How to Best Leverage J2EE, Struts, and ADF in Your Portal

An Oracle White Paper
December 2004
How to Best Leverage J2EE, Struts, and ADF in Your Portal

Introduction ....................................................................................................... 3
Building Portlets - Overview ............................................................................ 4
  The Portlet Development Spectrum.......................................................... 4
Web Clipping ..................................................................................................... 5
OmniPortlet ....................................................................................................... 7
  OmniPortlet Data Sources ......................................................................... 9
    Web Service............................................................................................ 9
    XML..................................................................................................... 10
    SQL..................................................................................................... 10
    Spreadsheet .......................................................................................... 10
    J2EE Connector Architecture (JCA)................................................... 10
    Web Page.............................................................................................. 11
  Filter and Sort the Data......................................................................... 12
  OmniPortlet Layouts............................................................................... 12
    Simple Parameter Form Portlet ........................................................... 14
Java Portlet Wizard ......................................................................................... 15
Building Custom Java Portlets (Oracle APIs) ................................................ 16
  Adding Features to Your Portlets ............................................................ 16
  Struts....................................................................................................... 17
  Declarative Architecture of the PDK....................................................... 19
  Support for J2EE-based Applications...................................................... 21
  Support for Web Services....................................................................... 22
    Using the Built-in Renderers ............................................................... 22
    Calling a Web Service From Your Java Portlet ................................. 23
Building Custom Java Portlets (JSR 168)..................................................... 24
  Testing and Displaying Portlets on WSRP Portals............................... 26
Conclusion........................................................................................................ 26
INTRODUCTION

The typical enterprise relies on a mixture of custom built and pre-built applications to drive the enterprise. These applications are often a mixture of many different technologies. With the popularity of Oracle Portal today, many developers and page designers have been able to come together to build successful enterprise portal solutions. However, many enterprises and ISVs face the challenge of building the same application for each portal vendor. Building the same application multiple times becomes unproductive very quickly. Oracle Portal helps developers solve these issues by providing a complete solution for portlet development, as well as supporting the new OASIS/WSRP and JSR 168 specifications. Developers can continue to work with page designers to leverage Oracle Portal to display content from many different technologies, but now developers can also build interoperable portlets: deploy the same portlet once to any WSRP portal without changing the portlet code.

Oracle Portal enables you to develop most, if not all, of your enterprise portal using its wizard-driven, declarative interface. A key goal of Oracle Portal is to support the development of semantically richer pages and portlets built using the declarative interface and minimizing if not eliminating the need for additional coding. So, enhancements were made to extend the platform of services for both pages and portlets. Declaratively you can build, publish, and manage pages, content, styles, templates, administration and security. Using Oracle Portal, you can also turn portal pages into JSPs, declaratively minimizing much of the manual coding that might be necessary to build a JSP portal.

However, to integrate with other existing applications, leverage the latest technologies, and build against the new portlet standards, the Oracle Portal Developer Kit (PDK) provides the tools for developing custom portlets.

The Oracle Portal Developer Kit (PDK), part of Oracle Portal Developer Services, is the framework for seamless integration into Oracle Portal. Developers use the PDK to develop portlets using the latest technologies and standards. The PDK enables you to develop portlets in many different technologies using Web services, Java (J2EE), XML, Struts, and PL/SQL. The PDK also provides the option to
build these portlets using Oracle's APIs or the new portlet standard APIs – OASIS/WSRP and JSR 168.

This paper discusses when to leverage the declarative features for building portlets and when to build custom portlets. This paper also discusses how to quickly and easily develop J2EE-based portlets, leveraging a wide variety of technologies such as Struts and building portlets using JSR 168, the new portlet standards.

For more information on the PDK or to view the latest information about building portlets with the PDK, visit: Portal Center (http://portalcenter.oracle.com).

