Note: The names of types presented in this section are the same both in IBM portlet API and JSR 168 API. The only difference is that the types will belong to different packages for each of API.

For IBM portlet API, look at the `com.ibm.wps.portletservice.credentialvault.credentials` package.

For JSR 168 API, look at the `com.ibm.portal.portlet.service.credentialvault.credentials` package.

20.4.1 Passive credential objects

These are containers for the credential's secret. Portlets that use passive credentials need to extract the secret out of the credential and do all the authentication communication with the back-end resource. All passive credential objects must implement the `PassiveCredential` interface. The following sections present the passive credential objects that are shipped with WebSphere Portal.

Note: Currently, the vault service in WebSphere Portal can store only `UserPasswordPassive` objects.

UserPasswordPassive

This type of credential stores secrets in the form of user ID/password pairs. It is implemented by the class `UserPasswordPassiveCredential` and provides the following methods:

- getPassword

  This method returns the password of the credential's secret.

- getSecretType

  This method returns the credential's secret type in terms of the constants declared in the `CredentialVaultService` interface.

- getUserId

  This method returns the user ID of this credential's secret.

SimplePassive

This type of credential stores secrets in the form of serializable Java objects. It is implemented by the class `SimplePassiveCredential` and provides the following methods:

- getSecretType

This method returns the credential's secret type in terms of the constants declared in the `CredentialVaultService` interface.

- getSecret

  This method returns the user's secret as an object.

JaasSubjectPassiveCredential

This type of credential relies on the Java Authentication and Authorization Service, and stores secrets in the form of javax.security.auth.Subject objects. It is implemented by the class `JaasSubjectPassiveCredential` and provides the following methods:

- getSecretType

  This method returns the credential's secret type in terms of the constants declared in the `CredentialVaultService` interface.

- getSecret

  This method returns the user's secret as an jaas subject.

20.4.2 Active credential objects

These objects hide the credential's secret from the portlet; there is no way of extracting it out of the credential. In return, active credential objects offer business methods that take care of all the authentication. All passive credential objects must implement the `ActiveCredential` interface. The following sections present the active credential objects that are shipped with WebSphere Portal.

Note: When using active credentials, portlets never get in touch with the credential secrets and thus there is no risk a portlet could violate any security rules such as, for example, storing the secret on the portlet session. While there might not always be an appropriate active credential class available, this is the preferred type of credential objects to use.

HttpBasicAuth

This type of credential stores userid/password secrets and supports HTTP Basic Authentication. It is implemented by the class `HttpBasicAuthCredential` and provides the following method:

- getAuthenticatedConnection

  This method returns a new HttpURLConnection with added authentication data. There are two different signatures available. One that receives a `String` object representing the url, and another that receives a `URL` object.

Archived

HttpFormBasedAuth

This credential object stores userid/password secrets and supports HTTP Form-Based Authentication. You must initialize this object prior to using it, using the provided `init` method. This method expects a `Map` of key/value pairs, and the value/name pairs that must be provided are shown in Table 20-1.

Table 20-1 Configuration data to initialize a HTTPFormBasedAuthCredential

Key name	Type	Description	Mandatory?
KEY_CREDENTIAL_SECRET	`UserPassword CredentialSecret`	The credential's secret	Y
KEY_USERID_ATTRIBUTE_NAME	`String`	The name under which the user ID is posted	Y
KEY_PASSWORD_ATTRIBUTE_NAME	`String`	The name under which the user password is posted	Y
KEY_LOGIN_URL	`String`	The url to which the login data is posted	Y
KEY_LOGOUT_URL	`String`	The url to which an HTTP GET request is send in order to log out the user	Y
KEY_USE_AUTH_COOKIES	`Boolean`	Specifies whether the authentication data are cookies [true] of URL rewriting [false]	Y
KEY_FORM_DATA	`List`	Any additional name=value pairs that need to be posted with the login POST	N

This type of credential is implemented by the class `HttpFormBasedAuthCredential` and provides the following methods:

- getAuthenticatedConnection

  This method returns a new HttpURLConnection with added authentication data. There are two different signatures available. One that receives a `String` object representing the url, and another that receives a `URL` object. This latter signature can be used only if cookies are used for authentication.

- login

  This method performs the HTTP form based login.

Archived

- logout

  This method performs the logout through an HTTP GET request to the logoutUrl.

JavaMailCredential

This credential object stores userid/password pairs and leverages the authentication functionality of the javax.mail API. It is implemented by the class `JavaMailCredential` and provides the following methods:

- getSecretType

  This method returns the credential's secret type in terms of the constants declared in the `CredentialVaultService` interface.

- getAuthenticatedSession

  This method authenticates an `javax.mail.Session`. There is a signature that receives the Session to authenticated and the mail server host name, and another that receives these two parameters and also the mail server port number.

LtpaTokenCredential

This credential object is for authenticating with a backend system that is within the same WebSphere Application Server single sign-on domain as the portal. It is implemented by the class `LtpaTokenCredential` and provides the following method:

- getAuthenticatedConnection

  This method returns a new HttpURLConnection with added authentication data. There are two different signatures available.

SiteMinderTokenCredential

A credential object for authenticating with a backend system that is within the same SiteMinder single sign-on domain as the portal. This credential should be used when SiteMinder is used as an authentication proxy for the portal. It is implemented by the class `SiteMinderTokenCredential` and provides the following method:

- getAuthenticatedConnection

  This method returns a new HttpURLConnection with added authentication data. There are two different signatures available.

WebSealTokenCredential

A credential object for authenticating with a backend system that is within the same WebSEAL single sign-on domain as the portal. This credential should be

Archived

used when a WebSEAL authentication proxy is used by the portal. It is implemented by the class `WebSealTokenCredential` and provides the following method:

- getAuthenticatedConnection

  This method returns a new HttpURLConnection with added authentication data. There are two different signatures available.

20.4.3 Storing credential objects in the PortletSession

You cannot store credential objects directly in the PortletSession because they do not implement java.io.Serializable. This is for security reasons, once that the credential classes store the actual credential secret as one of their private attributes. This could allow the secret to be found by anyone who has access to the application server session database.

However, if you ensure that the PortletSession is not serialized in a cluster setup, the credential object could be stored, for example, as a transient member of a container class. You only have to make sure to check if the credential object got lost during serialization and, in this case, retrieve it from the vault again.

Archived

21

The Credential Vault

This chapter provides a sample scenario for creating a sample JSR 168 portlet application that uses Credential Vault to log in and interact with back-end systems. You will create, deploy and run a sample portlet application using the Credential Vault to connect to a secure backend Web application. The sample scenario will allow you to understand the techniques used to develop portlets using the Credential Vault provided by IBM WebSphere Portal.

The sample scenario included in this chapter illustrates the following:

- How the Credential Vault with active credentials is used
- How the Credential Vault with passive credentials is used
- How to store credentials
- How to retrieve credentials
- How to log in to the Web application
- How to retrieve the Web application content in the portlet's View mode

Note: The portlet application described in this chapter has the following characteristics:

- Portlet API: JSR 168
- Application type: MVC

© Copyright IBM Corp. 2005. All rights reserved.

21.1 Sample scenario

In this sample scenario, you will create a sample portlet JSR 168 based on a Basic portlet (JSR 168) type using the Portlet Wizard. You will also use this wizard to enable Credential Vault to interact with back-end resources.

In this scenario, the protected back-end resource is a servlet and requires a user ID and password credentials to log in to the Web application (servlet) to retrieve some information. The servlet application has been secured with HTTP Basic Authentication.

In the first part of this scenario, active credentials are used to access a secure Web application using HTTP Basic Authentication. Once this is done, the sample scenario illustrates how the JSR 168 portlet is updated to use Credential Vault passive credentials.

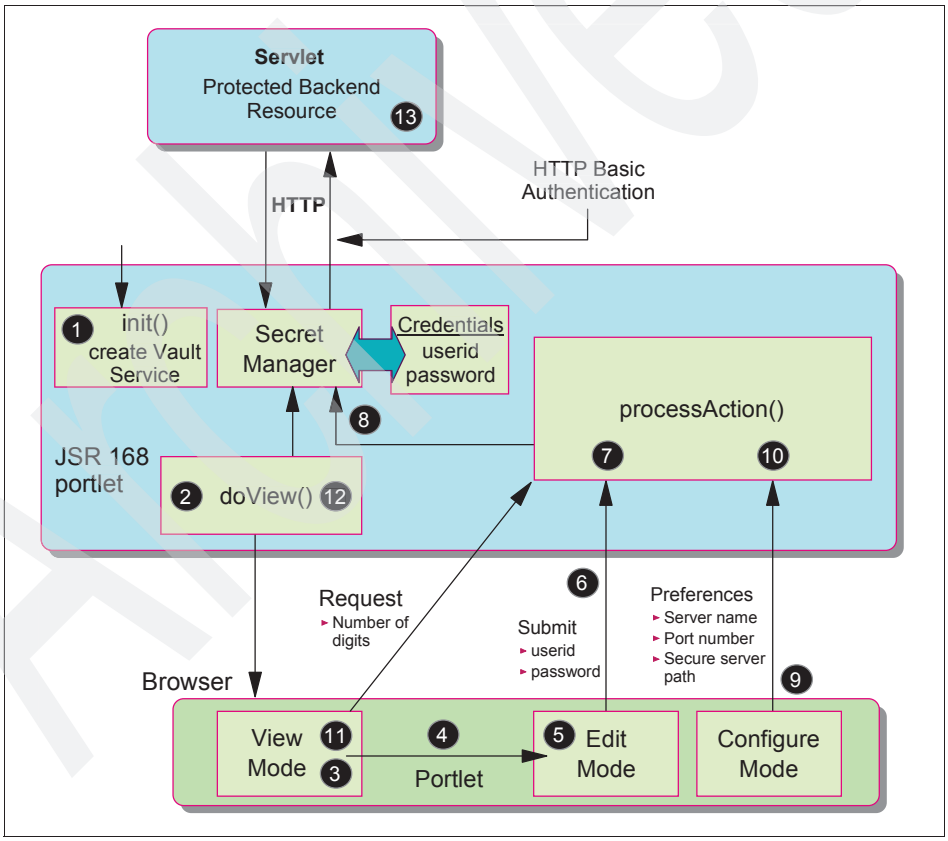


Figure 21-1 Credential Vault sample scenario

Arrived

The sequence flow for this scenario is as follows:

1. The init() method is used to initialize the Credential Vault Service.
2. Portal invokes the portlet doView() method. Since initially, no credentials have been stored, a message is written indicating that a user ID and password must be entered in Edit mode.
3. In the portlet View mode, a message is shown directing the user to use the Edit mode to enter credentials.
4. The user clicks **Edit** to go into Edit mode.
5. The Edit mode screen is displayed, that is, the doEdit() method is executed and a JSP displays a form to allow the user to enter credentials (user ID and password) and submit the action.
6. The user enters a user ID and password and select **Save**.
7. The processAction() method is executed to process the action.
8. The processAction() method invokes the wizard generated Secret Manager to create a slot and store the entered user ID and password information.
9. An administrator will switch to configure mode (through View mode) and enter or change preferences such as the backend server name, port number and secure server path.
10. The processAction() method is executed to process the action and save the preferences.
11. When the user returns to View mode, it displays a form to enter the value that you will send to the servlet as an argument. Once the user enters a value and clicks **Send** the processAction() method is executed to retrieve the value.
12. The doView() method is eventually executed. The following tasks are executed in this mode:
    a. An HttpBasicAuth active credential object is retrieved from the credential service. Because authentication is done in this object, we never get in touch with the real credentials.
    b. The authorization header is set in the request HTTP header.
    c. The connection to the back-end resource (protected servlet in this scenario) is invoked.
13. The user is authenticated and the servlet executes. The received content is rendered to the user in View mode.

Archived

21.2 Importing a secure servlet application

In this section, you will import a previously created secure servlet. This servlet will be the back-end secure resource you will access using Credential Vault. The servlet is only accessible via HTTP basic authentication.

Note: In this sample scenario it is assumed that the WAR file containing the secure Web application (SecureServlet) resides in the following path:

c:\LabFiles\CredentialVault\SecureServlet.war

Follow these steps to import the secure servlet:

1. If required, start IBM Rational Application Developer.
2. If needed, switch to the Web perspective.
3. From the main menu, select **File** → **Import...**.
4. Select **WAR file** and click **Next**.

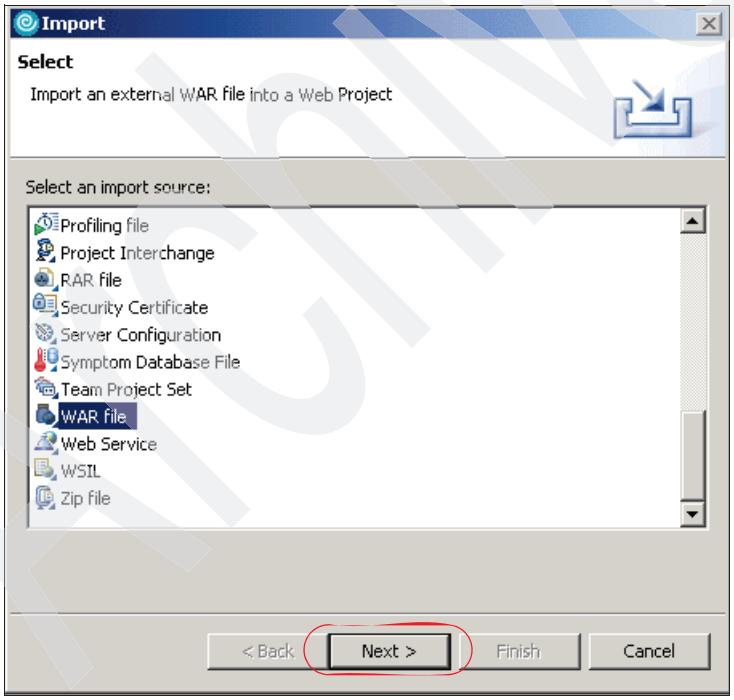


Figure 21-2 Importing a WAR file

5. In the next window, enter the following values:

Archived

a. Browse to the location of the SecureServlet.war file in c:\LabFiles\CredentialVault\SecureServlet.war.

b. Web project: `SecureServlet`

c. Target server: select **WebSphere Application Server V5.1**.

d. EAR project: `SecureServletEAR`.

Note: The SecureServletEAR project will also be created for you.

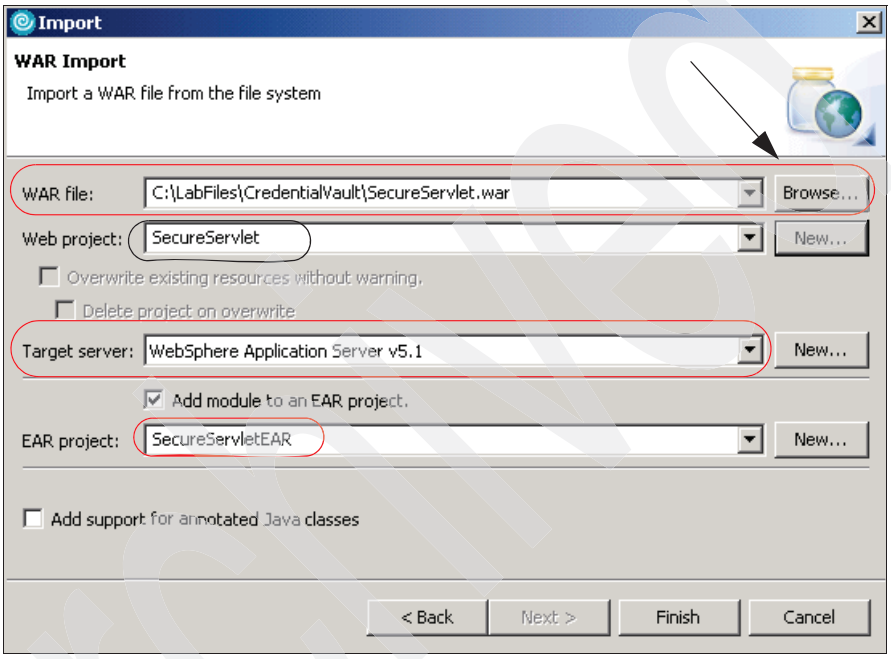


Figure 21-3 Import the SecureServlet WAR file

6. Click **Finish** to import the application.

7. Execute the secure servlet to check that it is running properly. In the Project Explorer view, expand the **SecureServlet/WebContent/jsp** folder.

8. Right-click **input.jsp** and select **Run** → **Run on server...** from the context menu.

9. If a WebSphere V5.1 Test Environment server already exists, select it from the list and click **Next**. If it does not exist, select **Manually define a server** and select WebSphere V5.1 Test Environment as server type, see Figure 21-4 on page 646. Click **Next**, leave the default port number (9080) and click **Next** again.

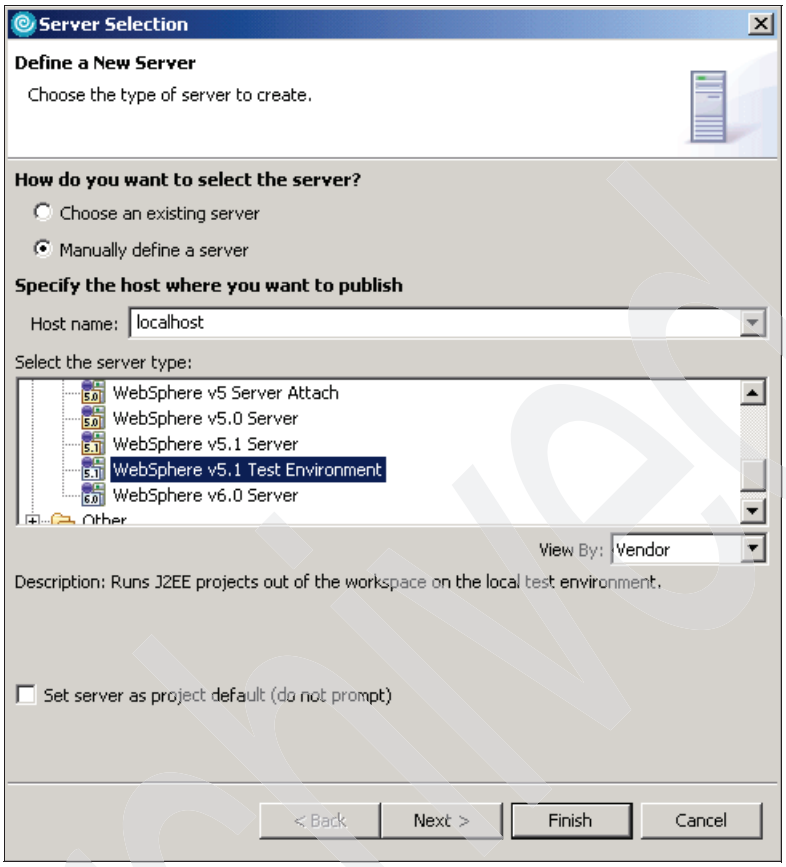


Figure 21-4 Manually define a server

10. In the last window ,you will see available projects and configured projects for this server. Make sure SecureServletEAR is a configured project.
11. Click **Finish**
12. The internal Web browser opens and you will see a form. You also can invoke the Web application from an external browser. Enter a number of digits you want the prime number to be. Click **send**.

Archived

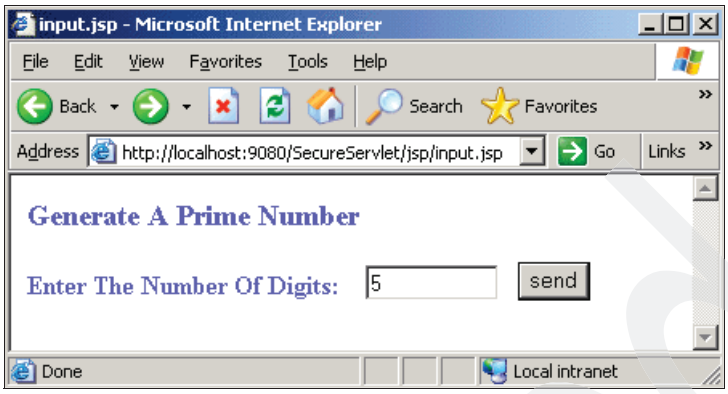


Figure 21-5 Running the secured servlet

13. Because this servlet is secure and requires HTTP Basic Authentication, you have to enter a user name and password. Enter `user1` as the user name and `password1` as the password. Click **OK** to invoke the secure servlet.

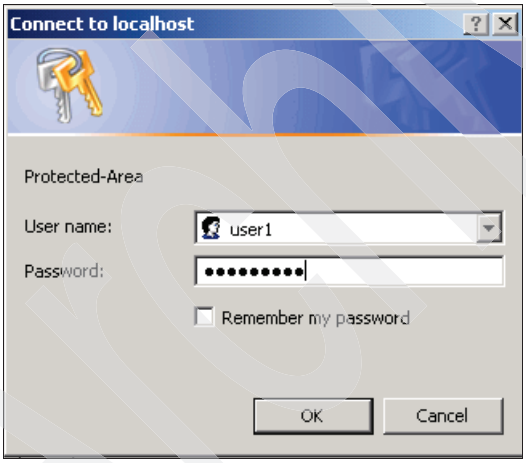


Figure 21-6 Basic authentication

14. The browser displays the authenticated user name and the generated prime number with the number of digits you entered in the form.

Redbooks

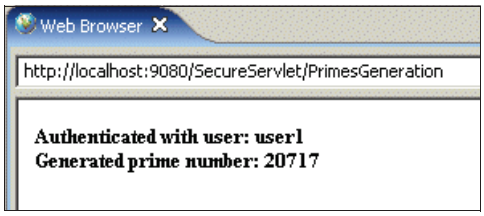


Figure 21-7 Secure Web application results

15. In the Servers view, select **WebSphere V5.1 Test Environment** server and click the red **Stop** button to stop the server.

21.3 Using active credentials

After importing and testing the protected servlet, you will build a portlet JSR 168 application to access the PrimesGeneration servlet and using active credential objects.The portlet will be created based on the Basic portlet type and will demonstrate the use of credentials. Once the project is created, you will execute it in the WebSphere Portal V5.1 Test Environment.

21.3.1 Creating the Credential Vault portlet application

For example, to create the new portlet project, follow these steps:

1. Select **File** → **New** → **Project**.
2. Select **Portlet Project (JSR 168)**.
3. Enter the project name CredentialJSR and select WebSphere Portal V5.1 as target server.

   Note: If required, click the **Show Advanced** button to see the complete window.
4. Click **Next**.

Archived

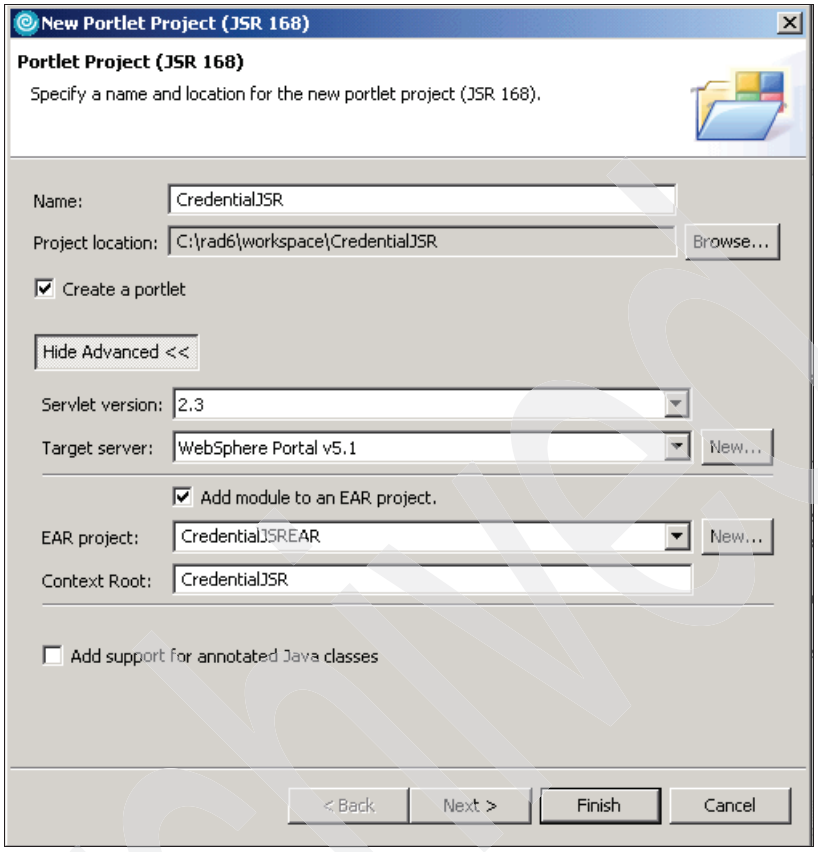


Figure 21-8 New Portlet Project JSR 168

5. In the Portlet Type window, select **Basic portlet (JSR 168)**. Click **Next**.
6. In the Features window, uncheck all the features and click **Next**.
7. In the Portlet Settings window, accept all the default values. Click **Next**.
8. In Actions and Preferences, uncheck **Add form sample** in Portlet action handling section so that only **Add action request handler** is checked. Click **Next**.
9. In the Single Sign-On window, check **Add credential vault handling** and enter the slot name `SecureCredentialSlot` as illustrated in Figure 21-9 on page 650. Click **Next**.

Archive

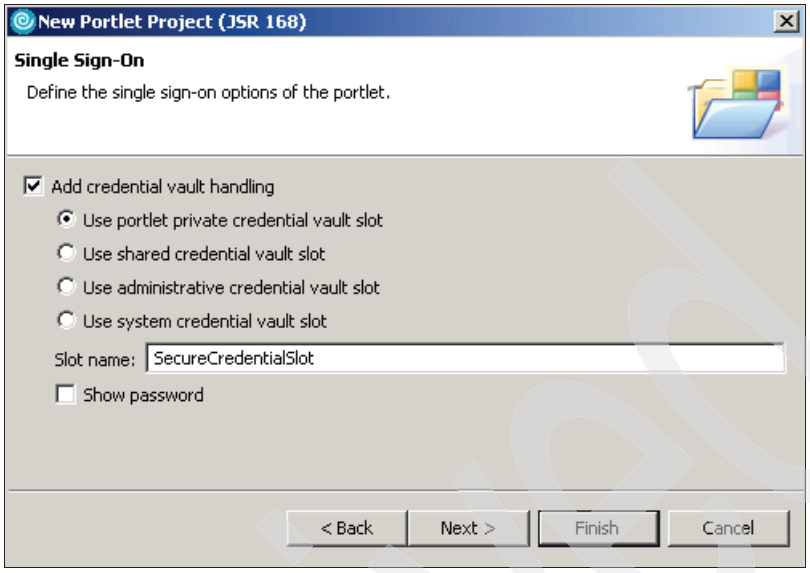


Figure 21-9 Single Sign-On page

10. In the Miscellaneous window, check **Add Edit mode** and **Add configure mode**. In Edit mode, you will enter the credentials to be stored and in configure mode you will establish the settings that will be needed to gain access to the secure servlet running in a different application server. Click **Finish** to generate the JSR 168 portlet.

Archived

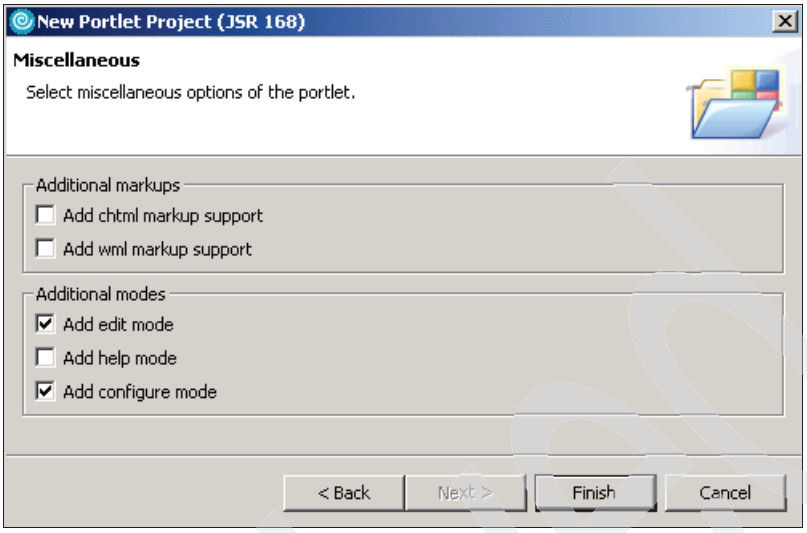


Figure 21-10 Additional modes of the new portlet

21.3.2 Reviewing the generated code

Before the portlet code is modified to access the secure portlet, let's examine the wizard generated code. Expand **CredentialJSR** → **Java Resources** → **JavaSource** to access the `credentialjsr` package. In this package, in addition to the portlet and bean classes, you will find the generated CredentialJSRPortletSecretManager class. This class is responsible for initializing the Credential Vault service and administering the credentials.

Archive

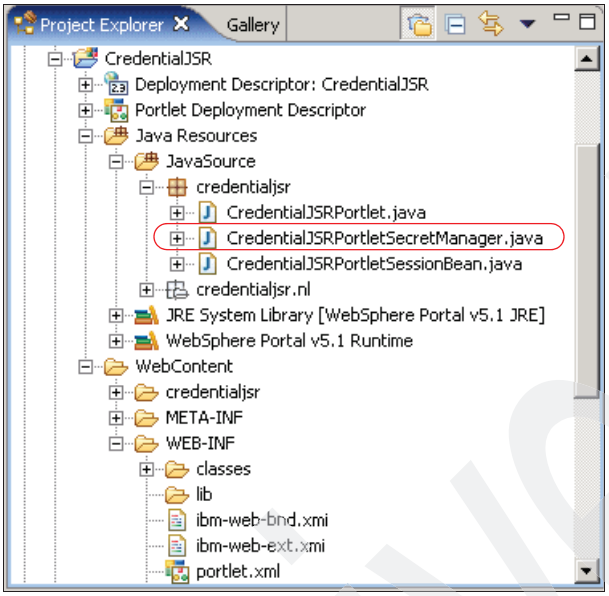


Figure 21-11 Reviewing CredentialJSRSecretManager class

Note: In an early release the CredentialJSRPortletSecretManager class generated by the wizard had an error. If this is your case, apply the required fixes or just replace the following import statement:

```
import com.ibm.wps.portletservice.credentialvault.CredentialSlotConfig;
```

with this one:

```
import com.ibm.portal.portlet.service.credentialvault.CredentialSlotConfig;
```

The following methods are provided in this class to handle Credential Vault issues:

- The init() method of this class initializes the vaultService data member.
- getCredential() returns the user name and password by using a string buffer.
- setCredential() sets the user name and password.
- getSlotId() returns the ID of the slot. Depending on the type of slot, this method uses Portlet Preferences or VaultService to get the ID.
- New slots are created in the createNewSlot() method.
- getPrincipalFromSubject() retrieves the specified Principal from the provided subject.

Archived

- isWritable() checks whether the userid and password can be saved.

The wizard has also created an input form for a user ID and password in the CredentialJSRPortletEdit.jsp. As previously described, when clicking the **Save** button, the processAction() method is called. This method retrieves the user ID and password from the form and uses the secret manager class to set the credentials.

The current version of the doView method retrieves the user credentials from the secret manager and displays them in the associated JSP. Because, in this sample scenario, you want to include the content of the secured PrimesGeneration servlet, you will replace this method in the next section of this scenario.

21.3.3 Updating the generated portlet

Modify the portlet application as follows:

1. Open the portlet deployment descriptor.
2. Select the Portlets panel and click **CredentialJSR**.
3. In Persistent Preference Store click **Add...** to enter new preferences.
4. In new Preference window, enter `.CredentialPortletServerKey` in the text field. Click **Add** and enter `localhost` in values field. Check **Read only**, this value will be modified only by portal administrators in configuration mode:

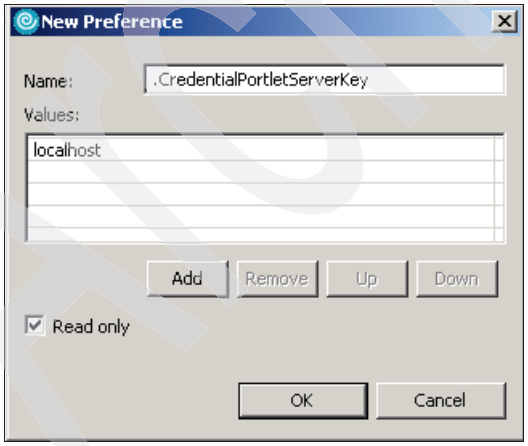


Figure 21-12 New preference

5. Click **OK** and repeat the same to add the following preferences checking Read only in all of them:

Archived

a. Name: `.CredentialPortletPortKey`, value: `9080`.

b. Name: `.CredentialPortletPathKey`, value: `/SecureServlet/PrimesGeneration`

c. Name: `.CredentialPortletAttrKey`, value: `number`

See the updated portlet deployment descriptor with the new preferences as shown in Figure 21-13.

6. Save and close the file.

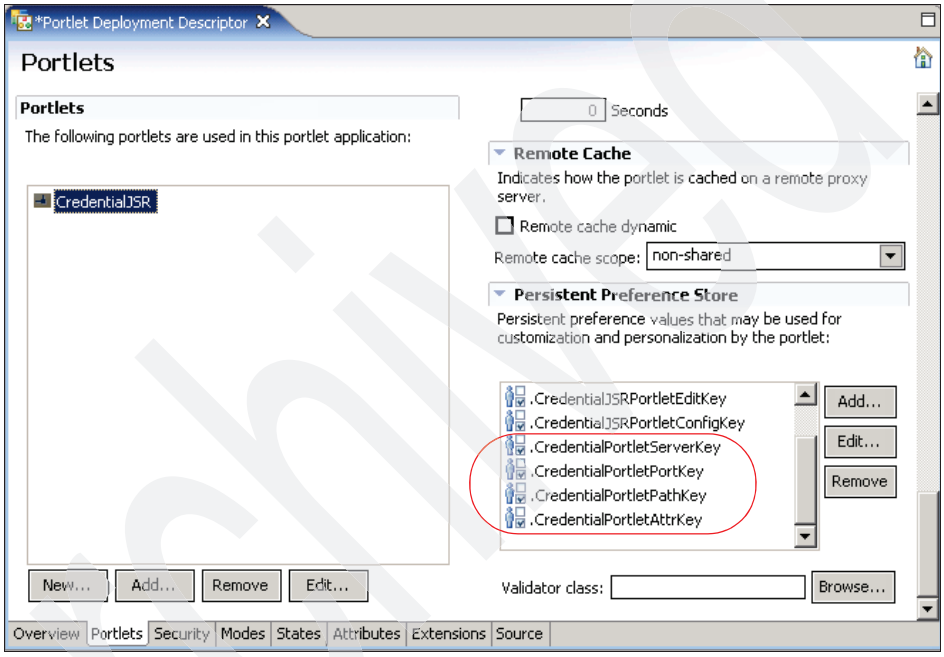


Figure 21-13 Updating Portlet Deployment Descriptor

7. Open CredentialJSRPortletSecretManager.java from the credentialjsr package.

8. Using the Java editor, add the method shown in Example 21-1 to the class.

Note: The getConnectionUsingActiveObject method returns an http connection.

Example 21-1 getConnectionUsingActiveObject method (active credentials)

```
public static HttpURLConnection getConnectionUsingActiveObject(
      PortletRequest portletRequest,
      CredentialJSRPortletSessionBean sessionBean,
      String host,
```

Archived

```
        String port,
        String path) {
        HttpURLConnection connection = null;
        try {
           URL urlSpec = new URL("http://" + host + ":" + port + path);
           String slotId = getSlotId(portletRequest, sessionBean, false);
           if (slotId != null) {
              HttpBasicAuthCredential credential =
                 (HttpBasicAuthCredential) vaultService.getCredential(
                    slotId,
                    "HttpBasicAuth",
                    new HashMap(),
                    portletRequest);

              connection = credential.getAuthenticatedConnection(urlSpec);
           }
        } catch (Exception e) {
           e.printStackTrace();
        }
        return connection;
     }
```

9. Some error codes appear because the required import statements are missing. To fix these errors, right-click the Java editor and select **Source** → **Organize Imports**.
10. In the Organize Imports dialog, choose the following:
    a. java.net.HttpURLConnection. Click **Next**.
    b. java.net.URL. Click **Next**.
    c. com.ibm.portal.portlet.service.credentialvault.credentials.HttpBasicAuthCredential. Click **Finish** to close the Organize Imports dialog.
11. Save and close the Java file.
12. Open the class CredentialJSRPortlet from the credentialjsr package.
13. Add the variables shown in Example 21-2.

Example 21-2 Define new variables in CredentialJSRPortlet class

```
   public static final String SERVER_KEY  = ".CredentialPortletServerKey";
// Key for the portlet preferences
   public static final String SERVER_TEXT  = "CredentialPortletServerText";
// Parameter name for the server text input
   public static final String PORT_KEY  = ".CredentialPortletPortKey";   //
Key for the portlet preferences
   public static final String PORT_TEXT  = "CredentialPortletPortText";   //
Parameter name for the port number text input
```

Archived

```
   public static final String PATH_KEY    = ".CredentialPortletPathKey";    //
Key for the portlet preferences
   public static final String PATH_TEXT    = "CredentialPortletPathText";   //
Parameter name for the servlet path text input
   public static final String ATTR_KEY    = ".CredentialPortletAttrKey";   //
Key for the portlet preferences
   public static final String ATTR_TEXT    = "CredentialPortletAttrText";   //
Parameter name for the servlet attribute text input
   public static final String NUMBER = "CredentialPortletFormNumber";   //
Parameter name for the number of digits text input
   public static final String NUMBER_SUBMIT =
"CredentialPortletGeneratePrime";   // Action name for prime number generation
submit form
   private String nDigits = null;
```

14. Next, you will modify the processAction method to store the preferences values that portal administrators could change when running in custom configuration mode.

    You will also add code to handle the new event when the user enters a number of digits and submits to generate a prime number by invoking the secure PrimesGeneration servlet.

    The new code to accomplish this is highlighted in Example 21-3.

Example 21-3 processAction method

```
      if( request.getParameter(CONFIG_SUBMIT) != null ) {
         PortletPreferences prefs = request.getPreferences();
         try {
            //prefs.setValue(CONFIG_KEY,request.getParameter(CONFIG_TEXT));
            prefs.setValue(SERVER_KEY, request.getParameter(SERVER_TEXT));
            prefs.setValue(PORT_KEY, request.getParameter(PORT_TEXT));
            prefs.setValue(PATH_KEY, request.getParameter(PATH_TEXT));
            prefs.setValue(ATTR_KEY, request.getParameter(ATTR_TEXT));
            prefs.store();
         }
         catch( ReadOnlyException roe ) {
         }
         catch( ValidatorException ve ) {
         }
      }
      if ( request.getParameter(NUMBER_SUBMIT) != null ) {
         // Get the NUMBER field
         if ( request.getParameter(NUMBER) != null &&
!request.getParameter(NUMBER).equalsIgnoreCase("") ) {
            nDigits = request.getParameter(NUMBER);
         }
      }
```

.....

15.Replace the doView () method so it looks as shown in Example 21-4.

Example 21-4 The doView method uses a Http connection from the SecretManager class

```
   public void doView(RenderRequest request, RenderResponse response) throws
PortletException, IOException {
       // Set the MIME type for the render response
       response.setContentType(request.getResponseContentType());
       // Check if portlet session exists
       CredentialJSRPortletSessionBean sessionBean = getSessionBean(request);
       if( sessionBean==null ) {
          response.getWriter().println("<b>NO PORTLET SESSION YET</b>");
          return;
       }
        // Retrieve user credentials
      StringBuffer userId = new StringBuffer("");
      StringBuffer password = new StringBuffer("");
      try {

CredentialJSRPortletSecretManager.getCredential(request,sessionBean,userId,
password);
      }
      catch( Exception e ) {
       getPortletContext().log("Exception on
CredentialJSRPortletSecretManager.getCredential(): "+e.getMessage());
      }
      // Portlet should use userId/password to log in to the backend systems on
behalf of the user.
      // Show curent userId/password on the portal page at this time.

       // Set current userId/password in the request attribute
       request.setAttribute(USERID,userId.toString());
      request.setAttribute(PASSWORD,password.toString());

      if (nDigits != null ) {
         StringBuffer result = new StringBuffer();

         try {
            PortletPreferences prefs = request.getPreferences();
            String host = prefs.getValue(SERVER_KEY, "");
            String port = prefs.getValue(PORT_KEY, "");
            StringBuffer path = new StringBuffer();
            path.append(prefs.getValue(PATH_KEY, ""));
            path.append("?");
            path.append(prefs.getValue(ATTR_KEY, ""));
            path.append("=");
            path.append(nDigits);
```

Archived

```
                System.out.println("Using Active Credentials");
                HttpURLConnection connection =

CredentialJSRPortletSecretManager.getConnectionUsingActiveObject(
                      request,
                      sessionBean,
                      host,
                      port,
                      path.toString());

                if (connection != null) {
                   connection.connect();
                   String responseMessage = connection.getResponseMessage();
                   int responseCode = connection.getResponseCode();
                   // Were we successful?
                   if (HttpURLConnection.HTTP_OK == responseCode) {
                      result.append("<H4>Successfully connected!</H4>");
                   } else {
                      result.append(
                         "<P>Unable to successfully connect to back-end server."
                            + ", HTTP Response Code = "
                            + responseCode
                            + ", HTTP Response Message = \""
                            + responseMessage
                            + "\"</P>");
                      }
                   BufferedReader br =
                      new BufferedReader(
                         new InputStreamReader(connection.getInputStream()));
                   String line;
                   while ((line = br.readLine()) != null)
                      result.append(line + "\n");
                } else {
                   result.append(
                      "<h2>Credential not found. Please set it in edit mode!
</h2>");
                   }
             } catch (IOException exc) {
                result.append(
                   "<h2>Single-sign-on error, login at back-end failed! </h2>");
                }
              // Set the result in the request attribute
              request.setAttribute("result",result.toString());
      }
          // Invoke the JSP to render
          PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher(getJspFilePath(request, VIEW_JSP));
          rd.include(request,response);
          nDigits = null;
```

```
}
```

16. You will need to organize the import statements again as you did previously.
17. Save and close the Java file.
18. Open CredentialJSRPortletConfig.jsp file under credentialjsr/jsp/html folder. Modify the form to display the default values of the preference variables you created before as illustrated in Figure 21-13 on page 654. The entire new JSP code is shown in Example 21-5.

Example 21-5 CredentialJSRPortletConfig.jsp

```
<%@ page session="false" contentType="text/html"
import="javax.portlet.*,credentialjsr.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Write to the PortletPreferences</H3>

<DIV style="margin: 12px; margin-bottom: 36px">
<% /******** Start of sample code ********/ %>
<%
  PortletPreferences preferences = renderRequest.getPreferences();
  if( preferences!=null ) {
    String server =
(String)preferences.getValue(CredentialJSRPortlet.SERVER_KEY,"");
    String port =
(String)preferences.getValue(CredentialJSRPortlet.PORT_KEY,"");
    String path =
(String)preferences.getValue(CredentialJSRPortlet.PATH_KEY,"");
    String attr =
(String)preferences.getValue(CredentialJSRPortlet.ATTR_KEY,"");
%>
  <FORM ACTION="<portlet:actionURL/>" METHOD="POST">
   <TABLE cellspacing="3" cellpadding="3" border="0">
   <tr>
      <td><LABEL for="<%=CredentialJSRPortlet.SERVER_TEXT%>">Server
name:</LABEL></td>
      <td><INPUT name="<%=CredentialJSRPortlet.SERVER_TEXT%>"
value="<%=server%>" type="text"/></td>
   </tr>
   <tr>
      <td><LABEL for="<%=CredentialJSRPortlet.PORT_TEXT%>">Port
number:</LABEL></td>
```

Archived

```
      <td><INPUT name="<%=CredentialJSRPortlet.PORT_TEXT%>" value="<%=port%>"
type="text"/></td>
   </tr>
   <tr>
      <td><LABEL for="<%=CredentialJSRPortlet.PATH_TEXT%>">Servlet
path:</LABEL></td>
      <TD><INPUT name="<%=CredentialJSRPortlet.PATH_TEXT%>" value="<%=path%>"
type="text" size="50"/></TD>
   </tr>
   <tr>
      <td><LABEL for="<%=CredentialJSRPortlet.ATTR_TEXT%>">Servlet attribute
name:</LABEL></td>
      <td><INPUT name="<%=CredentialJSRPortlet.ATTR_TEXT%>" value="<%=attr%>"
type="text"/></td>
   </tr>
   <tr>
      <td colspan="2"><INPUT name="<%=CredentialJSRPortlet.CONFIG_SUBMIT%>"
value="Save" type="submit"/></td>
   </tr>
   </TABLE>
  </FORM>
<%
  }
else {
  %>Error: PortletPreferences is null.<%
  }
%>
<% /******** End of sample code *********/ %>
</DIV>
</DIV>
```

19. Save and close the jsp file.

20. Finally, open CredentialJSRPortletView.jsp and add a new form when there are user credentials in the credential vault. For clarity, you will also modify the message to be displayed when there are no credentials found in the vault. Example 21-6 shows the new code.

Example 21-6 New form in updated CredentialJSRPortletView.jsp

```
<%@ page session="false" contentType="text/html"
import="java.util.*,javax.portlet.*,credentialjsr.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>

<%
   CredentialJSRPortletSessionBean sessionBean =
(CredentialJSRPortletSessionBean)renderRequest.getPortletSession().getAttribute
(CredentialJSRPortlet.SESSION_BEAN);
```

```
%>

<DIV style="margin: 6px">

<% if ( renderRequest.getAttribute("result") != null ) { %>
   <%=(String)renderRequest.getAttribute("result") %>
<% } %>

<H3 style="margin-bottom: 3px">Single sign-on sample</H3>
The following user credentials have been retrieved from the credential
vault.<BR>
Portlet should use these credentials to log in to the backend systems on behalf
of the user.
<DIV style="margin: 12px; margin-bottom: 36px">
<% /******** Start of sample code ********/ %>

  <%
  int secretType = sessionBean.getSecretType();
  String secretTypeName = "Unknown";
  switch (secretType) {
    case CredentialJSRPortletSecretManager.SECRET_PORTLET_PRIVATE_SLOT:
      secretTypeName = "Portlet Private Slot";
      break;
    case CredentialJSRPortletSecretManager.SECRET_SHARED_SLOT:
      secretTypeName = "Shared Slot";
      break;
    case CredentialJSRPortletSecretManager.SECRET_ADMINISTRATIVE_SLOT:
      secretTypeName = "Administrative Slot";
      break;
    case CredentialJSRPortletSecretManager.SECRET_SYSTEM_SLOT:
      secretTypeName = "System Slot";
      break;
    case CredentialJSRPortletSecretManager.SECRET_JAAS_SUBJECT:
      secretTypeName = "JAAS Subject";
      break;
  }
  %>Secret type is <B><%=secretTypeName%></B>.<BR><%
  String userId =
(String)renderRequest.getAttribute(CredentialJSRPortlet.USERID);
  if( userId.length()>0 ) {
    String password =
(String)renderRequest.getAttribute(CredentialJSRPortlet.PASSWORD);
    %>
    User id is <b><%=userId%></b>.
   <H3>Generate a prime number</H3>
   <FORM ACTION="<portlet:actionURL/>" METHOD="POST">
      <LABEL  for="<%=CredentialJSRPortlet.NUMBER%>">Enter the number of
digits:</LABEL>
      <INPUT name="<%=CredentialJSRPortlet.NUMBER%>" type="text">
```

Archived

```
      <INPUT name="<%=CredentialJSRPortlet.NUMBER_SUBMIT%>" value="Send"
type="submit"/>
    </FORM>

    <%
  }
  else {
    %><H5>There are no credentials in the vault for this portlet.<%
    if( CredentialJSRPortletSecretManager.isWritable(sessionBean) ) {
      %> Use <B>edit mode</B>.</H5><%
    }
  }
  %>

<% /******** End of sample code *********/ %>
</DIV>

</DIV>
```

21. Save and close the view JSP file.

21.3.4 Running the portlet

In this section, you will run the portlet using active credentials to access the back-end resource, a protected servlet in this case.

1. Right-click CredentialJSR portlet application and select **Run** → **Run on Server.**
2. Choose or manually define a WebSphere Portal V5.1 Test Environment.
3. Click **Finish** to start the server.
4. After few minutes, you will see a browser displaying the portlet content in View mode.

Archived

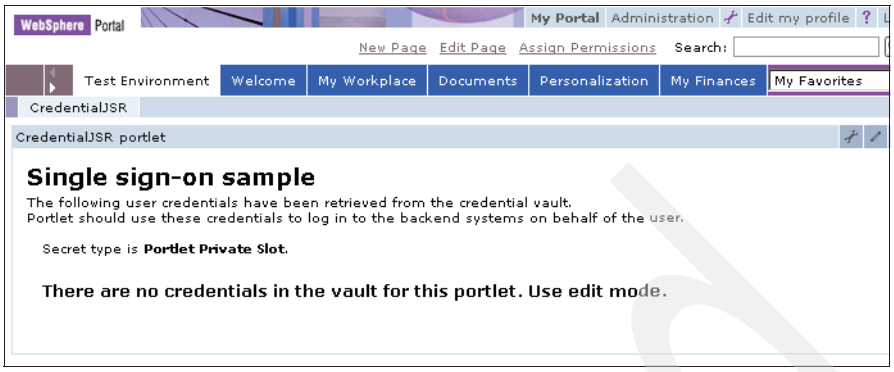


Figure 21-14 Portlet View mode display

Note: The portlet has executed the init method to initialize the Credential Vault Service and the doView method. Since there are no credentials yet, a message is displayed to indicate that the user should switch to Edit mode and enter the required credentials. That is, the user ID and password in this sample scenario.

5. Switch the portlet to Edit mode.

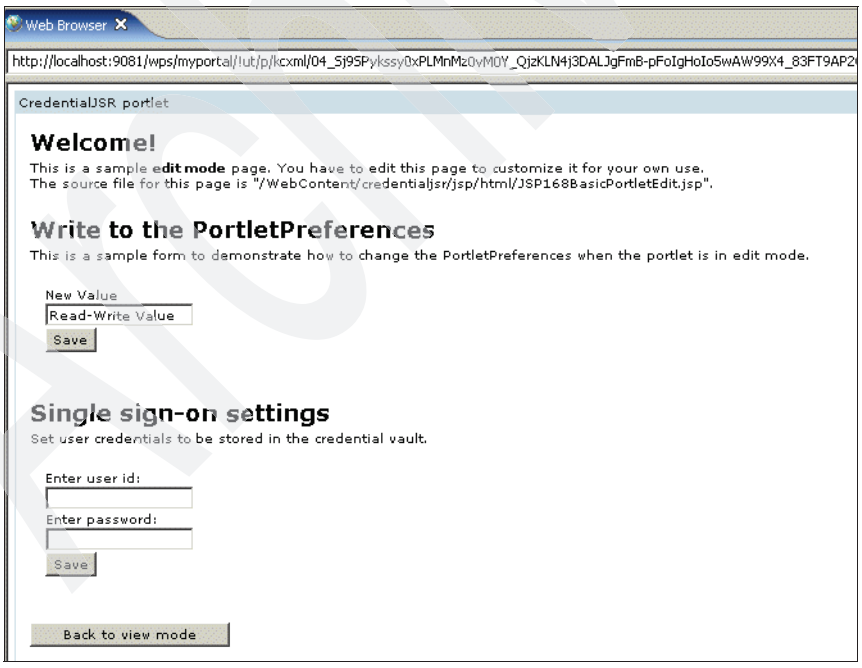


Figure 21-15 Portlet View mode display

Archived

6. Optionally, you may want to remove the top portion of portlet Edit mode display in the CredentialJSRPortletEdit.jsp page. Skip this step if you do not want to do this.

Example 21-7 Updated CredentialJSRPortletEdit.jsp

```
<%@ page session="false" contentType="text/html"
import="java.util.*,javax.portlet.*,credentialjsr.*"%>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>

<%
   CredentialJSRPortletSessionBean sessionBean =
(CredentialJSRPortletSessionBean)renderRequest.getPortletSession().getAttribute
(CredentialJSRPortlet.SESSION_BEAN);
%>

<DIV style="margin: 6px">

<H3 style="margin-bottom: 3px">Single sign-on settings</H3>
<% if( CredentialJSRPortletSecretManager.isWritable(sessionBean) ) {
  /******** Start of sample code ********/
  String userId =
(String)renderRequest.getAttribute(CredentialJSRPortlet.USERID);
  String password =
(String)renderRequest.getAttribute(CredentialJSRPortlet.PASSWORD);
  %>Set user credentials to be stored in the credential vault.
  <DIV style="margin: 12px; margin-bottom: 36px">

   <FORM ACTION="<portlet:actionURL/>" METHOD="POST">
      <LABEL  for="<%=CredentialJSRPortlet.USERID%>">Enter user id:</LABEL><BR>
      <INPUT name="<%=CredentialJSRPortlet.USERID%>" value="<%=userId%>"
type="text"><BR>
      <LABEL  for="<%=CredentialJSRPortlet.PASSWORD%>">Enter
password:</LABEL><BR>
      <INPUT name="<%=CredentialJSRPortlet.PASSWORD%>" value="<%=password%>"
type="password"><BR>
      <INPUT name="<%=CredentialJSRPortlet.USER_SUBMIT%>" value="Save"
type="submit"/>
    </FORM>

  <% /******** End of sample code *********/ %>
  </DIV>
  <%
}
else {
  %>This credential vault is read-only.<%
}
%>
```

Archived

```
<FORM ACTION="<portlet:renderURL portletMode="view"/>" METHOD="POST">
   <INPUT NAME="back" TYPE="submit" VALUE="Back to view mode">
</FORM>
</DIV>
```

7. Enter the following information:
   - User ID field: `user1`
   - Password field: `password1`



Figure 21-16 Running Edit mode of the portlet

8. Press the **Save** button. This will generate an action that will be checked by the processAction() method in the CredentialJSRPortlet class. The portlet returns to Edit mode, showing the credentials you entered. Click the **Back to view mode** button to return the portlet to View mode.
9. Make sure the WebSphere Test Environment running the secure servlet is running. Start this server if it is not running.
10. Since you are also running as administrator, switch to configure mode. You will find the default values stored in the PortletPreferences to access the back-end resources, check them out and return to View mode. The correct values should be:
    - Server name: `localhost`
    - Port number: `9080`
    - Servlet path: `/SecureServlet/PrimesGeneration`
    - Servlet attribute name: `number`.

Archived

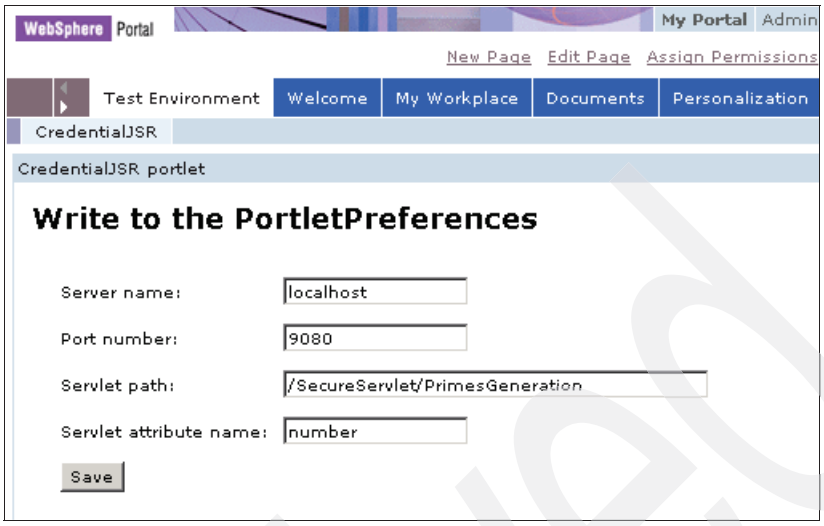


Figure 21-17 Portlet configure mode

11. Go back to View mode. Now the portlet displays a form to enter the number of digits to generate a prime number with this length. In View mode, enter a number of digits and click **Send** button. At the top of the portlet you will see the result generated by the PrimesGeneration servlet.

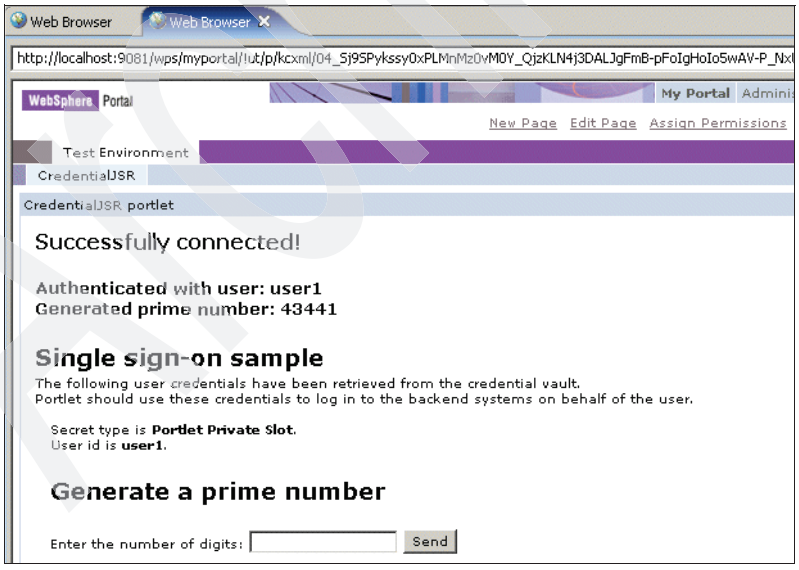


Figure 21-18 The CredentialJSR portlet in action

21.4 Using passive credentials

In the previous section, a portlet using an active credential object was built. Although this is the preferred type of credential object, there are certain cases where you have to use passive credential objects, for example when an appropriate active credential class is not available.

In this sample scenario, the portlet will be changed to use a passive credential object. To modify the portlet application, proceed as follows:

1. In the CredentialJSR project, open the CredentialJSRPortletSecretManager class.
2. Using the Java editor, add to the class the method shown in Example 21-8 to support passive credentials.

Example 21-8 The getConnectionUsingPassiveObject method (passive credentials)

```
public static HttpURLConnection getConnectionUsingPassiveObject(
       PortletRequest portletRequest,
       CredentialJSRPortletSessionBean sessionBean,
       String host,
       String port,
       String path) {
       StringBuffer userid = new StringBuffer("");
       StringBuffer password = new StringBuffer("");
       HttpURLConnection connection = null;
       try {
          getCredential(portletRequest, sessionBean, userid, password);
          if (!userid.toString().equals("")) {
             String userAndPassword =
                new String(userid.toString() + ":" + password.toString());
             byte[] userAndPasswordBytes = userAndPassword.getBytes();
             BASE64Encoder encoder = new BASE64Encoder();
             String basicAuth =
                new String(encoder.encode(userAndPasswordBytes));
             basicAuth = "Basic " + basicAuth;
             URL url = new URL("http://" + host + ":" + port + path);
             connection = (HttpURLConnection) url.openConnection();
             connection.setRequestProperty("authorization", basicAuth);
          }
       } catch (Exception e) {
          e.printStackTrace();
       }
       return connection;
    }
```

3. Organize the import statements as you did in the previous scenario. Choose com.ibm.misc.BASE64Encoder. Click **Finish**.

Note: The Sun.misc.Bas64Encoder class can also be used. In some specific cases this can be the correct choice.

4. Save and close the Java file.
5. Open the class CredentialJSRPortlet from the credentialjsr package. In the doView method, change the line:

```
HttpURLConnection connection =
CredentialJSRPortletSecretManager.getConnectionUsingActiveObject(request,
sessionBean, host, port, path.toString());
```

   to

```
HttpURLConnection connection =
CredentialJSRPortletSecretManager.getConnectionUsingPassiveObject(request,
sessionBean, host, port, path.toString());
```

6. Save and close the Java file.
7. Restart the project and access the secure servlet again as described in 21.3.4, “Running the portlet” on page 662. The portlet should run exactly as it did in the previous section when using active credentials. This time, the getConnectionUsingPassiveObject method has access to the credentials.

Note: Having access to credentials could be a security risk, so when possible, use always active credential objects.

Archived

22

Accessing JDBC databases from portlet applications

In this chapter, we introduce the process of gaining access to back-end JDBC databases from portlet applications. A simple portlet project is included to show how portlet applications access relational databases using the JDBC interface.

This chapter discusses the following topics:

- How to create a Portlet project on Rational Application Developer.
- How to create a connection to a database using Rational Application Developer.
- A portlet example to access a database.

© Copyright IBM Corp. 2005. All rights reserved.

22.1 Creating a portlet project

In this section, the process of creating a portlet project named HRPortlet is described. This project will be used later in 22.2, “Creating a sample database” on page 675 to show the JDBC connection.

22.1.1 Creating HRPortlet

In this section, you will create a portlet project named HRPortlet. The portlet will be created based on a Basic portlet type using the provided wizard. This portlet will be published and executed in the Portal Test Environment.

If not already running, start the Rational Application Developer; select **Start** → **Programs** → **IBM Rational** → **IBM Rational Application Developer V6.0** → **Rational Application Developer.**

1. Select **File** → **New** → **Project...**.

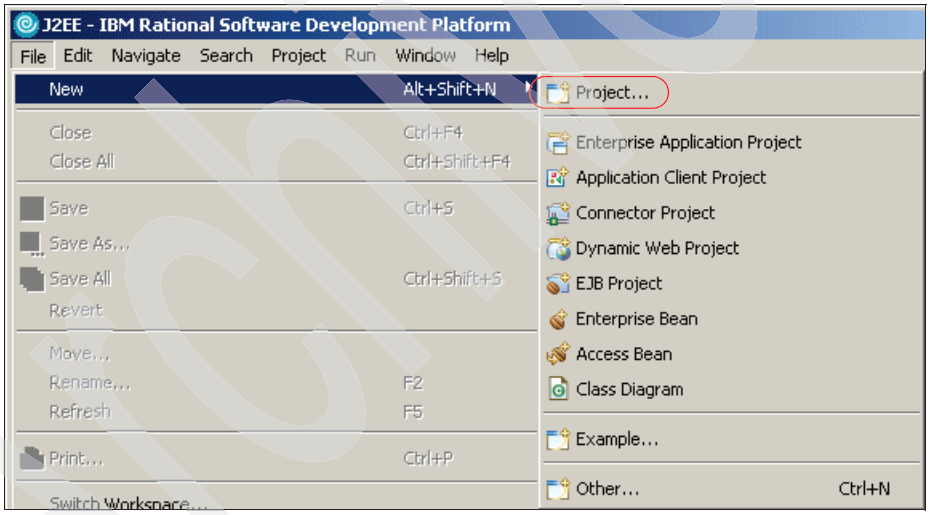


Figure 22-1 Creating a new project

2. In the *Select a wizard* window, select **Portlet Project**. Click **Next**.

Archived

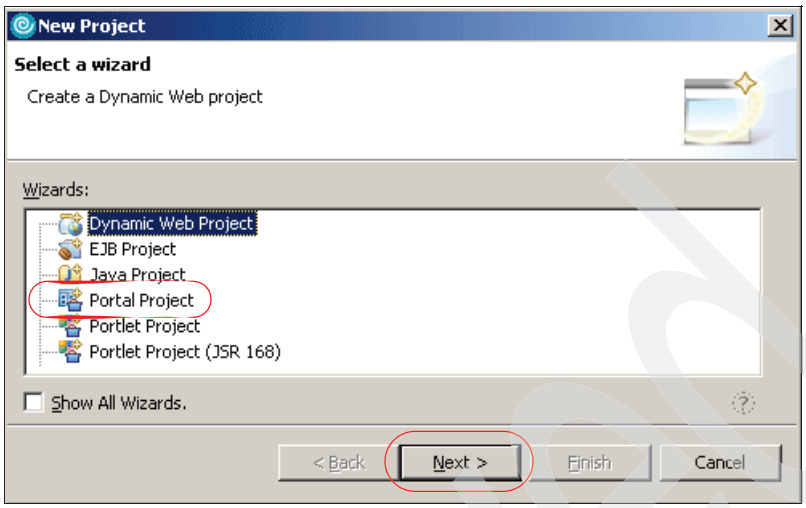


Figure 22-2 Select Portlet project

3. The Portlet Project window will appear; enter `HRPortlet` for the name of the project. Click the **Show Advanced** >> button and change Target Server to `WebSphere Portal V5.1 stub` and Context Root to `/HRPortlet`. Click **Next**.

Archived

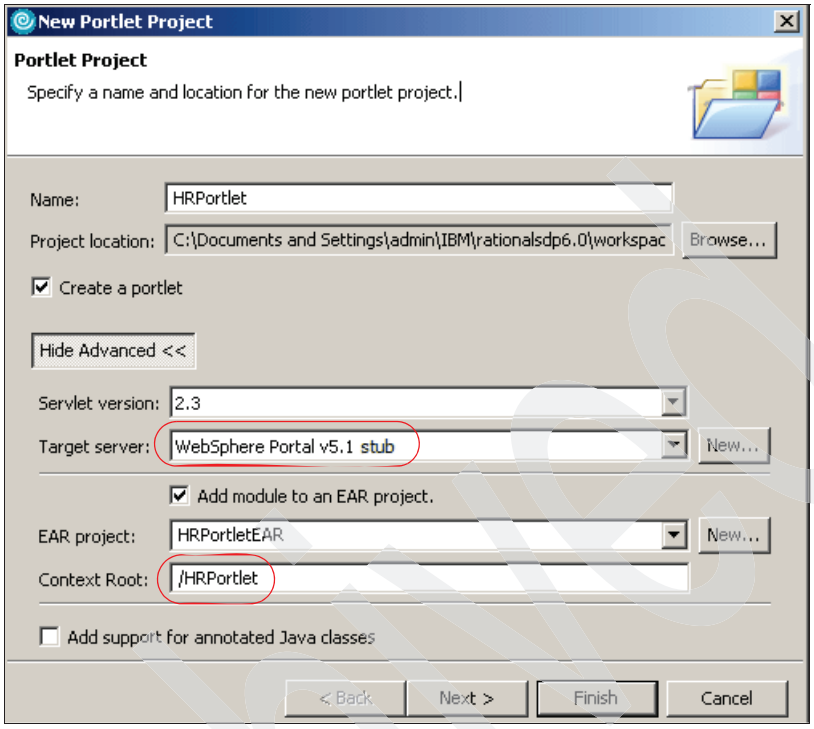


Figure 22-3 Portlet project configuration

4. In the Portlet Type window, select **Basic portlet**. Click **Next**.

Archive

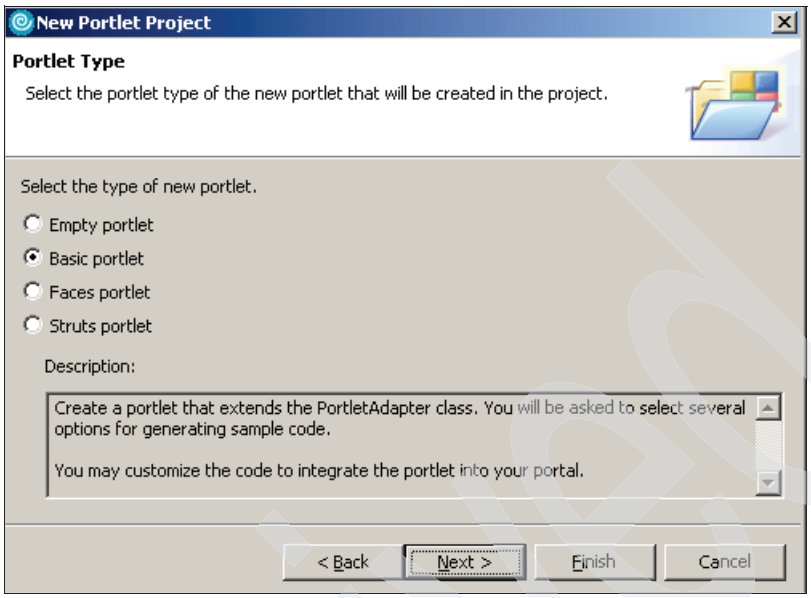


Figure 22-4 Basic portlet

5. In the Features window, leave the default values. Click **Next**.
6. In the Portlet Settings window, click the **Change code generations options** checkbox. Remove the `portlet` word from the Portlet name, Portlet Title and Class Prefix fields, so they will look as in Figure 22-5 on page 674. Click **Next**.

Archived

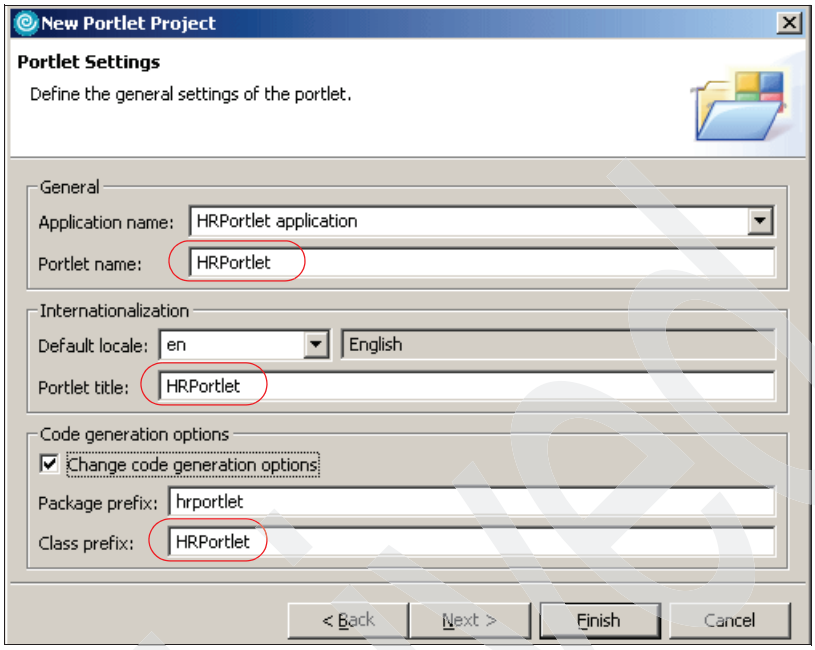


Figure 22-5 Portlet settings

7. Accept default values for the Event Handling window by clicking **Next**.
8. Accept default values for the Single Sign-on window by clicking **Next**. The Credential Vault is not used in this scenario.
9. Click **Finish** in the Miscellaneous window.

Note: If this is the first time that Rational Application Developer is used, you will be prompted to switch to the Web perspective. Click **Yes**.

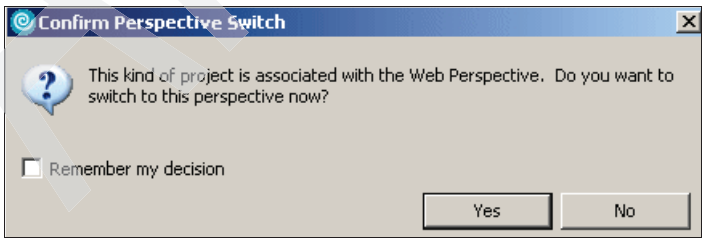


Figure 22-6 Switch to Web perspective

As a result of this process you should see, in the Project Explorer view, all the project artifacts created by Rational Application Developer, under **Dynamic Web Projects** → **HRPortlet**.

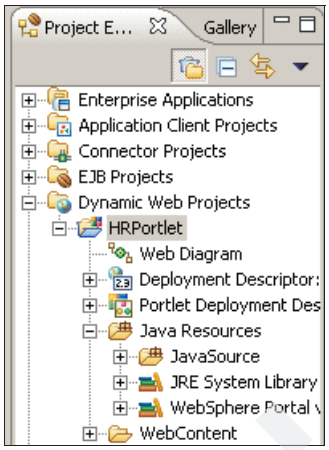


Figure 22-7 HRPortlet project

22.2 Creating a sample database

In this section, we describe the process of creating a database connection and generating the required Java classes; these classes are similar to the classes used in the sample scenario included in this book.

Note: In 22.3, “Sample scenario” on page 689, a portlet project to access a JDBC database will be imported into the HRPortlet project created in the previous section. In this implementation, the JDBC connection has already been included.

This section explains how to create a database and a database connection using the wizard provided by Rational Application Developer.

22.2.1 Creating the WSSAMPLE database

Before creating the connection, you need to create the database you are going to connect to. In this sample scenario, you will use Cloudscape embedded within Rational Application Developer.

Note: The sample scenario requires that you download the sample code available as additional materials.

Archived

1. Create the c:\HRproject\database directory.
2. Copy the following files to this directory:
   - CreateCloudTable.bat
   - Tables.sql
3. Edit the provided CreateCloudTable.bat file and make sure the paths for variables JAVA_HOME and DB2J_LIB point to the correct location. For example, in this sample scenario, these directories (java and lib) can be found at <RAD_root>\runtimes as shown in Example 22-1.

Example 22-1 Sample CreateCloudTable.bat file

```
set JAVA_HOME=C:\Progra~1\IBM\Rational\SDP\6.0\runtimes\base_v51\java
set DB2J_LIB=C:\Progra~1\IBM\Rational\SDP\6.0\runtimes\base_v51\cloudscape\lib
set
CLASSPATH=%DB2J_LIB%\db2j.jar;%DB2J_LIB%\db2jtools.jar;%DB2J_LIB%\db2jcview.jar
;%DB2J_LIB%\jh.jar
%JAVA_HOME%\bin\java -Dij.connection.myconn=jdbc:db2j:WSSAMPLE;create=true
-Dcloudscape.system.home=%DB2J_LIB% -ms16m -mx32m com.ibm.db2j.tools.ij
Tables.sql
```

4. Save and close the file.
5. Open a command window and change to the directory where these files are located.
6. Execute CreateCloudTable.bat to create and populate the sample database (WSSAMPLE) as illustrated in Figure 22-8.

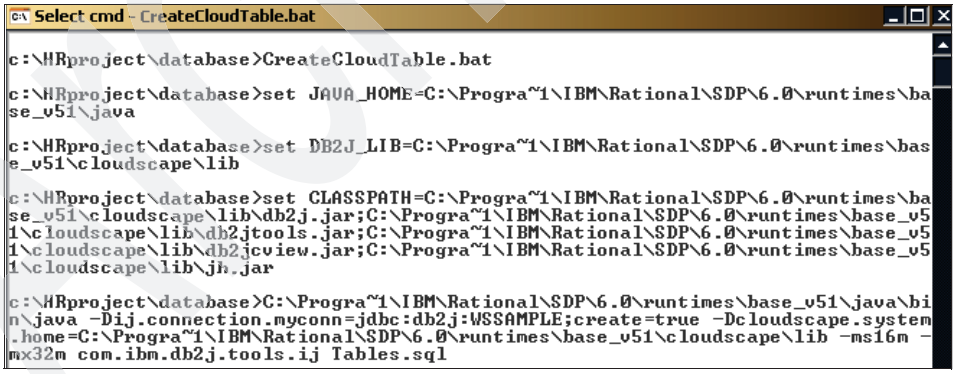


Figure 22-8 Creating and populating the Cloudscape sample database

7. After executing the batch file, the WSSAMPLE folder is created in c:\HRproject\database.

Archived

Accessing database information

As an option, you can review the sample database information. For example, execute the following steps:

1. Open a command window.
2. Go to the embedded directory located in:

   ```
   <RAD_root>\runtimes\base_51\cloudscape\bin\embedded
   ```

   For example, in this scenario the complete path is:

   ```
   C:\Progra~1\IBM\Rational\SDP\6.0\runtimes\base_v51\cloudscape\bin\embedded
   ```

3. Start Cloudview by executing the cview.bat file.
4. The Cloudview application starts. Select **File** → **Open...**.

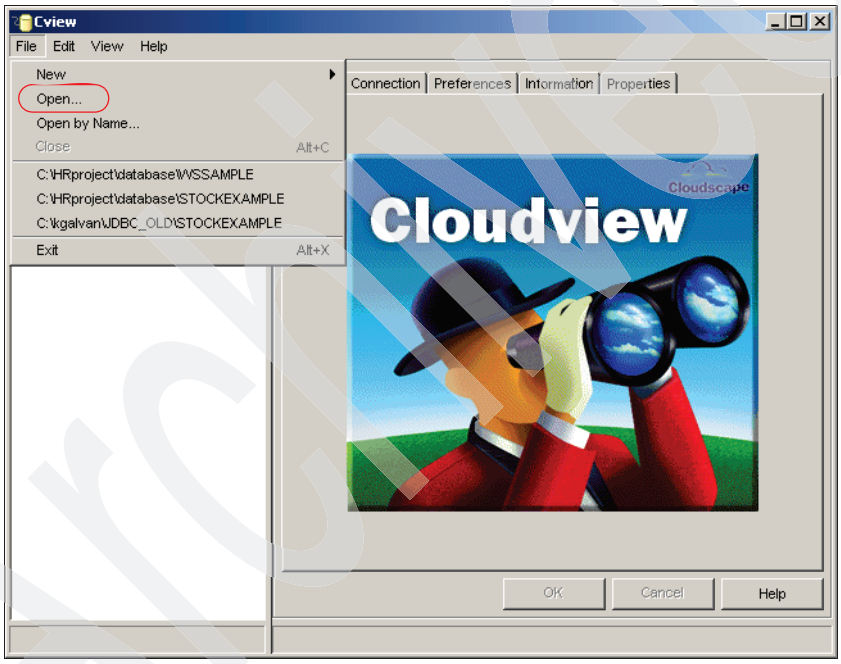


Figure 22-9 Executing Cloudview

5. In the directory c:\HRproject\database, select **WSSAMPLE**. Click **Open**.
6. You should see the tables and their content (data) by selecting them.

Archived

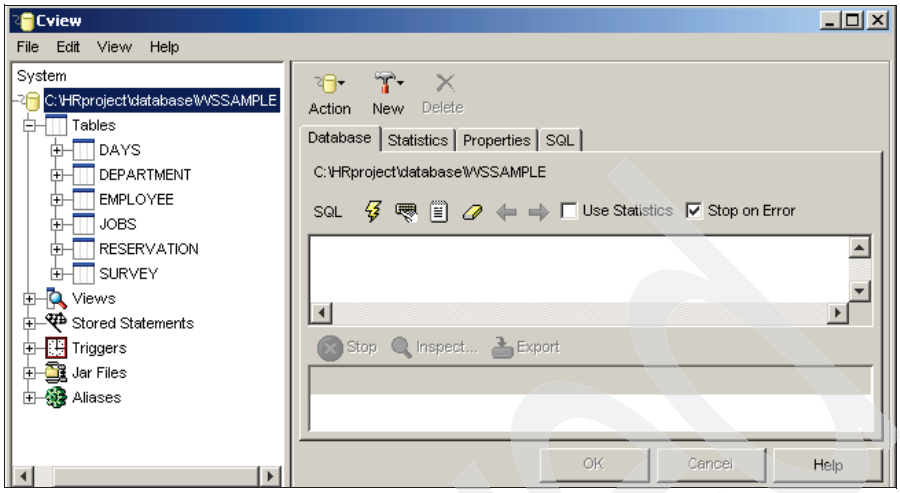


Figure 22-10 View of created sample database

7. *Click* **File** → **Exit** to end the program.

22.2.2 Creating a connection

Once the database is created, you can proceed to create the connection to the database and verify it. However, before you do this, you will need to create a folder to copy the database metadata.

Creating a database folder

On the Web perspective, execute the following steps:

1. Right-click the **WEB-INF** folder and select **New** → **Folder** from the contextual menu.

Archived

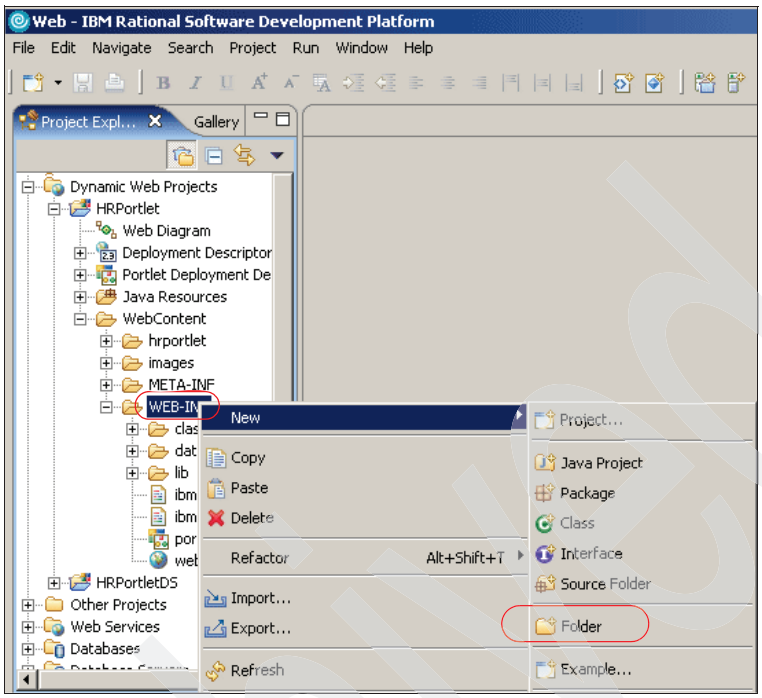


Figure 22-11 Select a new folder

2. Enter `database` as the folder name.

Archived

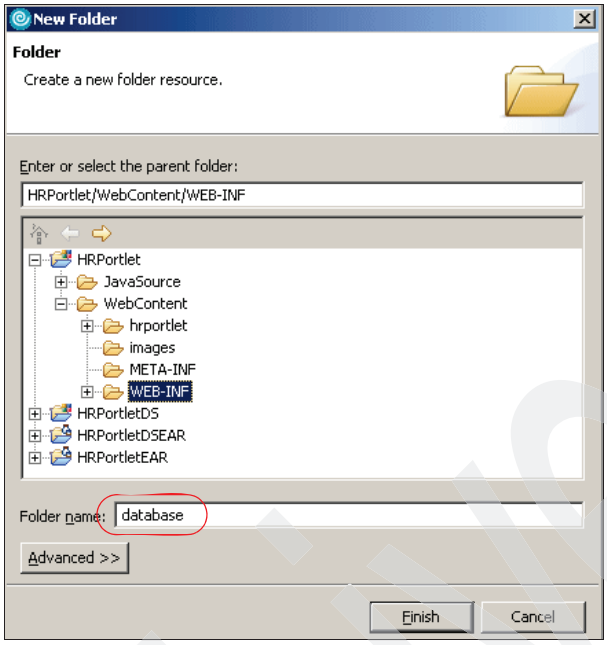


Figure 22-12 Entering a folder name

3. Click **Finish**.

Creating a new connection

Follow these steps to create a new database connection:

1. Change to the Data perspective by selecting **Window** → **Open Perspective** → **Data**.

Archived

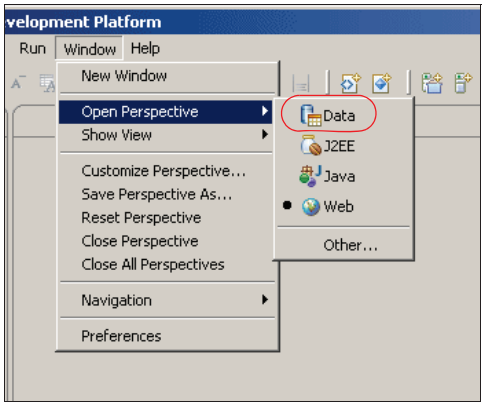


Figure 22-13 Data perspective

2. Right-click inside the Database Explorer view.

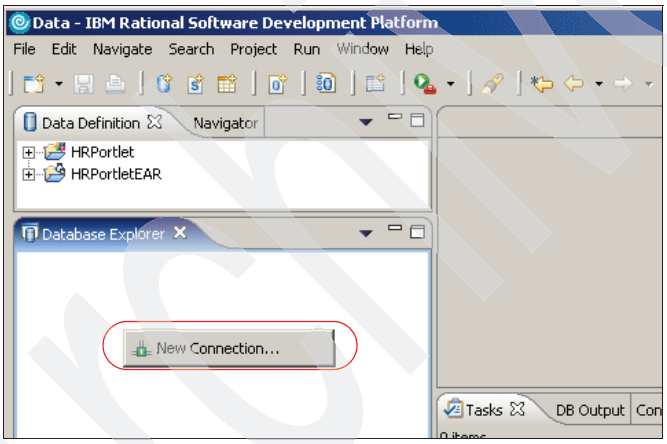


Figure 22-14 Creating a new connection

3. Click **New Connection...** from the context menu to create a new connection. The *Establish a connection to a database* window will appear.
4. Type `ConnHR` as connection name. Click **Next.**
5. In the Specify connection parameters window, select the following values:
   a. Select database manager: `V5.1`
   b. JDBC driver: `Cloudscape Embedded JDBC Driver`
   c. Enter the path where the database for this scenario was created. For example:

Archived

c:\HRProject\database\WSSAMPLE

d. Uncheck **Create the database** since you have already created the database.

e. Leave the default value for the Class location field.

f. As a user ID, enter `db2admin`. As a password, use `db2admin` also.

> **Note:** The user ID and password are optional in this sample scenario since Cloudscape does not verify it.

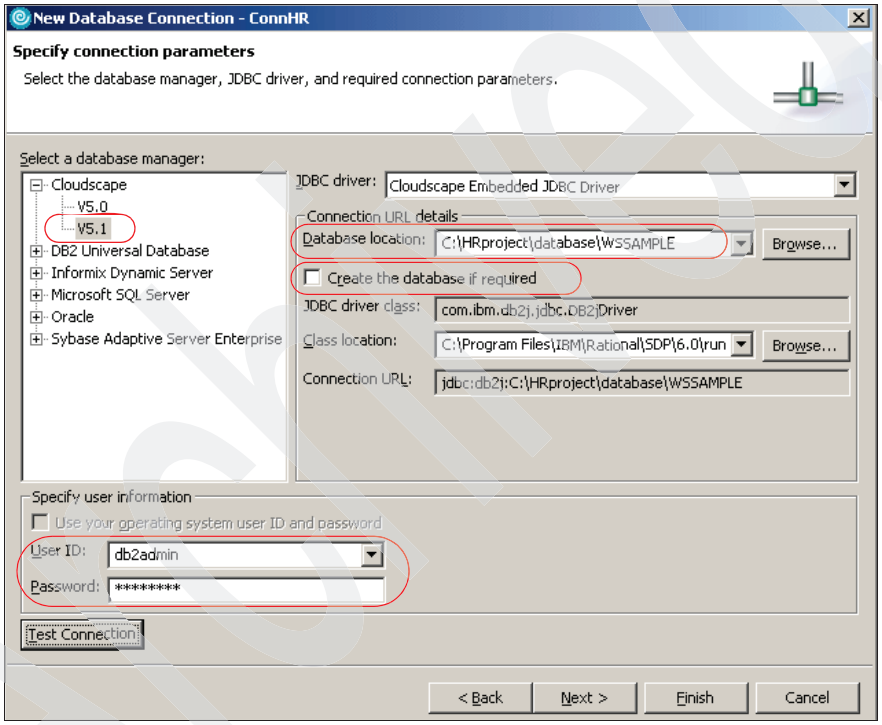


Figure 22-15 Connection parameters

6. Click the **Test Connection** button. You should see a dialog box with the message `Connection to WSSAMPLE is successful`. Click **OK**.

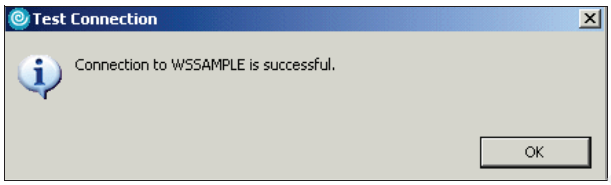


Figure 22-16 Successful database connection

Note: If you do not receive the successful message, make sure the values you entered in the Specify connection parameters window are correct and try again until you get a successful message indication.

7. Click **Finish**.
8. Click **Yes** to the question `Do you want to copy the database metadata to a project folder?`.
9. In the Copy to Project window, click the **Browse** button and select the database folder you previously created:

   /HRPortlet/WebContent/WEB-INF/database

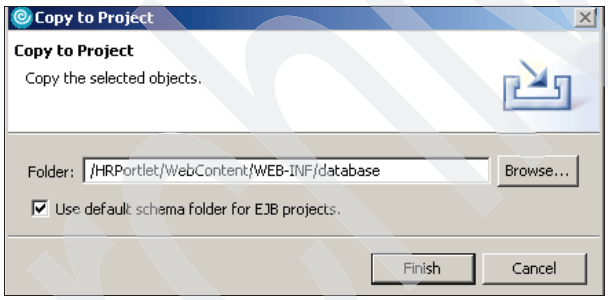


Figure 22-17 Copy objects

10. Click **Finish**.

22.2.3 Creating an SQL statement

In this section, a simple SQL statement to display all employees will be created. When the statement has been created, the editor opens and you have several options to create a SQL statement. For example, you can use the wizard or you can use the editor to write your query. In this section, you will use the wizard to create and execute the following SQL statement:

```
select * from employee
```

Archived

1. In the Data Definition view, right-click the statement folder under the sample WSSAMPLE database.
2. Select **New** → **Select Statement.**

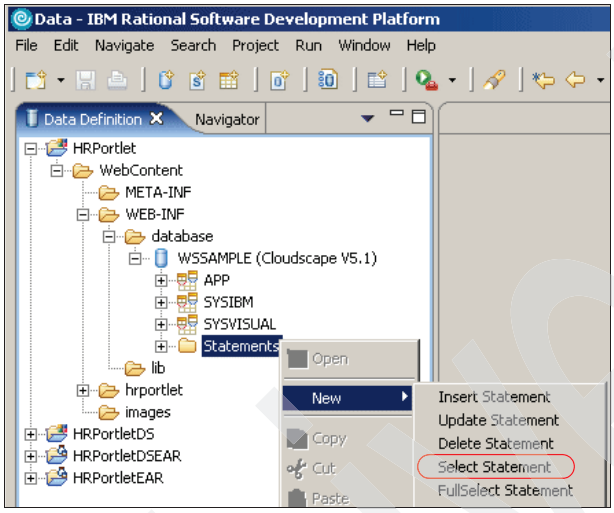


Figure 22-18 Creating the SWL select statement

3. Enter `SQLUtility` for the Statement Name and click **OK**.
4. The Editor will open; right-click the Tables area.
5. Select **Add Table...** on the context window.
6. Select the **APP.EMPLOYEE** from the Table name pop-up and click **OK**.

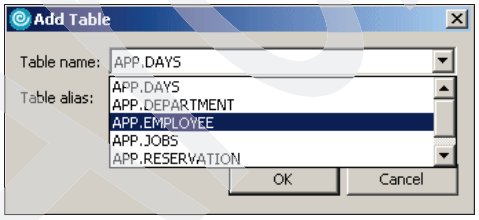


Figure 22-19 Add table

7. On the Data Definition view, expand the **Statement** folder.
8. Right-click the **SQLUtility** statement.
9. Select **Execute** from the context menu.

