Introduction to JSR 168—The Java Portlet Specification

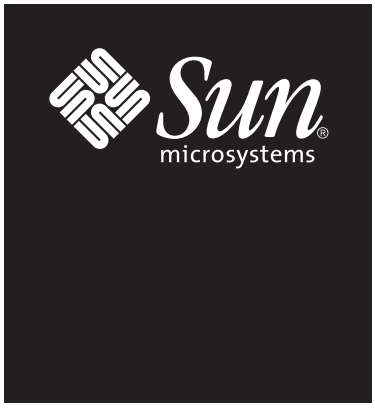


Table of Contents

Introduction to JSR 168—The Java Portlet Specification

Introduction

The *Java™ Specification Request 168 Portlet Specification (JSR 168)* standardizes how components for portal servers are to be developed. This standard has industry backing from major portal server vendors.

An example, the Weather Portlet, is provided to demonstrate the key functionality offered by the Portlet API: action request handling, render request handling, render parameters, dispatching to JavaServer Pages™ (JSP™) technology, portlet tag library, portlet URLs, portlet modes, portlet cache and portlet preferences.

This white paper is intended for the following audiences:

- Developers who write applications for portal servers.
- Employees of companies that deliver portal components or add-ons (i.e. portal tool vendors).
- Information Technology managers who use or are evaluating portal servers.

Advantages of The Portlet Specification

The specification defines a common Portlet API and infrastructure that provides facilities for personalization, presentation, and security. Portlets using this API and adhering to the specification will be product agnostic, and may be deployed to any portal product that conforms to the specification.

IT Managers benefit from the ability to support multiple portal products, thus accommodating the unique business needs of various departments and audiences. The compliant portlets can be deployed to all compliant portal frameworks without extensive engineering changes.

For developers, the specification offers code reusability. Developers who want to portal enable their applications can create and maintain one set of JSR 168 compliant portlets. These portlets can be run on any JSR 168 Portlet Specification compliant portal server with few, if any, modifications.

Because this API is specifically designed for portlet creation, developers will benefit from extra functionality beyond the standard servlet functionality. The next section, “What is the Portlet Specification,” discusses these benefits in detail.

This standard also creates a market opportunity for portal tools: IDE’s, test tools, performance measurement tools and reporting interfaces can be marketed to a wider audience. Portal tool companies can support all compliant portal servers without additional engineering and support costs.

What is The Portlet Specification?

The Portlet Specification addresses the following topics, all covered in this section:

- The portlet container contract and portlet life cycle management
- The definition of window states and portlet modes
- Portlet preferences management
- User information
- Packaging and deployment
- Security
- JSP tags to aid portlet development

Container Contract

Much like Java™ Servlet extensions run inside a servlet container, *The Portlet Specification* defines a portlet container that manages portlets. A contract is defined for the container to call methods during a portlet's life cycle. The portlet developer can implement these methods to provide the desired functionality.

The life cycle methods called directly by the container are:

init()	Called when the portlet is instantiated by the container. Intended to contain logic that prepares the portlet to serve requests.
destroy()	Called when the container destroys the portlet. Intended to contain logic that cleans up when portlet is no longer needed or the server shuts down.
processAction()	Called after the user submits changes to a portlet. Intended to process input from a user action.
render()	Called whenever the portlet is redrawn by the desktop.

In addition to the methods above, which are called directly by the container, a `GenericPortlet` class is provided that implements the `render()` method and delegates the call to more specific methods to display the portlet based on its mode. Developers can extend `GenericPortlet` and implement as many of these specialized render methods as are necessary for their portlet. These methods are:

doView()	Called by render() when the portlet is in View mode. Intended to contain logic that displays the View page for the portlet.
doEdit()	Called by render() when the portlet is in Edit mode. Intended to contain logic that displays the edit page for the portlet.
doHelp()	Called by render() when the portlet is in Help mode. Intended to contain logic that displays the help page for the portlet.

