not equal to, 169
  is between, 169
  is equal to, 169
  is greater than or equal to, 170
  is greater than, 169
  is included in, 170
  is less than or equal to, 170
  is less than, 170
  is not equal to, 170
(#) hash, 122
(.) period, 190
(+) query operator, 235
(-) query operator, 235
(_) underscore, 190

A

abstract portlet, 248, 250
AbstractAction class, 355
AbstractPortlet, 263
access controls
  overview of, 256
  migrating, 109
Action attribute, 301
Action classes
  overview of, 308, 341, 351, 358
  implementing, 309
action event processing, 345
action events, 257
ActionClassManager, 307, 358
ActionEvent object, 271, 306
ActionEvent.getActionString () method, 115
ActionForward instance, 356
ActionListener interface, 302
ActionListener, 338
actionPerformed() event listener method, 264
actionPerformed() method, 256, 271, 288
actionPerformed(ActionEvent event) method, 271
ActionRequest object, 264, 338, 341
ActionResponse object, 265, 338
ActionURL, 338, 340
Active Directory over SSL, configuring, 382
Active Directory, 375, 383
adapter-based integration, 442

adapters
calling procedures, 442
session handling, 442
transaction handling, 442
Add Entry function, 372
addAction(), 270
Administration label, 128
Administration page, 192, 332
administrative privileges, 128
Advance Options, 197
advance query operators, 10
advanced page properties, editing, 138
Advanced Search, 231
Always clause, 170
anatomy of theme, 152
annotation file (.ann), 490
annotation instructions, types of
external annotations, 490
internal annotations, 490
annotators for adapting content, using, 490
Apache Ant Tool, 66
Apache Jakarta project, 345
APGUIBrowserLang, 467
Appearance tab, 139
application code debugging, 330
Application Integrator portlet, 494
Application Integrator, 494
Application Programming Interface (API), 13
application server, 5, 408
application sharing, 224
application unique identifier (UID), 291
APPLICATION_SCOPE, 266
application-specific Web embedding, 466
Aptrix/Lotus Workplace Web Content Management, 165
Assign Permissions, 153
attribute types, 368
attributeAdded() method, 258
attributeRemoved() method, 258
attributeReplaced method, 258
attributeReplaced() method, 258
authentication engine, 388
authentication for large enterprises, implementing, 365
authentication registry, 365
authentication server, 388
authentication, 365
automated server management, 5

B

B2B portal, 223
B2C portal, 223
back-end application, 386, 455, 470
back-end integration patterns, 441
back-end integration, 463
back-end SSO, 455
back-end system, 456
Banner AD portlet, 494
base distinguished name (DN), 369
base project, creating, 346
Basic Portlet, 284
binary types, 329
binder rule, 170
Blackberry devices, 495
branching methodology, 370
breakpoints, 328
Broker class, 313
Broker class, abstract methods
deletePoll(String pollName), 320
getPoll(String pollName), 320
getPollNames(), 320
hasUserVoted(String, 321
hasVotingStarted(String pollName), 320
updatePoll(PollBean poll), 320
vote(String userid, String pollName, int choiceIndex), 320
voteCount(String pollName, int choiceIndex), 321
Broker class, static methods of
DbBroker, 321
MemoryBroker, 321
Broker getInstance() method, 321
built-in fragmentation transcoder, 484

built-in portal services, 455
built-in search engine, 235
Business Application Programming interface (BAPIs), 444
business integration, 6
bytes, 384

C

campaign management tools, features of, 11
Campaign Management, 171
capableof, 157
cascading portals, 137
Cascading Style Sheets (CSS), 280
category tree, 211
certificate authority (CA), 380
Certificate Services, 375
certificates, 380
challenges of high availability, 403
Change Code Generation, 334
cHTML (compact HTML), 476
class APIs, 165
Class prefix, 335
client/server, 380
CMS Key Database file, 381
code generation, 6
Collaboration Center
  People Finder, 10
  QuickPlace, 10
  Sametime, 10
  Web Conference Center, 10
collaboration feartures of WebSphere Portal
  Domino Web Access portlets, 10
  Domino, 10
  Notes, 10
collaboration software, 3
collaborative components Java APIs, 232
collaborative components, 232
collaborative features, 223
collaborative filtering, 164
collection, searching, 235
com.ibm.sample.poll.portlet, 298
commercial off-the-shelf (COTS), 439
common thread, 163
common UI facades, 256
common user interaction model, 256
common user interface (UI) framework, 242
components that impact availability, 406
concrete portlet applications, 247, 293
ConcretePortlet object, 248
ConfigPg2Action, 307
ConfigService.properties file, 273
configuration mode
  CancelAction, 310
  ConfigAddChoiceAction, 310
  ConfigDeleteChoiceAction, 310
  ConfigPg2Action, 310
  ConfigPg3Action, 310
  ConfigSaveAction, 310
  ConfigVerifyDatasourceAction, 310
configuration mode, action of, 310
configuration objects, 265
configuration parameters, 252
configuration synchronization, 409
configuration templates, 67
configure.jsp, 302
ConfigureFormData object, 301
ConfigurePg2View, 315
ConfigurePg3View, 315
ConfigureView, 315
DB2 Server, configuring, 70
ConfigVerifyDatasourceAction actionPerformed method, 329
ConfigVerifyDatasourceAction class, 308, 310
ConfigVerifyDatasourceAction, 307
Container Managed Persistence (CMP), 326
content access service, 243
content hierarchy, browsing, 129
Content Library, 215
content management tool, 201
content management, 4, 9

Content Root
  Labels, 128
  Pages, 128
  URLs, 128
Content Root composition, default, 129
content spots, 165
Content tab, 139
content template, creating, 210
Content Viewer, 216
content
  adding, 215
  categorizing, 211
ContentAccessService, 361
Context Parameters, 246
context parameters, 250
Control.jsp file, 161
controller, implementing, 298
cooperative portlets, 288
core migration properties
  includeApps, 109
  includePages, 109
  includePlaces, 109
  PortalAdminId4x, 109
  PortalAdminPwd4x, 109
  WpsContextRoot4x, 109
  WpsDefaultHome4x, 109
  WpsHostName4x, 109
  wpsinstallLocation4x, 109
  WpsPort4x, 109
createActionURL() method, 272
createRenderURL() method, 272
createReturnURI() method, 270
createURI() method, 270
credential vault data, migrating, 117
Credential Vault segments, 455
Credential Vault service API, using, 458
credential vault service, 270
Credential Vault slots
  Private credentials, 457
  Shared credentials, 457
  System credentials, 456
Credential Vault slots and segments, migrating, 117
Credential Vault
  creating, 217
  using, 454
CredentialSlotConfig class, 457
CredentialVault service, 361
CredentialVaultService interface, 454
CredentialVaultService, 270
cross-domain SSO, 389
CrossWorlds architecture, 442
CSEnvironment class, 232
CSEnvironment properties, 234
CSEnvironment.properties file, 233
CSS style sheets and images, 144
CSS style sheets, 150, 153
Currency Calculator portlet, 134
custom categorization, creating, 185
custom names, 199
custom unique names, modifying
  Pages, 182
  Portlet Applications, 182
  Portlets, 182
  URL Mapping Contexts, 182
  User Groups, 182
custom window states, 273
Customizing WebSphere Portal, 65
CV segment, creating, 456