**Figure 1: Oracle Portal Center**

**BUILDING PORTLETS - OVERVIEW**

Oracle Portal offers page designers and developers many options to integrate data, applications, and Web-based technologies. The PDK also provides tools for both page designers and developers. When building portlets, users should leverage the existing tools and only build custom portlets when they need to extend the declarative functionality of the portal.

**The Portlet Development Spectrum**

The following portlet development spectrum shows the options that are available to users when building portlets.

**Figure 2: Portlet Development Spectrum**
• Out-of-the-box Portlets – Portlets that are pre-installed and configured with Oracle Portal. These portlets are fully developed, registered, and available in the portlet repository. They include portlets such as Search, Saved Searches, Favorites, My Notifications, and many more.

• Other Sources of Pre-Built Portlets – These portlets include Oracle E-Business Suite, Integration Solutions, 3rd Party Partner Portlets, and Community Portlets. These portlets are available for download from Oracle Portal Center (Knowledge Exchange, Portal Catalog) and available with the E-Business Suite.

• Web Clipping – a generic portlet that allows users to clip all or a portion of an HTML page and use it as a portlet.

• OmniPortlet – a portlet building tools that allows users to build portlets from a variety of data sources. Data sources include Web Services, XML, CSV (spreadsheet), SQL (database), and Web Page. Users can select the layout of this data.

• Portlet Builder – Oracle Portal includes a number of portlet building wizards that are available through the Provider tab in the Navigator.

• Oracle Business Intelligence Tools Portlets – these include portlets that are produced by Oracle Reports and Oracle Discoverer.

• Java Portlet Wizard - Developing Java Portlets has never been easier. You can create, test and deploy Java portlets from within Oracle JDeveloper. This extension includes two wizards - Java Portlet and Oracle PDK Java Portlet. Both wizards guide you through the steps of creating the portlet skeleton and all you need to do then is to implement your business logic. The Java Portlet Wizard allows you to build portlets based on the standards (e.g. WSRP, JSR 168). The Oracle PDK Java Portlet Wizard allows you to build portlets based on the PDK APIs.

• Custom Portlets using the PDK APIs (Oracle & JSR 168) – The PDK provides APIs that enable developers to extend the functionality of Oracle Portal, integrate applications using J2EE-based technologies, integrate PL/SQL applications.

The following sections of this paper discuss in detail Web Clipping, OmniPortlet, the Java Portlet Wizard, and the Building Custom Portlets using the PDK APIs (Oracle & JSR 168).

WEB CLIPPING
During the construction of an enterprise portal, a big part of the job of the page designer is to assemble the information coming from different applications, Web sites (Internet or Intranet). To maintain the relevancy of the information, the content should not be duplicated but ‘portletized’.

The Web Clipping portlet offers an easy and intuitive solution to capture
content and functionality from existing Web sites or applications and present it as a portlet. The page designer and the end user can directly from the browser, without any help from the Information and Technologies (IT) department, create new portlets based on “clippings”. “Clippings” are pieces of existing Web content that can be used in other Web pages, particularly portals. The Web Clipping provider is packaged in the PortalTools application downloadable for free from Oracle PortalStudio (http://portalcenter.oracle.com/portaltools).

The Web Clipping Studio is a tool that the page designer uses to edit the portlet. The studio allows the user to navigate to a Web page similar to a standard browser, divide the page into “clipped” sections and choose the appropriate section for presentation in the portal. Behind the scenes, the studio records, in a secured repository, all the URLs, and “replays” the navigation portlet when the portlet is showed on the portal page.
For example, the basic flow using Web Clipping portlet to capture a page is shown in the next figure.

OMNI.PORTLET

OmniPortlet is a sub-component of Oracle Portal 10g specifically targeted at enabling “power-users” (portal page designers, content contributors, or developers) to quickly and easily publish data from various different data sources using a variety of different layouts, using a Web-based wizard. The OmniPortlet provider is packaged in the PortalTools application downloadable from Oracle PortalStudio (http://portalscenter.oracle.com/portaltools).