The `processAction()`, `render()`, and specialized methods called by `render()` accept portlet requests and response objects similar to those passed to the `Servlet service()` method. Using these objects, the portlet class can do the following:

- Maintain state and communicate with other portlets, servlets, and JSP's using the portlet session facilities.
- Receive user input from forms displayed in the portlet.
- Create the content for display by the Portal desktop that is sent back to the client via the response object.
- Query portal and portlet state information.

Portlet Mode and Window State

There are two main state items managed by the container for each portlet: portlet mode and window state. This is one of the differences between portlets and servlets.

Each portlet has a current mode, which indicates the function the portlet is performing. As implied by the methods mentioned above, the modes defined in *The Portlet Specification* are *View*, *Edit*, and *Help*. These modes are used by the default render method to decide which lower level display method to call. The mode can also be queried anywhere in the portlet code using the Portlet API to conditionally behave in a manner that is dependent on the current mode.

Window state indicates the amount of portal page space that will be assigned to a portlet. The portlet can use this information to decide how much information to render. The three window states for each portlet are *minimized*, *maximized*, or *normal*.

Portlet Preferences

Portlets are often configured to provide a custom view or behavior for different users. These configurations are represented as a persistent set of name-value pairs and are referred to as portlet preferences. In *The Portlet Specification*, the portlet container is responsible for retrieving and storing these preferences through the `PortletPreferences` interface via the `getValues()` and `setValues()` methods respectively.

Portlets have access to the `PortletPreferences` object when processing requests, but may only modify preference attributes during a `processAction` invocation. Prior to the end of the `processAction` method, the `store`() method must be invoked for changes to be permanent.

To validate preference values, implement the `PreferencesValidator` class. During runtime, the `store()` method invokes the `validate` method of the validator before writing changes to the data store.

User Information

The Portlet Specification provides a mechanism for portlets to access user information—such as name, email, phone, and address—in the portlet application's deployment descriptor, and then access them through an unmodifiable Map object. The Map object can be retrieved through the `USER_INFO` constant defined in the Request interface. When the Java standard for user information is defined, this mechanism will be discontinued in favor of it.

Packaging and Deployment

The Portlet Specification specifies the packaging and deployment of portlets as part of standard Web Application Archive (WAR) files that may contain other web components, such as JSPs and servlets.

In addition to the `web.xml` deployment descriptor now found in WAR files, there is an additional `portlet.xml` descriptor that defines all portlets and portlet-related configurations. An XML schema included in *The Portlet Specification* defines the standard elements for portlet configuration stored in the `portlet.xml` file.

It is up to each portal server vendor to provide a tool that parses the portlet descriptor file and deploys the portlet. It is likely that these tools will exist in GUI form as well as in command line form. Developers will be able to integrate the command line deployment tools with the Apache Ant tool. This opens up additional development environment possibilities as most development tools provide an interface for Apache Ant. Sun plans to include a deployment tool with Sun™ ONE Portal Server that automatically creates a display file in the portal directory and shows the portlet in the administration application.

Security

The Portlet Specification includes several features to help developers create secure portlets.

In the portlet deployment descriptor, a flag can be set to restrict the portlet to running only over HTTPS. This is appropriate for portlets that contain confidential information that should be encrypted when sent over the network.

In addition, the Portlet API includes authentication facilities for querying user and role information. This allows developers to programmatically control security by introducing business logic based on a user's information and role. Similar to other Java 2™ Platform, Enterprise Edition (J2EE™) technologies, *The Portlet Specification* does not address how users and roles are implemented, leaving it to each vendor to apply its own solution. It is likely that the implementations will range from simple file-based user configuration to robust solutions leveraging enterprise directory servers.

The security features addressed by the first version of *The Portlet Specification* are adequate to ensure that portlets can be used for any purpose, regardless of data sensitivity. Nevertheless, future versions will likely have expanded security and authentication features.