D

data beans, 357
data modeling, 173
data source, 325
data sources, 237, 392
data type, 384
database broker, 320
database properties
  DbDriver, 79
  DbDriverDs, 79
  DbLibrary, 80
  DbPassword, 79
  DbSafeMode, 79
  DbType, 79
  DbUrl, 79
  DbUser, 79
  WpsDbName, 79

WpsDbNode, 80
WpsDsName, 80
WpsXDbName, 80
database users, creating, 75
DatabaseBroker, 321
databases, creating, 75
DB2 administrator, defining, 73
DB2 client installation component list, 74
DB2 Client on WebSphere Portal Server, installing, 71
DB2 Content Manager, 205
DB2 features required on WebSphere Portal Server, 72
DB2 Information Integrator, 10
DB2 with WebSphere Portal, using, 69
debugging, 280
dedicated authentication server, 388
default screen, 149
default skins
Album, 146
Clear, 146
Corner, 146
Diamonds, 146
Fade, 146
Hint, 146
Noborder, 146
Noskin, 146
Outline, 146
Pinstripe, 146
Shadow, 146
Wave, 146
DefaultPortletMessage, 258
deployable WAR File, generating, 332
install administrative portlet, deploying, 332
XML configuration interface, deploying, 333
deployment descriptor editor, 291
deployment descriptor, portlet properties of
caching preferences, 246
markups, 246
modes, 246
portlet configuration parameters, 246
portlet display name, 246
supported portlet states, 246
deployment descriptors, 336
Deployment Manager Administration console, 414
Deployment Manager, 409
destroyConcrete() method, 266
digital keys, 6
directory information tree (DIT), 369
directory information tree, designing, 369
Directory Management, 372
directory. *See* LDAP or database, 366
Discovery Server, 10
disk drives, 408
Distinguished Name (DN), 388
DNS naming model, 370
DNS switchover, 370
do() method, 338
doConfigure method, 307, 338
document and content management, 201
document collections, managing, 187
document indexer, 10
Document Manager, 201
Document Search, 235
doDispatch() method, 338
doEdit(), 257, 264, 338
doHelp() method, 264
Domino servers, 10
Domino Web Access (iNotes) portlet, 225
Domino, 371
doView() method, 284, 294, 298, 309
dual network cards, 370
dynamic caching, 416
dynamic servlets, 416

E

EAI adapters, 450
EAR file, 285, 418
e-business applications, 7
e-business, 5

Eclipse IDE, 280
ECLIPSE open standard framework, 6
Eclipse technology, 280
Eclipse-based Studio environment, 281
e-commerce, 163
edit mode page, 359
edit mode, 314
Edit Page link, 148
Edit Page properties, 146
Edit Page, 153
EditSaveAction class, 314
EJB Agent, resource protection levels of
  Bean methods, 398
  Beans, 398
  entire package, 398
  method signatures, 398
EJB method, 398
EJBs, 386
e-mail rule
  bccRecipient, 171
  bodyURI, 171
  ccRecipient, 171
  Recipient, 171
  Sender, 171
  Subject, 171
embedding functional portals, 469
Empty Portlet, 284
Enable Tracing portlet, 330
encodeURL() method, 270
encryption algorithm, 380
encryption and authentication, 380
End user, 144
engine *See* foundation and tools, 5
engine.tld, 156
enhanced prtlets, 493
Enterprise Application
  PeopleSoft 7.5 with PeopleTools 7.54, 452
  PeopleSoft HRMS 8.30 with PeopleTools 8.18, 452
  SAP R/3, 452
  Siebel 7 or higher, 452
Enterprise Application (EAR), 326
enterprise application systems
  PeopleSoft, 14
  SAP, 14
  Siebel, 14
Enterprise Applications
  JAAS for JavaMail, 17
  Java API for XML Processing (JAXP), 17
  Java IDL, 17
  Java Transaction Architecture for persistent transactions, 17
  JavaMail, 17
  JDBC for database connectivity, 16
  JMS for messaging, 17
enterprise database, 366
enterprise identity management, 365, 371
enterprise information system, 443
Enterprise Resource Planning (ERP), 392
Enumeration findResourcesBy-PropertyQuery(String name, String value, ResourceContext context), 175
Enumeration findResourcesByQuery (Query query, ResourceContext context), 174
Enumeration keys(), 174
Event interface, 256
event listeners, 256
Event object, 340
Event parameter, 340
event processing, 257, 340
Everyplace Toolkit, 479
Excel spreadsheets, 184
Excel, 9
Exclude clause, 170
execute method, 355
EXPIRATION_CACHE attribute name, 268
Export utility, 283
Expressions view, 329
Extended Search, 237
extended user attributes, migrating, 111
external applications, 443

external security manager, 391
File server, 115

F

File server, 115
file transfer services, 409
foundation and tools, 5
fragmentation transcoder, 485
frame *See* foundation and tools, 5
free-text queries, 10
front-end integration, 463
functional layer, 421

G

gateways, 484
generated portlet code, 335
generated source code, 294
generateURL tag, 160
GenericPortlet class, 338
GenericPortlet, 263, 338
getApplicationSettings() method, 251
getAttribute() method, 115, 267
getAttributeNames() method, 115
getData() method, 252
getInitParameters() method, 250
getJSPExtenstion method, 294
getLastLoginTime() method, 115
getLastModified() method, 268
getName() method, 342
getPortletLog() method, 255
getPortletSession() method, 253
getResponseContentTypes() method, 263
getRuleContent() method, 166
getSupportedWindowStates() method, 273
getsurname() method, 174
getUser() method, 256
getUserPrinciple method, 342
getValue method, 341
GIF image files, 153
Global Settings, 179
GQL (Generalized Query Language), 237
Group permission portlet, 191
gsk6ikm, 380

