

Developing Portlets - Tips & Traps

Howard Ong

Principal Consultant

Aurora Consulting Pty Ltd

Abstract

One of the most useful functionalities of Oracle Portal is the ability to develop custom Java Portlets, leveraging the best of what Oracle Portal and J2EE have to offer. This paper examines the various tips and traps of Java Portlet development. For a higher level of realism, the paper uses a demo Portlet implementation as a case study. From the demo implementation, the reader would also experience some examples of how other Oracle and Java technologies may be integrated with Oracle Portal for a function-rich development environment.

About The Author

A Principal Consultant of Aurora Consulting Pty Ltd and a frequent presenter at Oracle Conferences, Howard has been an expert user of Oracle database and tools for the past 10 years and has been working on Internet Applications since 1995. Howard possesses in-depth experience in the planning and development of E-Business and other Web-Based Systems. Harboring a keen interest in the application of Oracle Technology in Internet application development, Howard and his team of consultants have helped many organisations deploy web-based solutions using technology such as the Oracle Portal, Oracle Toplink, Oracle 9iFS, Oracle 9iAS, Oracle JDeveloper, Oracle Database Server and other third-party tools. Aurora's recent successes include the Agriculture Market Intelligence Centre and several other projects at Department Of Agriculture; and Teachers' Learning Support Network at Department Of Education & Training. Howard has worked with a wide variety of organisations such as government departments; and companies from mining, finance and transportation industries. In addition to Internet Application and Business Intelligence development, other services that Howard rendered to these organisations include strategic planning, business analysis, data analysis and data administration.

For more information on Howard or Aurora Consulting, visit <http://www.aurora-consult.com.au> or email info@aurora-consult.com.au.

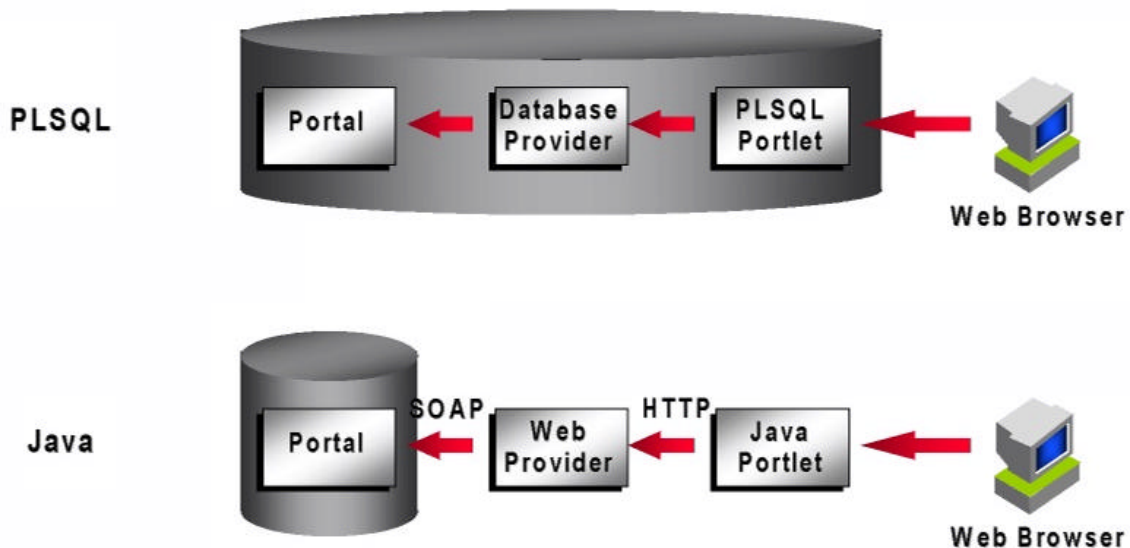
Introduction

One of greatest advantage of the Oracle Portal environment is the ability to develop custom-built portlets. It is in this capacity that Oracle Portal is becoming an effective container-application for most other web applications.

There are 2 common development platforms for Portlet development : Java and PLSQL. This paper examines some of the advantages, tips and traps of Java Portlets development.