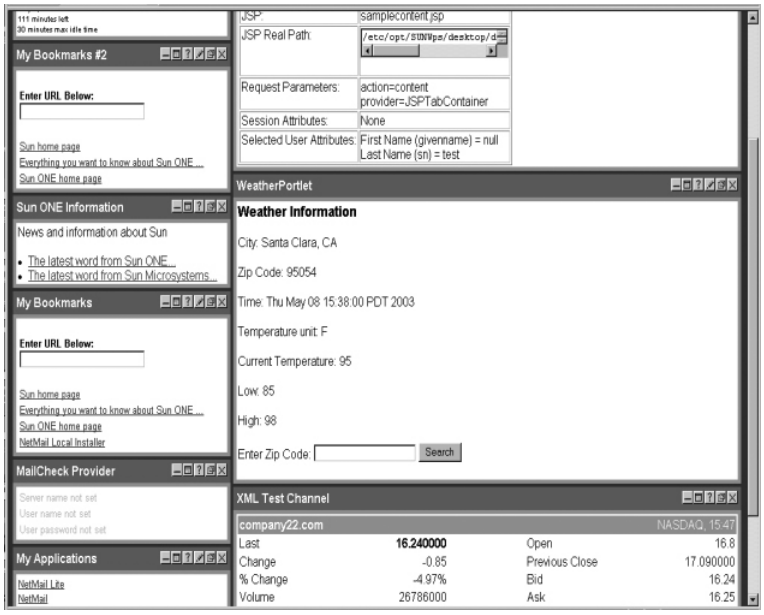
JSP Tag Library

A JSP Tag library is included to help display portlet pages with JSP technology. For example, a custom JSP tag automatically declares the portlet request and response objects so they can be used within the JSP. Another JSP tag helps construct URL's that refer back to the portlet.

The Weather Portlet

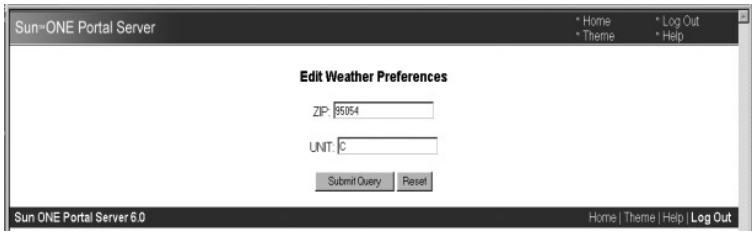
The Weather Portlet is a sample portlet for displaying the temperature on a portal page. This section includes detailed examination of the code, to demonstrate the key Portlet API functionality.

Figure 1. The Weather Portlet



The Weather portlet displays temperature information for a given zip code. Users may also query temperatures for alternate zip codes without any customization.

Figure 2. Editing Preferences



Users may customize their default zip code and the units for displaying the temperature.

This sample portlet retrieves weather information from a Web Service using JAX-RPC. For brevity, the example focuses on the portlet code; the JAX-RPC classes are not fully described.

The `init()` Method

The first method of the portlet that is called is the init() method. The init() method is called to initialize the portlet before the portal sends any request to it.

```
File: WeatherPortlet.java

public class WeatherPortlet extends GenericPortlet {
  private WeatherForecasterService _weatherService;

  public void init(PortletConfig config)
    throws PortletException, UnavailableException {

    super.init(config);

    String weatherWSURL = config.getInitParameter("weather.url");
    if (weatherWSURL==null) {
      throw new UnavailableException(
        "Weather Web Service URL not found",-1);
    }
    try {
      Context ctx = new InitialContext();
      _weatherService = (WeatherForecasterService)
        ctx.lookup("java:/comp/env/WeatherForecaster");
    }
    catch (Exception ex) {
      throw new UnavailableException(
        "Weather Service can not be initialized");
    }
  }

  ...
}
```

In the `init()` method, the Weather Portlet obtains the weather Web Service URL from an init parameter and initializes the JAX-RPC service object to it. If the init parameter is missing, or the JAX-RPC service cannot be initialized, the method throws an `UnavailableException` and the portlet is not available to serve requests.