Archived

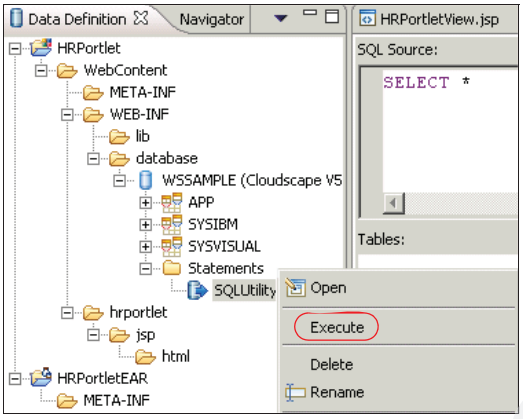


Figure 22-20 Execute SQL statement

10. This request returns the content from the EMPLOYEE table.

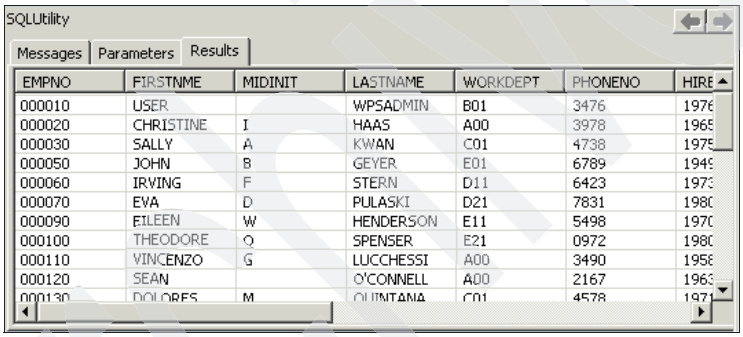


Figure 22-21 SQL statement results

22.2.4 Generating Java classes

In the sample code provided in 22.3, “Sample scenario” on page 689, the SQLUtilities.java class provides methods that execute the SQL statement, returning a DBSelect reference and an array of objects representing the rows in the result set.

Creating a utilities package

A package will be needed to store the generated java classes.

1. Open the Web perspective, right-click the **Java Resources** package and select **New** → **Package.**

Archived

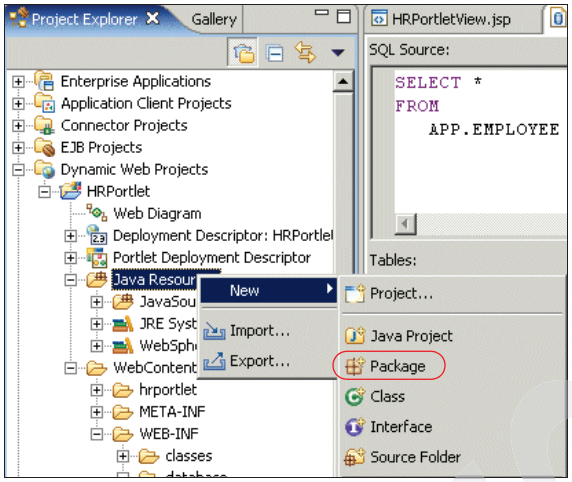


Figure 22-22 Java Resources package

2. For the Source Folder, enter `HRPortlet/JavaSource`. For the Name, enter `utilities`.
3. Click **Finish** to create the package.

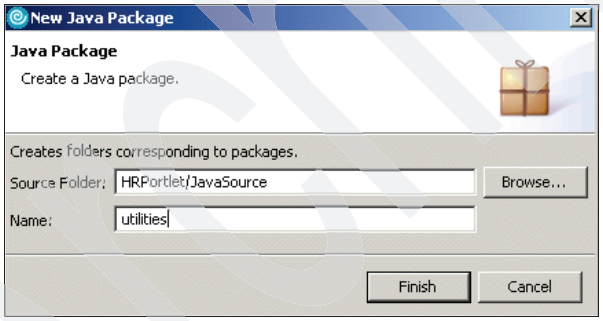


Figure 22-23 Create a Java package

Generating Java classes

Follow these steps to generate the Java classes:

1. On Data Perspective, right-click the **SQLUtility** statement and select **Generate Java Bean**.

Archived

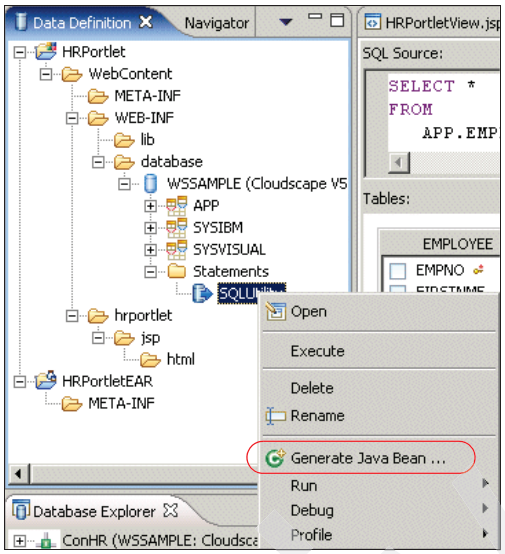


Figure 22-24 Generate JavaBean

2. Enter the following values for the Java Class Specification window:
   - Source Folder: `HRPortlet/JavaSource`
   - Package: `utilities`
   - Name: `SQLUtilities`

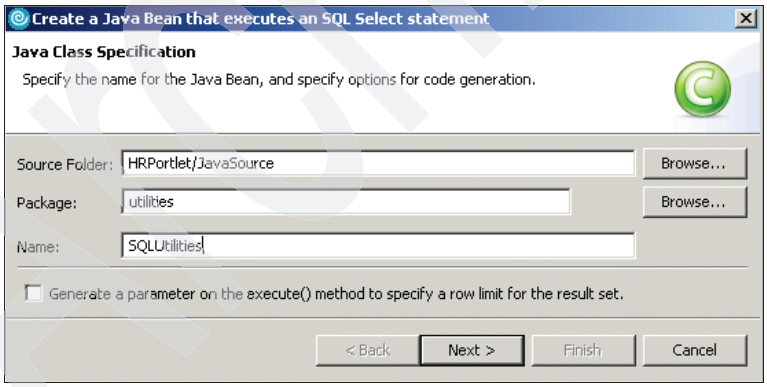


Figure 22-25 Java Class Specification

3. In the Specify Runtime Database Connection Information window, accept the default value for the `Use Driver Manager Connection` option. Click **Next**.

Archived

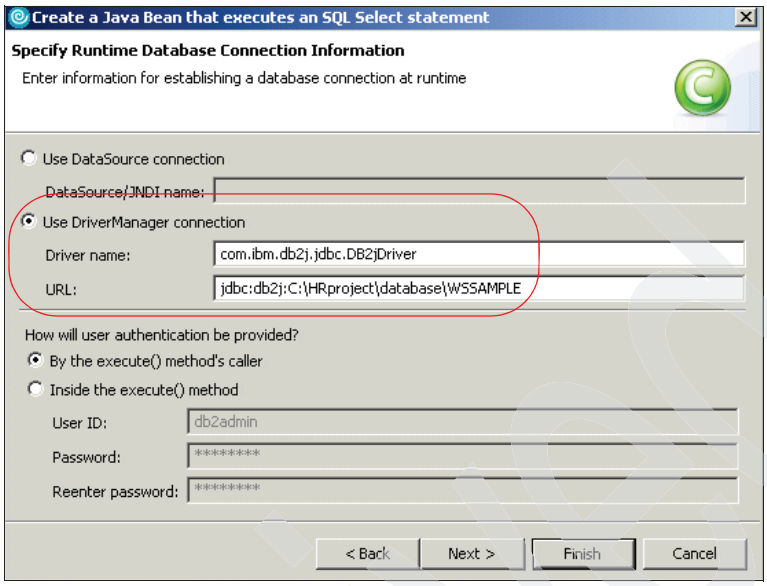


Figure 22-26 Runtime database connection information

4. Review the specification window information.

Archived

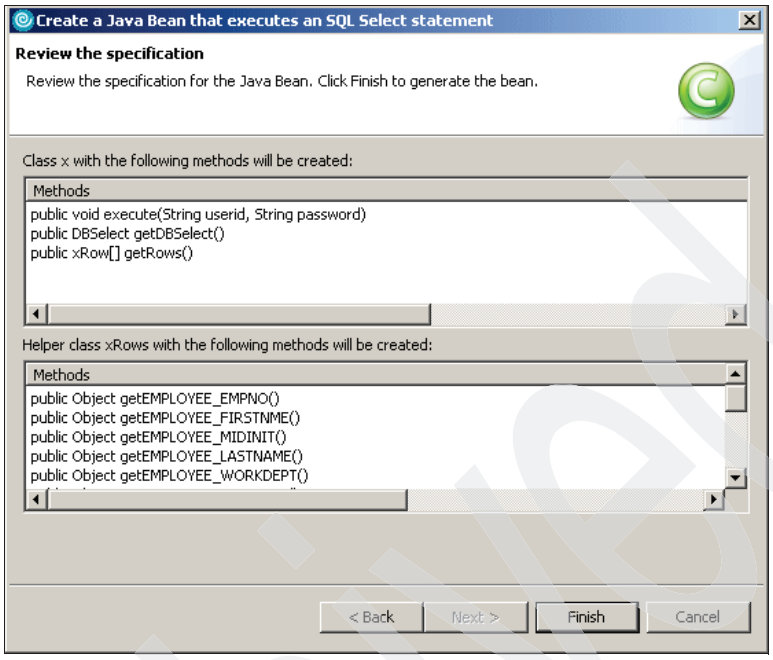


Figure 22-27 Review the specification

5. Click **Finish**.
6. Optionally, you can review the two Java classes created under Java Resources/utilities.

22.3 Sample scenario

This section provides a sample scenario of a portlet project that uses the JDBC interface to interact with relational database back end systems. You will import the sample portlet project to the project created on previous sections, deploy and run this portlet on a Test environment. This sample scenario will allow you understand the techniques used to develop portlets that retrieve information from databases using JDBC.

The development workstation has already been created for you and its components can be seen in Figure 22-28 on page 690.

Archived

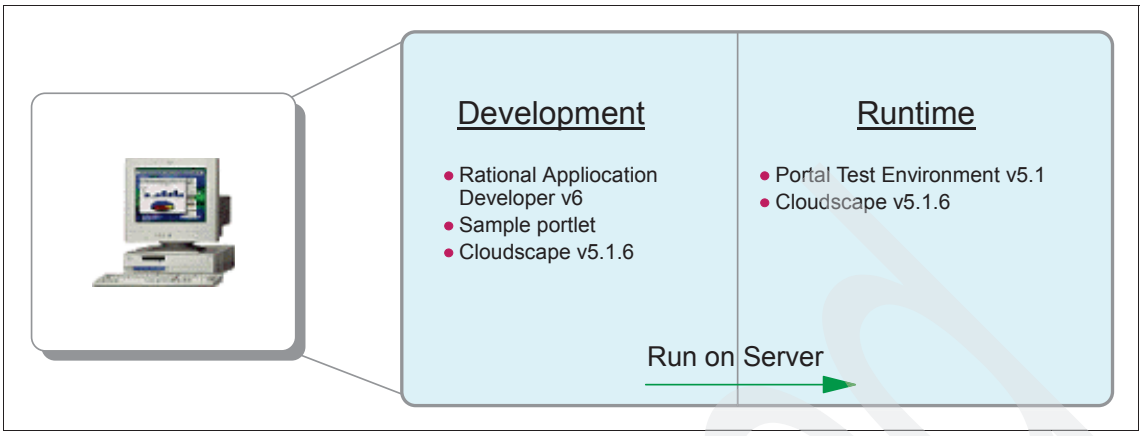


Figure 22-28 Development workstation

22.3.1 Overview

In this section, you will review and understand the sample scenario. You will import the code to use JDBC to interact with databases to the current HRPortlet project.

This example shows how the JDBC interface is used to read information from a Cloudscape sample table. In your portlet Edit mode, you will provide the information needed to establish a connection with the database in order to retrieve the content in View mode. The sample scenario is illustrated in Figure 22-29 on page 691.

Archived

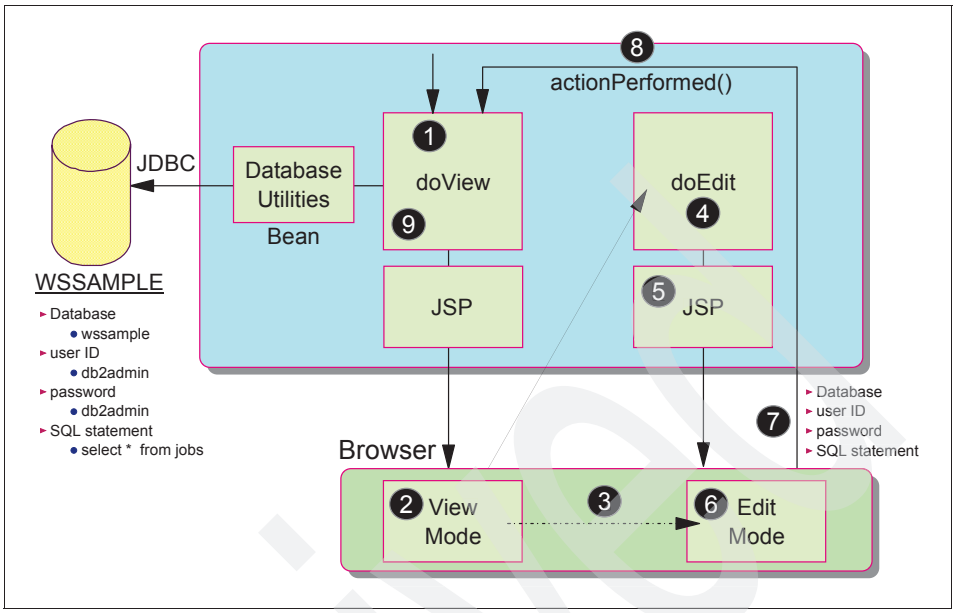


Figure 22-29 JDBC scenario

Note: It is assumed that the database has already been created and populated and the sample portlet WAR file is available.

The sequence flow for this sample scenario is as follows:

1. Initially, the doView method is executed.
2. A JSP is invoked in View mode to render the initial screen containing a welcome message telling the user to enter the database inquiry values in Edit mode.
3. The user clicks **Edit** to go into Edit mode.
4. The doEdit method executes and invokes a JSP (include).
5. The JSP renders the form. to fill the connection information.
6. The user enters the database name, user ID, password and SQL statement.

   Note: A user ID and password have not been implemented for this version of Cloudscape, therefore any user ID and password can be used

7. The user submits the request (post) to the previous mode (View mode). An action event is generated.

8. The actionPerformed() method executes and the following processes take place:
   a. The database, user ID, password, and SQL statement are extracted and sent to the JDBCPortletResults Bean.
   b. A connection object is created.
   c. A DBResults object is created. This object encapsulates the database inquiry.
   d. Database Utilities (another bean) is invoked to execute the actual database inquiry.
   e. Results are stored as a String in the request object.
9. The doView method executes again:
   a. Results are obtained in View mode.
   b. Results are rendered directly in View mode.

22.3.2 Importing the WAR file

In previous sections you created the portlet project and some basics components needed for database connection. In this section you will complete the Java code needed to finish the portlet sample scenario.

You will import the WAR file provided for this sample scenario in the HRPortlet project and you will replace the original files previously created by the wizard. These are the files that will be overwritten with the complete Java code:

- HRPortlet.java (Portlet class)
- HRPortletSessionBean.java (Session bean)
- HRPortletEditBean.java (Bean)
- HRPortletEdit.jsp (Edit Mode)
- HRPortletView.jsp (View Mode)
- portlet.xml (Portlet deployment descriptor)
- web.xml (Web deployment descriptor)
- A new class SQLUtilities.java (Utility) to set the DBSelect properties values and create methods to execute SQL statements
- A new class SQLUtilitiesRow.java (Utility) to retrieve each row of the result set
- A new class HRPortletViewBean.java (Bean) to store the DBResult object

To import the WAR file follow these steps:

1. Select **File** → **Import...**

Archived

2. In the Select window drag down and select **WAR file,** click **Next**.
3. In the `Import Resources from a WAR file` window, enter the following values:
   a. WAR file: `C:\LabFiles\JDBC\HR\HRPortlet.war`
   b. Web project: `HRPortlet`
   c. Select the **Overwrite existent resources without warning** checkbox.

   Note: Initially you will see a different Web project such as HRPortlet1. This is because HRPortlet already exists. Change this value to HRPortlet and therefore have the overwrite option available.

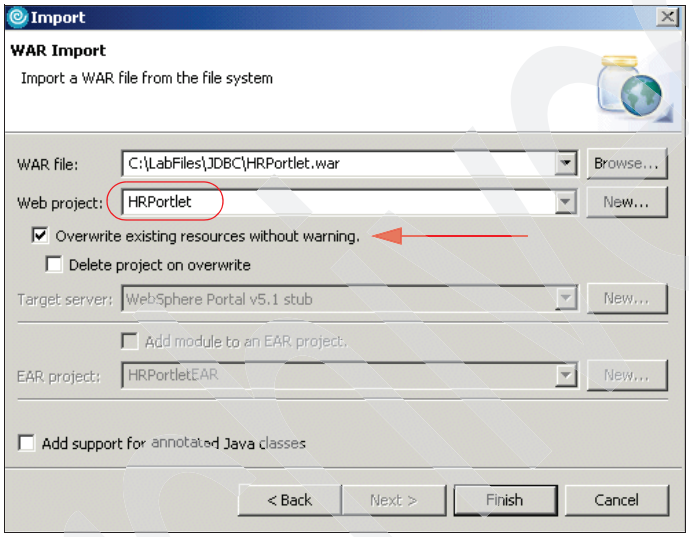


Figure 22-30 Importing the WAR file

4. Click **Finish**.
5. Select all resources to save.

22.3.3 Reviewing the portlet code

In this section, you will review the portlet code used in this sample scenario. To do this you can double-click the java files, the Java editor will open. These files are located in the /Java Source/hrportlet/ folder.

Double-click the HRPortlet.java file, to review its content:

1. The actionPerformed() method in this portlet does the following when an edit action occurs (see Example 22-2 on page 694).

a. It gets the parameters (database, user ID, password and SQL statement) from the request (PortletRequest).

b. It sets these values in the HRPortletSessionBean.java bean.

Example 22-2 HRPortlet.java - actionPerformed() method

```
.....
      // ActionEvent handler
      String actionString = event.getActionString();
      // Add action string handler here
      PortletRequest request = event.getRequest();

      HRPortletSessionBean sessionBean = getSessionBean(request);

      if (EDIT_ACTION.equals(actionString)) {
         String dbname = (String) request.getParameter("dbname");
         String userid = (String) request.getParameter("userid");
         String password = (String) request.getParameter("password");
         String sqlstring = (String) request.getParameter("sqlstring");
         sessionBean.setDbName(dbname);
         sessionBean.setUserId(userid);
         sessionBean.setPassword(password);
         sessionBean.setSqlString(sqlstring);

      }

      if (FORM_ACTION.equals(actionString)) {
         // Set form text in the session bean
         sessionBean.setFormText(request.getParameter(TEXT));
      }
   }
.....
```

2. The doEdit() method in the portlet does the following:

   a. It creates an instance of HRPorteltEditBean bean.

   b. It adds the action which will be executed when the form is submitted and sets this value in the Edit mode bean.

   c. When the database, user ID, password and SQL statement values exist for the session, it will also store these values in the Edit mode bean.

   d. The Edit mode bean is passed in the request to the HRPortletEdit.jsp.

Example 22-3 HRPortlet.java - doEdit() method

```
.....
      HRPortletEditBean editBean = new HRPortletEditBean();

      editBean.setFormAction(EDIT_ACTION);
```


Archived

```
        HRPortletSessionBean sessionBean = getSessionBean(request);
        String sqlstring = sessionBean.getSqlString();

        if (sqlstring != null) {
           String dbname = sessionBean.getDbName();
           String userid = sessionBean.getUserId();
           String password = sessionBean.getPassword();

           editBean.setDbname(dbname);
           editBean.setPassword(password);
           editBean.setUserid(userid);
           editBean.setSqlstring(sqlstring);
        }
        request.setAttribute(EDIT_BEAN, editBean);

        // Invoke the JSP to render
        getPortletConfig().getContext().include(EDIT_JSP +
getJspExtension(request),request,response);
.....
```

3. The doView() method does the following:
   a. It checks if there is a HRPortletSessionBean in the session. The first time this method is invoked, the session bean will be null.
   b. If there is an HRPortletSessionBean in session, it gets the database, user ID, password and SQL sentence values stored in it and creates an instance of HRPortletViewBean.
   c. The execute() method of SQLUtilities class is called to execute the SQL statement and the result is stored in the View mode bean by calling populateData() method in the SQLUtilities class.
   d. The View mode bean is passed in the request to HRPortletView.jsp.

Example 22-4 HRPortlet.java - doView() method

```
.....
        // Check if portlet session exists
        HRPortletSessionBean sessionBean = getSessionBean(request);
        if (sessionBean == null) {
           response.getWriter().println("<b>NO PORTLET SESSION YET</b>");
           return;
        }

        String sqlstring = sessionBean.getSqlString();
        if (sqlstring != null && !sqlstring.equals("")) {
           String dbname = sessionBean.getDbName();
           String userid = sessionBean.getUserId();
           String password = sessionBean.getPassword();
```

Archived

```
            // Make a view mode bean
            HRPortletViewBean viewBean = new HRPortletViewBean();

            SQLUtilities sqlUtility = new SQLUtilities();
            try {
               sqlUtility.execute(userid, password, dbname, sqlstring);
            } catch (SQLException e) {
               e.printStackTrace();
            }
            sqlUtility.populateData(viewBean);
            request.setAttribute(VIEW_BEAN, viewBean);
         }

         // Invoke the JSP to render
         getPortletConfig().getContext().include(VIEW_JSP +
getJspExtension(request),request,response);
.....
```

4. The HRPortletEdit.jsp is used to prompt for the database, user ID, password and SQL statement parameters. Double-click this file (located in /Web Content/hrportlet/jsp/html/ folder) to view its source code.
   a. Notice that the database name, user ID, password and SQL fields all have the HTML tag `value="<%=editBean.getXXX()%>"`. This allows the JSP to display the persistent data stored in the bean.

Example 22-5 HRPortletEdit.jsp (Edit mode sample code)

```
.....
<%
   HRPortletEditBean editBean = (HRPortletEditBean)
portletRequest.getAttribute("hrportlet.HRPortletEditBean");
%>

<HTML>
<BODY>
<h4><FONT size="3" face="Arial">Complete with your database information
and click Submit</FONT></h4>

<FORM method="post" action="<portletAPI:createReturnURI><portletAPI:URIAction
name='<%=editBean.getFormAction()%>'/></portletAPI:createReturnURI>">
<TABLE border="0">
   <TBODY>
      <TR>
         <TD width="805">
         <TABLE border="0">
            <TBODY>
               <TR>
```

Archived

```
                    <TD><FONT face="Arial" size="-1"><B>Database
:</B></FONT></TD>
                    <TD><INPUT type="text" value="<%=editBean.getDbname()%>"
                       name='<portletAPI:encodeNamespace value="dbname"/>'
size="20"></TD>
.....
```

5. Once the database parameters are collected, the request is processed by the actionPerformed() and doView() methods. The results are displayed by HRPortletView.jsp. Double-click this file to view its source code.

   This JSP checks if there is a HRPortletViewBean.java bean in the request. If there is not then it displays a message to go to Edit mode and configure an SQL statement. If the bean exists then the result of the query is displayed.

Example 22-6 HRPortletView.jsp (View mode sample code)

```
.....
<%
   HRPortletViewBean viewBean =
(HRPortletViewBean)portletRequest.getAttribute(HRPortlet.VIEW_BEAN);
   HRPortletSessionBean sessionBean =
(HRPortletSessionBean)portletRequest.getPortletSession().getAttribute(HRPortlet
.SESSION_BEAN);
%>

<% if (viewBean == null) { %>
<HTML>
<BODY>
<B>This is the JDBC Sample Portlet. Go to Edit mode and configure a SQL
query</B>
<% } else {
      try {
         com.ibm.db.beans.DBSelect results = viewBean.getResultFromDatabase();
%>
.....
```

6. The classes SQLUtilities.java and SQLUtilitiesRow.java have been generated in the data perspective. In the data perspective, you can create a connection to database, import the tables, create SQL statements and generate Java beans for the statements as you did in previous sections. These classes contain the methods to execute and retrieve information from database. The execute() method is called by the doView() method of HRPortlet.java when there is a statement in the session and the information retrieved is stored in the View mode bean by calling the populateData() method of SQLUtilities.java.

Archive

Example 22-7 SQLUtilities.java

```
   public void execute(String userid,String password,String url,String
command)
      throws SQLException {
      try {
         select.setUrl(url);
         select.setCommand(command);
         select.setUsername(userid);
         select.setPassword(password);
         select.execute();
      }

      // Free resources of select object.
      finally {
         select.close();
      }
   }

   public void populateData(HRPortletViewBean viewBean) {
      viewBean.setResultFromDatabase(select);
   }
```

22.3.4 Running the HRPortlet application

Finally, after creating the portlet project and importing the Java classes, you will run the HRPortlet.

1. Run the HRPortlet portlet by right-clicking **HRPortlet** in the Project Explorer view and selecting **Run** → **Run on server...**.
2. In the Define a New Server window, select the **Choose an existing server** option and for *Select the server that you want to use*, select **WebSphere Portal V5.1 Test Environment @ localhost**. Also, use default port 9081.

   Note: You can also check the `Set server as project default` checkbox, so you will not be prompted again when you run the portlet.

3. The portlet executes and you will see it in the built-in browser.

   The View mode is shown with a message indicating that you have to provide an SQL query; you will also need to switch the portlet into Edit mode (as indicated in Figure 22-31 on page 699) so you can enter these values.

Archived

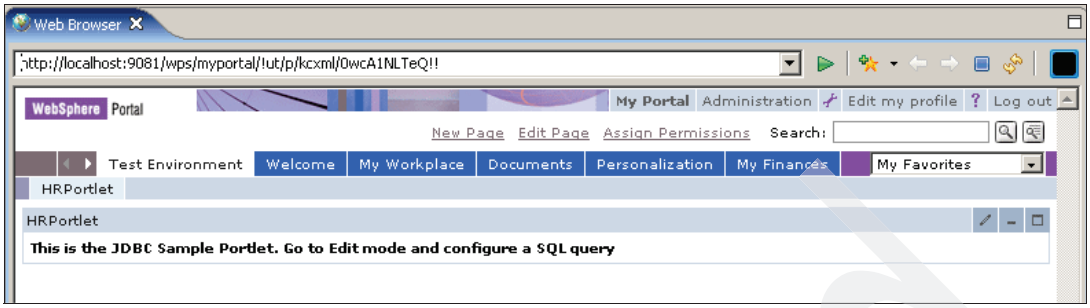


Figure 22-31 Portlet initial window

4. In Edit mode, the JSP for this mode renders the form requesting the database parameters. For the first time, enter the following information and select **Submit**:
   - Database: `jdbc:db2j:C:\HRproject\database\WSSAMPLE`
   - User: `db2admin`
   - Password: `db2admin`
   - SQL statement: `select * from jobs`

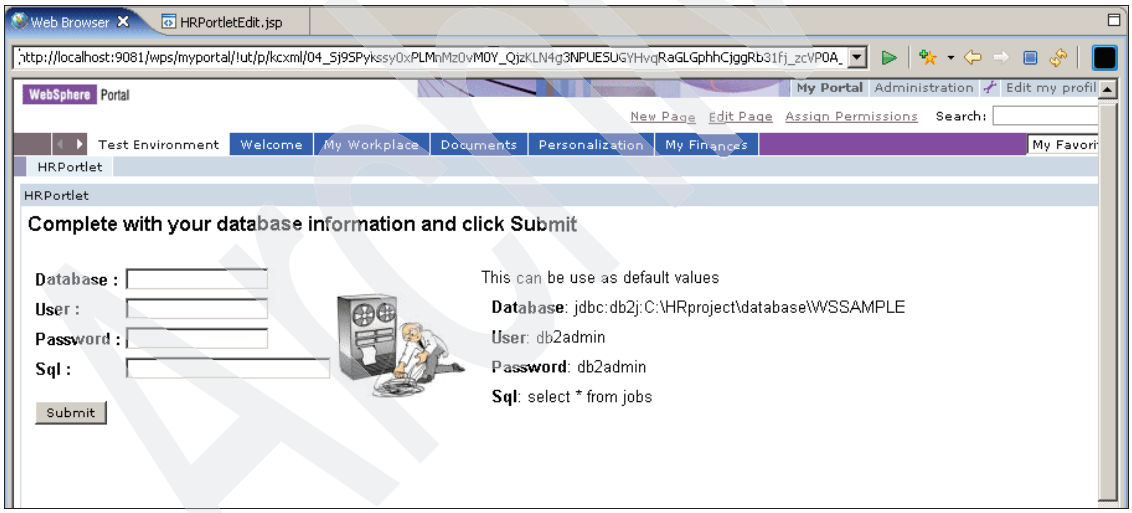
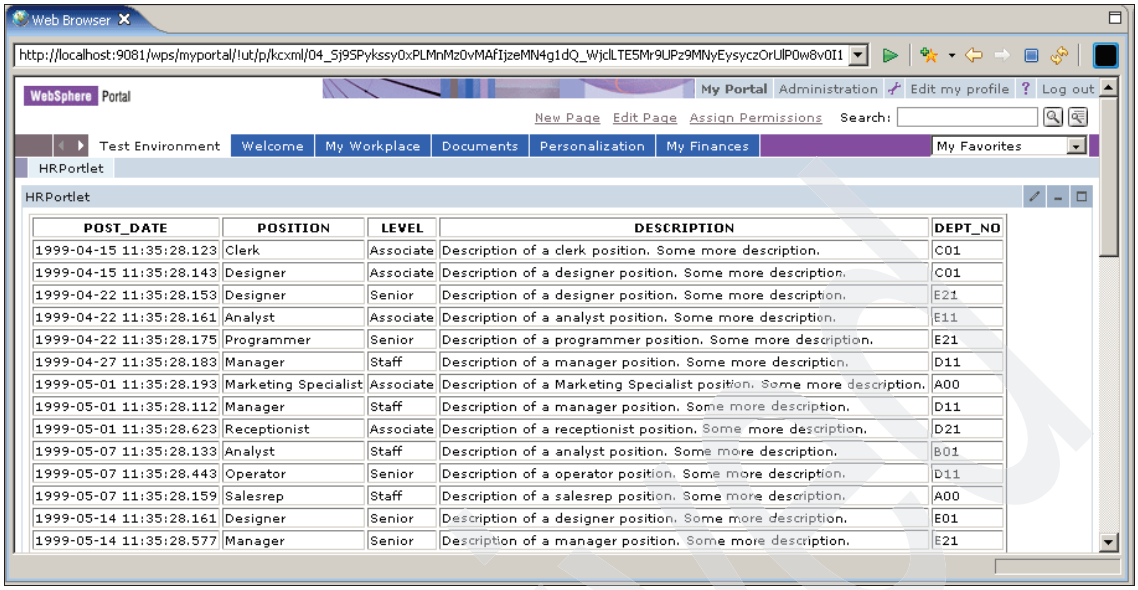


Figure 22-32 Portlet in Edit mode

5. Clicking **Submit** generates a creatEReturnURI and the portlet will return to View mode showing the results of your query against the WSSAMPLE database.

Archived



POST_DATE	POSITION	LEVEL	DESCRIPTION	DEPT_NO
1999-04-15 11:35:28.123	Clerk	Associate	Description of a clerk position. Some more description.	C01
1999-04-15 11:35:28.143	Designer	Associate	Description of a designer position. Some more description.	C01
1999-04-22 11:35:28.153	Designer	Senior	Description of a designer position. Some more description.	E21
1999-04-22 11:35:28.161	Analyst	Associate	Description of a analyst position. Some more description.	E11
1999-04-22 11:35:28.175	Programmer	Senior	Description of a programmer position. Some more description.	E21
1999-04-27 11:35:28.183	Manager	Staff	Description of a manager position. Some more description.	D11
1999-05-01 11:35:28.193	Marketing Specialist	Associate	Description of a Marketing Specialist position. Some more description.	A00
1999-05-01 11:35:28.112	Manager	Staff	Description of a manager position. Some more description.	D11
1999-05-01 11:35:28.623	Receptionist	Associate	Description of a receptionist position. Some more description.	D21
1999-05-07 11:35:28.133	Analyst	Staff	Description of a analyst position. Some more description.	B01
1999-05-07 11:35:28.443	Operator	Senior	Description of a operator position. Some more description.	D11
1999-05-07 11:35:28.159	Salesrep	Staff	Description of a salesrep position. Some more description.	A00
1999-05-14 11:35:28.161	Designer	Senior	Description of a designer position. Some more description.	E01
1999-05-14 11:35:28.577	Manager	Senior	Description of a manager position. Some more description.	E21

Figure 22-33 Query results

6. Enter Edit mode again. Notice that the database name, user ID, password and SQL statement, which have been stored in session object, are persistent.

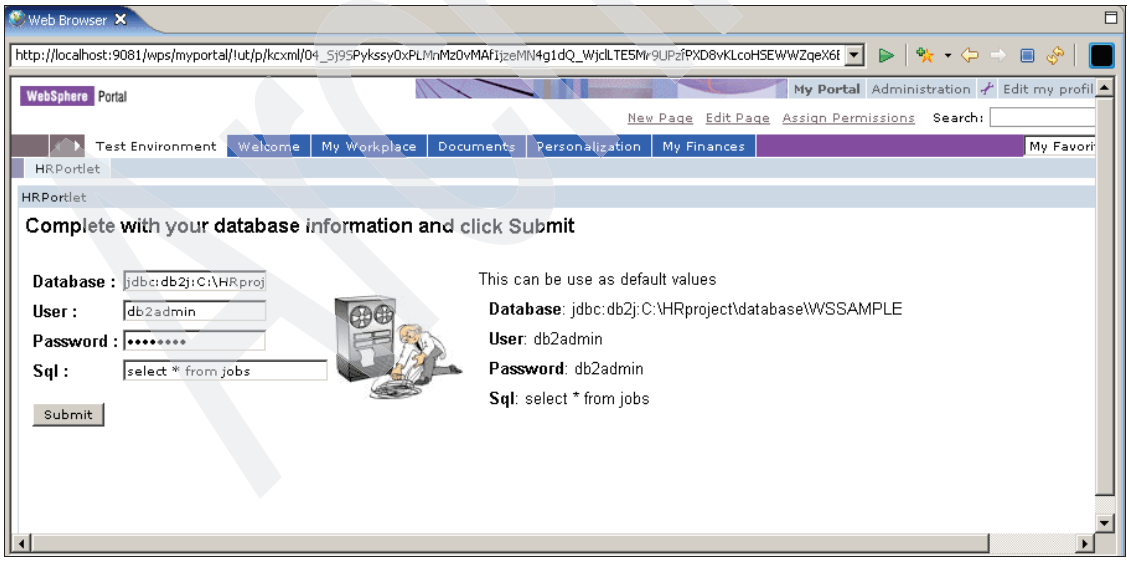


Figure 22-34 Database information and SQL query statement

7. Enter a new query, for example `select * from survey`, and click **Submit**. You will be presented with the results of your new query.

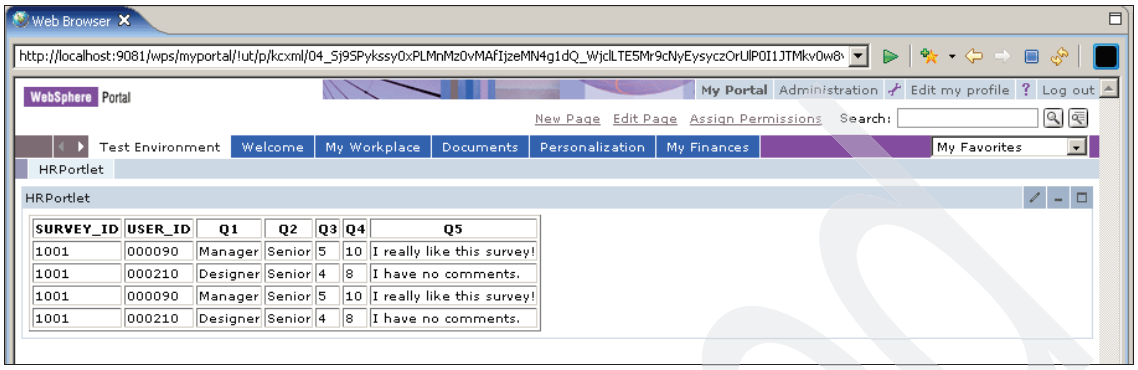


Figure 22-35 Query results

Archived

Archived

23

Accessing JDBC databases using Data Source in standard portlets

In this chapter, we introduce the process of gaining access to back-end JDBC databases using a Data Source for the database connection. The sample scenario is implemented using the JSR 168 portlet specification for classes and JSPs. A simple portlet project is included to show how portlets access relational databases using the JDBC interface.

This chapter discusses the following topics:

- How to create a portlet project on Rational Application Developer.
- How to create a Data Source connection to a database using Rational Application Developer.
- How to create Java classes that will provide a database connection through a datasource.
- A portlet example to access a database.

© Copyright IBM Corp. 2005. All rights reserved.

23.1 Data Source overview

Portlet applications access relational databases through the JDBC API to store, organize, and retrieve data. As an option, databases can also be accessed via DataSource objects. In this case, the connection object that the getConnection() method returns is a handle to a PooledConnection object rather than being a physical connection.

In the general sense, using a connection pooling has no impact on portlet applications except that connections should be closed. When a portlet application closes a connection that is pooled, the connection is returned to a pool of reusable connections. Using a connection pool will give you the following benefits:

- A connection does not need to be created every time one is requested.
- You configure the minimum and maximum number of connections and therefore you control the connectivity to a database to significantly improve performance.

In WebSphere Application Server V5, a data source is associated with a JDBC provider that supplies the specific JDBC driver implementation class. The data source represents the J2EE Connector Architecture (JCA) connection factory for the relational resource adapter. The JCA Connection Manager provides the connection pooling, local transaction, and security supports. The relational resource adapter provides both JDBC wrappers and JCA CCI implementation that allows BMP, JDBC applications, and CMP beans to access the database.

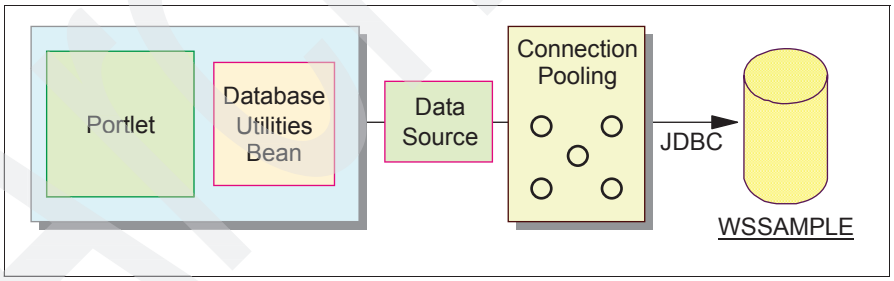


Figure 23-1 JDBC Data Source using connection pooling

Note: For more details about data source, see the WebSphere Portal Server V5 InfoCenter.

When an application uses a version 5 data source, the data source will use the JCA connector architecture to get to the relational database. Although there is no difference between the existing WebSphere JDBC support and the new support

in terms of application development, there will be some connection behavior changes because of different architectures. For example:

- The connection factory delegates the request to a connection manager.
- The connection manager looks for an instance of a connection pool in the application server. If no connection pool is available, then the manager uses the ManagedConnectionFactory to create a physical (nonpooled) connection.

23.2 Creating a JSR 168 portlet project

In this section, the process of creating a portlet project named HRPortlet is described. This project will be used later in 23.3, “Creating a sample database” on page 710 to show the JDBC connection.

23.2.1 Creating HRPortlet

In this section, you create a portlet project name HRPortlet168. The portlet will be created based on a Basic portlet type using the provided wizard. This portlet will be published and executed in the Portal Test Environment.

If not already running, start the Rational Application Developer; select **Start** → **Programs** → **IBM Rational** → **IBM Rational Application Developer V6.0** → **Rational Application Developer**.

1. Select **File** → **New** → **Project...**

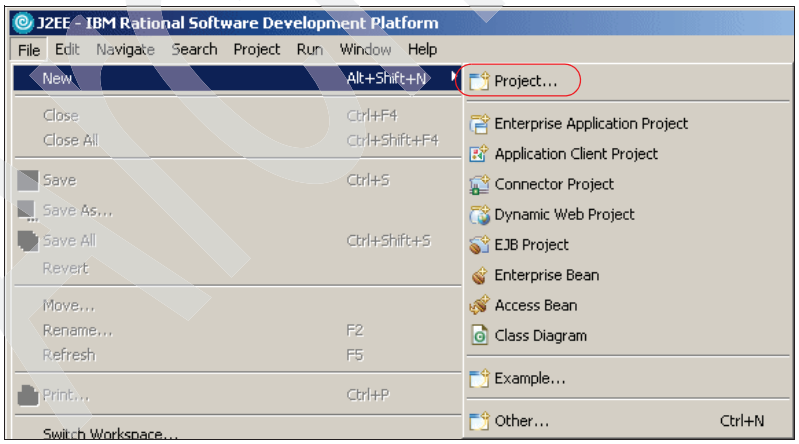


Figure 23-2 Creating a new project

2. In the Select a wizard window, select **Portlet Project** (JSR 168). Click **Next**.

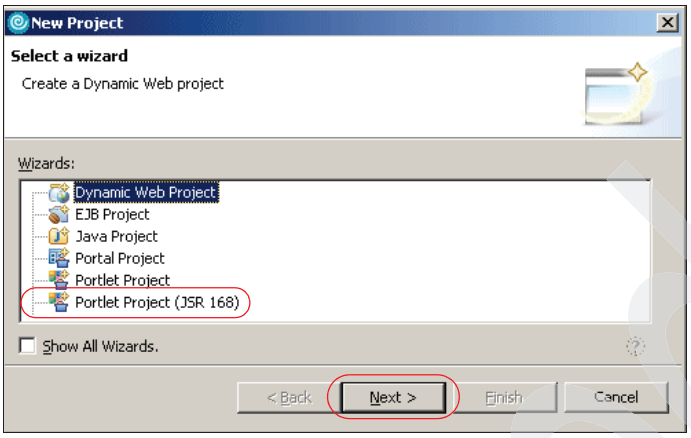


Figure 23-3 Select Portlet project

3. The Portlet Project window will appear; enter `HRPortlet168` for the name of the project. Click **Show Advanced >>** and change Target Server to `WebSphere Portal V5.1 stub` and Context Root to `/HRPortlet168`. Click **Next.**

Archived

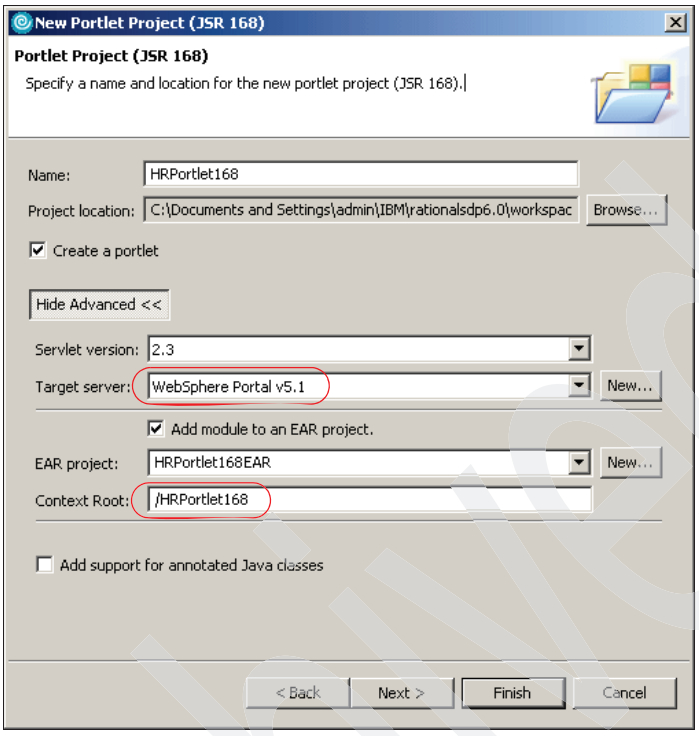


Figure 23-4 Portlet project configuration

4. In the Portlet Type window, select **Basic portlet**. Click **Next**.

Archived

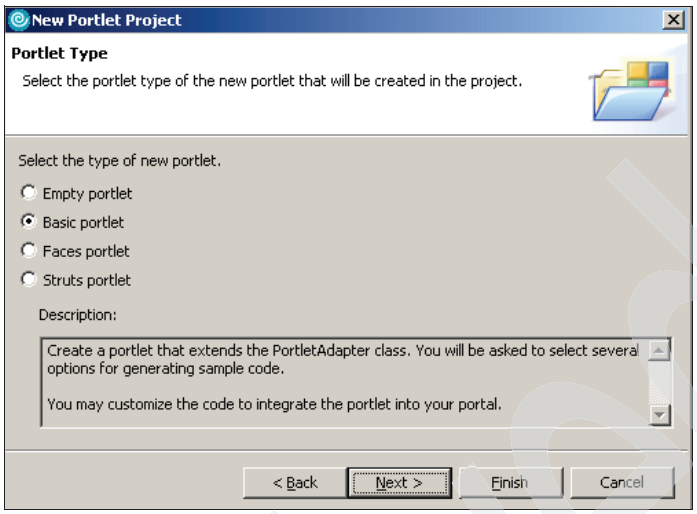


Figure 23-5 Basic portlet

5. In the Features window, leave the default values. Click **Next**.
6. In the Portlet Settings window, click the **Change code generations options** checkbox. Remove the `portlet` word from the Portlet name, Portlet Title and Class Prefix fields, so they will look as in Figure 23-6 on page 709. Click **Next**.

Archived

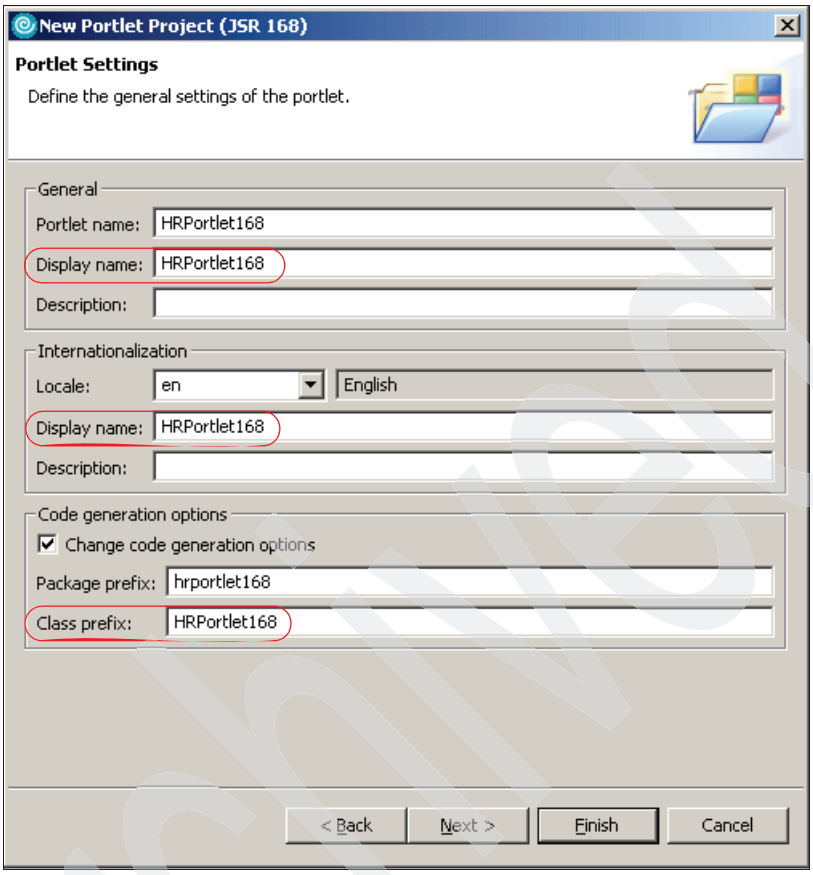


Figure 23-6 Portlet settings

7. Accept default values for the Event Handling window by clicking **Next**.
8. Accept default values for the Single Sign-on window by clicking **Next**. The Credential Vault is not used in this scenario.
9. Click **Finish** in the Miscellaneous window.

Note: If this is the first time that Rational Application Developer is used, you will be prompted to switch to the Web perspective. Click **Yes**.

Archived

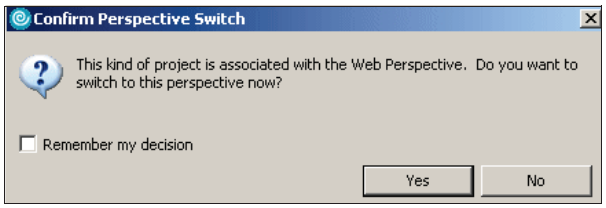


Figure 23-7 Switch to Web perspective

As a result of this process, you should see, in the Project Explorer view, all the project artifacts created by Rational Application Developer, under **Dynamic Web Projects → HRPortlet168.**

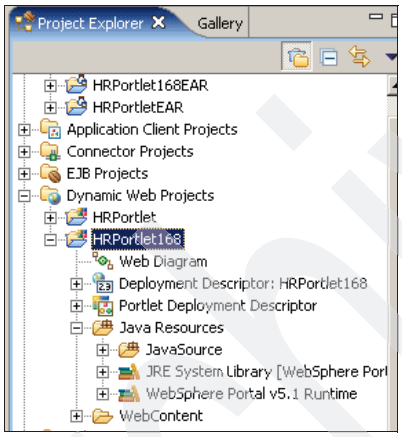


Figure 23-8 HRPortlet168 project

23.3 Creating a sample database

In this chapter, it is assumed that the sample database WSSAMPLE has been already created and populated with sample data. For details about how to create and populate WSSAMPLE, see Chapter 22, “Accessing JDBC databases from portlet applications” on page 669.

23.3.1 Creating a connection

Once the database is created, you can proceed to create the connection to the database and verify it. However, before you do this, you will need to create a folder to copy the database metadata.

Archived

Creating a database folder

On the Web perspective, execute the following steps:

1. Right-click the **WEB-INF** folder, select **New** → **Folder** from the contextual menu.

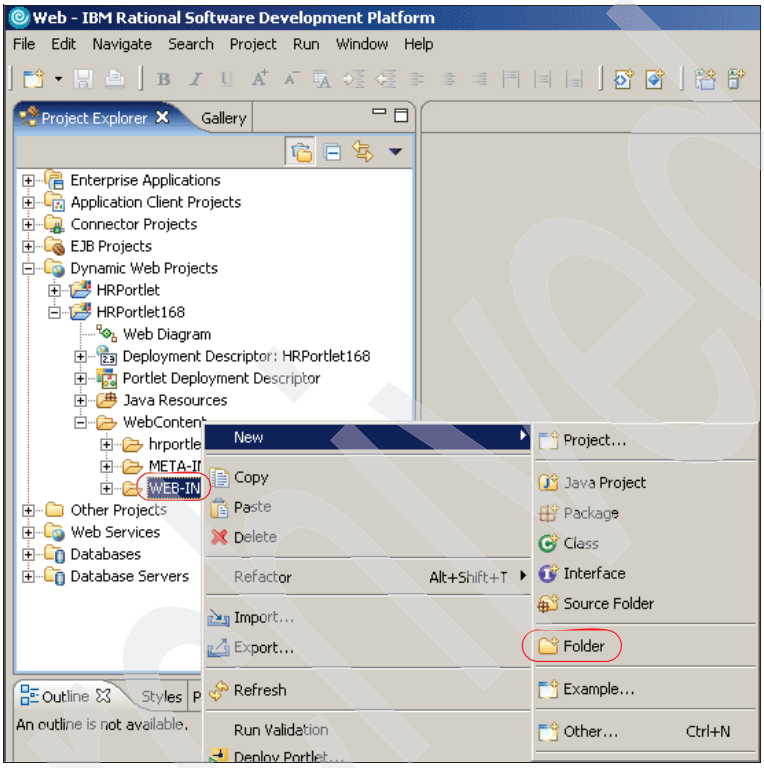


Figure 23-9 Select a new folder

2. Enter `database` as the Folder name.

Archive

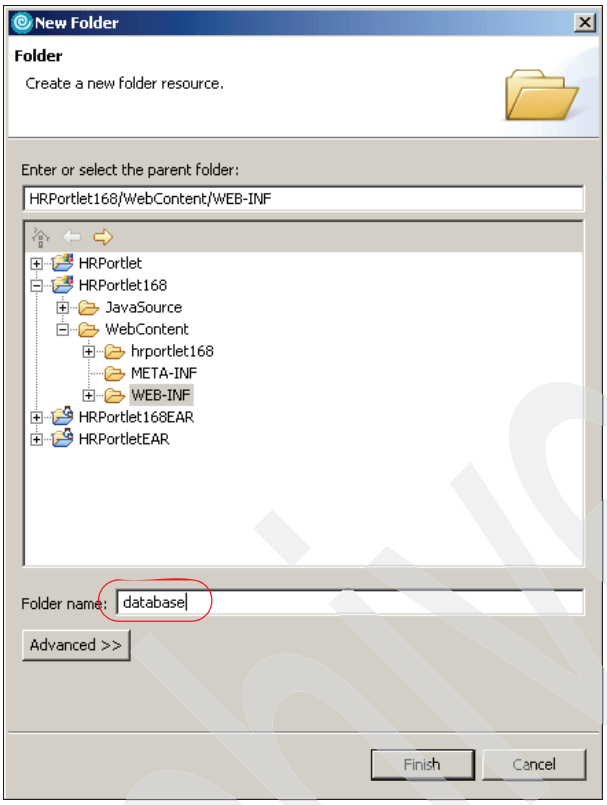


Figure 23-10 Entering a folder name

3. Click **Finish**.

Creating a new connection

Follow these steps to create a new database connection:

1. Change to the Data perspective by selecting **Window** → **Open Perspective** → **Data**.

Archived

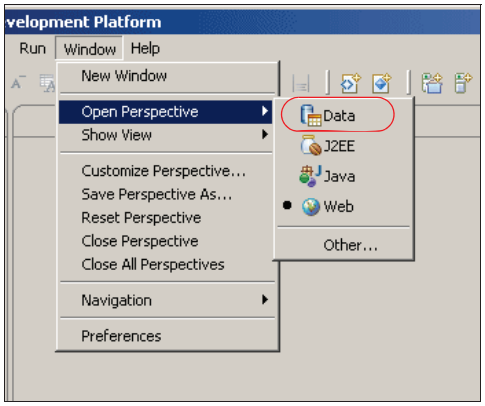


Figure 23-11 Data perspective

2. Right-click inside the Database Explorer view.

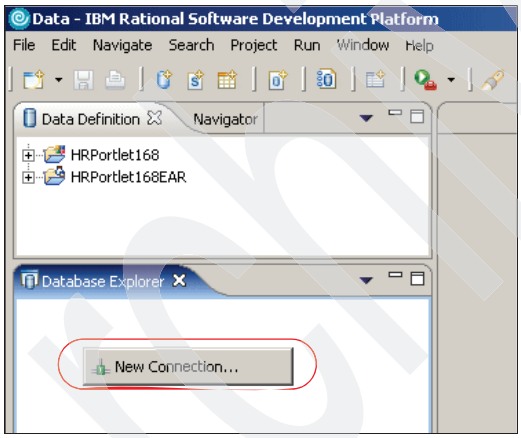


Figure 23-12 Creating a new connection

3. Click **New Connection...** from the context menu to create a new connection. The *Establish a connection to a database* window will appear.
4. Type `ConnHR` as the connection name. Click **Next**.
5. In the Specify connection parameters window, select the following values:
   a. Select database manager: `V5.1`
   b. JDBC driver: `Cloudscape Embedded JDBC Driver`
   c. Enter the path where the database for this scenario was created. For example:

Archived

c:\HRProject168\database\WSSAMPLE

d. Uncheck **Create the database** since you already created the database.

e. Leave the default value for Class location field.

f. As a user ID , enter `db2admin`. As a password, use `db2admin` also.

> **Note:** The user ID and password are optional in this sample scenario since Cloudscape does not verify it.

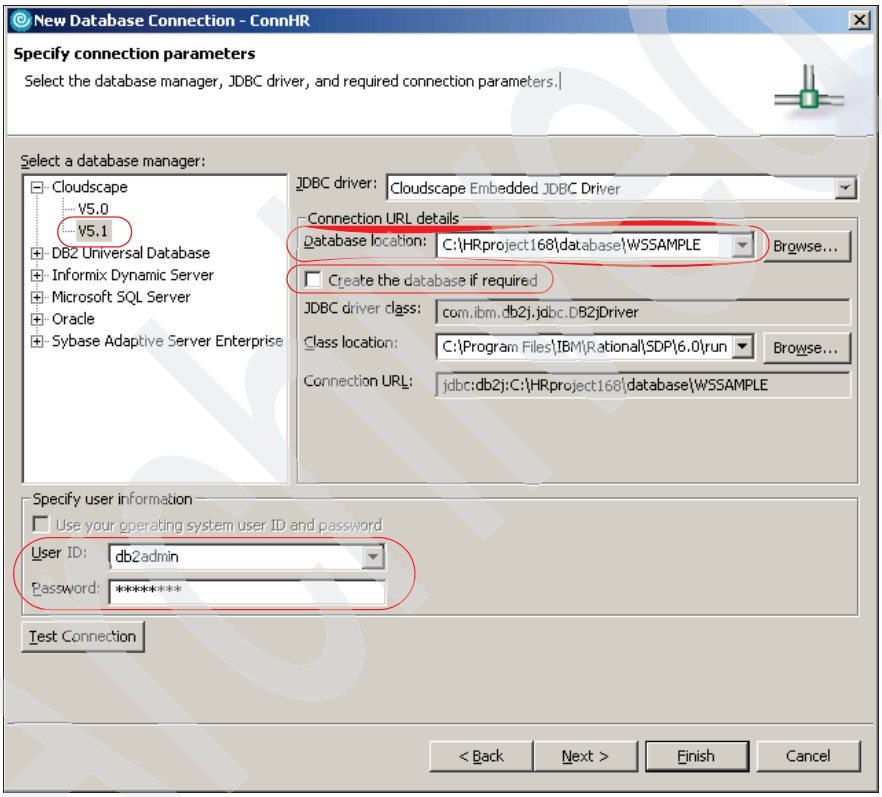


Figure 23-13 Connection parameters

6. Click the **Test Connection** button. You should see a dialog box with the message `Connection to WSSAMPLE is successful`. Click **OK**.

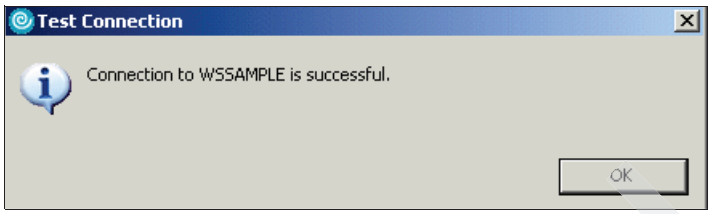


Figure 23-14 Successful database connection

Note: If you do not receive the successful message, make sure the values you entered in the Specify connection parameters window are correct and try again until you get a successful message indication.

7. Click **Finish**.
8. Click **Yes** to the question `Do you want to copy the database metadata to a project folder?`.
9. In the Copy to Project window, click the **Browse** button, and select the database folder you previously created:

   /HRPortlet168/WebContent/WEB-INF/database

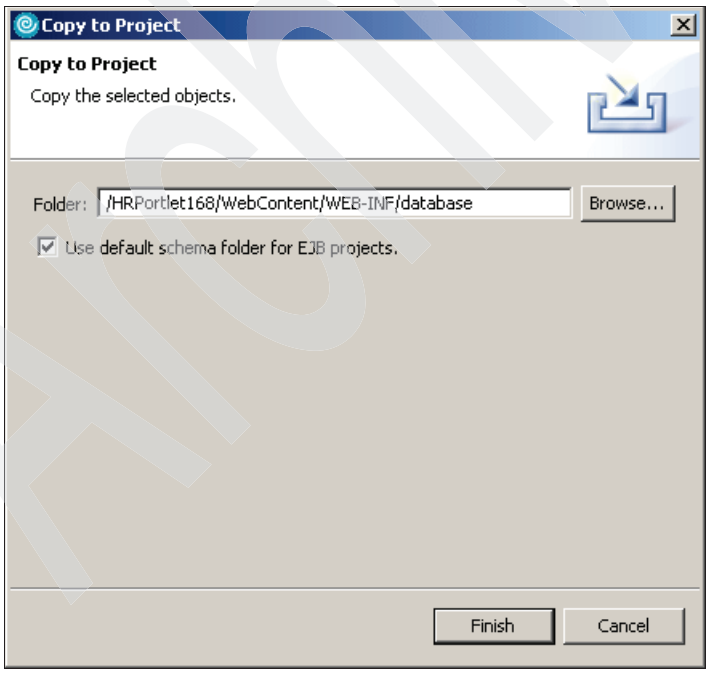


Figure 23-15 Copy objects

Archived

10. Click **Finish**.

23.3.2 Creating an SQL statement

In this section, a simple SQL statement to display all employees will be created. When the statement has been created, the editor opens and you have several options to create a SQL statement. For example, you can use the wizard or you can use the editor to write your query. In this section, you will use the wizard to create and execute the following SQL statement:

```
select * from employee
```

1. In the Data Definition view, right-click the **Statement** folder under the sample WSSAMPLE database.
2. Select **New** → **Select Statement.**

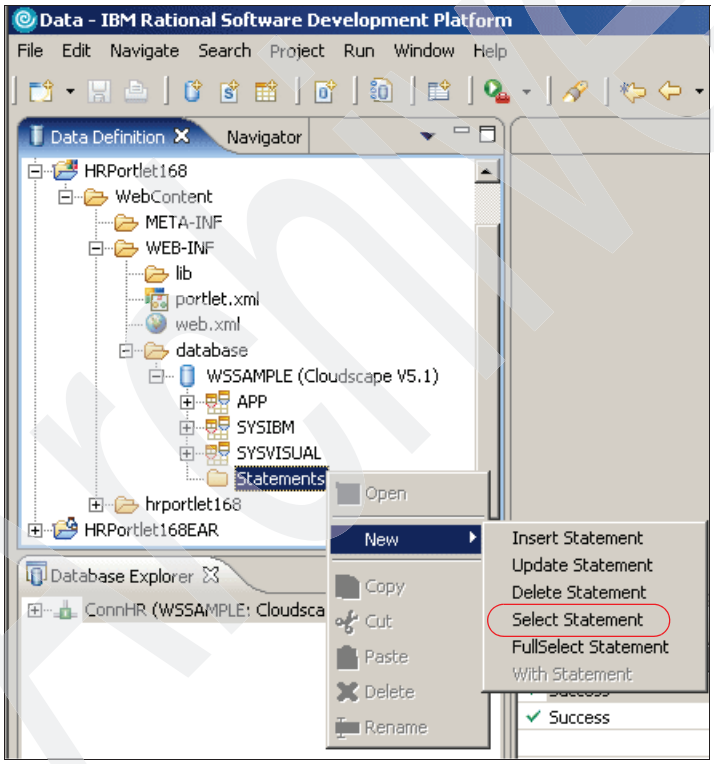


Figure 23-16 Creating the SWL select statement

3. Enter `SQLUtility` for the Statement Name and click **OK**.
4. The Editor will open; right-click the Tables area.

5. Select **Add Table...** in the context window.
6. Select the **APP.EMPLOYEE** from the Table name pop-up and click **OK**.

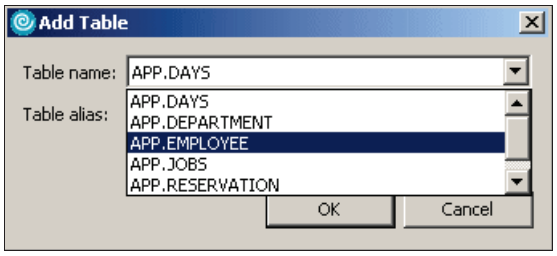


Figure 23-17 Add table

7. On the Data Definition view, expand the **Statement** folder.
8. Right-click the **SQLUtility** statement.
9. Select **Execute** from the context menu.

Archived

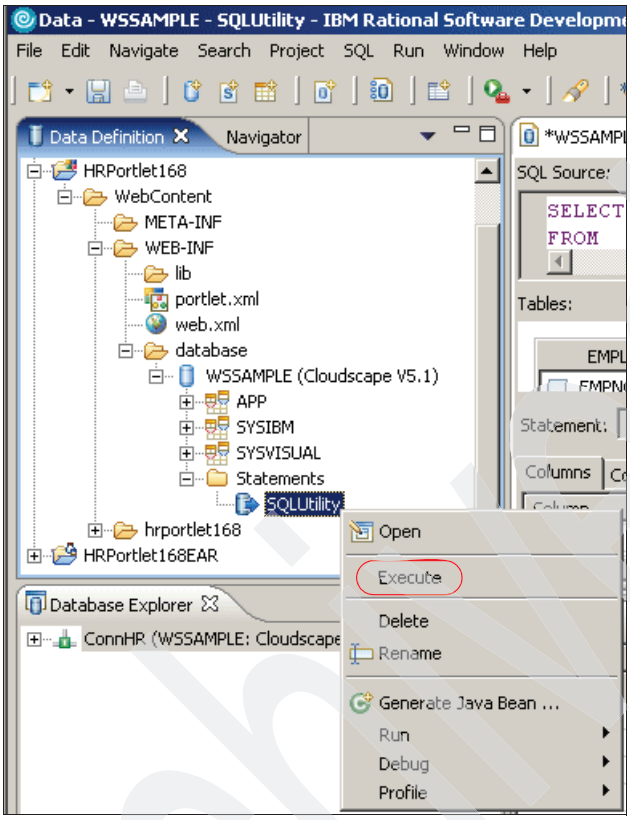


Figure 23-18 Execute SQL statement

10. This request returns the content from the EMPLOYEE table.

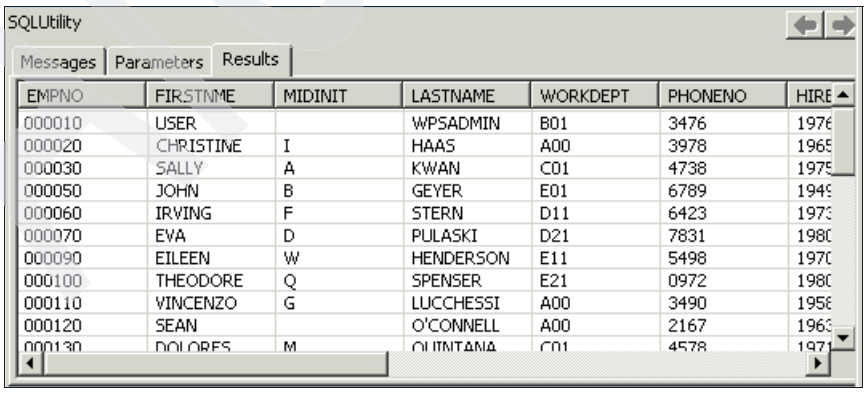


Figure 23-19 SQL statement results

23.3.3 Generating Java classes

In the sample code provided in 23.4, “Sample scenario” on page 724, the SQLUtilities.java class provides methods that execute the SQL statement, returning a DBSelect reference and an array of objects representing the rows in the result set.

Creating a utilities package

A package will be needed to store the generated Java classes.

1. Open the Web perspective, right-click the **Java Resources** package and select **New** → **Package**.

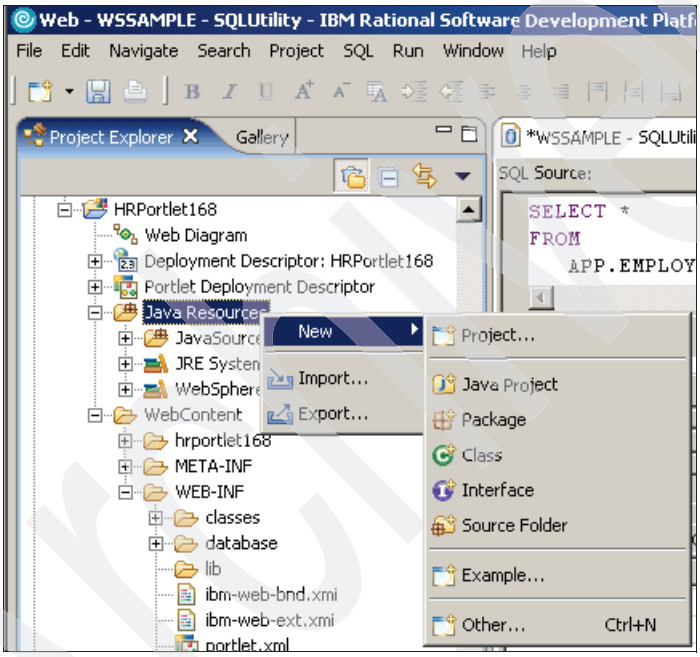


Figure 23-20 Java Resources package

2. For the Source folder, enter `HRPortlet168/JavaSource`. For the Name, enter `utilities`.
3. Click **Finish** to create the package.

Archived

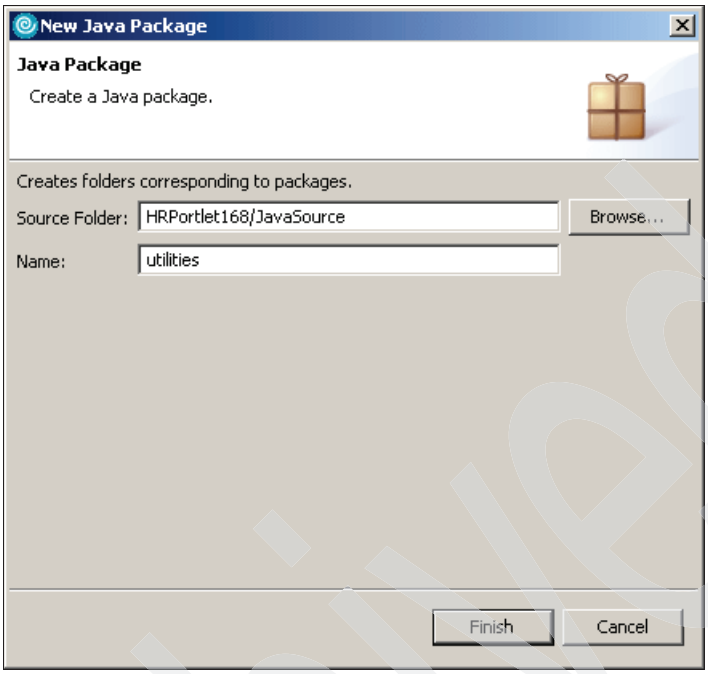


Figure 23-21 Create a Java package

Generating Java classes

Follow these steps to generate the Java classes:

1. On Data Perspective, right-click the **SQLUtility** statement and select **Generate Java Bean**.

Archived

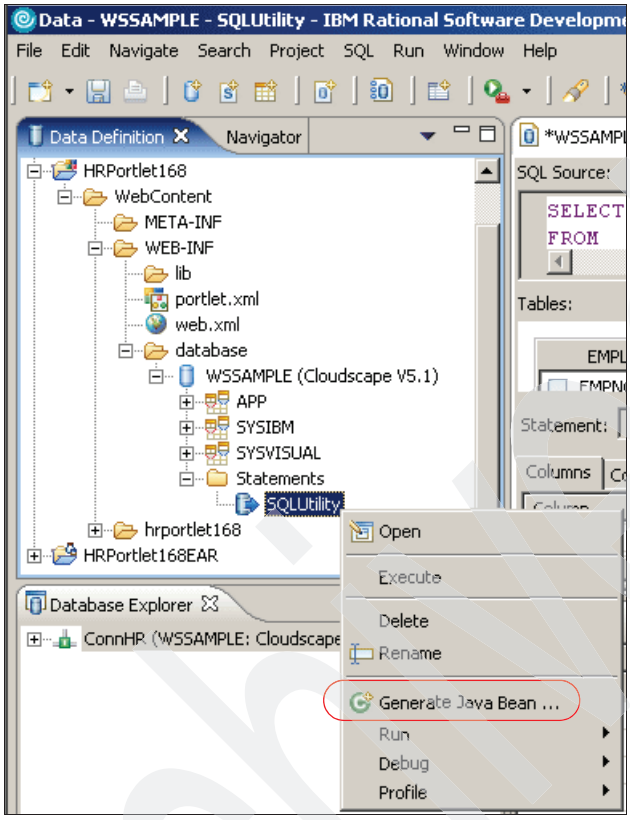


Figure 23-22 Generate JavaBean

2. Enter the following values for the Java Class Specification window:
   - Source Folder: `HRPortlet168/JavaSource`
   - Package: `utilities`
   - Name: `SQLUtilities`

Arnico

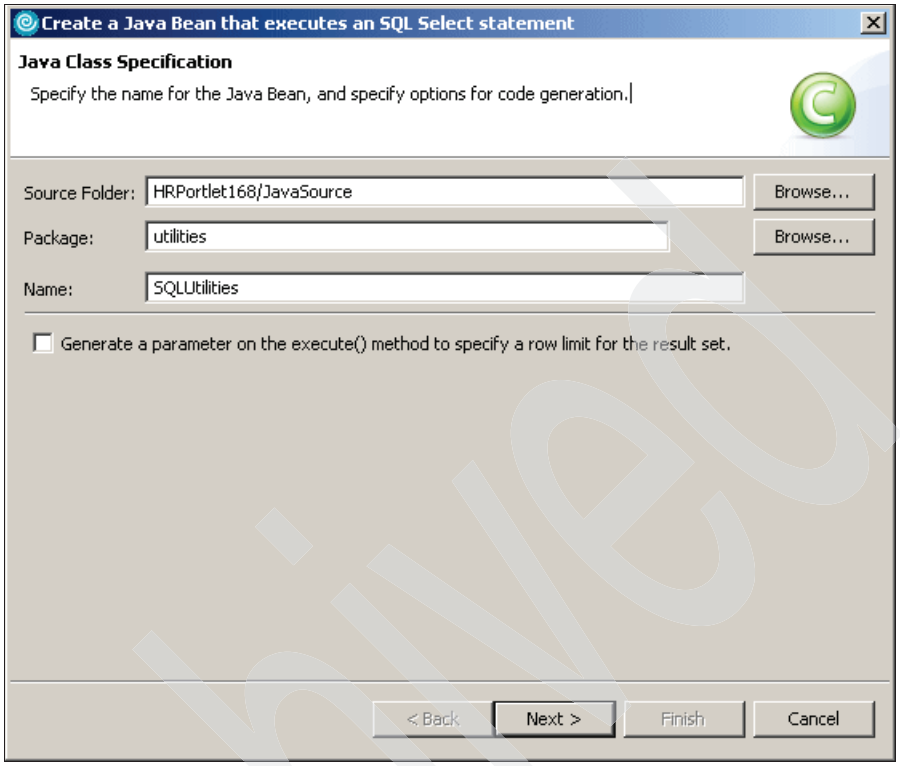


Figure 23-23 Java Class Specification

3. In Specify Runtime Database Connection Information window, accept the default value for the *Use Driver Manager Connection* option. Click **Next**.

Archive

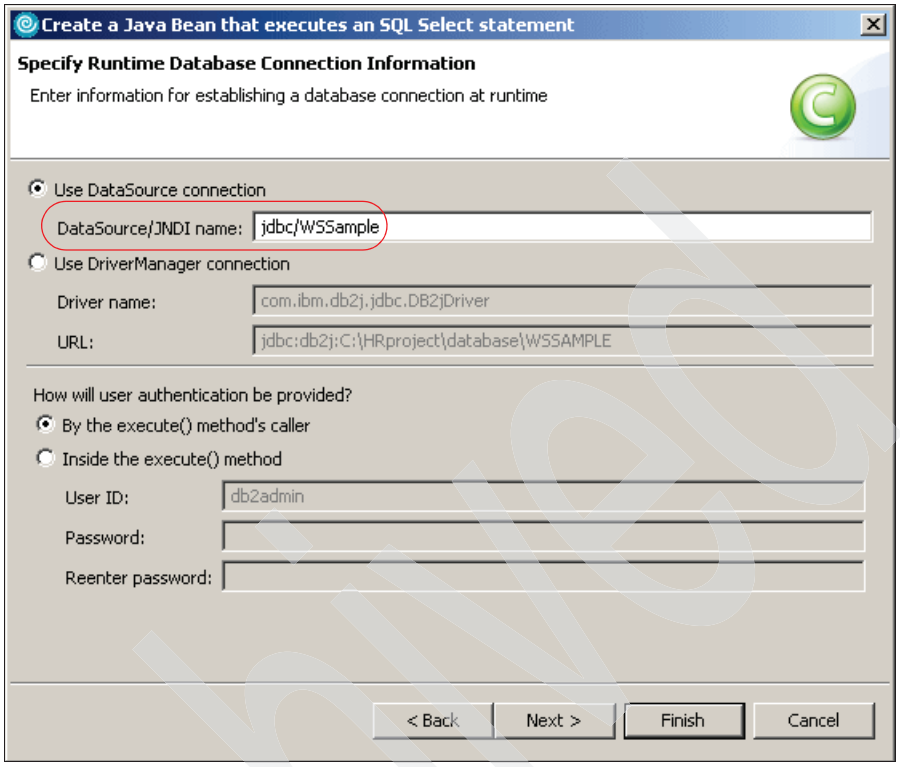


Figure 23-24 Runtime database connection information

4. Review the specification window information.

Archived

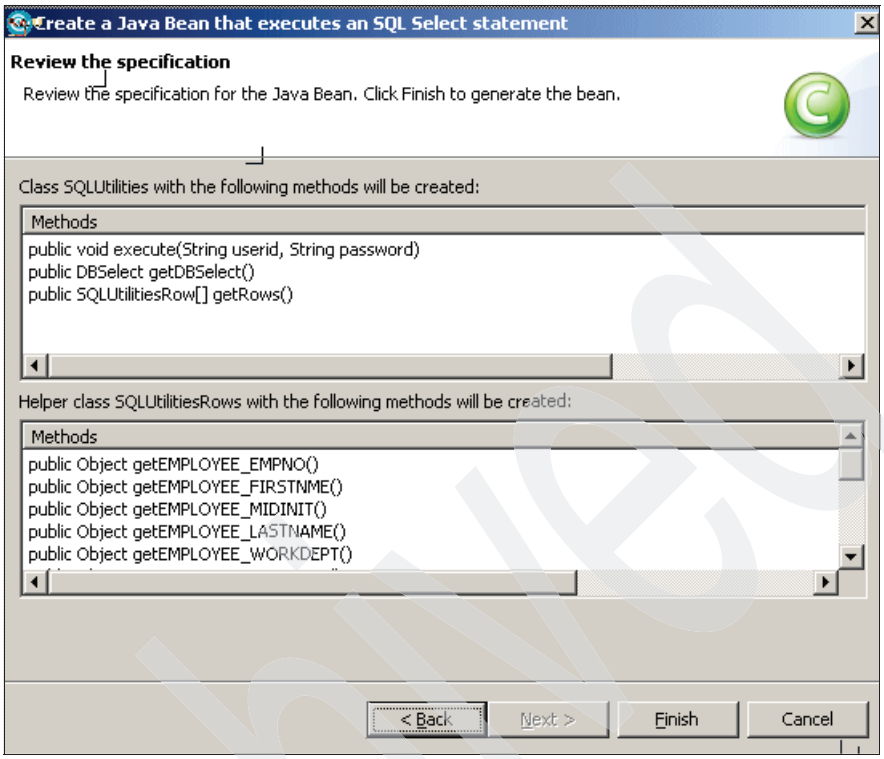


Figure 23-25 Review the specification

5. Click **Finish**.
6. Optionally, you can review the two Java classes created under Java Resources/utilities.

23.4 Sample scenario

This section provides a sample scenario of a JSR 168 portlet project that uses the JDBC interface to interact with relational database back end systems. You will import the sample portlet project to the project created in previous sections, deploy and run this portlet in a test environment. This sample scenario will allow you to understand the techniques used to develop portlets that retrieve information from databases using JDBC.

The development workstation has already been created for you and its components can be seen in Figure 23-26 on page 725.

Archived

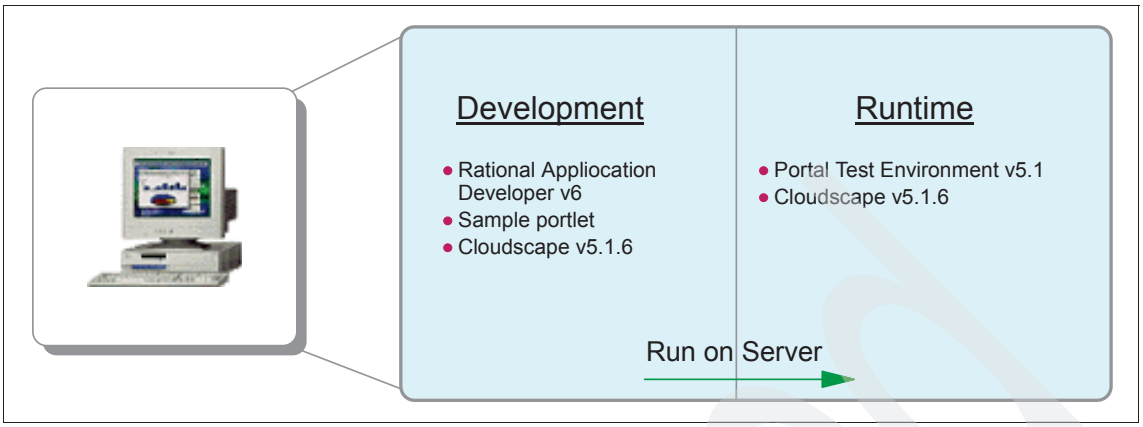


Figure 23-26 Development workstation

23.4.1 Overview

In this section, you will review and understand the sample scenario. You will import the java code to use JDBC to interact with databases to the current HRPortlet168 project.

This example shows how the JDBC interface is used to read information from a Cloudscape sample table. In your portlet Edit mode, you will provide the statement that is needed to establish a connection with the database in order to retrieve the content in View mode. The sample scenario is illustrated in Figure 23-27 on page 726.

Archived

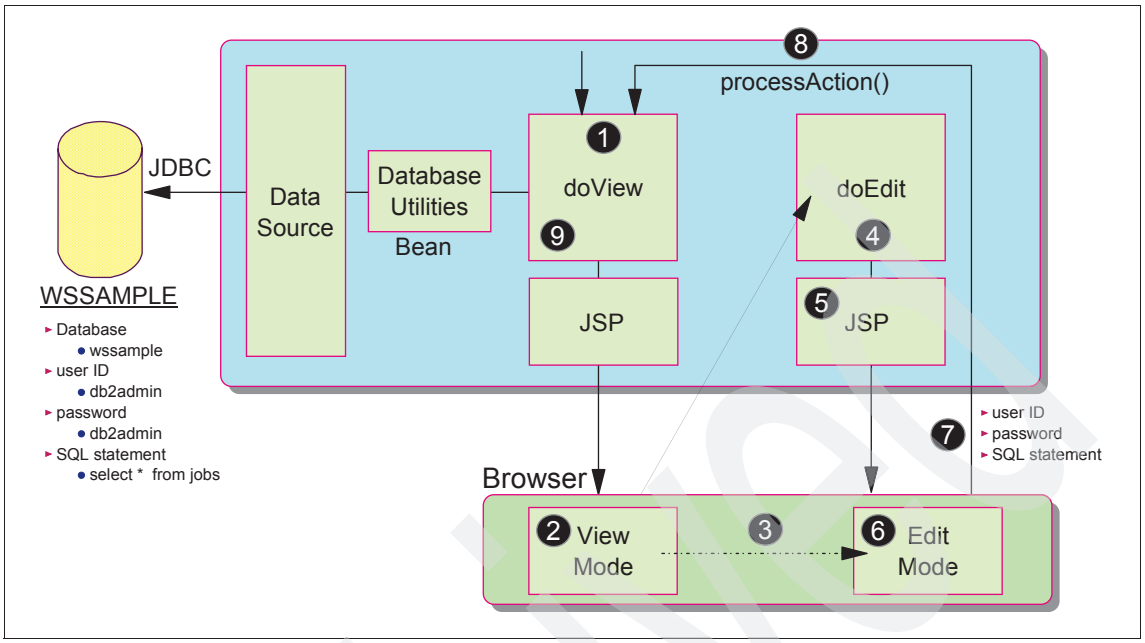


Figure 23-27 Sample JSR 168 portlet using data source

Note: It is assumed that the database has already been created and populated and the sample portlet WAR file is available.

The sequence flow for this sample scenario is as follows:

1. Initially, the doView method is executed.
2. A JSP is invoked in View mode to render the initial screen containing a welcome message telling the user to enter the database inquiry values in Edit mode.
3. The user clicks **Edit** icon to go into Edit mode.
4. The doEdit method executes and invokes a JSP (include).
5. The JSP renders the form to fill the connection information.
6. The user enters the user ID, password and SQL statement.

   Note: A user ID and password have not been implemented for this version of Cloudscape, therefore any user ID and password can be used.

7. The user submits the request (post). An actionURL is generated.

8. The processAction() method executes and the following processes take place:
   a. The user ID, password, and SQL statement are extracted and sent to the HRPortlet168SessionBean object.
   b. If the PortletMode is Edit mode, then change the PortletMode to VIEW mode, this execute the doView method.
9. In the doView:
   a. A DBSelect object is created and initialized with the DataSource.
   b. The user ID, password, and SQL statement are extracted and sent to the SQLUtilities through the execute method.
   c. Results are stored in the DBSelect object.
   d. A HRPortlet168ViewBean is sent to the SQLUtility through the populateData method.
   a. The BDSelect object is added to the HRPortlet168ViewBean sent in the previous activity.
   b. Finally, the HRPortlet168ViewBean is rendered in the request to the JSP.

23.4.2 Importing the WAR file

In previous sections, you created the portlet project and some basics components needed for database connection. In this section you will complete the Java code needed to finish the portlet sample scenario.

You will import the WAR file provided for this sample scenario in the HRPortlet168 project and you will replace the original files previously created by the wizard. These are the files that will be overwritten with the complete Java code:

- HRPortlet168.java (Portlet class)
- HRPortlet168SessionBean.java (Session bean)
- HRPortlet168EditBean.java (Bean)
- HRPortlet168Edit.jsp (Edit Mode)
- HRPortlet168View.jsp (View Mode)
- portlet.xml (Portlet deployment descriptor)
- web.xml (Web deployment descriptor)
- A new class SQLUtilities.java (Utility) to set the DBSelect properties values and create methods to execute SQL statements
- A new class SQLUtilitiesRow.java (Utility) to retrieve each row of the result set

Archived

- A new class HRPortlet168ViewBean.java (Bean) to store the DBSelect object

To import the WAR file, follow these steps:

1. Select **File** → **Import...**
2. In the Select window, drag down and select **WAR file**, then click **Next**.
3. In the Import Resources from a WAR file window, enter the following values:
   a. WAR file: `C:\HRPortlet168\5.1\HRPortlet168.war`
   b. Web project: `HRPortlet168`
   c. Select the **Overwrite existent resources without warning** checkbox.

   Note: Initially, you will see a different Web project such as HRPortlet1681. This is because HRPortlet already exists. Change this value to `HRPortlet168` to have the overwrite option available.

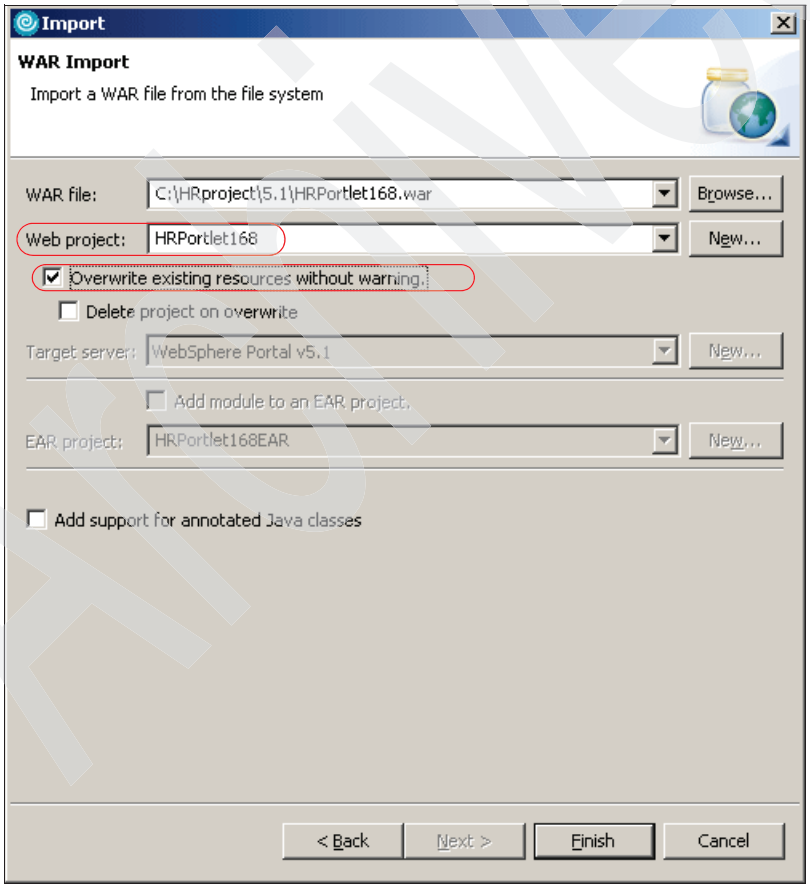


Figure 23-28 Importing the WAR file

Archived

4. Click **Finish**.
5. Select all resources to save.

23.4.3 Reviewing the portlet code

In this section, you will review the portlet code used in this sample scenario. To do this you can double-click the Java files, the Java editor will open. These files are located in the /Java Source/hrportlet168/ folder.

Double-click the **HRPortlet168.java** file, to review its content:

1. The processAction() method in this portlet does the following when an edit action occurs. (see Example 23-1)
   a. It gets the parameters (user ID, password and SQL statement) from the request (ActionRequest).
   b. It sets these values in the HRPortlet168SessionBean.java bean.

Example 23-1 HRPortlet168.java - processAction() method

```
.....
      //Add action string handler here
      HRPortlet168SessionBean sessionBean = getSessionBean(request);

      if(request.getPortletMode().equals(PortletMode.EDIT)) {
         String dbname = (String) request.getParameter("dbname");
         String userid = (String) request.getParameter("userid");
         String password = (String) request.getParameter("password");
         String sqlstring = (String) request.getParameter("sqlstring");
         sessionBean.setDbName(dbname);
         sessionBean.setUserId(userid);
         sessionBean.setPassword(password);
         sessionBean.setSqlString(sqlstring);
         if(request.getPortletMode().equals(PortletMode.EDIT))
            response.setPortletMode(PortletMode.VIEW);
      }

      if( request.getParameter(FORM_SUBMIT) != null ) {
         // Set form text in the session bean
         sessionBean = getSessionBean(request);
         if( sessionBean != null )
            sessionBean.setFormText(request.getParameter(TEXT));
      }
.....
```

2. The doEdit() method in the portlet does the following:
   a. It creates an instance of HRPortelt168EditBean bean.

b. When the user ID, password and SQL statement values exist for the session, it will also store these values in the Edit mode bean.

c. The Edit mode bean is passed in the request to the HRPortlet168Edit.jsp.