An OmniPortlet portlet can be based on almost any kind of data source, such as a XML, Web Service, database (JDBC), spreadsheet (CSV), J2EE Connector Architecture (JCA) or even application data from an existing Web page. To retrieve personalized data, the page designer can define the parameters for each type of data source to filter the result of a query and the credential information used to access secured data. Out-of-the-box, OmniPortlet provides the most common
layout for portlets: tabular, chart, news, bullet and form. The following is an example of an Oracle Portal page with different OmniPortlet layouts:

![OmniPortlets on a Portal Page](image)

**Figure 4: OmniPortlets on a Portal Page**

The OmniPortlet provides a unique way to build a new portlet from a Web-based wizard, which directs the user to:

1. Select of the type of data
2. Configure of the data source
3. Set filtering and sorting criteria (optional)
4. Configure the view and Select the type of layout
5. Configure the layout
OmniPortlet Data Sources

OmniPortlet provides out-of-the-box different data sources. For each data source, the OmniPortlet Wizard allows the user to enter specific information to configure it. One of the most important feature of a portal is to retrieve and show personalized content for every user. OmniPortlet supports parameterization and security that guarantees to retrieve personalized data to the user’s portlet on the page. OmniPortlet data sources are pre-configured to consume portal page parameters; this feature allows the page designer to render contextual data from any data sources.

Web Service

The OmniPortlet Web Service data source allows the user to build a portlet from a WSDL (Web Service Description Language) URL that describes the Web Service. The data source introspects the WSDL and provides an interface to specify the method and the parameter to use. As with the XML data source, data is expected in a rowset/row format. If it is not returned in this format, users may specify an XSL file to format it. Also similar to the XML data source, column names are determined by introspecting the WSDL, but users can specify an XML schema to describe the returning data set. If the Web Service is protected using HTTP authentication, the page designer can define the connection information.
XML
Although still a relatively new form of storing data, XML is increasingly used as a method of providing controlled access to data sets over the intranet, and even the Internet. The OmniPortlet data source allows the user to specify the URL of an XML data source.

The XML should be formatted in a rowset/row format. If the XML feed does not exactly match this format, the user can specify the URL to an XML style sheet (XSL) to format the data as it is being fetched. The OmniPortlet will also introspect the XML to determine the column names, which will then be used to define the layout. If users wish to specify this information themselves, they can supply a URL to an XML schema that describes the data. If the XML document is protected using HTTP authentication, the page designer can define the connection information.

SQL
The relational database is currently the most common place to store data. OmniPortlet offers an easy way to publish data from a database using the SQL data source. The data source uses standard JDBC drivers, and provides out-of-the-box access to Oracle and any ODBC database.

The user can enter the database connection information or reuse saved connection information to access to a specific database. Once the connection information is correctly entered, any SQL statement can be written and parameterized.

Spreadsheet
Spreadsheets are a very common method of storing small data sets. Users often use spreadsheets to meet their own personal publishing requirements, but then decide to share this information across a broader audience. The OmniPortlet makes it easy to share spreadsheets by supporting comma-separated values (CSV) as a data source. If the spreadsheet is protected using HTTP authentication, the page designer can define the connection information.

J2EE Connector Architecture (JCA)
J2EE™ Connector Architecture (JCA) 1.0 provides a mechanism to manipulate enterprise data such as that held in ERP systems (Oracle Financials, SAP, PeopleSoft, etc.). Using JCA to access an ERP is similar to using JDBC (Java Database Connectivity) to access a database. Due to the current limitations of the JCA 1.0 standard, it is not possible to provide a declarative user experience for specification of the JCA data for the portlet (which is why it is not displayed in the OmniPortlet Wizard). JCA 1.0 does not support object introspection (querying of a data source to determine the objects available). This feature will be available as part of a revision of the standard in the very near future. At this point, the JCA adapter with the OmniPortlet will be upgraded to be exposed in the user interface.
Use of this adapter with the OmniPortlet may be achieved today by editing the metadata definition of the portlet (held in the `provider.xml` file).