The `doView()` Method

Assuming the portlet initializes successfully, when users go to the portal (and have the Weather portlet in their page) the `doView()` method is invoked to render the content of the portlet. This method is called when the portlet is in View mode.

```
File: WeatherPortlet.java

public class WeatherPortlet extends GenericPortlet {
  ...

  public void doView(RenderRequest rReq, RenderResponse rRes)
    throws PortletException, IOException {

    rRes.setContentType("text/html");

    PortletPreferences prefs = rReq.getPreferences();
    String zip = rReq.getParameter("zip");
    if (zip==null) {
      zip = prefs.getValue("zip","10000");
    }
    String unit = prefs.getValue("unit","F");

    try {
      WeatherForecaster weatherForecaster =
        _weatherService.getWeatherForecasterPort();
      Weather weather =
        weatherForecaster.getWeather(zip,unit.charAt(0));
      weatherForecaster = null;
      rReq.setAttribute("weather",weather);
      PortletRequestDispatcher rd =
        getPortletContext().getRequestDispatcher(
          "/weather/weatherView.jsp");
      rd.include(rReq,rRes);
    }
    catch (Exception ex) {
      rRes.setProperty("expiration-cache","0");
      PortletRequestDispatcher rd =
        getPortletContext().getRequestDispatcher(
          "/weather/weatherServiceUnavailable.html");
      rd.include(rReq,rRes);
    }

  ...
}
```

First, the `doView` method sets the content type for the response. It then checks for any render parameters with a zip code to be used. If `zip` is not supplied as a parameter, then the value stored in the Portlet Preferences is used. Unit is retrieved from the portlet preference. Once the zip code and the unit have been determined, it invokes the Weather Web service. The `Weather` object is then stored in the request attributes under the key “weather”. A request

dispatcher to `/weather/weatherView.jsp` is obtained and the request is dispatched to it. If the invocation of the Weather Web service fails, the portlet renders an error page through the request dispatcher. Before invoking the dispatch, the cache is set to 0 (no cache) to ensure that, upon the next render request, the portlet will invoke the Weather Web service again.

```
File: weatherView.jsp

<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>

<h3>Weather Information</h3>
<p>
City: <jsp:getProperty name="weather" property="city"/>
<p>
Zip Code: <jsp:getProperty name="weather" property="zip"/>
<p>
Time: <jsp:getProperty name="weather" property="time"/>
<p>
Temperature unit: <jsp:getProperty name="weather" property="unit"/>
<p>
Current Temperature: <jsp:getProperty name="weather" property="currentTemp"/>
<p>
Low: <jsp:getProperty name="weather" property="minTemp"/>
<p>
High: <jsp:getProperty name="weather" property="maxTemp"/>
<p>
<form method="post" action="<portlet:actionURL/>">
 Enter Zip Code:
 <input type="text" name="zip" value="">
 <input type="submit" name="submit" value="Search">
</form>
<p>
```

`weatherView.jsp` first defines the portlet tag library. Then, using the JSP native tag library, it creates content with weather information by querying the Weather object stored in the request attributes. Finally, it creates a form that allows users to enter a zip code other than the default. This form uses the `actionURL` tag from the portlet tag library. Portlets must use `PortletURL's actionURL` portlet tag or `renderURL` portlet tag, to create links targeted to them. This is required because portlets are not bound to a URL, but rather invoked through the portal.