Java Or PLSQL

The key difference between Java and PLSQL Porlets is an architectural one. The diagrams below illustrate the difference.



PLSQL Portlets utilise Database Providers to communicate with Oracle Portal. Database Providers and PLSQL Portlets are in fact PLSQL packages residing in the same database as the Portal repository. To display web contents, a PLSQL Portlet uses the good-old PLSQL Toolkit API. It is this reason that PLSQL Portlets provide developer a less drastic means of converting their existing PLSQL applications to Portlets.

Java Portlets, on the other hand, communicate to Oracle Portal via Web Providers. A Java Portlet communicates to the Web Provider via HTTP, which in turn communicates to Oracle Portal via SOAP (SOAP, or Simple Object Access Protocol, is a light-weight XML-based protocol for information exchange in a distributed environment.) The Web Provider is in fact a Java Servlet; whereas Java Portlets can be either Java Servlets or JSPs.

Java Portlets are preferred over PLSQL Portlets for the following reasons :

- The PLSQL Portlets, Database Providers, Portal repository and the application schema all reside in the **same** database. This gives rise to a number of issues :
 - They compete for the same database resources.
 - One repository cannot be upgraded without affecting the other. For example, if the application database requires a new feature that is available only in a new version of Oracle. This upgrade would have affected the Portal Repository and may be infeasible if there is a compatibility issue between Portal and the new database version.

- Poor database design or inefficient query on the part of the application can affect Portal performance.
- With the Java Portlets architecture, the Portlets, Web Providers, Portal repository and application schema can all reside on separate servers. In fact, among themselves individual Portlets, Providers and application schemas can reside on separate servers as well. For example, it may be desirable to locate a mission-critical Porlets on a dedicated server, for security and performance reasons. Therefore, this architecture allows the developer maximum flexibility in distributing work load across the network.
- PLSQL Portlets and Providers are “configured” by hard-coding the configuration information into the PLSQL packages, eg `get_portlet` function in the Provider package and `get_portlet_info` function in the Portlet package. With the Provider package referencing the Portlet package, both packages have to be recompiled each time a Porlet is configured. Java Portlet configuration information, on the other hand, is stored in configuration files.

Being a multi-vendor, open platform, Java is a functionally richer and more versatile development environment than PLSQL. The sheer number of add-on libraries and well-tested Design Patterns that are readily available also makes Java a productive and low-cost environment to operate on.

Tips & Traps

Deploying A Java Portlets

The deployment of a Servlet Portlet involves the following steps :

- Create and deploy the Java Servlet as you normally would. Below is a simple Hello-World Servlet :