**Web Page**

Since many applications are now Web-based, information needed by the end user is often already accessible from the browser. The challenge is to leverage the existing application and repurpose this application into a portal.

The first answer to this challenge is the Web Clipping portlet, released in November 2002. Web Clippings are pieces of existing Web content that can be published as a portlet. For more information about the Web Clipping portlet, visit the [OmniPortlet and WebClipping](http://portalcenter.oracle.com/portaltools) page on Portal Center.

The Web Page data source enables the user to capture the data of a Web application and easily integrate it into a portal page. This data source uses the Web Clipping Studio to “surf” the application and while retaining the business logic, extract the data and publish it into a portlet.

![Selected Columns](image)

**Figure 6: Web Clipping Studio Page in the Web Page Data Source**

![Naming of the data](image)
Filter and Sort the Data

Independently of the type of data source to build the portlet, the user can further refine the retrieved data by filtering and sorting it. OmniPortlet supports filtering of data in two areas:

- In the data source itself, by specifying what data is retrieved from the actual data source. This is specified on the Source page.
- In the middle tier. This is specified on the Filter page.

It is always more efficient to filter and sort the data at the data source, since users can streamline the process by retrieving only the necessary data. The best practice is to only use the middle tier filtering when the data source does not support any filtering or sorting capability; for example, a spreadsheet.

OmniPortlet Layouts

Out-of-the-box OmniPortlet provide the most common style of layout for portlets:

- Tabular
- Chart
- News
The OmniPortlet layouts are based on the Oracle Portal styles, which means that all OmniPortlet instances will be consistent with the rest of the page. If the page designer changes the style at the template level, OmniPortlet will automatically reflect the change.

OmniPortlet maintains a separation between data and layout. That is, users may select a data source and select a layout, then may later change the layout without having to rebuild the portlet. For example, data may be rendered initially as a tabular layout and then changed to a chart layout, simply by re-entering the wizard and changing the layout type. This flexibility means that users can easily choose the method of publishing their data that is right for them.

Like the data sources, OmniPortlet provides specific configuration screens for each of the layouts:

- For the tabular layout: select whether to see alternating colors for each row.
- For the news layout: select whether to use an image (typically the image of the news provider), or display the news in an automatic scrolling zone.
- For the bullet layout: select the bullet format (disk, number, letter, Roman numerals, etc.).
- For the form layout: select whether the form fields are horizontal or vertical.
- For the chart layout: select the type of chart (e.g., bar, pie, line) and if the data is already grouped (if so, by what column).
OmniPortlet layouts are preconfigured to raise portal events and send as event parameters any value from the set of data that is rendered in the portlet.

**Simple Parameter Form Portlet**

In addition, the OmniPortlet Web provider contains another portlet: the Simple Parameter Form Portlet. This portlet allows the page designer to create simple parameter forms where end users may enter data or pick from a list of predefined options, then easily set Oracle Portal page parameters and events.
JAVA PORTLET WIZARD

The Oracle Portal Developer Kit (PDK) is a powerful environment to create portlets, but often building portlets involves learning new APIs, architecture, writing a lot of code, and manually deploying that portlet to an application server. The Java Portlet Wizard reduces the amount of work needed to wrap your business logic so you can concentrate on implementing the necessary functionality. The Java Portlet Wizard allows developers to build J2EE-based portlets using a wizard-based user interface directly from within Oracle JDeveloper. Using the wizard, developers can choose to build portlets based on the new portlet standard APIs (OASIS/WSRP & JSR 168) or the existing Oracle PDK APIs (JPDK). Oracle’s Support for OASIS/WSRP and JSR 168 is discussed later in this paper. Some of the benefits of the wizard include:

- Easily create, debug, and deploy portlets directly from within Oracle JDeveloper.
- Create a portlet skeleton through a wizard-based user interface.
- Create a portlet with most show modes and features directly from wizard.
- No manual deployment or import statements lowers the bar so developers can quickly begin building portlets.
- Reduces the amount of work to wrap business logic.
BUILDING CUSTOM JAVA PORTLETS (ORACLE APIS)

The Oracle Portal Developer Kit (PDK) is J2EE-based and includes APIs to build custom portlets and leverage technologies such as J2EE, Web Service, Struts, LDAP, and much more. The goal of the PDK is to provide developers as much control as they need, but still reduce the amount of actual coding needed by providing a declarative mechanism that lists and defines their portlets.