The `doEdit()` Method

When the user clicks on the portlet title bar's "edit" button, it changes the portlet mode to EDIT and invokes the `doEdit()` method.

```
File: WeatherPortlet.java

public class WeatherPortlet extends GenericPortlet {
  ...

  public void doEdit(RenderRequest rReq,RenderResponse rRes)
    throws PortletException, IOException {

    rRes.setContentType("text/html");
    PortletPreferences prefs = rReq.getPreferences();
    String zip = prefs.getValue("zip","1000");
    String unit = prefs.getValue("unit","F");
    rReq.setAttribute("zip",zip);
    rReq.setAttribute("unit",unit);
    String error = rReq.getParameter("error");
    if (error!=null) {
      error = "<font color=\"red\"><b>ERROR: </b>"
+error+"</error>";
    }
    else {
      error = "";
    }
    rReq.setAttribute("error",error);

    PortletRequestDispatcher rd =
      getPortletContext().getRequestDispatcher(
        "/weather/weatherEdit.jsp");
    rd.include(rReq,rRes);
  }

  ...
}
```

First, the `doEdit()` method sets the content type for the response. It then retrieves the current zip code and unit stored in the Portlet Preferences and stores them in the request attributes. It also checks if there is an error message from a previous request and stores it in the request attributes. A request dispatcher to `/weather/weatherEdit.jsp` is obtained and the request is dispatched to it.

```
File: weatherEdit.jsp

<%@ page import="javax.portlet.RenderRequest" %>
<%@ page import="javax.portlet.RenderResponse" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>

<portlet:defineObjects/>
<h3>Edit Weather Preferences</h3>
<p>
<%=renderRequest.getAttribute("error")%>
<p>
<form method="post" action="<portlet:actionURL/>">
 ZIP: <input type="text" name="zip" value="<%=renderRequest.getAttribute("zip")%>">
 <p>
 UNIT: <input type="text" name="unit" value="<%=renderRequest.getAttribute("unit")%>">
 <p>
 <input type="submit"> <input type="reset">
</form>
```

Using scriptlets, `weatherEdit.jsp` creates a form to edit the zip and unit portlet preferences with the current values pre-populated. It also displays any error messages. As in the `weatherView.jsp`, the URL for the action of the form is created with the `actionURL` tag of the portlet tag library.

The `doHelp()` Method

When the user clicks the "help" button on the portlet titlebar, it changes the portlet's mode to HELP and invokes the `doHelp()` method.

```
File: WeatherPortlet.java

public class WeatherPortlet extends GenericPortlet {
  ...

  public void doHelp(RenderRequest rReq, RenderResponse rRes)
    throws PortletException, IOException {

    rRes.setContentType("text/html");
    PortletRequestDispatcher rd =
       getPortletContext().getRequestDispatcher(
         "/weather/weatherHelp.html");
    rd.include(rReq,rRes);
  }

  ...
}
```

This method sets the content type and uses a request dispatcher to display the portlet's Help page.

The `processAction()` Method

The `processAction()` method is invoked when the user submits one of the forms rendered by the portlet (by the `doView()` or the `doEdit()` method.)

```
File: WeatherPortlet.java

public class WeatherPortlet extends GenericPortlet {
  ...

  public void processAction(ActionRequest aReq, ActionResponse aRes)
    throws PortletException, IOException {

    PortletPreferences prefs = aReq.getPreferences();
    String zip = aReq.getParameter("zip");

    if (aReq.getPortletMode().equals(PortletMode.VIEW)) {
      if (zip==null) {
        zip = prefs.getValue("zip","10000");
      }
      aRes.setRenderParameter("zip",zip);
    }
    else
    if (aReq.getPortletMode().equals(PortletMode.EDIT)) {
      boolean editOK;
      String errorMsg = null;
      String unit = aReq.getParameter("unit");
      prefs.setValue("zip",zip);
      prefs.setValue("unit",unit);
      try {
        prefs.store();
        editOK = true;
      }
      catch (ValidatorException ex) {
        editOK = false;
        errorMsg = ex.getMessage();
      }
      if (editOK) {
        aRes.setPortletMode(PortletMode.VIEW);
      }
      else {
        aRes.setRenderParameter("error",
URLEncoder.encode(errorMsg));
      }
    }

  ...
}
```

First, it obtains the `PortletPreferences` object. If the portlet is in VIEW mode, it checks if the zip code and units parameters are missing. If so, it sets the local variable to the default value. Finally, it sets the zip code as the render parameter, which will be received by subsequent invocations of the `doView()` method.