H

hardware requirements for installing WebSphere Portal, 26
HDML, 475
Head.jsp, 153, 154
helper class, 358
high-availability strategy for WebSphere portal, designing, 405
highly available WebSphere portal solution, implementing, 409
Home screen, 149
href attribute, 301, 466
href tag, 160
HTML links, 166
HTML pages, 470
HTML, 9, 136
HTTP Basic Authentication, 464
HTTP port number, 324
HTTP request, 257, 388
HTTP requests (URLs), 463
HTTP servlets, 12
HttpServletRequest object, 355
HttpServletResponse object, 355
HttpSession objects, 254
HttpSession, 253

I

IBM developer domain, 442
IBM Directory Server, 372, 412
IBM HTTP Server installation directory, specifying, 30
IBM HTTP Server, 25, 34
IBM HTTP Web Server, 412
IBM Lotus Extended Search product, 237
IBM Lotus Extended Search, 10
IBM Lotus Team Workplace, 228
IBM Lotus Workplace Web Content Management (ILWWCM), 9, 201

IBM Lotus Workplace Web Content Management 1.1, 87
IBM portal catalog, 7
IBM Portal Toolkit, 275
IBM portlet API addressability, 270
IBM portlet API events
  ActionEvent, 264
  Event phase life cycle, 264
  MessageEvent, 264
  PortletApplicationSettings-AttributeEvent, 264
  PortletSettingsAttributeEvent, 264
  Render phase life cycle, 264
  Session life cycle, 264
  WindowEvent, 264
IBM Portlet API jar files, 323
IBM portlet API listener
  ActionListener, 264
  EventPhaseListener, 264
  MessageListener, 264
  PortletApplicationSettings-AttributeListener, 264
  PortletPageListener, 264
  PortletSessionListener, 264
  PortletSettingsAttributeListener, 264
  WindowListener, 264
IBM Portlet API, 259, 262, 297
IBM proprietary SSO token, 388
IBM Tivoli Web Site Analyzer 4.5, 18
IBM Tivoli Web Site Analyzer, 12
IBM UDDI Business Test Registry, 427
IBM UDDI business test registry, using, 426
IBM UDDI Test Registry, 427
IBM WebSphere Application Server features panel, 37
IBM WebSphere Application Server, 17
IBM WebSphere Personalization
  recommendation-based personalization, 164
  rules-based personalization, 164
IBM WebSphere UDDI Registry, 426
IBM's Struts Portlet Framework, 345
IDE, 174
Identity Management component, 368
Identity Management systems, 370
identity management tools, 392
iFrames, 225
IllegalArgumentException, 263
ILWWCM Portlet Viewer, content settings for, 219
ILWWCM, objects supported by
  Component-Reference, 211
  File Resource, 211
  HTML, 211
  Image file, 211
  Rich Text, 211
  Text, 211
Image Viewer portlet, 494
images, 9, 150
i-mode phones, 476
Import function, 283
include() method, 264
includePages property, 116
index.jsp file, 358
inetOrgPerson LDAP object class, 267
Information and Content Exchange standard, 221
information model, designing, 368
information provider SPI, 243
infrastructure, designing, 370
init() method, 250
initConcrete() method, 250, 266
InitialEditView, 315
initialization parameter, 358
initializeResults method, 317
Install pages, 332
Install portlet, 333
installing SiteMinder, steps for, 399
Installing WebSphere Portal Fixpack 2, 39
installing when WebSphere security is enabled, 63
instant messages, 227
instant messaging, 224, 226

Integrated Development Environment (IDE), 280
integrates, 10
integration patterns
  adapter-based integration methods, 442
  portlet-based integration methods, 442
integration patterns within WP, classification of, 441
interactive debugging, 280
interface using WSDL, using, 446
Intermediate Documents (IDOCs), 450
internal security services, 393
Internet Explorer (IE), 475
Internet Information Services (IIS), 375
Internet Mailbox portlet, 494
Internet Transaction Server (ITS)
  AGate, 393
  WGate, 393
Internet, 3
Internet-enabled phones, 474
invalidateCache() method, 268
IP-based network, 482
ITS gateway interface, 393

J

J2EE Connector Architecture (JCA), 443
J2EE Connector Architecture, 5
J2EE Engine, 16
J2EE security, 386
J2EE servlet deployment, 258
J2EE Settings page, 334
J2EE WAR files, 242
J2EE Web application, 281
J2EE-compliant Web application, 283
JAAS architecture, 13
JAAS Authentication Entry, 325
JAAS layers, 386
Jakarta Struts, 345
Java 2 Platform, Enterprise Edition (J2EE), 5
Java API, 224
Java application development, 280
Java Application Model
  applets, 16
  Enterprise Java Pages (EJPs), 16
  Java application clients, 16
  JSPs, 16
  servlets, 16
Java applications, 5, 12
Java Authentication and Authorization Service (JAAS), 115
Java Build Path, 323
Java class, 388
Java clients, 386
Java code, 180
Java Community Process portlet, 259
Java Community Process specification, 259
Java Connector Architecture (JCA), 442
Java Cryptographic Architecture, 5
Java Cryptographic Extension, 5
Java development environment, 280
Java Management Extensions, 5
Java Messaging Service (JMS), 5
Java Portlet Specification JSR 168, 259
Java portlet, 216
Java programs, 422
Java Proxy, generating, 428
Java run-time environment, 16
Java Secure Socket Extension, 5
Java security model, 13
Java Server Faces (JSF), 361
Java Server Pages (JSP), 280
Java Server Pages Standard Tag Libraries (JSTL), 304
Java Source packages, 339
Java Specification Request (JSR), 259
Java wrapper bean, 165
java.security.Principle object, 342
JavaScript callback functions, 234
JavaScript editing, 280
JavaServer Pages, 348
JavaSource directory, 317
javax.portlet.GenericPortlet, 338
JCA adapter, 455
JCA resource adapters, 443

JCA Tool Plugin, 445
JDBC driver library, 325
JDK Native-to-ASCII converter, 180
JRE, 280
JSF environment, 361
JSF events, 361
JSP differences, 358
JSP files, 8, 288
JSP invocation and tag libraries, 264
JSP pages, 180
JSP scriptlet, 232, 360
JSP tag library, 272
JSP, 115, 150
JSR 168 API addressability, 272
JSR 168 API, 259, 361
JSR 168 portlet class, 263
JSR 168 portlets for WebSphere portal, developing, 273
JSR 168, 16
JVM, 410