Adding Features to Your Portlets

The PDK APIs provide a complete list of functionality and features that allow your portlet to be as powerful as any other application. The following are a few of the features available to you when building portlets:

- Turn existing components into portlets.
- Java/J2EE (e.g., JSPs, Java Servlets, EJBs)
- XML data sources (e.g., XSL, RSS)
- Web Service (e.g., Doc, RPC)
- Build your portlet user interface
- Browser technologies (e.g., HTML, JavaScript, CSS, XSL, Flash, Java Applets)
- Error Handling
- URL rewriting
- Internationalization (e.g., NLS, language support, character-set information)
- Session management
• Information storage
  • End user personalization (file, database, preference store)
  • Session storage
• Parameters and events
• Performance
  • Scalability (e.g., load balancing, failover)
  • Caching (invalidation-based, validation-based, expiry-based)
• Security
  • Portlet security (e.g., partner app, external app, portal app, mod_sso)
  • Application security (e.g., JAAS/JAZN)
  • LDAP security

The following sections discuss the latest features within the PDK that are available to users.

Struts

Application developers are always looking for frameworks that allow them to develop faster and cleaner to provide better applications to the end user. The Java Web application development did not have any standard available to developers using a proper architecture that allowed fast development and easy maintenance. Apache Struts became the de-facto standard because many projects starting to use it as the basic framework to build Web applications.

The Jakarta Project's Struts framework from Apache Software Organization is an open source framework for building Web applications that integrates with standard technologies, such as Java Servlets, JavaBeans, and JavaServer Pages (JSPs). Struts offers many benefits to the Web application developer, including Model 2 implementation of Model-View-Controller (MVC) design patterns in JSP Web applications.

Introduction to the Model-View-Controller pattern

The MVC Model 2 paradigm applied to Web applications lets you separate display code (for example, HTML and tag libraries) from flow control logic (action classes). Following is a brief overview of the MVC Model 2 design pattern. For complete information about how Struts implements the MVC design patterns, see the Introduction to the Struts User’s Guide on the Jakarta Project’s Web site: http://jakarta.apache.org/struts/userGuide/index.html.

• The Model portion of the MVC contains the core of the application functionality, persistence and business logic. Struts applications can use any mechanism of persistence for the model EJB, BC4J or simple Java classes.
• The **View** portion of the MVC provides the presentation of the model. In a Struts application, the view is based on JavaServer Pages and uses JSP tags and Expression Language (EL) to access the model.

• The **Controller** portion of the MVC receives the request from the client, typically a browser, and applies some application-specific business logic, such as data validation and application page flow. In a Struts application, the controller is implemented as a servlet, the `ActionServlet`.

**Publish Struts Applications as a Portlet**

If your team uses Struts as the framework to develop applications, these skills should be leveraged to publish some part of the application as portlets.

The Oracle PDK provides new APIs that allow the developer to easily publish part of Struts applications as portlets without changing the application code. The approach that Oracle AS Portal uses to do this is based on two components:

• A portal “view” of the application

• A set of tags that encapsulate the portal context

As an example, suppose you have an application that provides the capability to reply to a survey:

In the context of the stand-alone application, the form appears in a complete browser, with the application look and feel element such as header, footer and menus. This application view is a complete JSP that contains the complete HTML that use the application-specific style (CSS). When the user submits the form, the complete page is replaced by the survey result in the context of the application (same header and footer).