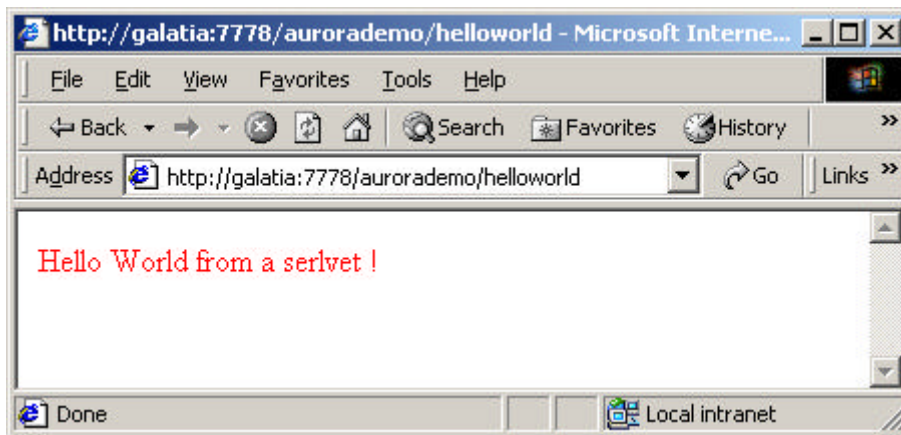
```
package au.wa.aurorademo;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

/**
 * A simple Hello World Servlet
 *
 * @author Howard Ong, Aurora Consulting
 */
public class HelloWorld extends HttpServlet {

    public void doPost (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        doGet(req, res);
    }

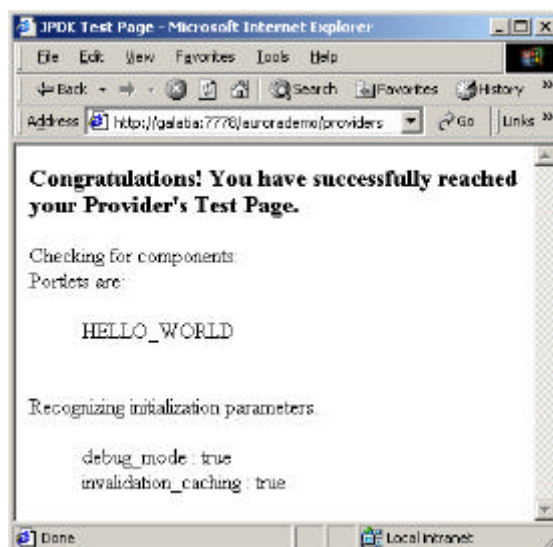
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        PrintWriter out = res.getWriter();
        out.println("<html><body>");
        out.println("<font color=\"#FF0000\">Hello World from a servlet !</font>");
        out.println("</body></html>");
    }
}
```



- Configure a Web Provider by creating a provider.xml file. The part of the provider.xml file that configures the Hello-World servlet as a portlet is shown below :

```
<portlet class="oracle.portal.provider.v2.DefaultPortletDefinition">
  <id>l</id>
  <name>HELLO_WORLD</name>
  <title>Hello World Servlet</title>
  <shortTitle>Hello World Servlet</shortTitle>
  <description>This is the Hello World Servlet wrapped in a portlet.</description>
  <timeout>6000</timeout>
  <timeoutMessage>The application has timed out</timeoutMessage>
  <hasHelp>>false</hasHelp>
  <hasAbout>>false</hasAbout>
  <showDetails>>false</showDetails>
  <showEdit>>false</showEdit>
  <showEditToPublic>>false</showEditToPublic>
  <showEditDefault>>false</showEditDefault>
  <showPreview>>false</showPreview>
  <acceptContentType>text/html</acceptContentType>
  <renderer class="oracle.portal.provider.v2.render.RenderManager">
    <contentType>text/html</contentType>
    <showPage>/helloworld</showPage>
  </renderer>
</portlet>
```

- Test if the Web Provider recognises the portlet.

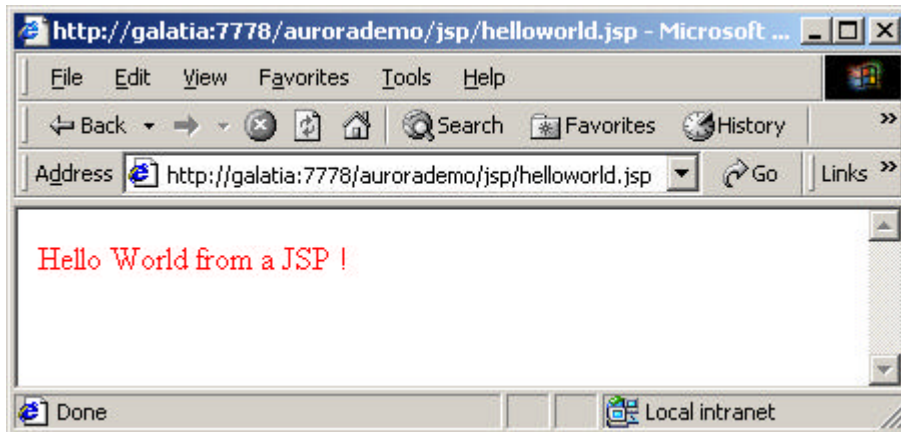


- Deploy the Hello-World Servlet-Portlet in a Portal Page.

The deployment of a JSP servlet is just as simple.