L

label nodes, 135
layout editing, advanced, 138
layout tools, 138
LDAP server certificates, 380
LDAP configuration, 267
LDAP Data Interchange Format, 372
LDAP database, 107
LDAP directory tree, 369
LDAP Directory Web Administrator, 372
LDAP directory, 230, 366
LDAP directory, information storage in, 368
LDAP or database, 366
LDAP properties configuration
  LDAPAdminPwd, 378
  LDAPAdminUid, 378
  LDAPBindId, 378
  LDAPGroupMember, 378
  LDAPGroupObjectClass, 378
  LDAPGroupPrefix, 378
  LDAPGroupSuffix, 378
  LDAPHostName, 378
  LDAPPort, 378
  LDAPServerType, 378
  LDAPsslEnabled, 379
  LDAPSuffix, 378
  LDAPUserObjectClass, 378
  LdapUserPrefix, 378
  LDAUserSuffix, 378
  LookAside, 377
LDAP Schema, 368
LDAP server, 365
LDAP Servers, setting up, 372
LDAP user registries, 362
LDAP user registry, configuring, 390
LDAPRepository tag, 384
LDAPs
  IBM Directory Server V5.1, 371
  Microsoft Active Directory, 371
  Sun One, 371
LDAPs, implementing, 372
LDIF file, 372
LDIF import, 373
LDIFDE utility, 375
legacy applications, 5
Lightweight Directory Access Protocol (LDAP), 6, 366
Lightweight Third Party Authentication (LTPA), 388
Lightweight Third Party Authentication (LTPA), 6
Linux configuration, 44
Linux, 372
load balancing, 370
local environment debugging, 324
local Portal test environment, 323
locale, 157
Localization, 262
locks, 139
log.properties file, 330
loggedIn, 157
logical structure of Portal, 128
Lotus Collaboration Components
  Lotus Discovery Server, 232
  Lotus Domino, 232

Lotus QuickPlace, 232
Lotus Sametime, 232
Lotus Domino server, 224
Lotus Domino, 224
Lotus Notes databases
Calendar, 10
Discussion, 10
Mail, 10
Notes View, 10
TeamRoom, 10
To Do, 10
Lotus Notes integration, 224, 225
Lotus Notes portlets
calendar, 493
contacts, 493
e-mail, 493
Journal, 494
To-do lists, 493
Lotus QuickPlace Integration, 228
Lotus QuickPlace, 224
Lotus Sametime Integration, 226
Lotus Sametime, 224
LTPA Token authentication, 270
LTPA token, 225
LTPS token, 388

M

mailto link, 231
MainView class, 316
MainView state class, 342
MainView, 315
Manage Pages window, 128
Manage Pages, 129
mapping of permissions to roles
Create, 110
Delegate, 110
Edit + Delegate, 110
Edit + Manage, 110
Edit, 110
Manage + Delegate, 110
Manage, 110
View + Delegate, 110
View + Edit + Delegate, 110
View + Edit + Manage + Delegate, 110
View + Edit + Manage, 110
View + Edit, 110
View + Manage + Delegate, 110
View + Manage, 110
View, 110
markup fragments, 258
markup language, 135
markup within WSSD, addding, 477
markup, 430
master server, 370
member database, 365
Member Manager attribute, 384
Member Manager attributes to LDAP attributes, mapping of, 383
Member Manager properties
WmmDbName, 83
WmmDbPassword, 83
WmmDbUrl, 83
WmmDbUser, 83
WmmDsName, 83
Member Manager, 365
memory broker, 320
memory management, 411
memory-to-memory session replication, 410
Menu component, 213
message authentication, 5
message events, 257
MessageListener interface, 258, 289
messageReceived() method, 256, 289
messaging middleware adapters, 442
microbrowsers, 473
Microsoft Active Directory, 375
Microsoft Exchange Integration
Microsoft Exchange Calendar, 226
Microsoft Exchange Contacts, 226
Microsoft Exchange Mail, 226
Microsoft Exchange Notes, 226
Microsoft Exchange Tasks, 226
Microsoft Exchange portlets
Contacts, 494
e-mail, 494
Notes, 494
Tasks, 494

Microsoft Exchange server, 226
Microsoft NetMeeting Integration, 229
Microsoft SQL Server, 75
Microsoft's Active Directory, 370
middleware products
  TIBCO, 450
  Vitria, 450
  WebMethods, 450
middleware, 408
migrate look aside database, properties to
  Db2Home, 113
  LDAPHasUid, 113
  LdapUuidName, 113
  WmmConfigType, 113
  WmmUid, 113
  WmsDbName, 113
  WMSDbPassword, 113
  WmsDbUrl, 113
  WmsDbUser, 113
migrating all user customizations, 117
migrating everything at one time, 118
migrating pages, 116
migrating places, 116
migration environment, recommended, 106
migration property values, setting up, 108
MIME types, 255
mobile phones, 473
mobile users, supporting, 473
Model classes, implementing, 317
model objects, 341
Model—View—Controller (MVC), 298
module initialization parameters, 351
multiple connectivity options
  CDMA, 483
  CDPD, 483
  EDACS, 483
  GPRS, 483
  GSM, 483
  Hot Spots and 802.11b networks, 483
  iDEN, 483
  Mobitex, 483
  satellite networks, 483
  TDMA, 483
  TETRA, 483
multiple HTTP requests, 253
multiple JVMs, 410
multiple markups, supporting, 476
multiple wireless environments, supporting, 482
multiple-carrier SDKs, 483
MVC framework, 361
My Portal, 129
mySAP.com integration, 392

N

native2ascii, 180
navigationAvailable, 157
Navigator pane, 323
navigators and menus, creating, 212
NavigatorService.properties file, 254
Netegrity SiteMinder, 270, 386, 392
NetMeeting Active-X component, 229
NetMeeting interface, 229
Netscape Directory Server, 374
Network Access Storage systems, 410
network attached storage (NAS), 408
network bandwidths, 370
network cards, 408
network configuration utility, 45
new nodes, adding, 131
New Page, 153
new portal, creating, 140
new project, creating, 345
new skin, creating, 161
new theme, creating, 150
newWindow, 157, 158
Node Manager, 411
node properties, editing, 135
non-graphic Linux installation, 56
Normal View, 171
Notes databases
  MyNotesCalendar, 225
  MyNotesMail, 225
  MyNotesToDo, 225
  NotesDiscussion, 225

NotesMail, 225
NotesTeamRoom, 225
NotesView, 225
NotesView portlet, configuration parameters
NotesDatabase, 225
NotesServer, 225
NotesView, 225
PortletType, 225
VAULT_SLOT, 225