![Figure 11: “Stand-Alone” Struts Application](image-url)
In the context of the same form published as a portlet, the form appears in the middle of other portlets. All the other application components such as the header and footer should not appear. For the portlet, the JSP contains only the HTML that builds the form, and should use the style provided by portal. This JSP is the “portal view” of the application. The other important item is that when the user submit the form inside portal, the feedback screen should be the portlet itself and not a complete new page.

Figure 12: Survey Application in a Portlet

To facilitate the development of portlets for the Struts developer, Oracle PDK provides new JSP tags. These tags are extensions of the Struts tags and encapsulate the portal logic to keep the context of the page when a user submits an action or clicks a link.

Declarative Architecture of the PDK

When building portlets using the PDK APIs, users only need to worry about the content that will render in their portlet and not how the portlet will communicate with the portal. For example, consider a JSP that displays welcome information to a logged in user. This portlet grabs the name of the logged in user and displays information appropriate to the user’s group. The steps to display this JSP as a portlet include the following:

- Use the PDK APIs to check the user’s group from the LDAP server.
- Place the JSP in the J2EE-based EAR file directory structure.
- From within the provider definition file (e.g. provider.xml), specify a relative URI to the resource. The URI should be specified from the root directory of the WAR file in which the provider is deployed.
• The provider definition file represents your provider, so register the provider with Oracle Portal.

• Display the portlet on a page.

The provider definition file is able to easily include your Web technologies because of the built-in renderers within the PDK that automatically know how to handle each technology. The built-in renderers within the PDK are:

• ResourceRenderer – a pluggable renderer that can be used to render any type of resource (static HTML page, JSP, Servlet, XML file, etc.).

• FileRenderer - a pluggable renderer that can be used to render static content such as HTML or XML files. Even though static content can be rendered using ResourceRenderer, FileRenderer adds two important functions that are not available in ResourceRenderer. Those functions are:
  • The ability to read static content in any character set supported by Java. This allows you to correctly render files that were written using multi-byte character sets such as UTF-8 and Shift-JIS (a common Japanese character set).
  • The ability to cache the file in memory so subsequent requests for the same content can be served very quickly.

• DocWebServiceRenderer – a pluggable renderer that can be used to render document-oriented (Doc) style Web Services.

• RPCWebServiceRenderer – a pluggable renderer that can be used to render procedure-oriented (RPC) style Web Services.

**Note:** support for Web Services is discussed later in this paper.

The provider definition is only one piece of the architecture that makes up the PDK. The PDK API framework can be logically split into the following areas:

• Provider Adapter - insulates the developer from the HTTP syntax defined by Oracle Portal for communicating to Web providers. It translates the information passed between Oracle Portal and your Web provider. Without an adapter, your provider would not only manage portlets, but would also be required to communicate this information directly to Oracle Portal in the language that the portal understands. The adapter eliminates the need for your Web provider to understand the portal language and vice-versa.

• Provider Interface - defines the APIs (functions) required by your Java implementation to integrate with the Provider Adapter. The Provider Adapter receives messages from the portal, translates them into calls to the Provider Interface and then takes the provider's response and translates the response into a format that the portal can understand. Using Java terminology, the Provider Interface is defined by a set of abstract classes. These classes define the methods your provider needs to implement, and in many cases provide, a standard implementation.

• Provider Runtime - provides a base implementation that follows the specification of the Web Provider Interfaces. The Provider Runtime includes a set of default classes that implement each one of the Web
Provider Interfaces and allow you to leverage the rendering, personalization and security frameworks provided with the PDK. These classes and the associated frameworks simplify the development of a Provider by implementing the common functions associated with a particular Portal request and providing a declarative mechanism for configuring the provider. Using the Provider Runtime, you can focus your development efforts on the portlets themselves rather than the infrastructure needed to communicate with the portal. If the standard behavior of the Provider Runtime does not meet your requirements, you can easily extend or override specific behaviors. The Provider Runtime includes a class file that corresponds to each of the Web Provider interfaces. The provider definition file (e.g. provider.xml) can be considered part of the Provider Runtime.