Example 23-2 HRPortlet168.java - doEdit() method

```
.....
      HRPortlet168EditBean editBean = new HRPortlet168EditBean();
      HRPortlet168SessionBean sessionBean = getSessionBean(request);
      String sqlstring = sessionBean.getSqlString();

      if (sqlstring != null) {
         String dbname = sessionBean.getDbName();
         String userid = sessionBean.getUserId();
         String password = sessionBean.getPassword();

         editBean.setDbname(dbname);
         editBean.setPassword(password);
         editBean.setUserid(userid);
         editBean.setSqlstring(sqlstring);
      }
      request.setAttribute(EDIT_BEAN, editBean);

      // Invoke the JSP to render
      PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher(getJspFilePath(request, EDIT_JSP));
      rd.include(request,response);
.....
```

3. The doView() method does the following:

   a. It checks if there is a HRPortlet168SessionBean in the session. The first time this method is invoked, the session bean will be null.

   b. If there is an HRPortlet168SessionBean in session, it gets the user ID, password and SQL sentence values stored in it and creates an instance of HRPortlet168ViewBean.

   c. The execute() method of SQLUtilities class is called to execute the SQL statement and the result is stored in the View mode bean by calling populateData() method in the SQLUtilities class.

   d. The View mode bean is passed in the request to HRPortlet168View.jsp.

Example 23-3 HRPortlet168.java - doView() method

```
.....
      //Set the MIME type for the render response
      response.setContentType(request.getResponseContentType());
```

Archived

```
      // Check if portlet session exists
      HRPortlet168SessionBean sessionBean = getSessionBean(request);
      if( sessionBean==null ) {
         response.getWriter().println("<b>NO PORTLET SESSION YET</b>");
         return;
      }
      String sqlstring = sessionBean.getSqlString();
      if (sqlstring != null && !sqlstring.equals("")) {
         String dbname = sessionBean.getDbName();
         String userid = sessionBean.getUserId();
         String password = sessionBean.getPassword();

         // Make a view mode bean
         HRPortlet168ViewBean viewBean = new HRPortlet168ViewBean();
         SQLUtilities sqlUtility = new SQLUtilities();
         try {
            sqlUtility.execute(userid, password, sqlstring);
         } catch (SQLException e) {
            e.printStackTrace();
         }
         sqlUtility.populateData(viewBean);
         request.setAttribute(VIEW_BEAN, viewBean);
      }
      // Invoke the JSP to render
      PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher(getJspFilePath(request, VIEW_JSP));
      rd.include(request,response);
.....
```

4. The HRPortlet168Edit.jsp is used to prompt for the user ID, password and SQL statement parameters. Double-click this file (located in /Web Content/hrportlet168/jsp/html/ folder) to view its source code.

   a. Notice that the user ID, password and SQL fields all have the HTML tag `value=”<%=editBean.getXXX()%>”`. This allows the JSP to display the persistent data stored in the bean.

Example 23-4 HRPortlet168Edit.jsp (Edit mode sample code)

```
.....
<portlet:defineObjects/>
<%
   HRPortlet168EditBean editBean =
(HRPortlet168EditBean)renderRequest.getAttribute(HRPortlet168.EDIT_BEAN);
%>
<HTML>
<BODY>
<h4><FONT size="3" face="Arial">Complete with   an SQL Statement information
and click Submit</FONT></h4>
```

Archived

```
<FORM method="post" action="<portlet:actionURL/>">
<TABLE border="0">
   <TBODY>
      <TR>
         <TD width="805">
         <TABLE border="0">
            <TBODY>
               <TR>
                  <TD></TD>
                  <TD></TD>
                  <TD rowspan="5"><IMG
src='<%=response.encodeURL(request.getContextPath() +
"/images/database.gif")%>'></TD>
                  <TD width="476" valign="top"><FONT size="2"
face="Arial">  This can be use as default values</FONT></TD>

               </TR>
               <TR>
                  <TD><FONT face="Arial" size="-1"><B>User
:</FONT></B></FONT></TD>
                  <TD><INPUT type="text" value="<%=editBean.getUserid()%>"
name="userid" size="20"></TD>
.....
```

5. Once the database parameters are collected, the request is processed by the processAction() and doView() methods. The results are displayed by HRPortlet168View.jsp. Double-click this file to view its source code.
   a. This JSP check if there is a HRPortlet168ViewBean.java bean in the request. If there is not then displays a message to go to Edit mode and configure and SQL statement. If the bean exists then the result of the query is displayed.

Example 23-5 HRPortlet168View.jsp (View mode sample code)

```
.....
<portlet:defineObjects/>
<%
   HRPortlet168SessionBean sessionBean =
(HRPortlet168SessionBean)renderRequest.getAttribute(HRPortlet168.SESSION_BEAN);
    HRPortlet168ViewBean viewBean=
(HRPortlet168ViewBean)renderRequest.getAttribute(HRPortlet168.VIEW_BEAN);
%>

<% if (viewBean == null) { %>
<HTML>
<BODY>
<B>This is the JDBC Sample Portlet. Go to Edit mode and configure a SQL
```

Archived

```
query</B>
<% } else {
       try {
          com.ibm.db.beans.DBSelect results = viewBean.getResultFromDatabase();
%>
.....
```

6. The classes SQLUtilities.java and SQLUtilitiesRow.java have been generated in the data perspective. In the data perspective, you can create a connection to database, import the tables, create SQL statements and generate Java beans for the statements as you did in previous sections. These classes contain the methods to execute and retrieve information from database. The execute() method is called by the doView() method of HRPortlet168.java when there is a statement in the session and the information retrieved is stored in the View mode bean by calling the populateData() method of SQLUtilities.java.

Example 23-6 SQLUtilities.java

```
      public void execute(String userid, String password, String command)
throws SQLException {
      try {
         select.setUsername(userid);
         select.setPassword(password);
         select.setCommand(command);
         select.execute();
      }

      // Free resources of select object.
      finally {
         select.close();
      }
   }

   public void populateData(HRPortlet168ViewBean viewBean) {
      viewBean.setResultFromDatabase(select);
   }
```

23.4.4 Creating the Data Source

Before you run the HRPortlet168 you will need to create the DataSource used by the SQLUtility to make the connection to the database.

On the Web perspective.

1. Select the **Server** view located at the bottom center of the workspace.

Archived

2. Double-click the **WebSphere Portal V5.1 Test Environment**. This will open the Server config editor.
3. Select the **Data source** tab.
4. In the Server Settings section, select **Cloudscape JDBC Driver** from the JDBC provider list.
5. Click **Add...** from Data source defined in the JDBC provider selected above.

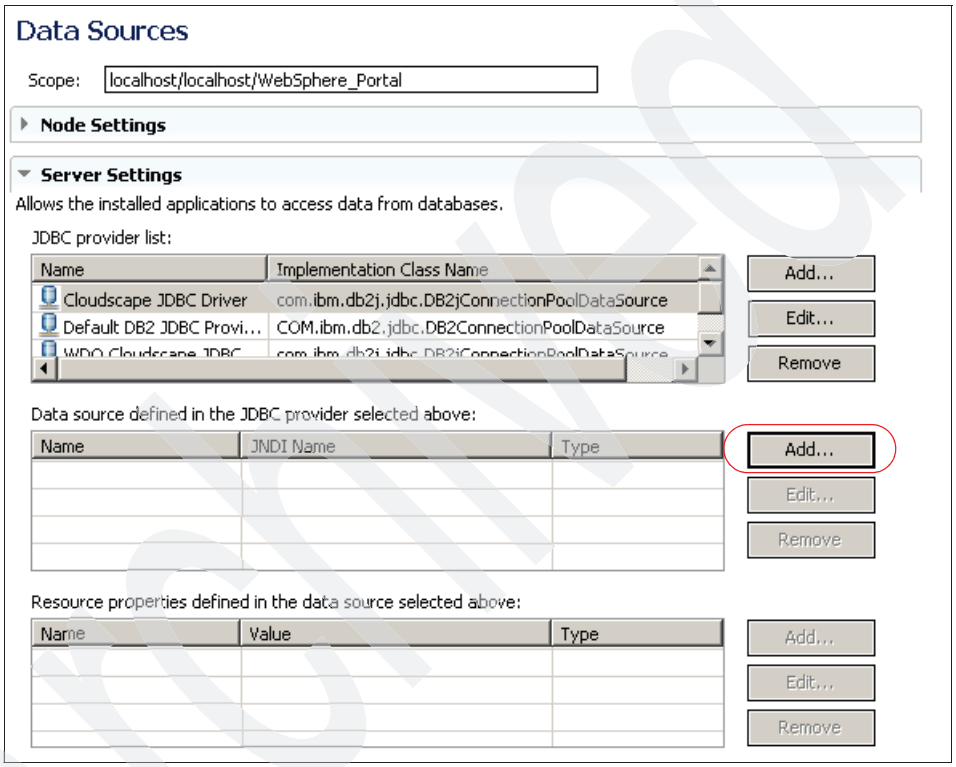


Figure 23-29 Create Data Source

6. In the Create a Data source window, select **Cloudscape JDBC provider** and click **Next**.
7. Enter the following values for the `Modify Data Source` window.
   - Name: `jdbc/WSSample`
   - JDBC Name: `WSSample`

Archived

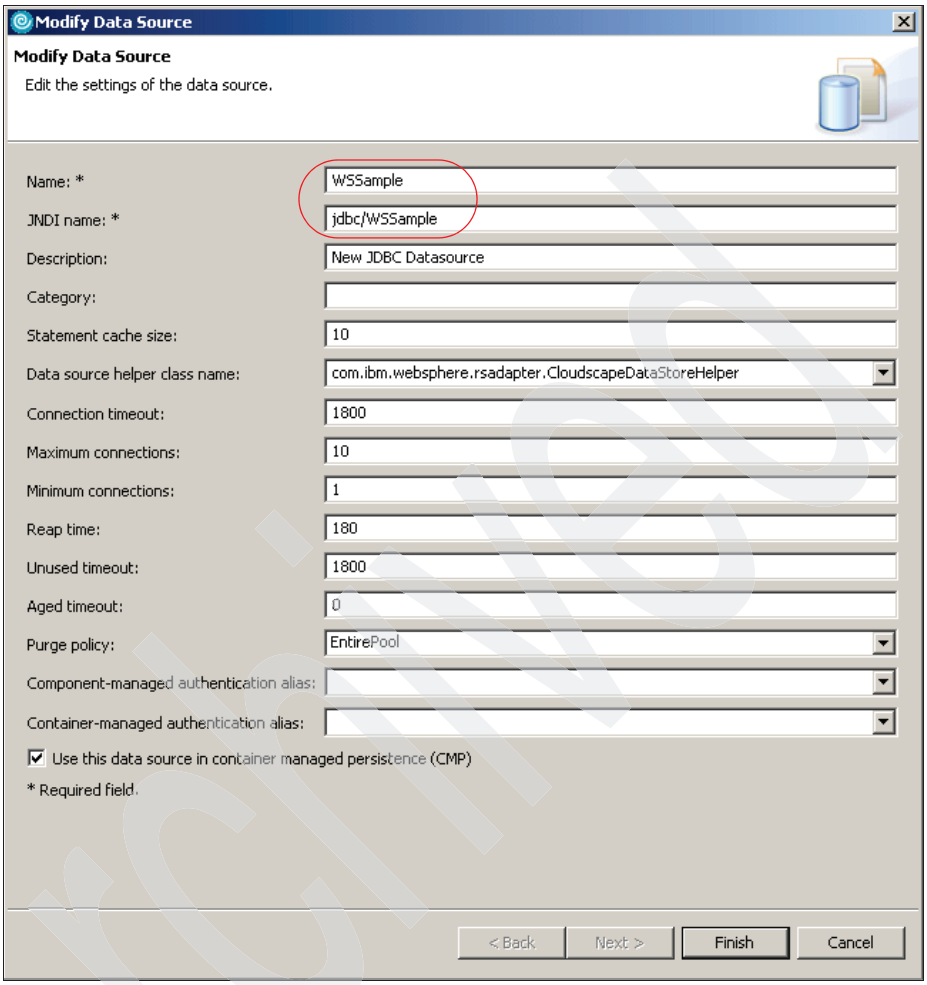


Figure 23-30

8. Click **Next**.
9. Select **databaseName** from Resource Properties, enter `C:\HRproject\database\WSSAMPLE` as the value (or the path where the database is stored).

Archived

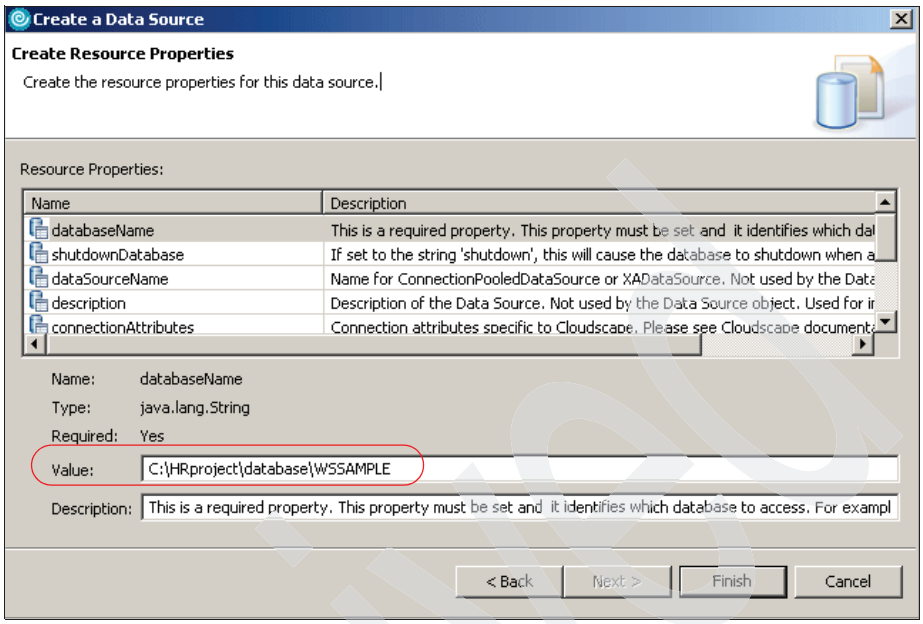


Figure 23-31 Create resource properties

10. Click **Finish**.
11. Close the Server configuration editor.
12. A window will prompt you if you want to save changes. Select **Yes.**

Important: Close the connection created on the previous section, otherwise this can cause conflict error on the WebSphere Portal Server V5 Test Environment since this create the same connection to the database through the data source just created.

23.4.5 Running the HRPortlet168 application

After the portlet project creation and after importing the java classes, you will run the HRPortlet168.

1. Run the HRPortlet168 portlet by right-clicking **HRPortlet168** in the Project Explorer view and selecting **Run** → **Run on server...**.
2. In the *Define a New Server* window, select the **Choose an existing server** option and for *Select the server that you want to use*, select **WebSphere Portal V5.1 Test Environment @ localhost**. Also, use default port 9081.

Note: You can also check the **Set server as project default** checkbox, so you will not be prompted again when you run the portlet.

3. The portlet executes and you will see it in the built-in browser.

   The View mode is shown with a message indicating that you have to provide an SQL query; you will also need to switch the portlet into Edit mode (as indicated in Figure 23-32) so you can enter these values.

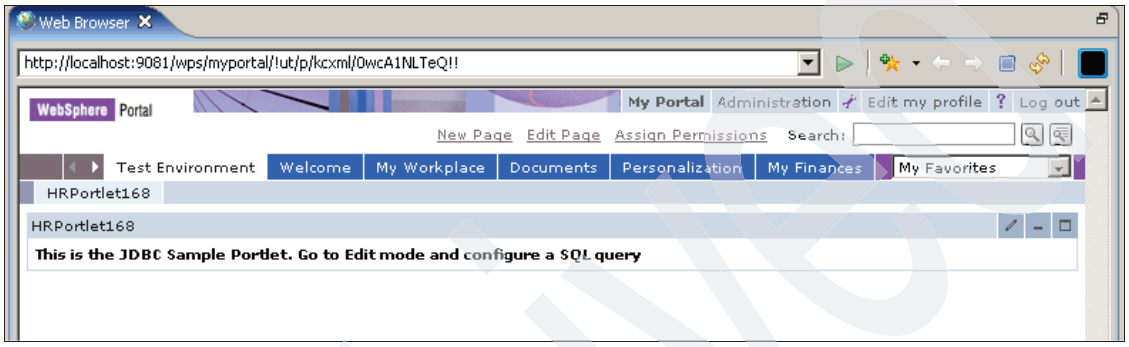


Figure 23-32 Portlet initial window

4. In Edit mode, the JSP for this mode renders the form requesting the database parameters. For the first time, enter the following information and select **Submit**:

   – User: `db2admin`
   – Password: `db2admin`
   – SQL statement: `select * from jobs`

Archive

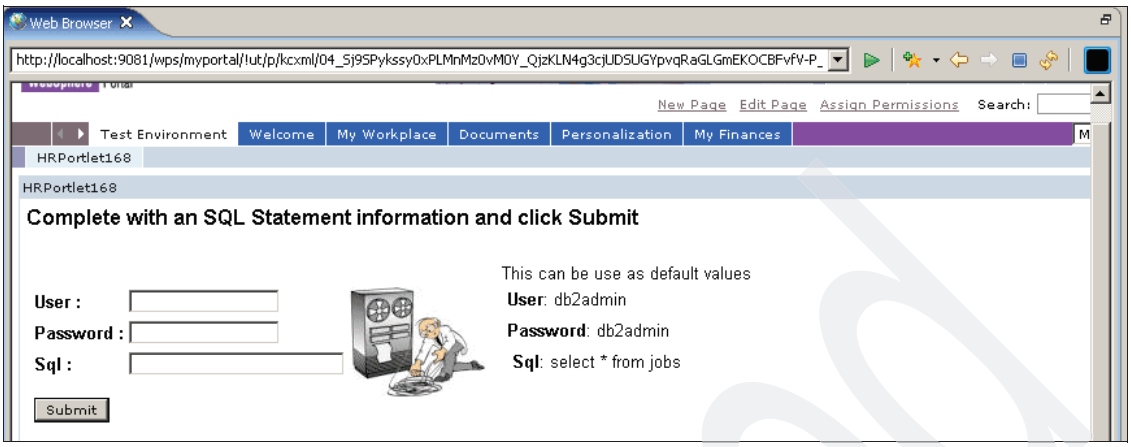
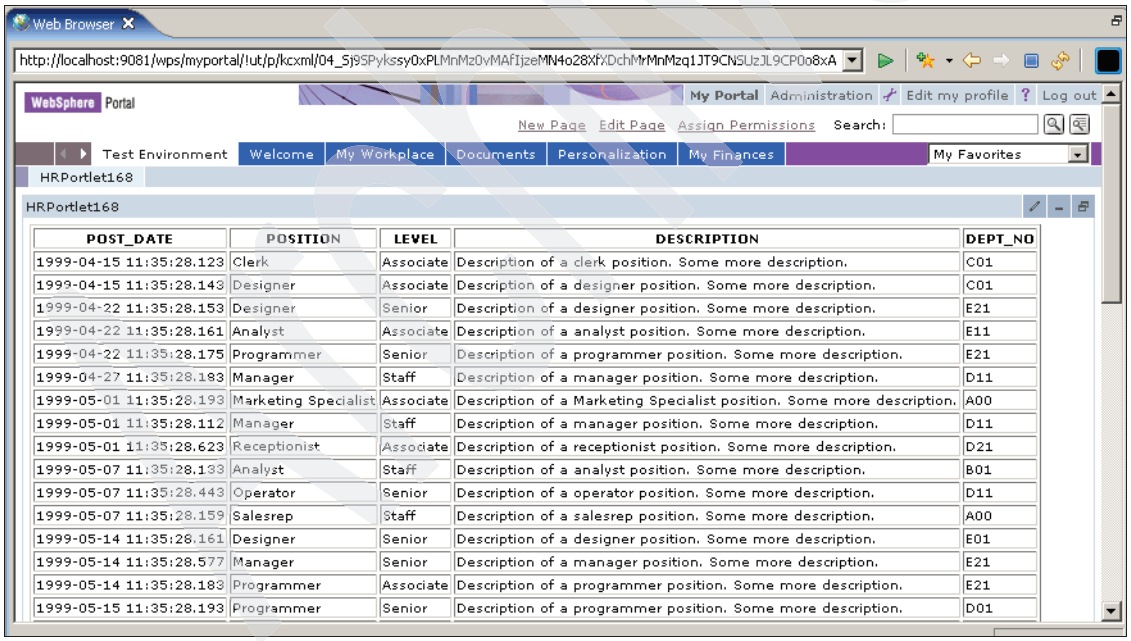


Figure 23-33 Portlet in Edit mode

5. Clicking **Submit** executes the processAction and the portlet will return to View mode showing the results of your query against the WSSAMPLE database.



POST_DATE	POSITION	LEVEL	DESCRIPTION	DEPT_NO
1999-04-15 11:35:28.123	Clerk	Associate	Description of a clerk position. Some more description.	C01
1999-04-15 11:35:28.143	Designer	Associate	Description of a designer position. Some more description.	C01
1999-04-22 11:35:28.153	Designer	Senior	Description of a designer position. Some more description.	E21
1999-04-22 11:35:28.161	Analyst	Associate	Description of a analyst position. Some more description.	E11
1999-04-22 11:35:28.175	Programmer	Senior	Description of a programmer position. Some more description.	E21
1999-04-27 11:35:28.183	Manager	Staff	Description of a manager position. Some more description.	D11
1999-05-01 11:35:28.193	Marketing Specialist	Associate	Description of a Marketing Specialist position. Some more description.	A00
1999-05-01 11:35:28.112	Manager	Staff	Description of a manager position. Some more description.	D11
1999-05-01 11:35:28.623	Receptionist	Associate	Description of a receptionist position. Some more description.	D21
1999-05-07 11:35:28.133	Analyst	Staff	Description of a analyst position. Some more description.	B01
1999-05-07 11:35:28.443	Operator	Senior	Description of a operator position. Some more description.	D11
1999-05-07 11:35:28.159	Salesrep	Staff	Description of a salesrep position. Some more description.	A00
1999-05-14 11:35:28.161	Designer	Senior	Description of a designer position. Some more description.	E01
1999-05-14 11:35:28.577	Manager	Senior	Description of a manager position. Some more description.	E21
1999-05-14 11:35:28.183	Programmer	Associate	Description of a programmer position. Some more description.	E21
1999-05-15 11:35:28.193	Programmer	Senior	Description of a programmer position. Some more description.	D01

Figure 23-34 Query results

6. Enter Edit mode again. Notice that the database name, user ID, password and SQL statement, which have been stored in session object, are persistent.

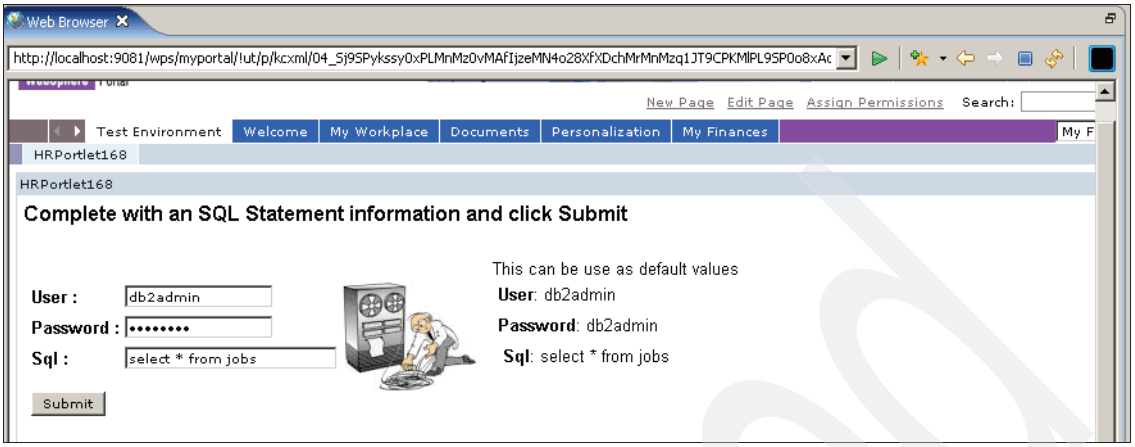


Figure 23-35 SQL query statement

7. Enter a new query, for example `select * from survey`, and click **Submit**. You will be presented with the results of your new query.



Figure 23-36 Query results

Archived

Archived

24

IBM API declarative cooperative portlets

This chapter describes the architecture and development of cooperative portlets. Cooperative portlets placed on a portal page can be developed independently but they interact with one another and share the same information. This enables an advanced user experience scenario where portlets automatically react to events and actions originated from other portlets.

After reading this chapter, you will be able to:

- Understand the architecture and value of cooperative portlets
- Develop source cooperative portlets with IBM Portlet API
- Develop target cooperative portlets with IBM Portlet API

Note: The sample scenarios included in this chapter require that you have completed Chapter 22, “Accessing JDBC databases from portlet applications” on page 669. You can also download the sample code available as additional materials. See Appendix A, “Additional material” on page 1003.

© Copyright IBM Corp. 2005. All rights reserved.

24.1 Overview

Cooperative portlets subscribe to a model for declaring, publishing and sharing information with each other using the WebSphere Portal property broker. At runtime, the property broker matches the data type of output properties from a source portlet with the data type of input properties from one or more target portlets. The transfer of the property can be initiated using the Portlet Wiring Tool. A user creates a persistent connection between two portlets, called a wire.

For IBM portlets a user can launch a Click-to-Action event from an icon on the source portlet. The icon presents a pop-up menu containing the list of targets for the action. Selecting such a menu item results in the execution of the actionPerformed() method on a target cooperative portlet. Figure 24-1 illustrates an example of such a cooperative portlet menu.

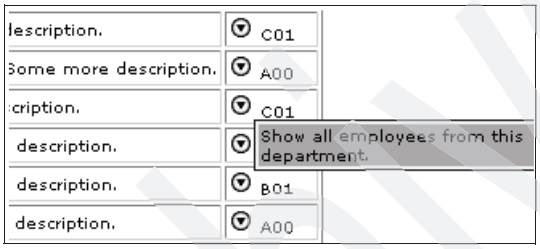


Figure 24-1 Click-to-Action menu for IBM portlets

In Figure 24-1, one piece of information, for example a department number, can be sent to a target portlet displaying all employees from the selected department number or to a portlet displaying detailed information of the selected department. In addition, the cooperative portlet technology also enables the broadcast of data to multiple portlets by sending multiple property values with only one click. The transfer of properties can be saved as wires using the **Ctrl** key. So the next time the user clicks the icon, the saved menu selection is used without prompting the user.

Note: An IBM portlet cannot be wired to a JSR 168 compliant portlet and vice versa.

Archived

Portlet messaging versus cooperative portlets

In general, both portlet messaging and cooperative portlets can be used to share data between two or more portlets. The most important difference is that cooperative portlets are more loosely coupled than portlets using messaging. Cooperative portlets do not have to know the name of the target portlet even if they do not broadcast data. The matching of source and target portlets is done at runtime based on registered properties and actions. Cooperative portlets can also include a menu with a list of executable portlet actions. For this menu, no additional programming is needed because it is part of the C2A tag library. It is recommended using cooperative portlets instead portlet messaging

24.1.1 The WebSphere Portal property broker

During runtime, the WebSphere Portal property broker is responsible for enabling cooperative portlets. This is done by matching the data type of output properties from a source portlet with the data type of input properties from one or more target portlets. Figure 24-2 shows the relationships between the two portlets and properties.

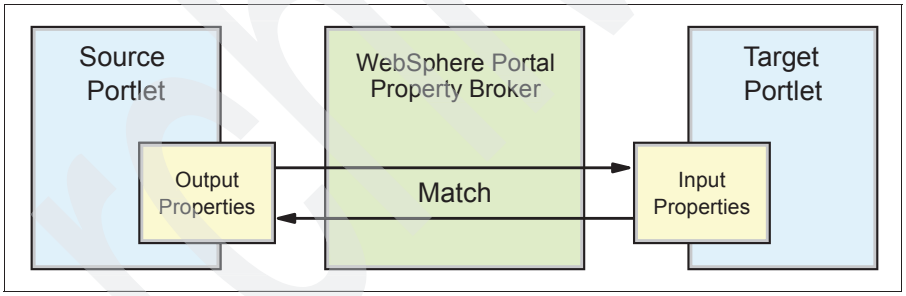


Figure 24-2 The property broker matches input and output properties

Target portlets optionally provide actions to process the properties that it receives. There is no difference between an action initiated by the portlet itself as mentioned in Chapter 6, “IBM Portlet API portlet development” on page 203 and an action initiated by a source cooperative portlet.

Cooperative portlets can be source portlets, target portlets, or both.

- Source portlets identify to the property broker properties which they are able to share with other portlets.
- Target portlets identify to the property broker actions which are able to process properties contributed by other portlets.

24.1.2 Property broker runtime components

To enable your portlet for cooperation as well as a broker component, you have to wrap your portlet class with a generic wrapper portlet. The wrapper portlet is only used for IBM portlets. This wrapper intercepts calls and interfaces with the broker. The wrapper is packaged in each cooperative portlet's WAR file. For JSR 168 compliant portlets the function of the wrapper is provided through a portlet filter which is part of the runtime component and not the WAR file.

24.2 IBM Portlets for cooperation

Cooperative portlets can use a declarative or programmatic approach, or a combination, to register and publish properties to the property broker. The programmatic approach to publish properties is discussed in Chapter 25, "IBM API programmatic cooperative portlets" on page 771. The declarative approach is simpler. Few changes need to be made to existing portlets to enable them to interact with other cooperative portlets on the page. Existing portlets that already use action processing simply declare their actions to the property broker using WSDL. Figure 24-3 shows how to develop source cooperative portlets.

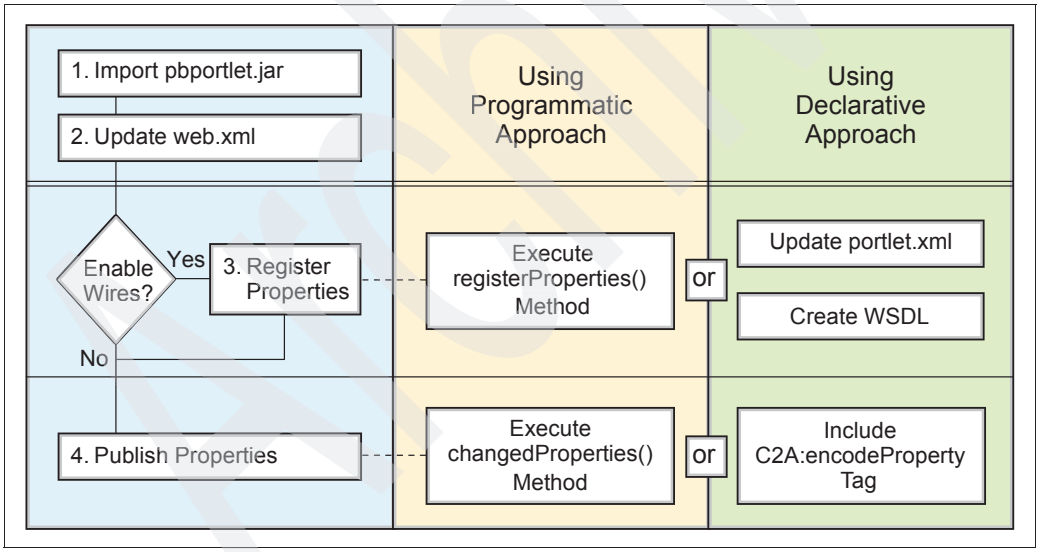


Figure 24-3 Steps to program a source cooperative portlet

In general terms, there are two steps at runtime to establish cooperative portlet communication:

1. All properties must be registered with the cooperative broker. This can be done by using the declarative approach which includes the creation of a Web

Service Description Language (WSDL) file and the configuration of the portlet deployment descriptor. To register properties programmatically, you can use the property broker API. Please note that registration of properties can only be done during the event phase of the request-response cycle.

2. Notify the property broker about property changes. The easiest way to achieve this is to include the *encodeProperty* tag in your JavaServer Page. As an alternative, in the programmatic approach, you will use the *changedProperties()* method to publish properties.

   Also, in the programmatic approach, you have to configure a wire before you publish properties. Figure 24-4 shows how to develop target portlets.

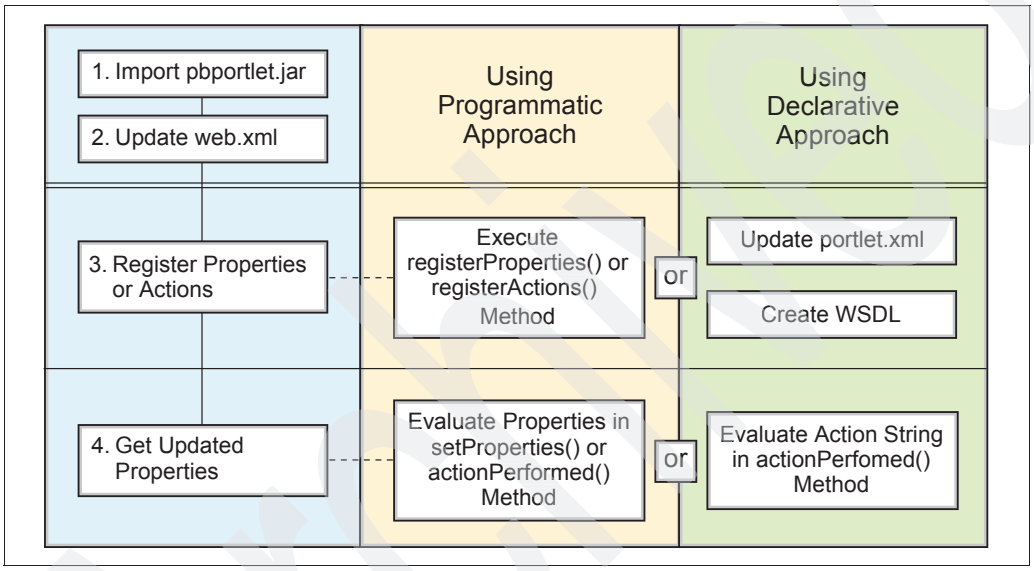


Figure 24-4 Steps to create a target cooperative portlet

To get information about changed properties, target portlets register properties and, optionally, actions with the property broker. When using the declarative approach both properties and actions are always registered. When using the programmatic approach, you can register properties without any actions.

In addition, during runtime process such programmatic target portlets are notified about property changes by using the *setProperties* method from the *PropertyListener* interface instead of the actionPerformed method.

24.2.1 Registering and publishing properties

For a portlet to be a source of data, programmers can use a custom JSP tag library to flag sharable data on their output pages. The tags require a data type to

be specified as well as a specific value corresponding to an instance of this type. If you want to use wires source portlets, you must register properties by using a declarative or programmatic approach.

Target portlets associate their actions with an input property which has been declared as an XML type. The actions are declared using WSDL, with a custom binding extension which specifies the mapping from the abstract action declaration to the actual action implementation. Associated with each action is a single input parameter described by an XML type and zero or more output parameters, each described by an XML type. Each input or output parameter encapsulates exactly one property. The input property's type is used for matching the action to sources, and its value is filled in when the end user triggers the action using Click-to-Action. The output parameters, if specified, are used to automatically trigger other compatible actions (ones which can consume the same type) on other wired portlets every time the action executes (this may be used to trigger chains of related actions).

Note: The location of the WSDL file is configured as a portlet parameter.

24.2.2 Struts integration

Portlets developed using Struts can take advantage of the property broker by following the steps of sample scenario with the following differences:

- In the web.xml file
  - modify servlet class to `com.ibm.wps.pb.wrapper.PortletWrapper`
  - Add an initialization parameter named `c2a-application-portlet-class` with value `com.ibm.wps.portlets.struts.WpsStrutsPortlet`.
- Struts actions need to be declared in a WSDL file by setting the type attribute on the action element to `struts`.

```
<portlet:action name="/nnnStruts.do" type="struts" caption="caption.text"
description="description.text"/>
```

Note: You can download a sample code available as additional materials using cooperative portlets with struts. See Appendix A, “Additional material” on page 1003.

24.2.3 Internationalization

To support translation for captions and descriptions associated with shared properties, portlets must provided resource bundles in the appropriate location in the WAR file. For IBM portlets, the resource file name to be used can be

specified using a configuration parameter in the portlet.xml file called c2a-nls-file. The value should be the file name, including the package but omitting the .properties suffix. See Example 24-4 on page 763.

For portlets using the programmatic approach for registration, `setTitleKey()` and `setDescriptionKey()` methods can be used to set caption and descriptions respectively.

24.3 Sample scenario (IBM portlets)

The sample application shown in this section is based on the HRPortlet from Chapter 22, “Accessing JDBC databases from portlet applications” on page 669. Two different versions of the HRPortlet will be used, as follows:

- The source cooperative portlet HRPortlet displays a list of jobs.
- The target cooperative portlet Employee Details Portlet displays a list of employees working in the same department.

Using the cooperative portlet technology, users can select a department number from the source cooperative portlet. After that, WebSphere Portal updates the target portlet displaying all employees from the selected department.

24.3.1 Description

This sample scenario provides step-by-step exercises to create a portlet project to work as a Click-to-Action source portlet. You will also create a portlet project to act as a source Click-to-Action portlet. You will create, deploy and run the portlet application. This exercise will allow you to understand the techniques used to develop portlets with Click-to-Action features using the C2A declarative approach.

Cooperative portlets subscribe to a model for declaring, publishing, and sharing information with each other using the WebSphere Portal property broker. Portlets subscribe to the broker by publishing typed data items, or properties, that they can share, either as a provider or as a recipient.

- The portlet that provides a property is called the source portlet.
- The properties that the source portlet publishes are called output properties.
- The portlet that receives a property is called the target portlet.
- The properties that are received by the target are called input properties.

The target portlets optionally provide actions to process the properties that they receive. Action processing in target portlets does not need to distinguish between an action initiated within its own portlet area and an action initiated by the transfer of a portlet property value. Each action is associated with a single

input parameter and zero or more output parameters, which provide information to the action about the objects in which the property value should be bound, such as the request or the session. Each parameter is associated with exactly one property. Parameters associated with input properties are called input parameters, while those associated with output properties are called output parameters. Instead of actions, the target portlet can receive property changes directly through the *PropertyListener* interface.

At runtime, the property broker matches the data type of output properties from a source portlet with the data type of input properties from one or more target portlets. If a match is determined, the portlets are capable of sharing the property. The actual transfer of the property can be initiated by one of the following methods:

- A user launches a Click-to-Action event from an icon on the source portlet. The icon presents a pop-up menu containing the list of targets for the action. After the user selects a specific target, the property broker delivers the data to the target in the form of the corresponding portlet action. Using the Click-to-Action delivery method, users can transfer data with a simple click from a source portlet to one or more target portlets, causing the target to react to the action and display a new view with the results. The user can also broadcast the property to all portlets on the page that have declared an action associated with a matching input property.
- A user holds down the **Ctrl** key while clicking an action and chooses to have the selection saved persistently as a connection between two portlets, called a wire. If a wire is present the next time the user clicks the icon, no selection menu is shown. Instead, the wired action(s) is/are automatically fired. Subsequent updates to that property are transferred without further deliberate user choice. Wires can also be created using the Portlet Wiring Tool.
- The source portlet can perform a programmatic publish of properties to the broker when it determines that property values have changed. Such property values are transferred to the target(s) only if wires have been created.

Cooperative portlets can be source portlets, target portlets, or both.

- Source portlets identify to the property broker properties which they are able to share with other portlets.
- Target portlets identify to the property broker actions which are able to process properties contributed by other portlets.

Arbuso

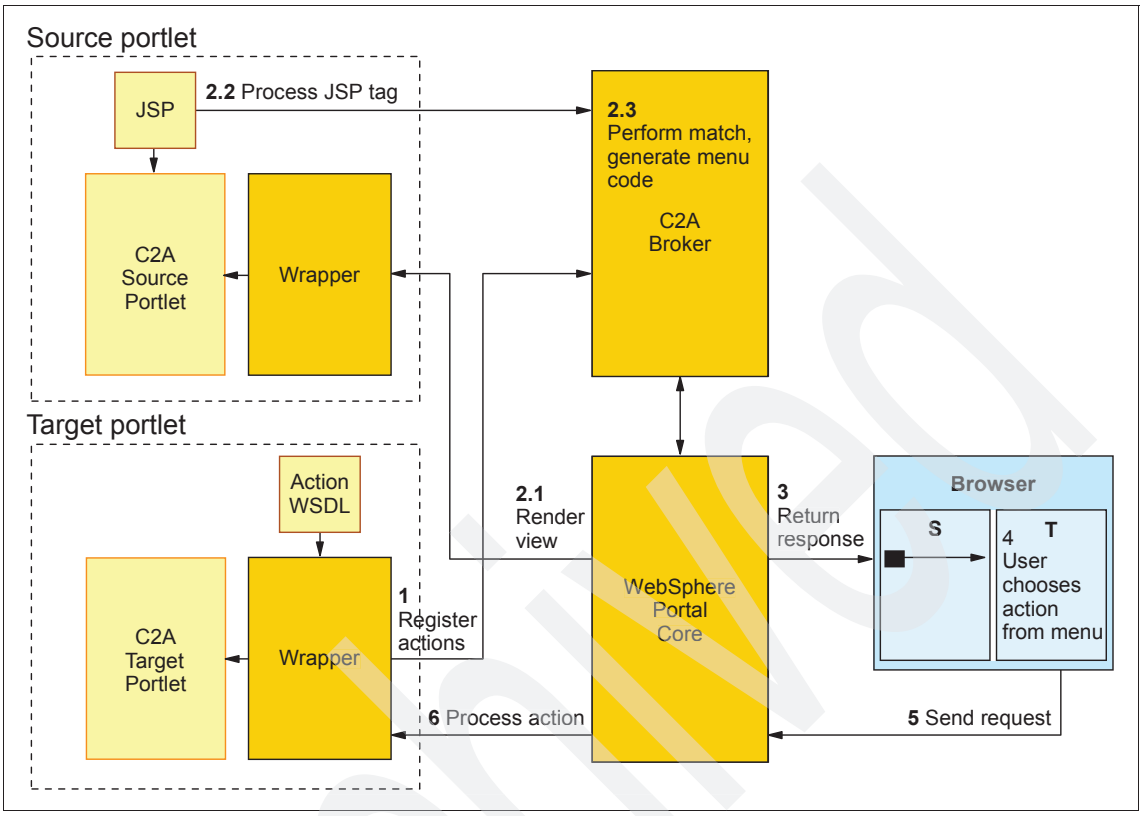


Figure 24-5 Click-to-Action architecture

The sequence flow for this sample scenario is as follows:

1. At portlet initialization time, the C2A wrapper processes any action WSDL file associated with the application portlet and registers the actions with the C2A broker.
2. During the render phase of a request cycle, JSPs associated with C2A source portlets are processed. The custom C2A tags produce calls to the C2A broker, which examines the type information to determine matching actions. The broker generates additional code to create an icon to be used to display a pop-up menu of actions, and adds code to dispatch actions on portlets upon user selection from the menu.
3. After all render phase portlet callbacks are complete, the WebSphere Portal core assembles the response page and returns it to the client (for example, a browser).
4. When the user clicks the C2A icon for a source, he or she sees a menu of compatible actions (on the page) and selects one.

Archived

5. The client (for example, a browser) generates a new request containing the chosen source and action information and sends it to the WebSphere Portal Server.
6. The WebSphere Portal Core delivers the action to the target portlet. The action is intercepted by the wrapper, which may interact with the broker to further process the request before delivering the action to the target.

All portlet actions are intercepted by the wrapper; however, actions which are invoked through direct interaction with the portlet (as opposed to interaction through the C2A menus) are passed through transparently to the portlet. In more advanced scenarios, such as the broadcast and scatter mentioned earlier, there will be more interactions between the wrappers and the broker to determine the appropriate target set and deliver the right data to the targets.

24.3.2 Source cooperative portlet

In this section, you will create a cooperative source portlet using the wizard.

You can also import an existing portlet project and execute the following tasks to enable it to act as a Click-to-Action source portlet using the declarative approach:

1. Import the original portlet if it is not in your workspace. In this scenario, the JDBC portlet (HRPortlet) from Chapter 22, “Accessing JDBC databases from portlet applications” on page 669 will be used.
2. You will import the property broker jar file (pbportlet.jar).
3. You will update web.xml to refer to the property broker classes.
   a. The servlet class entry should specify the `com.ibm.wps.pb.wrapper.PortletWrapper` class in the property broker:

      ```
      <servlet-class>com.ibm.wps.pb.wrapper.PortletWrapper</servlet-class>
      ```

   b. The original portlet application class should also be specified in the c2a-application-portlet-class initialization parameter. For example:

      ```
      <init-param>
      <param-name>c2a-application-portlet-class</param-name>
      <param-value>hrportlet.HRPortlet</param-value>
      </init-param>
      ```

4. You will update the JSP for View mode to include Click-to-Action menus.

Figure 24-6 on page 751 illustrates the source cooperative portlet for this sample scenario.

Archive

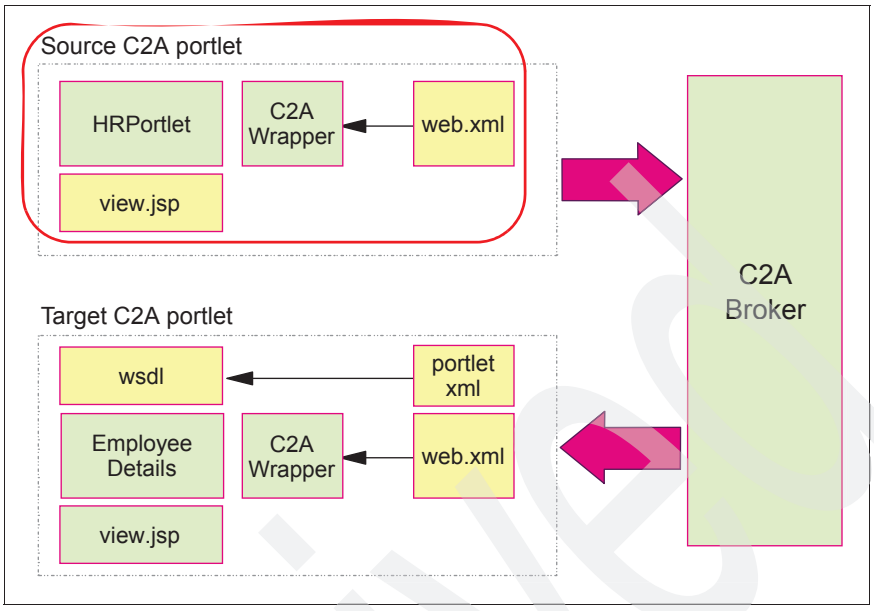


Figure 24-6 Cooperative portlets - sample scenario

Create a C2A source portlet

Follow these steps to create a new portlet project:

1. Select **File** → **New** → **Portlet Project**
2. In the Portlet Project window:
   - Specify `HRPortlet` as the portlet name
   - Select **WebSphere Portal V5.1** as the target server.

   Click **Next**.

Archive

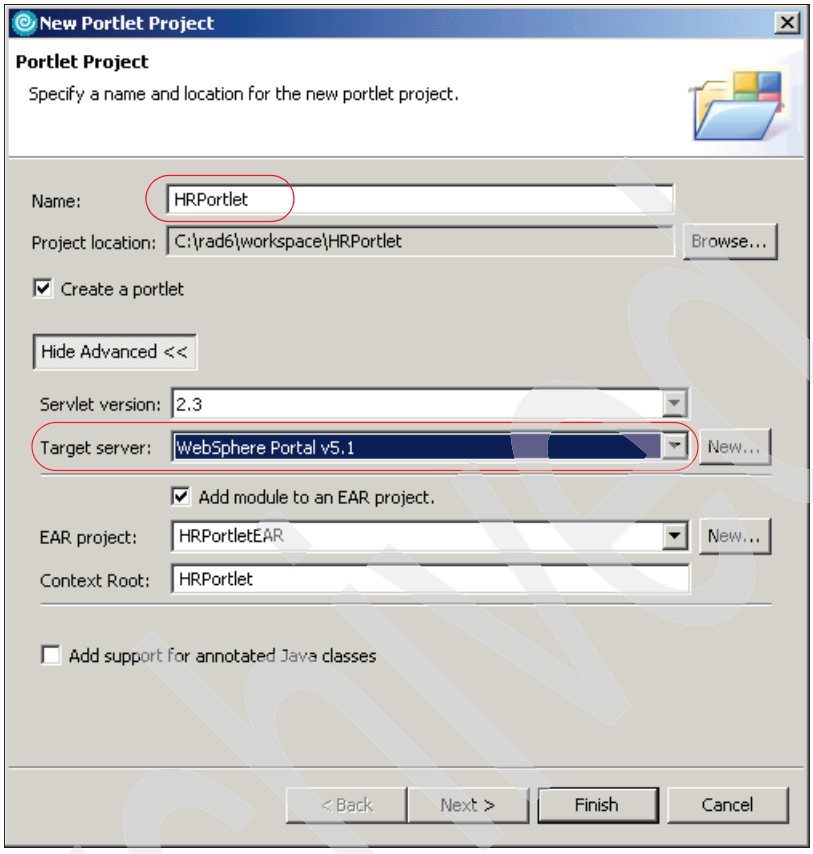


Figure 24-7 Create a new portlet project

3. In the next window, select **Basic portlet** as the Portlet Type. Click **Next**.
4. In the Features window, leave all Web project features unchecked. Click **Next**.
5. In the Portlet Settings window, select **Change code generation options** and enter `HRPortlet` as the Class prefix instead of `HRPortletPortlet`. Click **Next**.
6. In the Event Handling window, check **Enable cooperative source** and **Add Click-to-Action taglib**. Uncheck the rest of the options.

Archive

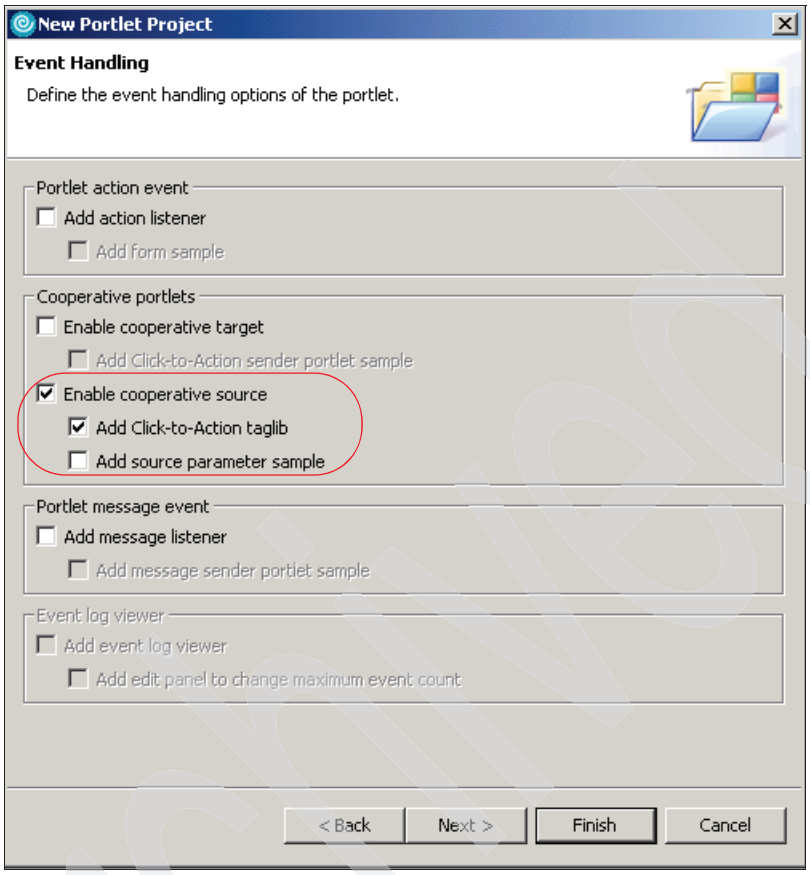


Figure 24-8 Enable cooperative source

7. Click **Next** until the last window where you have to check **Add Edit mode** in Additional modes. Click **Finish**.

Examining the generated code

Now take a minute to review the code generated by the wizard:

1. In the Project Explorer panel, expand the **HRPortlet\Java Resources** folder; you will see the property broker jar file (pbportlet.jar) has been imported.

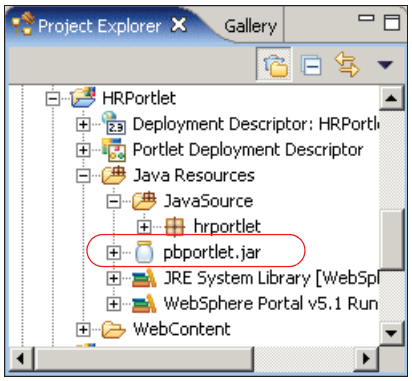


Figure 24-9 pbportlet.jar

2. Select **web.xml** under the HRPortlet\WebContent\WEB-INF folder and open it. Select the **Servlets** tab and check the following values:
   - Servlet class: `com.ibm.wps.pb.wrapper.PortletWrapper`
   - The value of Initialization param `c2a-application-portlet-class` is the portlet name including the package: `hrportlet.HRPortlet`

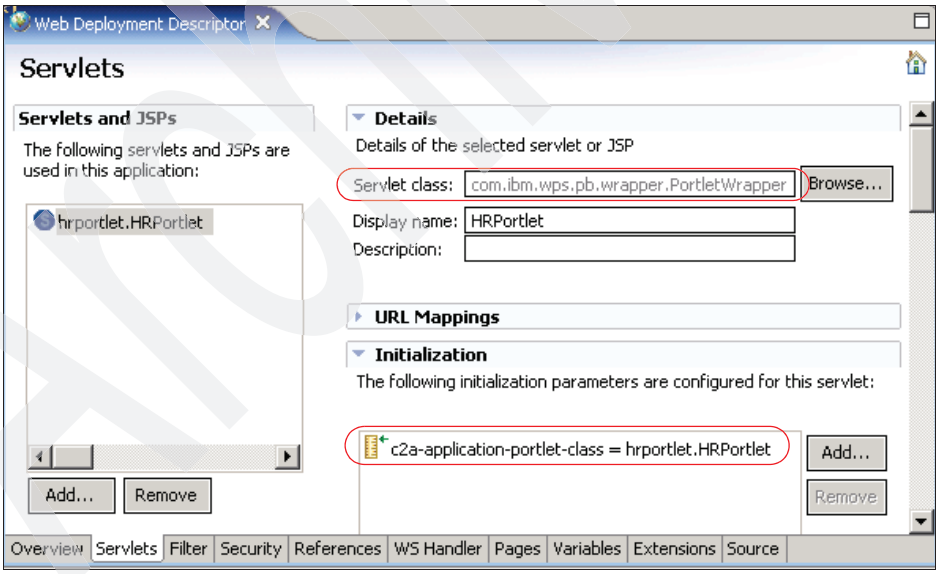


Figure 24-10 Reviewing web.xml file

3. Close the web.xml file.

4. In the Project Explorer, expand **HRPortlet\WebContent\hrportlet\jsp\html** and double-click **HRPortletView.jsp**. You will see the c2a.tld taglib included in the file:

```
<%@ taglib uri="/WEB-INF/tld/c2a.tld" prefix="C2A" %>
```

Importing the HRPortlet portlet

The HRPortlet portlet from the JDBC chapter (see Chapter 22, “Accessing JDBC databases from portlet applications” on page 669) will be used as a base for this sample scenario.

Unzip the HRPortlet.war and import or copy the following files into the directory containing the application. Be sure to import or copy them in the right directory, when you are prompted to overwrite existing files click yes:

- In HRPortlet\JavaSource\hrportlet copy all the files of the hrportlet folder.
- Copy the utilities folder in HRPortlet\JavaSource.
- In HRPortlet\WebContent\hrportlet\jsp\html copy HRPortletEdit.jsp and HRPortletView.jsp.
- Copy the images folder in the HRPortlet\WebContent directory
- Copy the database folder in the HRPortlet\WebContent\WEB-INF directory
- Copy dbbeans.jar in the HRPortlet\WebContent\WEB-INF\lib directory.

In Rational Application Developer, right-click in the **HRPortlet** project and select **Refresh**.

You will see your directory structure with the new files. Some errors appear; to fix them, open, for example, the HRPortletViewBean.java file under the HRPortlet\Java Resources\JavaSource\hrportlet directory, right-click the Java editor and select **Source** → **Organize Imports**. This should fix the errors.

Update JSP for C2A enablement

Since we replaced the existing HRPortletView.jsp containing the c2a.tld taglib, we need to include it now. You also need to include the c2a.tld taglib if you are enabling an existing portlet project as a source cooperative portlet.

1. In the Project Explorer, expand **HRPortlet/Web Content/hrportlet/jsp/html** and double-click **HRPortletView.jsp**.

Archived

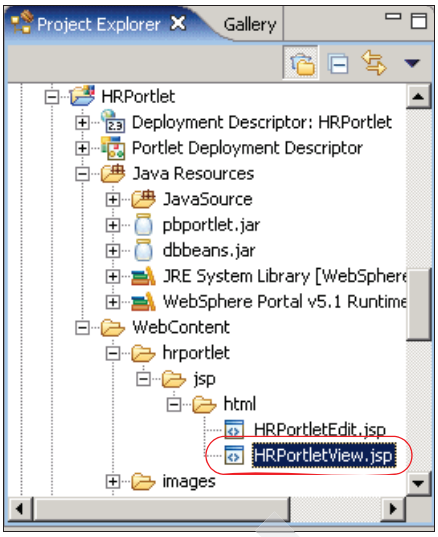


Figure 24-11 Selecting HRPortalView.jsp

2. In the JSP editor, switch to the Source tab.

   Note: Source portlets can publish their output properties by inserting tags from a custom JSP library in their JSPs. A JSP tag library is provided to allow source properties to be identified in JSPs.

3. In the third line of this JSP, include the following line:

   ```
   <%@ taglib uri="/WEB-INF/tld/c2a.tld" prefix="C2A" %>
   ```

   For example, see Figure 24-12.

HRPortletView.jsp

```
<%@ page contentType="text/html" import="java.util.*, hrportlet.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ taglib uri="/WEB-INF/tld/c2a.tld" prefix="C2A" %>
<portletAPI:init />
```

Figure 24-12 Adding tag library c2a.tld

4. Two JSP tags can be used to declare output properties in the source portlet:
   - **<c2a:encodeProperty/>**

     Uses source data and type information to insert markup that displays the icon, generating a pop-up menu.
   - **<c2a:encodeProperties/>**

Archived

Used to enclose normal HTML markup and one or more encodeProperty tags with the markup. This tag is provided to support the scatter scenario, where a user can optionally send more than one unit of data to target portlets.

You will now declare the output properties. Scroll down to the following lines:

```
<TD>
   <P><%=results.getCacheValueAt(row, col)%></P>
</TD>
```

Change the line so it looks as follows:

```
<P>
   <C2A:encodeProperty
name="<%=results.getColumnName(col).toString()+\"Param\"%>"
   namespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
   type="<%=results.getColumnName(col)%>"
   value="<%=results.getCacheValueAt(row, col).toString()%>" />
   <%=results.getCacheValueAt(row, col)%>
</P>
```

Note: As you can see, the table column name is used as an output property type. Therefore, a target portlet using the specified namespace should provide an inbound property with this name.

Save all your files (you can use **Ctlr-S**).

Note: At this time, you have your source cooperative portlet enabled for Click-to-Action but you will not be able to establish a wire between source and target portlets.

Implementing a wire

To prepare the source cooperative portlet to establish a persistent connection with the target cooperative portlet do the following steps:

1. Instead of inserting the C2A:encodeProperty tag manually, use the Palette panel provided by Rational Application Developer. This will create a wsdl file and update the portlet.xml to specify the path for that wsdl file. You need to open the HRPortletView.jsp to see the Palette contents. Figure 24-13 on page 758 shows this Palette.

Archived

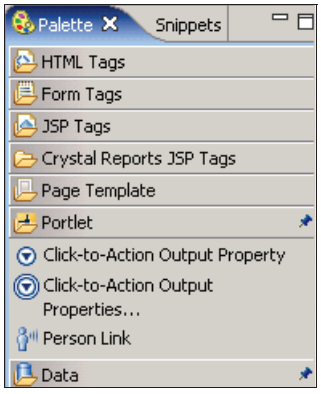


Figure 24-13 Palette

In HRPortletView.jsp scroll down to the following lines and locate the cursor after the <P> tag:

```
<TD>
   <P><%=results.getCacheValueAt(row, col)%></P>
</TD>
```

2. Right-click **Click to Action Output Property** and click **Insert** in the popped menu.

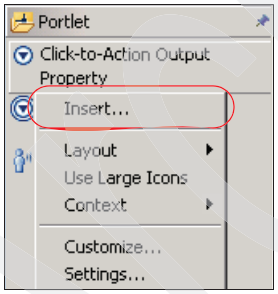


Figure 24-14 Insert Click-to-Action output property in a jsp

3. The window illustrated in Figure 24-15 on page 759 appears. Enter the following values:
   a. Data type: `DEPT_NO`
   b. Namespace: `http://www.ibm.com/wps/c2a/examples/hrdetails`
   c. Source portlet: `HRPortlet portlet`

   Click **OK**.

Archived

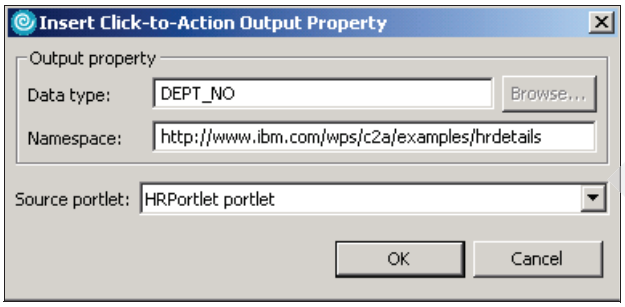


Figure 24-15 Inserting Click-to-Action output property

4. The following code has been generated in the jsp:

```
<C2A:encodeProperty type="DEPT_NO"
   namespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
   name="DEPT_NO"></C2A:encodeProperty>
```

   Change the HRPortletView.jsp code to display the C2A menu in the DEPT_NO column. Example 24-1 shows the snippet code for jsp, note that some attributes have been modified.

Example 24-1 Snippet code for jsp

```
for(int col=1; col<=results.getColumnCount(); col++) { %>
      <TD><P>
      <% if (results.getColumnName(col).equalsIgnoreCase("DEPT_NO")) { %>
         <C2A:encodeProperty
            name="DEPT_NOParam"
            namespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
            type="DEPT_NO"
            value="<%=results.getCacheValueAt(row, col).toString()%>" />
      <% } %>
      <%=results.getCacheValueAt(row, col)%>
      </P></TD>
<% }
```

5. Under the HRPortlet\WebContent\wsdl folder, you can see the **HRPortletportlet.wsdl gen**erated. Open this file and modify its content as indicated in Example 24-2.

Example 24-2 WSDL file associated to source cooperative portlet

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HRPortletportlet_Service"
   targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:portlet="http://www.ibm.com/wps/c2a"
```

Archived

```
   xmlns:tns="http://www.ibm.com/wps/c2a/examples/hrdetails"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <types>
      <xsd:schema
targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails">
         <xsd:simpleType name="DEPT_NO">
            <xsd:restriction base="xsd:string"></xsd:restriction>
         </xsd:simpleType>
      </xsd:schema>
   </types>
   <message name="DEPT_NO_Response">
      <part name="DEPT_NO_Output" type="tns:DEPT_NO" />
   </message>
   <portType name="HRPortletportlet_Service">
      <operation name="HRPortletportlet">
         <output message="tns:DEPT_NO_Response" />
      </operation>
   </portType>
   <binding name="HRPortletportlet_Binding"
      type="tns:HRPortletportlet_Service">
      <portlet:binding />
      <operation name="HRPortletportlet">
         <portlet:action type="simple"/>
         <output>
            <portlet:param name="DEPT_NOParam" partname="DEPT_NO_Output"
boundTo="request-parameter" caption="Department ID" description="Department
ID"/>
         </output>
      </operation>
   </binding>
</definitions>
```

The wsdl file registers the output property `DEPT_NOParam`. The value `simple` of the type attribute in the <portlet:action> element indicates a simple portlet action String is used.

6. If you open the portlet.xml file, you will also see that a reference to wsdl file has been included as a parameter under the concrete portlet application.

Example 24-3 Referencing wsdl file in portlet.xml

```
   <config-param>
      <param-name>c2a-action-descriptor</param-name>
      <param-value>/wsdl/HRPortletportlet.wsdl</param-value>
   </config-param>
```

Archived

24.3.3 Target cooperative portlet

In this section, you will create a target cooperative portlet using the wizard and import source files from HRPortlet. This target portlet will execute a fixed SQL statement with a variable *where* clause.

The code for the target portlet class must meet the following requirements:

- The action must be implemented either as a portlet action or a Struts action. For portlet actions, you should use the simple action Strings rather than the deprecated DefaultPortletAction class.
- Portlet actions must accept a single parameter. The parameter may appear as a request parameter, a request attribute, a session attribute, or an action attribute (deprecated), as specified in the action declaration or registration.

The HRPortlet is already prepared for this situation, so only a few changes are needed in the portlet class code.

Figure 24-16 illustrates the target cooperative portlet for this sample scenario.

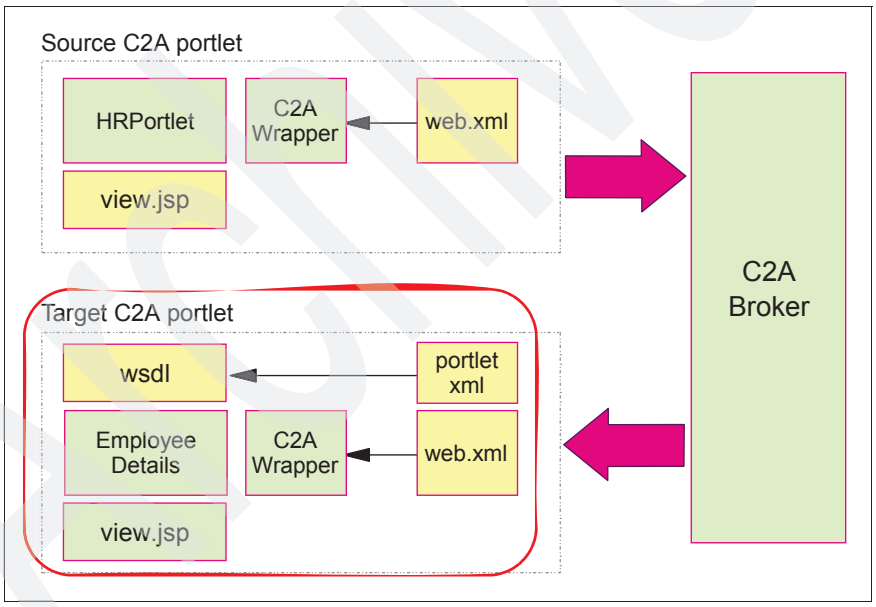


Figure 24-16 Cooperative portlets - sample scenario

To create the target portlet project, proceed as follows:

1. In Rational Application Developer, select **File** → **New** → **Portlet Project**.

Archived

2. In the Create a Portlet Project window, enter a project name of `EmployeeDetailsPortlet` and select **WebSphere Portal V5.1** as the target server. Click **Next**.
3. In the Portlet Type window, select the **Basic** portlet. Click **Next**.
4. In the next window, uncheck the **Web diagram** feature and click **Next**.
5. In the Portlet Settings window, select **Change code generation options** and enter the following values:
   - Package prefix: `hrportlet`
   - Class prefix: `HRPortlet`

   Click **Next**.

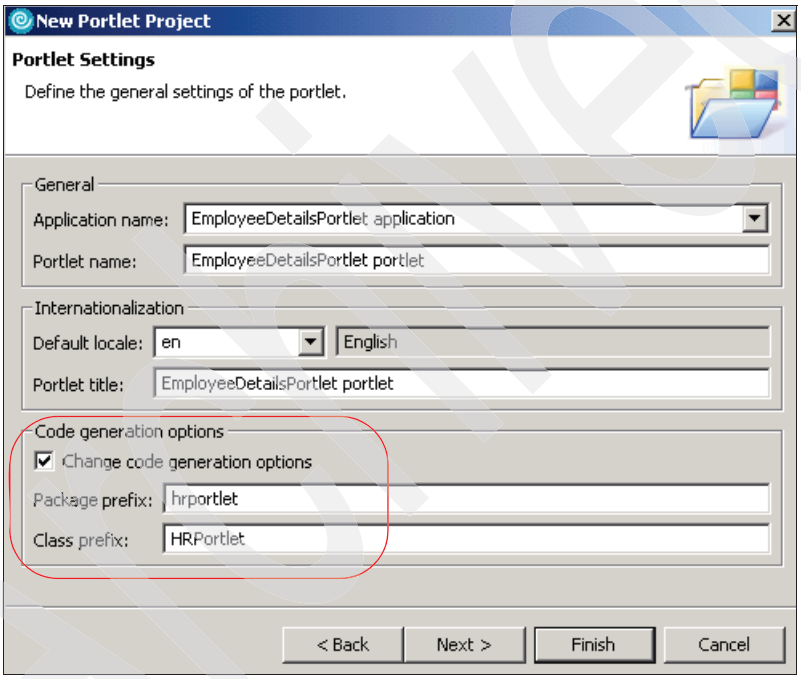


Figure 24-17 Portlet Settings page

6. In the Event Handling window, check **Enable cooperative target** in the Cooperative Portlets section. Click **Next**.
7. Click **Next** again and in the last window check **Add Edit mode** in Additional modes.
8. Click **Finish**.

Archived

Reviewing the code generated by the wizard for the target portlet

Look at the code generated by the wizard:

- In the Project Explorer panel, expand the **EmployeeDetailsPortlet\Java Resources** folder; you will see the property broker jar file (pbportlet.jar) has been imported.
- Open the web.xml file and select the **Servlets** tab. Select the **hrportlet.HRPortlet** servlet and check the following entries:
  - servlet class: com.ibm.wps.pb.wrapper.PortletWrapper
  - There is an initialization parameter configured with name c2a-application-portlet-class and value `hrportlet.HRPortlet`.
- Open portlet.xml. Under the concrete portlet application, you will see configuration parameters to locate the wsdl file and resource file for internationalization support.

Example 24-4 Configuration parameters in concrete portlets

```
<config-param>
   <param-name>c2a-action-descriptor</param-name>
   <param-value>/hrportlet/wsdl/HRPortletC2A.wsdl</param-value>
</config-param>
<config-param>
   <param-name>c2a-nls-file</param-name>
   <param-value>hrportlet.nls.HRPortletC2A</param-value>
</config-param>
```

  For IBM portlets, the c2a-nls-file configuration parameter is used to specify the resource file name. The value should be the file name, including the package but omitting the `.properties` suffix.
- A WSDL file named HRPortletC2A.wsdl has been created.

Modifying the generated code

Import the source files from HRPortlet project as you did in “Importing the HRPortlet portlet” on page 755.

After importing the files you need to modify the wsdl with actions that can process the data transferred using C2A.

- Open the HRPortletC2A.wsdl file under EmployeeDetailsPortlet\W**ebContent\hrportlet\wsdl** folder. In the editor, switch to the Source tab and enter the following code shown in Example 24-5 on page 764.

Note: You can also download the sample code available as additional materials. See Appendix A, “Additional material” on page 1003.

Example 24-5 HRPortletC2A.wsdl file

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="HRPortlet_Service"
   targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:portlet="http://www.ibm.com/wps/c2a"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ibm.com/wps/c2a/examples/hrdetails"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <types>
      <xsd:schema
targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails">
         <xsd:simpleType name="DEPT_NO">
            <xsd:restriction base="xsd:string" />
         </xsd:simpleType>
      </xsd:schema>
   </types>

   <message name="HRPortlet_Request">
      <part name="HRPortlet_RequestPart" type="tns:DEPT_NO" />
   </message>

   <portType name="HRPortlet_Service">
      <operation name="hrportlet.HRPortletFormAction">
      <input message="tns:HRPortlet_Request" />
      </operation>
   </portType>

   <binding name="HRPortlet_Binding" type="tns:HRPortlet_Service">
      <portlet:binding />
      <operation name="hrportlet.HRPortletFormAction">
         <portlet:action name="hrportlet.HRPortletDetailsAction" type="simple"
caption="show.all.employees" description="Get all employees for specified
department id" />
         <input>
            <portlet:param name="DEPT_NOParam"
partname="HRPortlet_RequestPart" caption="department.ID"
description="department.id.for.retrieving.details" />
         </input>
      </operation>
   </binding>
</definitions>
```

Archived

The namespace and type associated with the input parameter matches the source declaration usign the encodeProperty tag in the JSP file of the source portlet.If the source and target portlets were placed on the same page, the C2A broker would detect a match.

- Open HRPortletC2A.properties under the EmployeeDetailsPortlet\Java Resources\JavaSource\hrportlet\nls folder. Add the following key-value pairs to localize text of the HRPortlet C2A portlets captions.

Example 24-6 Including new captions in resource bundles

```
# localized text of the HRPortlet c2a portlets captions
Get.all.employees.for.specified.department.id=Get all employees for specified
department id
department.ID=Department ID (HRPortlet)
department.id.for.retrieving.details=Department ID for retrieving details
show.all.employees=Show all employees from this department.
```

- The last step is to update the actionPerformed() method in the portlet class. In this scenario, you will use a special action string.
  a. Open the HRPortlet.java file from the EmployeeDetailsPortlet project.
  b. Insert the lines shown in Example 24-7 at the end of the actionPerformed() method.

Example 24-7 The hrportlet.HRPortletDetailsAction updates the SQL string

```
if (actionString.equals("hrportlet.HRPortletDetailsAction")) {
         HRPortletSessionBean bean = this.getSessionBean(request);
         bean.setSqlString(
            "select * from employee where workdept='"
              + (String) request.getParameter("DEPT_NOParam")
              + "'");
      }
```

9. Save all your files; you can use **Ctrl-S**.

24.3.4 Running the cooperative portlets

Execute the following steps to run the cooperative portlets scenario:

1. To run the projects in Rational Application Developer, it is necessary to add the portlet project to the test environment:
   a. On Servers panel, right-click **WebSphere Portal V5.1 Test Environment**.
   b. Select **Add and remove projects**.

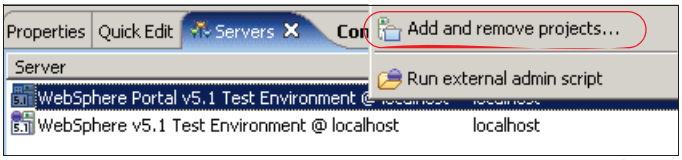


Figure 24-18 Adding a project to the test environment

c. In the next window, move `HRPortletEAR` and `EmployeeDetailsPortletEAR` to the right to configure them on the server.

d. Click **Finish**.

2. If the Cloudscape sample database has not been populated, run the batch file to populate the test database to be used in this scenario. Click **C:\HRproject\database\CreateCloudTable.bat** to do this.

   Note: You can also download the sample code available as additional materials. See Appendix A, “Additional material” on page 1003.

3. Next, click the **Project Explorer** tab to see your project again.

4. Right-click **HRPortlet**. Then click **Run** → **Run on Server**. Select **WebSphere Portal V5.1 Test Environment** server and click **Finish**. This will load your project into the test environment so that you can view it in the Rational Application Developer Web browser. It may take a minute or two for this process to complete.

5. You will now see your newly created portlets project running in the Web browser. You need to add EmployeeDetailsPortlet to the same page of HRPortlet to test the cooperation.

6. Click **Edit Page**.

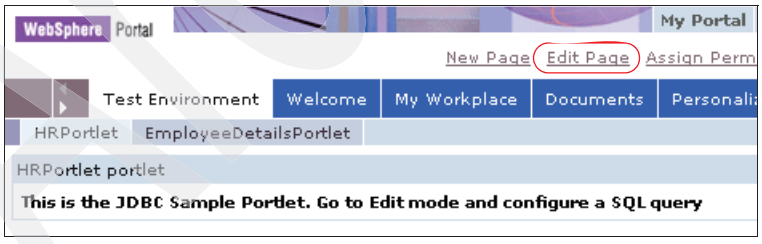


Figure 24-19 Edit Page to add portlets

7. In the Edit Layout window, you can see that the page contains HRPortlet. Click **Add portlets** button and search EmployeeDetailsPortlet from the list.

8. When you find it, check the EmployeeDetailsPortlet and click **OK**.

9. Now you will see the two portlets in the Edit Layout window. Click **Done**.

10. Switch to Edit mode in HRPortlet (c2a source portlet).
11. Enter the following information and click **Submit**.
    - Database: `jdbc:db2j:C:\HRproject\database\WSSAMPLE`
    - User: `db2admin`
    - Password: `db2admin`
    - SQL: `select * from jobs`



Figure 24-20 HRPortlet portlet in Edit mode

12. The HRPortlet now displays the jobs table, including a cooperative portlet menu in the DEPT_NO column. Before you can use this menu, you have to configure the data source in EmployeeDetailsPortlet.

Archived

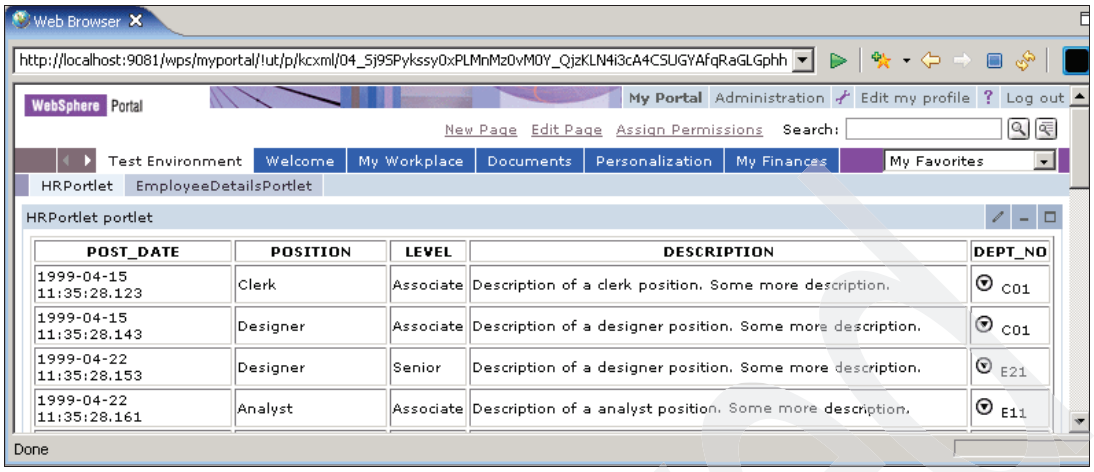


Figure 24-21 HRPortlet displays a cooperative menu in DEPT_NO column

13. Switch to Edit mode in EmployeeDetailsPortlet (c2a target portlet).
14. Enter the following information and click **Submit**. It is not necessary to enter an SQL command here, because it is built during the processing of the cooperative menu.
    - Database: `jdbc:db2j:C:\HRproject\database\WSSAMPLE`
    - User: `db2admin`
    - Password: `db2admin`

    Note: There is no need to enter an SQL statement (optional).
15. In the source cooperative portlet View mode, click a DEPT_NO column, for example **C01** or **A00.**
16. Click **Show all employees from this department** (see Figure 24-22 on page 769).

    Note: The EmployeeDetailsPortlet (target portlet) should now display all employees in the selected department.

Archive

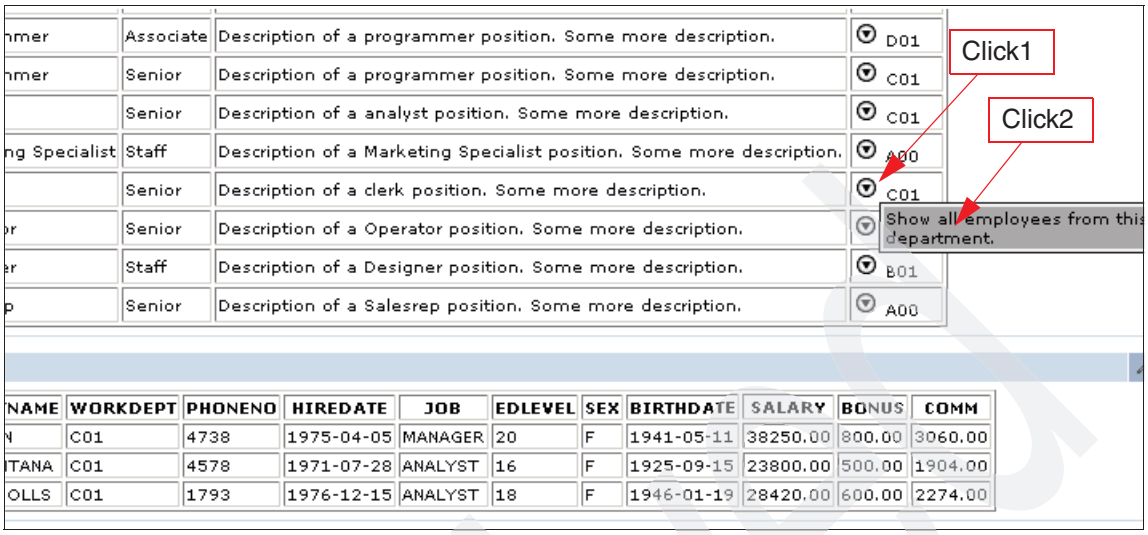


Figure 24-22 EmployeeDetailsPortlet displays all employees from same department

17. Now select another department and hold **Ctrl** key while clicking a Click-to-Action icon. You will be prompted with the dialog illustrated in Figure 24-23



Figure 24-23 Dialog to establish a wire

18. Select **Yes**. A persistent connection between the two portlets is created. This connection is called wire.
19. The next time you click the icon, no selection menu is shown, the wire action is automatically fired.
20. Select **Edit Page** and click the **Wires** tab. Figure 24-24 on page 770 shows the new wire.

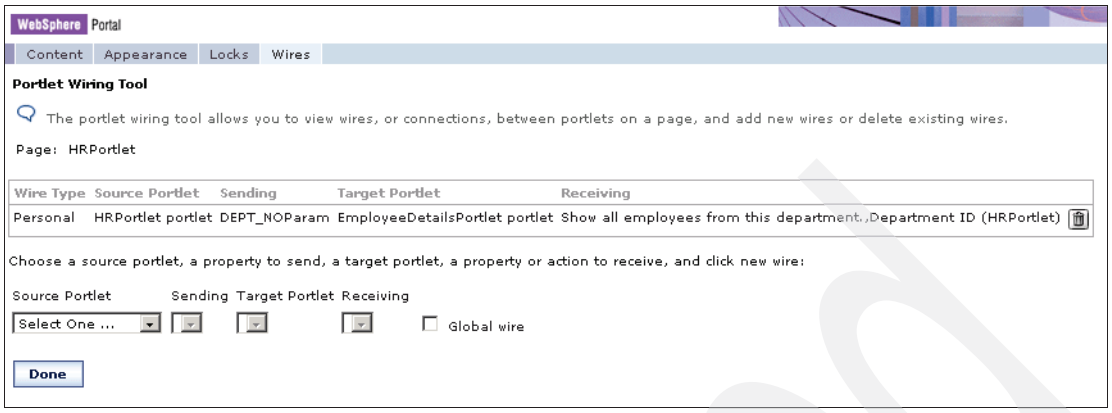


Figure 24-24 Portlet Wiring Tool

Wire type is personal; that means its effect would be manifested only for the creator. If, in the menu shown in Figure 24-23 on page 769, you had selected **Yes, all users**, a global wire would have been created. Global wire means that wire effects are visible to all users who can view the page and portlets.

Archived

25

IBM API programmatic cooperative portlets

This chapter discusses advanced cooperative portlet topics.

After reading this chapter, you will be able to:

- Develop source cooperative portlets using a programmatic approach to publish properties.
- Develop target cooperative portlets using a programmatic approach to publish properties.
- Develop source cooperative portlets broadcasting data to two or more portlets.

Note: The sample scenario included in this chapter requires that you have read Chapter 24, “IBM API declarative cooperative portlets” on page 741. You can also download the sample code available as additional materials. See Appendix A, “Additional material” on page 1003.

© Copyright IBM Corp. 2005. All rights reserved.

25.1 Publishing properties programmatically

As mentioned in Chapter 24, “IBM API declarative cooperative portlets” on page 741, each action in a cooperative portlet is associated with a single input parameter and zero or more output parameters that provide information to the action about the objects in which the property value should be bound, such as the request object or the session object.

Note: Cooperative portlets using the programmatic approach require that you create a wire; for details about creating wires, see 25.5.4, “Wire portlets” on page 793 and 25.4, “Wiring tool” on page 778. This is because property values are transferred to the target portlets only if wires have been created.

Each parameter is associated with exactly one property. Parameters associated with input properties are called *input parameters*, while those associated with output properties are called *output parameters*. Instead of actions, target portlets can receive property changes directly through the *PropertyListener* interface.

The actual transfer of the property can be initiated by one of the following methods:

1. A user launches a Click-to-Action event from an icon on the source portlet. The icon presents a pop-up menu containing the list of targets for the action. After the user selects a specific target, the property broker delivers the data to the target in the form of the corresponding portlet action.
2. A user presses the **Ctrl** key while clicking an action and chooses to have the selection saved persistently as a connection between two portlets, called a wire.
3. The source portlet can perform a programmatic publish of properties to the broker, when it determines that property values have changed. Such property values are transferred to the target(s) only if wires have been created.

The property broker provides APIs to give developers more control over how portlets handle the input and output properties. In general terms, the programmatic approach might be a better option over the declarative approach when the portlet needs to do the following:

- Activate or deactivate actions for a session.
- Change the portlet state but not requiring the portlet to react immediately.
- Publish output properties using the *changedProperties()* method.
- Register actions programmatically instead of declaring them in a WSDL file. This may be necessary when the action or property is not known at

development time, such as when a portlet is generated by a builder application.

- Generate markup content directly in the portlet rather than using JSPs.

The following packages are provided for portlets to publish properties to the property broker programmatically:

- com.ibm.wps.pb.property
- com.ibm.wps.pb.portlet
- com.ibm.wps.pb.service

25.2 Portlet event handling

The portlet programming model involves an event phase and a render phase in each request-response cycle.

- The event phase is when the property broker delivers notifications to cooperative portlets and when the cooperative portlets can notify the property broker of property value changes. During the event phase, an action may be delivered on one portlet. If the property broker is used, this may result in other actions being triggered on other portlets.
- The event phase is followed by the render phase, in which each portlet is asked to return markup, which is then aggregated in a single page. The markup may embed actions which can be invoked by the user. The page is then returned to the client (such as a browser).

At any point during the event phase, a portlet may explicitly publish the value of an output property to the property broker by invoking the *changedProperties()* method. This is an alternative to the declaration of output parameters for actions and binding the output parameter values to the request or session when the action is invoked. This may happen in the following cases:

- In the callback method associated with the start of the event phase (beginEventPhase method)
- In the invocation of the setProperties() method in a target portlet
- In the portlet action method invocation

The publishing calls are dealt with by the property broker in the same manner as output parameters of actions: wires associated with output properties are examined and the property values propagated using the information in the target end of the wire.

Archived

Note: The process may continue recursively; however, the property broker detects loops and breaks them. Also, during the event phase of the subsequent request, the action is invoked on the corresponding target portlet or portlets.

Figure 25-1 illustrates a simplified version of cooperative portlets implemented using the programmatic approach.

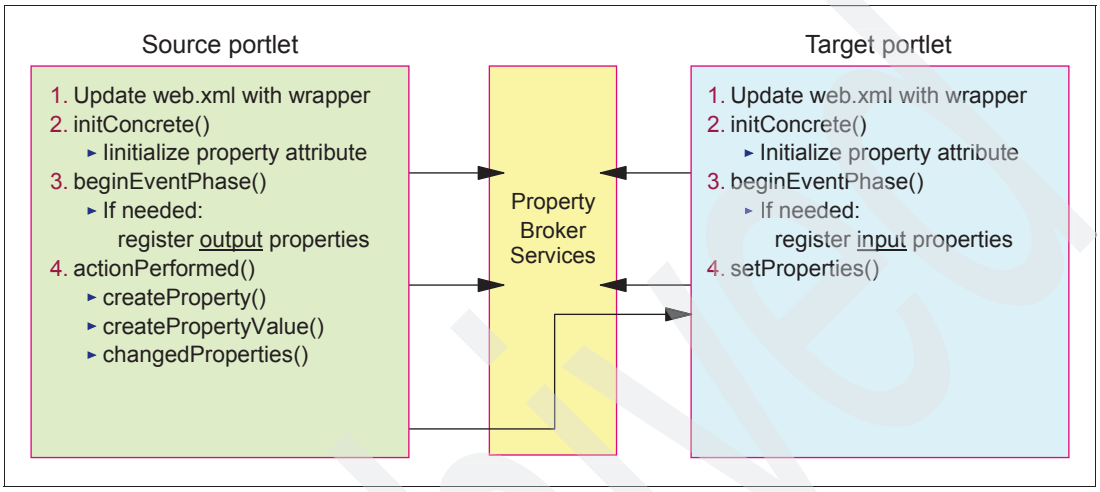


Figure 25-1 Sample summary of a programmatic approach

In the general case using a programmatic approach, the source portlet needs to implement the following:

1. Specify the C2A wrapper in the web.xml descriptor as explained in Chapter 24, “IBM API declarative cooperative portlets” on page 741.
2. The property broker attribute needs to be initialized; this can be done in the *initConcrete()* method.
3. The portlet will need to register its output properties by using the *registerProperties()* method. This can be done by implementing a *beginEventPhase()* method so the portlet is notified when the event phase starts.

   Note: The EventPhaseListener interface requires that you also provide the endEventPhase() method. If needed, some cleanup can be done in this method.
4. The source portlet publishes its output properties, for example when processing an action in the *actionPerformed()* method.

In a similar way, the target portlet needs to be updated as follows:

1. Specify the C2A wrapper in the web.xml descriptor as explained in Chapter 24, “IBM API declarative cooperative portlets” on page 741.
2. The property broker attribute needs to be initialized; this can be done in the *initConcrete()* method.
3. The portlet will need to register its input properties by using the *registerProperties()* method. This can be done by implementing a *beginEventPhase()* method so the portlet is notified when the event phase starts.
4. The target portlet implements the *setProperties()* method to be notified of property changes reported by other source portlets.

   Note: The target portlet must implement the PropertyListener interface.

25.2.1 PropertyListener interface

This interface is implemented for cooperative portlets using the programmatic approach. This interface may optionally be implemented by portlets. It is an alternate mechanism by which interested portlets may be notified of changed properties.

Other options are to be notified through portlet actions (the actionPerformed method of the ActionListener interface), or Struts actions. The PropertyListener interface may be implemented by portlets that only wish to update their current state based on property changes, rather than execute an action immediately.