O

object classes, 368
Object get () method, 174
offline page, 480
online meetings, 226
open platform, 280
open-source framework, 345
operating systems
AIX, 6
HP-UX, 6
Linux, 6
OS/400, 6
Solaris, 6
Windows 2000/2003, 6
zOS, 6
Oracle and SQL Server with WebSphere Portal, using, 75
Oracle database users, creating, 77
Oracle database, 75
org.apache.jetspeed.portlets .AbstractPortlet class, 248
org.apache.jetspeed.portlets .PortletAdapter class, 248
org.apache.pluto.tags.* class, 338
Organize Favorites, 128
Otherwise clause, 170
own skins, defining, 161
own themes, defining, 149

P

pacCheck, 158
Package prefix, 335
packaged business applications
J.D. Edwards, 439
Oracle, 439
PeopleSoft, 439
SAP, 439
Siebel, 439
page aggregation, 243, 258
page content, defining, 133
Page Customizer, 128
Page Design, 213
Page Designer Classic tool, 491
Page Designer, 213
page inheritance with shared pages, 137
page nodes, 128
page object references, 359
Page Properties, 128
page2Action method, 307
pageAvailableNext, 157
PageBarInclude.jsp, 153
pageBookmarkable, 157
pageCompletelyActive, 157
pbportlet.jar file, 115
PDA, 479
PDM, features of, 10
People Finder portlet, 230
People Finder, 229
peopleservice tag, 233
PeopleSoft portal, 469
PeopleSoft, 392
performAction method, 356
performance monitoring, 409
performView method, 356
permissions, 191
Persistence classes, 358
persistence classes, implementing, 320
Personal Digital Assistants (PDAs), 473
Personalization APIs, 173
personalization resources, 165
PlaceBarInclude.jsp, 153, 155
platform for privacy preferences (P3P) keys, 267
Pluggable Authentication Service (PAS), 393
pluginAttributeName, 384

plug-ins, 281
plug-n-play, 13
Pocket IE, 475
Pocket PC operating system, 474
pocket-PC-based PDAs, 494
Poll instance, 320
poll portlet implementation, remaining, 357
Poll portlet project, creating, 283
Poll portlet with JSR 168 API, developing, 333
Poll portlet, 275, 297
Poll project source, modifying, 339
Poll project, creating, 334
Poll Struts, 348
PollBean object, 317
PollConstants interface, 321
PollMessageException, 321
PollPortlet class, 338, 339
PollPortlet controller class, 358
PollPortlet service method, 340
PollPortlet.java file, 298
PollPortletSessionBean class, 294
PollPortletViewBean class, 294
PollWrapperException, 321
pop-up menu, 323
portal access control, 190
Portal administration
  access, 179
  portal analysis, 179
  portal settings, 179
  portal user interface, 179
  portlets, 179
Portal Administration console, 414
portal analysis, 198
portal application integrator, using, 452
portal architecture, 242
Portal config database, 119
portal debug mode, 328
Portal Default Theme, 146
portal engine, 243
portal environment, 241, 243
Portal Information Center, 330
Portal JSP tags
  constants, 156
  if, 156
  navigation, 156
  screenRender, 156
  text, 156
portal page
  banners, 242
  headers, 242
  navigation area, 242
portal server configuration, 282, 324
Portal Server Credential Vault (CV), 454
Portal Server engine, 156
Portal Server
  features of, 235
  starting, 34
portal settings, 179
Portal structure, customizing, 131
Portal Toolkit 5.0.2, 87
Portal Toolkit installer, 282, 324
Portal Toolkit, 275, 280, 297
Portal Toolkit, system requirements, 88
portal trace logging, 330
Portal User Interface, 179
portal Web application URI, 270
Portal Web design, tailoring, 143
portal, 3, 223
portal, need for, 3
PortalContext, 267
Portals and Pages, 127
portals and pages, defining, 127
PortalStrutsBlank.war file, 345
PortletSession object, 249
portlet API comparison, 260
portlet API in WebSphere portal, 260
portlet API, 243, 248, 254
portlet API, features of
  action event processing, 297
  database access, 298
  internationalization, 298
  multiple portlet modes, 297
  page navigation, 297

portlet configuration parameters, 298
trace logging, 298
user customization parameter, 298
portlet Application Programming Interfaces (APIs), 259
portlet application properties, 246
portlet application settings attributes events, 258
portlet application WAR file, building, 283
portlet application, 246, 284
portlet applications and portlets, 246
portlet applications
debugging, 324
migrating, 114
portlet caching, 268
portlet class definition, 263
portlet configuration objects
PortletData, 250
PortletSession, 250
PortletSettings, 250
portlet configure mode, 353
portlet container for JSR 168, enabling, 273
portlet container, 16, 241, 243, 256
portlet content types, 263
Portlet Controller Class, 358
portlet creation wizard files
portlet-api.jar, 338
wps.jar, 338
wpsportlets.jar, 338
portlet deployment descriptor, 246, 252, 349
portlet development, 259, 281, 297
portlet edit mode, 353
portlet event model, 256
Portlet interface, 263
portlet invoker API, 243
portlet jar files, 323
portlet life cycle, 248, 265
portlet management, 430
portlet menus, 270
portlet modes and states, 244
portlet navigation, 242
portlet objects, 254
portlet perspective, 281, 283
portlet preview, 282
portlet processing model, 263
portlet project, 281
portlet render phase, 264
portlet rendering, 258
portlet request object, 243
portlet response object, 243
portlet services
ContentAccessService, 269
CredentialVaultService, 269
portlet services and objects, 267
portlet session object, 243
portlet setting information, accessing, 243
portlet settings attributes events, 258
portlet settings, 219
Portlet states
maximized, 155
minimized, 155
normal, 155
solo, 155
portlet tag library, 265
portlet toolkit, 323, 330
portlet URI addressability, 270
portlet URIs, generating, 360
portlet URL addressability, 359
portlet view mode, 351
portlet window, 273
portlet, implementing, 345
portlet.tld tag library, 342
portlet.xml editor, 281
PORTLET_SCOPE, 266
PortletAdapter, 248
PortletApplicationSettings, 250, 255
PortletApplicationSettingsAttributes-Listener interface, 258