Provider Utilities - provides methods for simplifying the rendering of portlets. The utilities include methods for constructing valid links (hrefs), rendering the portlet's container (including title bar), rendering HTML forms that work within a portal page, and supporting portlet caching.

Support for J2EE-based Applications
Oracle Portal is 100% J2EE compliant and the PDK provides complete support for building J2EE portlets. Oracle Portal runs on Oracle Application Server (OracleAS) which provides a fast, lightweight, highly scalable, and easy-to-use J2EE environment, OracleAS Containers for J2EE (OC4J). You can leverage a number of industry-wide standards while deploying your portlets to OC4J. OC4J is written entirely in Java and executes on the standard Java Development Kit (JDK) virtual machine. It provides a complete J2EE environment that includes a JSP Translator, a Java Servlet engine and an Enterprise JavaBeans (EJB) container.

The following runtime containers and APIs are provided:

- Java Servlets 2.3
- JSP (JavaServer Pages) 1.2
- EJB (Enterprise JavaBeans) 2.0
- JNDI (Java Naming and Directory Interface) 1.2 - ability to set deployment property information for your provider directly from the web.xml using the JNDI environment variables. Review, Using JNDI with the PDK Release 2.
- JTA (Java Transaction API) 1.0.1
- JMS (Java Messaging Service) 1.0.1
- JDBC (Java Database Connectivity) 2.0
- JavaMail 1.1.2
- JAF (Java Activation Framework) 1.0
- JCA (Java Connector Architecture) 1.0

In addition to the fully implemented J2EE 1.3 containers and APIs, Oracle Portal is fully J2EE 1.3 compliant and has been certified with JDK 1.4.
Support for Web Services
The PDK provides an out-of-the-box capability for invoking Web Services. It provides an abstraction to handle communication with Web Services. It allows you to declaratively invoke Web Services with the built-in Web Services Renderer classes. Using the PDK, developers can control their Web Services by calling a Web Service directly from their Java applications. This section will describe both scenarios.

Using the Built-in Renderers
With the PDK, you can invoke both procedure-oriented RPC style Web Services as well as document-oriented Doc style Web Services. For Doc style services, the SOAP Body carries a generic XML document, whereas in RPC style services, there is an explicit concept of "method invocation". The PDK supports both these styles with specialized renderer classes.

To build a portlet using a Web Service, you need to perform three main steps:

- Point the PDK framework to your Web Service by filling in the appropriate tags in the provider.xml file. You can fill in portlet information, and the Web Service related details using the built-in renderers.
- Register your provider with Oracle AS Portal.
- Display the portlet on a page.

For building your own Web Service, you can use Oracle JDeveloper, which has a great feature set especially for Web Services. The Web Services Description Language is an XML vocabulary that provides a standard way of describing Web Service details such as the service location, method parameters and data types. With Oracle JDeveloper, you can use features such as the client stub generation wizard that takes a WSDL URL and generates the stub class automatically. This stub class provides a wrapper for communication between the portal and the Web Service.

For an RPC style Web Service, you need you need to use the built-in RPCWebServiceRenderer class, and fill in relevant tags in the provider.xml.

- You can set up your Web Service to use the generated stub classes that facilitate communication between the Portal and the Web Service.
- You can also have Web Service portlets communicate with other portlets, and pass parameters to Web Service portlets with declarative parameter bindings.
- You can also control how the portlet displays the data from a Web Service by passing the data through XSL style sheets.

For a Doc style Web Service, you need to use the built-in DocWebServiceRenderer class.

- You can set up your Web Service to use the WSDL details such as endPointURL and SOAPAction that facilitate communication automatically. You can plug in such information from the WSDL doc of the Web Service, directly into the provider.xml.
- You can point to the built-in classes provided by the PDK to handle literal XML.
You can also have Web Service portlets communicate with other portlets, and pass parameters to Web Service portlets with declarative parameter bindings.

You can also control how the portlet displays the data from a Web Service by passing the data through XSL style sheets. If you do not provide a style sheet, a default one is generated for you by PDK-Java.