Note: The PropertyListener interface requires that you implement the setProperties method to be notified of property changes.

setProperties

Invoked by the Property Broker to deliver new property values which were changed in the current event cycle of the current request. The Property Broker may be notified of such changes when a portlet invokes the changedProperties in the PropertyBrokerService interface (explicit notification), or when a portlet action which has declared output parameters is invoked (implicit notification).

This method is only invoked during the event phase. Since multiple explicit or implicit property change notifications may be made during an event cycle, one or more setProperties calls may be invoked on a single portlet instance during a single event cycle. The runtime may batch property values from multiple changedProperties calls in a single setProperties call. All properties are guaranteed to be delivered before the first endEventPhase call is delivered, which marks the start of the render phase.

Archived

Source cooperative portlets report property changes may be made by using the changedProperties method.

changedProperties method

This method publishes changes in properties and may be used during the portlet's event phase only. This includes the beginEventPhase method of the EventPhaseListener interface, the actionPerformed method of the ActionListener interface, and the setProperties method of the PropertyListener interface.

All properties must have been registered earlier, implicitly or explicitly. A simpler alternative to explicitly invoking this method is often applicable. For example, declare output parameters for registered actions (either programmatically or via an WSDL declaration). In this case, the action may bind the values of the output parameters on invocation, and at runtime the values will be transferred as if the changedProperties method had been explicitly invoked.

25.2.2 EventPhaseListener interface

This interface allows developers to get control of the portlet before the portlet receives a notification of a changed property.

This interface provides the following methods.

beginEventPhase method

At any point during the event phase, a portlet may explicitly publish the value of an output property to the property broker by invoking the changedProperties() method. This is an alternative to the declaration of output parameters for actions and binding the output parameter values to the request or session when the action is invoked. This may happen in the callback method associated with the start of the event phase (beginEventPhase()), in the invocation of the setProperties() method, or in the portlet action method invocation.

Such publish calls are dealt with by the property broker in the same manner as output parameters of actions: wires associated with output properties are examined and the property values propagated using the information in the target end of the wire. To register properties, you will use the beginEventPhase method of EventPhaseListener, because only during the event phase is it possible to register and unregister properties.

endEventPhase method

The property broker guarantees the completion of all property value notifications to target portlets by the end of the event phase, whether through portlet actions or through the special setProperties() method. The end of the event phase is indicated by the invocation of the endEventPhase() method.

During the render phase of each request cycle, source portlets can write visual controls representing source data to their output stream. The end user interacts with the visual control in the response to trigger one or more actions on other portlets on the page. During the event phase of the subsequent request, the action is invoked on the corresponding target portlet or portlets.

25.3 Broadcasting source data

Using the broadcast feature of the cooperative broker, users can send the same data to all portlets on the page with matching actions. The target cooperative portlet of a broadcast can use the declarative or programmatic approach to publish properties.

To include the broadcast menu item to the HRPortlet, proceed as follows:

1. Open the JSP HRPortlet/Web Content/hrportlet/jsp/html/HRPortletView.jsp.
2. In the encodeProperty tag, include the broadcast attribute so it looks as shown in Example 25-1.

Example 25-1 The broadcast attribute enables the broadcast feature.

```
<TD><P>
   <% if (results.getColumnName(col).equalsIgnoreCase("DEPT_NO")) { %>
      <C2A:encodeProperty
         name="DEPT_NOParam"
         namespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
         type="DEPT_NO"
         value="<%=results.getCacheValueAt(row, col).toString()%>"
         broadcast="true"/>
   <% } %>
   <%=results.getCacheValueAt(row, col)%>
</P></TD>
```

3. Test the application as described in the last chapter. You need to include HRPortlet, EmployeeDetailsPortlet and DepartmentDetailsPortlet in the same page. Do not forget to enter the database attribute in the Edit mode of both the Employee and Department Details Portlets.
4. From the cooperative portlet menu, choose **Invoke all actions**. Now both details portlets display the details of the selected department number.

Archived

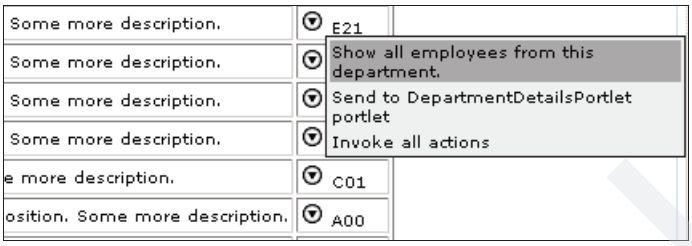


Figure 25-2 Choose Invoke all actions to broadcast the data to all portlets

25.4 Wiring tool

The portlet wiring tool allows you to view the properties that portlets on the page can send or receive. If a match is available between two portlets, you can create a wire between the two portlets. Existing wires may also be deleted using the tool. This is an alternative to the wire creation or deletion while interacting with the portlets using the **Ctrl** key.

The wiring tool allows wires to be created in situations which are not handled by the interactive approach. For example, the tool does not require the existence of Click-to-Action menus to initiate wire creation, and can be used to create multiple wires from a single source property (using the interactive approach, a single source can be wired to a single target or all targets, not an arbitrary subset). Wire creation or deletion is subject to the access control checks.

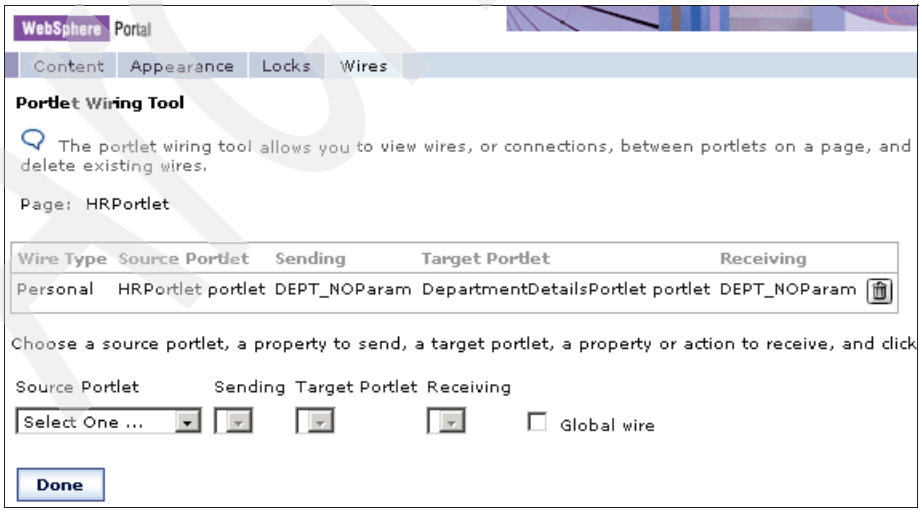


Figure 25-3 Creating wires programmatically using Portlet Wiring Tool

WebSphere Portal V5.1 includes the Portlet Wiring Tool as part of the product and deployed to the Page Customizer.

25.5 Sample scenario

In this section, a sample scenario is provided to illustrate how to develop cooperative portlets using the programmatic approach.

25.5.1 Declarative source cooperative portlet

In this scenario, you will implement a combined scenario where the source cooperative portlet uses the declarative approach to interact with a target cooperative portlet using the programmatic approach. The sample scenario is shown in Figure 25-4.

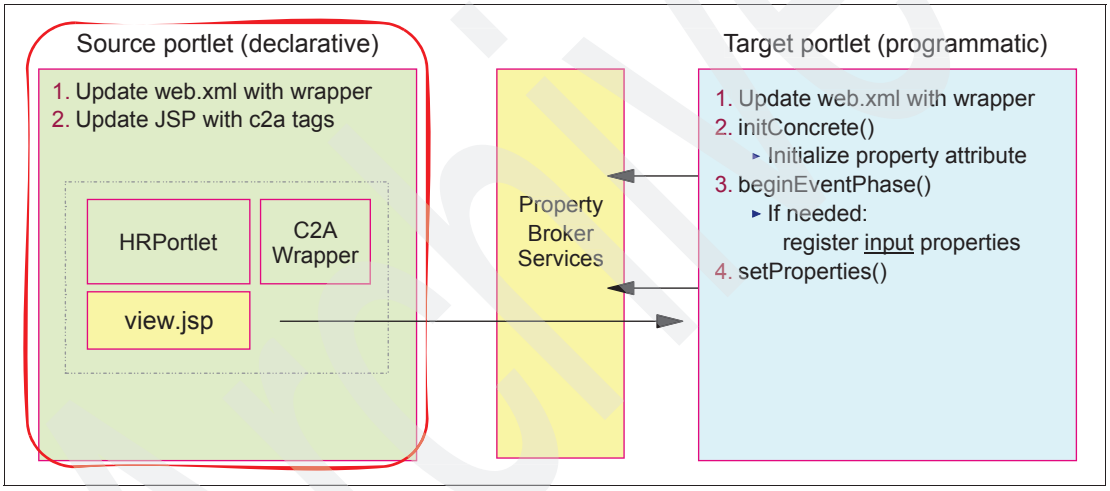


Figure 25-4 A sample scenario using a combined approach

Creating a source portlet by importing the HRPortlet portlet

The HRPortlet portlet from Chapter 24, “IBM API declarative cooperative portlets” on page 741 will be used as a base for this scenario. This portlet will be enabled to act as a source cooperative portlet in this scenario. You will need to import this portlet if it is not in your workspace.

Follow these steps to import this portlet if it is not in your workspace:

1. Import the WAR file by selecting **File** → **Import**.
2. Select the WAR file and click **Next**.

3. In the *WAR Import* window, enter the following information:
   a. WAR file: browse to `C:\LabFiles\HRPortlet.war`.

      Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix A, “Additional material” on page 1003.

   b. Web project: `HRPortlet`.
   c. Target server: select `WebSphere Portal V5.1`.
   d. EAR project: `HRPortletEAR`.
   e. Click **Finish** to import the WAR file.

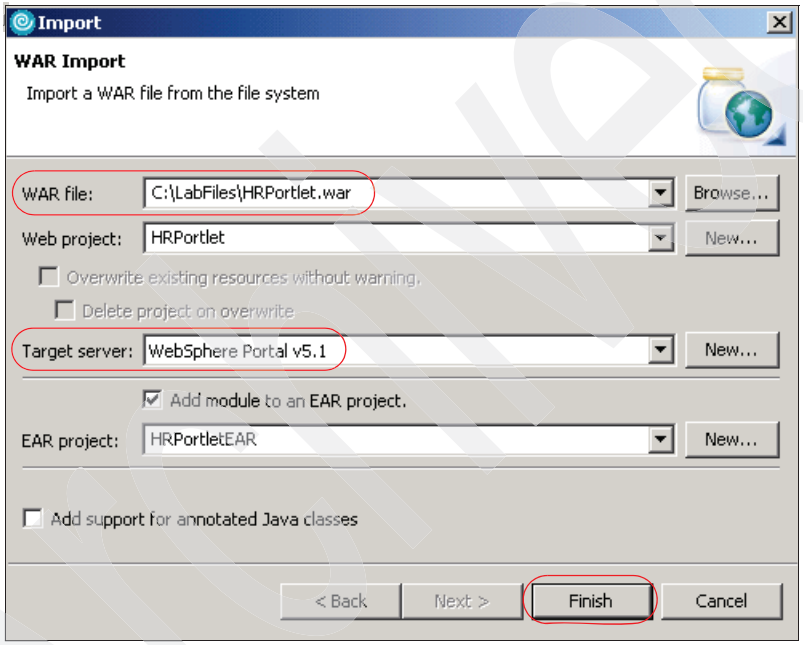


Figure 25-5 Import HRPortlet.war file

4. Make sure the web.xml descriptor has been updated with the c2a wrapper.
5. Make sure the PortletView.jsp has been updated with the c2a tags and c2a tag library. See Example 25-2.

Example 25-2 C2a library and tags

```
<%@ page contentType="text/html" import="java.util.*, hrportlet.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ taglib uri="/WEB-INF/tld/c2a.tld" prefix="C2A" %>
```

```
.................................................
<TD><P>
   <% if (results.getColumnName(col).equalsIgnoreCase("DEPT_NO")) { %>
      <C2A:encodeProperty
      name="DEPT_NOParam"
      namespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
      type="DEPT_NO"
      value="<%=results.getCacheValueAt(row, col).toString()%>" />
   <% } %>
   <%=results.getCacheValueAt(row, col)%>
</P></TD>
```

6. Save your files (or press **Ctrl-S** to save the files).

25.5.2 Enabling the portlet for target C2A programmatic

In this section, you will import a second copy of the HRPortlet and update it to support Click-to Action as a target cooperative portlet using the programmatic approach. This target portlet will execute a fixed SQL statement with a variable *where* clause.

The code for the target portlet class must meet the following requirements:

- The action must be implemented either as a portlet action or a Struts action. For portlet actions, you should use the simple action Strings rather than the deprecated PortletAction class.
- Portlet actions must accept a single parameter. The parameter may appear as a request parameter, a request attribute, a session attribute, or an action attribute (deprecated), as specified in the action declaration or registration.

The HRPortlet is already prepared for this situation, so only a few changes are needed in the portlet class code. Figure 25-6 on page 782 illustrates the target cooperative portlet for this sample scenario.

Archive

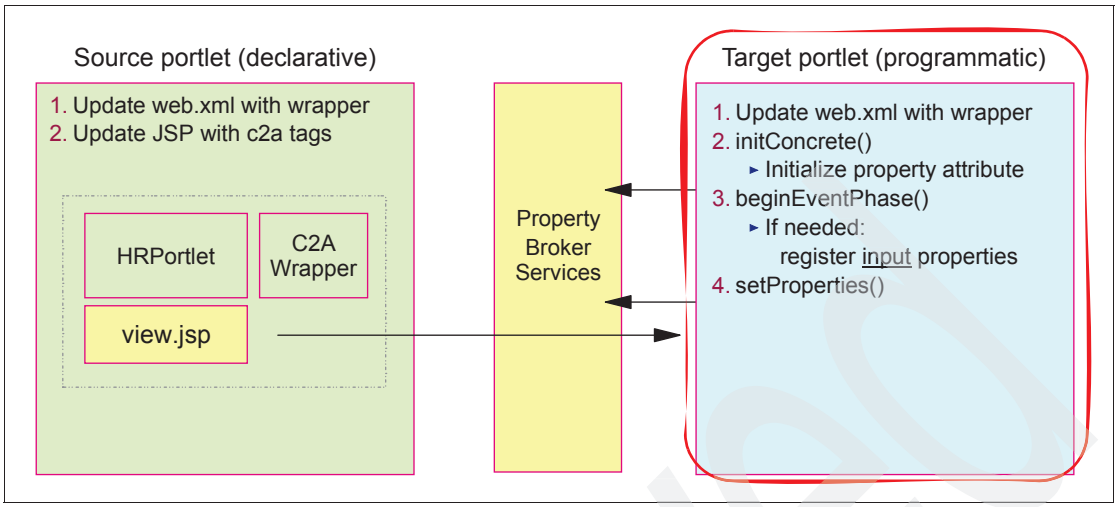


Figure 25-6 Cooperative portlets - programmatic sample scenario (target portlet)

Creating a target portlet by importing HRPortlet portlet

To import a second version of the HRPortlet, follows these steps:

1. From the main menu, select **File** → **Import** to import the original HRPortlet.
2. Choose **WAR file**, click **Next** and configure as follows:
   a. Browse to the location of the HRPortlet.war file in c:\LabFiles\HRPortlet.war.

      Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix A, “Additional material” on page 1003.

   b. As the Web Project, enter `DepartmentDetailsPortlet`.
   c. Select **WebSphere Portal V5.1** as target server.
   d. Enter `DepartmentDetailsPortletEAR` as the EAR project.
   e. Click **Finish**.

Archive

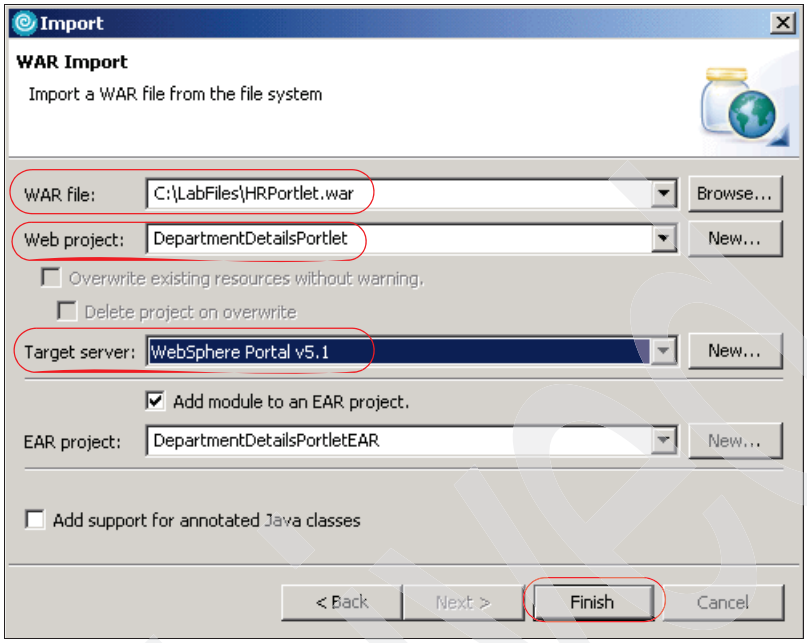


Figure 25-7 Import a second version of HRPortlet (target c2a portlet)

3. Since the HRPortlet and DepartmentDetailsPortlet portlet applications use the same UID, a warning message will appear in the Problems pane.

Archive

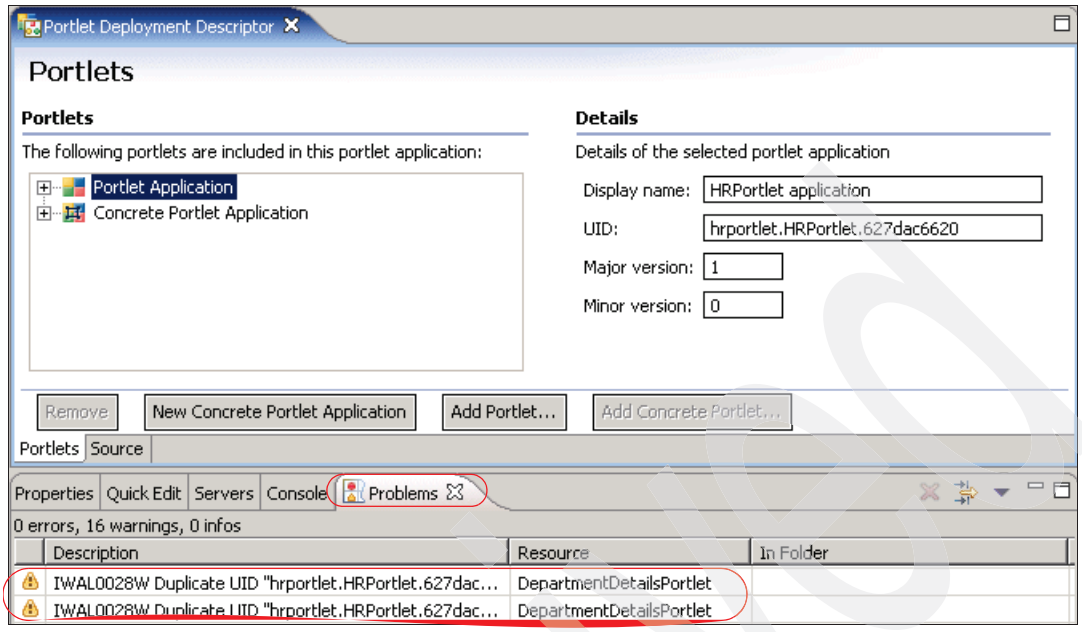


Figure 25-8 Duplicate UID messages

4. To fix this problem, expand **DepartmentDetailsPortlet/WebContent/WEB-INF** in the Project Explorer view. Double-click **portlet.xml**.

Archive

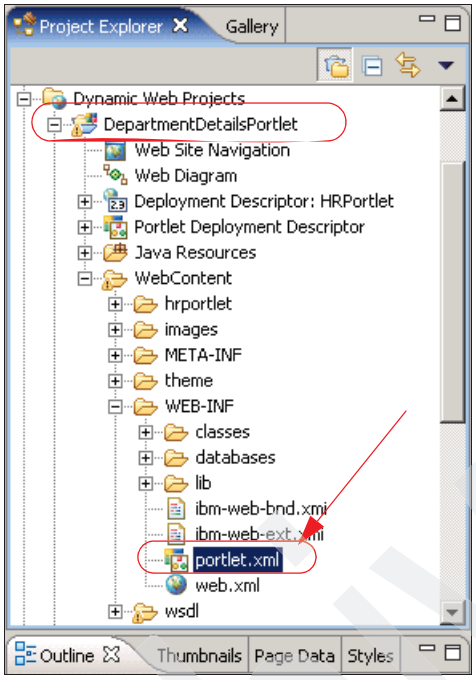


Figure 25-9 Selecting portlet.xml

5. In the portlet deployment descriptor editor, select **Portlet Application** and change the last digit of the UID for this portlet application. For example, in this sample scenario, the last digit was 0 and it was changed to 1.

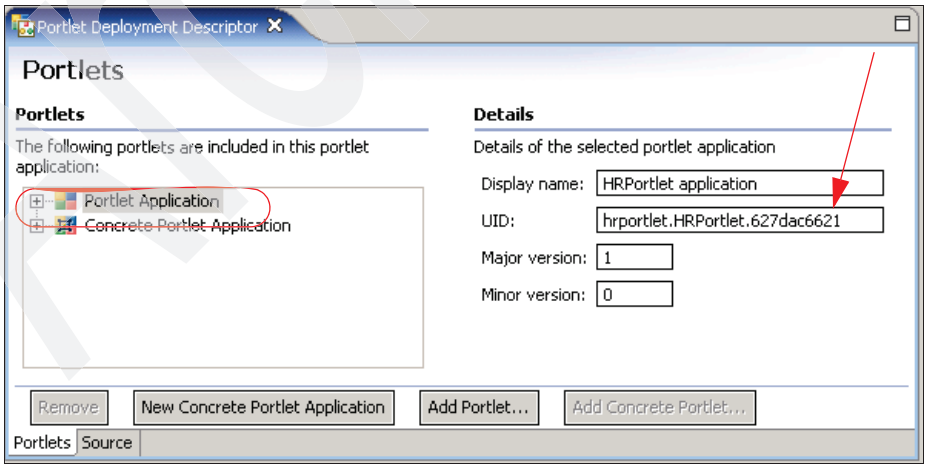


Figure 25-10 Changing a digit in portlet application UID

Archived

6. In a similar way, select **Concrete Portlet Application** and change the last digit before the last dot of the UID to give it the same value as in the previous step.

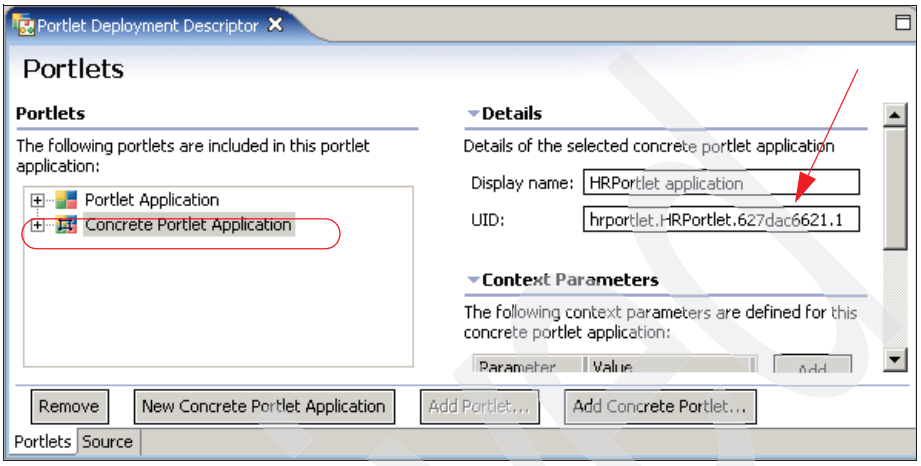


Figure 25-11 Changing digit in concrete portlet application UID

7. Change the portlet name and portlet title to DepartmentDetailsPortlet portlet to identify the portlet when you run the application.
8. Save your changes.
9. Make sure the web.xml descriptor has been updated with the c2a wrapper.
10. If `HRPortlet/WebContent/wsdl/HRPortlet.wsdl` file exists delete it. When you delete this file an error appears in portlet.xml file. Open portlet deployment descriptor and delete the following config param to fix the error:

```
<config-param>
   <param-name>c2a-action-descriptor</param-name>
   <param-value>/wsdl/HRPortletportlet.wsdl</param-value>
</config-param>
```

11. Save your changes.

Updating the DepartmentDetailsPortlet

In this section, you will update the portlet to act as a target C2A portlet using the programmatic approach. The DepartmentDetailsPortlet is a copy of the HRPortlet and it will register the DEPT_NO property by using the property broker API instead of a WSDL file used in the declarative approach.

Follow these steps to update the portlet class to publish the property and create call back methods to recognize the property changes:

1. Open the file DepartmentDetailsPortlet/Java Resources/JavaSource/hrportlet/HRPortlet.java.
2. Update the class definition so it implements the PropertyListener and EventPhaseListener interfaces.

   ```
   public class HRPortlet extends PortletAdapter implements PropertyListener,
   EventPhaseListener, ActionListener
   ```

   Note: The setProperties method of the PropertyListener interface receives updates of property values. To register properties, you will use the *beginEventPhase* method of the EventPhaseListener interface, since only during the event phase is it possible to register and unregister properties.
3. Insert the following two new class attributes so the code looks like this:

   ```
   public class HRPortlet extends PortletAdapter implements PropertyListener,
   EventPhaseListener, ActionListener {

   PropertyBrokerService pbService;
   PortletConfig portletConfig;
   ```

   Note: `pbService` is an interface to the property broker and `portletConfig` stores the portlet config.
4. Update the *init* method so it looks as shown in Example 25-3.

Example 25-3 Update init method to obtain portlet configuration

```
public void init(PortletConfig portletConfig) throws UnavailableException {
      super.init(portletConfig);
      this.portletConfig=portletConfig;
   }
```

5. Insert the *initConcrete* method shown in Example 25-4 to initialize the property broker attribute.

Example 25-4 Initialize property broker

```
public void initConcrete(PortletSettings settings)
   throws UnavailableException {
   try {
      pbService =
         (PropertyBrokerService) getPortletConfig()
            .getContext()
            .getService(
            PropertyBrokerService.class);
   } catch (PortletServiceUnavailableException e) {
      throw new UnavailableException("Could not locate
PropertyBrokerService.");
   } catch (PortletServiceNotFoundException e) {
```

Archived

```
            throw new UnavailableException("Could not locate
PropertyBrokerService.");
        }
    }
```

6. For simplicity, create the *registerPropertiesIfNecessary* method to register the property we are interested in.

   Note: This method performs the same as the WSDL file when using the declarative approach. Notice also that you need to specify a direction of *Property.IN*. In addition, actions are not registered (which is possible using the registerActions method) in this scenario. In other words, this portlet will be invoked by the property broker using the setProperties method instead of actionPerformed.

Example 25-5 registerPropertiesIfNecessary method

```
    private void registerPropertiesIfNecessary(PortletRequest request)
        throws PropertyBrokerServiceException {
        PortletSettings settings = request.getPortletSettings();
        Property[] properties = pbService.getAllProperties(request, settings);
        if (properties == null || properties.length == 0) {
            PortletContext context = getPortletConfig().getContext();
            //not registered, register now
            properties = new Property[1];
            properties[0] = PropertyFactory.createProperty(settings);
            properties[0].setName("DEPT_NOParam");
            properties[0].setDirection(Property.IN);
            properties[0].setType("DEPT_NO");

properties[0].setNamespace("http://www.ibm.com/wps/c2a/examples/hrdetails");
            properties[0].setTitleKey("HRDetails.Department");
            properties[0].setDescriptionKey("Display department details");
            pbService.registerProperties(request, settings, properties);
        }
    }
```

7. Add the *beginEventPhase* and *endEventPhase* methods, which are callback methods called during the event phase.

Example 25-6 The beginEventPhase methods registers the property if necessary

```
    public void beginEventPhase(PortletRequest request) {
        try {
            registerPropertiesIfNecessary(request);
        }
        catch (Throwable e) {
            e.printStackTrace();
        }
```

```
    }
    public void endEventPhase(PortletRequest request) {
    }
```

8. Add the setProperties method. Using this method, the class is notified about property changes.

Example 25-7 setProperties updates the sql statement

```
public void setProperties(
    PortletRequest request,
    PropertyValue[] properties) {
    PortletSession session = request.getPortletSession();
    HRPortletSessionBean sessionBean = getSessionBean(request);
    for (int i = 0; i < properties.length; i++) {
        System.out.println(properties[i].toString());
        if (properties[i].getProperty().getName().equals("DEPT_NOParam")) {
            String value = (String)properties[i].getValue();
            if (value.equals("*"))
                sessionBean.setSqlString("select * from department");
            else
                sessionBean.setSqlString(
                    "select * from department where deptno='"
                        + value
                        + "'");
        }
    }
}
```

9. Right-click anywhere in the Java editor and select **Source** → **Organize Imports** to include the missing import statements. In the Organize Imports dialog, choose to import the following classes:
   a. com.ibm.wps.pb.service.PropertyBrokerService
   b. org.apache.jetspeed.portlet.service.PortletServiceUnavailableException
   c. com.ibm.wps.pb.service.PropertyBrokerServiceException
   d. com.ibm.wps.pb.property.Property
   e. org.apache.jetspeed.portlet.PortletContext
   f. com.ibm.wps.pb.property.PropertyFactory
   g. com.ibm.wps.pb.property.PropertyValue
10. Save and close the HRPortlet.java file.

25.5.3 Running the cooperative portlets

Execute the following steps to run the cooperative portlets scenario:

1. To run the project you can use a previously created test server or you can create a new server. Follow these steps:

a. In Servers panel, right-click **WebSphere Portal V5.1 Test Environment**.

b. Select **Add and remove projects** to add your project to the Test Environment (see Figure 25-12); add HRPortletEAR and DepartmentDetailsPortletEAR to the right panel, Configured projects, by selecting them in Available projects panel and clicking the **Add** button.

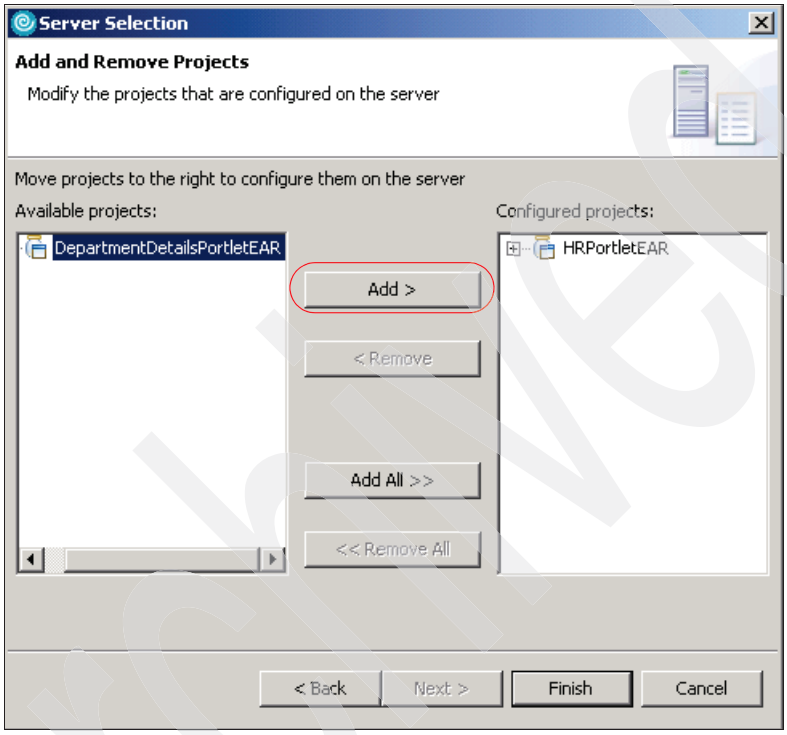


Figure 25-12 Adding a project to the Test Environment

2. If the Cloudscape sample database has not been populated, run the batch file to populate the test database to be used in this scenario. Click **C:\HRproject\database\CreateCloudTable.bat** to do this.

   Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix A, "Additional material" on page 1003.

3. In Project Explorer panel, right-click **HRPortlet**. Then click **Run** → **Run on Server**. Select **WebSphere Portal V5.1 Test Environment** from the server list and click **Finish**. This will load your projects into the Test Environment so that you can view them in the Rational Application Developer internal Web browser. It may take a minute or two for this process to complete.

4. You will now see your newly created portlets project running in the Web browser in two different pages. You need to add DepartmentDetailsPortlet to the same page of HRPortlet to test the cooperation.

   a. Click **Edit Page**.

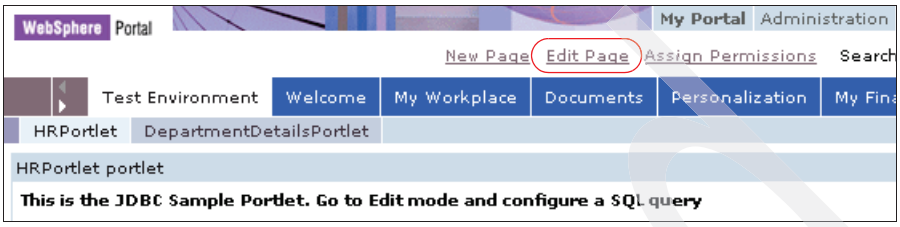


Figure 25-13 Edit Page to add portlets

   b. In the Edit Layout window you can see that the page contains HRPortlet. Click **Add portlets** button and search DepartmentDetailsPortlet from the list.

   c. When you find it check the DepartmentDetailsPortlet and click **OK**.

   d. Now you will see the two portlets in the Edit Layout window. Click **Done**.

> **Note:** You need to enable base portlets for portal administration and customization in server configuration options.

5. Switch to Edit mode in HRPortlet (c2a source portlet).
6. Enter the following information and click **Submit**.
   - Database: `jdbc:db2j:C:\HRProject\database\WSSAMPLE`
   - User: `db2admin`
   - Password: `db2admin`
   - SQL: `select * from jobs`

Archive

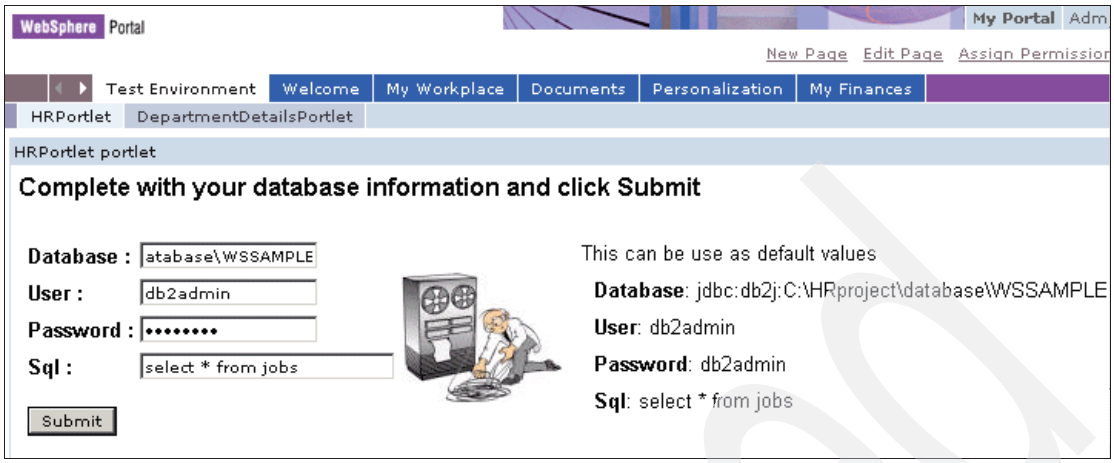


Figure 25-14 HRPortlet portlet in Edit mode

7. The HRPortlet now displays the jobs table, including a cooperative portlet menu in the DEPT_NO column. Before you can use this menu, you have to configure the data source in DepartmentDetailsPortlet.



Figure 25-15 HRPortlet displays a cooperative menu in the DEPT_NO column

8. Switch to Edit mode in DepartmentDetailsPortlet (c2a target portlet).
9. Enter the following information and click **Submit**. It is not necessary to enter an SQL command here, because it is built during the processing of the cooperative menu.

- Database: `jdbc:db2j:C:\HRProject\database\WSSAMPLE`
- User: `db2admin`
- Password: `db2admin`

Note: There is no need to enter an SQL statement (optional).

10. In the source portlet View mode, click the **c2a** icon in the DEPT_NO column, for example B01.
11. Click the **c2a** menu again to send the changed property to C2A.

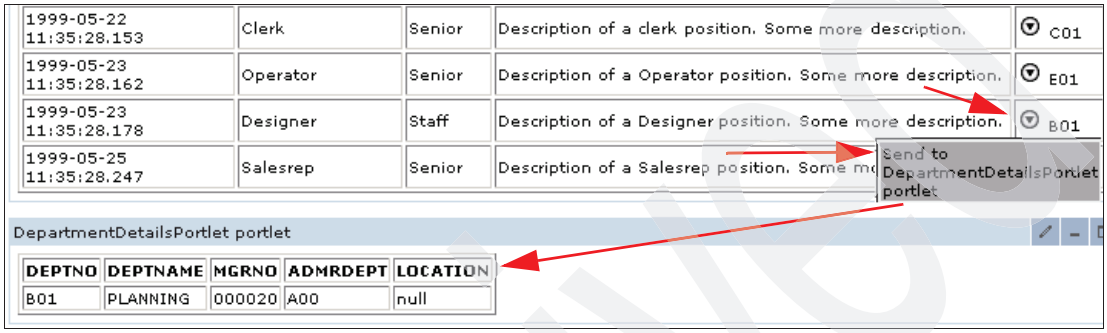


Figure 25-16 Cooperative portlets

25.5.4 Wire portlets

In this section, you will wire the source and target portlets for C2A. To create a wire it is necessary the wsdl file exists in the source portlet (HRPortlet portlet) declaring the properties that this portlet is going to share.

Follow these steps:

1. Make sure the HRPortlet/WebContent/wsdl/HRPortlet.wsdl file exists and portlet deployment descriptor contains the following config parameter:

```
<config-param>
    <param-name>c2a-action-descriptor</param-name>
    <param-value>/wsdl/HRPortlet.wsdl</param-value>
</config-param>
```

2. Select one of the C2A menu items but this time press the **Ctrl** key during the selection.
3. A new dialog opens asking whether to `Automate the action in future`. Select **Yes**.

Archive

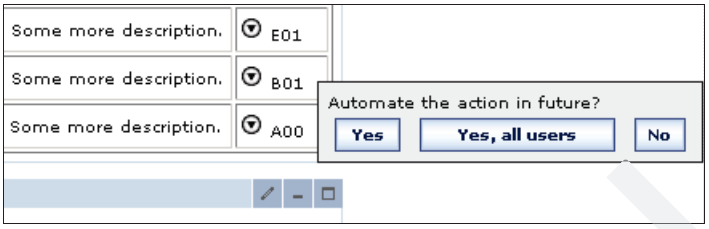


Figure 25-17 Pressing Ctrl key during menu selection to create a wire

4. Click **Yes** to activate the wire.
5. Try other departments again.
6. Press the **Ctrl** key again when using C2A to disable the wire.

Note: Wiring can also be accomplished by using the Portlet Wiring Tool.

25.5.5 Enabling HRPortlet for programmatic source C2A

In this section, you will be required to enhance the source C2A portlet application to implement the programmatic approach. Figure 25-18 illustrates the source cooperative portlet for this sample scenario.

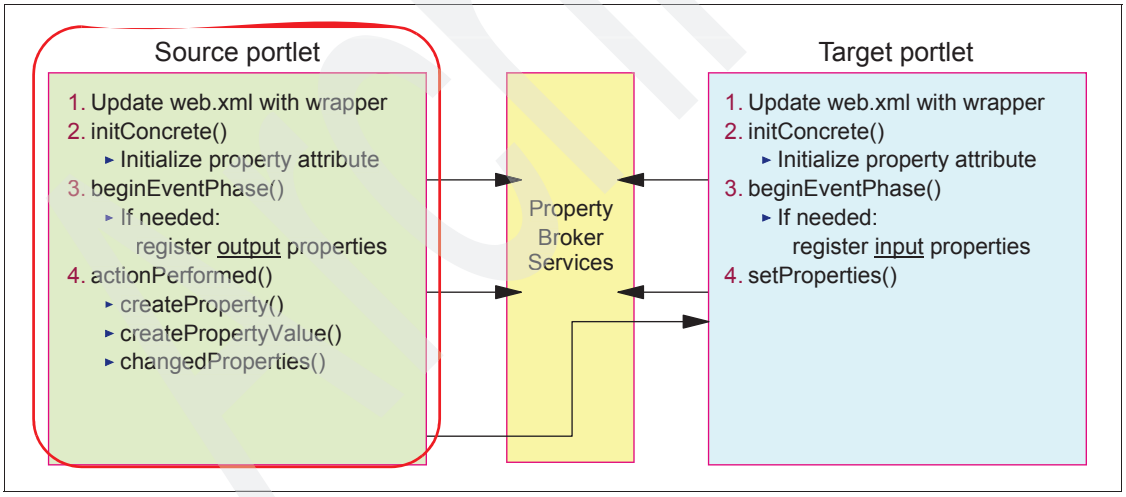


Figure 25-18 Cooperative portlets - programmatic sample scenario (source portlet)

You will update HRPortlet to change property data using the programmatic approach. You will also insert a button in the HRPortlet page to offer the display of department details in the DepartmentDetailsPortlet (C2A target portlet).

Notice that in this portlet, you will need to register the DEPT_NO property using the Property.OUT direction. In addition, the update of the property will be done in the actionPerformed method.

Note: The sample scenario included in this chapter requires that you download the sample code available as additional materials. See Appendix A, “Additional material” on page 1003.

Proceed as follows to update the HRPortlet:

1. Open the file HRPortlet/Java Resources/JavaSource/hrportlet/HRPortlet.java.
2. Update the class definition so it implements the *EventPhaseListener* interface:

   ```
   public class HRPortlet extends PortletAdapter implements
   EventPhaseListener, ActionListener {
   ```

   Note: The *beginEventPhase* method of the *EventPhaseListener* interface is used to register the output properties; this is because it is only possible to register and unregister properties during the event phase.
3. At the beginning of this class, insert two new class attributes so the code looks like this:

   ```
   public class HRPortlet extends PortletAdapter implements
   EventPhaseListener, ActionListener {
   PropertyBrokerService pbService;
   PortletConfig portletConfig;
   ```

   Note: `pbService` is an interface to the property broker and `portletConfig` stores the portlet configuration.
4. Change the *init* method so it looks as follows:

   ```
      public void init(PortletConfig portletConfig) throws
   UnavailableException {
          super.init(portletConfig);
          this.portletConfig=portletConfig;
      }
   ```
5. Add the following initConcrete method to initialize the property broker attribute.

Example 25-8 The initConcrete method initializes the property broker attribute

```
public void initConcrete(PortletSettings settings)
   throws UnavailableException {
   try {
      pbService =
         (PropertyBrokerService) getPortletConfig()
            .getContext()
            .getService(
            PropertyBrokerService.class);
```

```
      } catch (PortletServiceUnavailableException e) {
         throw new UnavailableException("Could not locate
PropertyBrokerService.");
      } catch (PortletServiceNotFoundException e) {
         throw new UnavailableException("Could not locate
PropertyBrokerService.");
      }
   }
```

6. For simplicity, you will now add a new method called registerPropertiesIfNecessary to register the property we are interested in.

Example 25-9 Register a new output property

```
   private void registerPropertiesIfNecessary(PortletRequest request)
      throws PropertyBrokerServiceException {
      PortletSettings settings = request.getPortletSettings();
      Property[] properties = pbService.getAllProperties(request, settings);
      if (properties == null || properties.length == 0) {
         PortletContext context = getPortletConfig().getContext();
         //not registered, register now
         properties = new Property[1];
         properties[0] = PropertyFactory.createProperty(settings);
         properties[0].setName("DEPT_NOParam");
         properties[0].setDirection(Property.OUT);
         properties[0].setType("DEPT_NO");

properties[0].setNamespace("http://www.ibm.com/wps/c2a/examples/hrdetails");
         properties[0].setTitleKey("HRDetails.Department");
         properties[0].setDescriptionKey(
            "Display department details");
         pbService.registerProperties(request, settings, properties);
      }
   }
```

7. Implement the eventPhaseListener interface by inserting the *beginEventPhase* and *endEventPhase* methods, which are callback methods invoked during the event phase. The beginEventPhase invokes the method to register the output property DEPT_NOParam.

   Note: The endEventPhase method does nothing in this scenario but needs to be included in the interface.

Example 25-10 beginEventPhase and endEventPhase methods

```
public void beginEventPhase(PortletRequest request) {
      try {
         registerPropertiesIfNecessary(request);
      }
      catch (Throwable e) {
```

```
            e.printStackTrace();
        }
    }
public void endEventPhase(PortletRequest request) {
    }
```

8. For simplicity, insert a new method with name changeProperty to notify the broker about a changed property value. This method uses the *changedProperties* method to notify property changes.

Example 25-11 Notify the broker of a property value change

```
private void changeProperty(PortletRequest request, String value) {
       System.out.println("send data");
       PortletSettings settings = request.getPortletSettings();
       if (pbService != null) {
          try {
             Property p = PropertyFactory.createProperty(settings);
             p.setName("DEPT_NOParam");
             p.setDirection(Property.OUT);
             p.setType("DEPT_NO");
             p.setNamespace("http://www.ibm.com/wps/c2a/examples/hrdetails");
             PropertyValue[] pva = new PropertyValue[1];
             pva[0] = PropertyFactory.createPropertyValue(p, value);
             pbService.changedProperties(request, getPortletConfig(), pva);

          } catch (Exception e) {
             e.printStackTrace();
          }
       }
    }
```

9. At the end of the actionPerformed method, include the following code to invoke the internal changeProperty method and notify the output property change.

Example 25-12 Invoke method to notify the property value change

```
       if (actionString.equals("DisplayAllDepartmentDetails")) {
          this.changeProperty(request, "*");
       }
```

10. Right-click anywhere in the Java editor and select **Source** → **Organize Imports** to include the missing import statements. In the Organize Imports dialog, choose to import the following class:

    a. com.ibm.wps.pb.service.PropertyBrokerService
    b. org.apache.jetspeed.portlet.service.PortletServiceUnavailableException
    c. com.ibm.wps.pb.service.PropertyBrokerServiceException

Archived

d. com.ibm.wps.pb.property.Property
e. org.apache.jetspeed.portlet.PortletContext
f. com.ibm.wps.pb.property.PropertyFactory
g. com.ibm.wps.pb.property.PropertyValue

11. Save and close the updated source cooperative portlet HRPortlet.java file.

12. Update the HRPortletView.jsp to insert a new action button offering to display department details from all departments. Open the HRPortletView.jsp file and insert the following code at the end of the file and before the </BODY> tag.

    Note: The action name is DisplayAllDepartmentDetails and it will be processed in the actionPerformed method. See Example 25-13.

Example 25-13 New button to offer display of all department details

```
<p align="center">
<FORM method="post"
action="<portletAPI:createReturnURI><portletAPI:URIAction
name='DisplayAllDepartmentDetails'/></portletAPI:createReturnURI>"">
<INPUT type="submit" name="submit" value="Display all department details">
</FORM>
</p>
```

13. Optionally, preview the JSP and see the new button.

14. Save and close the file.

15. Since the source portlet is now using programmatic approach, if there is a HRPortlet/WebContent/wsdl/HRPortlet.wsdl file, delete it and its reference in portlet.xml file. Save and close the file.

25.5.6 Running the programmatic source portlet

Follow these steps to run the updated scenario:

1. Right-click **HRPortlet** and select **Run** → **Run on Server**. Select **WebSphere Portal V5.1 Test Environment** from the server list and click **Finish**.

2. Click the **Display all department details** button. The DepartmentDetailsPortlet displays all department details, as shown in Figure 25-19 on page 799.

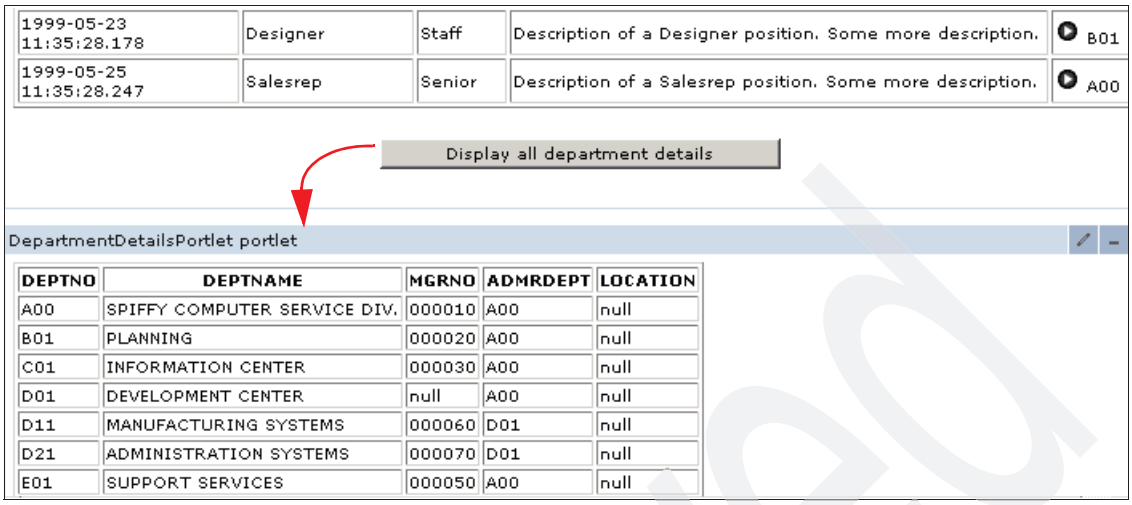


Figure 25-19 Department Details Portlet displays details from all department

3. A portlet wire is required. Try the **Display all department details** button before and after adding the portlet wire.

Important: The method changedProperties() will only trigger events on wired targets. If no wires exist for the published properties, it has no effect.

Archived

Archived

26

JSR 168 cooperative portlets

This chapter describes how to implement the cooperative portlets paradigm using JSR 168 compliant portlets. You will find step-by-step instructions to enable JSR 168 portlets to work in a Cooperative Portlets environment.

The sample scenario included in this chapter illustrates the following:

- Enable a JSR 168 portlet to act as a source cooperative portlet
- Enable a JSR 168 portlet to act as a target cooperative portlet
- Use the wiring tool to wire JSR 168 Cooperative Portlets
- Run the sample scenario in the Portal Test Environment

Note: The portlet application described in this chapter has the following characteristics:

- Portlet API: JSR 168
- Application type: MVC

© Copyright IBM Corp. 2005. All rights reserved.

26.1 Overview

These are some considerations when implementing cooperative portlets using JSR 168 compliant portlets:

- The only way JSR 168 portlets communicate is using wires. Client-to-Action (C2A) is not supported.
- A JSR 168 portlet cannot be wired to an IBM API portlet. You only can create wires between portlets of the same standard.
- There is no c2a tag library for JSR 168 portlets
  a. Implementing a WSDL is the only way for JSR 168 portlets to provide information about properties and actions to the property broker.
  b. JSR 168 portlets must use the setProperties() method to publish properties to the broker.
- JSR 168 portlets only support the declarative approach to register properties and actions. The programmatic approach cannot be used as it is only available for IBM API portlets.

The typical sequence flow for cooperative portlets using JSR 168 compliant portlets is illustrated in Figure 26-1.

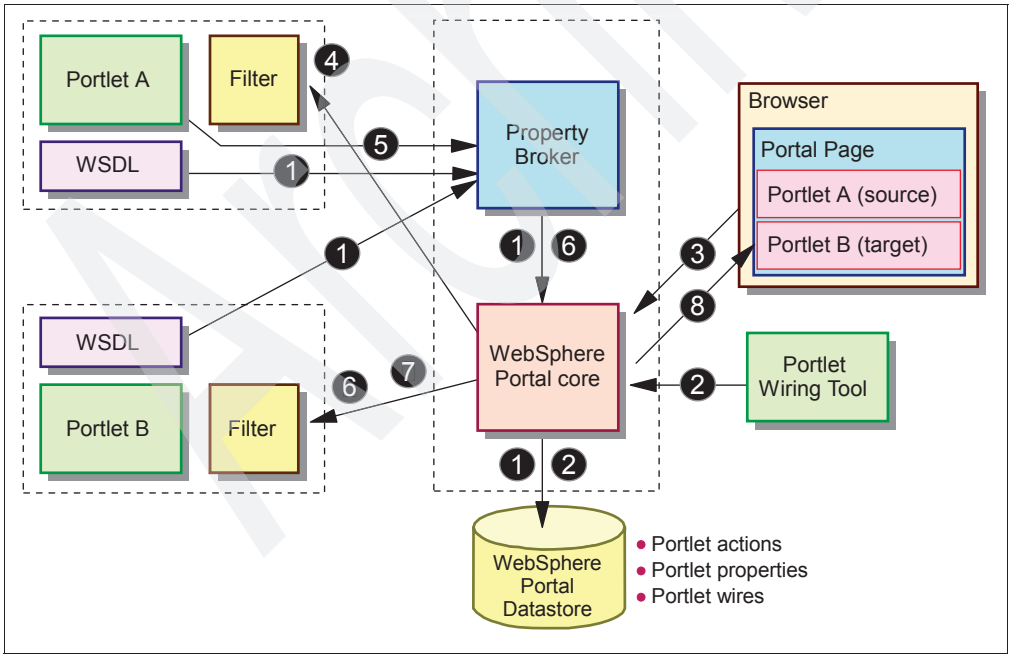


Figure 26-1 Typical processing flow

Archived

Make sure you understand the message flow by walking through the following steps illustrated in Figure 26-1 on page 802:

1. Cooperative portlets deployment:
   a. WSDL files are processed at deployment time
   b. Action and property metadata stored persistently in the portal datastore
2. Wiring:
   a. User (administrator) selects instances of cooperative portlets on a portal page and creates one or more wires between them
   b. Wiring information is also stored persistently in the portal datastore
3. User submits action in source portlet (Portlet A)
4. Portal engine receives the request:
   a. Determines that an action needs to be invoked
   b. Passes action to portlet container
   c. Portlet container delivers action to the source portlet
5. The portlet updates its state and produces new values for output parameters.
   a. Action filter determines the presence of output parameters
   b. For each active wire, it triggers an action invocation on the target portlet (Portlet B) through the portal container
6. Portal container delivers action(s) on the target portlet(s)
7. After action processing terminates:
   a. Render processing begins
   b. Portlet container delivers render method invocations
8. Portal collects markup from each portlet and generates the portal page.

26.2 Source cooperative portlet

In this section, you will execute the following tasks to enable a JSR 168 portlet as a source cooperative portlet using the declarative approach:

- Import the portlet if it is not in your workspace.
- Declare exchange capabilities using WSDL.
- Update the portlet.xml file to point to the WSDL file.
- Update the portlet as follows:
  - In the init() method add a reference to the property broker service

Archived

- In the processAction() method, send the output properties
- In the doView() method, check if the wire is active.

- Update the JSP for View mode to create dynamic links.

26.2.1 Importing the HRPortlet168 portlet

You will enable the HRPortlet168 portlet to act as a source cooperative portlet. Follow these steps to import this portlet into your workspace:

1. Select **File** → **Import**. In the import panel select WAR file. Click **Next**.
2. Enter the following information:
   a. WAR file: browse to C:\LabFiles\JDBC168\HRPortlet168.war
   b. Web project: enter `HRPortlet168`
   c. Target server: select **WebSphere Portal V5.1**
   d. Select **Add module to an EAR project**
   e. EAR project: `HRPortlet168EAR`
3. Verify that you selected **WebSphere Portal V5.1** as the target and click **Finish**.

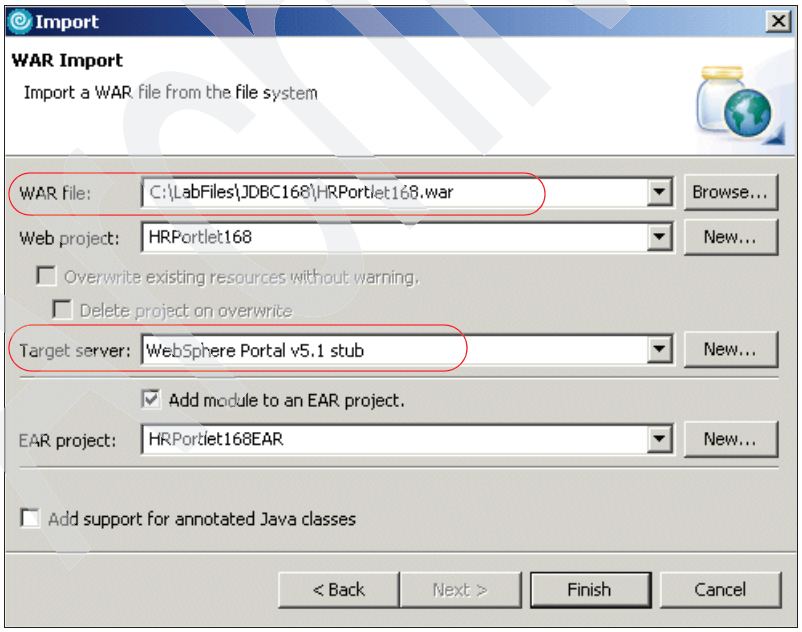


Figure 26-2 Importing the original portlet HRPortlet168

Archived

4. Click **Yes** if you are prompted to switch to the Web perspective.

26.2.2 Internationalization

In this section you will do the following:

- Change the portlet titles. Since you are using the same portlet version as source and target, it is recommended that you change the portlet titles to avoid confusion (English and Default bundles).
- Add key value pairs for the following (English and Default bundles):
  - `department.ID`
  - `show.all employees` (English bundle):

Execute the following steps to add this support to the cooperative source portlet:

1. In **HRPortlet168 → Java Resources → JavaSource → hrportlet168.nl → HRPortlet168Resource.properties**, add the entries highlighted in Example 26-11 on page 818.

Example 26-1 Target portlet HRPortlet168Resource.properties (Default)

```
en Resource Bundle
#
# filename: HRPortlet168Resource_en.properties
# Portlet Info resource bundle example
javax.portlet.title=HRPortlet168 Source
javax.portlet.short-title=
javax.portlet.keywords=

department.ID=department ID
show.all.employees=Show all employees from this department
```

2. In **HRPortlet168 → Java Resources → JavaSource → hrportlet168.nl → HRPortlet168Resource_en.properties**, add the entries highlighted in Example 26-12 on page 819.

Example 26-2 Target portlet HRPortlet168Resource_en.properties (English)

```
en Resource Bundle
#
# filename: HRPortlet168Resource_en.properties
# Portlet Info resource bundle example
javax.portlet.title=HRPortlet168 Source
javax.portlet.short-title=
javax.portlet.keywords=

department.ID=department ID
```

Archived

```
show.all.employees=Show all employees from this department
```

26.2.3 Declaring exchange capabilities using WSDL

You must declare the exchange capabilities of this portlet using a WSDL file. The WSDL file for the source portlet is provided for you as additional materials. Follow these steps to create the WSDL file:

1. From the Project Explorer view, right-click the **HRPortlet168\WebContent** folder and select **New** → **Folder**.
2. Type `wsdl` for the Folder name field and click **Finish**.
3. The directory structure should look as illustrated in Figure 26-3.

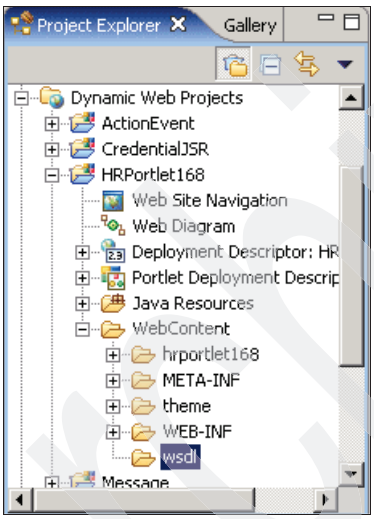


Figure 26-3 New wsdl folder

4. Right-click the **wsdl** folder just created and choose **Import...** → **File system** from the context menu.
5. Browse the directory where the HRPortlet168.wsdl file is located and select this file. The WSDL file is provided as additional material in the following directory:

   c:\LabFiles\Cooperative Portlets\JSR168\HRPortlet168.wsdl
6. Select the **HRPortlet168.wsdl** to be imported.
7. Make sure the **Create selected folders only** option is checked.
8. Click **Finish**.

Archived

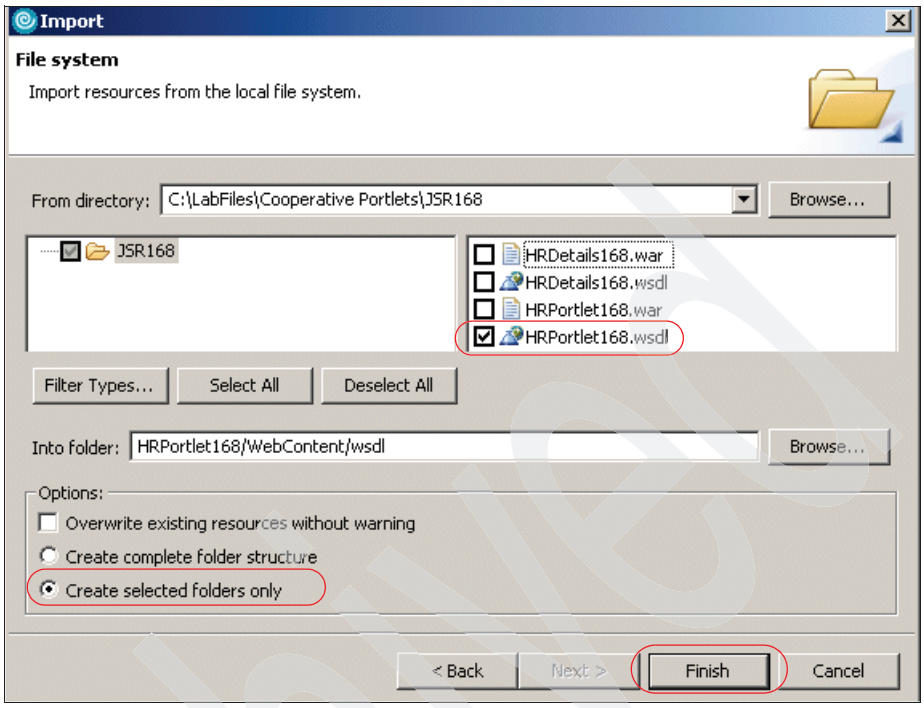


Figure 26-4 Importing wsdl file

9. Example 26-3 illustrates the HRPortlet168.wsdl file

Example 26-3 WSDL file for HRPortlet168 portlet

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DeptResults_Service"
  targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/examples/hrdetails"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<types>
  <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails">
    <xsd:simpleType name="DEPT_NO">
      <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</types>
```

```
<message name="GetResultsMessageNameResponse">
  <part name="dept_Id" type="tns:DEPT_NO"/>
</message>

<portType name="DeptResults_Service">
  <operation name="dept_Detail">
     <output message="tns:GetResultsMessageNameResponse"/>
  </operation>
</portType>

<binding
    name="DeptResultsBinding"
      type="tns:DeptResults_Service">
  <portlet:binding/>
  <operation name="dept_Detail">
    <portlet:action name="departmentList" type="standard"
caption="show.all.employees" description="Get.Results.for.specified.sql.string"
    actionNameParameter="ACTION_NAME"/>
    <output>
      <portlet:param name="DEPT_NO" partname="dept_Id"
            boundTo="request-parameter" caption="department.ID"/>
    </output>
  </operation>
</binding>
</definitions>
```

10. Review the wsdl file:
    - The syntax used in this WSDL file is the standard WSDL with some custom extensions for cooperative portlets.
    - The `<binding>` tag is extended to associate portlet actions with operations.
    - The elements prefixed with `portlet:` are part of the custom WSDL extension schema.
    - For each operation, the portlet action name must be provided using the `name` attribute of the `action` tag.
    - For each operation parameter, the action parameter name must be provided using the `name` attribute of the `param` tag.
    - The `boundTo` attribute may be used to specify where the parameter will be bound, the options are `request-parameter`, `request-attribute`, `session-attribute` or `action-attribute`.
    - In this wsdl file you see that this portlet declares a single action called `departmentList`, using the `portlet:action` element.
    - The type attribute `standard` indicates that it will be implemented as a standard portlet action (JSR 168).

Archived

– Other type attribute options are:

  i. `default:` indicates a DefaultPortletAction object is used

  ii. `simple:` indicates a simple portlet action String is used

  iii. `struts:` indicates a Struts action is used

  iv. `standard-struts:` indicates a struts action is used with a JSR portlet.

– The `portlet:param` element identifies that the action has an output parameter. The **`name`** and **`boundTo`** attributes need to take into account in processAction method.

26.2.4 Updating the portlet deployment descriptor

You will need to update the portlet deployment descriptor (portlet.xml) to include a reference to the WSDL file using the portlet preference parameter called `com.ibm.portal.propertybroker.wsdllocation`.

Execute the following steps:

1. In the Project Explorer view, expand HRPortlet168/WebContent/WEB-INF and double-click in **portlet.xml**.
2. In the Portlets tab:
   a. Select the **HRPortlet168**
   b. Scroll down to the Persistent Preference Store section
3. Add the following portlet preference:
   a. Name: `com.ibm.portal.propertybroker.wsdllocation`
   b. Value: `/wsdl/HRPortlet168.wsdl`

Archived

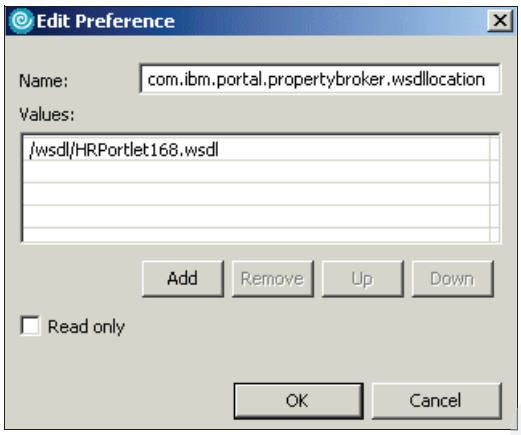


Figure 26-5 Preference

4. In the Source tab, visually verify the new added preference as highlighted in Example 26-4.

Example 26-4 Updating portlet deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
id="hrportlet168.HRPortlet168.8d364d6120">
   <portlet>
      <portlet-name>HRPortlet168</portlet-name>
      <display-name>HRPortlet168</display-name>
      <display-name xml:lang="en">HRPortlet168</display-name>
      <portlet-class>hrportlet168.HRPortlet168</portlet-class>
      <init-param>
         <name>wps.markup</name>
         <value>html</value>
      </init-param>
      <expiration-cache>0</expiration-cache>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>view</portlet-mode>
         <portlet-mode>edit</portlet-mode>
      </supports>
      <supported-locale>en</supported-locale>
      <resource-bundle>hrportlet168.nl.HRPortlet168Resource</resource-bundle>
      <portlet-info>
         <title>HRPortlet168</title>
      </portlet-info>
```

Archived

```
      <portlet-preferences>
         <preference>
            <name>com.ibm.portal.propertybroker.wsdllocation</name>
            <value>/wsdl/HRPortlet168.wsdl</value>
         </preference>
      </portlet-preferences>
   </portlet>
</portlet-app>
```

Note: The portlet descriptor (portlet.xml) should always point to the location of the WSDL file so that Portal will register the portlet properties at initialization time and when the portlet is deployed.

5. If you select the **Problems** tab, you will find that the WSDL file gives you two warning messages; this is not a problem.
6. Save and close the portlet.xml file.

26.2.5 Updating the HRPortlet168 portlet code

The following steps describe the updates you need to make in the HRPortlet168 source cooperative portlet code:

1. From the Project Explorer view, open the `HRPortlet168` portlet. Double-click **HRPortlet168/Java Resources/JavaSource/hrportlet168/HRPortlet168.java**
2. Make sure you include the following import statements:

```
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.portal.portlet.service.PortletServiceHome;
import com.ibm.portal.propertybroker.service.PropertyBrokerService;
```

3. Declare two new variables: `pbService` and `pbServiceAvailable`:

```
PropertyBrokerService pbService = null;
boolean pbServiceAvailable = false;
```

4. Update the `init` method to obtain a reference to the property broker services as highlighted in Example 26-5:

Example 26-5 Obtaining a reference to the property broker service

```
public void init(PortletConfig portletConfig) throws PortletException, UnavailableException {
      super.init(portletConfig);
   try {
      Context ctx = new InitialContext();
      PortletServiceHome serviceHome = (PortletServiceHome)
ctx.lookup("portletservice/com.ibm.portal.propertybroker.service.PropertyBrokerService");
```

Archived

```
      pbService =
(PropertyBrokerService)serviceHome.getPortletService(com.ibm.portal.propertybroker.service.Prop
ertyBrokerService.class);
      pbServiceAvailable = true;
   }catch(Throwable t) {
      getPortletContext().log("OrderDetailPortlet could not find property broker service!");
   }
}
```

Note: The boolean variable pbServiceAvailable will set to true only if the service is available. This variable can be used to guard access to IBM-only services (WebSphere Portal) and it can be used to ensure that the portlet can still function in other environments where the service is not available.

5. You need to declare three new variables: ACTION_NAME, DEPARTMENT_LIST and DEPT_NO as follows:

```
public static final String ACTION_NAME = "ACTION_NAME";
public static final String DEPARTMENT_LIST = "departmentList";
public static final String DEPT_NO = "DEPT_NO";
```

6. In the processAction method, publish an output property, as highlighted in Example 26-6. The property must be set as an attribute (setAttribute) in the request as configured in the WSDL.

Example 26-6 Publishing the output property

```
   public void processAction(ActionRequest request, ActionResponse response)
throws PortletException, java.io.IOException {
      //Add action string handler here
      HRPortlet168SessionBean sessionBean = getSessionBean(request);

      if(request.getPortletMode().equals(PortletMode.EDIT)) {
         String dbname = (String) request.getParameter("dbname");
         String userid = (String) request.getParameter("userid");
         String password = (String) request.getParameter("password");
         String sqlstring = (String) request.getParameter("sqlstring");
         sessionBean.setDbName(dbname);
         sessionBean.setUserId(userid);
         sessionBean.setPassword(password);
         sessionBean.setSqlString(sqlstring);
         if(request.getPortletMode().equals(PortletMode.EDIT))
            response.setPortletMode(PortletMode.VIEW);
      }
      if( request.getParameter(FORM_SUBMIT) != null ) {
         // Set form text in the session bean
         sessionBean = getSessionBean(request);
         if( sessionBean != null )
            sessionBean.setFormText(request.getParameter(TEXT));
      }
```

```
      // add this code
      String actionName = request.getParameter(ACTION_NAME);
      String dept_number = request.getParameter(DEPT_NO);
      if (actionName != null && actionName.equalsIgnoreCase(DEPARTMENT_LIST))
{
         request.getPortletSession().setAttribute(ACTION_NAME,
               DEPARTMENT_LIST);
         request.setAttribute(DEPT_NO, dept_number);
      }
      // end of added code
   }
```

7. Save your files.
8. Review the code:
   - Here is where you need to take into account the attributes `name` and `boundTo` of `portlet:param` element mentioned previously in Example 26-3 on page 807.
   - The `name` attribute of `portlet:action` element must match with the request parameter to determine the action name (`departmentList` in our example).
   - When the action matches it sets the value of output parameter (DEPT_NO).
   - Depending on the value of boundTo attribute, the variable will be set as:
     - request attribute, if the value of boundTo attribute is request-attribute
     - session attribute, if the value of boundTo attribute is session
     - request parameter, if the value of boundTo attribute is request-parameter. This is the default value for this attribute.
9. In the Project Explorer view:
   a. Open the HRPortlet168ViewBean bean located in HRPortlet168/Java Resources/JavaSource/hrportlet168/.
   b. Add a variable to indicate if the department code property is wired to an active action and add its corresponding getter and setter methods.
   c. Necessary changes on HRPortlet168ViewBean bean are highlighted in Example 26-7.

Example 26-7 Setting an indicator on HRPortlet168ViewBean

```
package hrportlet168;
import com.ibm.db.beans.*;

public class HRPortlet168ViewBean {
   private DBSelect resultFromDatabase;
   // add this code
```


Archived

```
   private boolean deptIDActive;

   public boolean isDeptIDActive() {
      return deptIDActive;
   }

   public void setDeptIDActive(boolean deptIDActive) {
      this.deptIDActive = deptIDActive;
   }
   // end of added code

   public DBSelect getResultFromDatabase() {
      return resultFromDatabase;
   }

   public void setResultFromDatabase(DBSelect resultFromDatabase) {
      this.resultFromDatabase = resultFromDatabase;
   }
}
```

10. Save the HRPortlet168ViewBean.java file.

11. Return to HRPortlet168 portlet code and add the following code to the doView method to inform the JSP whether or not the service is available and there are wires active so that dynamic links should be shown.

Example 26-8 Set the indicator value in doView method

```
   public void doView(RenderRequest request, RenderResponse response) throws
PortletException, IOException {
      response.setContentType(request.getResponseContentType());
      HRPortlet168SessionBean sessionBean = getSessionBean(request);
      if( sessionBean==null ) {
         response.getWriter().println("<b>NO PORTLET SESSION YET</b>");
         return;
      }
      String sqlstring = sessionBean.getSqlString();
      if (sqlstring != null && !sqlstring.equals("")) {
         String dbname = sessionBean.getDbName();
         String userid = sessionBean.getUserId();
         String password = sessionBean.getPassword();
         HRPortlet168ViewBean viewBean = new HRPortlet168ViewBean();
         SQLUtilities sqlUtility = new SQLUtilities();
         try {
            sqlUtility.execute(userid, password, sqlstring);
         } catch (SQLException e) {
            e.printStackTrace();
         }
         sqlUtility.populateData(viewBean);
```

Archived

```
            // add this code
            if (pbServiceAvailable == true) {
               try {
                  viewBean.setDeptIDActive(pbService.areWiresActive(request,
DEPT_NO));
               } catch (Exception e) {

               }
            }
            // end of added code
            request.setAttribute(VIEW_BEAN, viewBean);
         }
         PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher(getJspFilePath(request, VIEW_JSP));
         rd.include(request,response);
      }
```

12. Review the added code:
    - The areWiresActive method of the PropertyBrokerService interface tests if a property is currently wired to active actions.
    - It receives the following:
      - PortletRequest object for the current portlet request
      - The name of the property to check.
13. Save the HRPortlet168.java file.

26.2.6 Updating the JSP to generate a link

In this section you will update the HRPortlet168View.jsp located in `HRPortlet168/WebContent/hrportlet168/jsp/html` to generate links in the DEPT_NO column.

1. In HRPortlet168View.jsp replace the code:

```
<TD>
   <p><%=results.getCacheValueAt(row, col)%></p>
</TD>
```

with the code shown in Example 26-9. The code illustrates how to generate a link when the boolean variable, indicating that the property is wired to an action, is true.

Example 26-9 Generating a link in the HRPortlet168View.jsp

```
<TD>
<%-- If the output property "deptNo" is wired with active actions, then set a link to trigger
action--%>
```

```
<%  if ( results.getColumnName(col).equalsIgnoreCase(HRPortlet168.DEPT_NO) &&
viewBean.isDeptIDActive() ) {
   PortletURL actionURL = renderResponse.createActionURL();
   actionURL.setParameter(HRPortlet168.DEPT_NO, (String)results.getCacheValueAt(row, col));
   actionURL.setParameter(HRPortlet168.ACTION_NAME, HRPortlet168.DEPARTMENT_LIST);
%>
<A href="<%= actionURL%>">
   <p><%=results.getCacheValueAt(row, col)%></p>
</A>
<% } else { %>
   <p><%=results.getCacheValueAt(row, col)%></p>
<% } %>
</TD>
```

2. Save your files.

Note: Ignore the warnings indicating that there are unhandled exception types (SQLException). This is because there are no try and catch sections in the code.

26.3 Target cooperative portlet

In this section, you will import a second copy of the HRPortlet168 and update it to support cooperation as a target portlet. This portlet will execute a fixed SQL statement with a variable `where` clause.

Execute the following steps:

1. To import a second version of the HRPortlet168, proceed as described in “Importing the HRPortlet168 portlet” on page 804, replace the HRPortlet168 Web project name with HRDetails168 and the EAR project with HRDetails168EAR.

   Note: Make sure you select **WebSphere Portal V5.1** as a target server.

Archived

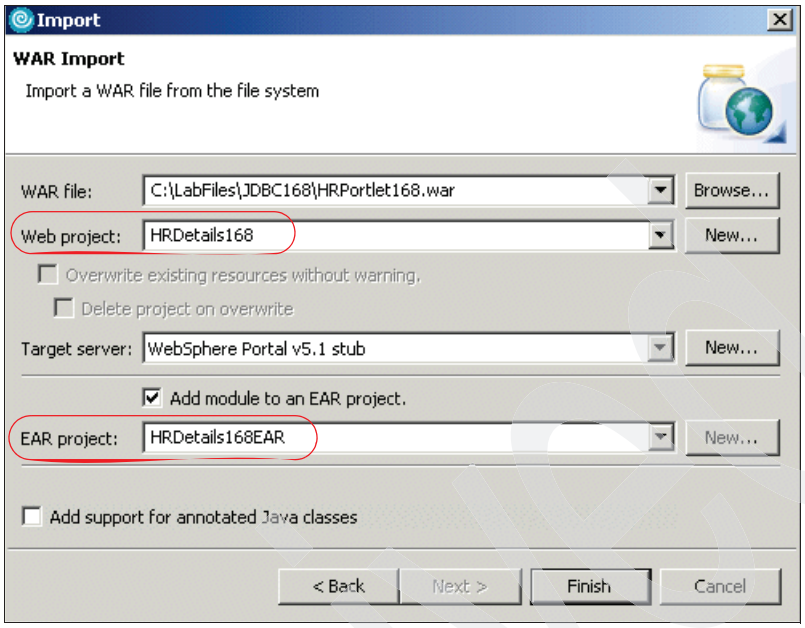


Figure 26-6 Importing a second version of the portlet

2. HRPortlet168 and the new HRDetails168 portlet application use the same UID so it will cause an error executing the portlets. To fix this problem follow these steps:

   a. Expand the **HRDetails168/WebContent/WEB-INF** folder and double-click in portlet.xml.

   b. Change the portlet UID to avoid duplicates. In the portlet deployment descriptor editor, select the **Source** panel and change the last digit of the UID of this portlet. For example, in this scenario, the last digit is 0 and you will change it to 1, as highlighted in Example 26-10.

Example 26-10 Changing a digit in portlet application ID

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
id="hrportlet168.HRPortlet168.8d364d6121">
   <portlet>
      <portlet-name>HRPortlet168</portlet-name>
      <display-name>HRPortlet168</display-name>
      <display-name xml:lang="en">HRPortlet168</display-name>
      <portlet-class>hrportlet168.HRPortlet168</portlet-class>
```

```
      <init-param>
         <name>wps.markup</name>
         <value>html</value>
      </init-param>
      <expiration-cache>0</expiration-cache>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>view</portlet-mode>
         <portlet-mode>edit</portlet-mode>
      </supports>
      <supported-locale>en</supported-locale>
      <resource-bundle>hrportlet168.nl.HRPortlet168Resource</resource-bundle>
      <portlet-info>
         <title>HRPortlet168</title>
      </portlet-info>
   </portlet>
</portlet-app>
```

3. Save your changes.

26.3.1 Internationalization

In this section, you will do the following:

- Change the portlet titles. Since you are using the same portlet version as source and target, it is recommended that you change the portlet titles to avoid confusion (English and Default bundles).
- Add key value pairs for the following (English and Default bundles):
  - `department.ID`
  - `show.all employees` (English bundle):

Execute the following steps to add this support to the cooperative target portlet:

1. Target portlet. In **HRDetails168** → **Java Resources** → **JavaSource** → **hrportlet168.nl** → **HRPortlet168Resource.properties**, add the entries highlighted in Example 26-11.

Example 26-11 Target portlet HRPortlet168Resource.properties (Default)

```
en Resource Bundle
#
# filename: HRPortlet168Resource_en.properties
# Portlet Info resource bundle example
javax.portlet.title=HRPortlet168 Target
javax.portlet.short-title=
javax.portlet.keywords=

department.ID=department ID
```

```
show.all.employees=Show all employees from this department
```

2. Target portlet. In **HRDetails168** → **Java Resources** → **JavaSource** → **hrportlet168.nl** → **HRPortlet168Resource_en.properties**, add the entries highlighted in Example 26-12.

Example 26-12 Target portlet HRPortlet168Resource_en.properties (English)

```
en Resource Bundle
#
# filename: HRPortlet168Resource_en.properties
# Portlet Info resource bundle example
javax.portlet.title=HRPortlet168 Target
javax.portlet.short-title=
javax.portlet.keywords=

department.ID=department ID
show.all.employees=Show all employees from this department
```

26.3.2 Declaring exchange capabilities using WSDL

The following updates are needed in order to enable the HRDetails168 to work as a target cooperative portlet.

1. From the Project Explorer view right-click the **HRDetails168\WebContent** folder and select **New** → **Folder**.
2. Type `wsdl` for the Folder name field and click **Finish**.
3. Right-click the wsdl folder just created and choose **Import...** → **File system** from the context menu.
4. Browse the directory where the `HRDetails168.wsdl` file is located and select this file. The WSDL file is provided as additional material in the following directory:

   c:\LabFiles\Cooperative Portlets\JSR168\HRDetails.wsdl
5. Select the **HRDetails168.wsdl** to be imported.
6. Make sure the **Create selected folders only** option is checked.

Archived

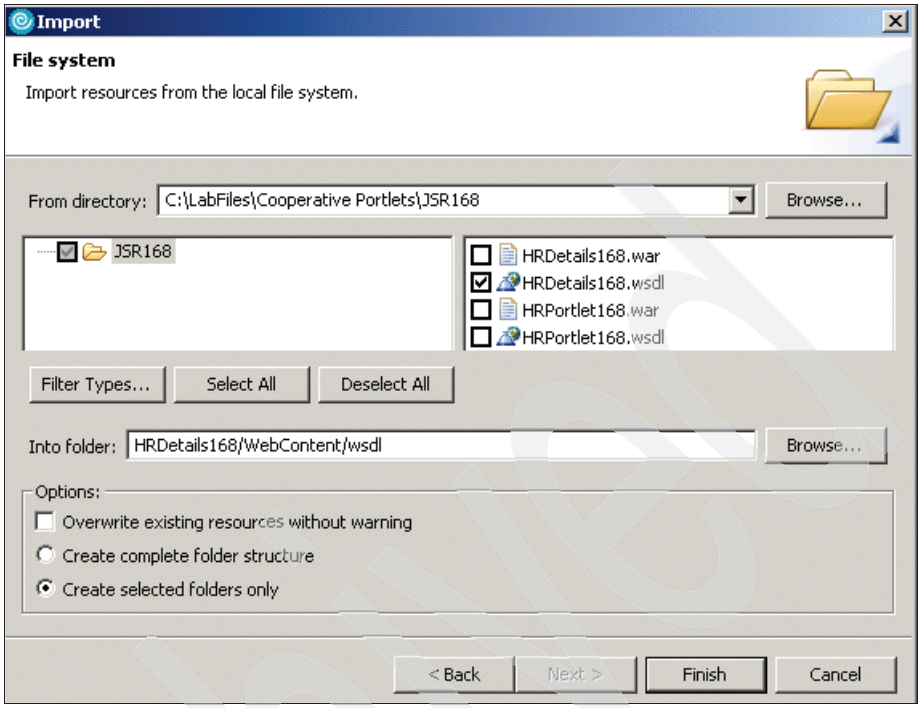


Figure 26-7 Importing HRDetails.wsdl

7. Click **Finish**.
8. Example 26-13 illustrates the HRDetails168.wsdl file.

Example 26-13 HRDetails168.wsdl file

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DeptResults_Service"
  targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/examples/hrdetails"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<types>
  <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails">
    <xsd:simpleType name="DEPT_NO">
      <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
```

```
  </xsd:schema>
</types>

<message name="GetResultsMessageNameRequest">
  <part name="dept_Id" type="tns:DEPT_NO"/>
</message>

<portType name="DeptResults_Service">
  <operation name="dept_Detail">
     <input message="tns:GetResultsMessageNameRequest"/>
  </operation>
</portType>

<binding
    name="DeptResultsBinding"
      type="tns:DeptResults_Service">
  <portlet:binding/>
  <operation name="dept_Detail">
    <portlet:action name="departmentList" type="standard"
caption="show.all.employees" description="Get.Results.for.specified.sql.string"
    actionNameParameter="ACTION_NAME"/>
    <input>
      <portlet:param name="DEPT_NO" partname="dept_Id"
caption="department.ID"/>
    </input>
  </operation>
</binding>
</definitions>
```

Note: Here the departmentList action has an input parameter element named DEPT_NO.

Note: While the number of output parameters associated with an action is unlimited, input parameters can only be received one at a time in the processAction() method.

26.3.3 Updating the portlet deployment descriptor

You will need to update the portlet deployment descriptor (portlet.xml) to include a reference to the WSDL file using the portlet preference parameter called `com.ibm.portal.propertybroker.wsdllocation`.

Execute the following steps:

1. In the Project Explorer view, expand **HRDetails168/WebContent/WEB-INF** and double-click in **portlet.xml**.

Archived

2. In the Portlets tab:
   a. Select the **HRPortlet168**
   b. Scroll down to the Persistent Preference Store section
3. Add the following portlet preference:
   a. name: `com.ibm.portal.propertybroker.wsdllocation`
   b. value: `/wsdl/HRDetails168.wsdl`
4. The portlet.xml with the new reference to the WSDL file is illustrated in Example 26-14.