PortletConfig object, 250
PortletContext getService() method, 269
PortletContext object, 255
PortletContext, 254
PortletData object, 249, 252
PortletLog object, 255
portletMaximed, 157
PortletMode instance, 340
PortletPreferences, 341
PortletRequest object, 252, 255, 267, 306
PortletRequest.getUser() method, 115
PortletRequestDispatcher include() method, 264
PortletResponse object, 255
portlets and servlets, differences between, 247
portlets into a WebSphere portal cluster, deploying, 417
portlets on page, setting up, 217
portlets using adapters, developing, 443
portlets using EAI connectors, developing, 449
portlets, 4, 7, 241
portlets, dynamic administration, 247
portlets, processing phases of
  event phase, 256
  rendering phase, 256
PortletService interface, 269
PortletSession object, 250, 254
PortletSettings object, 248
PortletSettings, 252, 255, 322, 341
PortletSettingsAttributesListener interface, 258
portletState, 155
portlet-to-portlet communication, 258
PortletURI tags, 307, 309
portletWindowState, 158
predefined static taxonomy, 236
Presentation Editor, 203
presentation layer, 421
private radio networks, 476
processAction method, 338
processAction method, parameters of
  ActionRequest, 340
  ActionResponse, 340
processAction() method, 264, 272
profiling services, 11
Project Navigator view, 291
projects, adding and removing, 327
property broker, 269
protocol version, 380
Public Key Infrastructure (PKI), 5
public registries, 426
public.session property, 254
public-key encryption, 380

Q

query assistance, 10
query word completion, 10
Quick Search, 231
QuickLinks portlet, 494
QuickPlace (Lotus Team Workplace), 10
QuickPlace portlets, 229

R

R/3 Basis, 393
R/3 screens, 393
RDBMS
  DB2, 237
  Oracle, 237
Receive Updates option, 203
recommendation-based personalization, 164, 175
RedHat Enterprise Linux, 43
registration, 430
Reminder portlet, 494
Remote Function Call (RFCs), 444
remote portal server, 282
remote portlets in WP, using, 424
removing nodes, 139
render parameters, 265
render() method, 264
RenderRequest object, 264, 338
RenderResponse object, 263
RenderResponse setContentType() method, 263

RenderURL, 338
resource adapter archive (RAR) file, 443
resource bundles, 180
Resource Engine, 165
Resource findById(String name, ResourceContext context), 174
Resource getForUpdate(String id, ResourceContext context), 175
resource interface, 174
Resource Permissions
  Pages, 193
  Portlet Applications, 193
  Portlets, 193
  URL Mapping Contexts, 193
  User Groups, 193
  Virtual resources, 193
  Web Modules, 193
  WPCP Projects, WPCP Editions, WPCP resource Collections, WPCP Directories, WPCP Resources, 193
resource, user, and group permissions, 192
ResourceBundles, 322
ResourceContext class, 174
ResourceDomain3 interface, 174
ResourceDomain3, 174
ResourceManager3 interface, 175
resumeLevel, 158
resumeOption, 158
reverse proxy, 387
Rich Site Summary (RSS)format, 9
role definitions
  Administrator, 193
  Delegator, 193
  Editor, 193
  Manager, 193
  No Access, 193
  Privileged User, 193
  Security Administrator, 193
  User, 193
roles. *See* permissions, 192
RSS (XSLT) portlet, 115
rule editor, 166
rules for designing and maintaining LDAP server, 367
rules
  defining, 166
  editing, 166
rules-based personalization, 164, 175
runtime configuration, 244
run-time environment, 5, 243

S

Sametime
  electronic meeting support, 11
  shared white boards, 11
Sametime (Lotus Instant Messaging), 10
Sametime connect applet, 226
Sample Page, 133
SAP applications, 395
SAP BW (business information warehouse), 392
SAP Enterprise Integration portlets, 462
SAP enterprise portal integration, 395
SAP Enterprise Portals
  SAP Enterprise Collaboration Portal (SECP), 395
  SAP Enterprise Information Portal (SEIP), 395
  SAP Enterprise Unification Portal (SEUP), 395
SAP portals, 469
SAP R/3 modules, 392
SAP R/3 system, 444
SAP service and using SAP Proxy, generating, 446
SAP software, 392
SAPGUIBrowserClient, 467
SAPGUIBrowserDetach, 469
SAPGUIBrowserHost, 467
SAPGUIBrowserIFrameHeight, 468
SAPGUIBrowserIFrameWidth, 468
SAPGUIBrowserPathExtension, 467

SAPGUIBrowserTxCode, 468
search administration, 183
search capabilities, 234
search engines, 183, 223, 235
secure communication channels (TLS/SSL), 5
secure interoperability, 6
secure proxy server (WebSEAL), 389
Secure Sockets Layer (SSL), 375
security and identity management tools, integrating, 385
security encryption, 5
security features
  authentication, 385
  authorization, 385
security key management utility, 380
security, setting up, 206
server certificates, setting up, 380
server configuration, 326
server in Studio, creating, 325
service description, 430
service method, 331, 338
service objects
  Calendar Service, 233
  DiscoverServer Service, 233
  Domino Service, 233
  PeopleService, 233
  QP Service, 233
Service page, 196
Service Proxy wizard, 448
service() method, 256, 264
service(), 257
services
  collaboration, 16
  Document Manager, 16
  Search, 16
servlet class, 247
servlets, 242, 386
session and connection, 380
session scopes, 266
SessionData object, 301
setAttribute() method, 268
setContentType() method, 263
setData() method, 252
setRenderParameters() method, 265
setSurname(), 174
setView method, 307, 309, 356
shared pages, 137
showTools, 157
Siebel, 392
Simple Object Access Protocol (SOAP), 422
single sign-on (SSO), 225, 385
single sign-on, 454
site
  building, 214
  creating, 206
SiteMinder as external security manager, using, 399
SiteMinder as third-party authentication server, using, 398
SiteMinder Policy Server, components of
  EJB Agent Controller, 397
  Servlet Agent Controller, 397
  Web Agent Controller, 397
SiteMinder support on WP, installing, 397
slave servers, 370
SOAP, 421
software platform, 4
source code, 323
Spreadsheet Editor, 203
SQL query syntax, 237
SQL Server 2000 database users, creating, 76
SSL
  authentication, 380
  data encryption, 380
  data integrity, 380
SSO Architectures, 387
SSO environment, 385
SSO, setting up, 392
state. *See* portlet modes and states, 244
static HTML, 164
std-portlet.tld library, 342
std-portlet.tld tag library, 338

step-by-step debug mode, 330
steps to add jare files, 323
store() method, 252
string constants, 321
String getId() method, 174
string, 384
struts action classes, 355
Struts action mappings, 359
struts configuration files, 357
Struts configuration in portlet configure mode, 353
Struts configuration in portlet edit mode, 353
Struts configuration in portlet view mode, 351
Struts Portlet Framework, 345
Struts Portlet Framework, importing, 346
Struts, 345
struts-config.xml, 351
Studio Advanced Edition, 174
Studio development environment, 275
Studio wizard, 333
styles, modifying, 162
Styles.css, 144
subscription capability, 10
Sun One Directory Server, importing, 375
Supported Clients, 182
Supported Markups, 182
supporting offline access, 479
syndicate, 9
syndication and subscription, 220
system management process, 406
system management products
  CA, 5
  Candle, 5
  Tivoli, 5