Figure 14: Web Services Sample from PDK download

Calling a Web Service From Your Java Portlet

Developers can have unlimited control from their Web Service portlets using the PDK. Both OmniPortlet and the PDK offer declarative ways to integrate Web Services as portlets into Oracle Portal. Even though OmniPortlet offers a lot of functionality and features for the Web Services data source, developers may need to have complete control over the way a Web Service is rendered. Using Oracle JDeveloper and the PDK, developers can make a call directly from their Java portlet to a Web Service and completely control the way that Web Service is rendered within the portlet. The simple steps to call a Web Service from a Java application are:

- Create a Stub of an RPC-based Web Service.
• Use the Java Portlet Wizard to create a Java portlet. This Java portlet will be a skeleton that includes the necessary import files and deployed to the recommended Web provider structure.

• Call the Stub directly from the Java portlet.

• Make any changes needed to control the display of the Web Service.

BUILDING CUSTOM JAVA PORTLETS (JSR 168)

Until now, users have had to develop portlets using proprietary APIs for a single portal platform and often are faced with a limited number of available portlets from a particular portal vendor. All this changes with the introduction of the Web Services for Remote Portlets (WSRP) and Java Specification Request (JSR 168) standards. These two standards enable development of portlets that can be interoperable between different portal products and therefore increases the availability of portlets to an organization. This, in turn, can dramatically increase an organization’s productivity when building enterprise portals.

WSRP is a Web Service standard that allows the plug-n-play of visual, user-facing Web Services with portals or other intermediary Web applications. Being a standard, WSRP enables interoperability between a standards-enabled container and any WSRP portal. WSRP defines:

• WSDL interface description for invocation of WSRP services.

• Markup Fragment Rules for markup emitted by WSRP services.
• The method to Publish, Find, Bind WSRP services and metadata.

JSR 168 is a specification that defines a set of APIs to enable interoperability between portlets and portals, addressing the areas of aggregation, personalization, presentation, and security. JSR 168 defines:

• The portlet API (Portlet container), which provides a runtime environment to invoke portlets.
• A URL-rewriting mechanism for creating user interaction within a portlet container.
• Handling of the security and personalization of portlets.

So, what is the relationship between WSRP and JSR 168? WSRP is a communication protocol between portal servers and portlet containers, while JSR 168 is a Java API for portlets to work with WSRP portals. This API enables developers to integrate their applications from any internal/external content as portlets with WSRP portals. Building portal pages becomes as simple as selecting portlets from the Oracle Portal repository. The illustration below shows the architecture of the WSRP specification. Notice that portlets are using only the WSRP Protocol.

![Figure 16: Portal with only WSRP connections to a JSR 168 Portlet Container](image)

Since Oracle’s existing architecture is so similar to the one specified by the WSRP committee, Oracle Portal is able to support communication between our portal and both the new JSR 168 APIs as well as our existing PDK APIs (JPDK). The illustration below shows the architecture of Oracle’s WSRP portal. Notice that the JSR 168 portlet container uses the WSRP protocol for communication and the JPDK portlet container uses Oracle’s proprietary protocol for communication.
Testing and Displaying Portlets on WSRP Portals

Oracle’s WSRP-enabled portal will be available in a later production release, but Oracle is offering a hosted version of the WSRP portal that allows vendors, developers, and partners who are building WSRP producers to test that their implementations run in Oracle’s environment. This hosted portal is called Oracle Portal Verification Service and it provides an environment for registering your WSRP producer and adding its portlets to a portal page.

The service also contains an Oracle WSRP producer containing sample portlets implemented using Oracle’s standards-based PDK. It demonstrates the ability of OracleAS to utilize portlets built using the WSRP and JSR 168 standards.

CONCLUSION

This paper provides an overview of developing portlets using the PDK. For more
information, go to Oracle Portal Center (http://portalcenter.oracle.com) and click on Portlet Development.