Example 26-14 Update HRDetails168 portlet.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
id="hrportlet168.HRPortlet168.8d364d6121">
   <portlet>
      <portlet-name>HRPortlet168</portlet-name>
      <display-name>HRPortlet168</display-name>
      <display-name xml:lang="en">HRPortlet168</display-name>
      <portlet-class>hrportlet168.HRPortlet168</portlet-class>
      <init-param>
         <name>wps.markup</name>
         <value>html</value>
      </init-param>
      <expiration-cache>0</expiration-cache>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>view</portlet-mode>
         <portlet-mode>edit</portlet-mode>
      </supports>
      <supported-locale>en</supported-locale>
      <resource-bundle>hrportlet168.nl.HRPortlet168Resource</resource-bundle>
      <portlet-info>
         <title>HRPortlet168</title>
      </portlet-info>
      <portlet-preferences>
         <preference>
            <name>com.ibm.portal.propertybroker.wsdllocation</name>
            <value>/wsdl/HRDetails168.wsdl</value>
         </preference>
      </portlet-preferences>
   </portlet>
</portlet-app>
```

Archived

5. From the Project Explorer view, open the HRPortlet168 portlet located in:

   /HRDetails168/Java Resources/JavaSource/hrportlet168/HRPortlet168.java

6. Include the following import statements in the HRPortlet168 portlet:

```
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.portal.portlet.service.PortletServiceHome;
import com.ibm.portal.propertybroker.service.PropertyBrokerService;
```

7. Declare two new variables: pbService and pbServiceAvailable:

```
PropertyBrokerService pbService = null;
boolean pbServiceAvailable = false;
```

8. In the same way as with the source portlet, update the init method to obtain a reference to the property broker services as highlighted in Example 26-15:

Example 26-15 Obtaining a reference to the property broker service

```
public void init(PortletConfig portletConfig) throws PortletException, UnavailableException {
      super.init(portletConfig);
   try {
      Context ctx = new InitialContext();
      PortletServiceHome serviceHome = (PortletServiceHome)
ctx.lookup("portletservice/com.ibm.portal.propertybroker.service.PropertyBrokerService");
      pbService =
(PropertyBrokerService)serviceHome.getPortletService(com.ibm.portal.propertybroker.service.Prop
ertyBrokerService.class);
      pbServiceAvailable = true;
   }catch(Throwable t) {
      getPortletContext().log("OrderDetailPortlet could not find property broker service!");
   }
}
```

9. You need to declare three new variables: ACTION_NAME, DEPARTMENT_LIST and DEPT_NO as follows:

```
public static final String ACTION_NAME = "ACTION_NAME";
public static final String DEPARTMENT_LIST = "departmentList";
public static final String DEPT_NO = "DEPT_NO";
```

10. Update the processAction method to receive the property value. as highlighted in Example 26-16.

Example 26-16 Receiving a property value in processAction method

```
   public void processAction(ActionRequest request, ActionResponse response)
throws PortletException, java.io.IOException {
      HRPortlet168SessionBean sessionBean = getSessionBean(request);
      if(request.getPortletMode().equals(PortletMode.EDIT)) {
         String dbname = (String) request.getParameter("dbname");
```

Archived

```
            String userid = (String) request.getParameter("userid");
            String password = (String) request.getParameter("password");
            String sqlstring = (String) request.getParameter("sqlstring");
            sessionBean.setDbName(dbname);
            sessionBean.setUserId(userid);
            sessionBean.setPassword(password);
            sessionBean.setSqlString(sqlstring);
         }
         if( request.getParameter(FORM_SUBMIT) != null ) {
            sessionBean = getSessionBean(request);
            if( sessionBean != null )
               sessionBean.setFormText(request.getParameter(TEXT));
         }
         // add this code
         String actionName = request.getParameter(ACTION_NAME);
         if (actionName != null && actionName.equalsIgnoreCase(DEPARTMENT_LIST))
{
            String dept_number = (String)request.getParameter(DEPT_NO);
            request.getPortletSession().setAttribute(ACTION_NAME,
DEPARTMENT_LIST);
            HRPortlet168SessionBean bean = getSessionBean(request);
            bean.setSqlString("select * from employee where workdept='" +
dept_number + "'");
         }
         // end of added code
      }
```

Note: The input parameter DEPT_NO is retrieved as request parameter because boundTo attribute has been omitted in the WSDL file.

26.4 Running the cooperative portlets

Before you run the scenario in the Portal test Environment, you will need to populate the database and create a data source required by the portlet application.

26.4.1 Populating the sample database

If the Cloudscape sample database has not been populated, run the batch file to populate the test database to be used in this scenario. For example:

1. Create the c:\HRproject\database\ directory.
2. From c:\Labfiles\JDBC168\ folder, copy the following files to this directory:
   – CreateCloudTable.bat

– Tables.sql

3. Edit the provided CreateCloudTable.bat file and make sure the paths for variables JAVA_HOME and DB2J_LIB point to the correct location. For example, in this sample scenario, these directories (java and lib) can be found at <RAD_root>\runtimes as shown in Example 26-17.

Example 26-17 Sample CreateCloudTable.bat file

```
set JAVA_HOME=C:\Progra~1\IBM\Rational\SDP\6.0\runtimes\base_v51\java
set DB2J_LIB=C:\Progra~1\IBM\Rational\SDP\6.0\runtimes\base_v51\cloudscape\lib
set
CLASSPATH=%DB2J_LIB%\db2j.jar;%DB2J_LIB%\db2jtools.jar;%DB2J_LIB%\db2jcview.jar
;%DB2J_LIB%\jh.jar
%JAVA_HOME%\bin\java -Dij.connection.myconn=jdbc:db2j:WSSAMPLE;create=true
-Dcloudscape.system.home=%DB2J_LIB% -ms16m -mx32m com.ibm.db2j.tools.ij
Tables.sql
```

4. Save and close the file.
5. Open a command window and change to the directory where these files are.
6. Execute the CreateCloudTable.bat to create and populate the sample database (WSSAMPLE).
7. After executing the batch file, the WSSAMPLE folder is created in the c:\HRproject\database.
8. Display the directory to make sure WSSAMPLE was created.

26.4.2 Creating a data source

Before you run the HRPortlet168, you will need to create the DataSource used by the SQLUtility to establish the connection to the database.

For example, in the Web perspective:

1. Select the Server view in located on the bottom center of the workspace.
2. Double-click the **WebSphere Portal V5.1 Test Environment**. This will open the Server config editor.

   Note: Right-click the server view to create a new instance of the WebSphere Portal V5.1 Test Environment if you do not have one already.
3. Select the **Data source** tab.
4. In the Server Settings section, select **Cloudscape JDBC Driver** from the JDBC provider list.
5. Click **Add...** from the Data source defined in the JDBC provider selected above.

Archive

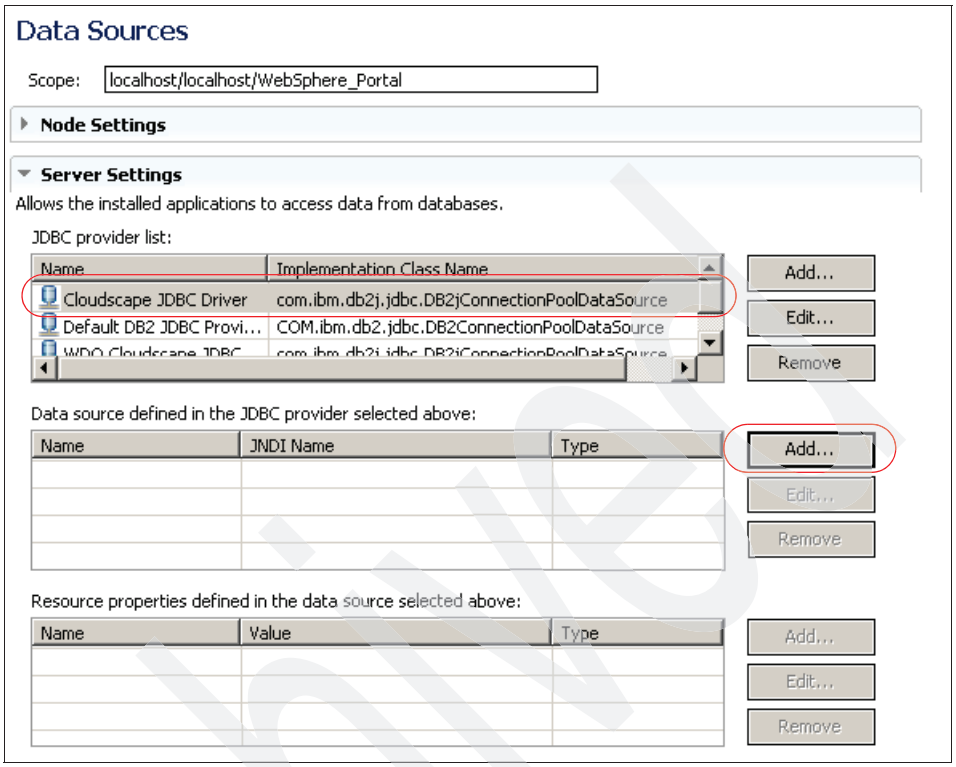


Figure 26-8 Create Data Source

6. In the Create a Data source window, select **Cloudscape JDBC provider** and **data source V5**. Click **Next**.
7. Enter the following values for the Modify Data Source window.
   - Name: `WSSample`
   - JNDI Name: `jdbc/WSSample`

Archive

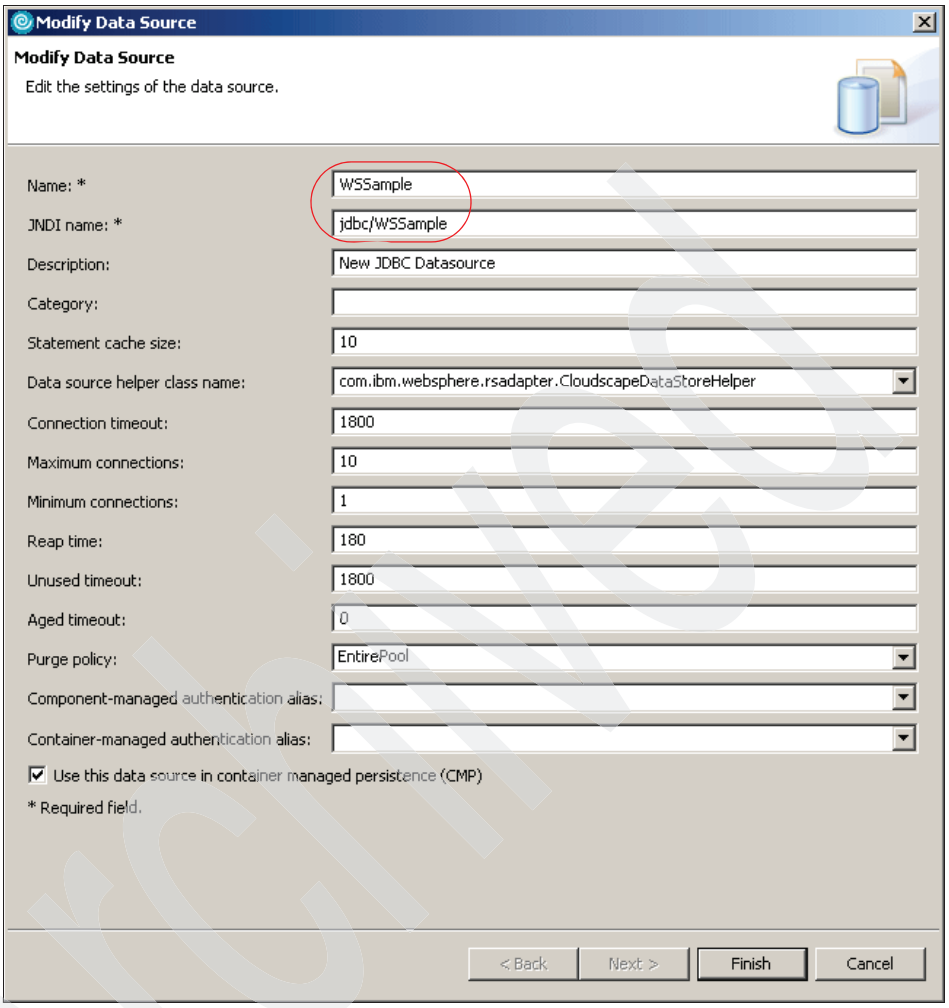


Figure 26-9 Data Source modifications

8. Click **Next**.
9. Select **databaseName** from Resource Properties, enter `C:\HRproject\database\WSSAMPLE` as the value (or the path where the database is stored).

Archived

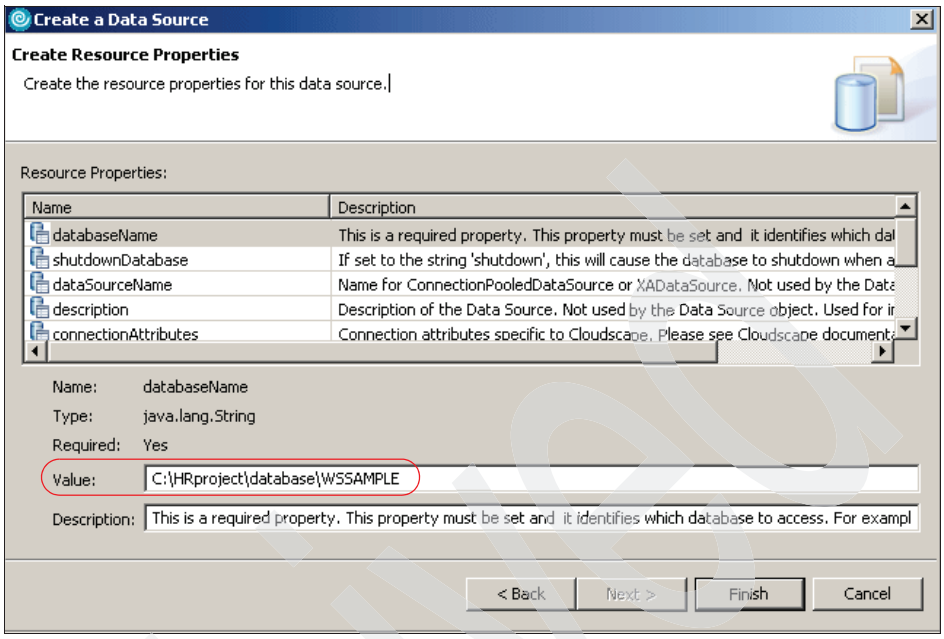


Figure 26-10 Create resource properties

10. Click **Finish**.
11. Close the Server configuration editor.
12. A window will prompt you if you want to save changes. Select **Yes.**

Important: As a general rule, make sure there are no other connections active to the database you plan to use as this could create conflict errors.

26.4.3 Running and wiring the cooperative portlets

Execute the following steps to prepare the cooperative portlets scenario:

1. By default, WebSphere Portal Test Environment enables multiples pages when deploying portlet projects. For cooperative portlets you need the portlets on the same page. You also need administration capabilities to execute the wiring tool. Therefore you need to configure these options:
   a. In the Servers tab, double-click the WebSphere Portal Test Environment
   b. Select **Portal Options** as shown in Figure 26-11 on page 829.
      i. Check the option **Enable base portlets for portal administration and customization**.

ii. Uncheck the option **Enable multiple pages when deploying portlet projects**.

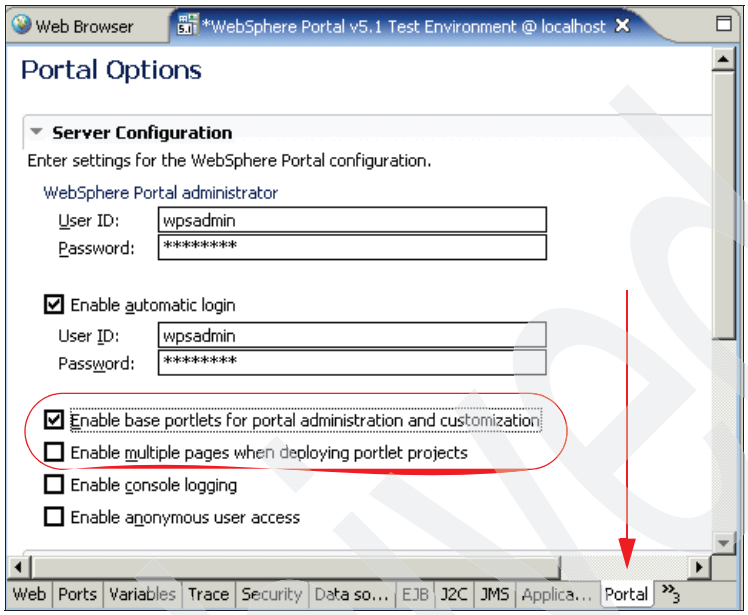


Figure 26-11 Portal options

2. Save the configuration and close the editor.
3. Add the portlet projects to the WebSphere Portal V5.1 Test Environment server:
   a. Right-click the server and select **Add and remove projects**.
   b. From the list of available projects, select `HRPortlet168` and click **Add >** to add this project to the list of configured projects on the right side.
   c. Repeat the same steps to add the `HRDetails168` project.
   d. Click **Finish**.
4. Right-click **HRPortlet168**. Then click **Run** → **Run on server,** select **WebSphere Portal V5.1 Test Environment** server and click **Finish**.

   Note: This will load your project into the test environment so that you can view it in Rational Application Developer internal browser. It may take few minutes for this process to complete.

5. Wiring. Click **Edit Page** and you will see the two portlets in the Edit Layout page.

Archived

6. Click the **Wires** tab. You will see different combos to select a source portlet, a property to send, a target portlet and a property or action to receive. If you select **Global wire**, that means that their effects are manifested to all users that can view the page and portlets.

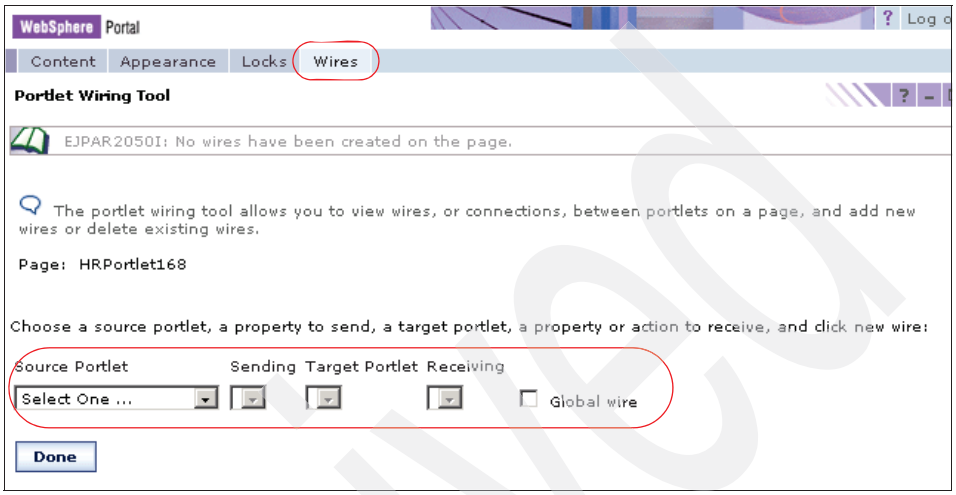


Figure 26-12 Portlet Wiring Tool

7. Select the following values:
   a. `HRPortlet168` as the Source Portlet.
   b. Sending combo: select **department ID**.
   c. `HRPortlet168` as the Target Portlet.
   d. Receiving combo: select **Show all employees, department ID**.
   e. Check **Global wire.**
   f. Click **Add wire.**

Archived

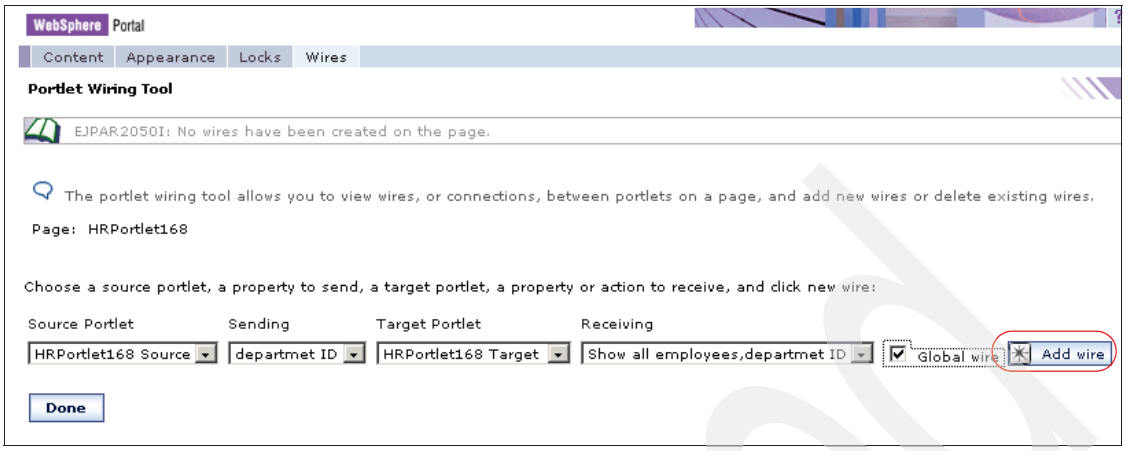


Figure 26-13 Portlet Wiring Tool

8. You will now see the new wire as shown in Figure 26-14.

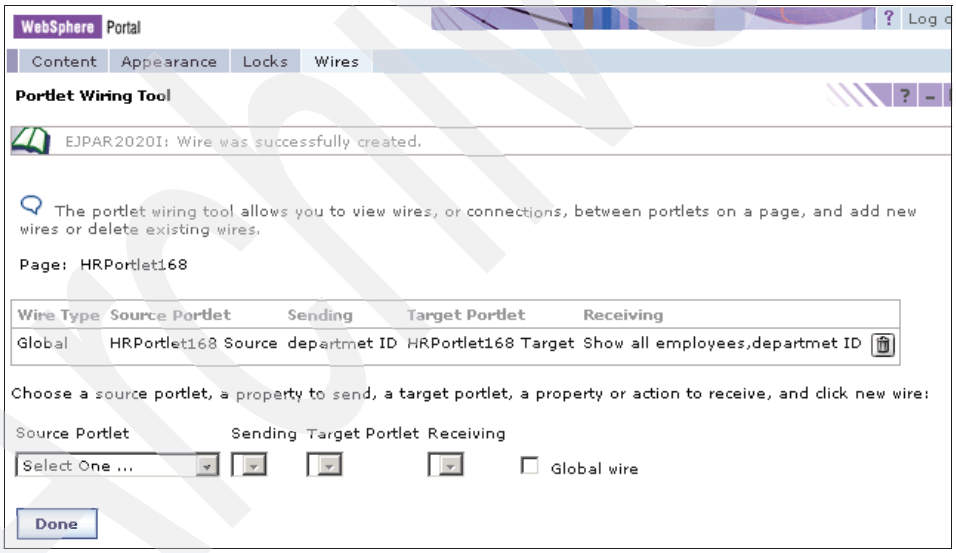


Figure 26-14 Portlet Wiring Tool with new wire

9. Click **Done**.
10. Switch to Edit mode in HRPortlet168 Source portlet by clicking the pencil icon in the top right-hand corner of the portlet.
11. Enter the following information and click **Submit**:
    a. User: `db2admin`

Archived

b. Password: `db2admin`

c. SQL: `select * from jobs`

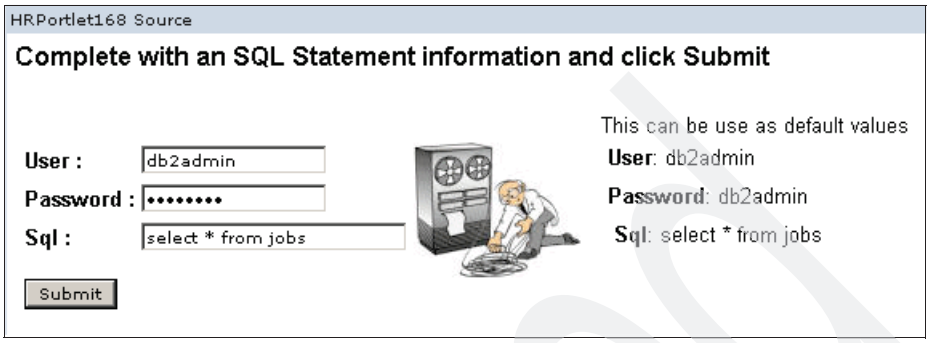
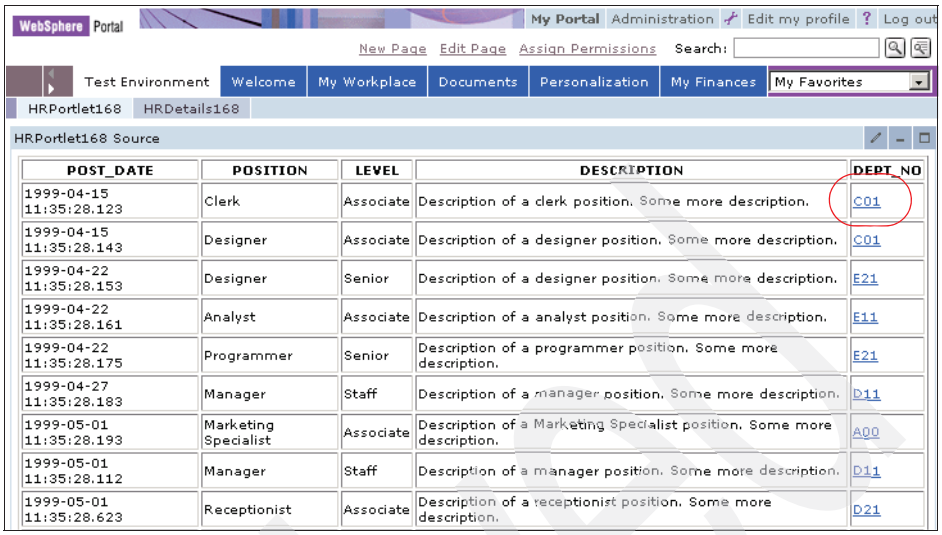


Figure 26-15 Portlet Wiring Tool with new wire

12. The HRPortlet168 Source portlet now displays the jobs table, including links in the DEPT_NO column. Before you can use the links, you have to configure the user and password in HRDetails168 portlet.
13. Switch to Edit mode in HRDetails168 (HRPortlet168 Target) portlet by clicking the pencil icon in the top right-hand corner of the portlet.
14. Enter the following information and click Submit:
    a. User: `db2admin`
    b. Password: `db2admin`
    c. It is not necessary to enter an SQL statement.
15. In the source cooperative portlet View mode, click a DEPT_NO column link, for example **C01**.

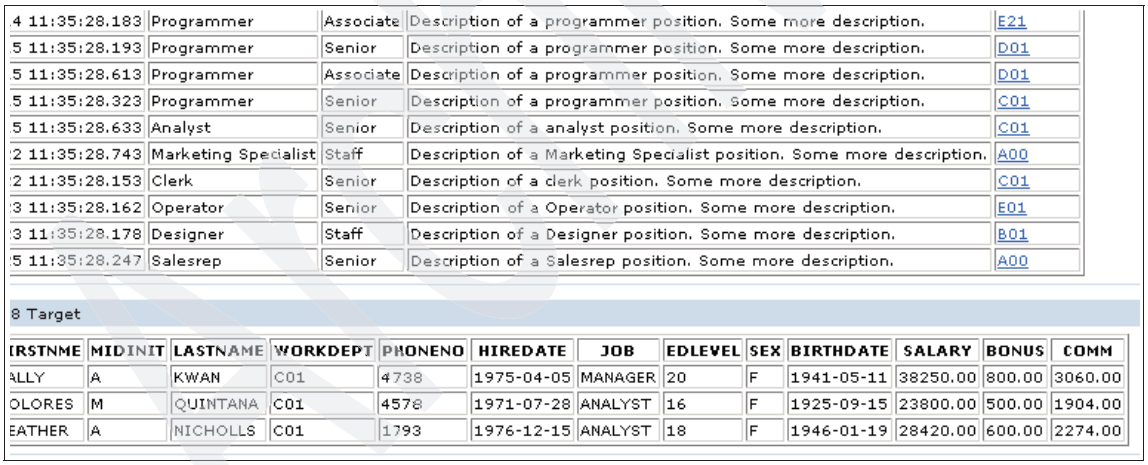
Archived



POST_DATE	POSITION	LEVEL	DESCRIPTION	DEPT_NO
1999-04-15 11:35:28.123	Clerk	Associate	Description of a clerk position. Some more description.	C01
1999-04-15 11:35:28.143	Designer	Associate	Description of a designer position. Some more description.	C01
1999-04-22 11:35:28.153	Designer	Senior	Description of a designer position. Some more description.	E21
1999-04-22 11:35:28.161	Analyst	Associate	Description of a analyst position. Some more description.	E11
1999-04-22 11:35:28.175	Programmer	Senior	Description of a programmer position. Some more description.	E21
1999-04-27 11:35:28.183	Manager	Staff	Description of a manager position. Some more description.	D11
1999-05-01 11:35:28.193	Marketing Specialist	Associate	Description of a Marketing Specialist position. Some more description.	A00
1999-05-01 11:35:28.112	Manager	Staff	Description of a manager position. Some more description.	D11
1999-05-01 11:35:28.623	Receptionist	Associate	Description of a receptionist position. Some more description.	D21

Figure 26-16 Running HRPortlet168 portlet

16. You will see all the employees for this department listing in HRDetails168 portlet.



4 11:35:28.183	Programmer	Associate	Description of a programmer position. Some more description.	E21
5 11:35:28.193	Programmer	Senior	Description of a programmer position. Some more description.	D01
5 11:35:28.613	Programmer	Associate	Description of a programmer position. Some more description.	D01
5 11:35:28.323	Programmer	Senior	Description of a programmer position. Some more description.	C01
5 11:35:28.633	Analyst	Senior	Description of a analyst position. Some more description.	C01
2 11:35:28.743	Marketing Specialist	Staff	Description of a Marketing Specialist position. Some more description.	A00
2 11:35:28.153	Clerk	Senior	Description of a clerk position. Some more description.	C01
3 11:35:28.162	Operator	Senior	Description of a Operator position. Some more description.	E01
3 11:35:28.178	Designer	Staff	Description of a Designer position. Some more description.	B01
5 11:35:28.247	Salesrep	Senior	Description of a Salesrep position. Some more description.	A00

8 Target

IRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
ALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
OLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00
EATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00

Figure 26-17 HRDetails168 portlet displays all employees for same department

Archived

27

Struts cooperative portlets

This chapter describes how to implement cooperative portlets with Struts portlets using the JSR 168 API. Two Struts portlets (source and target) will be used to implement a messaging communication path to allow the Struts portlets to share information at execution time.

Note: The portlet application described in this chapter has the following characteristics:

- Portlet API: JSR 168
- Application type: Struts

© Copyright IBM Corp. 2005. All rights reserved.

27.1 Overview

In this scenario, two different versions of the same portlet, called `Echo` portlet, are used. You will change the portlet UID of one of the portlets to be able to run the two portlets in the Portal Test Environment. As and option, you will also change the portlet name and the portlet title to visually identify the portlets.

The `Echo` portlet has a simple form that allows the user to enter a message. When the message is submitted, it is displayed back on the same portlet. In this scenario, the portlets will be enhanced to work in a cooperative portlet environment.

The following portlets are used:

- The source cooperative portlet, named `EchoSource`, shows a form to submit a message and will be enhanced to send the message to the Cooperative Portlets property broker.
- The target cooperative portlet, named `EchoTarget`, will be enhanced to receive the message from the Cooperative Portlets property broker and display it.

Note: The Cooperative Portlets property broker uses Struts Actions to notify input properties.

The sample scenario is illustrated in Figure 27-1. The Struts source portlet publishes (broadcasts) an output property and the Struts target portlet receives the message.

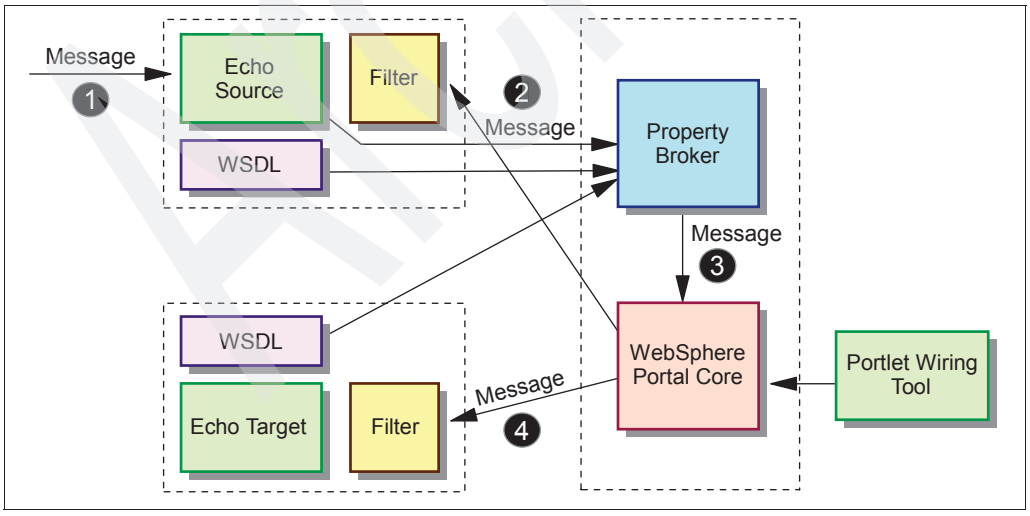


Figure 27-1 Cooperative Portlets sample scenario

Archived

This scenario shows various cooperative portlet concepts in Struts portlets using the JSR 168 API.For example:

- You will enable portlet cooperation using a declarative approach. For JSR 168 compliant portlets, WSDL is the only way to provide information about portlet actions as programmatic cooperative portlets are not available for JSR 168 portlets.
- You will create wires between cooperative portlets using the Portlet Wiring Tool. This is the only way to connect JSR 168 compliant portlets as Click-to-Action is not available for JSR 168 portlets.

27.2 Source cooperative portlet

In this section, you will execute the following tasks to enable a source portlet for cooperative portlets using the declarative approach:

1. Import the `Echo` portlet. This Struts portlet is provided as additional material and we assume that it has been downloaded to the following directory:

   c:\LabFiles\Struts Cooperative Portlets\
2. Declare exchange capabilities using WSDL.
3. Update the portlet descriptor (portlet.xml) file to refer to the WSDL file.
4. Update the `welcome` struts action to get a reference to the property broker service.
5. Add internationalization of captions and descriptions.

27.2.1 Importing the Echo portlet

Follow these steps to import the `Echo` portlet:

1. Select **File** → **Import**. In the import panel select WAR file and click **Next**.
2. In the next window, enter the following information:
   a. WAR file: browse to C:\LabFiles\Struts Cooperative Portlets\Echo.war
   b. Web project: enter `EchoSource`
   c. Target server: select **WebSphere Portal V5.1**
   d. Select **Add module to an EAR project**
   e. EAR project: `EchoSourceEAR`
3. Click **Finish**.

Archived

It is recommended that you change the portlet title to be able to identify the portlets since they will be deployed on the same page. Follow these steps for each resource bundle:

1. Open the resource bundles (EchoPortletResource properties file), located in the EchoSource/Java Resources/JavaSource/echo.nl directory.
2. Add a title in the line `javax.portlet.title`. For example:

   ```
   javax.portlet.title=Echo Source portlet
   ```

3. Save the file and close the editor.

27.2.2 Declaring exchange capabilities using WSDL

You must declare the exchange capabilities of this portlet using a WSDL file. Follow these steps to create the WSDL file:

1. From the Project Explorer view, right-click the **EchoSource\WebContent** folder and select **New** → **Folder**.
2. Type `wsdl` for the Folder name field and click **Finish**.
3. The directory structure should look as illustrated in Figure 27-2.

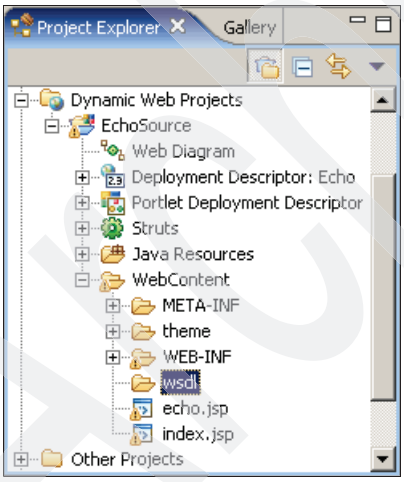


Figure 27-2 New wsdl folder

4. Right-click the wsdl folder just created and choose **Import...** → **File system** from the context menu.
5. Browse the directory where the EchoSource.wsdl file is located and select this file. The WSDL file is provided as additional material of this redbook.

Archived

6. Make sure you check the **Create selected folders only** option and click **Finish**.

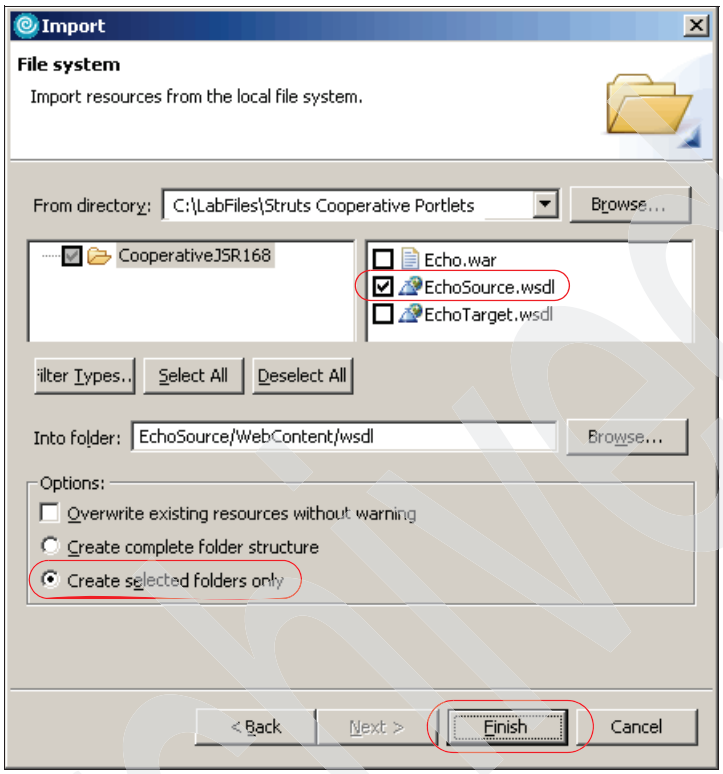


Figure 27-3 Importing wsdl file

7. Example 27-1 illustrates the EchoSource.wsdl file.

Example 27-1 WSDL file for EchoSource portlet

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DeptResults_Service"
  targetNamespace="http://www.ibm.com/wps/c2a/examples/echosource"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/examples/echosource"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<types>
  <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/examples/echosource">
    <xsd:simpleType name="inmsg">
```

```
      <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</types>

<message name="GetResultsMessageNameResponse">
  <part name="input_msg" type="tns:inmsg"/>
</message>

<portType name="Message_Service">
  <operation name="msg_Detail">
     <output message="tns:GetResultsMessageNameResponse"/>
  </operation>
</portType>

<binding name="MessageBinding" type="tns:Message_Service">
  <portlet:binding/>
  <operation name="msg_Detail">
    <portlet:action name="/welcome.do" type="standard-struts"
actionNameParameter="spf_strutsAction" caption="show.message"
description="Get.Message.from.user"/>
    <output>
      <portlet:param name="inmsg" partname="input_msg"
         boundTo="request-parameter" caption="in.message"/>
    </output>
  </operation>
</binding>

</definitions>
```

The syntax used in this file is the standard WSDL with some custom extensions for cooperative portlets. The `<binding>` tag is extended to associate portlet actions with operations. The elements prefixed with `portlet:` are part of the custom WSDL extension schema:

- For each operation:
  - The portlet action name must be provided using the `name` attribute of the `action` tag.
  - The type attribute must be set as `standard-struts` to indicate that it is a struts action used with a JSR portlet. Other type attribute options are `standard` (indicates a standard portlet action, invoked using processAction method), `default` (indicates a DefaultPortletAction object is used), `simple` (indicates a simple portlet action String is used) and `struts` (indicates a Struts action is used).

Archived

– The `actionNameParameter` attribute always must be set as `spf_strutsAction` to indicate the property broker to use a special binding to support Struts actions.

In the above file you see that this portlet declares a single action called `/welcome.do`, using the `portlet:action` element.

```
<portlet:action name="/welcome.do" type="standard-struts"
actionNameParameter="spf_strutsAction" caption="show.message"
description="Get.Message.from.user"/>
```

- For each operation parameter:
  – The `portlet:param` element identifies that the action has an output parameter.
  – The action parameter name must be provided using the `name` attribute of the `param` tag.
  – The `boundTo` attribute may be used to specify where the parameter will be bound, the options are `request-parameter`, `request-attribute`, `session-attribute` or `action-attribute`.
  – Each operation parameter appears as a child element of a `<input>` or `<output>` tag, to specify if it is a parameter consumed or produced by the portlet action, respectively.

In the above file, you see that this portlet has produced a parameter named `inmsg`.

```
<output>
   <portlet:param name="inmsg" partname="input_msg"
boundTo="request-parameter" caption="in.message"/>
</output>
```

27.2.3 Updating the portlet deployment descriptor

You need to update the portlet deployment descriptor to include a reference to the WSDL file using the portlet preference parameter called `com.ibm.portal.propertybroker.wsdllocation`.

1. In the Project Explorer view, expand **EchoSource/WebContent/WEB-INF** and double-click in portlet.xml.
2. Click the **Source** tab and add the code highlighted in Example 27-2.

Example 27-2 Updating portlet deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app
   xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
   version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
   xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
   id="echo.EchoPortlet.0190a47430">
   <portlet>
   .....
      <portlet-preferences>
         <preference>
            <name>com.ibm.struts.portal.page.view.html</name>
            <value>index.jsp</value>
         </preference>
         <preference>
            <name>com.ibm.portal.propertybroker.wsdllocation</name>
            <value>/wsdl/EchoSource.wsdl</value>
         </preference>
      </portlet-preferences>
   </portlet>
</portlet-app>
```

3. Save and close the portlet.xml file.

27.2.4 Updating the EchoSource portlet code

The next steps describe the changes you have to make to the `EchoSource` portlet code:

1. Open the `EchoSource` portlet double-clicking in EchoSource/Java Resources/JavaSource/echo.actions/EchoAction.java from the Project Explorer view.
2. You will set an attribute in the session object to indicate whether the Cooperative Portlets service must be initialized or not. Include the following import statement:

   ```
   import javax.portlet.PortletSession;
   ```

3. In `EchoAction` class, declare two new variables (`pbService` and `pbServiceAvailable`).

Example 27-3 Declaring variables

```
public class EchoAction extends StrutsAction {

   PropertyBrokerService pbService = null;
   boolean pbServiceAvailable = false;
```

4. Modify the `execute` method to obtain a reference to the property broker only the first time the method is invoked. For purposes of this sample, you will use the portlet session to set an attribute the first time the method is invoked and avoid referencing the property broker for successive method invocations.

JSR 168 portlets using cooperation should be programmed so that they perform correctly in other containers other than IBM Portal container. A good practice is to use a boolean variable that indicates if the PropertyBrokerService interface is available. The variable `pbServiceAvailable` will set to true only if the service is available. This variable can be used to guard accesses to IBM-only services and it is used to ensure that the portlet can still function in environments where the service is unavailable.

Example 27-4 Obtaining a reference to the property broker service

```
   public ActionForward execute(ActionMapping mapping, ActionForm form,
PortletRequest request, PortletResponse response) throws Exception {
      PortletSession session = request.getPortletSession(true);
      if (session.getAttribute("INIT")==null) {
         System.out.println("obtaining a reference to the property broker
services ======");
         session.setAttribute("INIT","INIT");
         try {
            Context ctx = new InitialContext();
            PortletServiceHome serviceHome =
(PortletServiceHome)ctx.lookup("portletservice/com.ibm.portal.propertybroker.se
rvice.PropertyBrokerService");
            pbService =
(PropertyBrokerService)serviceHome.getPortletService(com.ibm.portal.propertybro
ker.service.PropertyBrokerService.class);
            pbServiceAvailable = true;
         } catch(Throwable t){
            System.out.println("Echo portlet could not find property broker
service!");
         }
      }
      System.out.println("executing action======");
      UserBean userBean = (UserBean) form;
      if (userBean.getInmsg()== "")
         userBean.setInmsg("No message");
      else
         userBean.setInmsg("Message received: " + userBean.getInmsg());
      ActionForward forward = mapping.findForward("result");
      return forward;
   }
```

5. Save the EchoSource.java file.

Archived

27.2.5 Internationalization

To support translation of captions and descriptions associated with shared properties or actions you need to provide resource bundles in the appropriate location in the WAR file. These captions and descriptions are shown when you create wires using the Portlet Wiring Tool. For JSR 168 compliant portlets, the resource file is specified using the `resource-bundle` parameter located in the portlet deployment descriptor (portlet.xml file), for example:

```
<resource-bundle>echo.nl.EchoPortletResource</resource-bundle>
```

Caption and description attributes of the `portlet:action` and `portlet:param` elements in WSDL file specify the name of the key in the resource bundles where the value of these attributes is to be retrieved. For example:

```
<portlet:action name="/welcome.do" type="standard-struts"
actionNameParameter="spf_strutsAction" caption="show.message"
description="Get.Message.from.user"/>
<output>
   <portlet:param name="inmsg" partname="input_msg"
boundTo="request-parameter" caption="in.message"/>
</output>
```

The values of these attributes are obtained from the appropriate resource bundle (EchoPortletResource properties file), located in the EchoSource/Java Resources/JavaSource/echo.nl directory. For example:

Example 27-5 EchoPortletResource.properties.

```
#
# PortletInfo Resource Bundle
#
javax.portlet.title=Echo portlet Source
javax.portlet.short-title=
javax.portlet.keywords=

show.message=Show message
in.message=Input message
Get.Message.from.user=Get message from user
```

27.3 Target cooperative portlet

In this section, you will import a second copy of the `Echo` portlet and update it to support cooperation as a target portlet.

Archive

27.3.1 Importing the Echo portlet

Follow these steps to import the `Echo` portlet:

1. Select **File** → **Import**. In the import panel, select the WAR file and click **Next**.
2. In the next window, enter the following information:
   a. WAR file: browse to C:\LabFiles\Struts Cooperative Portlets\Echo.war
   b. Web project: enter `EchoTarget`
   c. Target server: select **WebSphere Portal V5.1**
   d. Select **Add module to an EAR project**
   e. EAR project: `EchoTargetEAR`
3. Click **Finish**.

The `EchoSource` and `EchoTarget` portlet applications use the same UID so it will cause an error executing the portlets. To fix this problem expand EchoTarget/WebContent/WEB-INF folder and open the portlet deployment descriptor file (portlet.xml). In the portlet deployment descriptor editor, select **Source** panel, change the last digit of the UID for this portlet application and save your changes.

For example, in this sample scenario, the last digit was 0 and it was changed to 1, as highlighted in Example 27-6.

Example 27-6 Changing a digit in portlet application ID

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app
   xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
   version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
   id="echo.EchoPortlet.0190a47431">
   <portlet>
   .....
   </portlet>
</portlet-app>
```

It is recommended that you change the portlet title to identify each portlet when they will be on the same page. Follow these steps for each resource bundle:

1. Open the resource bundles (EchoPortletResource properties file), located in the EchoTarget/Java Resources/JavaSource/echo.nl directory.
2. Add a title in the line `javax.portlet.title`. For example:

```
javax.portlet.title=Echo Target portlet
```

3. Save the file.

27.3.2 Declaring exchange capabilities using WSDL

You must declare the exchange capabilities of this portlet using a WSDL file. Follow these steps to create the WSDL file:

1. From the Project Explorer view, right-click the **EchoTarget\WebContent** folder and select **New** → **Folder**.
2. Type `wsdl` for the Folder name field and click **Finish**.
3. The directory structure should look as illustrated in Figure 27-4.

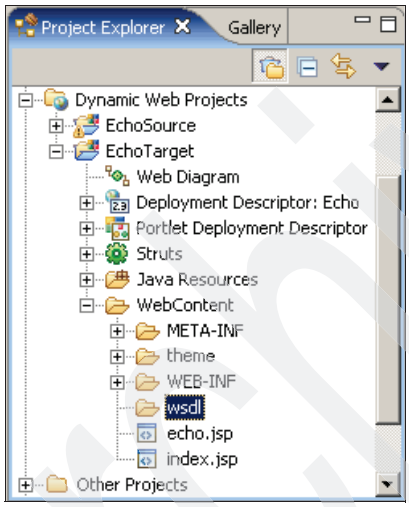


Figure 27-4 New wsdl folder

4. Right-click the wsdl folder just created and choose **Import...** → **File system** from the context menu.
5. Browse the directory where the EchoTarget.wsdl file is located and select this file. The WSDL file is provided as additional material of this redbook.

   Be sure to check the **Create selected folders only** option and click **Finish**.

Archived

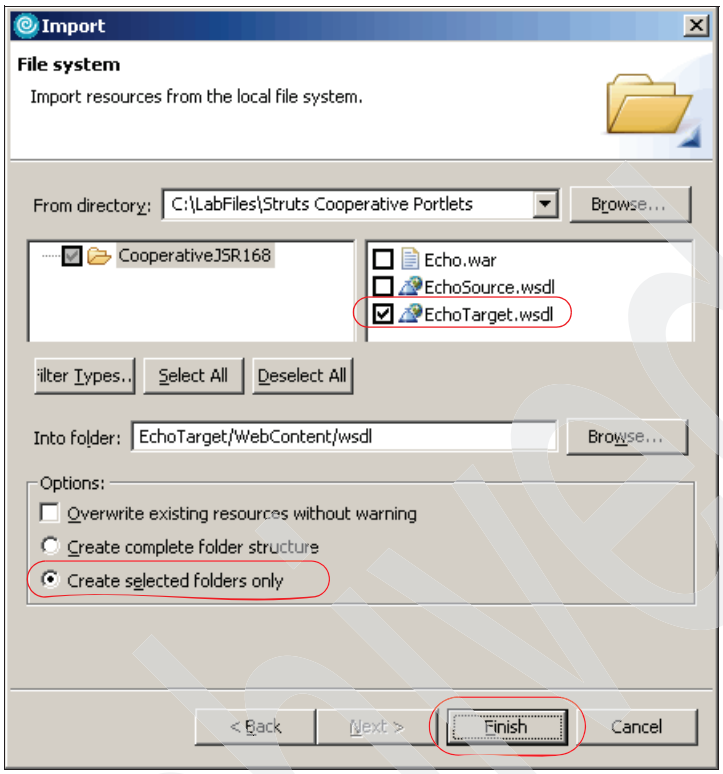


Figure 27-5 Importing wsdl file

6. Example 27-7 illustrates the EchoTarget.wsdl file.

Example 27-7 EchoTarget.wsdl file

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DeptResults_Service"
  targetNamespace="http://www.ibm.com/wps/c2a/examples/echosource"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/examples/echosource"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<types>
  <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/examples/echosource">
    <xsd:simpleType name="inmsg">
      <xsd:restriction base="xsd:string">
      </xsd:restriction>
```

```
    </xsd:simpleType>
  </xsd:schema>
</types>

<message name="GetResultsMessageNameRequest">
  <part name="input_msg" type="tns:inmsg"/>
</message>

<portType name="Message_Service">
  <operation name="msg_Detail">
     <input message="tns:GetResultsMessageNameRequest"/>
  </operation>
</portType>

<binding
    name="MessageBinding"
      type="tns:Message_Service">
  <portlet:binding/>
  <operation name="msg_Detail">
    <portlet:action name="/welcome.do" type="standard-struts"
actionNameParameter="spf_strutsAction" caption="show.message"
description="Get.Message.from.user"/>
    <input>
      <portlet:param name="inmsg" partname="input_msg" caption="in.message"/>
    </input>
  </operation>
</binding>
</definitions>
```

7. Update the portlet.xml to include a reference of WSDL file, as highlighted in Example 27-8.

Example 27-8 Updating portlet deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app
   xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
   version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
   id="echo.EchoPortlet.0190a47431">
   <portlet>
   .....
      <portlet-preferences>
         <preference>
            <name>com.ibm.struts.portal.page.view.html</name>
            <value>index.jsp</value>
         </preference>
         <preference>
```


Archived

```
            <name>com.ibm.portal.propertybroker.wsdllocation</name>
            <value>/wsdl/EchoTarget.wsdl</value>
         </preference>
      </portlet-preferences>
   </portlet>
</portlet-app>
```

8. Save and close the portlet.xml file.

27.3.3 Updating the EchoTarget portlet code

The next steps show the changes you have to make in `EchoTarget` portlet code:

1. From the Project Explorer view, open the `EchoTarget` portlet located in `/EchoTarget/Java Resources/JavaSource/echo.actions/EchoAction.java`
2. Be sure to include the following import statement:

   ```
   import javax.portlet.PortletSession;
   ```

3. In `EchoAction` class, declare two new variables (`pbService` and `pbServiceAvailable`).

Example 27-9 Declaring variables.

```
public class EchoAction extends StrutsAction {

   PropertyBrokerService pbService = null;
   boolean pbServiceAvailable = false;
```

4. Modify the `execute` method to obtain a reference to the property broker only the first time the method is invoked. For purposes of this sample, we use the portlet session to set an attribute the first time the method is invoked and avoid to reference the property broker for successive method invocations.

   JSR 168 portlets using cooperation should be programmed so that they perform correctly in other containers than IBM container. A good practice is to use a boolean variable that indicates if the PropertyBrokerService interface is available. The variable `pbServiceAvailable` will set to true only if the service is available. This variable can be used to guard accesses to IBM-only services and it is used to ensure that the portlet can still function in environments where the service is unavailable.

Example 27-10 Obtaining a reference to the property broker service

```
   public ActionForward execute(ActionMapping mapping, ActionForm form,
PortletRequest request, PortletResponse response) throws Exception {
      PortletSession session = request.getPortletSession(true);
      if (session.getAttribute("INIT")==null) {
```

Archived

```
         System.out.println("obtaining a reference to the property broker
services ======");
         session.setAttribute("INIT","INIT");
         try {
            Context ctx = new InitialContext();
            PortletServiceHome serviceHome =
(PortletServiceHome)ctx.lookup("portletservice/com.ibm.portal.propertybroker.se
rvice.PropertyBrokerService");
            pbService =
(PropertyBrokerService)serviceHome.getPortletService(com.ibm.portal.propertybro
ker.service.PropertyBrokerService.class);
            pbServiceAvailable = true;
         } catch(Throwable t){
            System.out.println("Echo portlet could not find property broker
service!");
         }
      }
      System.out.println("executing action======");
      UserBean userBean = (UserBean) form;
      if (userBean.getInmsg()== "")
         userBean.setInmsg("No message");
      else
         userBean.setInmsg("Message received: " + userBean.getInmsg());
      ActionForward forward = mapping.findForward("result");
      return forward;
   }
```

5. Save the EchoSource.java file.

27.3.4 Internationalization

To support translation of the captions and descriptions associated with shared properties or actions you need to provide resource bundles in the appropriate location in the WAR file. These captions and descriptions are shown when you create wires using the Portlet Wiring Tool. For JSR 168 compliant portlets, the resource file is specified using the `resource-bundle` parameter located in the portlet deployment descriptor (portlet.xml file), for example:

```
<resource-bundle>echo.nl.EchoPortletResource</resource-bundle>
```

Caption and description attributes of the `portlet:action` and `portlet:param` elements in WSDL file specify the name of the key in the resource bundles where the value of these attributes is to be retrieved. For example:

```
<portlet:action name="/welcome.do" type="standard-struts"
actionNameParameter="spf_strutsAction" caption="show.message"
description="Get.Message.from.user"/>
<input>
```


Archived

```
   <portlet:param name="inmsg" partname="input_msg" caption="in.message"/>
</input>
```

The values of these attributes are obtained from the appropriate resource bundle (EchoPortletResource properties file), located in `EchoTarget/Java Resources/JavaSource/echo.nl` directory. For example:

Example 27-11 EchoPortletResource.properties

```
#
# PortletInfo Resource Bundle
#
javax.portlet.title=Echo portlet Source
javax.portlet.short-title=
javax.portlet.keywords=

show.message=Show message
in.message=Input message
Get.Message.from.user=Get message from user
```

27.4 Running the cooperative portlets

Execute the following steps to run the cooperative portlets scenario:

1. Add the portlet projects to the WebSphere Portal V5.1 Test Environment server:
   a. Right-click the server and select **Add and remove projects**.
   b. From the list of available projects, select **EchoSource** and click the **Add >** button to add this project to the list of configured projects on the right side. Repeat the same steps to add the EchoTarget project.
   c. Click **Finish**.
2. In the Servers section, double-click the **WebSphere Portal V5.1 Test Environment**.

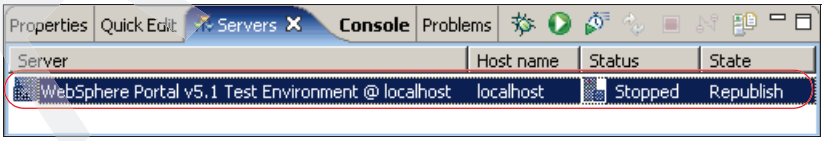


Figure 27-6 WebSphere Portal V5.1 Test Environment

3. In Portal Options, do the following:

Archived

– Select the option **Enable base portlets for portal administration and customization**. This option is necessary to create wires.

– Uncheck the option **Enable multiple pages when deploying portlet projects**. This allows you to deploy both the `EchoSource` and the `EchoTarget` portlets on the same page.

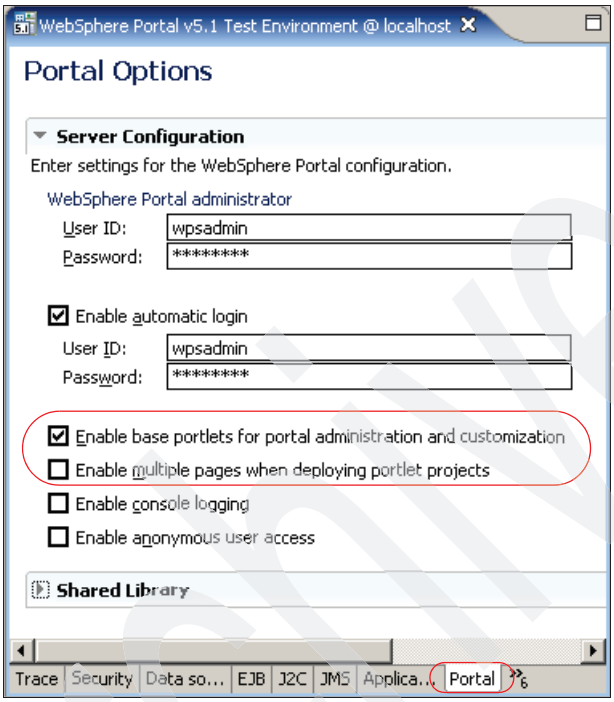


Figure 27-7 Portal options in WebSphere Portal V5.1 Test Environment

4. Save the changes.
5. From the Project Explorer view, right-click **EchoSource** project. Then click **Run → Run on server,** select **WebSphere Portal V5.1 Test Environment** server and click **Finish**. This will load your project into the test environment so that you can view it in Rational Application Developer Web browser. It may take few minutes for this process to complete.
6. Now you will see the two portlets in the browser. Click **Edit Page** and select the **Wires** tab to make the cooperation happen at runtime. You will see different combos to select a source portlet, a property to send, a target portlet and a property or action to receive. If you select **Global wire** that means that their effects are manifested to all users that can view the page and portlets.
7. Select the following values:

Archived

- Source Portlet: `Echo Source portlet`
- Sending: `Input message`
- Target Portlet: `Echo Target portlet`
- Receiving: `Show message, Input message`.
- Check **Global wire**
- Click **Add wire**

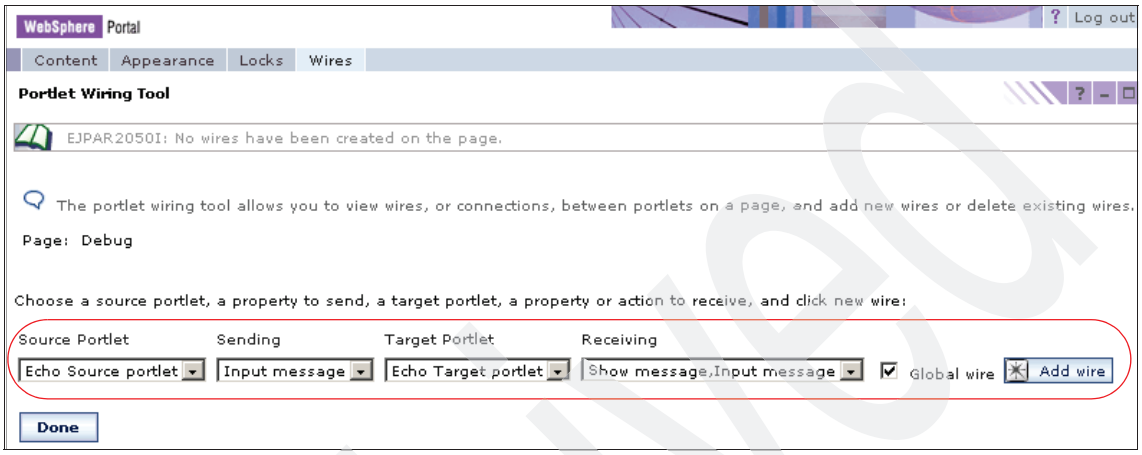


Figure 27-8 Portlet Wiring Tool

8. You will see the new wire as shown in Figure 27-9 on page 854.

Archived

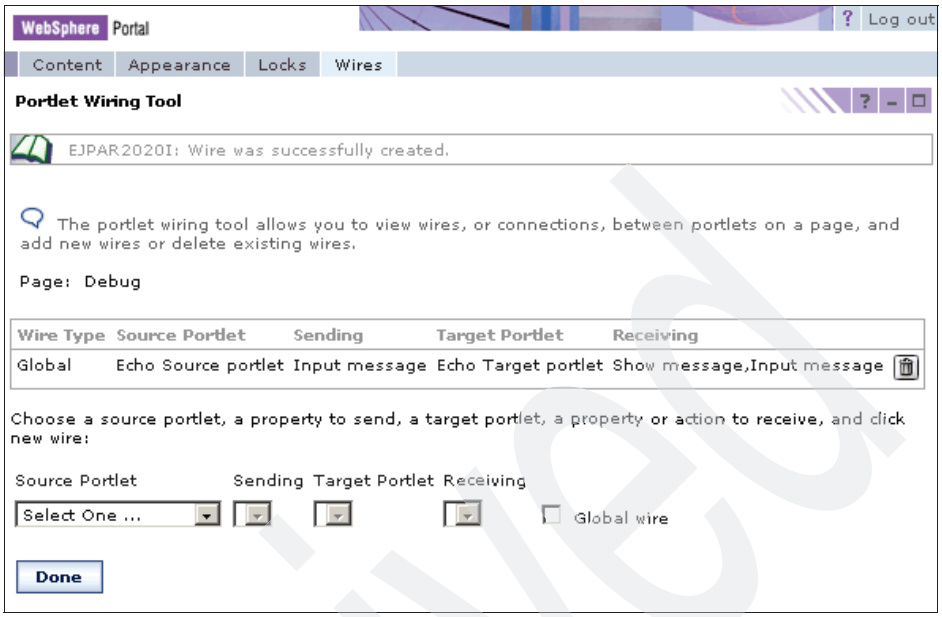


Figure 27-9 New wire

9. Click **Done**.
10. Enter a message in the `Echo Source` portlet, for example: `Testing cooperative portlets`.
11. Click **Submit**.
12. The message will be echo back in the source portlet and sent to the `Echo Target` portlet.

Archived

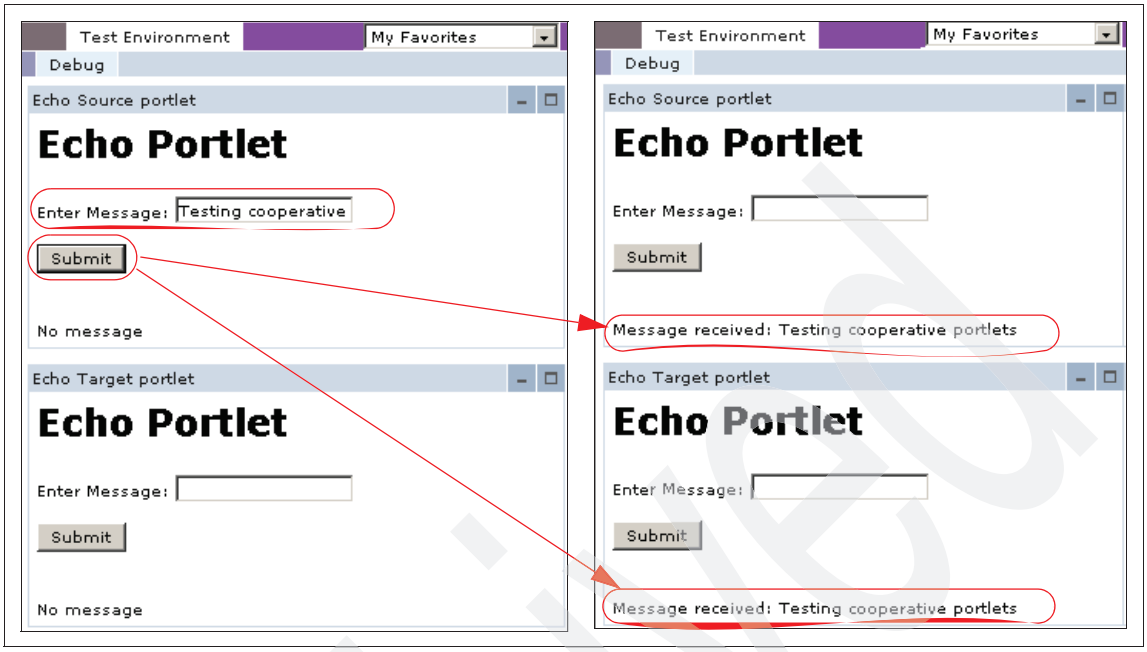


Figure 27-10 Sending a message

Archived

Archived

28

Accessing Web Services from portlet applications

This chapter contains an overview and a sample scenario of how to create a sample portlet project that will work as a Web Service client that interacts with a Web Service. The Web Service client portlet is created using the wizard provided by the Rational Application Developer V6. The sample scenario in this chapter will allow you to understand the technique used to develop portlets that retrieve information using Web Services

© Copyright IBM Corp. 2005. All rights reserved.

28.1 Overview

There are two things that need to be done in order to access a Web Service through a portlet. The first is have access to a Web Service either through UDDI registry, WSDL URL or running in the Rational Application Developer workspace. And the second is create a portlet that provides access to this Web Service. In Rational Application Developer V6 Web Service Client portlet applications are developed within Faces portlet projects and a wizard is provided to adding Web Services to Faces portlet pages.

Figure 28-1 shows how portlet applications can be easily integrated with existent Web Services without the need to write extra code.

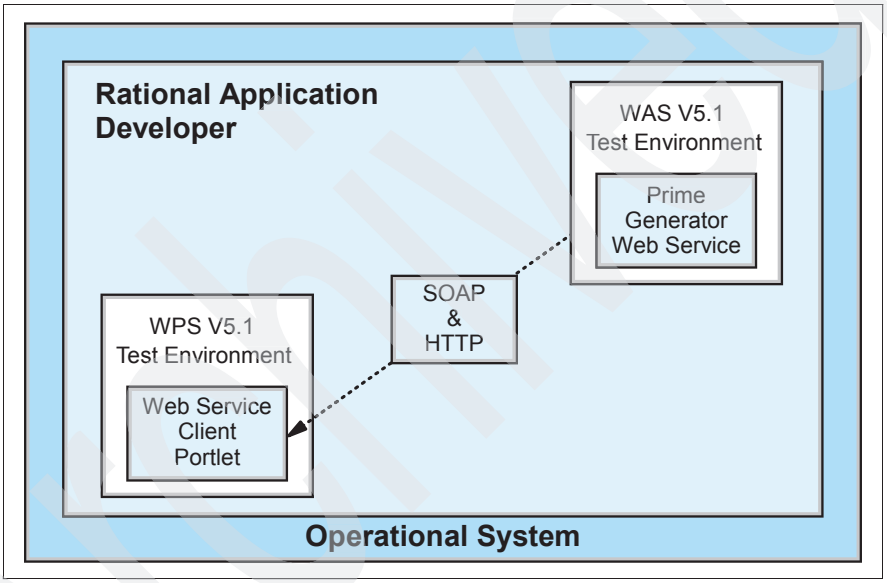


Figure 28-1 Accessing Web Services from portlet applications

28.2 Sample scenario

In this section, the process to develop a Web Service client portlet accessing a local Web Service is describe. The following tasks will be performed:

1. Develop a sample Web Service from a JavaBean. This Web Service will be used to test and run the Web Service client portlet. The following tasks are required:
   a. Create a Dynamic Web project and import an existing JavaBean class.

b. Use the available wizards in Rational Application Developer V6, to transform this JavaBean into a Web Service so it can be accessed from portlet applications.

2. Using the wizards provided by Rational Application Developer V6, create a Web Services client portlet project to access the sample Web Service.

Note: This sample scenario requires of a Web Services be available. A java bean sample code can be downloaded from additional materials. Also, the generated Web Service and a test client are available. See Appendix A, “Additional material” on page 1003.

28.2.1 Creating a Web Service

In this section are described the two process needed to create the Web Service. Also, a TestClient is created in order to test the Web Service and to make sure the Web Service is working before create the portlet client.

Creating a Dynamic Web Project

1. If not already running, start the Rational Application Developer V6; click **Start → Programs → IBM Rational → IBM Rational Application Developer V6.**
2. Select **File → New → Dynamic Web Project.**

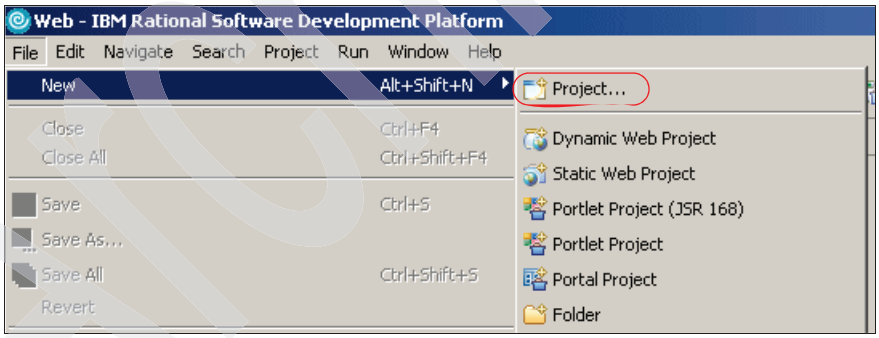


Figure 28-2 Dynamic Web project

3. In the Dynamic Web Project window, click the **Show Advanced>>** button.
4. Enter the following values:
   - Name: `PrimesWebService`
   - Servlet version: `2.3`
   - Target Server: `Websphere Application Server V5.1`

   Leave the other values as default.

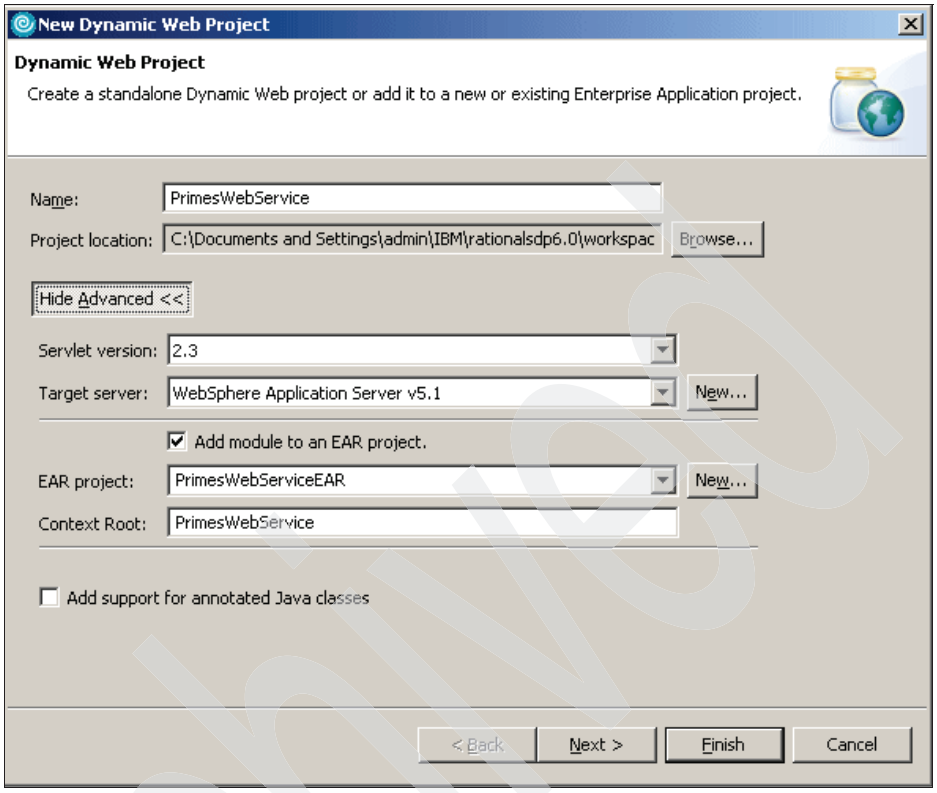


Figure 28-3 Dynamic Web Project window

5. Click **Next.**
6. In the Features window, deselect the **Web Diagram**.

Archive

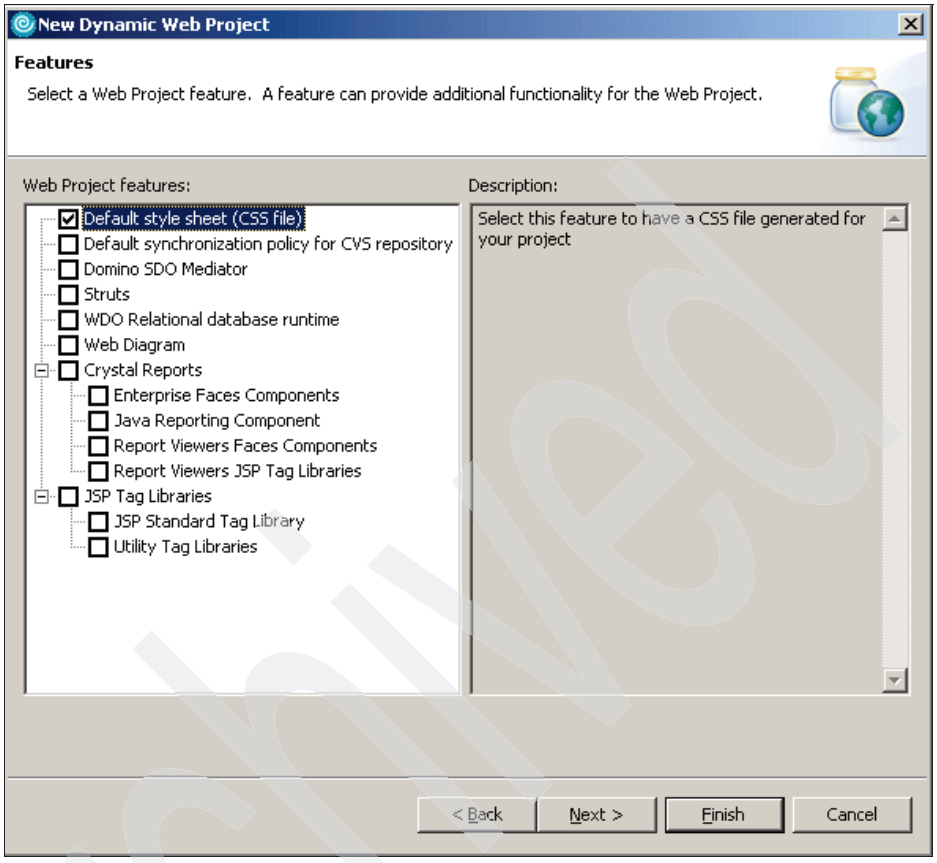


Figure 28-4 Features window

7. Click **Next**.
8. In the Select a Page Template for the Web Site window, check the **Use a default Page Template for the Web site** box.
9. Select **A_blue.htpl**.

Archived

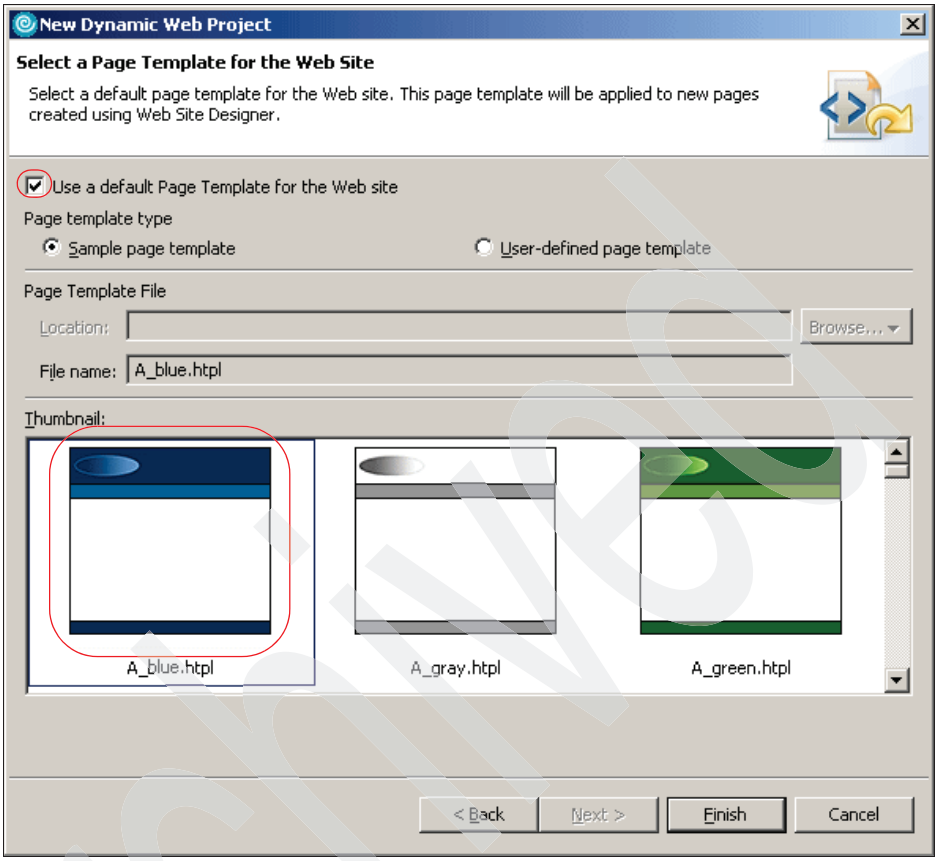


Figure 28-5 Select a Page Template for the Web Site window

10. Click **Finish**.

Creating a Package for the Java Bean class

1. In the Project Explorer, expand the **PrimesWebService** under Dynamics Web Project.
2. Expand the **JavaResources** folder.
3. Right-click the **JavaSource** folder.
4. Select **New** → **Package.**
5. In the Java Package window, enter **com.itso** as the Name value.
6. Click **Finish**.

Archive

Importing the JavaBean

The Primes java bean is a java class that contains a getPrime() method. This method receive a number of digits and returns a prime number with the specified length of digits. This is the method that concerns this scenario.

When executing this method, try generating prime numbers of 20 or fewer digits to avoid long computations. A short version of the prime number generator is shown in Example 28-1.

Example 28-1 Primes.java

```
package com.itso;
import java.math.BigInteger;

public class Primes {

   private static final BigInteger ZERO = new BigInteger("0");
   private static final BigInteger ONE = new BigInteger("1");
   private static final BigInteger TWO = new BigInteger("2");
   private String prime = "";

   ..........
   ..........
   ..........

   public  String getPrime(int numDigits) {

      BigInteger start = random(numDigits);
      start = nextPrime(start);

      return start.toString();
   }
```

To review this class, once imported, double-click **Primes.java** under PrimesWebService/JavaResources/JavaSource/com/itso.

1. Right-click the **com.itso** new package under the PrimesWebService/JavaResources/JavaSource/ in the ProjectExplorer view.
2. From the context menu, select **Import...**

Archived

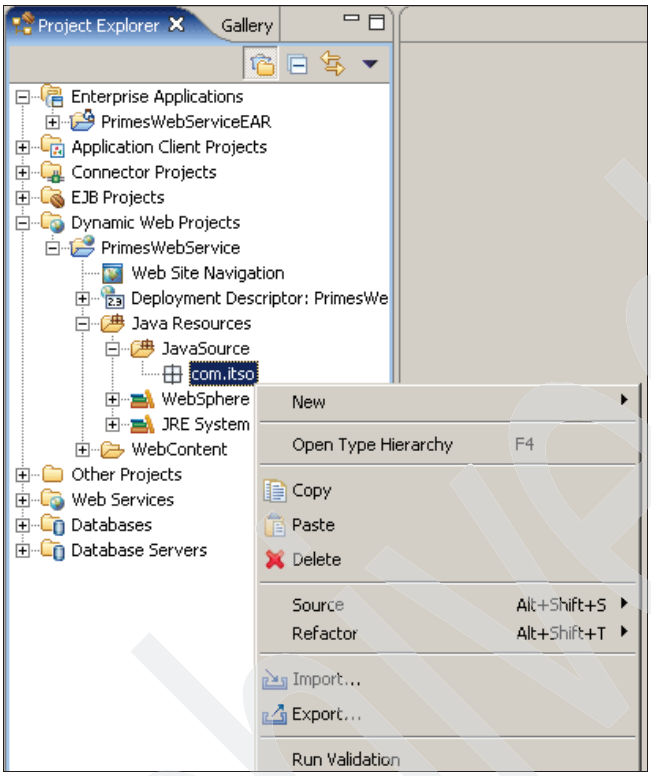


Figure 28-6 Right-click new package

3. In the Select window, select **File System** from the Select an import source field.
4. Click **Next**.
5. In the File System window, click the **Browse...** button right next to the From directory field and select the directory where download the Primes.java file.
6. Select the directory on the left column; this will show the files contained in that directory.
7. Select the **Primes.java** file on the right column, by checking the box next to it.
8. Make sure the `PrimesWebService/JavaSource/com/itso` value is already entered in the Into folder field. If not, click the **Browse...** button right next to this field and select this value.

Archive

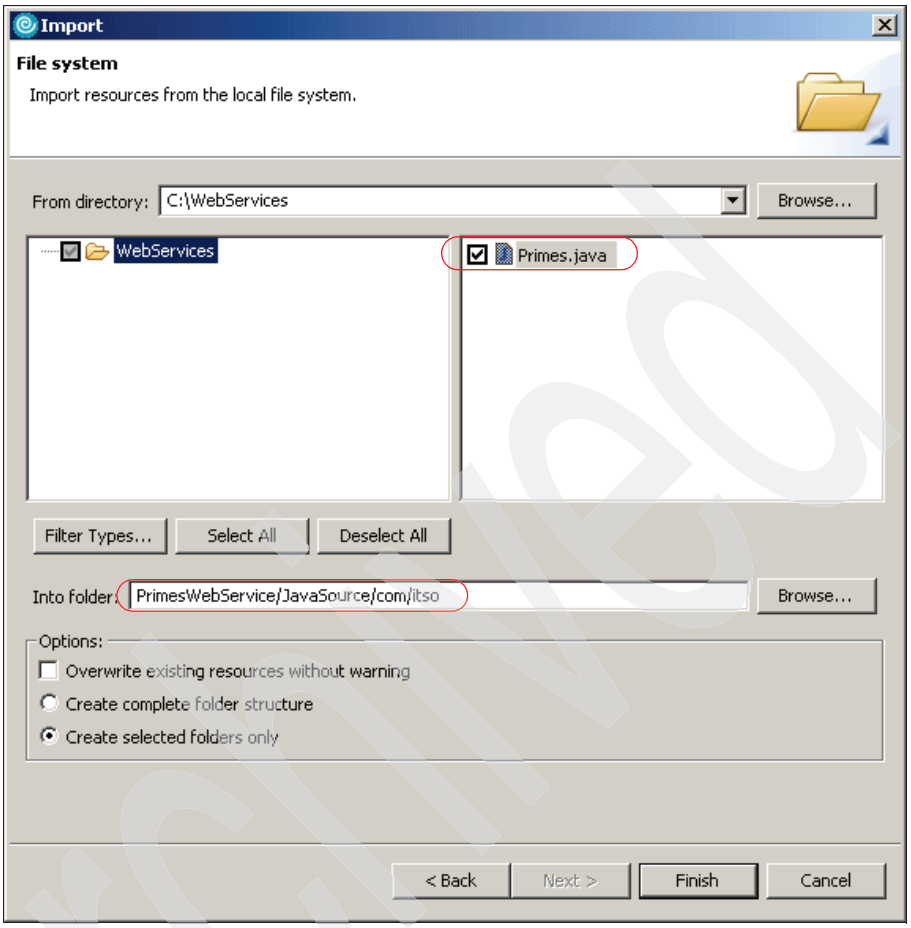


Figure 28-7 File System import window

9. Click **Finish**.

Creating the Web Service and the TestClient

In this section, a Web Service is created in the PrimesWebService project that was created in the previous section.The wizards provided by Rational Application Developer V6 will be used to do this. Once created, the Web Service will be published and invoked to be executed by the PrimesWebServiceClient in a WebSphere Application Server V5.1 Test Environment.

1. From the main menu, select **File** → **New** → **Others...**
2. In the Select Wizard window, scroll down the Wizards field until you see the Web Service folder.

Archive

3. Expand the **Web Services** folder.
4. Select the **Web Services** wizard.

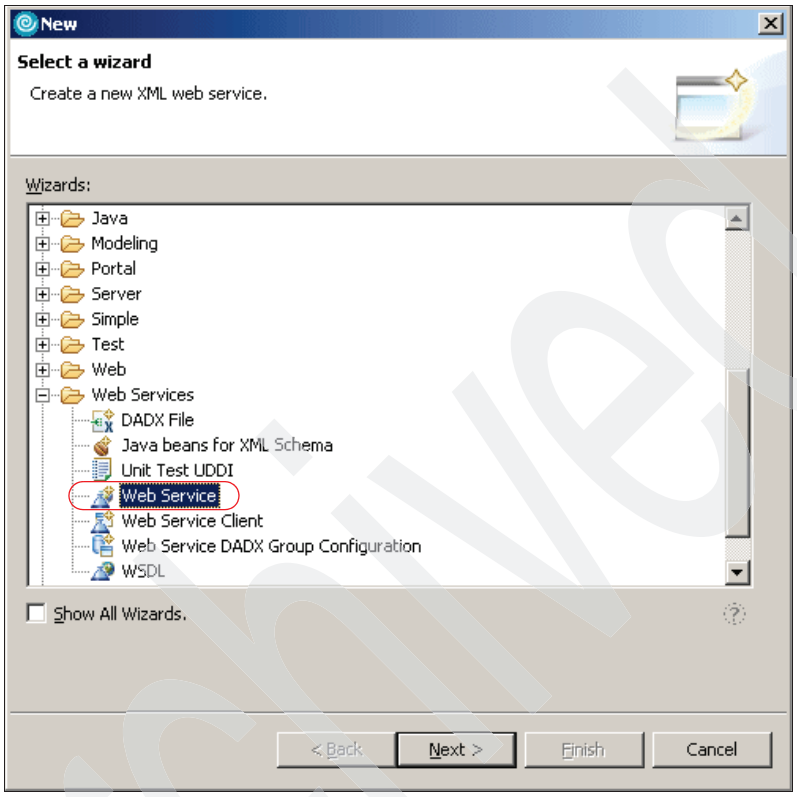


Figure 28-8 Select Web Service wizard

5. Click **Next**.
6. In the Web Services window, enter the following values:
   - Web Service type: `Java Bean Web Service`.
   - Check the **Start Web service in Web Project** box.
   - Check the **Generate a Proxy** box.
   - Client proxy type: Java proxy.
   - Check the **Test the Web service** box.

   Leave the rest of the values as default.

Archived

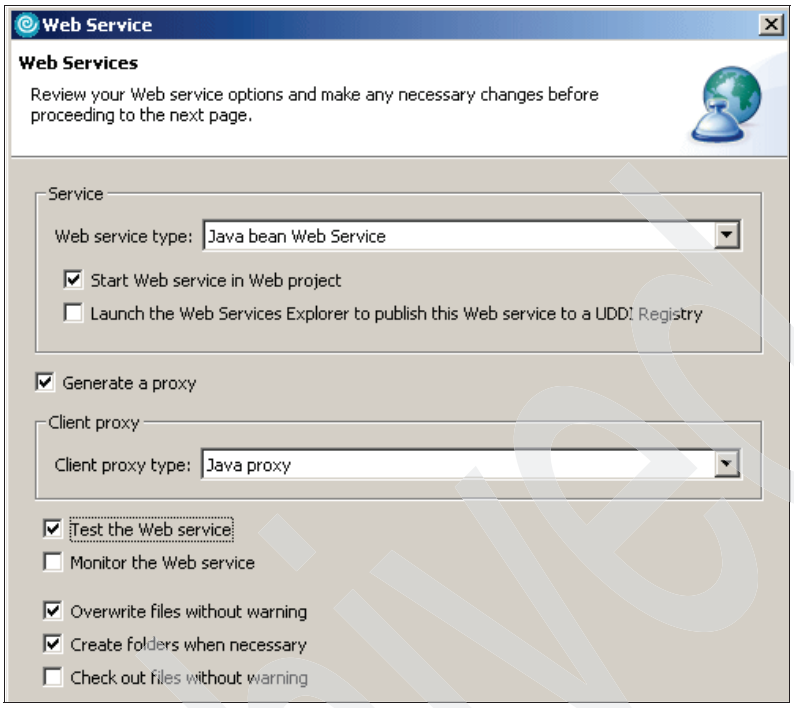


Figure 28-9 Web Services window

7. Click **Next**.
8. In the Object Selection Page, click the **Browse files...** button.

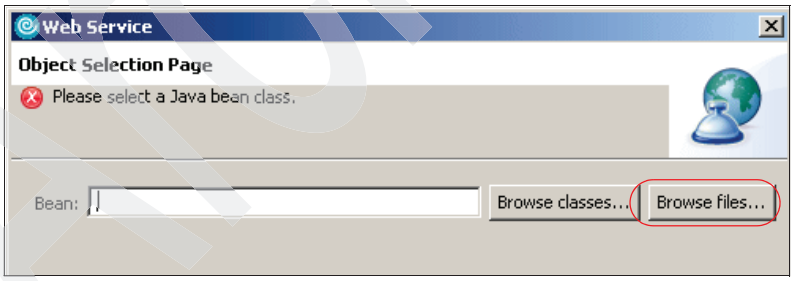


Figure 28-10 Object Selection page

9. Select **Primes.java** from PrimesWebService/JavaSource/com/itso.

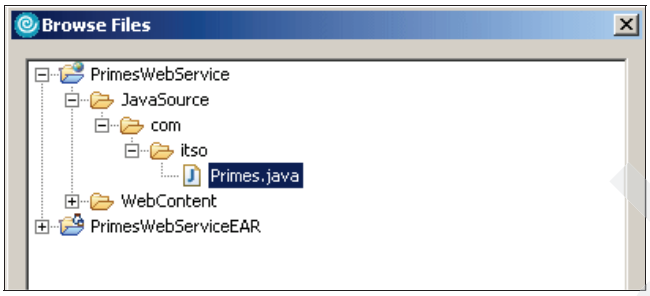


Figure 28-11 Select Primes.java class

10. Click **OK**.
11. Verify the value in the Bean field in the Object Selection Page window.



Figure 28-12 Bean value

12. Click **Next**.
13. In the Service Deployment Configuration window, accept the default values:
    a. Server-Side Deployment Selection:
       - **Web service runtime:** `IBM WebSphere`
       - **Server**: `WebSphere V5.1 Test Environment @ localhost`
       - **J2EE version**: `1.3`
       - **Service project**: `PrimesWebService`
       - **EAR project**: `PrimesWebServiceEAR`
    b. Client-Side Environment Selection:
       - **Web service runtime**: `IBM WebSphere`
       - **Server**: `WebSphere V5.1 Test Environment @ localhost`
       - **J2EE version**: `1.3`
       - **Client type**: `Web`
       - **Client project**: `PrimesWebServiceClient`
       - **EAR project**: `PrimesWebServiceClientEAR`

Archived

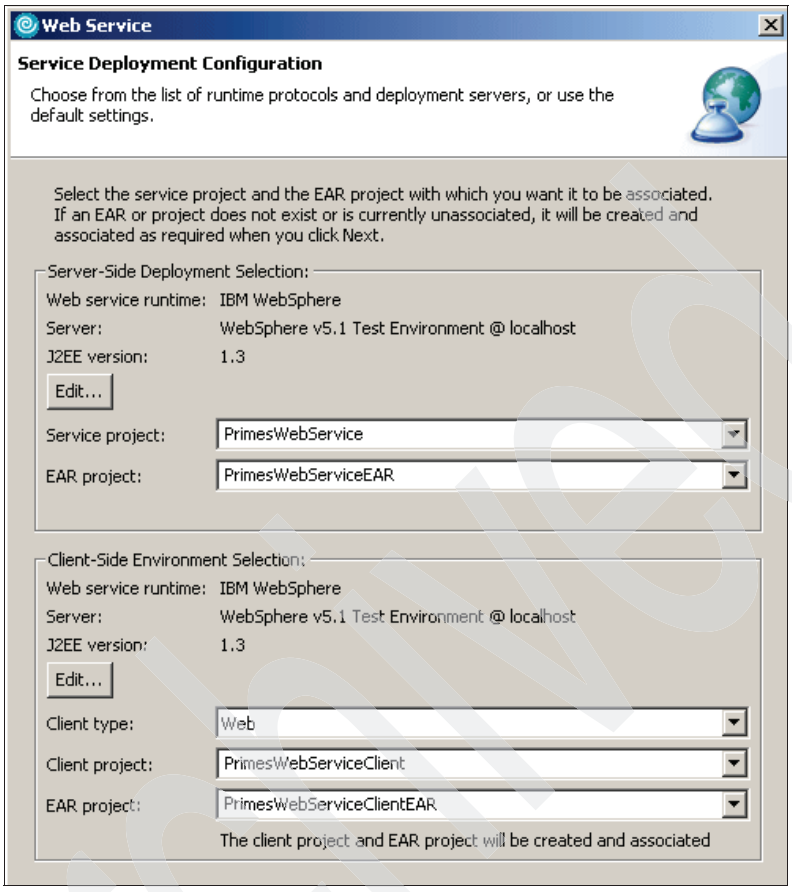


Figure 28-13 Service Deployment Configuration window

14. Click **Next**.
15. Click **Next** in the Service Endpoint Interface Selection window.
16. In the Web Service Java Bean Identity window, deselect the all the methods except the **getPrime(int)** method. Leave the rest as default.