T

TAM security, 388
TAM support on WP, installing, 389
TAM TAI, 389
TAM with WP, integrating, 388
taxonomy and categorizing, managing, 236
Taxonomy Manager portlet, 236
TCP/IP, 13
team programming, 280
themes and Skins into WebSphere portal cluster, deploying, 418
themes and skins
  choosing, 146
  default, 144
themes, customizing, 153
themes, skins, and style sheets, migrating, 113
third-party authentication and SSO architectures, 387
third-party authentication server, 387
third-party authentication, 366
Tivoli Access Manager (TAM), 386
Tivoli Directory Server, 372
Tivoli Web Site Analyzer, 87
tool integration platform, 280
ToolBarInclude.jsp, 153
top-level links, adding, 159
toSring() method, 270
trace logging, 330
trace messages, 330
trace string, 330
transcoding plug-ins TT, 484
transcoding plug-ins, types of
  HTML-to-cHTML transcoding plug in, 493
  HTML-to-WML transcoding plug-in, 493
  image transcoding plug-in, 493
  text transcoding plug-in, 493
Transcoding Technology (TT), 473, 483
translation services and unified messaging
  e-mail, 7
  faxes, 7
  voice mail, 7
Trust Association Interceptor (TAI), 388
TT-specific constructs, 490

U

UDDI and tModelKeys, 436
UDDI browser, 425
UDDI registries, 421, 426
UDDI repository, 425
UI-oriented integration, 454
UnavailableException, 339
Unicode, 180
unique identifier (UID), 261
Universal Description Discovery and Integration (UDDI), 422
Unix systems, 43
Update rule, 170
URI format, 359
URI query string, 255
URI string, 360
URL mappings, 180
URLs, 128
urlType attribute, 360
user and groups, 190
user authentication and authorization, 256
user data schema, 165
user identifier, 342
user interaction events, 258
User Manager portlet, 392
User objects, 254
user registry, 365
user-defined taxonomies, advantages of, 236
useReqID, 158
Utilities classes, 358
utility classes, implementing, 321

V

VaultAdapter class, 454
verifyAction method, 307
version control, 6
View classes, implementing, 315
view mode, 313
ViewClassManager getInitialView method, 340
ViewClassManager, 309, 358
ViewInfo, 233
virtual desktops, 7
VisualAge for Java, 174
voice recognition, 7
voice-enabled portlets, 479
void add(Resource resource, ResourceContext context), 175
void delete(Resource resource, ResourceContext context), 175
void put(), 174
void remove(), 174
void syncdelete(Resource resource, ResourceContext context), 175
VoteAction action class, 342
VoteAction, 313
VXML, 479

W

W3C Web Accessibility Initiative, 9
WAR file, exporting, 332
WAS installation, choosing type of, 38
WAS platform, 386
WAS security model, 388
WAS-specific security features, 388
WEA client, 480
WEA, features of, 479
Web application deployment descriptor, 247
Web application development, 280
Web applications, 150, 163
Web Archive (WAR) files, 280
Web browsers
  Internet Explorer, 196
  Netscape Communicator, 196
Web Clipping authentication options, 464
Web Clipping Editor, 197
Web Clipping options for URL rewriting, 465
Web Clipping, 13, 196, 463
Web content management (WCM), 205
Web Content Management products
  Documentum, 9
  Vignette, 9

Web content, 9
Web deployment descriptor, 250, 292, 348
Web descriptor servlet class, 247
Web descriptor, 247, 336
Web Designer, 144
Web page content, customizing, 163
Web page link, 231
Web page structure, 480
Web pages, 7, 12, 162, 171
Web screen scraping technique, 463
Web screens, 393
Web server, 380
Web Service Client Portlet Project wizard, 88
Web Service Description Language (WSDL), 422
Web Service Remote Portlets or WSRP, 420
Web Service, discovering, 427
Web Services and remote portlets, review of, 422
Web Services as 'Back-End' APIs, using, 425
Web Services for Remote Portlets (WSRP), 13, 421, 422
Web services, 5, 421
Web site, 143, 150, 223
Web user interface, 440
WebCache servlet, 480
WebContent folder, 358
WebContent images, 339
WebSEAL Global Sign-On (GSO) junction, 396
WebSEAL junction, 396
WebSEAL proxy, 389
WebSphere Adapter for mySAP.com and WSADIE, 444
WebSphere administrative console, 63
WebSphere Application Server (WAS), 5
WebSphere Application Server installation directory, specifying, 30
WebSphere Application Server Network Deployment, 5, 409
WebSphere application server properties
  WasPassword, 376
  WasUserid, 376
WebSphere Application Server security, 63
WebSphere Application Server, 133, 227, 243
WebSphere applications, 6
WebSphere Business Integration Message Broker, 6
WebSphere Business Portal
  WebSphere Commerce, 7
  WebSphere Everyplace, 7
  WebSphere Portal, 7
  WebSphere Voice, 7
WebSphere Commerce, 7
WebSphere content management, 205
WebSphere Content Publishing, 9
WebSphere Everyplace Access (WEA), 473
WebSphere family
  business integration, 4
  business portals, 4
  foundation and tools, 4
WebSphere family of products
  IBM CrossWorlds, 450
  WebSphere MQ, 450
WebSphere Host Integration family, 6
WebSphere Member Manager, 111
WebSphere Member Services database, 111
WebSphere MQ, 7, 442
WebSphere Network Deployment, 420
WebSphere Personalization server, 11
WebSphere Portal 4.x, 111
WebSphere Portal 5.0 Test Environment, 324
WebSphere Portal 5.0, features of, 21
WebSphere Portal Administrator, defining, 33