- Create and deploy the JSP as you normally would. Below is a simple Hello-World JSP :

```
<html>
<body>
<font color="#FF0000"><% out.println("Hello World from a JSP !"); %></font>
</body>
</html>
```



- Configure the Web Provider as before, the only difference is in the <showPage> tag.

```
<portlet class="oracle.portal.provider.v2.DefaultPortletDefinition">
  <id>2</id>
  <name>HELLO_WORLD_JSP</name>
  <title>Hello World JSP</title>
  <shortTitle>Hello World JSP</shortTitle>
  <description>This is the Hello World JSP wrapped in a portlet.</description>
  <timeout>6000</timeout>
  <timeoutMessage>The application has timed out</timeoutMessage>
  <hasHelp>false</hasHelp>
  <hasAbout>false</hasAbout>
  <showDetails>false</showDetails>
  <showEdit>false</showEdit>
  <showEditToPublic>false</showEditToPublic>
  <showEditDefault>false</showEditDefault>
  <showPreview>false</showPreview>
  <acceptContentType>text/html</acceptContentType>
  <renderer class="oracle.portal.provider.v2.render.RenderManager">
    <contentType>text/html</contentType>
    <showPage>/jsp/helloworld.jsp</showPage>
  </renderer>
</portlet>
```

- Test the Web Provider and deploy the new portlet on a Portal Page.

Below is a screen shot of the 2 Hello-World portlets :



Note that in the above example, the <HTML> and <BODY> tags from the servlet and JSP are included within the Portal HTML code. In other words, strictly speaking, the syntax is incorrect and may be rejected by some browsers. To overcome this problem, portlet code should not include tags such as <HTML>, <HEAD>, and <BODY>.

Qualifying Form & Parameter Names

The ability to include multiple portlets on a Portal Page gives the Portal Developer the ability to rapidly construct a Portal Page with the appropriate portlets, catered to specific user's needs. A Portal Page is always rendered as a single body of HTML code, with all the codes from multiple portlets embedded within it. This implies that all parameters received by the page are in fact visible to all portlets. This could also generate syntactically incorrect HTML with duplicate form name. More seriously, it could cause the parameter to be unintentionally received and processed by other portlets on the page, causing unpredictable results. It is therefore a good practice to always qualify all forms and parameter names with a unique identifier of that particular portlet instance. On receiving parameters, the portlet must then check the qualified parameter to ascertain that it is the intended audience of the parameters. The `HttpPortletRendererUtil.portletParameter()` does exactly that, prefixing the parameter name with unique identifier. However, because the unique identifier always starts with a digit, the qualified form and parameter names becomes unsuitable for use in a Java Script. To overcome this, you should prefix it further with an alphabetic character.

Below is a snippet of JSP code that qualifies a parameter name in this manner :

```
<form name="<%= "A." +HttpPortletRendererUtil.portletParameter(portletRequest, "actionForm") %>"
  action="<%=actionUrl%>" method="post">
  ...
  <input class="form-field" type="input"
    name="<%= "A." +HttpPortletRendererUtil.portletParameter(portletRequest, "message") %>"
    size="40">
  ...
</form>
```

The result is the following HTML being generated :

```
<form name="A.52.4.286_SIMPLE_FORM_2_52.actionForm"
  action="http://galatia:7778/servlet/page" method="post" >
  ...
  <input class="form-field" type="input"
    name="A.52.4.286_SIMPLE_FORM_2_52.message"
    size="40">
  ...
</form>
```

Inter-Portlet Communication

The previous example explains why it is important to qualify form and parameter names. One limitation of the qualification mechanism is that there is no way one portlet can determine the qualifier of another portlet. In other words, a portlet is not able to qualify parameters in such a way that another portlet on the page can receive it. As such, there is no reliable way by which one portlet can communicate with another.

Form Resubmission When Switching Tabs

When the user switches to another tab and switches back again, in an attempt to re-display the page with its previous content, Oracle Portal resubmits all the forms on the page with the same parameters. A more sensible approach would be to cache the previous content of the page and simply re-display it.