Archived

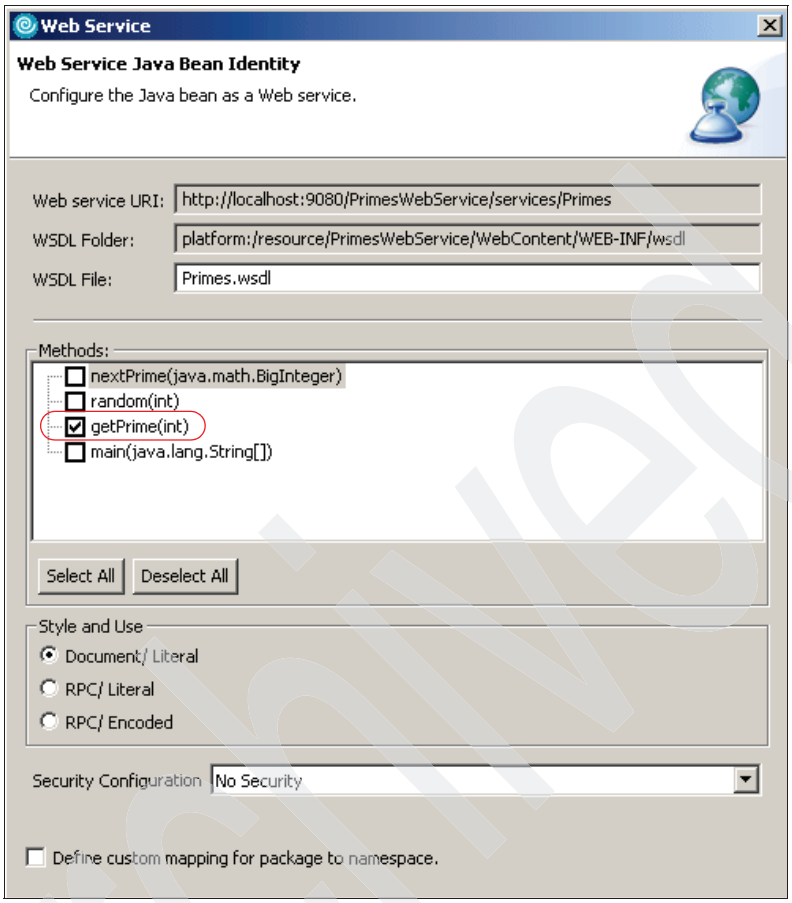


Figure 28-14 Select getPrimes(int)

17. Click **Next**.
18. Leave defaults in Web Service Test Page window, click **Next**.
19. Leave defaults in Web Service Proxy Page window.

Archived

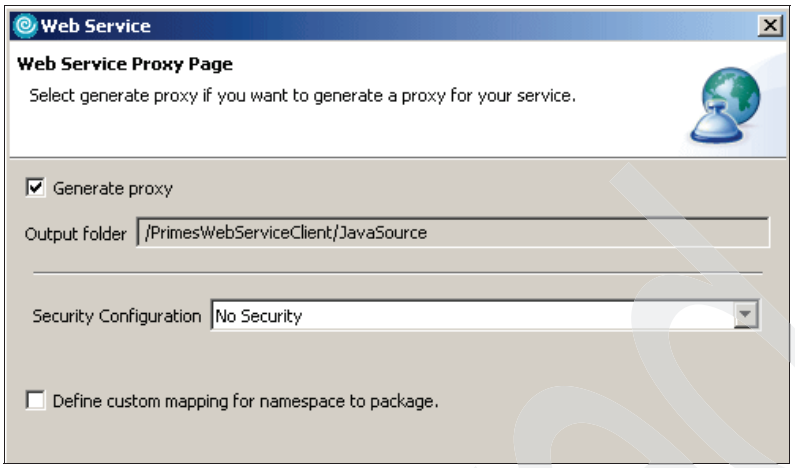


Figure 28-15 Web Service Proxy Page window

20. Click **Next**.
21. In the Web Service Client Test window, deselect all the methods except the **getPrime(int)** method. Leave the rest values as default.

Archived

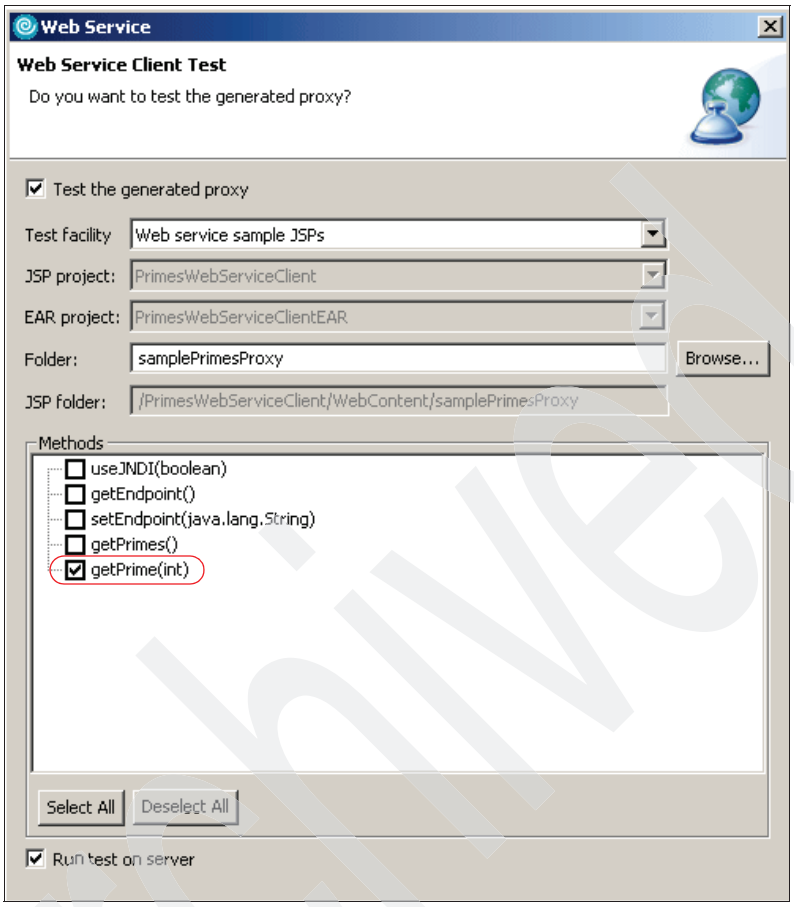


Figure 28-16 Web Service Client Test window

22. Click **Next**.

23. In the Web Service Publication window, leave the default values.

Archive

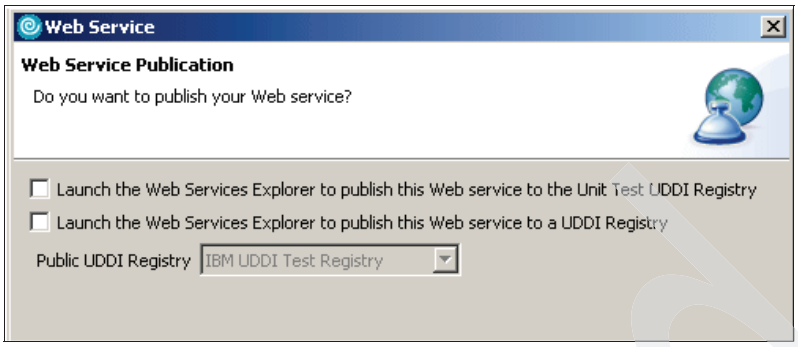


Figure 28-17 Web Service Publication window

24. Click **Finish**.

25. A Web browser will open within the Rational Application Server workspace as shown in Figure 28-18.

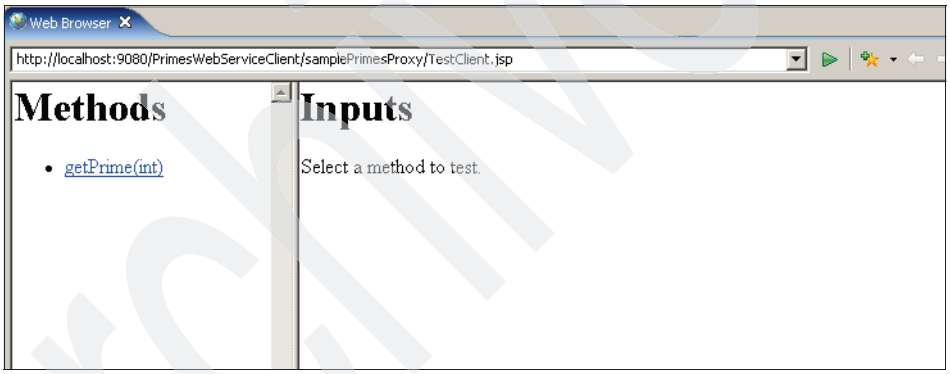


Figure 28-18 PrimesWebServiceClient

26. Click the **getPrime(int)** method link.

Archived

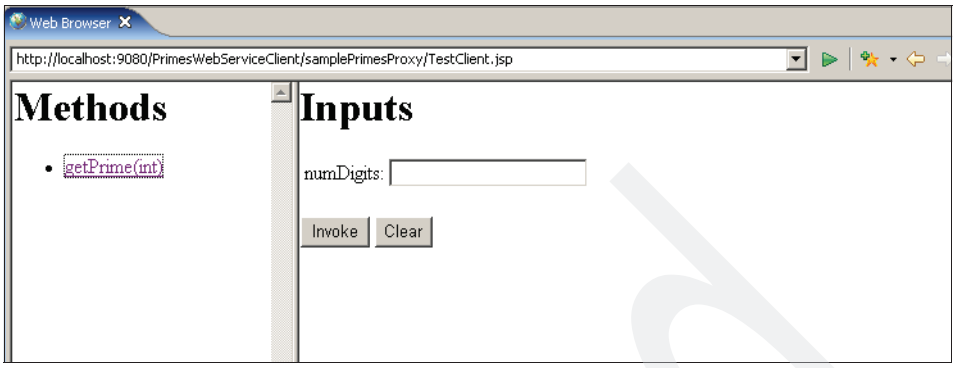


Figure 28-19 getPrime(int) method

27. Enter a value in the numDigits field and click the **Invoke** button. This will show the result as illustrated in Figure 28-20.

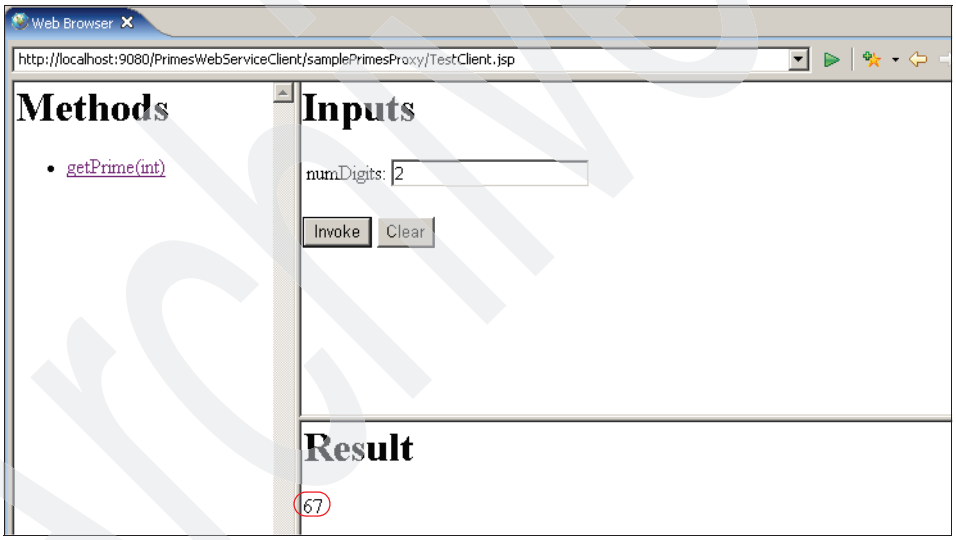


Figure 28-20 Result od invoke getPrime(int) method

Every time you invoke this Web Service to generate a prime number, you will probably get a different result.

28.2.2 Creating a Web Services client portlet

Once the Web Service is ready, a Web Services client portlet to access the prime number generator Web Service will be created. Web Service Client portlet

Archived

applications are developed within Faces portlet projects. Rational Application Developer provides tool support for adding Web Services to Faces portlet pages.

Creating a Portlet Project

1. From the main menu, select **File** → **New** → **Portlet Project.**

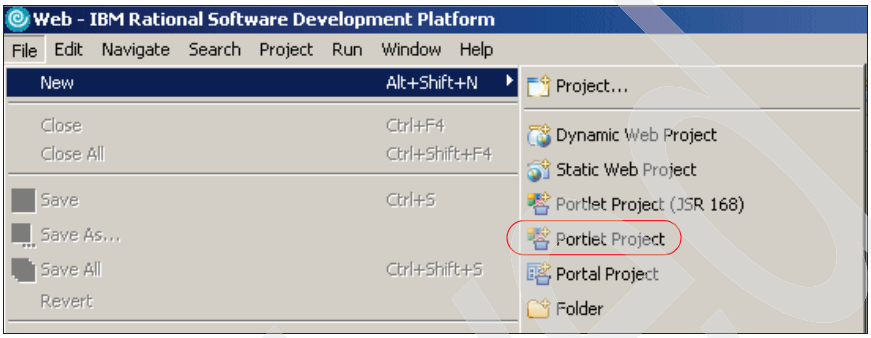


Figure 28-21 Create a new Portlet project

2. In the Portlet Project window, click the **Show Advanced >>** button.
3. Enter the following values:
   - **Name**: `WSClientPortlet`
   - **WebSphere Portal version**: `5.1`
   - **Servlet version**: `2.3`
   - **Target server**: `WebSphere Portal V5.1 stub`
   - **EAR project**: `WSClientPortletEAR`
   - **Context Root**: `WSClientPortlet`

Archive

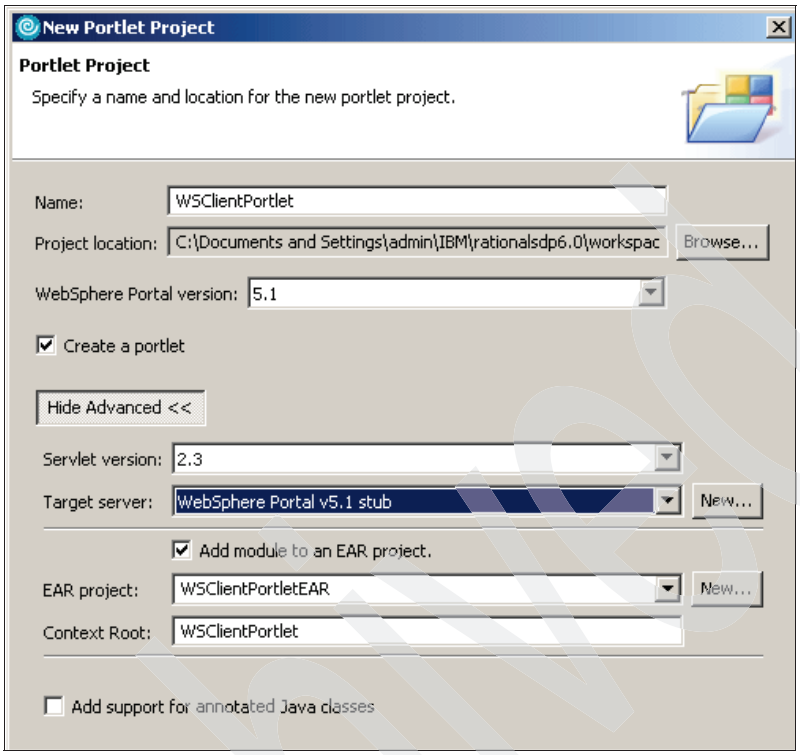


Figure 28-22 Project Portlet window

4. Click **Next**.
5. In the Portlet Type window, select **Faces portlet**.

Archive

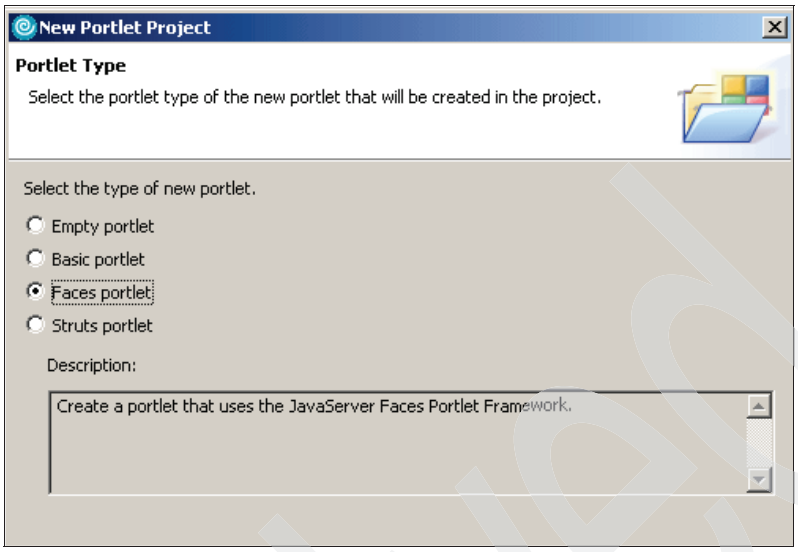


Figure 28-23 Portlet Type window

6. Click **Next**.
7. In the Features window, deselect **Web Diagram**.

Archived

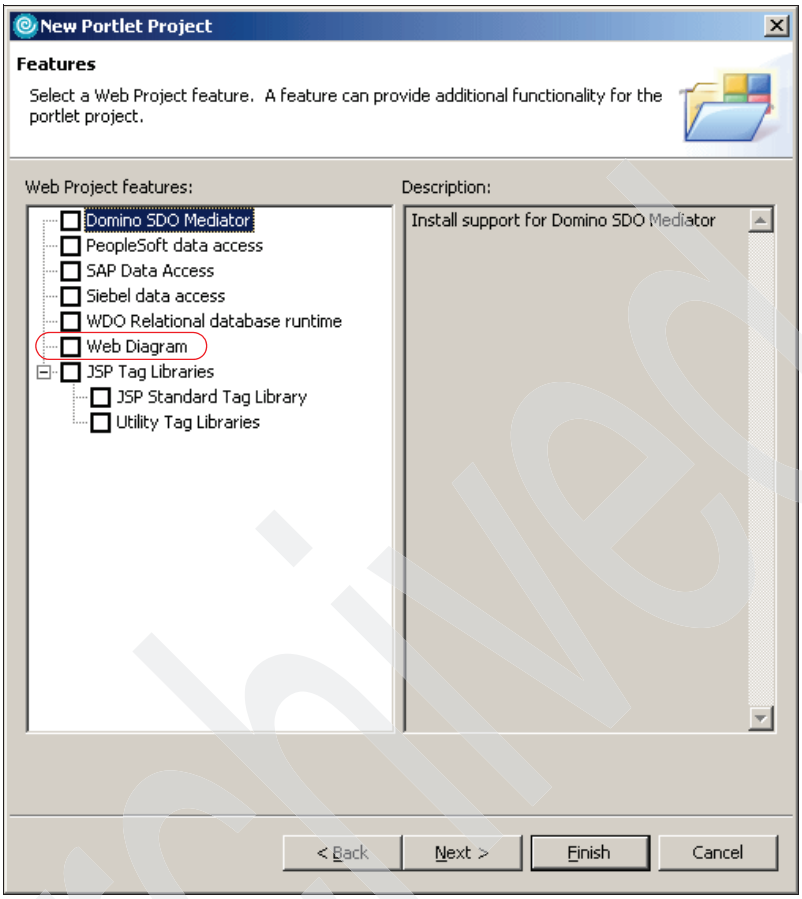


Figure 28-24 Features window

8. Click **Next**.
9. In the Portlet Settings window, leave defaults.
10. Click **Next**.
11. In the Miscellaneous window, leave defaults
12. Click **Finish**.
13. The WSClientPortletView.jsp will open in the JSP Editor, from the palette select the **Web Service** item under the Data drawer.

Archived

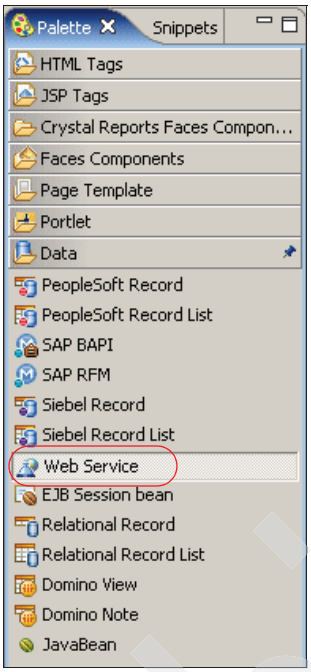


Figure 28-25 Select Web Services form the palette

14. Drag and Drop the **Web Service** item form the palette to the WSClientPortletView.jsp.

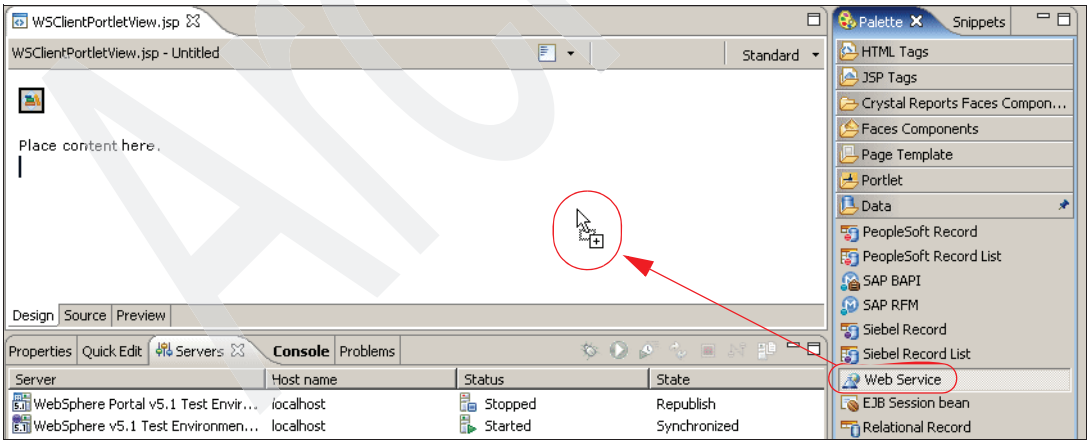


Figure 28-26 Drag and drop the Web Service item

Archived

15. The Web Service Discovery Home window will show. Click the **Web Services from your workspace** link.



Figure 28-27 Web Service Discovery Home

Important: The WebSphere Application Server V5.1 Test Environment must be running in order to access the Web Service during the wizard process.

16. The Web Service information is shown, click the **Primes.wsdl** link.

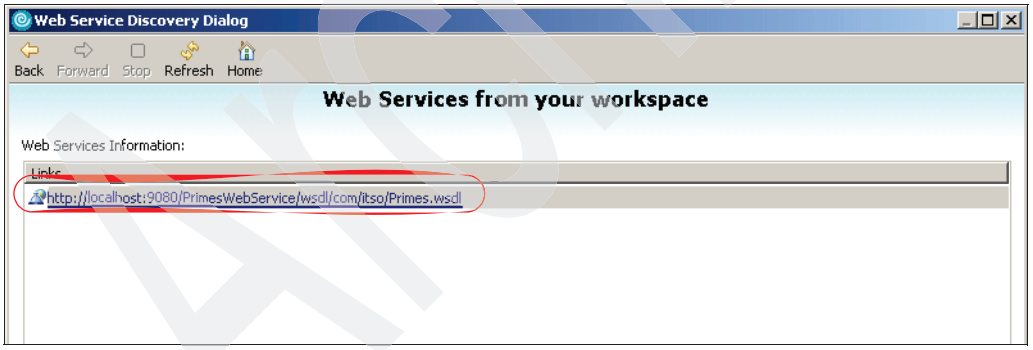


Figure 28-28 Web Services from your workspace

17. In Web Service from a known URL, select the **Port:Primes** from the Web Services Information field.

Archived



Figure 28-29 Web Services from a know URL

18. Click the **Add to Project** button.
19. The Add Web Service window will open. Leave defaults. The class com.itso.PrimesProxy and the method getPrime(int) of the class for the Web Service should already be enter in the Select a Web Service and Select a method field respectively.

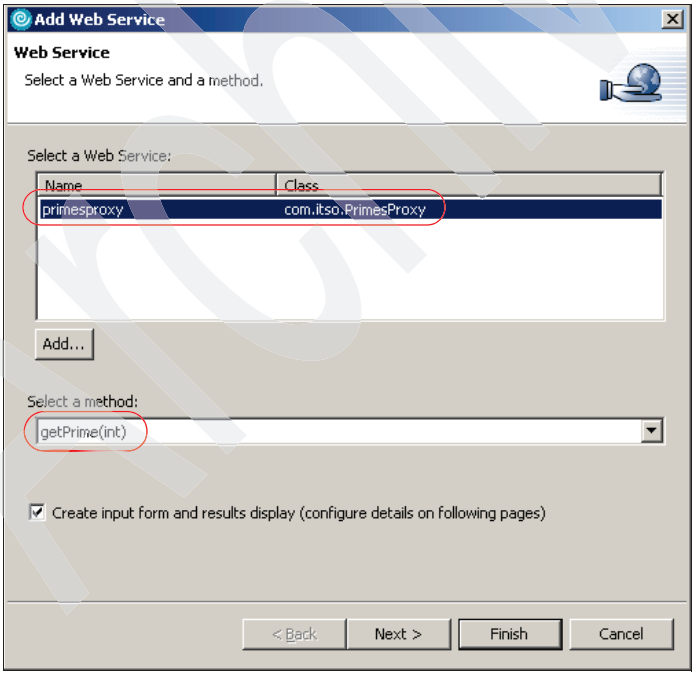


Figure 28-30 Web Service class and method

20. Click **Next**.

21. In the Import From window, leave the default values. This information is used to construct the form where the needed values for the getPrimes method will be entered.

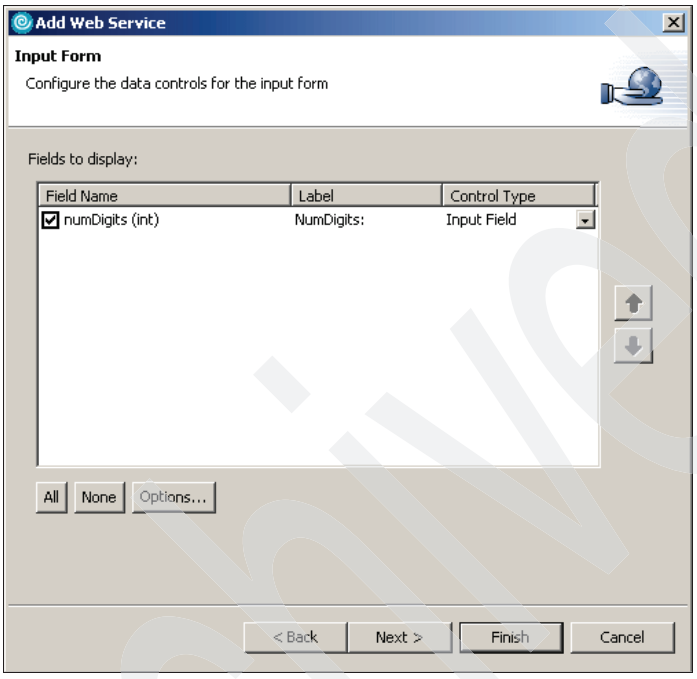


Figure 28-31 Import form configuration

22. Click **Next**.

23. In Result From, leave the default values. These values are going to be used to construct the form where the result will be shown.

Archived

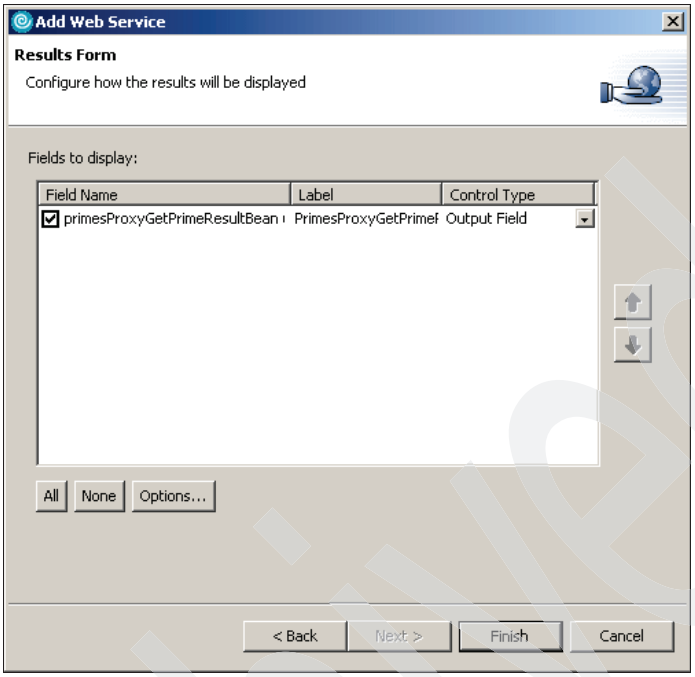


Figure 28-32 Result form configuration

24. Click **Finish**.
25. The JSP Editor will open showing the WSClientPortletView.jsp. From the main menu, select **File** → **Save.**

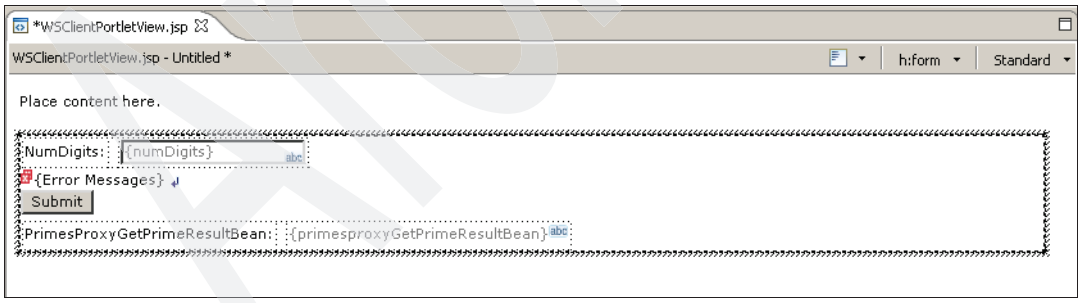


Figure 28-33 Resulting WSClientPortletView.jsp file

Testing the WSClientPortlet

1. In the Project Explorer view, right-click in the WSClientPortlet.
2. Select **Run** → **Run on Server...**

Archived

3. In the Define a New Server window, select **WebSphere Portal V5.1 Test Environment.**
4. Click **Finish**.
5. Once the server starts, a Web browser will open within the workspace, showing the WSClientPortlet in the WPS V5 Test Environment server.

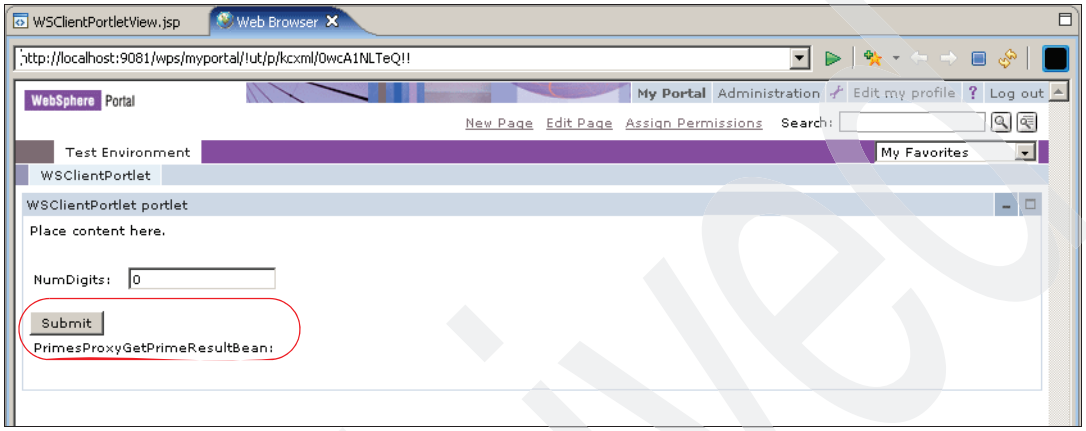


Figure 28-34 WSClientPortlet running on UTE

6. Enter a value in the NumDigits field and click the **Submit** button.

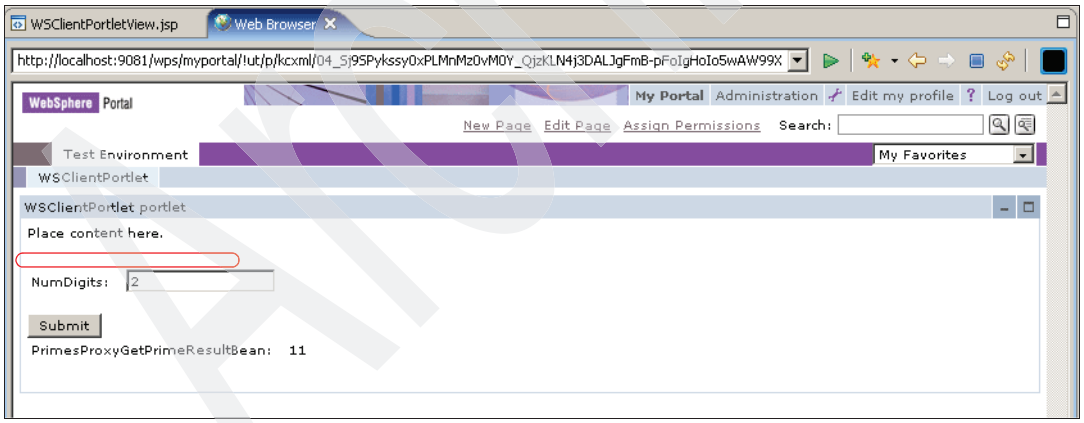


Figure 28-35 Results of Submit a value

7. The result is shown in the PrimesProxyGetPrimeResultBean field.

Archive

29

Web Services for Remote Portlets (WSRP)

This chapter contains an overview and a sample scenario of how to configure Producer and Consumer portal servers to support Web Services for Remote Portlets (WSRP) with IBM WebSphere Portal V5.1.

Archived

© Copyright IBM Corp. 2005. All rights reserved.

29.1 Overview

The Web Services for Remote Portlets (WSRP) standard allows the integration of remote content and Web applications into an end-user portal, simplifying the effort of portal administrators in selecting and including a rich variety of remote content and applications from external sources into their portals without no programming effort.

Some terms used in WSRP are:

- Producer: is a portal that provides one or more portlets as WSRP services and makes them available to Consumers.
- Consumer: is a portal that invokes WSRP services from Producer portals and integrates them to be used by its users.
- Users: use remote portlets integrated in their Consumer portals.
- Remote portlets: standard portlets that have been published to be WSRP services by a Producer.

As illustrated in Figure 29-1, desktop browsers connected to the Consumer portal server can have access to the following portlets:

- Local portlets in Consumer portal
- Remote portlets in Producer portal

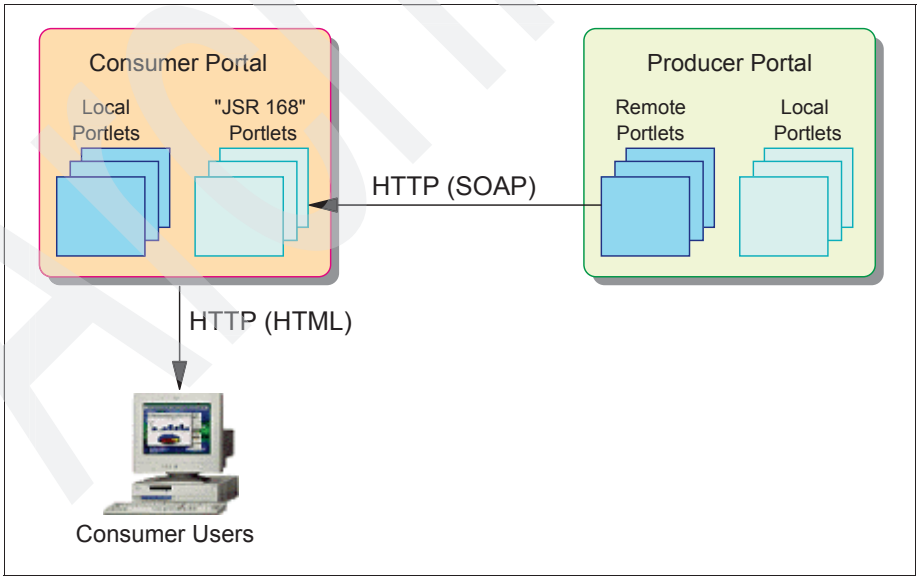


Figure 29-1 Accessing remote portlets

Archived

WSRP services are presentation-oriented, providing the application data and the presentation layer of the portlet. On the other hand, Web Services are data-oriented, providing just the application data, as shown in Figure 29-2.

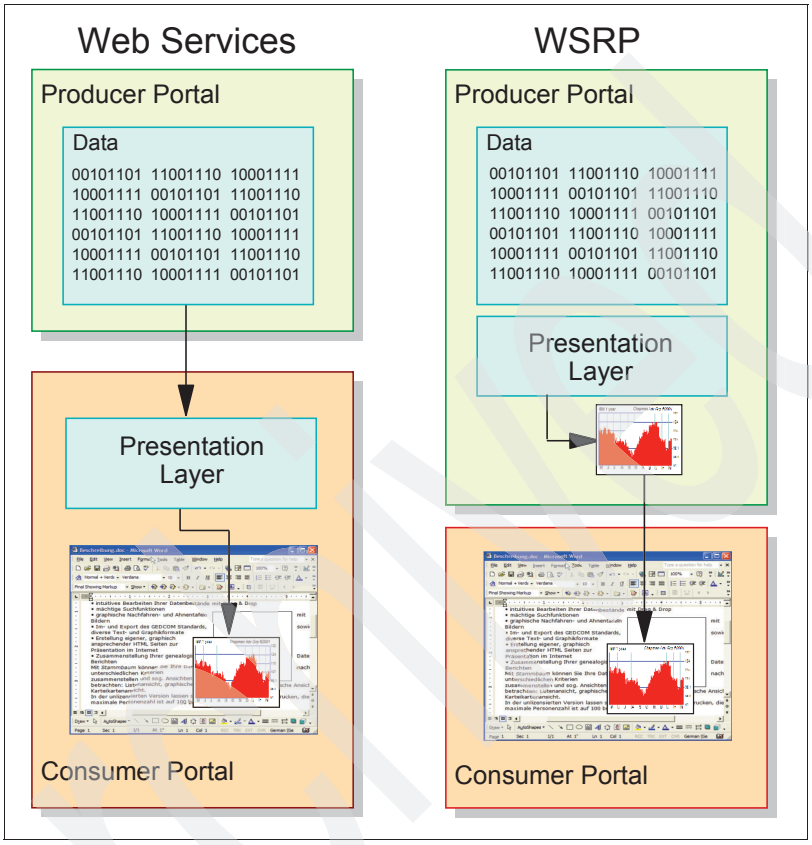


Figure 29-2 A comparison of Web Services and WSRP

WSRP standard defines a set of interfaces provided by Producer portals to be exposed to the Consumers. These WSRP interfaces are described in a Web Services Description Language (WSDL) document accessed by the Consumer portal. It provides information to Consumers about how to bind to the Producer.

The interfaces are:

- Service Description: is a self-description of the Producer, its capabilities and its available portlets. It provides further information to Consumers about the Producer and its properties. This interface is mandatory.

- Markup: is an interface to get and process interactions with markup fragments. It allows to get portlet markups from the Producer and submit portlet requests from Consumer users. This interface is mandatory.
- Portlet Management: defines operations for cloning, customizing and deleting portlets. It allows to customize and manage remote portlets preferences in Consumer portals. This interface is optional.
- Registration: an optional interface for Consumers registration in a Producer portal. It allows Producer to identify each Consumer.

The following is an example of a WSDL document (Example 29-1).

Example 29-1 WSDL document sample

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="urn:oasis:names:tc:wsrp:v1:wsdl"
xmlns:bind="urn:oasis:names:tc:wsrp:v1:bind"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
 <import namespace="urn:oasis:names:tc:wsrp:v1:bind"
location="wsrp_v1_bindings.wsdl" />
 <wsdl:service name="WSRPService">
  <wsdl:port binding="bind:WSRP_v1_Markup_Binding_SOAP" name="WSRPBaseService">
   <soap:address
location="http://wps51rem.itso.ral.ibm.com:9081/wsrp/WSRPBaseService" />
  </wsdl:port>
  <wsdl:port binding="bind:WSRP_v1_ServiceDescription_Binding_SOAP"
name="WSRPServiceDescriptionService">
   <soap:address
location="http://wps51rem.itso.ral.ibm.com:9081/wsrp/WSRPServiceDescriptionServ
ice" />
  </wsdl:port>
  <wsdl:port binding="bind:WSRP_v1_PortletManagement_Binding_SOAP"
name="WSRPPortletManagementService">
   <soap:address
location="http://wps51rem.itso.ral.ibm.com:9081/wsrp/WSRPPortletManagementServi
ce" />
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

29.2 Implementing WSRP in WebSphere Portal

WebSphere Portal Version 5.1 provides support for the WSRP standard enabling portal administrators to provide portlets as WSRP services and integrate WSRP

Archived

services as remote portlets. These remote portlets act in the same manner as local JSR 168 portlets in the Consumer portal, regardless of how they are implemented on the Producer portal.

To implement WSRP with WebSphere Portal is necessary to do the following tasks:

- For Producer portals:
  - Provide or withdraw a portlet.
- For Consumer portals:
  - Register an existing Producer in the Consumer portal.
  - Consume a WSRP service provided by the Producer.

For purposes of this scenario we have two WebSphere Portal V5.1 servers, one of them acting as a Provider server and the other one acting as a Consumer server. We will show the necessary steps to implement remote portlets on these servers.

Note: WebSphere Application Server Version 5.1.1 Cumulative Fix 1 is required to use WSRP with WebSphere Portal.

29.2.1 Tasks for Producer portals

Producers can provide portlets to be available remotely to Consumers or can cancel this service. To provide or withdraw a portlet, portal administrators can use the Manage Portlets portlet or use the XML configuration interface.

In this section we explain the common way to provide and withdraw a portlet in WebSphere Portal, that is using the Manage Portlets portlet.

Note: For information about providing or withdrawing a portlet using the XML configuration interface see the WebSphere Portal InfoCenter at:

`http://publib.boulder.ibm.com/infocenter/wp51help`

Providing a portlet using the Manage Portlets admin portlet

To explain how to provide a portlet we will use a sample portlet named `ActionEvent`, however, you could use any installed portlet in your Producer portal.

1. Log in as an administrator user of the Producer portal.
2. Select **Administration →Portlet Management →Portlets.**
3. In the Manage Portlets portlet, search the portlet you want to provide (ActionEvent portlet for this example) and click **Provide portlet**.

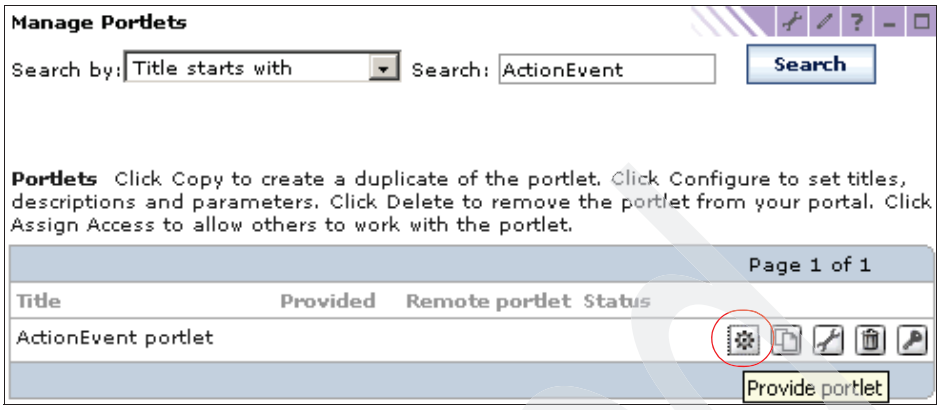


Figure 29-3 Selecting a portlet to be provided

4. Click **Yes** to provide the portlet.

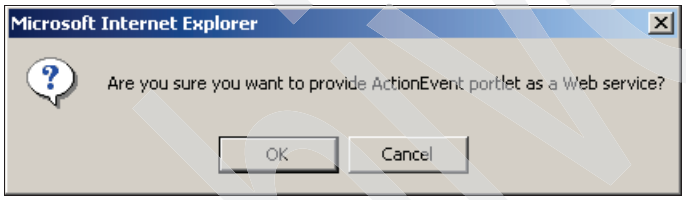


Figure 29-4 Confirming the operation

5. A message will appear indicating that the portlet was successfully provided. A check will appear in the Provided column of the portlet.

Archived

.

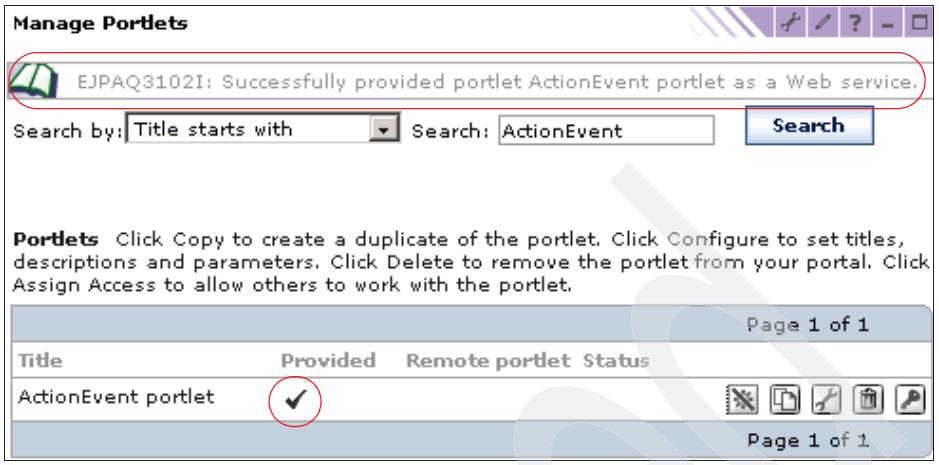


Figure 29-5 The portlet has been provided as a Web service

Withdrawing a portlet using the Manage Portlets portlet

To withdraw a portlet that has been previously provided, follow these steps:

1. Log in as an administrator user of the Producer portal.
2. Select **Administration →Portlet Management →Portlets**. In the Manage Portlets portlet search the portlet you want to withdraw and click **Withdraw portlet**.

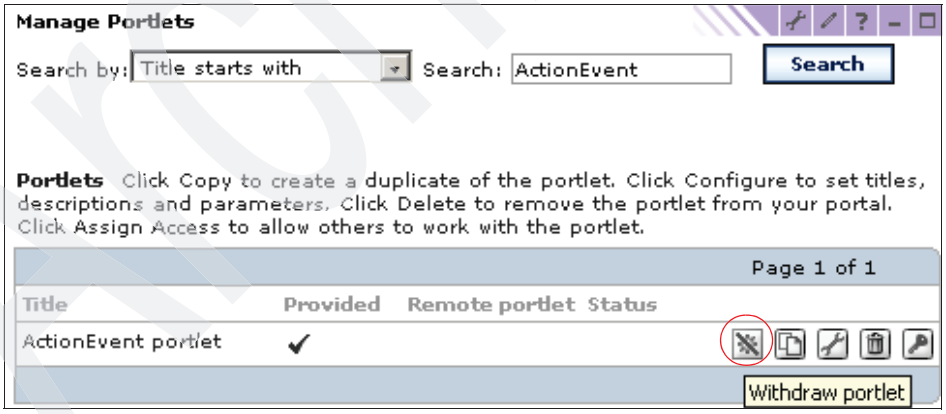


Figure 29-6 Selecting a portlet to be withdrawn

3. Click **Yes** to withdraw the portlet. A message will appear indicating that the portlet was successfully withdrawn.

Archive

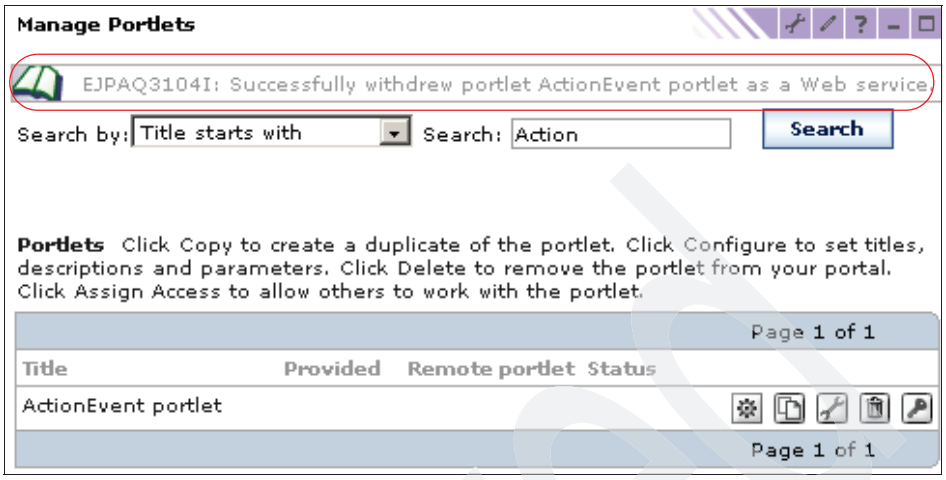


Figure 29-7 The portlet has been withdrawn

29.2.2 Tasks for Consumer portals

In this section we describe the tasks performed by Consumer portals to integrate WSRP services as remote portlets. The following tasks are required in the Consumer portal in order to enable remote portlets for local use:

- Register an existing Producer portal
- Consume the WSRP service

Registering an existing Producer portal

Registering a Producer portal allows the Producer to be known to the Consumer and make available it list of WSRP services that could be consumed by the Consumer portal.

The following scenarios are possible when you try to create a Producer:

- The Consumer has online access to the Producer: in this scenario is possible to use both Web Service Configuration portlet or the XML configuration interface to create the Producer registration in the Consumer portal. Depending the type of Producer you want to create, you have the following options:
  - The Producer does not require registration.
  - The Producer requires registration. In this case the Producer could be enable or not for WSRP registration
- The Consumer works offline with the Producer: in this case is possible to use only the XML configuration interface to create a Producer.

Archived

At the moment of this publication, WebSphere Portal does not support WSRP registration interfaces for Producers, however, Consumers can handle Producers that support this interface.

In this section we explain how to create a Producer that does not require registration, using the Web Service Configuration portlet.

Note: For information about registering Producer portals using the XML configuration interface, see the WebSphere Portal InfoCenter at:

`http://publib.boulder.ibm.com/infocenter/wp51help`

1. Log in as an administrator user of the Consumer portal.
2. Select **Administration** →**Portlet Management** →**Web Services** and click **New Producer**.

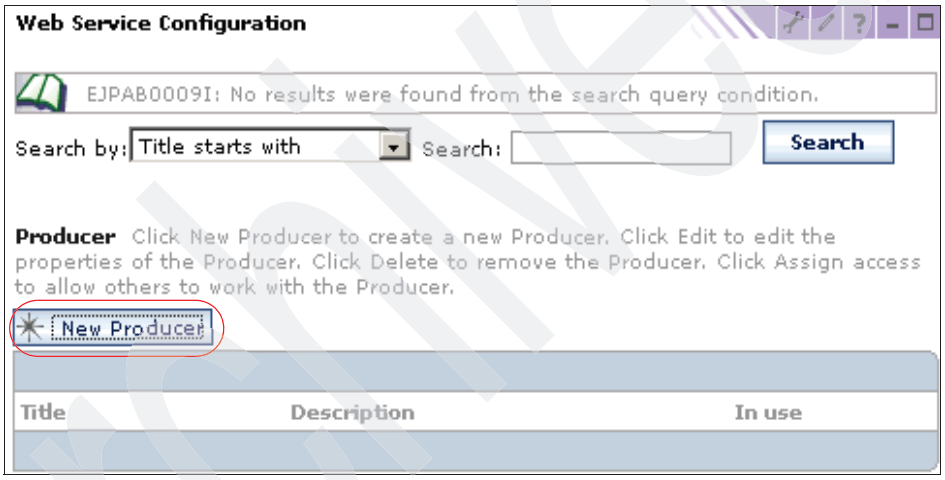


Figure 29-8 Creating a new Producer

3. Enter a name and the URL for the WSDL service definition of the Producer. A description and user attributes are optional.

   In WebSphere, the WSDL of the Producer is located at the following URL:

   `http://<Producer portalhost>:<port>/<wp_contextRoot>/wsdl/wsrp_service.wsdl`

   For this example, we use the following values:

   – Title: `WPS51-Remote`
   – URL:
   `http://wps51rem.itso.ral.ibm.com:9081/wps/wsdl/wsrp_service.wsdl`

   Click **OK**.

Archived

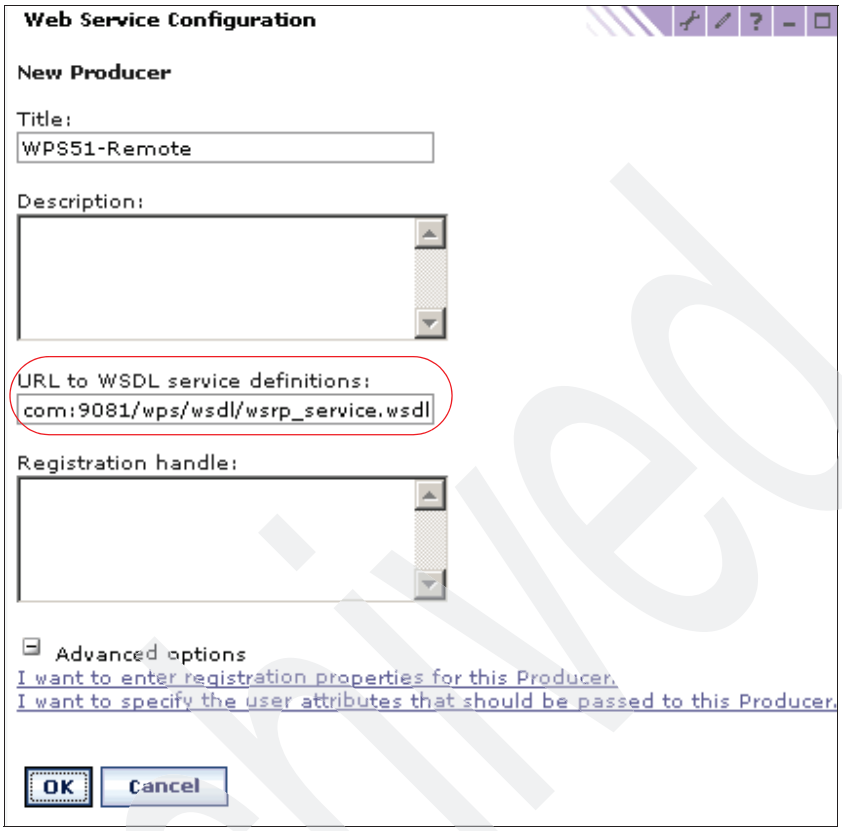


Figure 29-9 New Producer information

4. The new Producer portal will appear in the list.

Archived

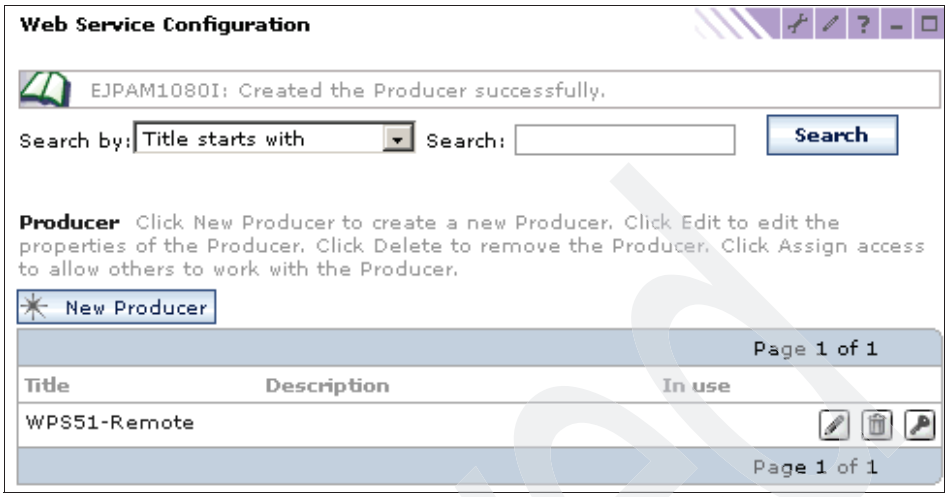


Figure 29-10 The Producer has been created

Consuming a WSRP service

This task allows you to integrate WSRP services from registered Producers into the Consumer portal and interact with them as they were local portlets. To consume a WSRP service, portal administrators can use both the Manage Web Modules portlet or the XML configuration interface.

In this section we explain how to use the Manage Web Modules portlet to integrate a remote portlet.

1. Log in as an administrator user of the Consumer portal.
2. Select **Administration** →**Portlet Management** →**Web Modules** and click **Consume**.

Archived

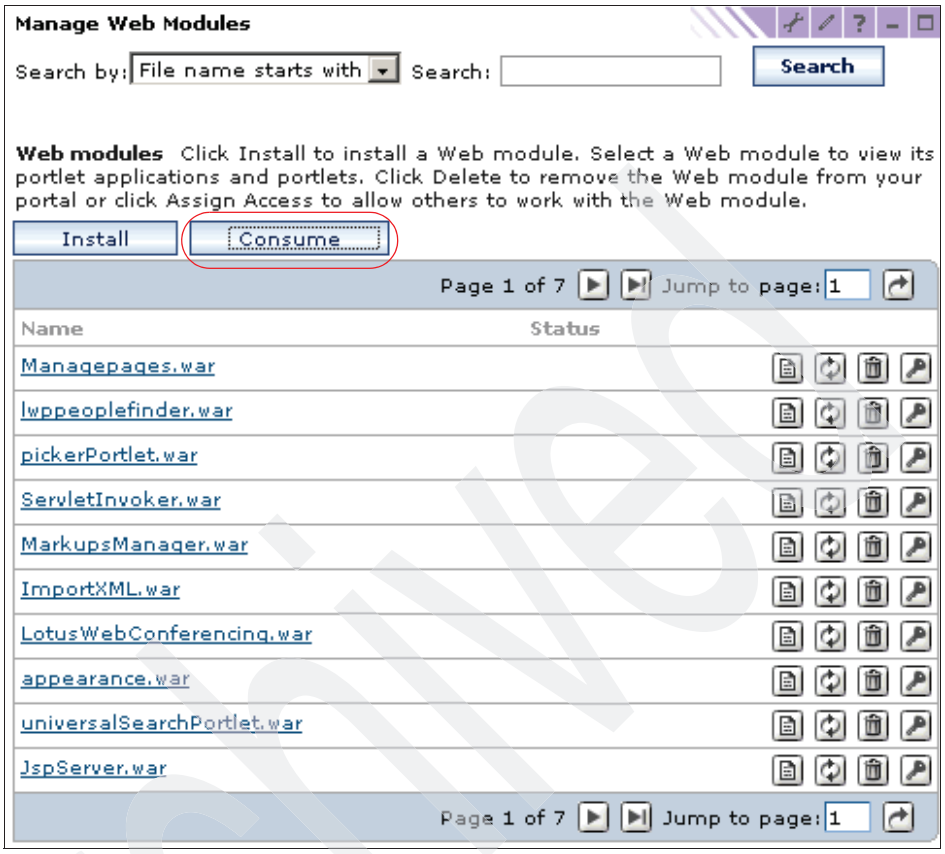


Figure 29-11 Consuming a WSRP service

3. Select a Producer (the Producer created in “Registering an existing Producer portal” on page 892).

Archived

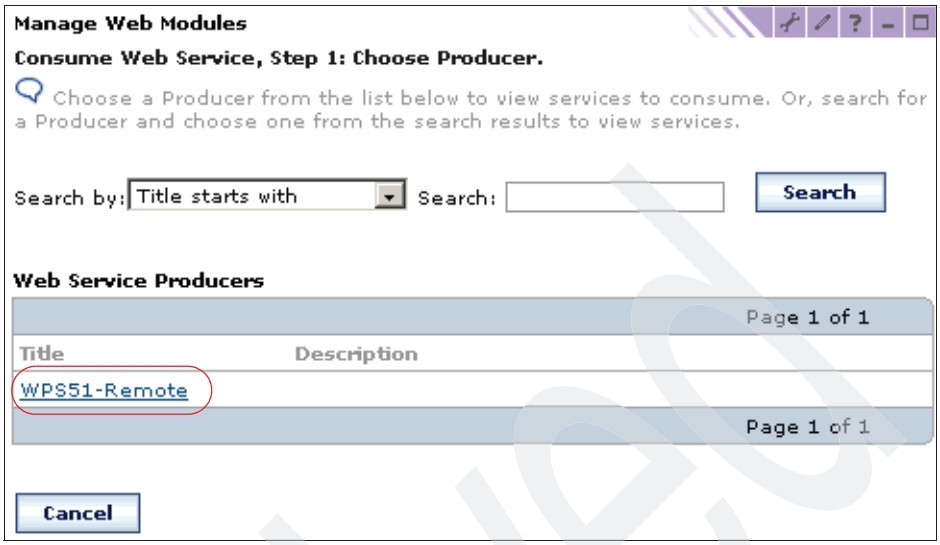


Figure 29-12 Choosing a Producer

4. Select the Web service you want to integrate from the Producer selected previously. Click **OK**.

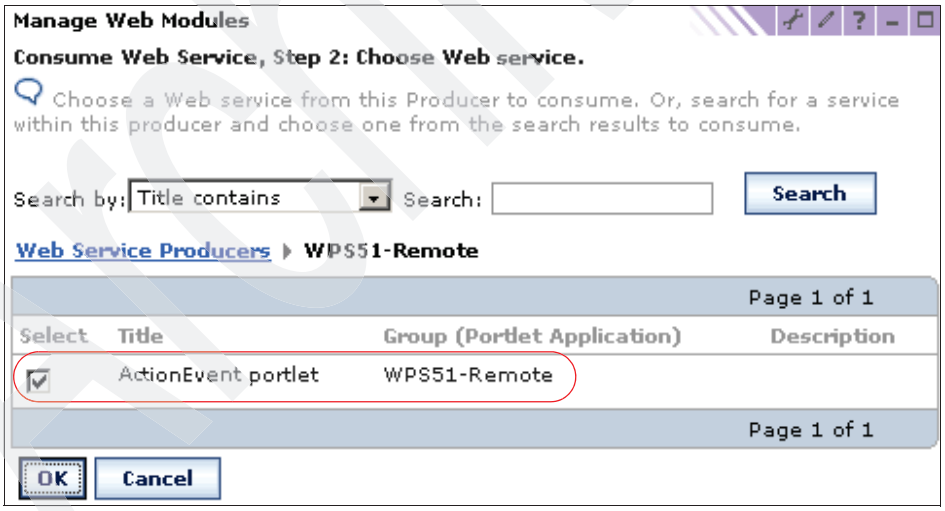


Figure 29-13 Choosing a Web service

5. The remote portlet has been integrated into the Consumer portal and it is available like any other local portlet.

Archived

Remote portlets appear in WebSphere Portal Consumers under the name `RP:[<remote portlet name>]`, for example `RP:[ActionEvent portlet]`. You can see remote portlets selecting the option **Administration →Portlet Management →Portlets** and searching for all the portlets whose title starts with the prefix `RP`, as shown in Figure 29-14.

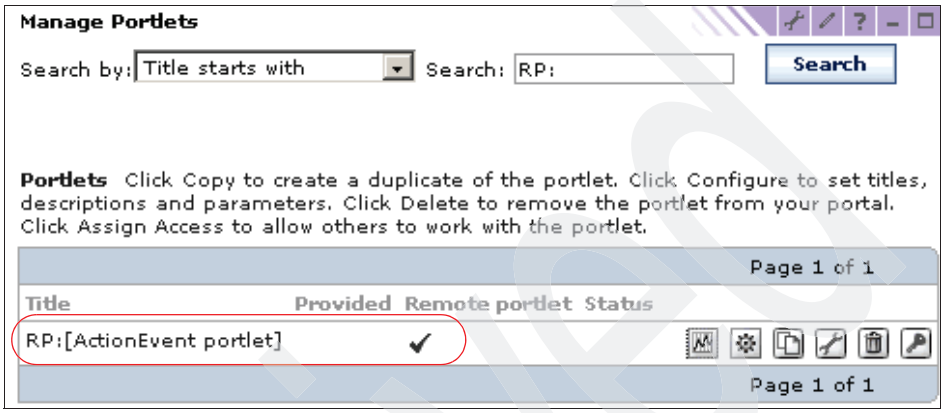


Figure 29-14 Remote portlets in a Consumer server

29.2.3 Testing the scenario

To test this scenario, you should add the integrated remote portlet to a page of the Consumer portal. Follow these steps to add the ActionEvent remote portlet to the Welcome page:

1. Login into the Consumer portal with an user with sufficient access rights to modify a page. For example, you can login as a portal administrator user.
2. Navigate to the Welcome page and click **Edit Page**.
3. Click **Add Portlets** in the container where you want to add the portlet.
4. Search the ActionEvent remote portlet (named `RP:[ActionEvent portlet]`), check the box next to it and click **OK**.
5. Click **Done**. The Welcome page is displayed the ActionEvent remote portlet.

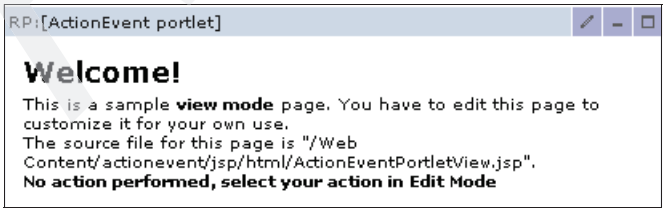


Figure 29-15 The ActionEvent remote portlet

6. Navigate through the portlet and validate that it works as expected.

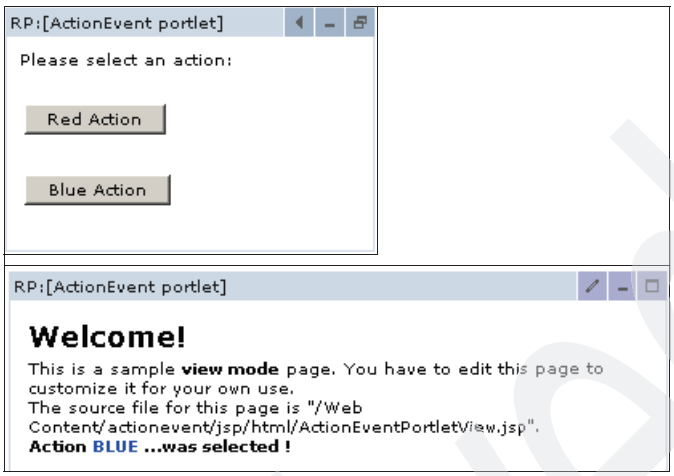


Figure 29-16 Using the ActionEvent remote portlet

29.3 Security

Web Services for Remote Portlets (WSRP) standard does not specify additional security mechanism, therefore the same security considerations than for other kinds of Web services may be applied for WSRP.

In WebSphere Portal implementation of WSRP the following security options can be configured:

Security using Secure Socket Layer (SSL)

Secure Socket Layer (SSL) is a protocol that enables secure message communication between servers and clients over the Internet. In a SSL authentication, the server exchanges the server certificate with the client and, optionally, the client exchanges the client certificate with the server. This certificate exchange is for purposes of verify both server and client (optional) identities.

Using SSL for WSRP with WebSphere Portal allows authentication of Producer portals, as well as authentication of Consumer portals if Client Certificate Authentication is used. The following scenarios are supported, as shown in Figure 29-17 on page 900:

1. Producers authentication, using SSL in Producer portals.
2. Producers and Consumers authentication, using SSL with Client Certificate Authentication in Producers and Consumer portals.

Archived

3. Consumer authentication for portal users, using SSL.
4. Consumer and portal user authentication (if supported), using SSL with Client Certificate Authentication in Consumer portals and portal users.

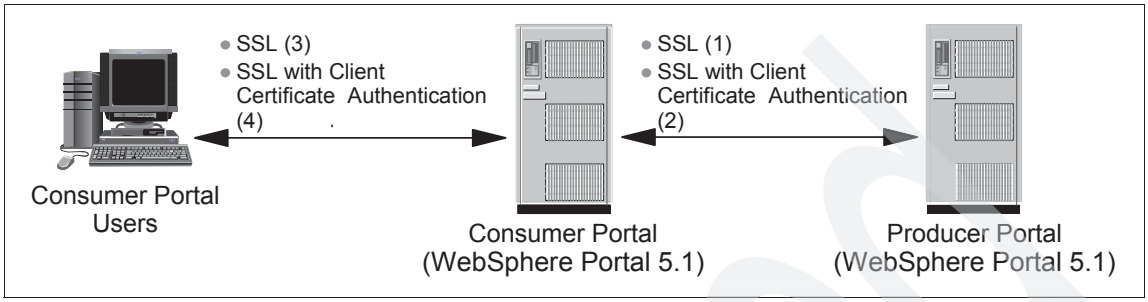


Figure 29-17 SSL scenarios for WSRP with WebSphere Portal

Note: For more information about implementing SSL between a producer portal and a consumer portal see the WebSphere Portal InfoCenter at:

`http://publib.boulder.ibm.com/infocenter/wp51help`

LTPA token authentication

Lightweight Third Party Authentication is an IBM proprietary protocol that uses cryptography to support security in a distributed environment. LTPA allows authentication of the end users using LTPA token forwarding.

When a client authenticates to a Consumer portal using the LTPA authentication mechanism, a unique LTPA token is created for this client and it is used for all client requests of that session. This token is stored in a browser cookie to support SSO with other LTPA enabled application servers and it contains information about the cookie domain, user information, digital signature and date of expiration. The Consumer portal forwards the client LTPA token to the Producer who has to have the same LTPA keys that the Consumer.

Note: For more information about LTPA configuration for WSRP with WebSphere Portal see the WebSphere Portal InfoCenter at:

`http://publib.boulder.ibm.com/infocenter/wp51help`

30

Portlet debugging

Rational Application Developer provides a powerful debugger tool for suspending launches, stepping through the program code, and examining the contents of variables. This chapter gives you a brief introduction to the techniques used to debug portlets and discusses how to detect error during compile and runtime.

Archived

© Copyright IBM Corp. 2005. All rights reserved.

30.1 Overview

For software development, we can distinguish two different kinds of errors:

- Compile errors appear during compile and are thrown by the Java compiler. A typical example for this type of error is an improperly typed method or class name. This type of error can be found very easily because the compiler checks the code and presents a meaningful message.
- There are also runtime errors, which cannot be found by the compiler; thus they appear only during runtime. An example might be a loop stepping through an array with a size smaller than the loop variable. These kinds of errors are typically fixed using a debugger.

30.2 Sample scenario

This section provides a sample scenario to illustrate how to fix compile and runtime errors in portlets using the debug functions provided by Rational Application Developer. You will use the portlet used in Chapter 6, “IBM Portlet API portlet development” on page 203 as a base for this scenario. These activities will allow you to understand the techniques used to debug portlets.

30.2.1 Fixing compile errors

In this section, an example is provided to illustrate a Java code validation. An invalid character will be entered in the Java code to introduce an error and illustrate the correction process.

Rational Application Developer provides different validators for different types of project resources, for example XML, HTML or Java files. In general, a validator is a process which validates a certain resource. It can be invoked manually (code validation is explicitly invoked by the user) or automatically (it occurs whenever a resource changes or is saved). After the validation process is finished, it displays the results of the validation in the Problems view.

To customize the validation settings of a project, right-click it from the Project Explorer view and select **Properties**. In the Validation page, you can disable all or only certain validators, as shown in Figure 30-1 on page 903.

Archived

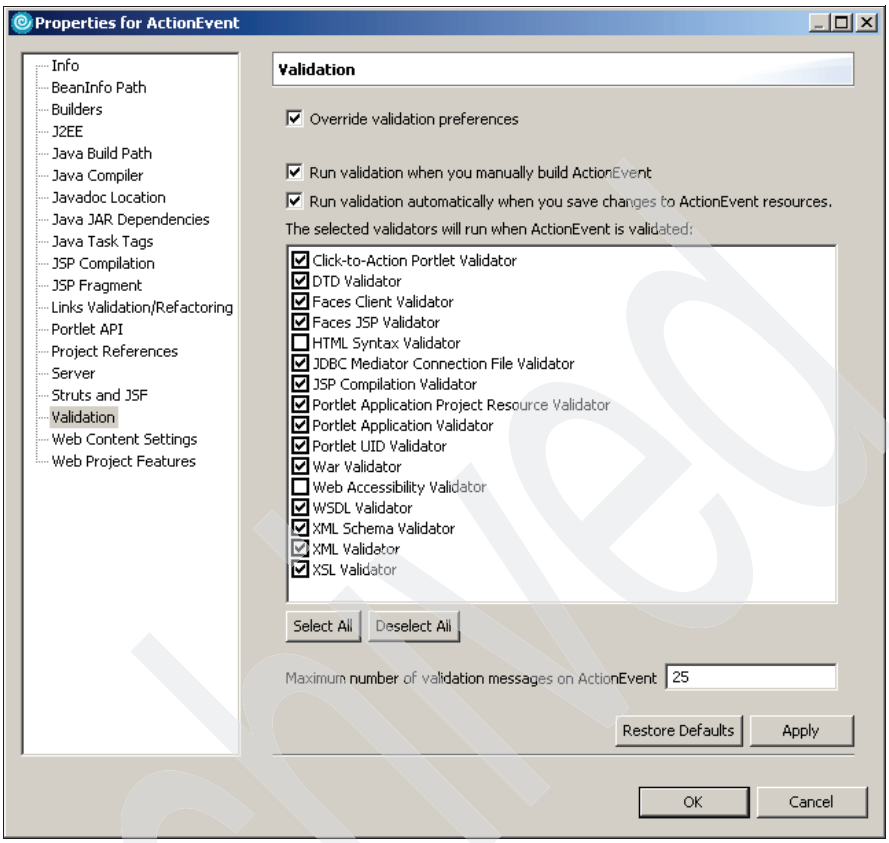


Figure 30-1 Validators

To create a compile error, proceed as follows:

1. In the Project Explorer view, select **ActionEvent** → **Java Resources** → **JavaSource** → **actionevent** → **ActionEventPortlet.java** as shown in Figure 30-2 on page 904. Double-click the Java file; it will open in the editor in the upper right-hand portion of the screen. If the file is already opened, scroll to the top.

Archived

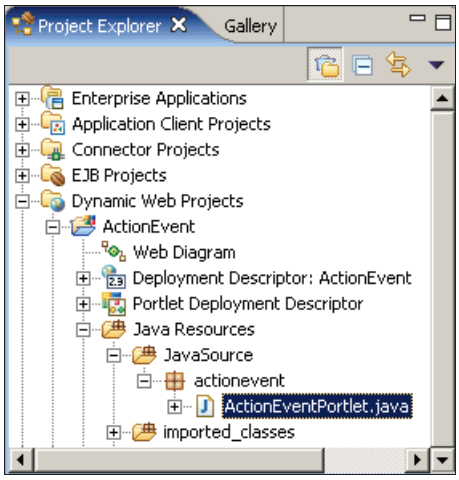


Figure 30-2 Double-click ActionEventPortlet.java to open

2. In the editor window, you should see a declarative statement:

```
public class ActionEventPortlet extends PortletAdapter implements
ActionListener
```

3. Place the letter `x` at the beginning of this statement to create an error. See Figure 30-3.

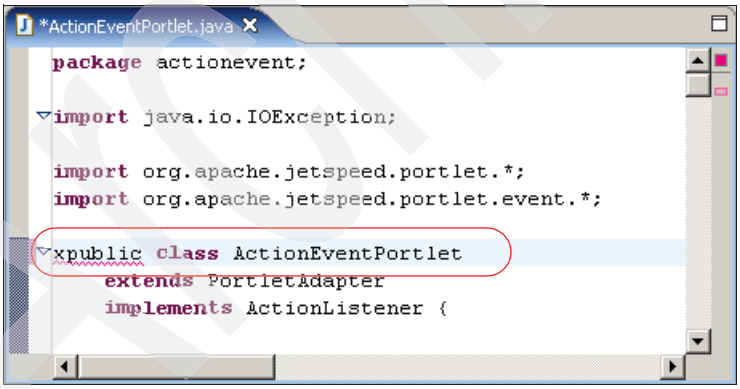


Figure 30-3 Incorrect public class declaration

4. Press **Ctrl-S** to save this file.
5. The compilation process fails due to the error you introduced. In the Problems view in the lower right-hand portion of the screen, an error message appears, as shown in Figure 30-4 on page 905.

Archived

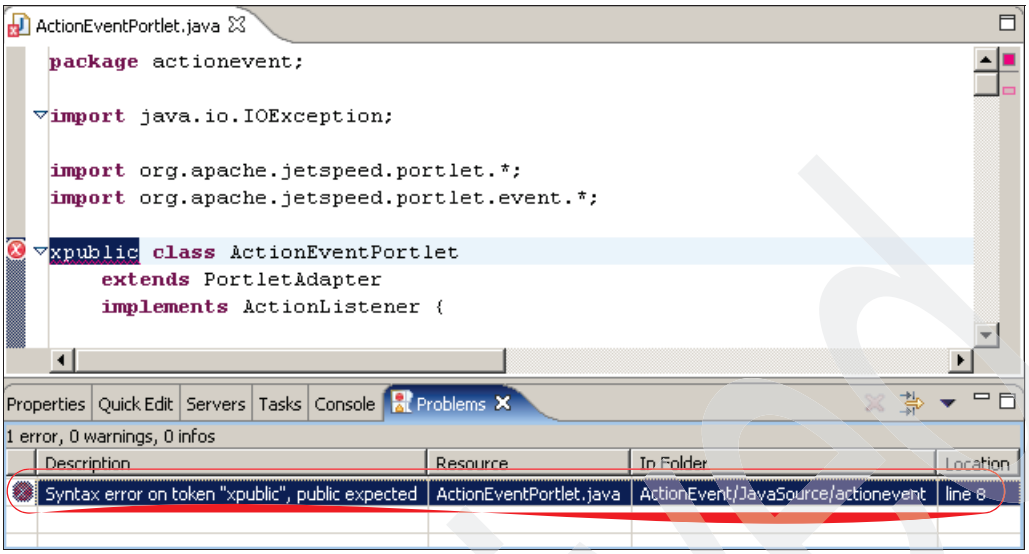


Figure 30-4 Result of saving an incorrect Java file

6. Double-click the red error icon in the Problems view. The problem area in the code will be highlighted.

   Tip: If you cannot see the whole error message in the Problems view because of its length, move over the red error symbol to the left of the Java editor. A small help window appears with the whole error message.

7. Remove the letter `x` before `public` statement to return the code to its original condition. Press **Crtl-S**. The code will be validated again, and the error message will disappear from the Problems view.
8. Close the editor by clicking the **X** on the ActionPortlet.java tab.

30.2.2 Debugging a portlet application

In this sample scenario, you will set a breakpoint, start WebSphere Portal Test Environment in Debug mode and modify the value of a variable.

1. In the Project Explorer view, select **ActionEvent** → **Java Resources** → **JavaSource** → **actionevent** → **ActionEventPortlet.java** as shown in Figure 30-5 on page 906. Double-click the Java file to edit it.

Archived

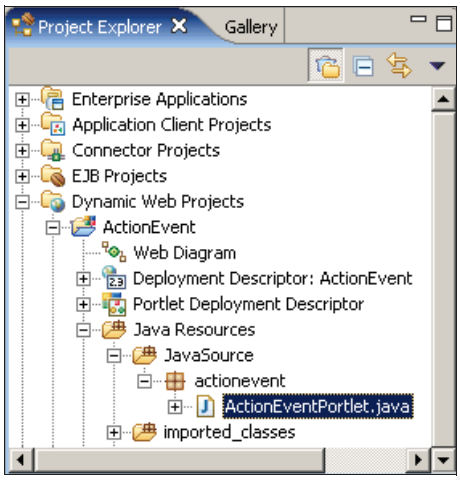


Figure 30-5 Select ActionEventPortlet.java to open it

2. The editor will open in the upper right-hand corner of the screen.
3. In this portlet class, there are five methods:
   - init
   - doView
   - doEdit
   - actionPerformed
   - getJspExtension
4. In the actionPerfomed method, you will set a breakpoint. Place the cursor on the `setAttribute` statement as highlighted on Example 30-1.

Example 30-1 Setting a breakpoint

```
public void actionPerformed(ActionEvent event) throws PortletException {
   if( getPortletLog().isDebugEnabled() )
      getPortletLog().debug("ActionListener - actionPerformed called");
   String actionString = event.getActionString();
   PortletRequest request = event.getRequest();
   if(actionString.equalsIgnoreCase(ACTION_RED)){
      String value = "Action <FONT color=\"#ff0000\">RED</FONT>";
      PortletData portData = request.getData();
      try{
         portData.setAttribute("value", value);
         portData.store();
      }
      catch (AccessDeniedException ade){
      }catch (IOException ioe){}
   }
```

Archived

```
        if(actionString.equalsIgnoreCase(ACTION_BLUE)){
            String value = "Action <FONT color=\"#0000ff\">BLUE</FONT>";
            PortletData portData = request.getData();
            try{
                portData.setAttribute("value", value);
                portData.store();
            }
            catch (AccessDeniedException ade){
            }catch (IOException ioe){}
        }
    }
```

Note: This is the statement where the actionPerformed method has identified the action and set an attribute (setAttribute) in the request object; the attribute is to be rendered later in View mode.

5. Right-click the context bar to the left of the `setAttribute` statement, then select **Toggle Breakpoint** from the context menu as shown in Figure 30-6.

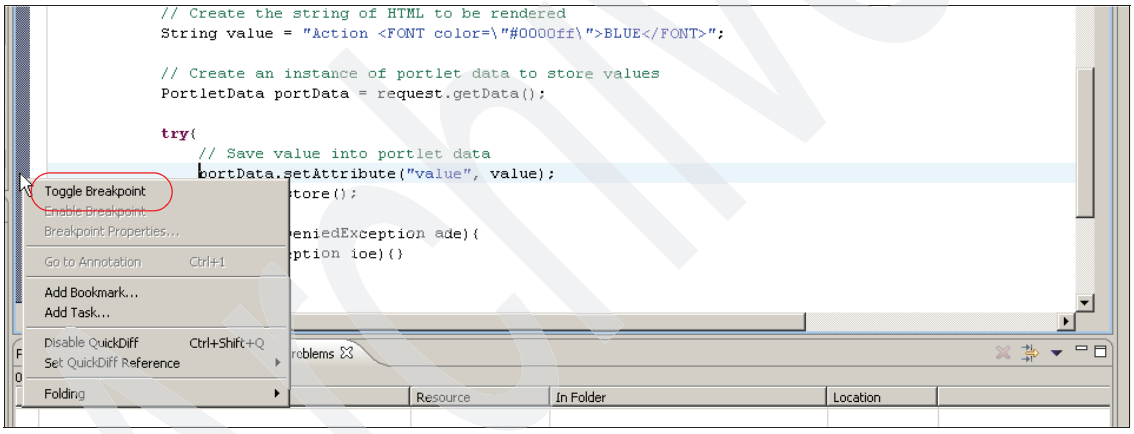


Figure 30-6 Adding a breakpoint

6. After setting the breakpoint, you should see a blue dot on the marker bar, that means the breakpoint is enabled. Enabled breakpoints are shown with a checkmark overlay after their class has been loaded by the Java VM.

   If the breakpoint is disabled, it will not cause threads to suspend and is indicated with a white circle.

Archived

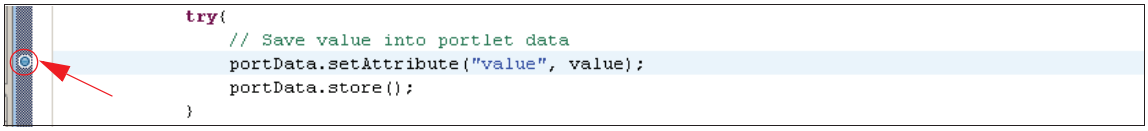


Figure 30-7 Breakpoint mark

7. Click the **Servers** tab. If the WebSphere Portal V5.1 Test Environment has been started, right-click and select **Stop** and wait until you see the message indicating that WebSphere Portal has stopped.
8. Once the Test Environment is stopped, right-click **ActionEvent** from the Project Explorer view, then select **Debug on Server...**

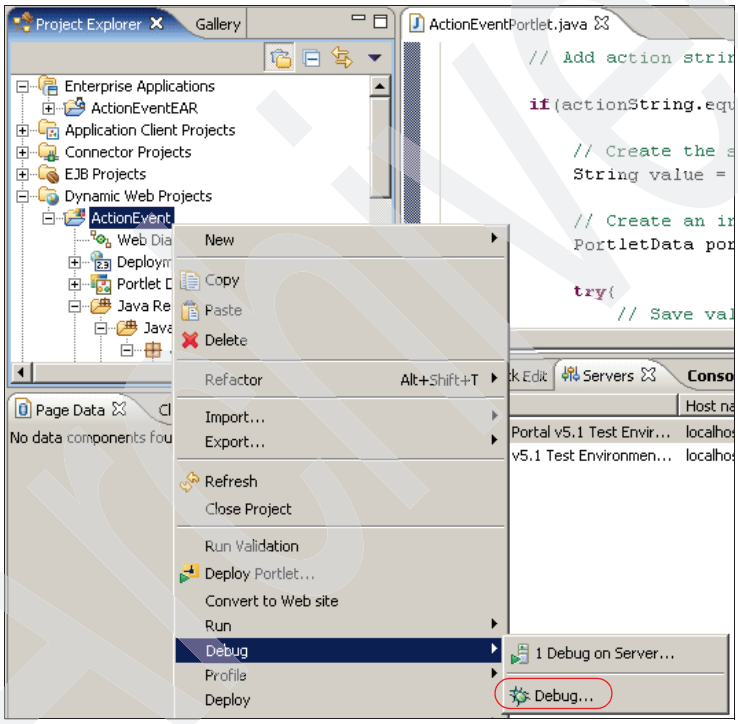


Figure 30-8 Debug the portlet on Test Environment

9. Select **WebSphere Portal V5.1 Test Environment** and click **Finish**.

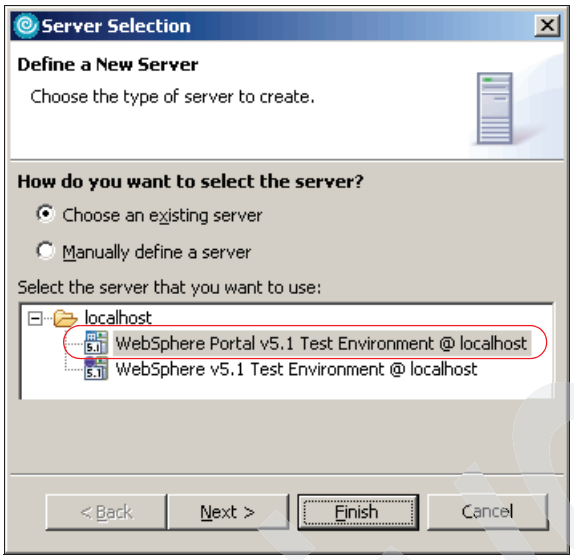


Figure 30-9 Debug on WebSphere Portal V5.1 Test Environment

> **Note:** Starting WebSphere Portal Test Environment in debug mode will take a few minutes.

10. Click the **Debug** button at the upper-right corner of the workbench to change to the Debug perspective. It will look as shown in Figure 30-10 on page 910.

Archived

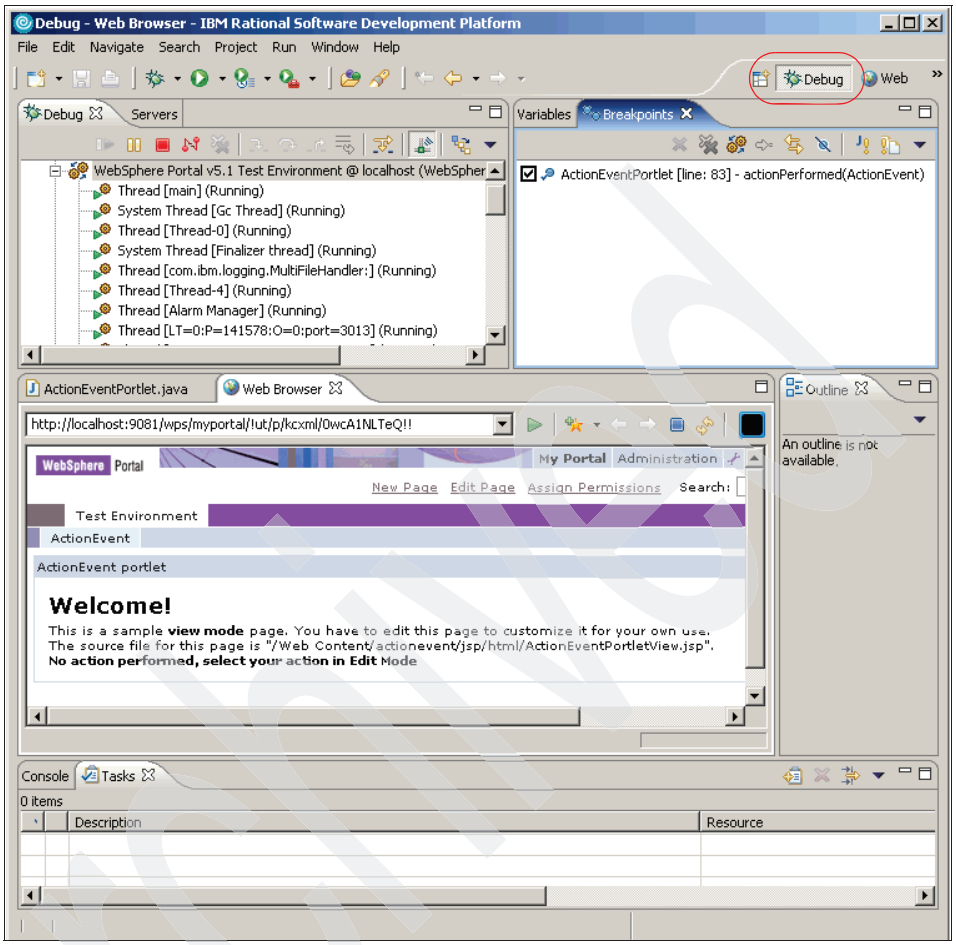


Figure 30-10 Debug perspective of Portal running the ActionEvent portlet

11. The portlet will run on the built-in browser shown in the middle left panel (in the Debug perspective) or the upper right panel (in the Portal perspective).
12. Now click **Edit** icon to select an action.

Archived

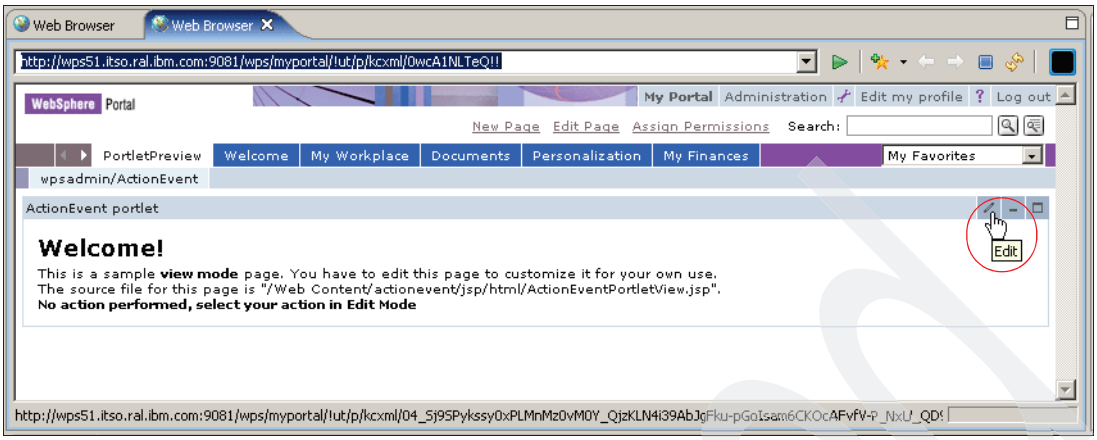


Figure 30-11 Change the portlet to Edit mode

13. Select the **Red Action** button; remember that the breakpoint has been set in this action path (actionPerformed method).
14. The action (Red Action) will now execute up to the breakpoint you have previously set. When the breakpoint is reached, the Java editor displays the code and the statement with the breakpoint is highlighted.