WebSphere Portal Application Integrator (WPAI), 452
WebSphere Portal architecture
page aggregator, 15
portlet container, 15
services, 15
WebSphere Portal authentication, 365
WebSphere Portal cluster, horizontal scaling with, 411
WebSphere Portal cluster, implementing, 415
WebSphere Portal cluster, testing, 417
WebSphere Portal cluster, vertical scaling with, 410
WebSphere Portal clustering, 409
WebSphere Portal configuration tasks
create-local-database-db2, 67
database-transfer-export, 67
database-transfer-export-linux, 67
database-transfer-import, 67
enable-security-ldap, 67
httpserver-config, 67
WebSphere Portal configuration templates
config_http.properties, 68
security_active_directory.properties, 68
security_disable.properties, 68
security_domino.properties, 68
security_ibm_dir_server.properties, 68
security_sun_one.properties, 68
transfer_db2.properties, 68
transfer_informix.properties, 68
transfer_oracle.properties, 68
transfer_sqlserver.properties, 68
WebSphere Portal Content Publishing (WPCP), 165
WebSphere Portal content publishing database properties, 81
FeedbackDbHostName, 82
FeedbackDbName, 82
FeedbackDbPassword, 82
FeedbackDbUrl, 82
FeedbackDbUser, 82
FeedbackXDbName, 81
WpcpDbEjbPassword, 82
WpcpDbHostName, 81
WpcpDbName, 81
WpcpDbNode, 81
WpcpDbPassword, 81
WpcpDbPznadminPassword, 82
WpcpDbUrl, 81
WpcpDbUser, 81
WpcpXDbName, 81
WebSphere Portal Custom User Registry, 366
WebSphere Portal customization, 144
WebSphere Portal Document Manager (PDM), 9
WebSphere portal Enterprise Identity Manager, implementing, 371
WebSphere Portal Extend, 25
WebSphere portal for LDAP server, configuring, 375
WebSphere Portal identity manager, 362
WebSphere Portal in clustered environment, configuring, 412
WebSphere Portal InfoCenter, 196, 372
WebSphere Portal infrastructure, 439
WebSphere Portal Installation Directory, specifying, 33
WebSphere Portal Member Services, 14
WebSphere Portal on Linux
installing, 43–44
uninstalling, 60
WebSphere Portal on Windows Server, installation of, 28
WebSphere Portal packaging
IBM DB2 Universal Database Enterprise Server Edition 8.1 with Fix Pack 1, 17
IBM Directory Server version 5.1, 17
IBM Lotus Collaboration Center, 18

IBM Lotus Collaborative Components, 18
IBM Lotus Extended Search 4.0, 18
IBM WebSphere Translation server 5.0, 18
Portal server, 17
Portal toolkit 5.0, 18
QuickPlace, 18
Sametime, 18
Web Content Management, 18
WebSphere Application Server Enterprise 5.0 Fix Pack 1, 17
WebSphere Studio Site Developer 5.0 with Fix Pack 1, 18
WebSphere Portal personalization, 163–64
WebSphere Portal platforms
Databases, 20
Hardware, 20
LDAP Servers, 21
Software, 20
Web Browsers, 21
WebSphere portal properties configuration
PortalAdminGroupId, 377
PortalAdminGroupIDShort, 377
PortalAdminId, 377
PortalAdminShort, 377
WebSphere Portal Server collaborative components
Lotus Collaborative Components, 223
Lotus QuickPlace 3.0, 223
Lotus Sametime 4.0, 223
WebSphere Portal Server Distribution, 175
WebSphere Portal Server nodes, 412
WebSphere Portal Server, 163
WebSphere Portal Server, installation, 25
WebSphere Portal spackaging
WebSphere portal to access LDAP server over SSL, enabling, 380
WebSphere Portal to access remote database, 78
WebSphere portal to cell, adding, 414
WebSphere portal to support SSL, configuring, 382
WebSphere Portal Toolkit V5.0.2 MP, installing, 89
WebSphere Portal V4.x to WebSphere Portal V5.x, migrating, 106
WebSphere Portal, 3, 128, 280
WebSphere portal, configuring, 216
WebSphere Portal, defined roles of
Administrator, 14
Delegator, 14
Editor, 14
Manager, 14
Privileged User, 14
Security Administrator, 14
User, 14
WebSphere Portal, features of, 13
WebSphere Portal, performance of, 12
WebSphere Portlet Catalog, 442
WebSphere Portlet catalog, using, 459
WebSphere Studio Advanced Edition, 173
WebSphere Studio Application Developer (WSAD), 281
WebSphere Studio Application Developer Integration Edition (WSADIE), 443
WebSphere Studio Application Developer, 87, 165
WebSphere Studio Site Developer (WSSD or WebSphere Studio), 275
WebSphere Studio tools
XML editor, 488
XML-to-XML mapping editor, 488
XSL editor, 488
WebSphere Studio WAS runtime environment, updating, 88
WebSphere Studio, 6, 275, 280, 323
WebSphere Test Environment, 87
WebSphere Translation Server, 9
WebSphere Voice, 7
Web-style querying, 237
welcome file, 358

WeSphere Portal Application Integrator, 442
wildcards, 10
Windows 2000 environment, 372
Windows 2000 High Encryption Pack, 375
Windows Active Directory Users and Groups administrative tool, 375
Windows Server, 27
wireless markup language (WML), 475
wireless network, types of
Dial-up networks, 483
IP-based networks, 483
wireless networks, 482
wireless-enabled laptops, 495
wizard tool, 281
WML phones, 484
WML, 136
wmmAttrributeName, 384
Word documents, 9
Word Editor, 203
workflow, creating, 208
Workplace Server, 393
World Clock portlet, 494
World Wide Publishing Service, 27
WP 5.0, 87
WP 5.0.2, 87
WP cliplets, advanced features, 463
WP crawler, 10
WP Credential Vault, 13
WP Extend, 395
WP for WSRP, configuring, 431
WP Page Navigation, 8
WP Portlet API, 16
WP security, 388
WP user provisioning and manipulation, disabling, 391
WP V4.2, 87
WPFamilyName key, 108
WPS tags, 156
WPSconfig, 65
WpsRequestProcessor, 355
writing portlets, rules for, 481
WSDL (Web Service Definition Language), 444
WSRP (Web Service for Remote Portlets), 274
WSRP communications, 431
WSRP consumers, setting up, 434
WSRP layers
SOAP, 436
UDDI, 436
WSDL, 436
WSRP producers, setting up, 433
WSRP scenario, 436
WSRP services, 429
WSRP within WP, using, 429

X

XML access tags, description of
client, 120
component, 120
content-node, 120
credential-segment, 120
credential-slot, 120
event-handler, 120
global-settings, 120
group, 120
markup, 120
portal, 120
portlet, 120
portletinstance, 120
services-settings, 120
servlet, 120
skin, 120
theme, 120
url-mapping-context, 120
User, 120
virtual-resource, 120
web-app, 120
XML Config engine, 119
XML configuration file, 230
XML Configuration Interface for version 5.0, changes in, 123
XML Configuration Interface special properties, 122

XML Configuration Interface
transactional support, 122
XML Configuration Interface utility, working of
Export, 119
Update, 119
XML configuration interface, 282, 333, 430
XML Configuration tool, 118
XML input file, 121
XML pages, 164
XML schema, 333
XML tags, 119
XML transcoding, 488
XPath expression, 490
XSL for adapting content, using, 488
XSL style sheets, 488