If the portlet is in EDIT mode, the new values are set into the Portlet Preferences object. The `store()` method of the preferences object stores the new values. The `store()` method invocation will fail if the values for zip code and units are invalid (this is explained below). If the store fails, an error flag is set and the error message is stored in the `errorMsg` local variable. If the EDIT action is successful, the portlet mode is changed to VIEW in the `ActionResponse`. Otherwise, the error message is set in the request attributes and the portlet remains in EDIT mode.

The *WeatherPreferencesValidator* Class

A portlet may define a `validator` class in its deployment descriptor. This object is invoked by the container when the `store()` method of the portlet preferences is called. If the validator throws an exception, the `store()` method is cancelled. If not, the new portlet preferences are saved to the persistent store.

```
File: WeatherPreferencesValidator.java

public class WeatherPreferencesValidator implements PreferencesValidator {

  public void validate(PortletPreferences preferences)
   throws ValidatorException {
    String zip = preferences.getValue("zip",null);
    String unit = preferences.getValue("unit",null);
    if (zip==null || unit==null) {
      Set set = new HashSet();
      set.add("zip");
      set.add("unit");
      throw new ValidatorException(
        "preferences must contain zip and unit keys",set);
    }
    try {
      int i = Integer.parseInt(zip);
      if (i<0 || i>99999) {
        throw new Exception();
      }
    }
    catch (Exception ex) {
      Set set = new HashSet();
      set.add("zip");
      throw new ValidatorException(
"zip must be a valid Zip code",set);
    }
    if (!unit.equals("F") && !unit.equals("C")) {
      Set set = new HashSet();
      set.add("unit");
      throw new ValidatorException(
        "unit must be 'F' (Fahrenheit) or 'C' (Celsius)",set);
    }
  }

}
```

This validator ensures that the zip code entered is a valid US zip code and that the units are Fahrenheit or Celsius.

The `portlet.xml` Deployment Descriptor

The `portlet.xml` deployment descriptor defines the meta information the portlet container needs to run the portlet. For each portlet definition in the `portlet.xml` deployment descriptor, the portlet container will instantiate a single object of that class to serve all the requests targeted to that portlet. This is identical to a servlet container's behavior.

```
File: portlet.xml

<portlet-app>
  <portlet>
    <portlet-name>WeatherPortlet</portlet-name>
    <portlet-class>sample.portlet.WeatherPortlet</portlet-class>
    <init-param>
      <name>weather.url</name>
      <value>java:/comp/env/WeatherProvider</value>
    </init-param>
    <expiration-cache>3600</expiration-cache>
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>EDIT</portlet-mode>
      <portlet-mode>HELP</portlet-mode>
    </supports>
    <portlet-info>
      <title>WeatherPortlet</title>
    </portlet-info>
    <portlet-preferences>
      <preference>
        <name>zip</name>
        <value>95054</value>
      </preference>
      <preference>
        <name>unit</name>
        <value>F</value>
      </preference>
      <preferences-validator>
        sample.portlet.WeatherPreferencesValidator
      </preferences-validator>
    </portlet-preferences>
  </portlet>
</portlet-app>
```

The `portlet-name`, `portlet-class` and `init-param` elements define the name, class and initial parameters for the portlet.

The `expiration-cache` element defines how long (one hour in this example) the portal/portlet-container should cache content created by the portlet. If the portlet does not set the `expiration-cache` property during a render request, the value specified in the deployment descriptor will be used.