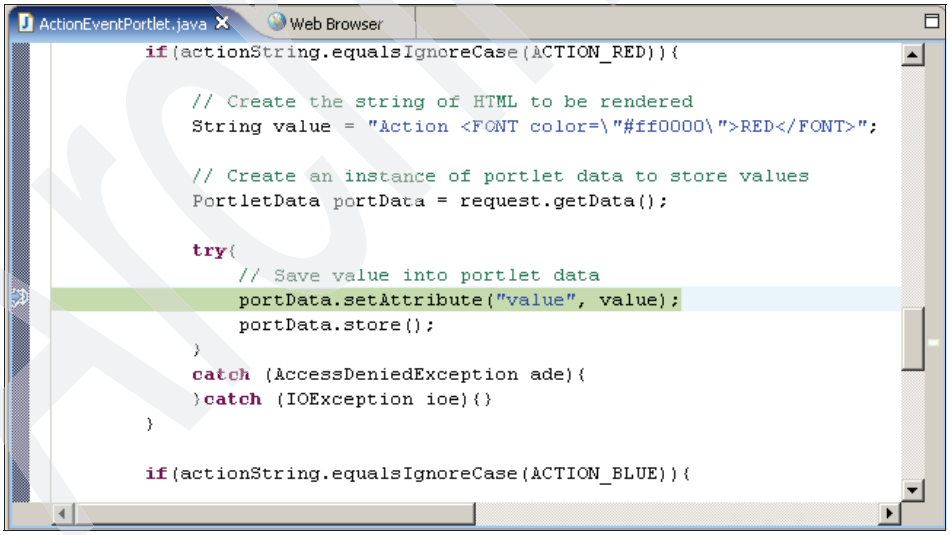


Figure 30-12 Debugger stops execution at the breakpoint

15. Place the cursor in the context bar where the breakpoint is located and right-click to select **Toggle Breakpoint** from the context menu to remove the break. Take a moment to examine the code before proceeding.
16. From the Debug perspective, select the **Variables** tab.
17. Locate the variable with name *value*.
18. Select the `value` variable (`value=Action <FONT color=\"#ff0000\">RED</FONT>`), then right-click it and select **Change Value...** from the context menu.

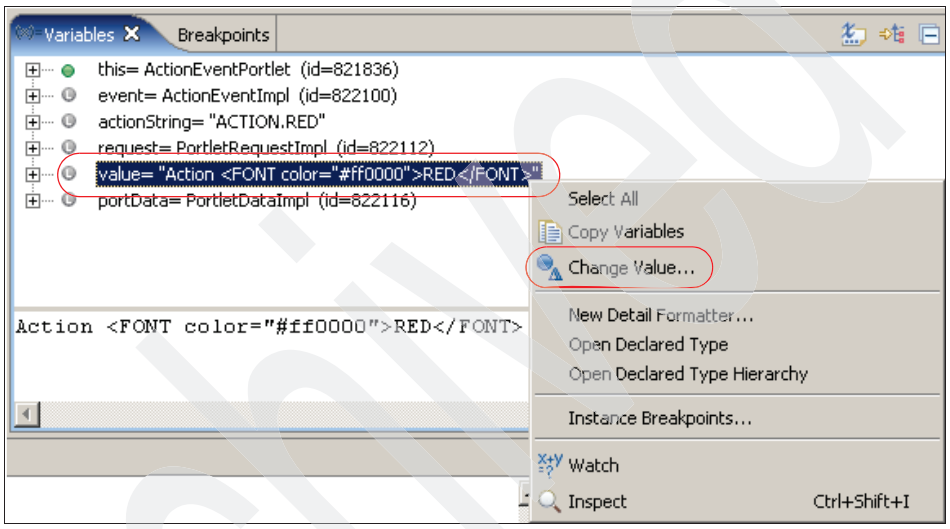


Figure 30-13 Changing the variable value in the Variables view

19. Enter `Action <FONT color=\"#00ff00\">GREEN</FONT>` as the new value. Click **OK**.

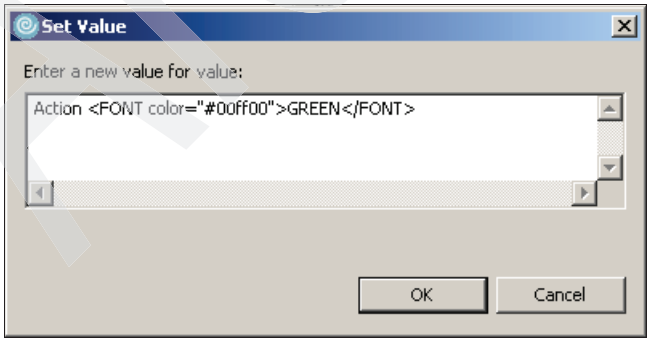


Figure 30-14 Change value

Archived

20. Click the **Resume** icon located in the Debug tab (green triangle on the left side of the toolbar icons) to let the portlet continues its execution.

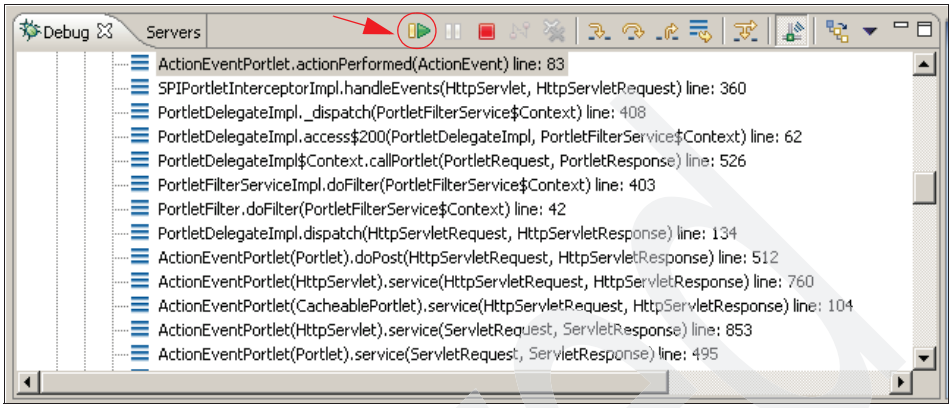


Figure 30-15 Click the Resume icon to resume the execution of the portlet

21. Select the **Web browser** tab from the display toolbar in the middle left panel of the screen. The result of the action will be displayed.

    Note that the action displayed in View mode is now `Action GREEN ...was selected !` (and not `Action RED ...was selected !` as originally shown).

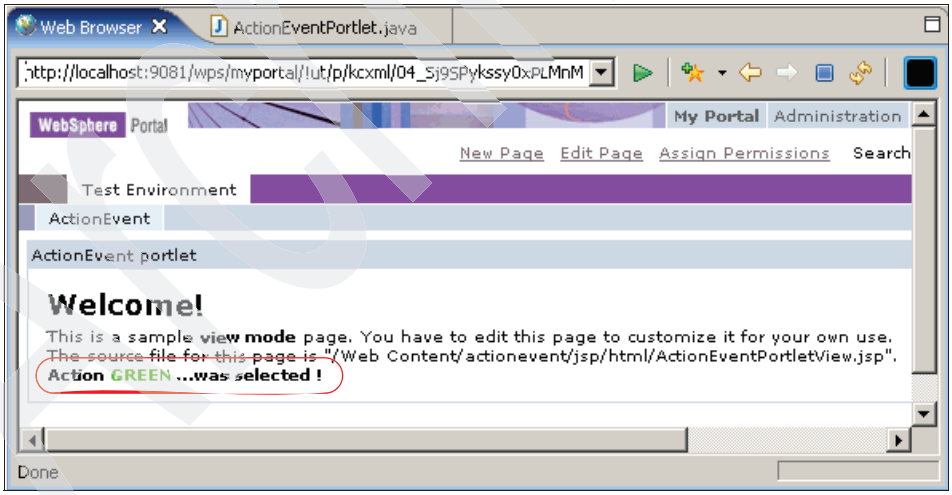


Figure 30-16 The ActionEvent portlet displaying a new variable value

Archived

31

Remote Server Attach

This chapter provides an overview and a sample scenario to test and debug portlet projects running on a remote WebSphere Portal Server. The topics presented in this chapter will allow you to understand the techniques and the configuration required to test and debug portlets remotely.

Archived

© Copyright IBM Corp. 2005. All rights reserved.

31.1 Overview

Testing and debugging portlets are tasks that involve running portlets on a server, either on the test environment server within the workbench (local), or on a separate portal server (remote).

A WebSphere Portal Server Attach is a server type that allows Rational Application Developer tool to attach to a remote WebSphere Portal Server already started, so that you can run and debug a portlet application running on a remote WebSphere Portal system accessible through a network connection.

The sample development workstation and the remote Portal server are illustrated in Figure 31-1.

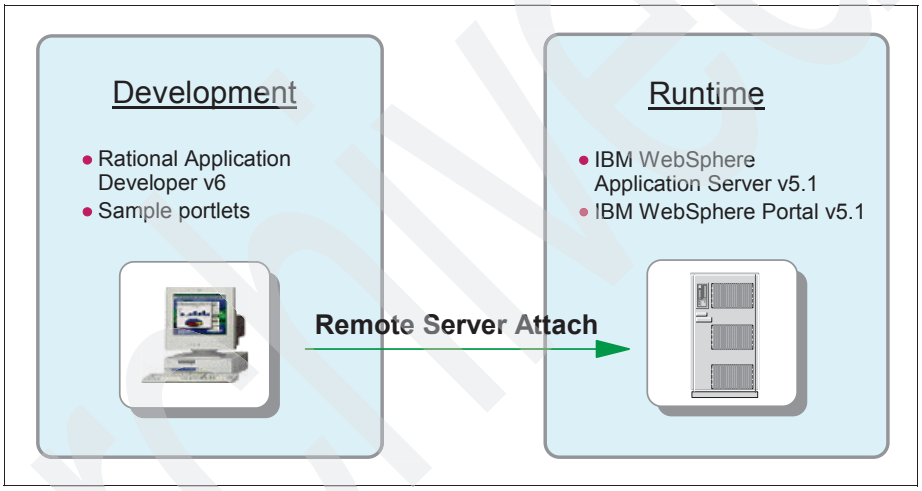


Figure 31-1 Remote Server Attach

In order to debug a portlet on a remote server, the following main task must be performed:

1. Prepare the WebSphere Portal server to work in debug mode.
2. Create a user on the WebSphere Portal server for the developer that will be performing debug tasks (optional).
3. Create a WebSphere Portal Server Attach on the Rational Application Developer workbench.
4. Run and debug a portlet on the remote WebSphere Portal server.

Figure 31-2 on page 917 shows how portlets are debugged on a remote server.

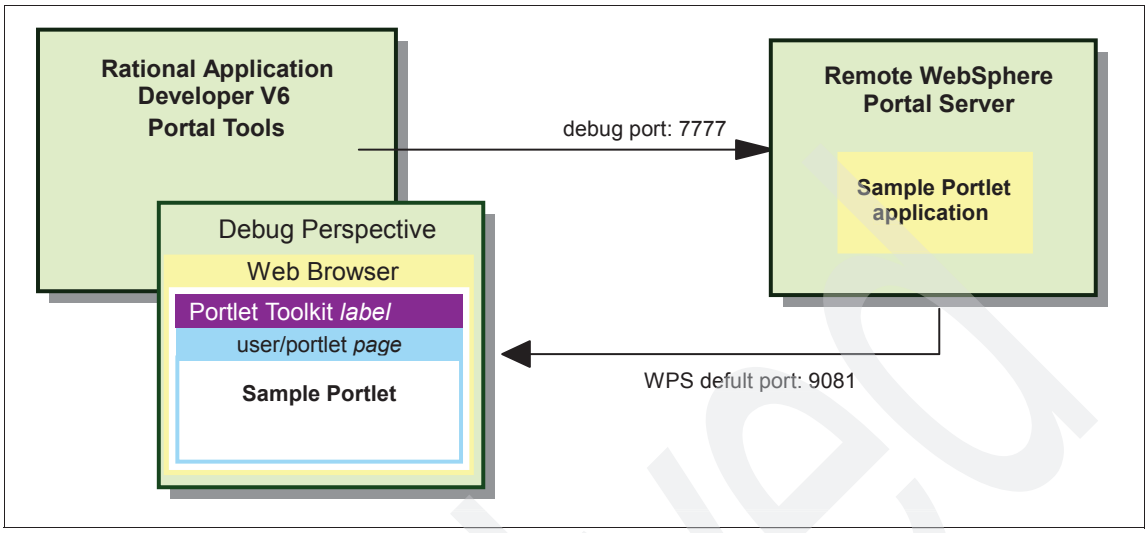


Figure 31-2 Debugging a portlet on Remote Server Attach

31.2 Sample scenario

In this scenario, we debug a portlet on a remote WebSphere Portal server V5.1. We will use the `ActionEvent` portlet created in Chapter 6, "IBM Portlet API portlet development" on page 203 and provided as additional material of this redbook.

The remote WebSphere Portal server and the Rational Application Developer run on separate machines.

31.2.1 Preparing Remote Portal server to debug

The following configurations are needed to configure WebSphere Portal Server to start in debug mode

1. Start the WebSphere Application server by clicking **Start** → **Programs** → **IBM WebSphere** → **Application Server** → **Start the Server.**
2. Once WebSphere Application Server has started, open the Administrative Console. In a Web browser, enter `http://localhost:9090/admin`
3. Log on to the console as an administrative user, for example `wpsadmin`.
4. In the left pane, expand **Servers** and select **Application Servers**.



Figure 31-3 Application Server link

5. In the Application Servers page, select **WebSphere_Portal**.

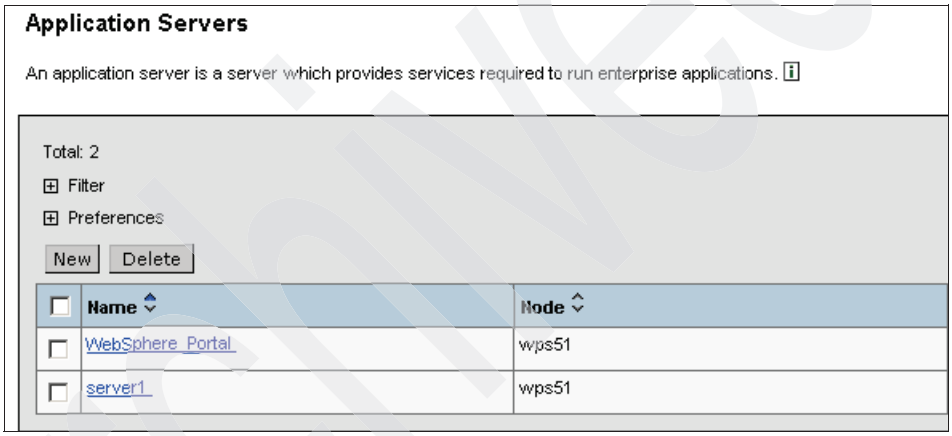


Figure 31-4 Application Server Page

6. In the Additional Properties section, scroll down and select **Debugging Service**.

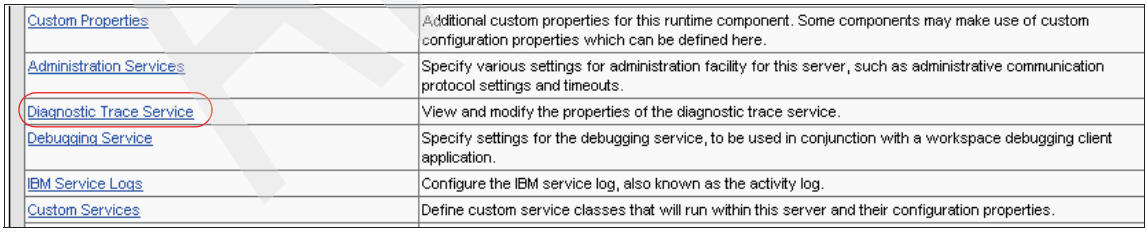


Figure 31-5 Debugging Service link

7. On the Debugging Services page do the following:

Archived

– Select the **Startup** check box.
– Leave the default value for the JVM debug port (7777).
– Enter the following JVM debug arguments:

```
-Djava.compiler=NONE -Xdebug -Xnoagent
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=7777
```

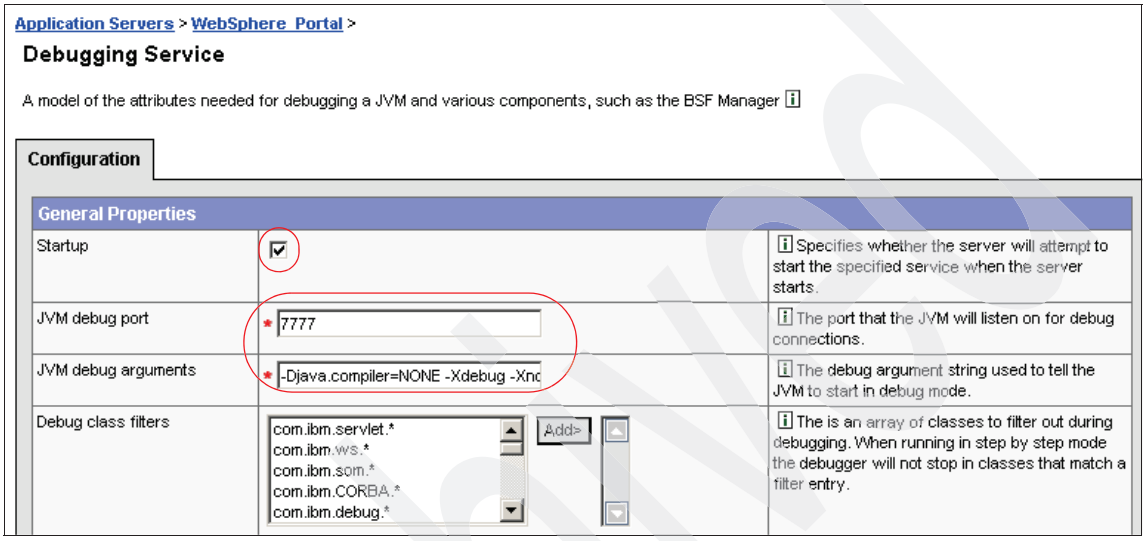


Figure 31-6 Debugging Service page

8. Click **Ok**
9. Click the **Save** link.
10. Click the **Save** button to apply changes to the master configuration.
11. Configuring the debugging service should also configure the JVM. Review this by doing the following:
    a. In the Administrative console of the WebSphere Application Server, expand **Servers** → **Application Servers** and select **WebSphere_Portal**
    b. Click **Process Definition** link in Additional Properties and scroll down to select **Java Virtual Machine**.
    c. In the General Properties section, verify the following:
       - The **Debug Mode** check box must be selected.
       - The **Debug arguments** must be:

```
-Djava.compiler=NONE -Xdebug -Xnoagent
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=7777
```

Archive

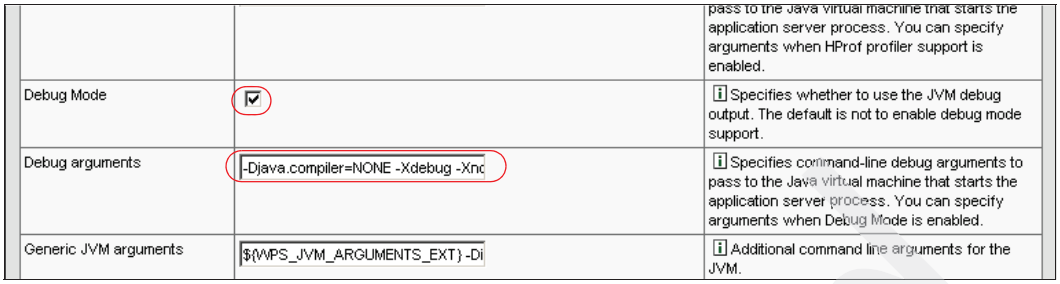


Figure 31-7 Java Virtual Machine page

Note: The address value must match with the value enter in the previous procedure. For this scenario the default value 7777 is used.

12. Logout from WebSphere Application Server Administrative console.
13. Stop the WebSphere Application Server by clicking **Start** → **Programs** → **IBM WebSphere** → **Application Server** → **Stop the Server**.
14. Start the WebSphere Application Server
15. Restart the WebSphere Portal server.
    - Stop the Portal server by clicking **Start** → **Programs** → **IBM WebSphere** → **Portal Server** → **Stop the Server**.
    - Start the Portal server by clicking **Start** → **Programs** → **IBM WebSphere** → **Portal Server** → **Start the Server**. Wait a few minutes for Portal to be started (open for e-business message).

31.2.2 Creating Remote Portal server users

It is recommended that you create necessary users on the remote portal if you want to test or debug portlets with multiple users on the same remote portal server. A user ID and password for a WebSphere Portal should be created for each portlet developer that will debug on a remote server. This user must be created manually and the required edit permissions are automatically added. The user ID is also part of the label of the preview page where the portlet is shown in debug time.

For this scenario, we will use the administrator user `wpsadmin`.

31.2.3 Creating a WebSphere Portal Server Attach

On the Rational Application Developer machine, follows these steps to create a WebSphere Portal Server Attach:

1. Open the Rational Application Developer and select **File** → **New** → **Other...**
2. Select **Server** → **Server** and click **Next**.

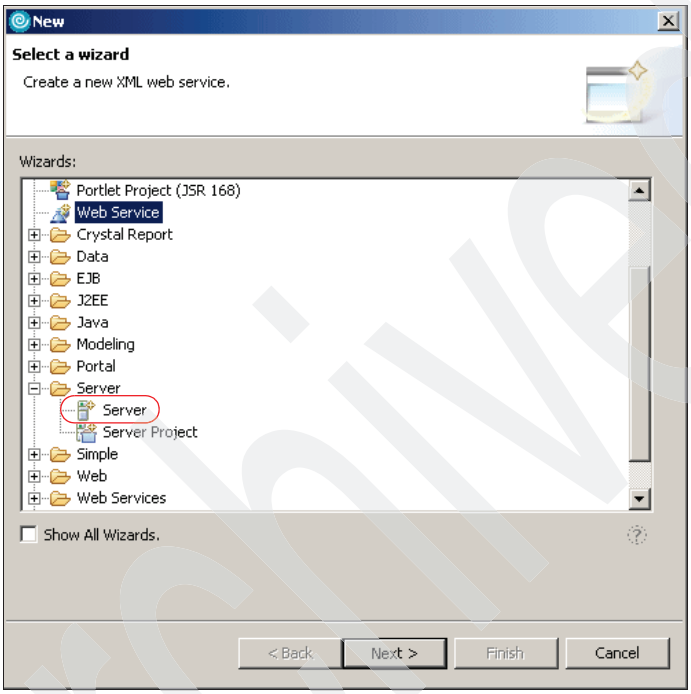


Figure 31-8 Select a Wizard window

3. In the Define a New Server window, enter the following values:
   - **Host Name**: the hostname of the remote portal server, in our scenario it is `wps51.itso.ral.ibm.com`
   - **Select the server type**: WebSphere Portal V5.1 Server Attach

Archived

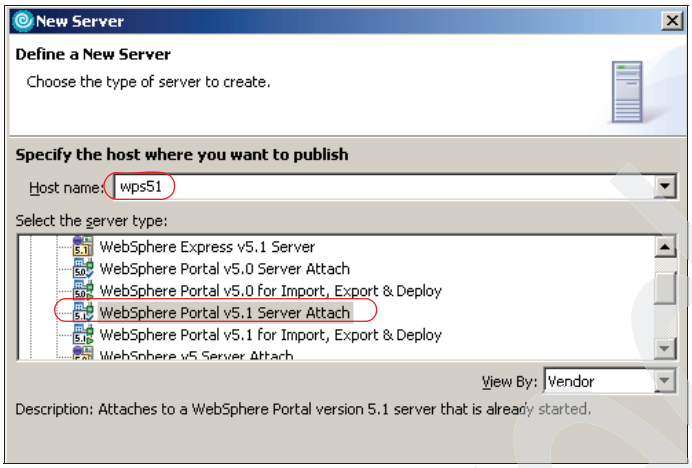


Figure 31-9 Define a New Server window

4. Click **Next.**
5. In the Server Ports window, leave the default values. Note that the JVM port value must be equal to the JVM debug port entered in the WebSphere Application Server of the remote portal. Click **Next.**

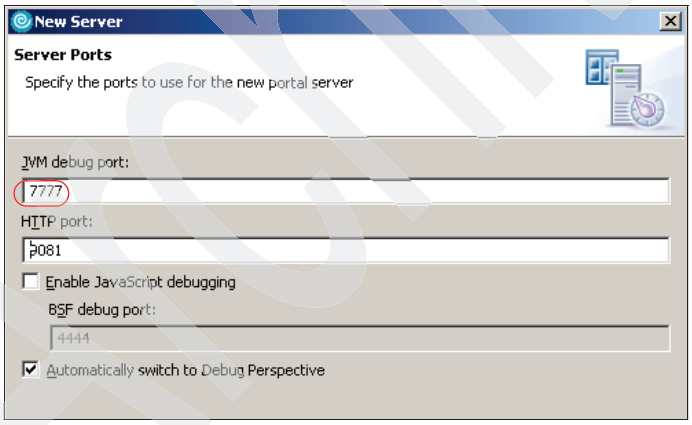


Figure 31-10 Server Ports window

6. In the next window, set the values according to the remote WebSphere Portal server configuration. You must enter the WebSphere Portal administrator user and the user you want to log on to the server for running the portlets (wpsadmin for this scenario).

Archived

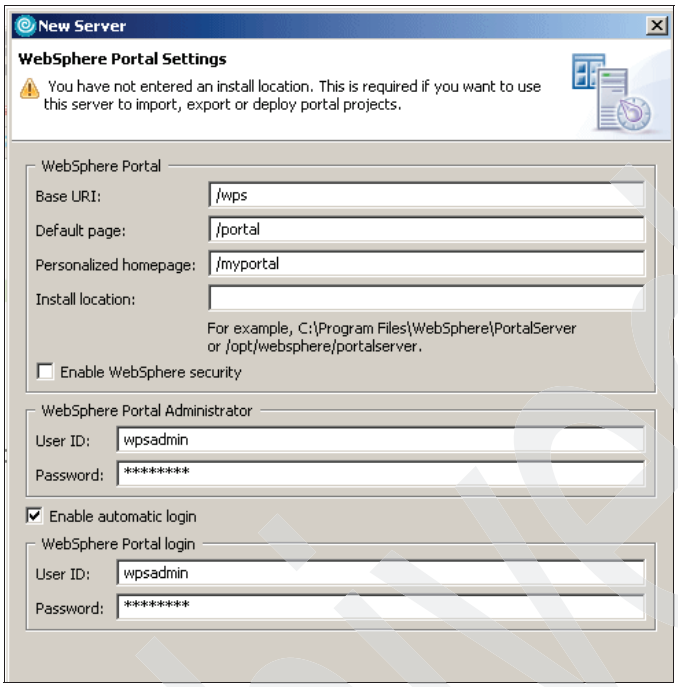


Figure 31-11 WebSphere Portal Settings

Note: The WebSphere Portal Install location field is only used for portal projects and is not covered in this scenario.

7. Click **Next**.
8. Leave the default values for the Publishing Settings window. This is information needed only for portal projects. Click **Next**.

Archive

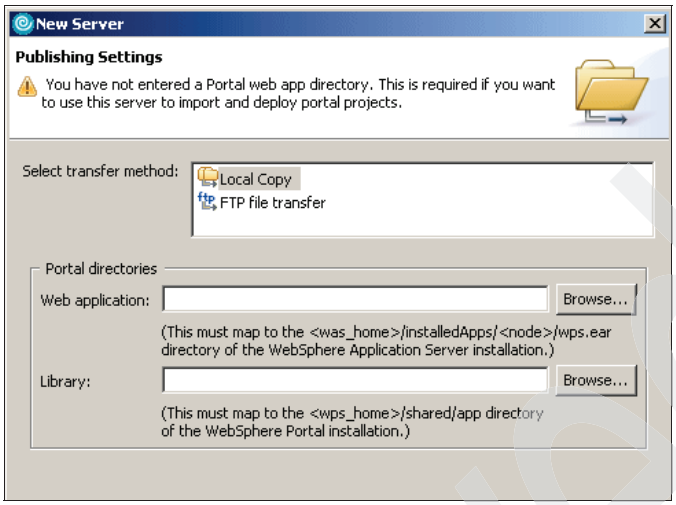


Figure 31-12 Publishing Settings

9. Click **Finish**. A new portal entry is added to the servers view.

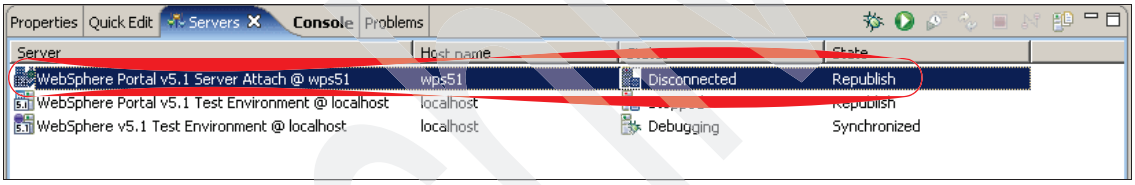


Figure 31-13 New WebSphere Portal Server Attach

31.2.4 Debugging a portlet on WebSphere Portal Server Attach

This section shows how to debug a portlet on a WebSphere Portal Server Attach. We use the `ActionEvent` portlet created in Chapter 6, “IBM Portlet API portlet development” on page 203.

Note: For more information about debugging functionality on Rational Application Developer, see Chapter 30, “Portlet debugging” on page 901.

Import the `ActionEvent` portlet provided as additional material for this redbook and follow these steps:

1. Open the `ActionEvent` portlet by double-clicking in ActionEvent/Java Resources/JavaSource/actionevent/ActionEventPortlet.java from the Project Explorer view.

2. Go to the actionPerformed method and select the line highlighted in Example 31-1.

Example 31-1 Setting a breakpoint

```
public void actionPerformed(ActionEvent event) throws PortletException {
   if( getPortletLog().isDebugEnabled() )
      getPortletLog().debug("ActionListener - actionPerformed called");
   String actionString = event.getActionString();
   PortletRequest request = event.getRequest();
   if(actionString.equalsIgnoreCase(ACTION_RED)){
      String value = "Action <FONT color=\"#ff0000\">RED</FONT>";
      PortletData portData = request.getData();
      try{
         portData.setAttribute("value", value);
         portData.store();
      }
      catch (AccessDeniedException ade){
      }catch (IOException ioe){}
   }
   if(actionString.equalsIgnoreCase(ACTION_BLUE)){
      String value = "Action <FONT color=\"#0000ff\">BLUE</FONT>";
      PortletData portData = request.getData();
      try{
         portData.setAttribute("value", value);
         portData.store();
      }
      catch (AccessDeniedException ade){
      }catch (IOException ioe){}
   }
}
```

3. Set a breakpoint in the line `portData.setAttribute("value", value);` by selecting the code line, right-clicking the marker bar (first left column of the JSP Editor) and selecting **Toggle Breakpoint**, as shown in Figure 31-14 on page 926.

Archived

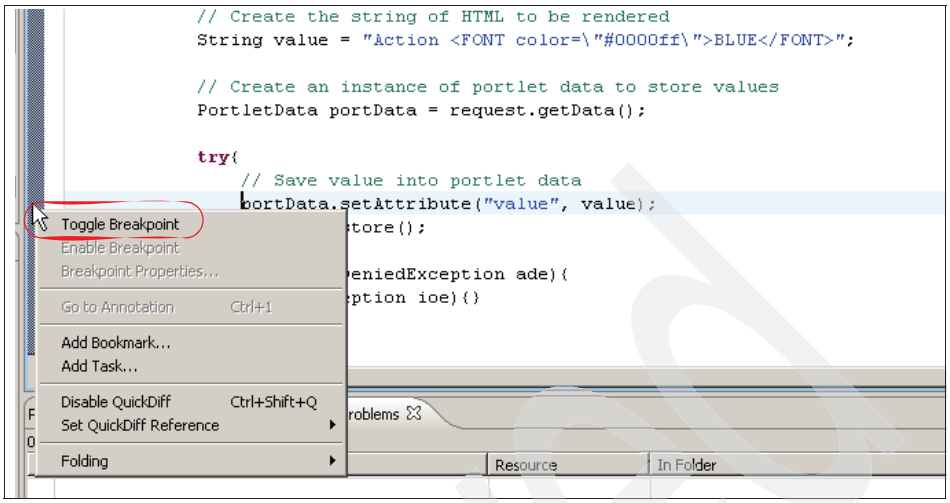


Figure 31-14 Set a breakpoint

4. A new breakpoint marker will appear on the marker bar. Enabled breakpoints are identificated with a blue circle.

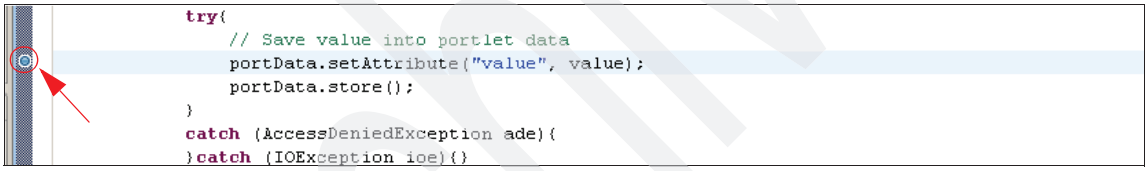


Figure 31-15 Breakpoint mark

5. From the Project Explorer view, right-click the **ActionEvent** project and select **Debug → Debug on Server...**

Archived

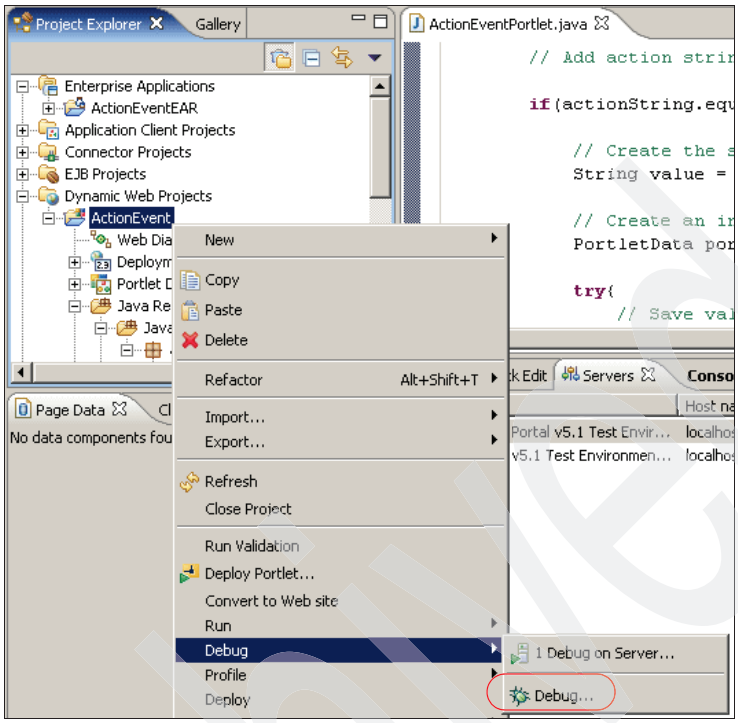


Figure 31-16 Debug on server

6. In the Define a New Server window, select **WebSphere Portal V5.1 Server Attach**.

Archived

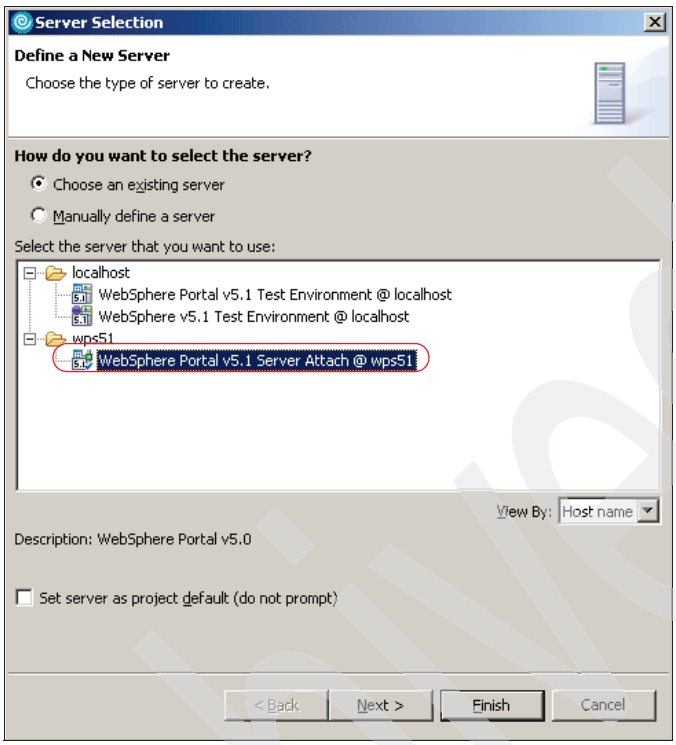


Figure 31-17 Debug on WebSphere Portal V5.1 Server Attach

7. Click **Finish**. It may take few minutes for this process to complete.
8. An internal Web browser will open within the workspace and the portlet will be shown.

Archived

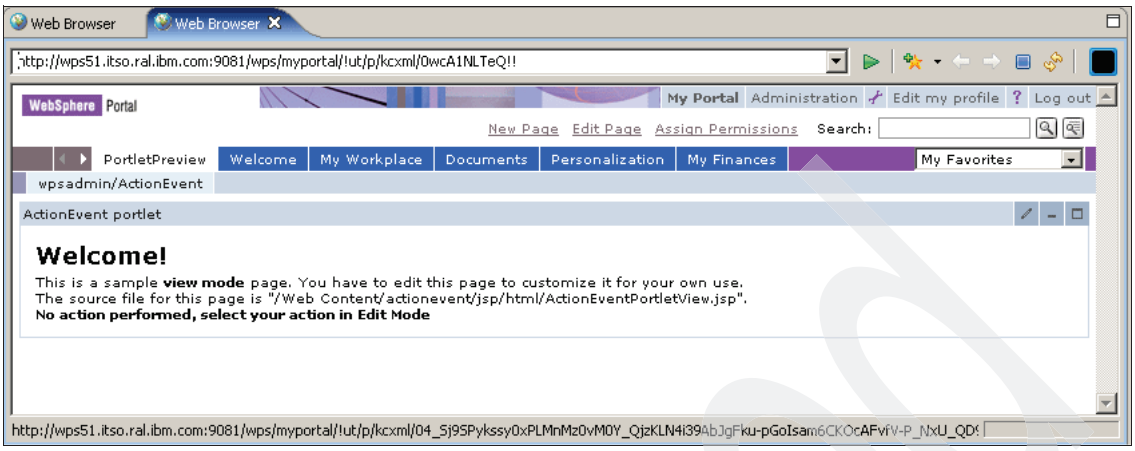


Figure 31-18 Portlet running on WebSphere Portal V5.1 Server Attach

9. Click the portlet edit icon to change the portlet mode.

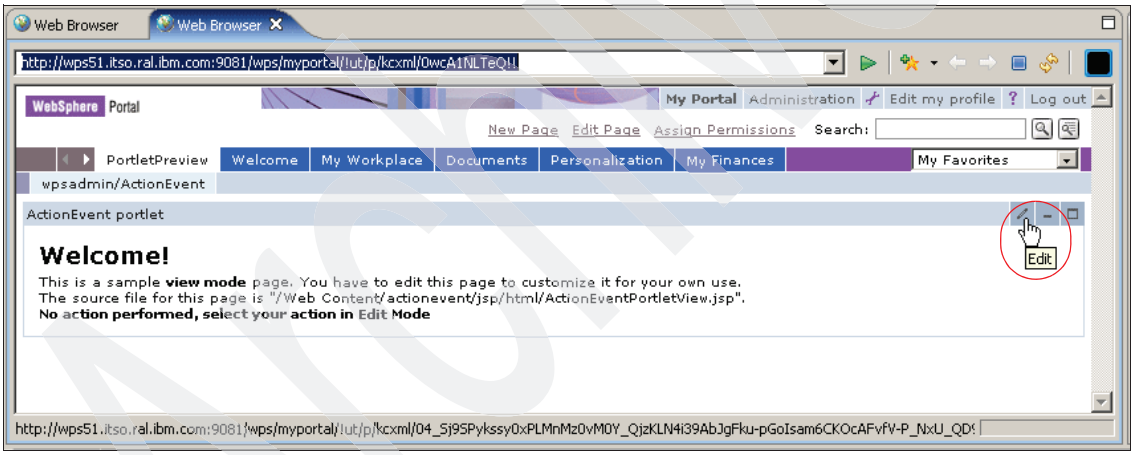


Figure 31-19 Change the portlet to Edit mode

10. Click the **Red Action** button.
11. If the following message box appears, click **Yes**.

Archived

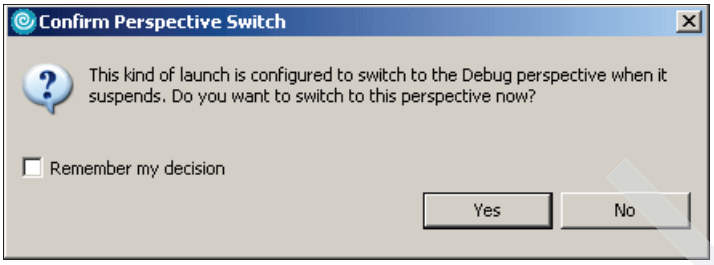


Figure 31-20 Launch the Debug perspective

12. The Debug perspective will be open and since the breakpoint is reached, the Java editor displays the code and the statement with the breakpoint highlighted.

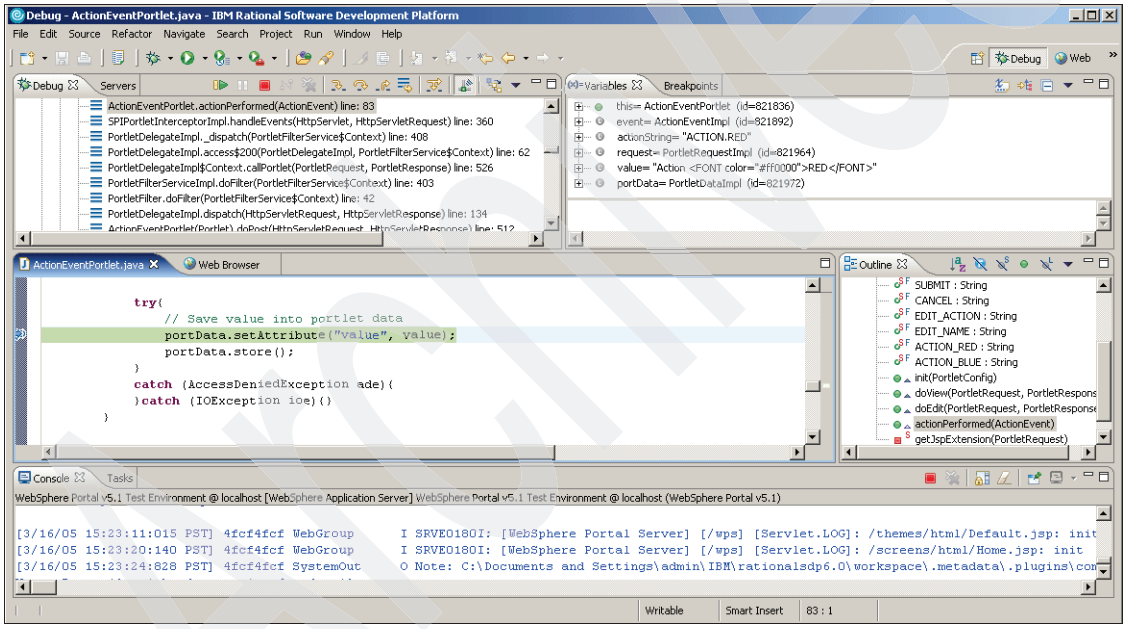


Figure 31-21 Debug Perspective

13. In the Variables tab, right-click **value="Action <FONT color="#ff0000">RED</FONT>** and select **Change value...**

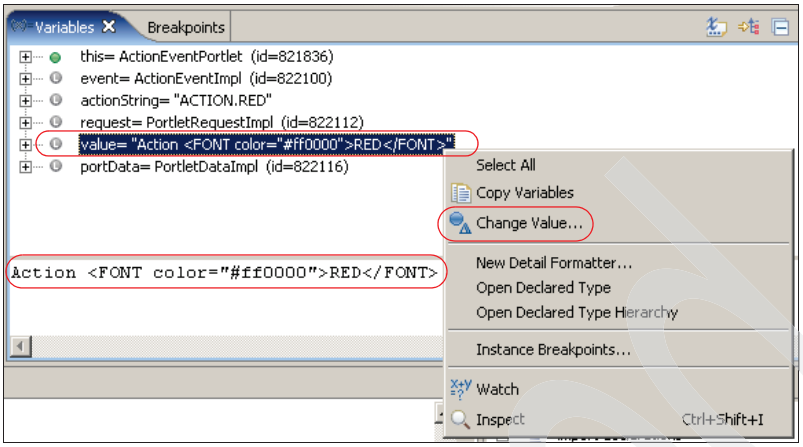


Figure 31-22 Right-click the value variable

14. In the Set Value window, change the value to:

    ```
    Action <FONT color="#00ff00">GREEN</FONT>
    ```

    as shown in Figure 31-23.

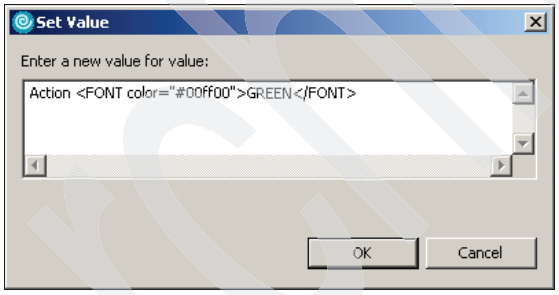


Figure 31-23 Change value

15. Click **OK**.

16. From the Debug tag, click the **Resume** button to let the portlet continues its execution.

Archived

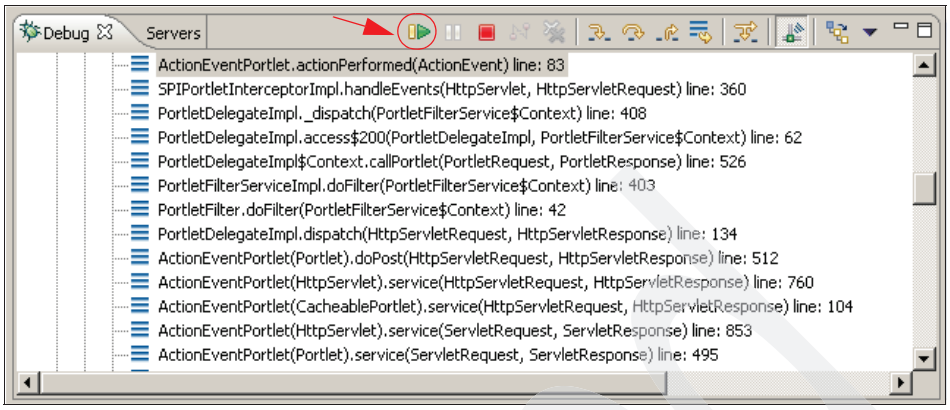


Figure 31-24 Resume portlet execution

17. View the internal Web browser where the portlet is shown. Note that the action displayed in View mode is now `Action GREEN ...was selected !`

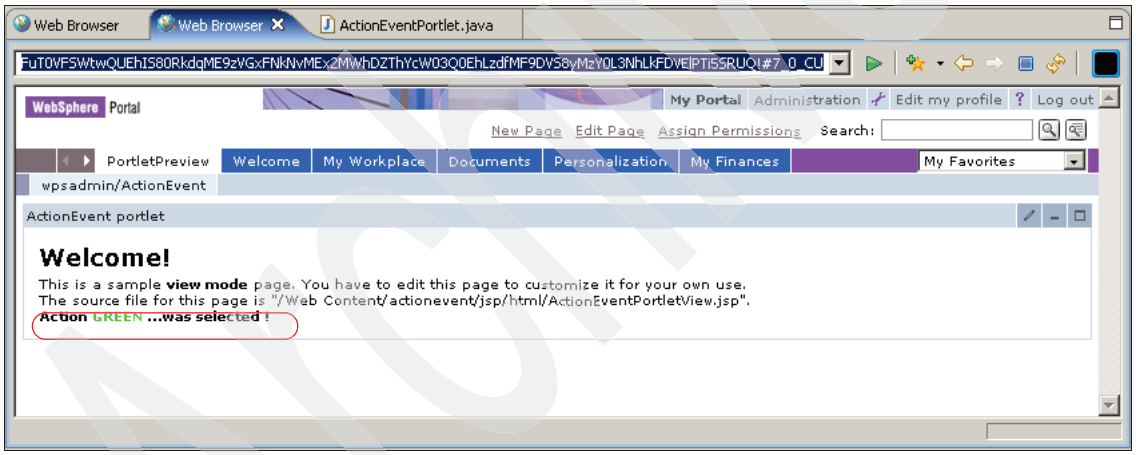


Figure 31-25 ActionEvent portlet

31.3 Defining Web browsers and emulator devices

Rational Application Developer provides an internal Web browser as part of the workbench on Windows platform. If you want to configure external Web browsers or device emulators to display your projects, do the following:

1. In the Rational Application Developer, select **Window** → **Preferences** → **Internet** → **Web browser**.

2. You can add a new Web browser or device emulator by providing a name, the location of the executable program and any parameter you want to pass to the Web browser or device emulator.
3. You can edit the values for an existing Web browser by selecting it and clicking the **Edit** button
4. You can delete a Web browser or device emulator entry by clicking the **Remove** button.
5. Select the check in the left side of the Web browser entry to indicate Rational Application Developer to use this browser.

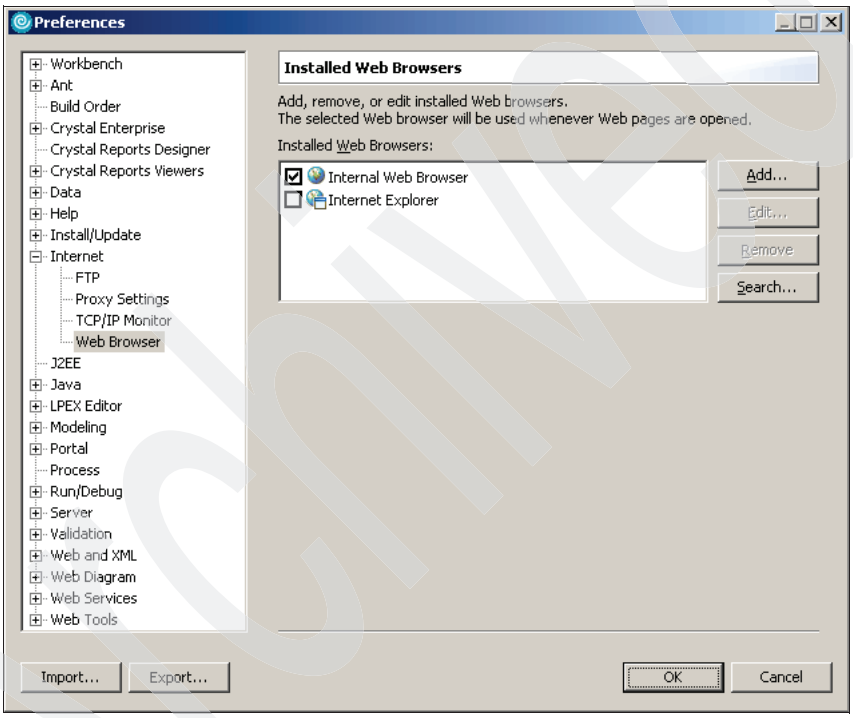


Figure 31-26 Installed Web browsers option

Archived

Archived

32

Updating a portal layout

This chapter discusses a sample scenario of a Portal project, recommendations and related issues. In this sample scenario, an existing WebSphere Portal Server V5.1 configuration will be imported into the Rational Application Developer V6 Portal project to allow modifications of the layout of the portal configuration. The updated Portal project will then be tested in the WebSphere Portal V5.1 Test Environment. The sample scenario will allow you to understand the benefits of this new feature provided in this release.

© Copyright IBM Corp. 2005. All rights reserved.

32.1 Overview

Importing a Portal configuration from an existing Portal site is one of several ways available to create a new Portal project. It is also a recommended practice since the new and updated configuration will overwrite the original Portal configuration of the Portal site where the deployment takes place; in this way, part of the default original configuration can be lost.

In this section, you will be guided to execute the following tasks:

- Create a connection to the Portal server
- Import a WebSphere Portal Server V5.1 configuration
- Modify the Layout navigation by creating additional labels, pages, etc.
- Modify the Layout content areas by adding new columns and a portlet to a page
- Change the page look and feel to use a different theme
- Test the new configuration on the WebSphere Portal V5.1 Test Environment

The development workstation components for this scenario can be seen in Figure 32-1 on page 937.

Archived

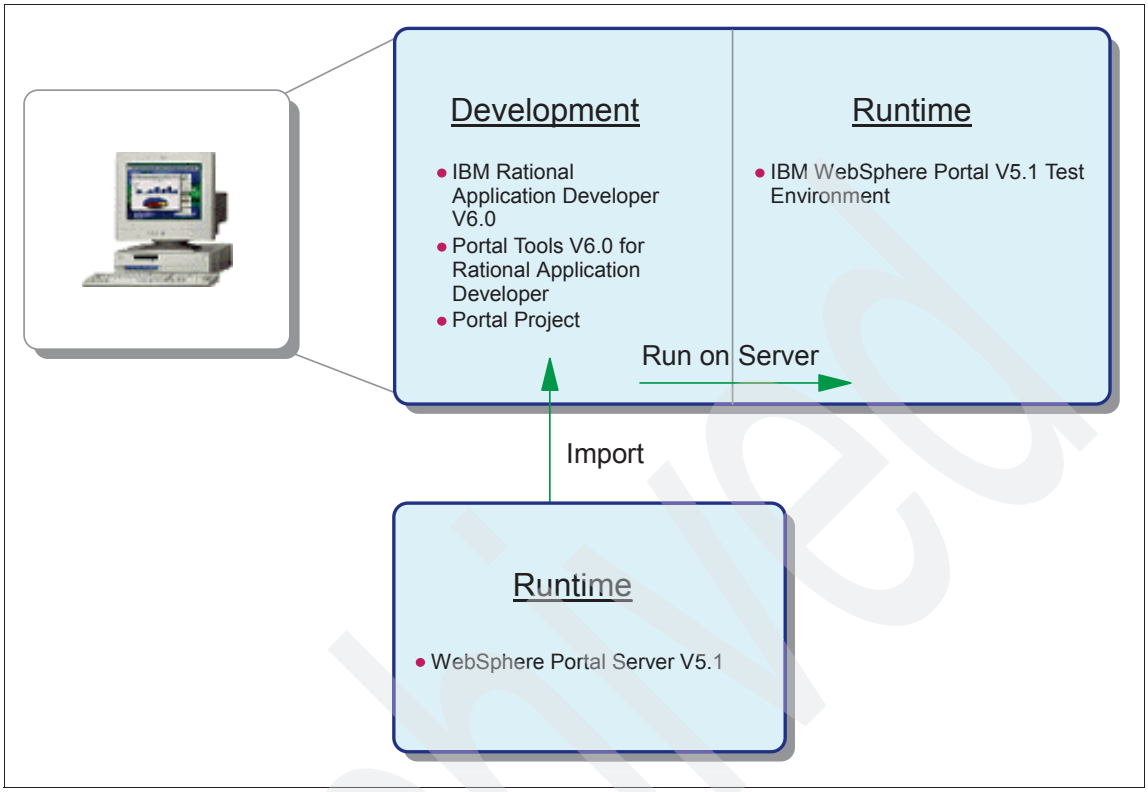


Figure 32-1 Development workstation

32.2 Creating a Portal server connection

You will need to create a server connection to be used by the Portal configuration to import, export and deploy functions.

Note: Although a connection is required when invoking the import wizard, the connection can also be created as part of the import wizard process.

As illustrated in Figure 32-1, a WebSphere Portal server with host name wps51 is used on a second machine. Therefore, the current machine where Rational Application Development V6 is running must have network access to this server machine.

For example, execute the following steps to gain access to the Portal server:

1. For this scenario, a Map Network Drive connecting to the WebSphere Portal Server V5.1 machine is created. This connection will use W as the drive letter.

   Note: Use the proper user ID and password to access the Portal server machine.

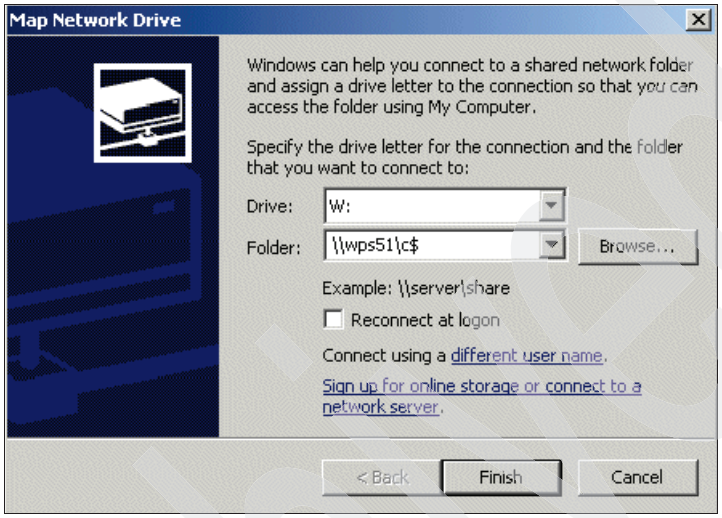


Figure 32-2 Mapping Portal server network drive

2. If not already running, start the Rational Application Developer; select **Start → Programs → IBM Rational → IBM Rational Application Developer V6.0 → Rational Application Developer**.
3. Select **New → Other** from the File menu.
4. In the *Select a wizard* window, drag down to the **Server** folder and expand it.
5. Select **Server**.

Arcitura

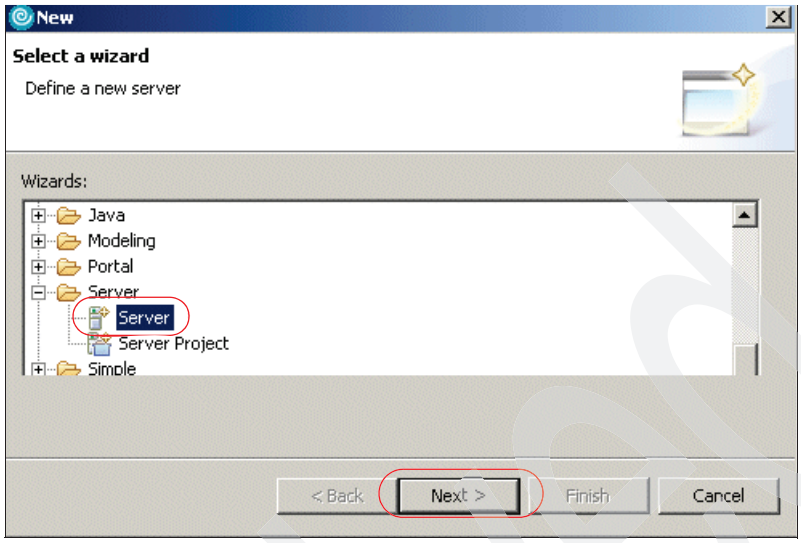


Figure 32-3 Select the Server create wizard

6. Click **Next** to define the server.
7. In the *Define a New Server* window, enter the following values:
   a. **Host name:** `wps51`. This is the host name of the WebSphere Portal server you want to import into Rational Application Development.
   b. **Select the server Type:** `WebSphere Portal V5.1 for Import, Export & Deploy`.

Archived

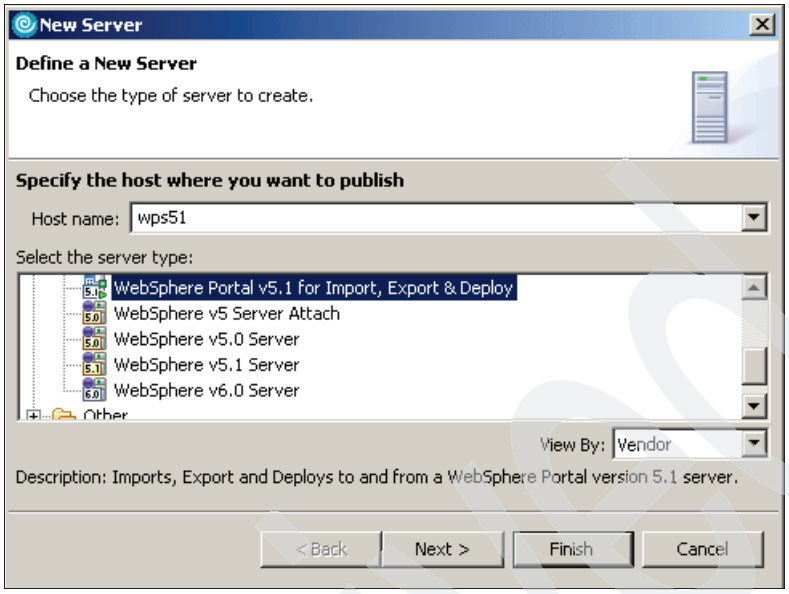


Figure 32-4 Define the target WebSphere Portal server

8. Click **Next**.
9. In the WebSphere Portal Settings windows:
   a. Except for Install location, accept default values for the fields.
   b. Install location: enter the location where WebSphere Portal is installed. In this sample scenario, the correct location is:

      w:\WebSphere\PortalServer

      Note: If your portal server uses values other than the default values, you must enter the proper paths for the fields shown in Figure 32-5 on page 941.

Archived

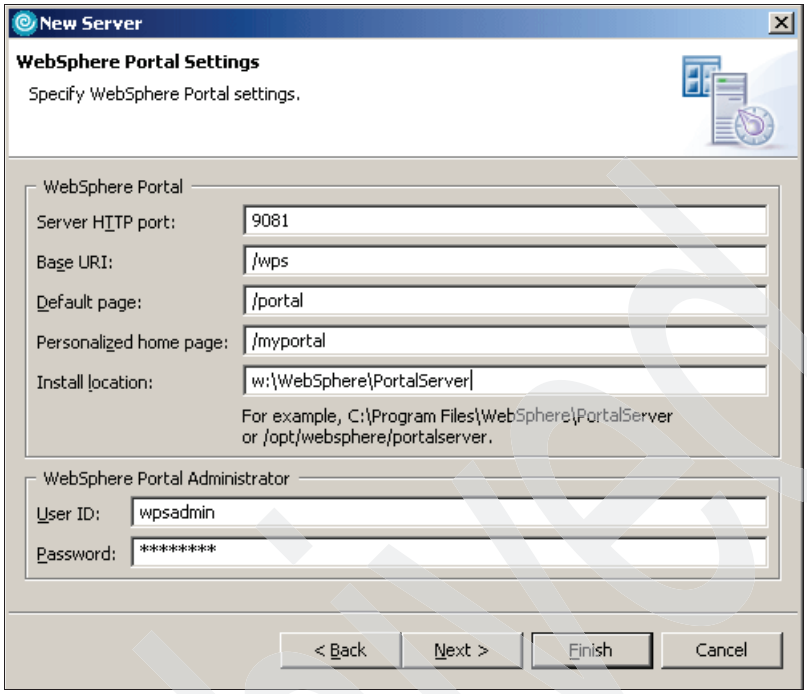


Figure 32-5 Portal settings

10. Click **Next**.
11. In the *Publishing Settings* windows, enter the following values:
    a. Transfer method can be local or FTP. For this scenario, select the following transfer method:

       `Local Copy`

    b. Web application:

       `w:\WebSphere\AppServer\installedApps\wps51\wps.ear`

    c. Library:

       `w:\WebSphere\PortalServer\shared\app`

Archived

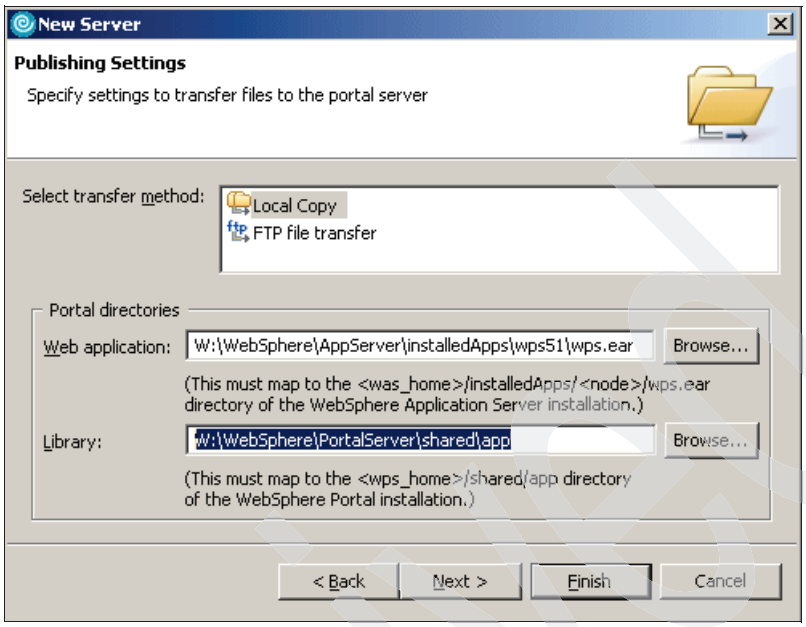


Figure 32-6 Publish Settings

12. Click **Finish** to create the server. The Add and Remove Projects configuration window is not needed to configure for the Import, Export & Deploy server connection.

13. The newly created server can now be seen in the Servers view, as illustrated in Figure 32-7.

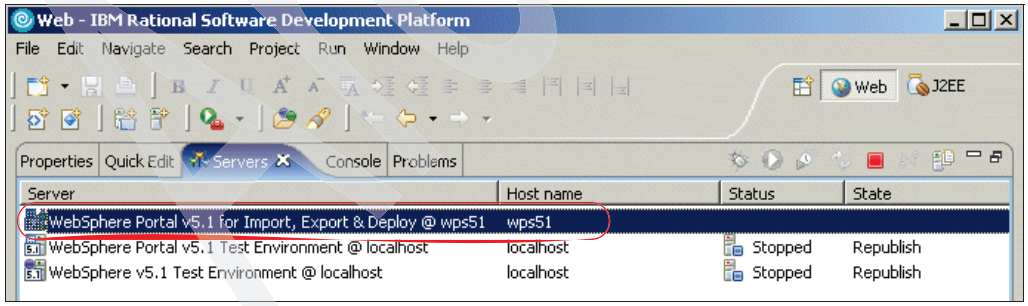


Figure 32-7 New server has been created

32.3 Importing the WebSphere Portal server configuration

Once the connection to the Portal server has been configured, you will need to create a new Portal Project. In addition, it is recommended that instead of starting with a new configuration, you import and deploy from the same existing WebSphere Portal server.

Note: Deploying a configuration that differs from the current configuration could result in a partial lost of the portal default layout and functionality since most of the current configuration will be overwritten.

Follow this procedure to import a portal site from the WebSphere Portal server:

1. Select **File** → **Import**.
2. From the Select window, choose **Portal and** click **Next**.
3. In the Import Portal Project window, enter the following values:
   a. In *Select the portal server that you want to use*, select the **WebSphere Portal V5.1 for Import, Export & Deploy @ wps51** server that we created in the previous process.
   b. In *Portal project*, enter `TestPortalServer`.

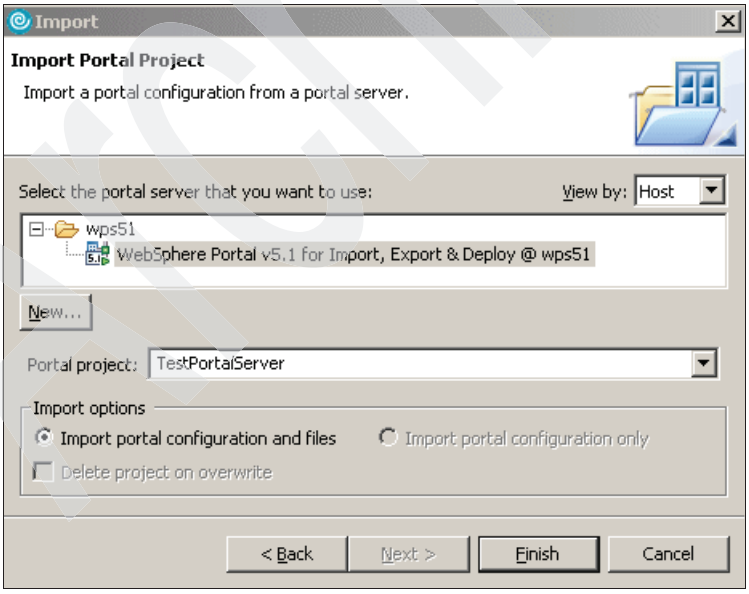


Figure 32-8 New Portal Project

4. Click **Finish**.
5. You will be prompted with a message indicating that this process may take a long time and it should not be interrupted.
6. Click **Yes** to continue.
7. A status bar at the bottom of the current window shows the process status as shown in Figure 32-9.

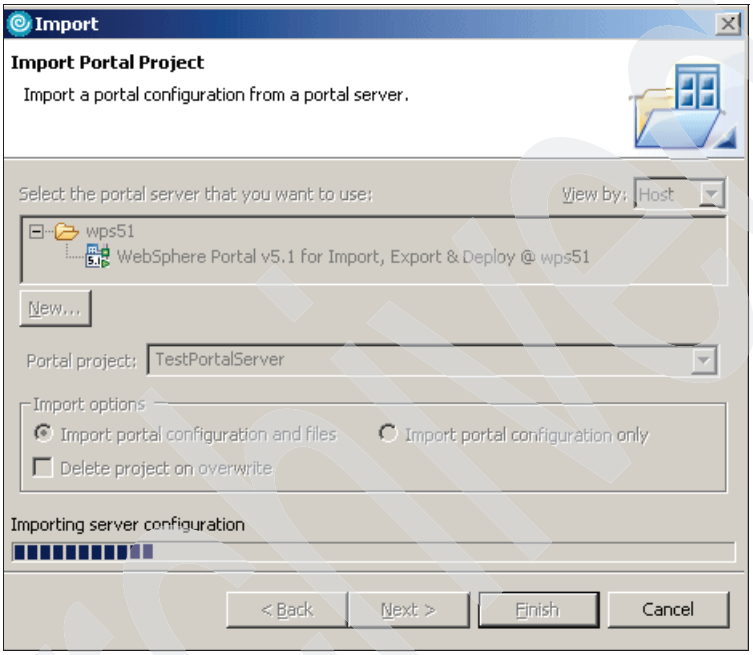


Figure 32-9 Importing the target server configuration

8. When the process finishes you can verify that `TestPortalServer` has been created in the Dynamic Web Projects folder**.** Also, the following EAR files have been created in the Enterprise Applications folder:
   - `TestPortalServerEAR`
   - `TestPortalServerPortletsEAR`

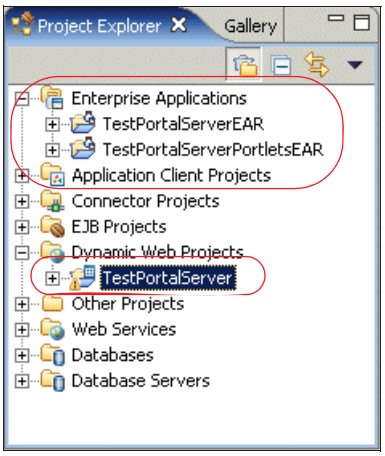


Figure 32-10 Test Portal server

32.4 Modifying the Portal Navigation and Layout

Now that the Portal configuration has been imported into Rational Application Development, you will make the following modifications:

- Navigation section. Add new labels and pages.
- Layout. Modify the portlet content areas within a portal page and change the themes.

Adding navigation

Execute the following instructions:

1. Expand the **TestPortalServer** under Dynamic Web Projects by clicking the plus symbol next to it and double-click **Portal Configuration.**

Archived

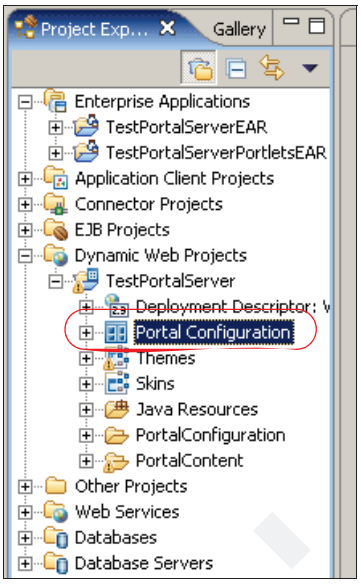


Figure 32-11 Portal configuration

2. The Portal Designer editor will open in the center of the workbench. This view serves as the design editor for your portal project. The Portal configuration shows the WebSphere Portal Server V5.1 layout as illustrated in Figure 32-12.

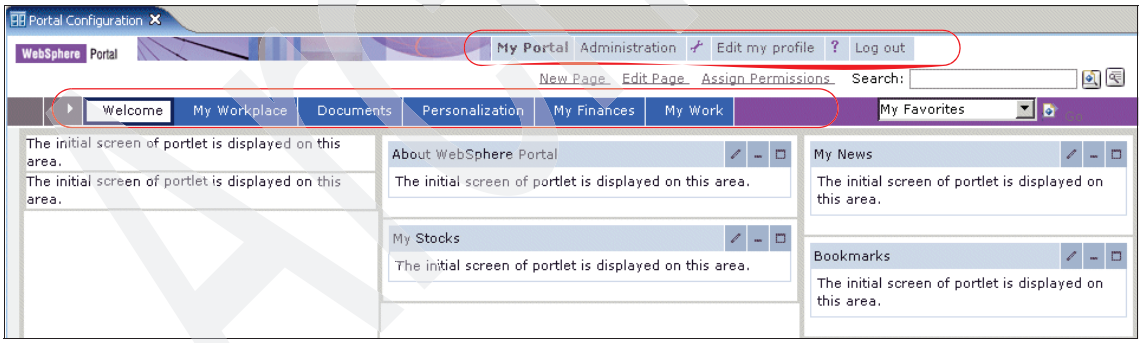


Figure 32-12 WebSphere Portal server configuration

3. Add a new Label next to the Welcome page
   a. Click **Label** from the palette to create a new label.
   b. Drag it to the portal configuration on the Portal Designer and next to the Welcome label as shown in Figure 32-13 on page 947.

Archived

Tip: Once you position the cursor arrow, a plus symbol appears meaning that you can create the label at that location.

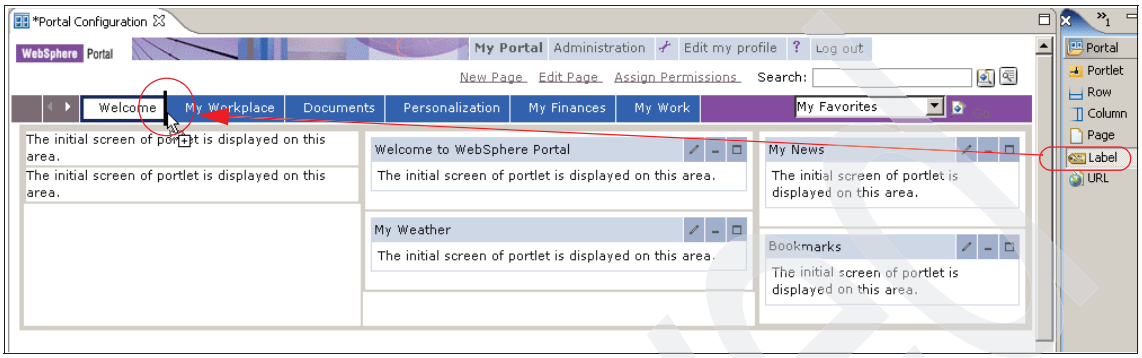


Figure 32-13 Drag a label to the portal configuration

c. In the Properties view, select the **Title tab.**

d. Change the Title from `New Label` to `WebSphere` as shown in Figure 32-14.

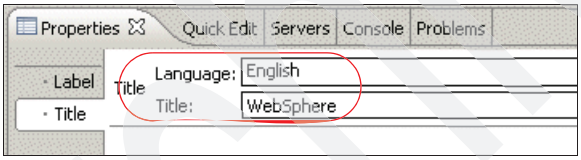


Figure 32-14 Changing the Label name

e. Select **File** → **Save** to save the new values.

4. Create a new page

a. In the Outline view, right-click the `WebSphere` label you just created.

b. Select **Insert Page** → **As Child.**

Archive

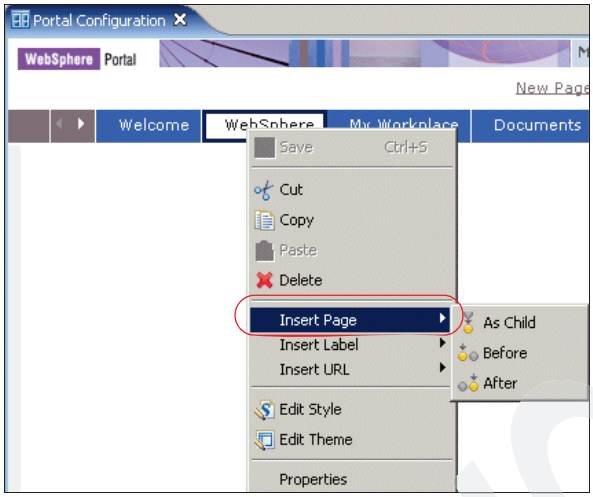


Figure 32-15 Inserting a new page as a child

5. When a New Page is created the Outline takes you to the Layout root node, change to the Navigation root node so you can see the result hierarchy for the `WebSphere` label. For WebSphere Portal V5.1 portal projects, `Content Root` is the root navigation node. Select **New Page** in the outline view.

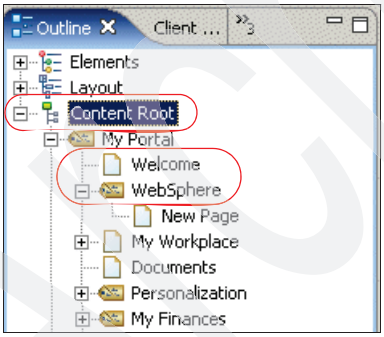


Figure 32-16 WebSphere Portal root node

6. In the properties view, select **Title tab** and change the **Title** from `New Page` to `Sample`.

Archived

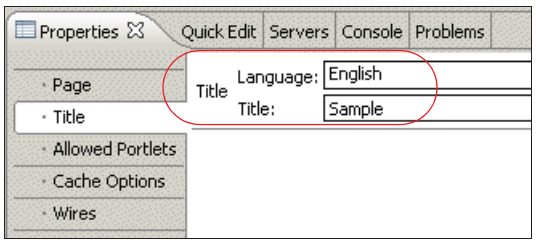


Figure 32-17 Changing the name of a page

7. Select **File** → **Save** to save the new values. The resulting hierarchy will now look as shown in Figure 32-18.

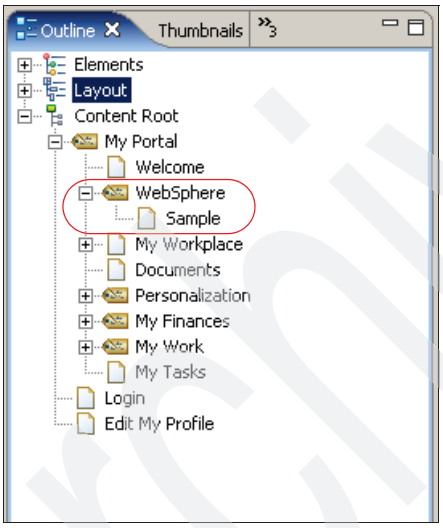


Figure 32-18 Resulting hierarchy

Changing the page layout

Execute the following steps:

1. Select the previously created `Sample` page.
2. Highlight the existing column where it says `Place portlet here`.

   Note: When a new page is created a Parent Row and a Child Column are automatically created.

3. From the Palette, select **Column** and drag it next to the current column in the page, as shown in Figure 32-19 on page 950. Notice the black vertical mark

Archived

next to the highlighted column indicating where the new column will be placed.

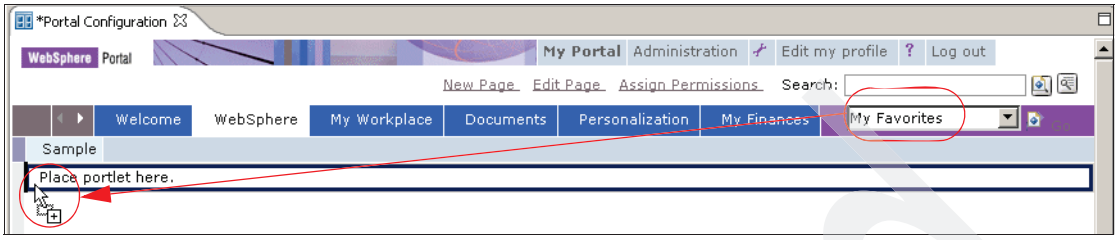


Figure 32-19 Adding a new column to a page

4. After adding the second column, the result should look as illustrated in Figure 32-20.

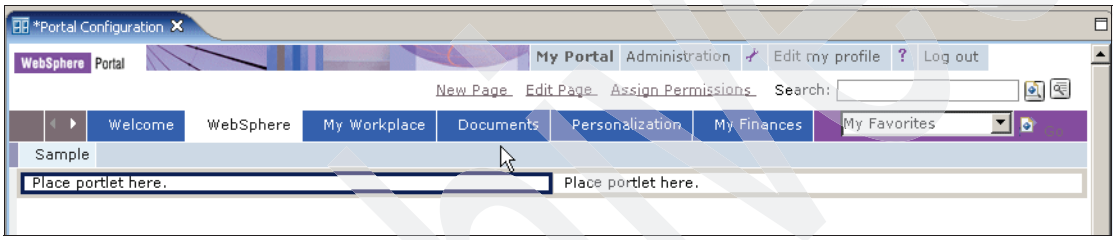


Figure 32-20 Resulting page layout

5. Select **File** → **Save.**

   Note: The column widths can be changed using the mouse pointer, or by setting new values in the Properties view.

32.5 Adding portlets

A portlet will be added to the new page. First of all, the portlet needs to be added to the workbench. The portlet that will be used for this sample scenario is a simple portlet that was previously created. The portlet name is HelloWorld and is available as WAR file.

Importing a portlet

Execute the following steps to import the HelloWorld sample portlet:

1. Select **File** → **Import**
2. From the Select window, select **WAR file**.

3. Click **Next**.
4. In the WAR Import window enter the following values:
   a. WAR file. The location of the portlet WAR file. For example:

      `c:\LabFiles\Portal Project\HelloWorld.war`

   b. Web Project. This field will be automatically set when the WAR file is selected. `HelloWorld` in this sample scenario.
   c. Target server: `WebSphere Portal V5.1 stub`.
   d. Take defaults for other values.
5. Click **Finish**.
6. Select **File** → **Save**

Adding portlets using the Drag and Drop feature

Execute the following steps to add the HelloWorld sample portlet:

1. In the Portal Designer select the `Sample` page.
2. Select **Portlet** from the palette and drag it to the left column, as shown in Figure 32-21.

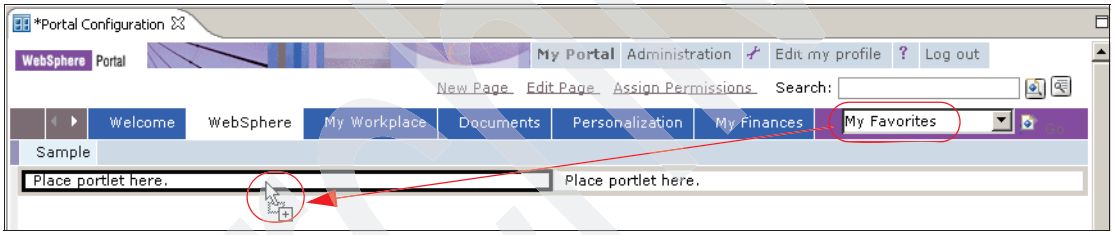


Figure 32-21 Dragging portlets

3. The Insert Portlet dialog box appears, select the **HelloWorld** portlet.

   Note: All portlets associated with the workspace will appear in the Insert Portlet dialog box.

Archived

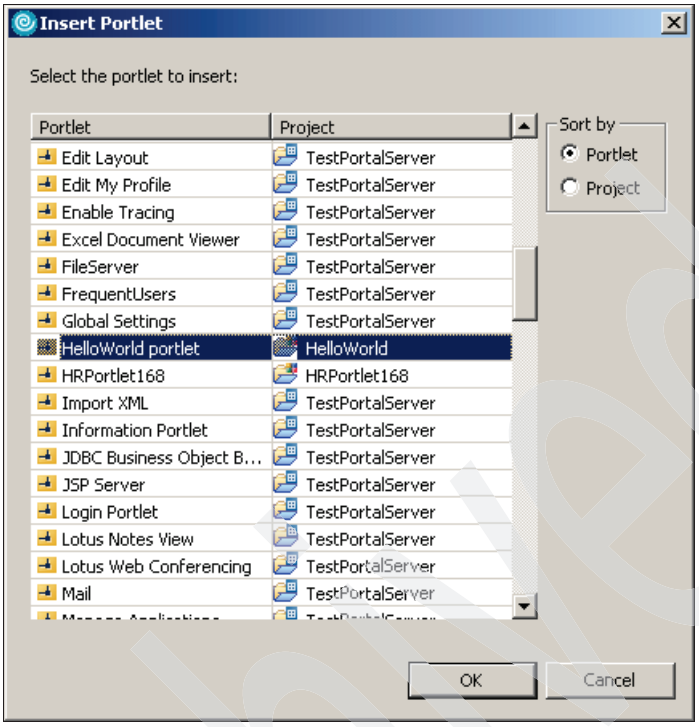


Figure 32-22 Select Portlet window

4. Click **OK** and select **File** → **Save.**

> **Tip:** Adding portlets to a page can also be done by right-clicking the column or row where the portlet is to placed and selecting the `Insert Portlet` option.

32.6 Additional ways to add portlets

In this section we illustrate other alternatives to add portlets to a page. If you do not want to review this, skip to 32.7, “Testing the updated portal configuration” on page 954.

Using the Insert menu option

The following procedure can be used to add portlets to a page using this method:

1. Select a Page from the Portal Designer.
2. Highlight the column by clicking where you want the portlet.

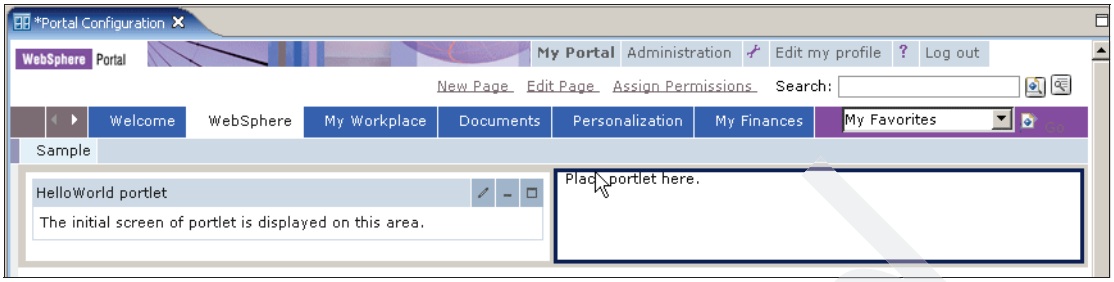


Figure 32-23 Highlighted column

3. From the menu bar, click **Insert** → **Portlet** → **As Child...**

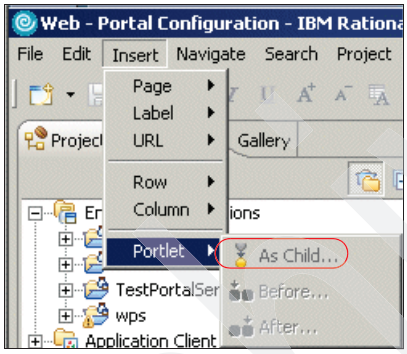


Figure 32-24 Adding a portlet from Insert menu

4. The **Insert Portlet** dialog box appear, select the portlet.
5. Click **OK**.
6. Select **File** → **Save**

Using a column right-click

The following procedure can be used to add portlets to a page using this method:

1. Select the page.
2. Highlight the column where you want the portlet.
3. Right-click the column and select the **Insert Portlet** → **As Child...**
4. The **Insert Portlet** dialog box appears, select the portlet.
5. Click **OK**.
6. Select **File** → **Save.**

Archived

Using the Outline view

The following procedure can be used to add portlets to a page using this method:

1. In the Layout node of the `Outline view`, select the column.
2. Right-click the column and select the **Insert Portlet** → **As Child...**

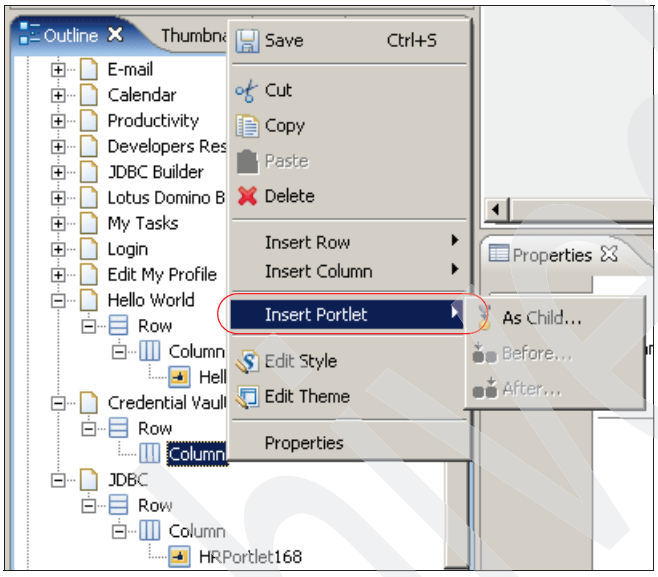


Figure 32-25 Adding portlets from the layout node in the Outline view

3. The **Insert Portlet** dialog box appears, select the portlet.
4. Click **OK**.
5. Select **File** → **Save.**

32.7 Testing the updated portal configuration

A Portal Project can be tested and debugged locally in the Local Test Environment or in a Remote Server Attach server. In this sample scenario, the updated Portal Project `TestPortalServer` will be locally tested using the Local Test Environment for WebSphere Portal Server V5.1. If you need more details about this topic see for example the *Rational Application Developer's Help Contents.*

Follow this procedure to test the portal project:

1. From Project Explorer, right-click **TestPortalServer**.

Archived

2. Select **Run** → **Run on Server...**

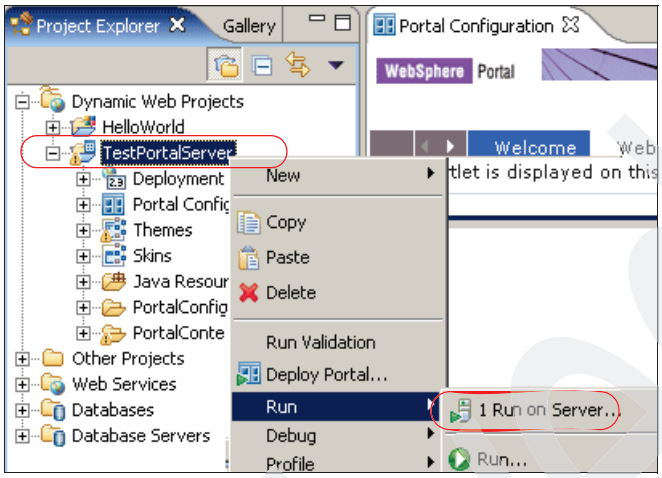


Figure 32-26 Run on Server

3. In the **Define a New Server** window select **WebSphere Portal V5.1 Test Environment**. Click **Finish**.

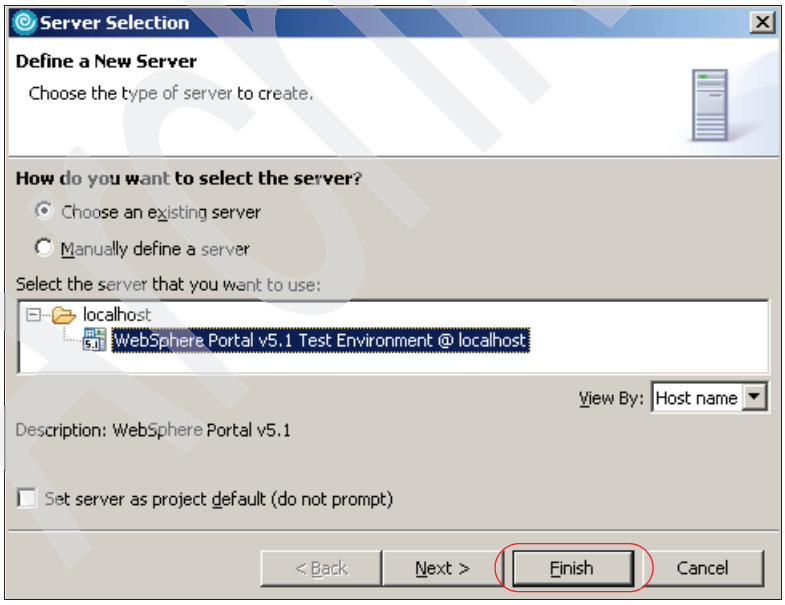


Figure 32-27 Select the Portal Test Environment Server

4. A Repair Server Configuration will appear, click **OK**.

Archived

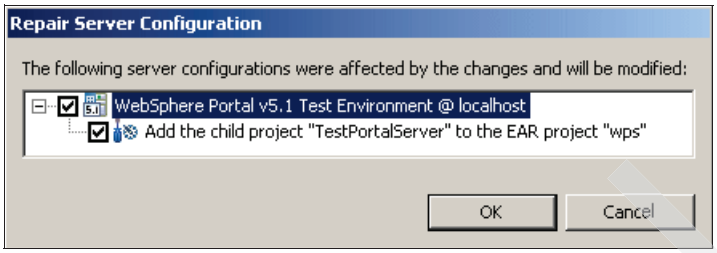


Figure 32-28 Repair Server Configuration

5. The internal Web browser will open within the Rational Application Developer workbench, showing the Portal Server Welcome page.