The re-submission of forms cause 2 problems :

- On a query form, it consumes unnecessary clock cycles by performing the query again.
- On an insert, update or delete form, it attempts to perform the operation again, possibly causing data errors.

One way of overcoming this is to create a “first-time” flag in the HttpSession or ProviderSession object when the form is first submitted. When the form is processed for the first time, the flag is deleted. Subsequent attempts by Oracle Portal to resubmit the form will not find the “first-time” flag and hence the operation will not be repeated.

Multipart Forms

Multipart Forms are essential when submitting large amount of data, such as the uploading of a file. The Provider Servlet, which accepts all submitted forms on behalf of all portlets, does not support Multipart Forms. In other words, Oracle Portal does not support file upload.

A workaround to this limitation is to submit the form to a standalone Servlet running outside Portal. With a few clever forwarding, the user may not even be aware that an external Servlet is involved ! The limitation of the workaround is that the external Servlet would not run as a portlet session and is hence unable to access the same session-specific information from the original Portlet.



Restricting Portlet Access

While it is easy to suppress the display of a tab to unauthorised users, there is no easy way to suppress the display of a portlet in a similar way. Depending on the type of portlets, 2 workarounds are available :

- For portlets based on the folder content, access restrictions may be set on the folder to achieve similar result.
- For custom built portlets, custom codes would have to be written to suppress the display of all HTML for unauthorised users.

Is This User In This Group ?

While JPDK supports ample means to get user information (such as the `PortletRenderRequest.getUser()` method), it is rather limited in providing Group-based information. For example, a commonly required piece of information is whether the current user belongs to a certain group. This information is only available by invoking the PLSQL function `wwsec_api.is_user_in_group()`. The use of the PLSQL API is not ideal for the following reasons :

- There is a fair amount of overheads required on the part of the Java Portlet : get the connection information, establish a connection, issue the PLSQL call, get the result.
- JPDK is a more consistent and reliable API. For example, JPDK Release 2 has retained many of the classes and methods of Release 1. In addition, you may use JPDK Release 2 and all its new features on an instance of Oracle Portal Release 1.

PLSQL on the other hand, are subjected to changes from one Portal version to another. For example, the `wwsec_api` package mentioned above was available in Portal Release 1 but was **not** available in the earlier versions of Release 2. It has been reintroduced from Release 2 version 9.0.2.6 onwards.

To access any Group-base operation that is specifically available on the PLSQL API only, a custom-built Java API would have to be written. The custom API would make the appropriate PLSQL call and wrap the result in a Java method. Below is a simplified example :

PLSQL :

```
create or replace function is_user_in_group
(p_user_name in varchar2, p_group_name in varchar2)
return varchar2 is
  v_usr_in_grp boolean;
begin
  v_usr_in_grp
    := wwsec_api.is_user_in_group(wwsec_api.id(p_user_name),
                                wwsec_api.group_id(p_group_name));
  if v_usr_in_grp then
    return 'Y';
  else
    return 'N';
  end if;
end;
/
```

Java :

```
static public boolean isUserInGroup (String userName, String groupName)
throws PortalUserNotFoundException, PortalGroupNotFoundException, Exception {

  boolean rtn;
  String booleanStr;

  try {
    ...
    query = "{?= call is_user_in_group(?, ?)}";
    statement = (CallableStatement) connection.prepareCall(query);
    statement.registerOutParameter (1, Types.VARCHAR);
    statement.setString (2, userName);
    statement.setString (3, groupName);
    statement.execute();
    booleanStr = statement.getString (1);
    ...
  }
  catch (SQLException e) {
    ...
  }
  if (booleanStr.toUpperCase().equals("Y")) {
    rtn = true;
  }
  else {
    rtn = false;
  }
  return rtn;
}
```

Conclusion

In spite of a number of significant limitations in Oracle Portal, as this paper demonstrates, many of the limitations can be worked around. As such, Oracle Portal remains a promising product that gives the developer the ability to more precisely tailor to the information needs of users.