The `supports` element defines what content types the portlet supports and the modes supported for that content type. The portal/portlet-container will not invoke a portlet in modes the portlet has not defined as supported.

The `portlet-info` element defines information such as the default title for the portlet.

The `portlet-preferences` element defines the default values for the Portlet Preferences (zip and unit in this example), as well as the `validator` class to use when invoking the store method on the `PortletPreferences` object.

The `web.xml` Deployment Descriptor

As portlet applications are extended web applications, they must include a `web.xml` file.

```
File: web.xml

<web-app>
  <display-name>PortletApp Sample</display-name>
</web-app>
```

The `web.xml` file in this example simply defines the name of the web application.

What The Portlet Specification Does Not Address

The Portlet Specification is rather complete in addressing needs for developing individual portlets. However, it is important to remember that portal servers usually offer additional functionality that is not addressed by *The Portlet Specification*. It is also important for enterprises and service providers to consider such add-ons when deciding on a specific portal server product.

Portlet Aggregators

Many developers want to format their portlets on the desktop in a specific way—as tables or tabs on the desktop, for example. Sun™ Open Net Environment (Sun ONE) Portal Server software users are familiar with the Container Provider that provides this display flexibility.

Each portal server vendor will implement its own aggregation and desktop customization features. The Portlet Specification reference implementation provides a basic aggregation framework, which is simpler than those offered by most commercial portal servers.

Pre-built Portlets

Many portal server products come with existing portlets that can be deployed and used without any customization. However, besides the `GenericPortlet`, which is a convenience class for developers, *The Portlet Specification* does not define any portlets that should be shipped with portal servers.

Administration and Portlet Deployment

Most portal server products provide an administrative interface to configure users, groups, security, and other information. These administration interfaces may include facilities for portlet deployment. If not, a separate deployment tool is required for deploying portlets.

Though the specification does stipulate a standard for packaging portlets in a WAR file, each portal server software vendor must provide a mechanism for deploying the WAR files and administering the Portal. Tool vendors that wish to distribute portlets that work with any compliant product should be aware that they must document any product-specific steps to portlet deployment.

Common Portlet Functionality

Portlet developers are accustomed to implementing certain functionality across a wide range of portlets—enabling several portlets to identify the user from one login screen, for example. *The Portlet Specification* and the Portlet API do not address this, but developers can still implement that functionality, using the portal's user principal object.

Miscellaneous

Besides those already listed, portal server software vendors will continually attempt to differentiate themselves with new features and product additions. For example, the Sun ONE Portal Server, Secure Remote Access product includes the Netlet, patented secure remote access technology that allows users to access applications running inside a corporate firewall from any browser, without compromising security. Another example is the Sun ONE Portal Server, Mobile Access product that enables portlets such as email, calendar, address book and other enterprise content to be delivered to mobile clients. *The Portlet Specification*, at least in its first iteration, will not address these specialized features.

Portlet Specification Adoption

As with any new standard, wide industry adoption of *The Portlet Specification* is key. Since all the major portal vendors are actively participating in the Java Community Process℠ forum for developing JSR 168, the specification is positioned to become ubiquitous among portal server vendors. It also indicates that the portal server market is maturing, similar to the application server marketplace after the J2EE Specification.

Key tasks for adopting the standard include:

- Transitioning from proprietary portlet API's
- Ensuring proper product adoption
- Providing a well-defined timeline for specification release.

Porting Existing Portal Server Integrations

Before *The Portlet Specification*, each portal server product provided its own API for creating portlets. Portal server vendors, and developers who have built portlets with these proprietary API's, must continue to support their existing customers as *The Portlet Specification* gains acceptance. Fortunately, the migration of proprietary portlets to specification-compliant portlets does not necessarily mean completely rewriting existing code.

The reusability of existing code depends on the portlet's modularity. With the Sun ONE Portal Server, if the code using the Provider API is separate from the business logic, porting will be simpler, as it will be easy to locate the code that needs to be modified. Otherwise, transitioning to the new specification will require more effort, since each call to the Provider API will need to be located and modified throughout the code.

Though the existing Provider API does not provide all the functions of *The Portlet Specification*, it provides most of them. Therefore, modifications around those functions will be mostly syntax changes. Portlets that make extensive use of Provider API functions not included in *The Portlet Specification* will require more effort to port. Fortunately, Sun will continue to support the Provider API, so the porting can be done gradually.

Sun ONE Portal Server Adoption

Sun Microsystems remains committed to the adoption of open standards, and has adopted *The Portlet Specification*. Sun Microsystems will continue to support their existing Provider API (PAPI) for the next releases of the Sun ONE Portal Server, to provide a comfortable migration path to *The Portlet Specification* for existing developers. The company will solicit input from the developer community to make sure that transition issues are addressed.

Additional Information

Java Specification Request 168 Web Site

http://www.jcp.org/en/jsr/detail?id=168

Sun ONE Portal Server Product Page

http://www.sun.com/software/products/portal_srvr/home_portal.html

Send email to

jsr168-interest@sun.com with questions about:

- The Sun ONE Portal Server Early Access Program
- Partnership opportunities
- Questions about *The Portlet Specification*.

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 800 786-7638 or +1 512 434-1577 **Web** sun.com

We make the net work.

Sun Worldwide Sales Offices: Argentina: +5411-4317-5600, Australia: +61-2-9844-5000, Austria: +43-1-60563-0, Belgium: +32-2-704-8000, Brazil: +55-11-5187-2100, Canada: +905-477-6745, Chile: +56-2-3724500, Colombia: +571-629-2323 Commonwealth of Independent States: +7-502-935-8411, Czech Republic: +420-2-3300-9311, Denmark: +45 4556 5000, Egypt +202-570-9442, Estonia: +372-6-308-900, Finland: +358-9-525-561, France: +33-134-03-00-00, Germany: +49-89-46008-0 Greece: +30-1-618-8111, Hungary: +36-1-489-8900, Iceland: +354-563-3010, India–Bangalore: +91-80-2298989/2295454; New Delhi: +91-11-6106000; Mumbai : +91-22-697-8111, Ireland: +353-1-8055-666, Israel: +972-9-9710500 Italy: +39-02-641511, Japan: +81-3-5717-5000, Kazakhstan: +7-3272-466774, Korea: +822-2193-5114, Latvia: +371-750-3700, Lithuania: +370-729-8468, Luxembourg: +352-49 11 33 1, Malaysia: +603-21161888, Mexico: +52-5-258-6100 The Netherlands: +00-31-33-45-15-000, New Zealand–Auckland: +64-9-976-6800; Wellington: +64-4-462-0780, Norway: +47 23 36 96 00, People's Republic of China–Beijing: +86-10-6803-5588; Chengdu: +86-28-619-9333 Guangzhou: +86-20-8755-5900; Shanghai: +86-21-6466-1228; Hong Kong: +852-2202-6688, Poland: +48-22-8747800, Portugal: +351-21-4134000, Russia: +7-502-935-8411, Saudi Arabia: +9661 273 4567, Singapore: +65-6438-1888 Slovak Republic: +421-2-4342-94-85, South Africa: +27 11 256-6300, Spain: +34-91-596-9900, Sweden: +46-8-631-10-00, Switzerland–German: 41-1-908-90-00; French: 41-22-999-0444, Taiwan: +886-2-8732-9933, Thailand: +662-344-6888 Turkey: +90-212-335-22-00, United Arab Emirates: +9714-3366333, United Kingdom: +44-1-276-20444, United States: +1-800-555-9SUN OR +1-650-960-1300, Venezuela: +58-2-905-3800, Or Online at sun.com/store

Printed in USA 7/03 FE2026-0/1K