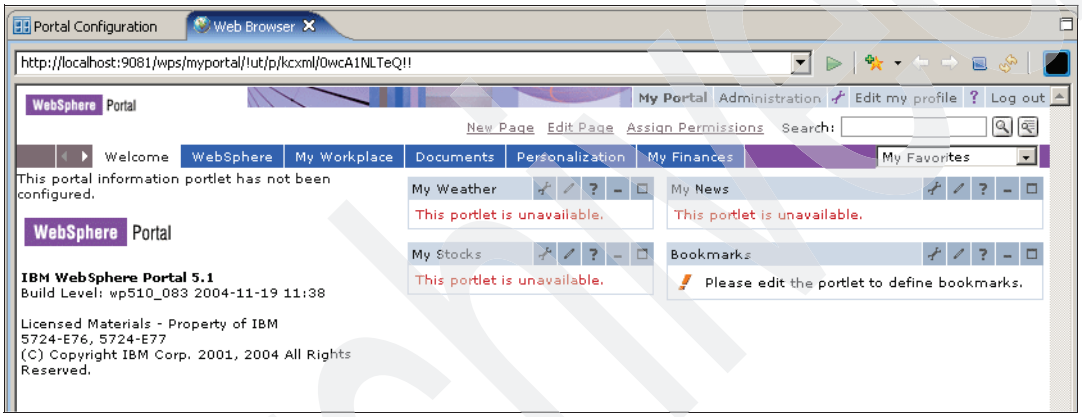


Figure 32-29 WebSphere Portal Server V5.1 Welcome page

Note: When you run a portal project in the local Portal Test Environment, some portlets may display error messages.

6. Click the **WebSphere** page you created.



Figure 32-30 WebSphere page

7. The new label, page and current theme are shown.

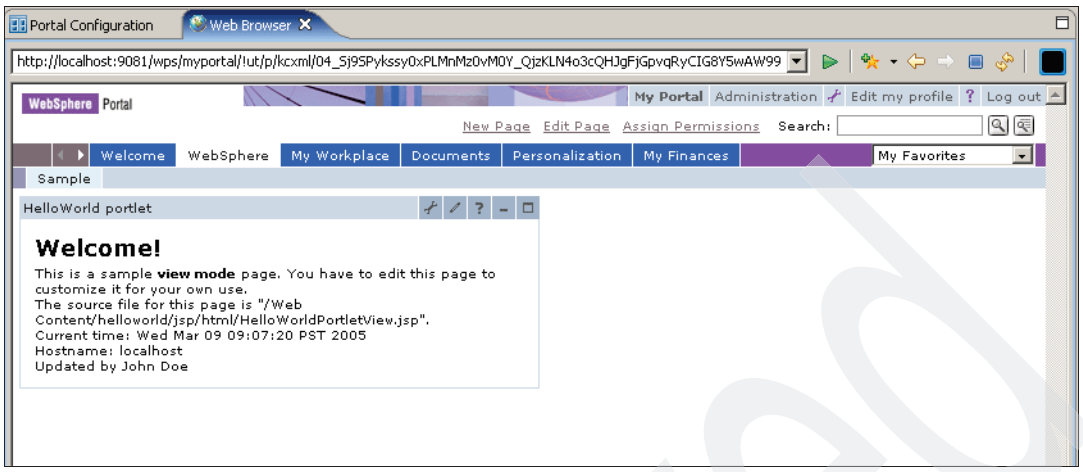


Figure 32-31 Portal page and added portlet

8. Stop the Portal server. In the Servers view right-click the WebSphere Portal V5.1 Test Environment and select **Stop.**

32.8 Applying themes

In this section you will apply new Themes to your Portal Project.

1. Select the **WebSphere** label from the Portal Configuration.
2. In the Properties view select the **Label** tab.
3. Click the Theme list box to display it and select the **Finance** theme.

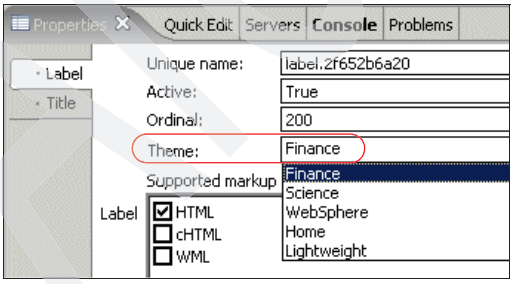


Figure 32-32 Label properties

4. Wait for the Portal Configuration to display the change.

Archived

Note: Child pages for this label will also be displayed with the Finance theme.

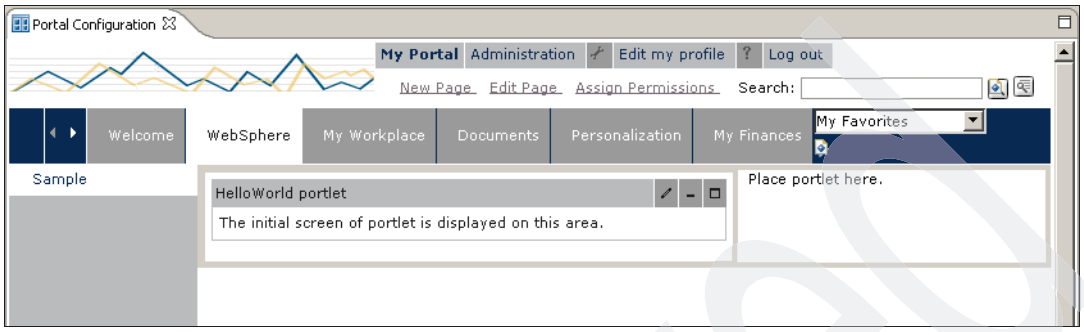


Figure 32-33 Finance theme

5. Select **File** → **Save** to save the new property values.
6. From Project Explorer right-click the **TestPortalServer** and select **Run** → **Run on Server...**
7. In the **Define a New Server** window select `WebSphere Portal V5.1 Test Environment`
8. Click **Finish**. Wait until the browser is started showing the WebSphere Portal Server again.
9. Select the **WebSphere** label. The Finance theme that you previously selected will now be shown.

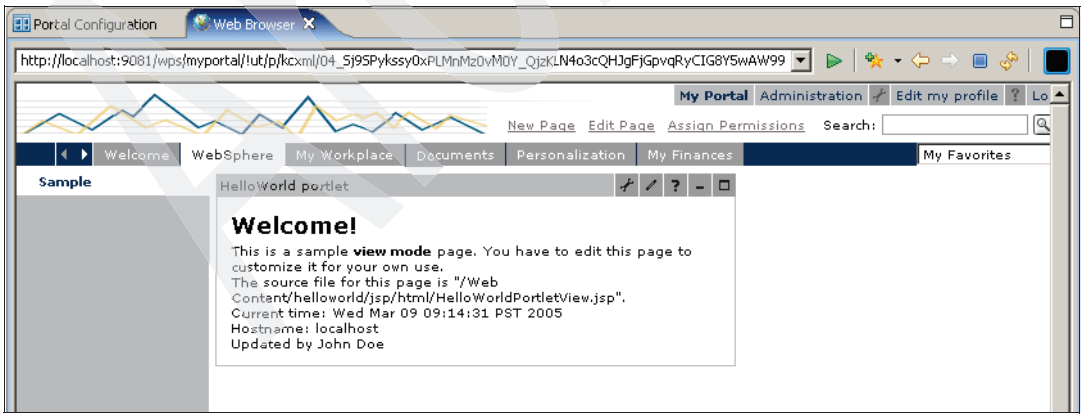


Figure 32-34 Page showing the Finance theme

10. Stop the server. In the Servers view right-click the WebSphere Portal V5.1 Test Environment and select **Stop.**

Applying the Corporate theme (optional)

As an option, repeat the previous procedure to apply the Corporate theme to the Portal Configuration.

1. In the Label properties, select the Corporate theme.
2. After applying the Corporate theme, the page should look as shown in Figure 32-35.

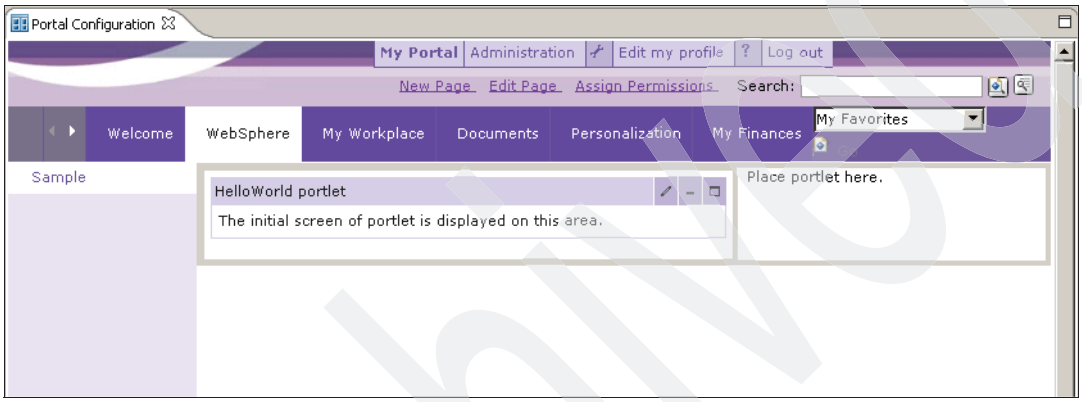


Figure 32-35 Corporate theme

3. Test again the Portal Project, the result should look as illustrated in Figure 32-36.



Figure 32-36 Page with the selected Corporate theme

Archived

33

Creating new portal themes

This chapter guides you through the process of creating and modifying new themes to customize the look and feel of a Portal site with Rational Application Developer V6. The sample scenario from Chapter 32, “Updating a portal layout” on page 935 will be used to apply new themes generated and it will then be tested in a WebSphere Portal V5.1 server.

Archived

© Copyright IBM Corp. 2005. All rights reserved.

33.1 Overview

In this section, you will be guided to execute the following tasks:

- Create a new theme
- Edit a new theme.
- Edit styles
- Apply new themes and skins.
- Test the new configuration on WebSphere Portal V5.1 Test Environment.
- Export and publish the Portal project from Rational Application Developer to WebSphere Portal.

33.2 Creating a new theme

Themes provide the look and feel of a portal site. Rational Application Developer provides a basic set of themes and skins. You can use these for your site, or you can use them as base to create new themes and skins to design a unique appearance for your site.

In this sample scenario, we use the Corporate theme provided by RAD to create a new Theme. Follow these steps:

1. From the Project Explorer, right-click the **TestPortalServer** portal project.
2. Select **New → Theme.**
3. In the New Theme window, enter `MyCorp` as title and select **Corporate** as the source theme.

Archived

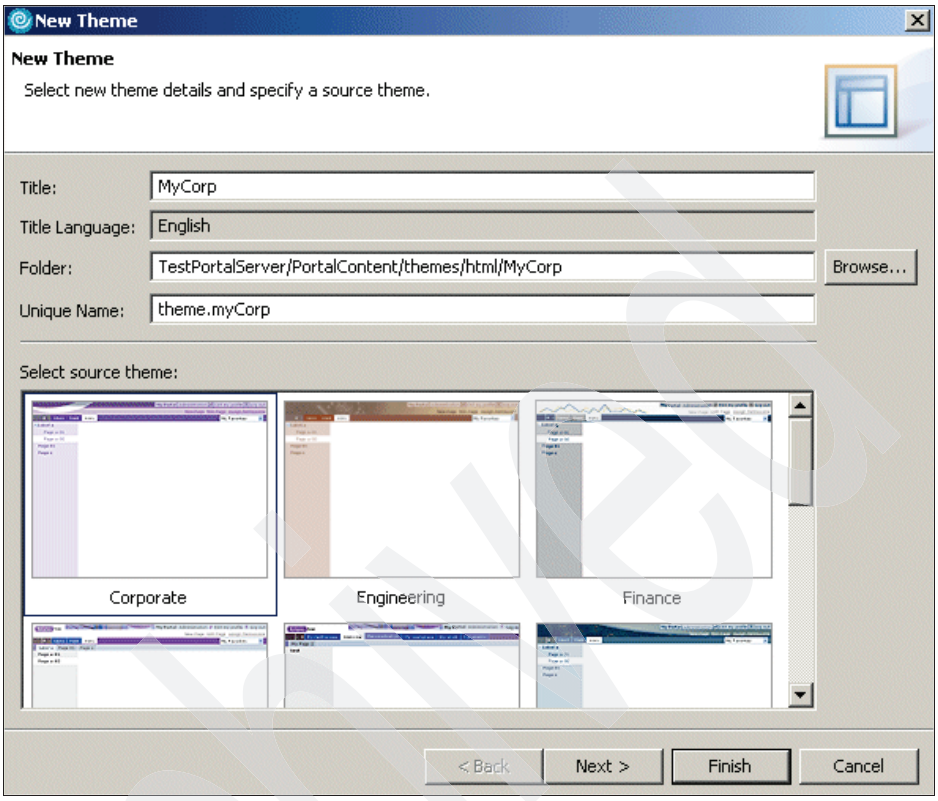


Figure 33-1 Creating a New Theme window

4. Click **Next**.
5. In the Select Skins window, do the following:
   a. Select **Shadow**, **Outline**, **Clear** and **Fade** as skins allowed for this theme.
   b. Select the **Outline** skin and click **Set as Default Skin**.
   c. Click **Finish**.

Archived

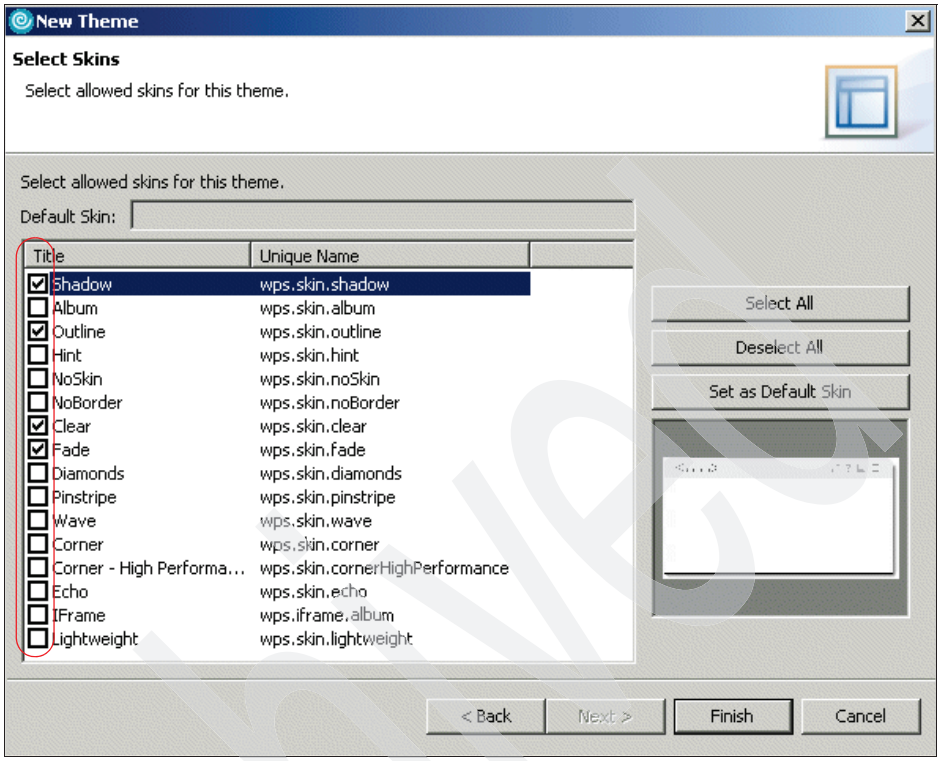


Figure 33-2 Select skins for the theme

6. As a result you should find a new theme entry in the folder Themes under TestPortalServer, as shown in Figure 33-3 on page 965. The associated files will be under PortalContent\themes\html\MyCorp.

Archive

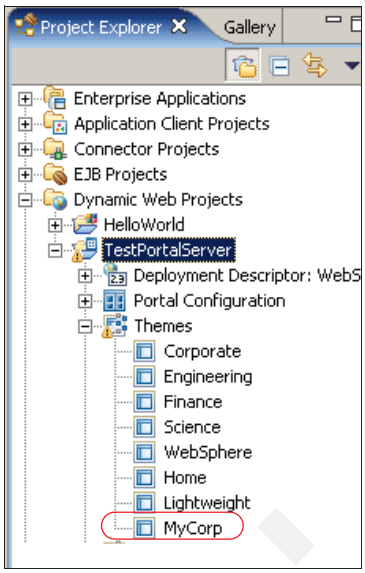


Figure 33-3 Generated new Theme

33.3 Editing a theme

To edit the `MyCorp` theme using Rational Application Developer, follow these steps:

1. From the Project Explorer, double-click **MyCorp** located in **Dynamic Web Projects** → **TestPortalServer** → **Themes**.
2. The Page Designer will show the Default.jsp file.

Archived

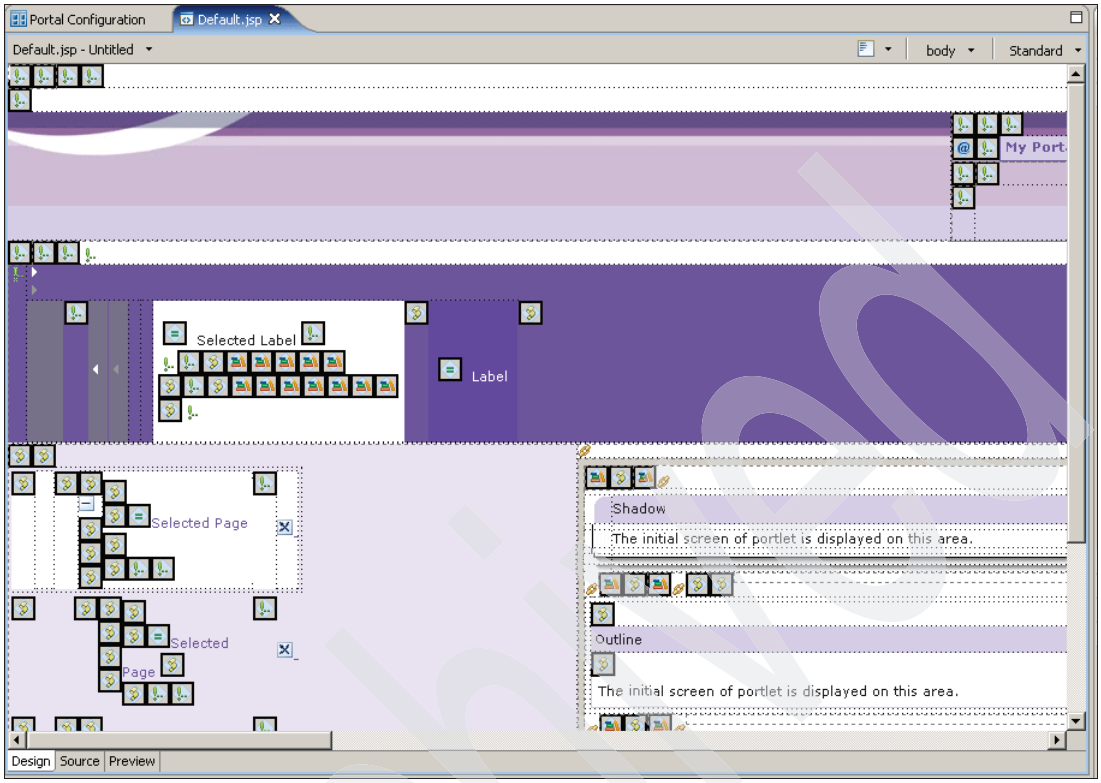


Figure 33-4 Default.jsp file

3. By default, the Page Designer shows the JSP tags with symbols. You can hide these tags as follows:

   a. From the menu, select **Window** → **Preferences** → **Web Tools** → **Page Design** → **Editing Symbols.**

   b. Select **None** and click **OK** as illustrated in Figure 33-5 on page 967.

Archived

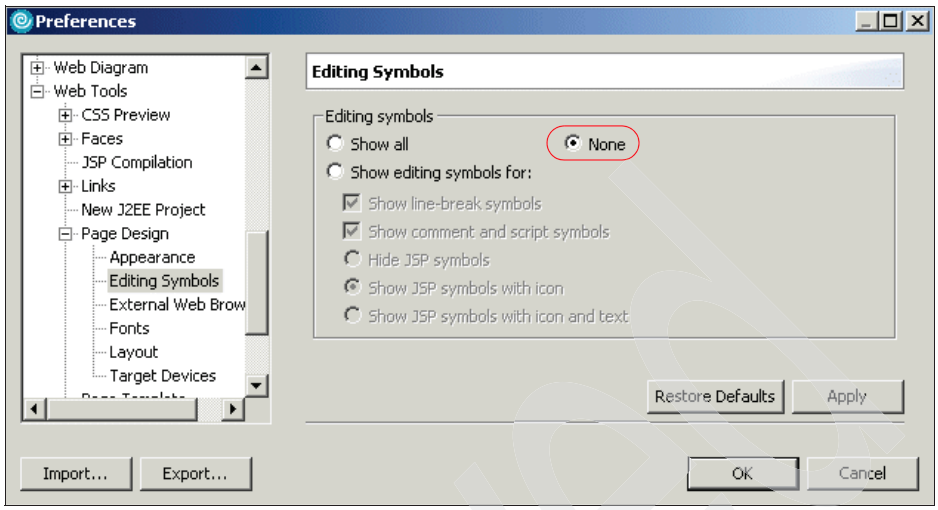


Figure 33-5 Editing symbol preferences

4. Remove the following design elements from the theme using the Page Design:
   a. Select `My Favorites` combo box and press the **Delete** key.
   b. Repeat this procedure to delete the little `blue icon` (to the right of `My Favorites` combo box).
   c. Repeat the same procedure to delete the label `Go` (to the right of the `blue icon`).

Archive

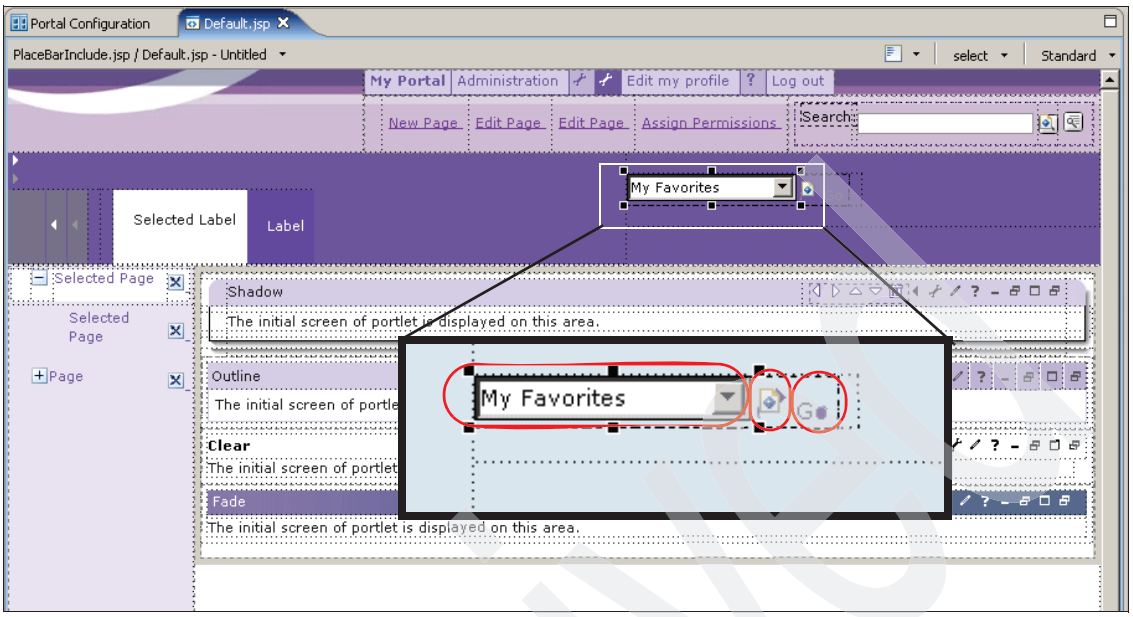


Figure 33-6 Delete My Favorites combo box, blue icon and Go label

5. Select **File** → **Save.**
6. Use the File system option to import two images that will be used in this scenario:
   a. Select **File** → **Import** → **File system**.
   b. Select the files `d-w_logo.gif` and `line.gif` as shown in Figure 33-7 on page 969.
   c. Click **Finish** to import the files into the following subdirectory:

      `TestPortalServer\PortalContent\themes\html\MyCorp`

Archive

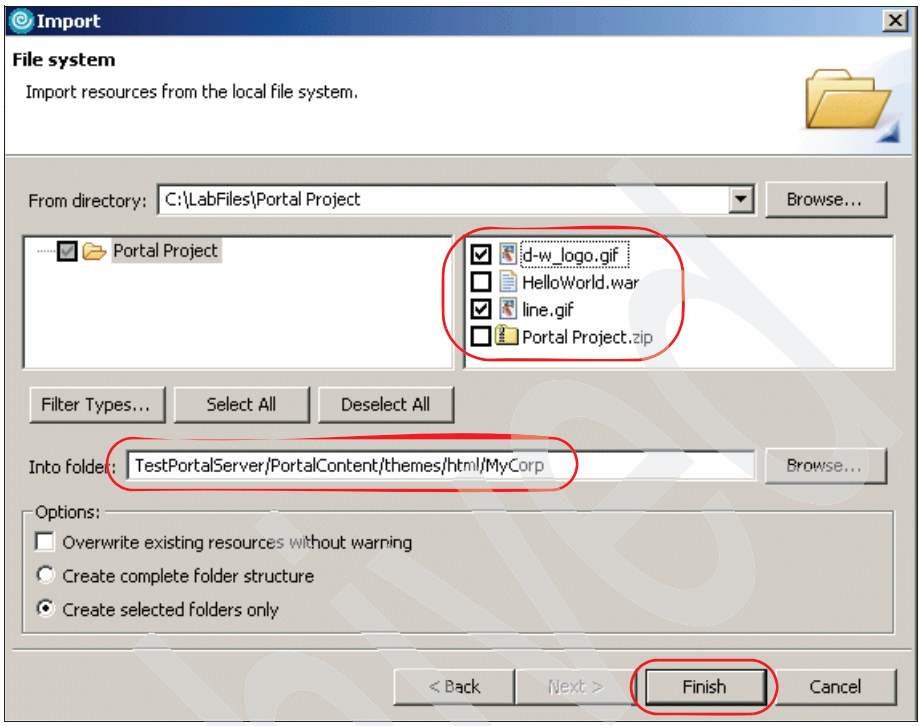


Figure 33-7 Importing graphics

7. Change the background image:

   a. Get to the Switch Active Document, right-clicking the Design area of the Default.jsp file and select **Switch Active Document**.

   b. Right-click the current background image.

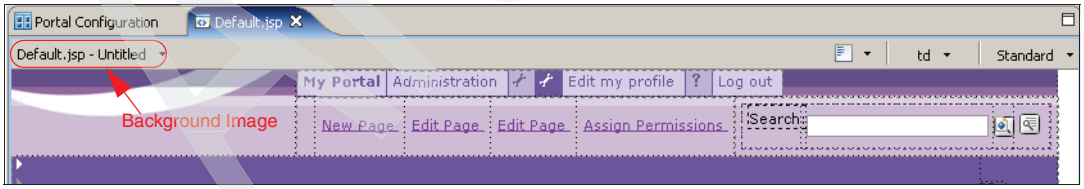


Figure 33-8 Select background image

   c. Select **Style** → **Show Applied Styles**.

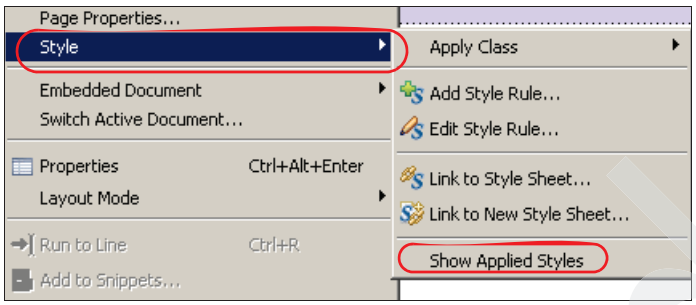


Figure 33-9 Show Applied Styles option

d. In the Style view right-click **<<<table class="wpsToolbarBannerBackground">>>**

e. Select **Edit.**

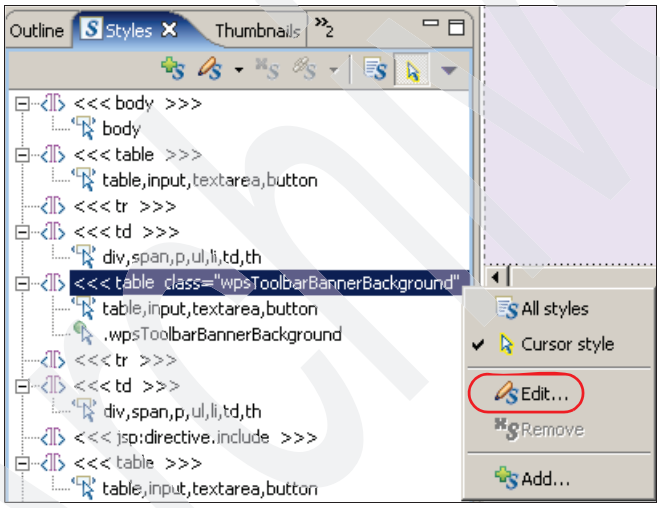


Figure 33-10 Edit the class

f. You can view the active properties of the selected class by clicking **>>Preview Properties** button.

g. Leave the default values in the `Edit Style` window and click **OK**.

Archived

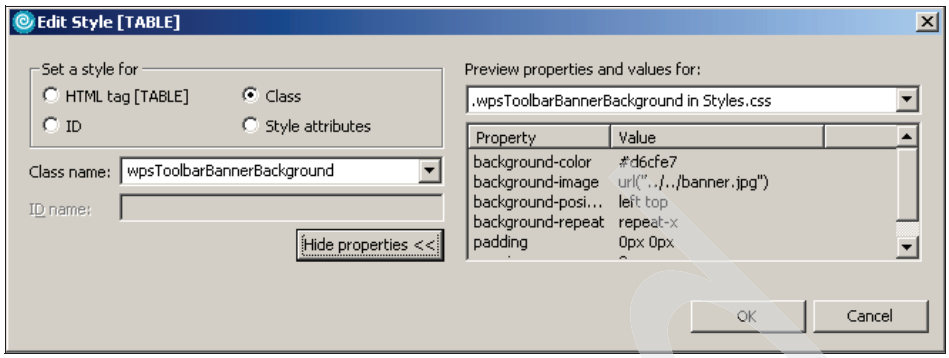


Figure 33-11 Class properties

h. Leave the style that appears selected by default in the `Select one style window` and click **OK**.

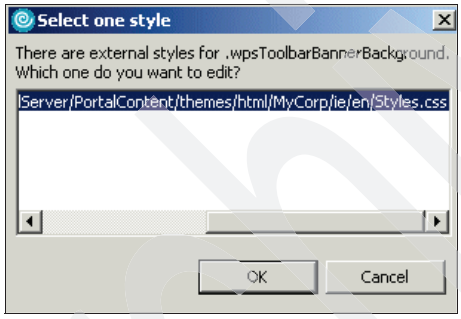


Figure 33-12 Select one style

i. In the `Set Style Properties` window, select **Background** and modify the following values :

- **Color**: `#FFFFFF`
- **Image**: browse to the `TestPortalServer\themes\html\MyCorp\d-w_logo.gif`
- **Repeat**: No repeat
- **Top**: Center

Archived

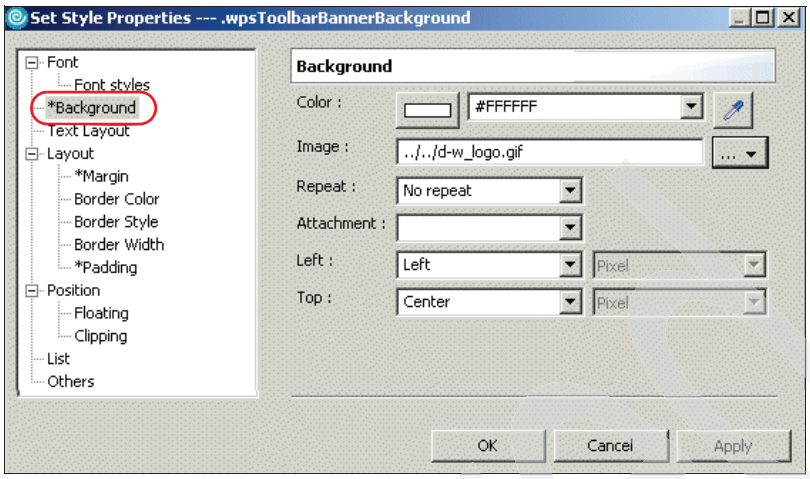


Figure 33-13 Changing Properties

j. Click **OK**.

k. Also notice that the `Styles.css` file has been opened in the Page Designer.

l. Select **File** → **Save all**

8. The resulting new style should looks as shown in Figure 33-14.

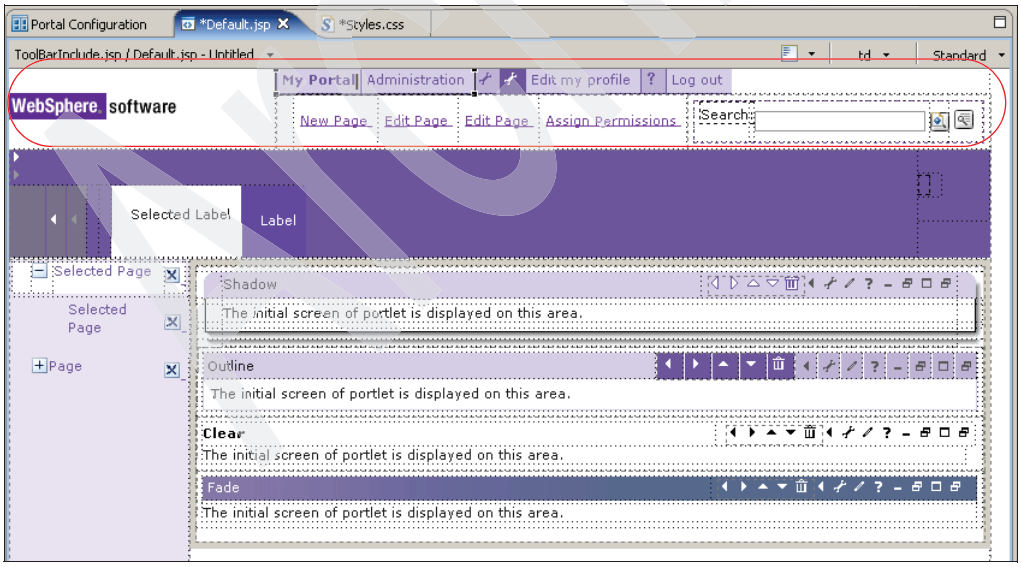


Figure 33-14 Resulting style after changing the WPSToolbarBannerBackground class

9. Change the **PlaceBarInclude.jsp:**

   a. Select the Switch Active Document to switch to **PlaceBarInclude.jsp**

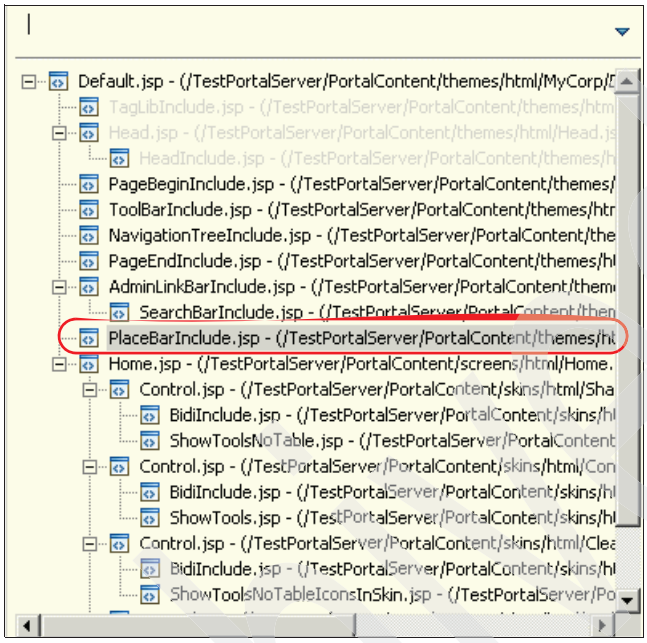


Figure 33-15 Switch to PlaceBarInclude.jsp

Tip: Selecting the Source tab of the Page Designer, you will see the JSP selected in the Switch Active Document. In this Editor you can also select any of the embedded JSP to edit it, except the Control.jsp of Skins.

   b. Right-click the purple row of the theme, select **Style** → **Show Applied Styles** from context menu.

Ahmed

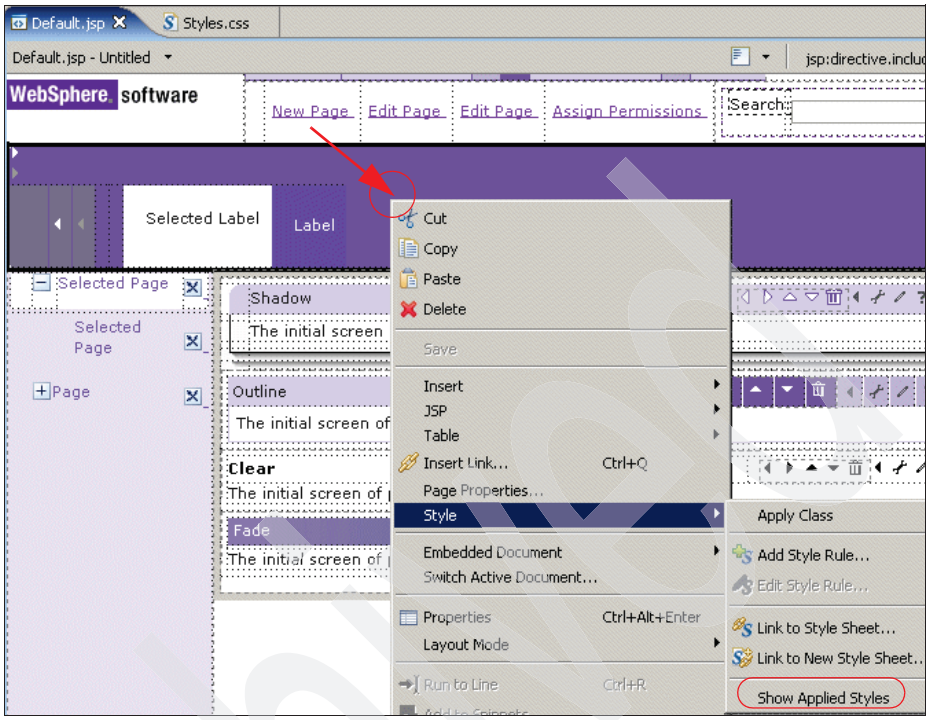


Figure 33-16 Show Applied Styles

c. In Styles view right-click **<<<table class="wpsPlaceBar">>>** and select **Edit**.

Archive

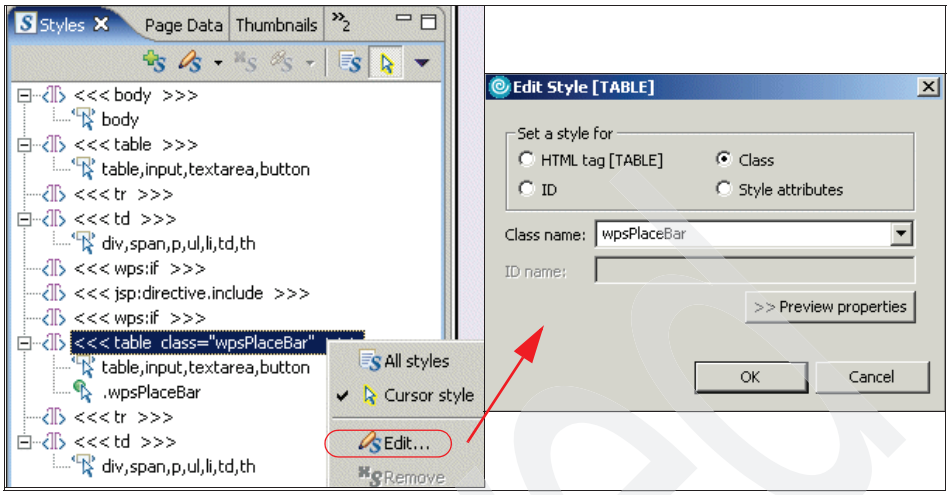


Figure 33-17 Edit the class

d. Leave the default values in the `Edit Style` window and click **OK**.

e. Leave the style that appears selected by default in the `Select one style window` and click **OK**.

f. In the `Set Style Properties` window, select **Background** and modify the following values :

- **Image**: browse to the `TestPortalServer\themes\html\MyTheme\line.gif`
- **Repeat**: Repeat Horizontally

Archive

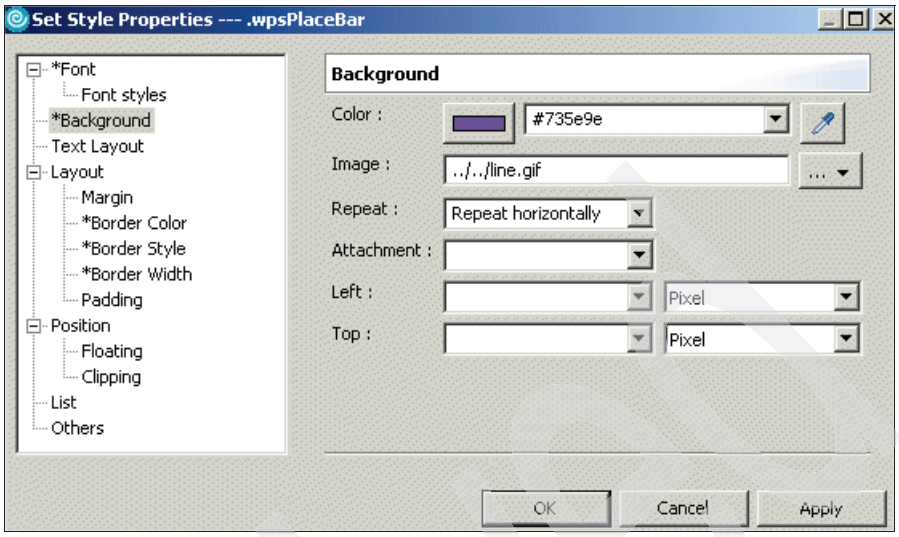


Figure 33-18 Changing properties

g. Click **OK**.

h. Select **File** → **Save all**

10. The resulting new style should looks as shown in Figure 33-19.

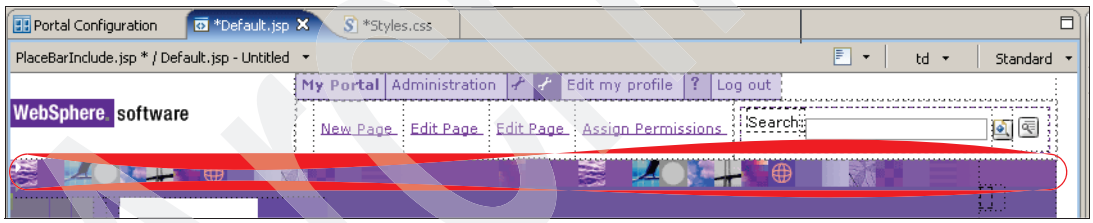


Figure 33-19 Resulting style after changing the wpsPlaceBar class

33.4 Editing Styles

To edit a style sheet, follow these steps:

1. Edit the **Unselected Label** Bar.

   a. Change to the Style.css file next to the `Default.jsp` in the Page Designer section. The Style.css file should be opened as a result of the steps followed in section “Editing a theme” on page 965. If not, you can open it

from TestPortalServer\PortalContent\Project Explorer\themes\html\MyCorp\ie\en\Style.css.

Note: The themes\html\MyCorp\ie\en\Style.css file is the default style sheet. In case you need to modify any other style sheet, you can find it in the `themes` root. For example, in our scenario the `themes` root is:

TestPortalServer\PortalContent\Project Explorer\themes\html\MyCorp

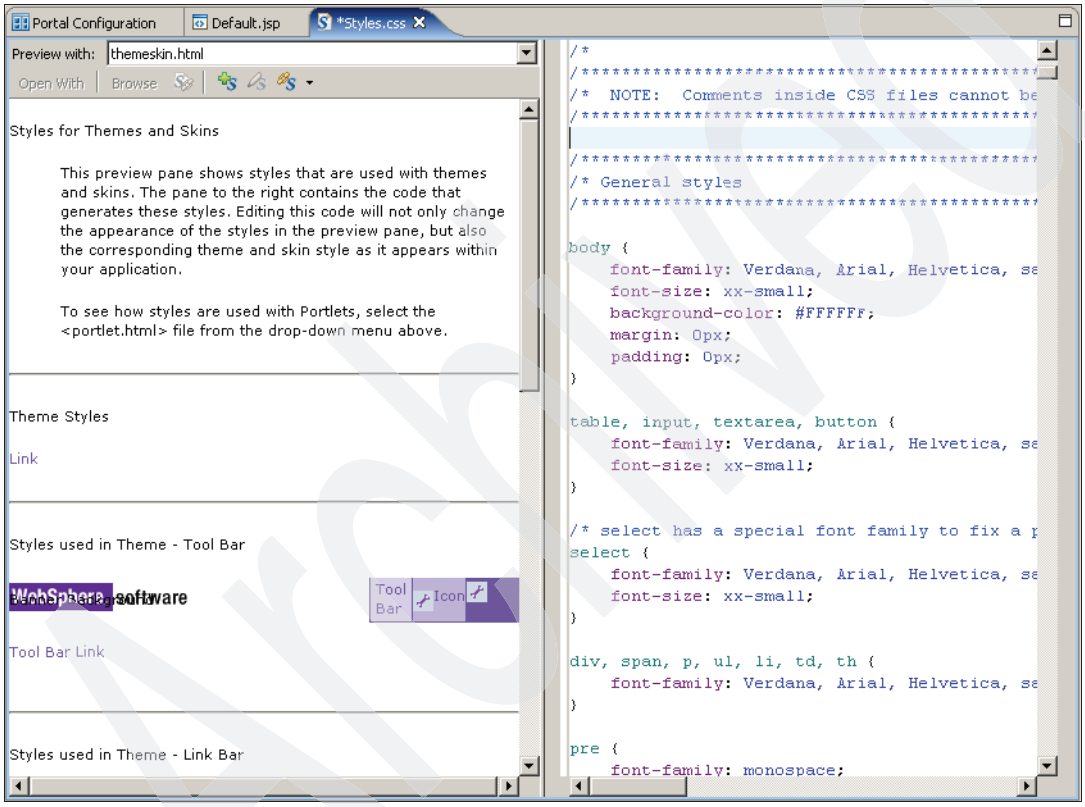


Figure 33-20 Edit the style sheet

b. Right-click the **Unselected Label** style rule and select **Edit Style Rule [.wpsUnSelectedPlace, .wpsUnSelectedPlace:visited, .wpsUnSelectedPlace:hover, .wpsUnSelectedPlace:active]...** option, as shown in Figure 33-21 on page 978.

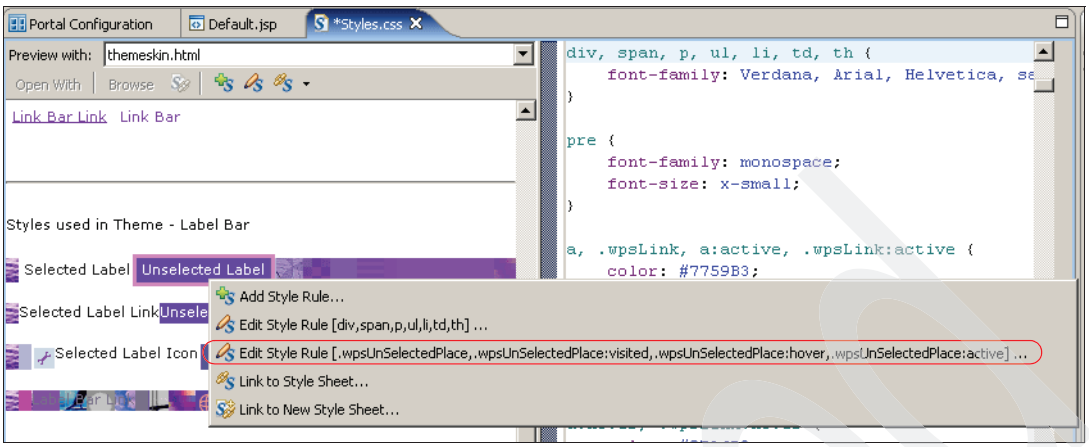


Figure 33-21 Edit Style Rule

c. In the Set Style Properties window, do the following:

- Select **Background** and change the value Color to `#d6cfe7`.

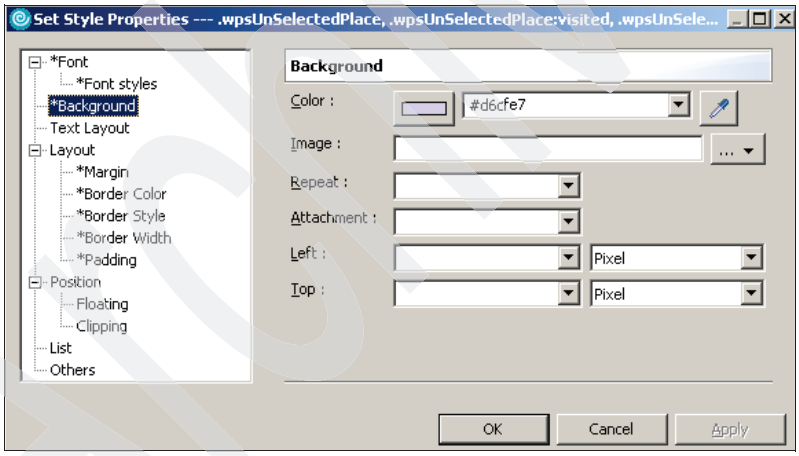


Figure 33-22 Background

- Select **Font** and change the value Color to `#969696`.

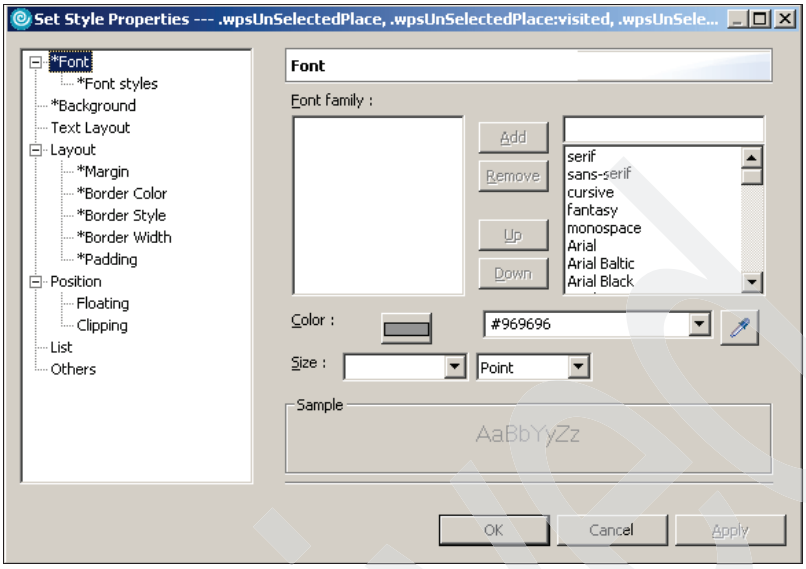


Figure 33-23 Font

- Select **Border** and change the value Color (right and left) to `#d6cfe7`.

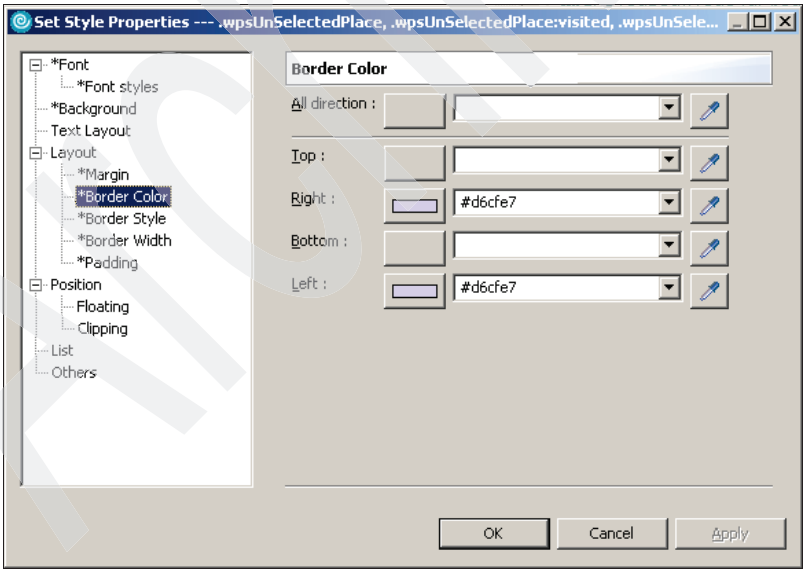


Figure 33-24 Border color

d. Click **OK**.

2. Edit the Unselected Label Link.

   a. Right-click the **Unselected Label Link** style rule and select **Edit Style Rule [.wpsUnSelectedPlaceLink, .wpsUnSelectedPlaceLink:visited, .wpsUnSelectedPlaceLink:hover, .wpsUnSelectedPlaceLink:active]...** option**.**

   b. In the Set Style Properties window, do the following:

      - Select **Background** and change the value Color to `#d6cfe7`.

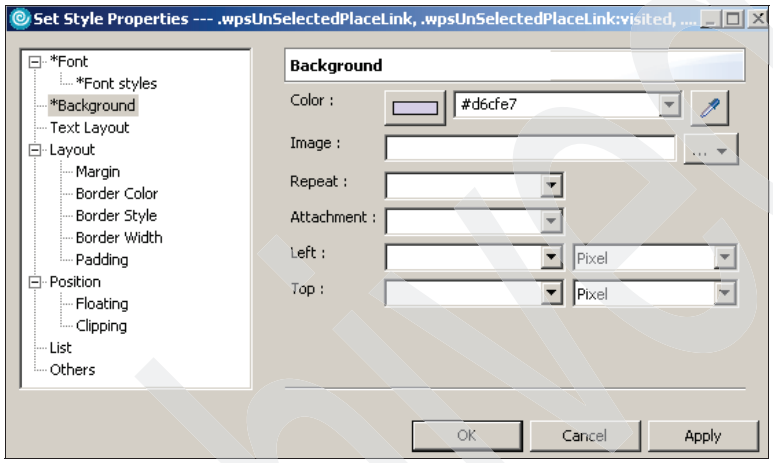


Figure 33-25 Background

      - Select **Font** and change the value Color to `#969696`.

Archived

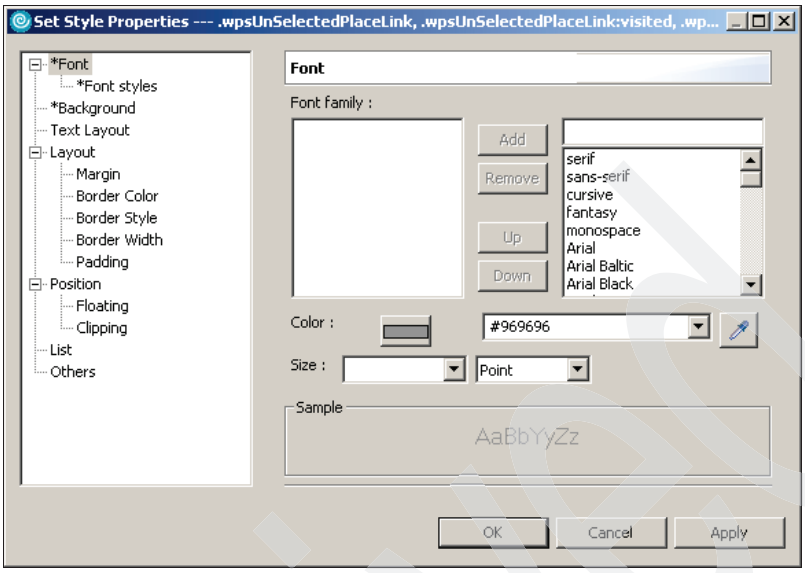


Figure 33-26 Font

c. Click **OK**.

d. Select **File** → **Save** to save the Styles.css file.

3. Edit a Skin.

   a. Select the **Switch Active Document** to select **Control.jsp (/TestPortalServer/PortalContent/skins/html/clear/Control.jsp)**.

   Note: Notice how the corresponding embedded element is highlighted in the editor, when you mouse over the jsp files in the list.

Archive

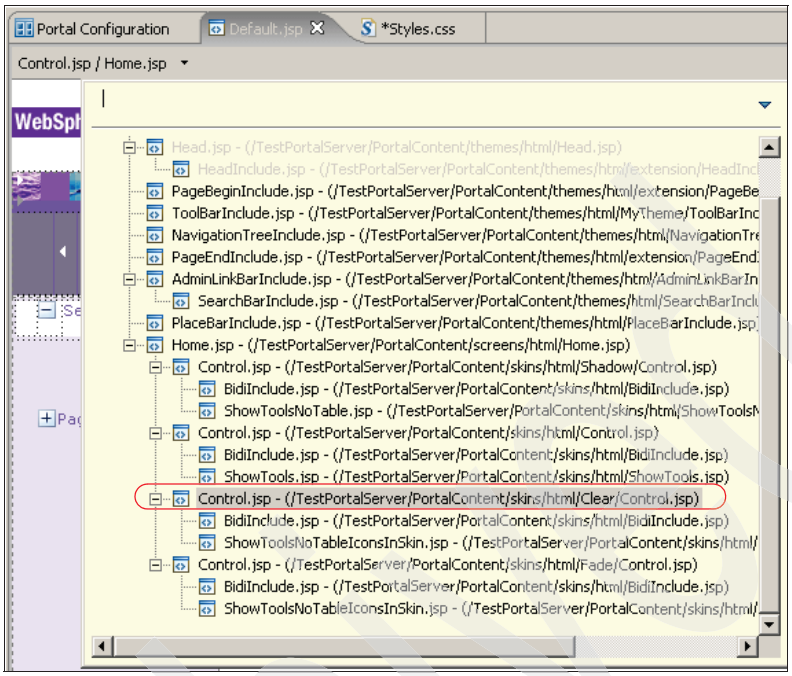


Figure 33-27 Select Control.jsp

b. Click the **Title** row of the skin by moving the mouse next to the left margin of the skin until a black arrow appears, as shown in Figure 33-28 on page 983.

Archive

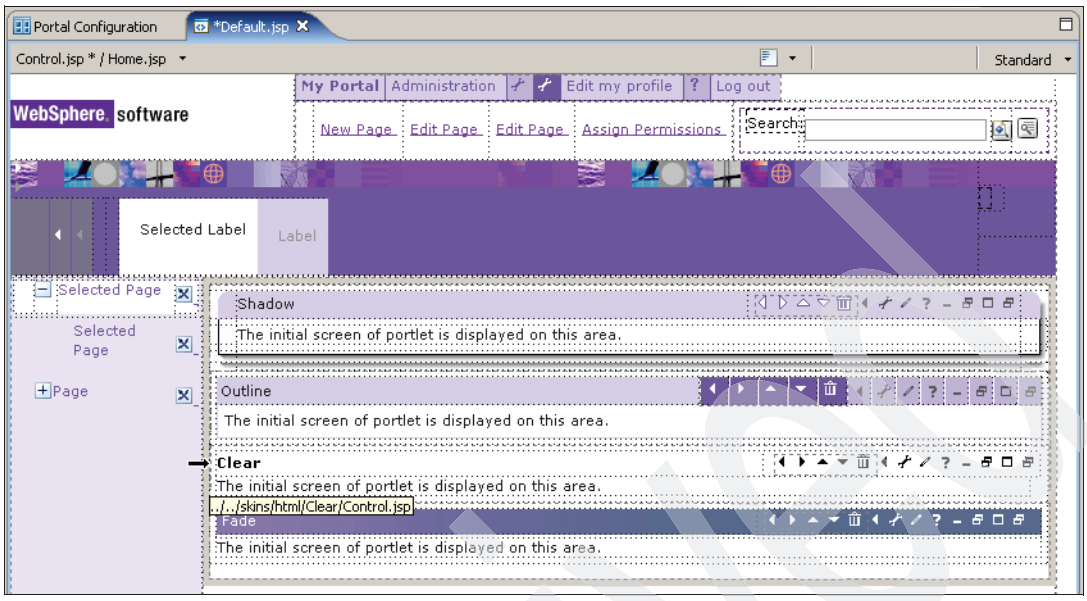


Figure 33-28 Skin Title row

c. The properties for this element are already active in the properties view. In the **Background** option change the value of Color to `#e5eff7`.

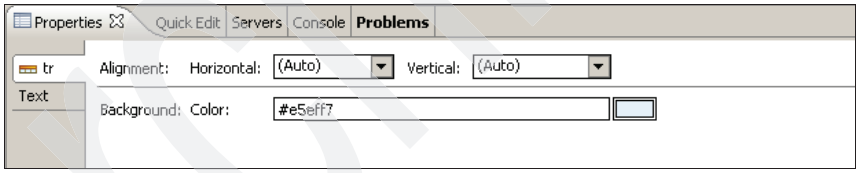


Figure 33-29 Properties

d. Select **File** → **Save** to save the changes.

4. The final theme should look like Figure 33-30 on page 984.

Archive

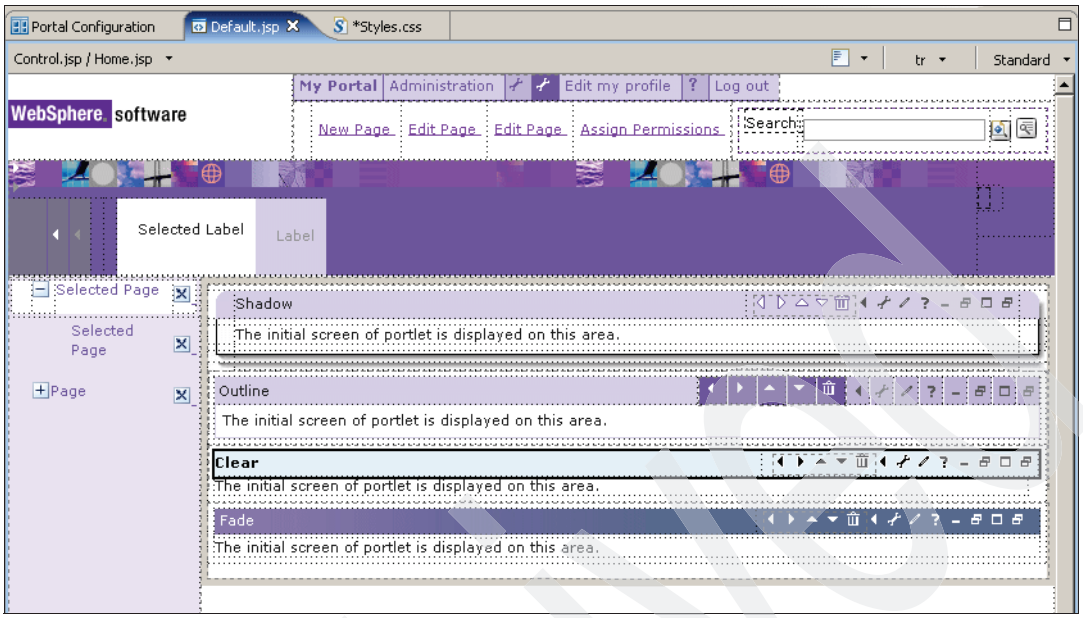


Figure 33-30 MyTheme

33.5 Applying MyCorp theme

Execute the following steps:

1. Open the Portal Configuration under the TestPortalServer portal project.
1. Select the **WebSphere** label (next to the Welcome label).
2. In the Properties view, select the **Label** tab.
3. Click the **Theme** list box and select **MyCorp** at the bottom of the list of available themes.

Archive

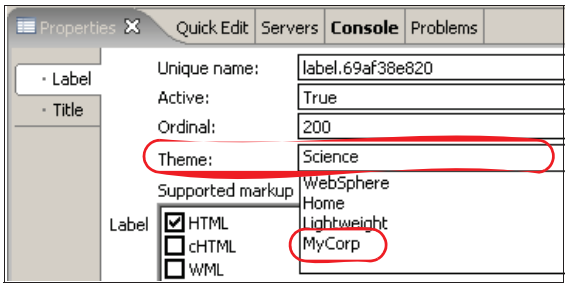


Figure 33-31 Label Properties

4. Wait for the Portal Configuration to display the change.

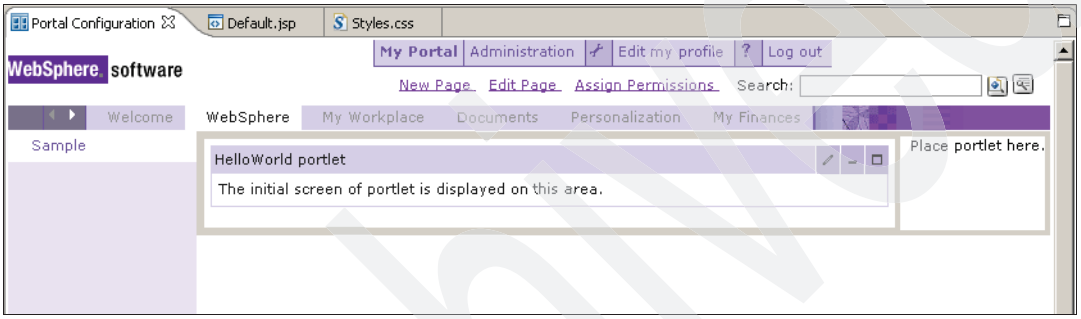


Figure 33-32 Applied Theme

> **Note:** Note how the child pages for this labels are also displayed with the new theme.

5. Select **File** → **Save**.

33.6 Applying a Skin

Execute the following steps:

1. Select the **HelloWorld** portlet from the `WebSphere` page
2. In Properties view, click the **Skin** list box to display the available skins.
3. Select **Clear** from the list of skins.

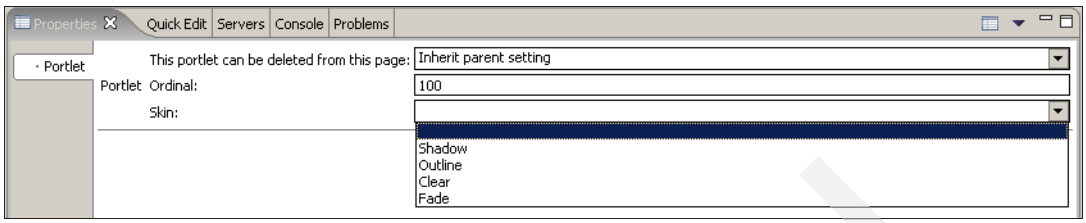


Figure 33-33 Skin Properties

4. Wait for the Portal Configuration to display the change.

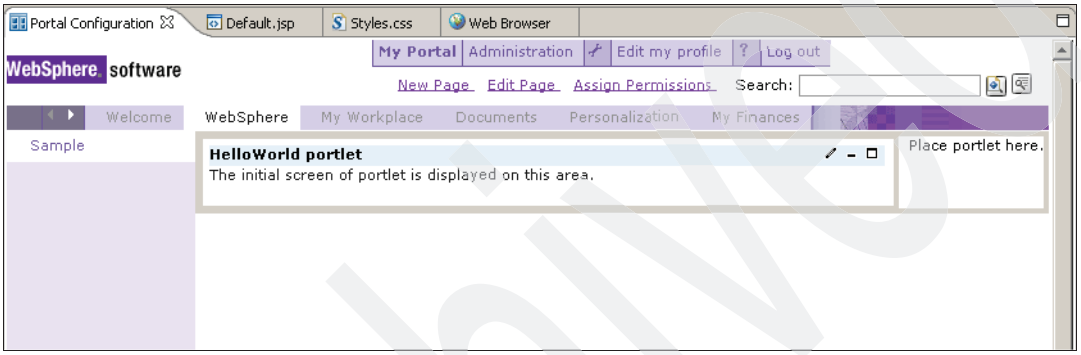


Figure 33-34 Applied Skin

5. Select **File** → **Save**.

33.7 Testing the new Portal Project

A portal project can be tested or debugged in a Local Test Environment or in a Remote Server Attach server. In this sample scenario, the portal project *TestPortalServer* will be locally tested using the Local Test Environment for WebSphere Portal Server V5.1.

To test a portal project, follow this steps:

1. From the Project Explorer, right-click **TestPortalServer**.
2. Select **Run** → **Run on Server...**.

Archived

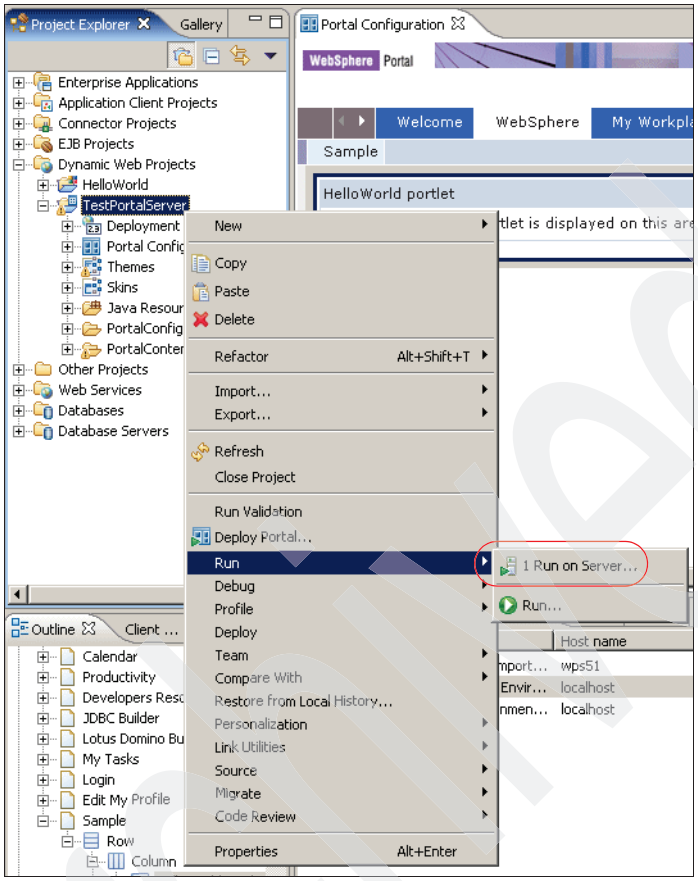


Figure 33-35 Run on Server

3. In the Define a New Server window, select **WebSphere Portal V5.1 Test Environment**.

Archived

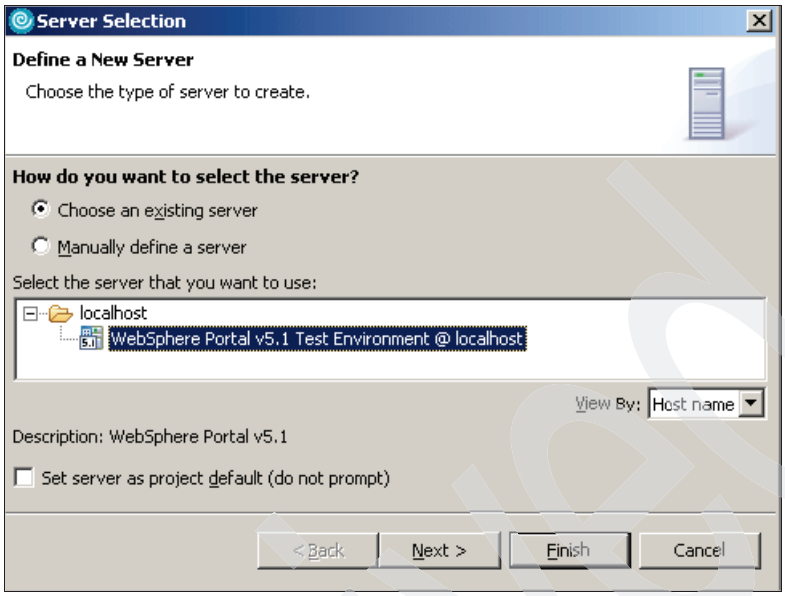


Figure 33-36 Select Test Environment Server

4. Click **Finish**.
5. If a Repair Server Configuration message appears, click **OK**.

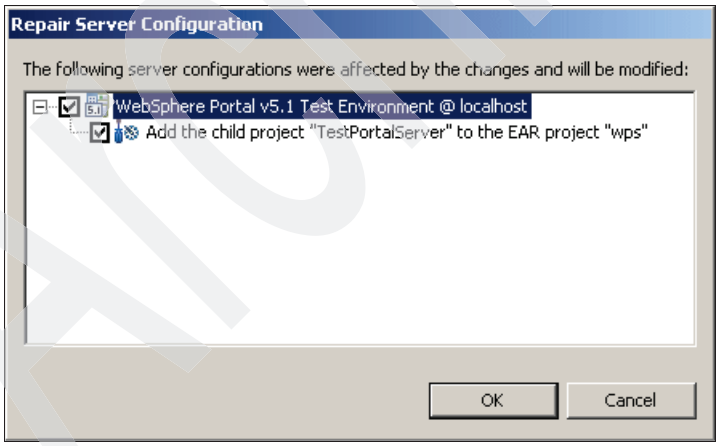


Figure 33-37 Repair Server Configuration

6. A Web browser will open within the Rational Application Developer workbench showing the Portal Server Welcome page.

Archived

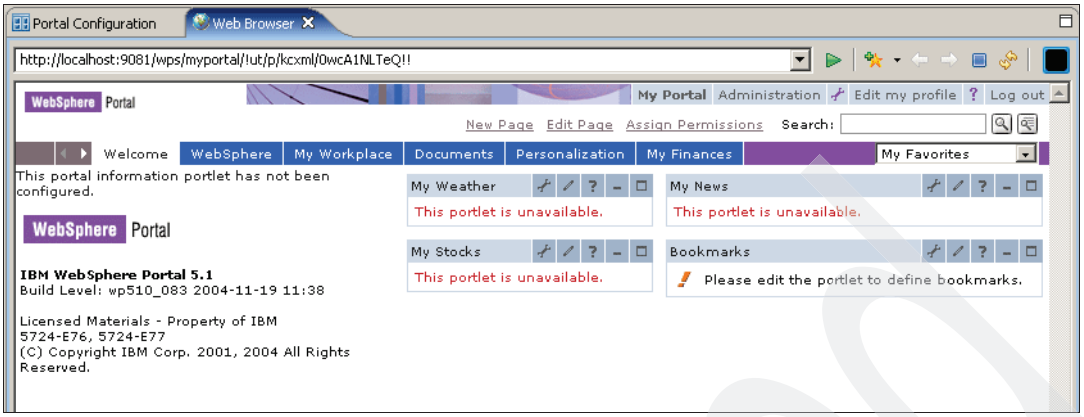


Figure 33-38 WebSphere Portal Server V5.1 Welcome page

7. Select the **WebSphere** page.



Figure 33-39 WebSphere page

8. The new themes and skins are shown in the WebSphere Portal V5.1 (Figure 33-40).

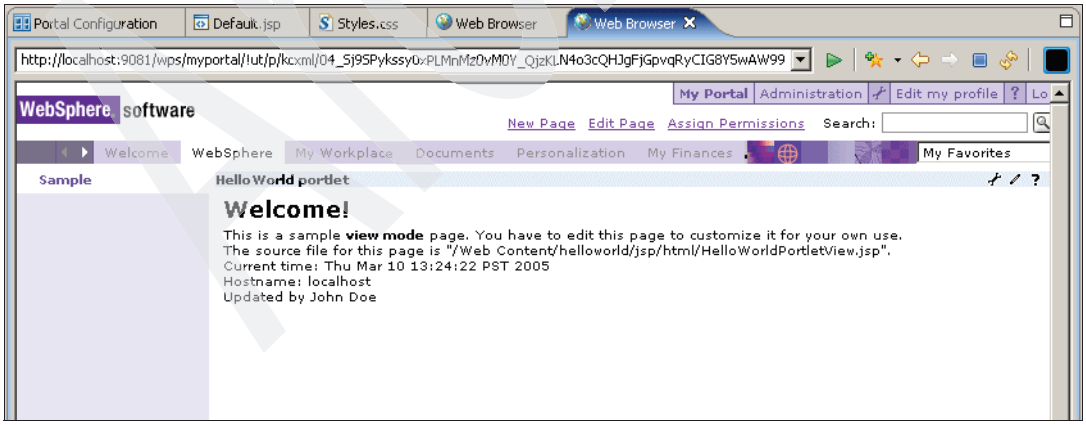


Figure 33-40 Portal page result

9. In the Servers view, right-click the **WebSphere Portal V5.1 Test Environment**.
10. Select **Stop**.

33.8 Publishing the Portal Project

Once all the modifications and the new design elements have been tested in the development environment, you should publish them to a Portal Site. To publish a Portal Project into a WebSphere Portal server, two activities are needed. First, export the portal project configuration using the Export Wizard, to generate the files required to publish to the Portal Site. Secondly, manually publish the changes into the Portal server.

Exporting the Portal Project

1. In the Project Explorer, right-click **TestPortalServer.**
2. Select **Export** → **Export...**

Archive

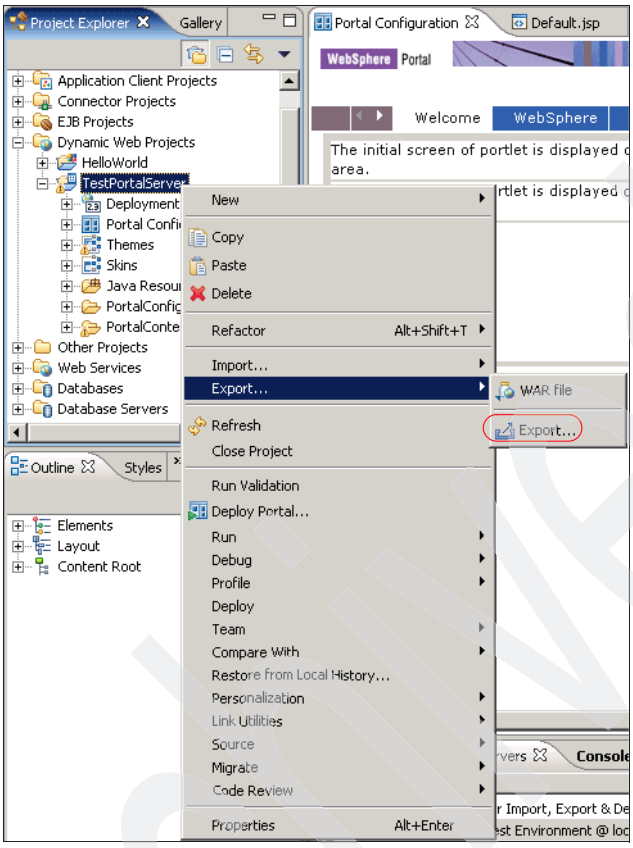


Figure 33-41 Select Export

3. Select **Portal Project Deploy Set** from `Select` window and click **Next**.

Archived

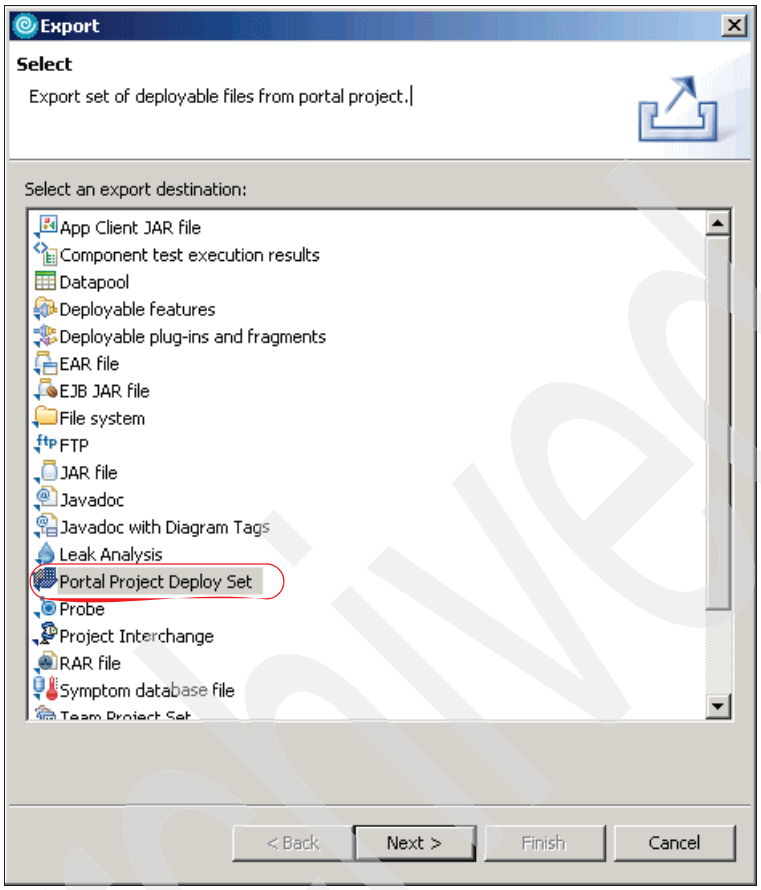


Figure 33-42 Portal Project Deploy Set

4. In the Portal Project Export window, enter the following values:
   - Portal Project EAR: `TestPortalServer`
   - Destination: `c:\PortalProject`
   - Select the portal server that you want to use: `WebSphere Portal V5.1 Import, Export & Deploy`. This is the portal connection created in the scenario described in Chapter 32, “Updating a portal layout” on page 935.

Important: Do not attempt to manually deploy the exported files to a portal server other than the one you selected in step 4. The generated files contain information from this portal server and the procedure will not work in other servers.

Archived

5. Click **Finish**.
6. Click **Yes** in the message box that appears alerting the following: `This process may take a long time and should not be interrupted. Do you wish to continue?`

> **Note:** Even the Export wizard does not automatically update your project with server information, it needs to communicate with the portal server during export.

The following files are generated by the Export wizard:

- WPS.ear
- Deployproject.xml for deployment portal configuration.
- DeployInstructions.txt file with instructions for deploying to a server.

These files could also be included:

- WAR files for each portlet project used in the portal project.
- DeployPortlets.xml script for deploying portlets.

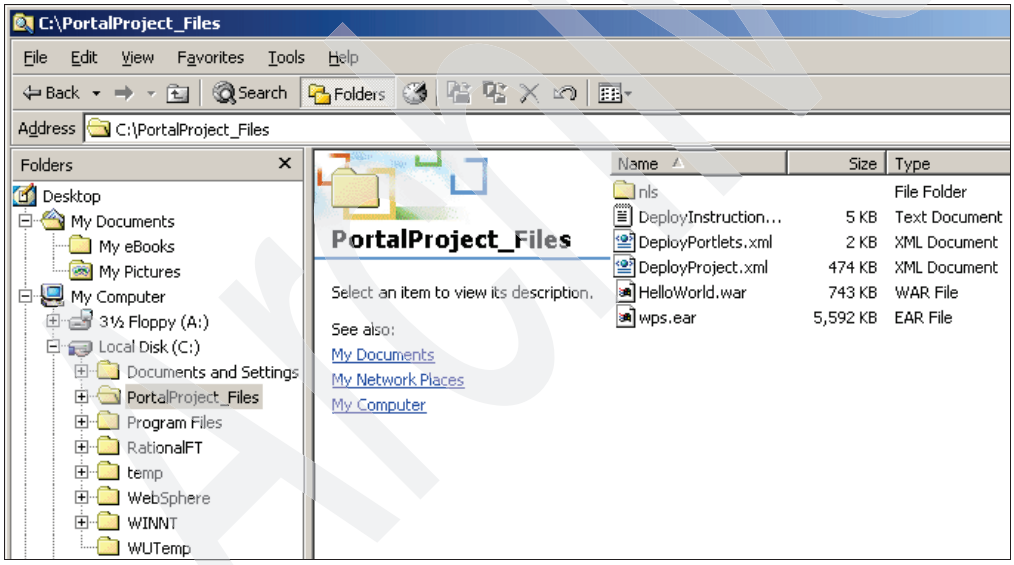


Figure 33-43 Files Created

33.9 Deploying the Portal Project

Follow these steps to deploy a Portal Project in WebSphere Portal server.

Archived

Important: You can deploy the project only into the Portal server specified in the RAD Export wizard.

1. Copy the generated files in a directory of the server where WebSphere Portal is running.
2. Open the DeployInstruccions.txt from the destination directory where the exported files where created.

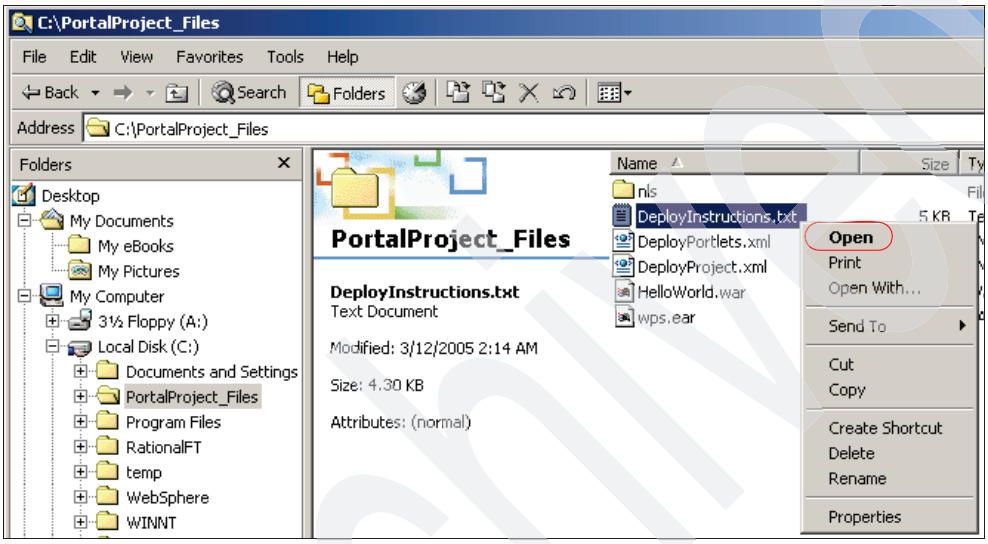


Figure 33-44 Open the DeployInstructions file

3. The DeployInstructions.txt file should contain information as shown in Example 33-1.

Example 33-1 DeployInstructions.txt file

```
(C) Copyright IBM Corporation 2004. All Rights Reserved.

This file provides instructions for deploying an IBM Rational Application
Developer Portal Project to a portal server.

Important Note: These instructions are to be used only for deploying to server
wps51 and should not be used with a different portal server.
Information specific to server wps51 has been included in the configuration
files.
Both the WebSphere Application Server and the WebSphere Portal server must be
running to successfully complete these steps.
```

Archived

Each step must be successfully completed before starting subsequent steps.

Step 1
Open a command prompt and change directories to the Deployment Manager bin directory. For example, c:\WebSphere\DeploymentManager\bin, or if the Deployment Manager is not installed:
c:\WebSphere\appserver\bin.
Copy the wps.ear that was exported by IBM Rational Application Developer to this directory.
Use the wsadmin command to update the WebSphere Portal EAR. This action will automatically cause the application to be synchronized across each node in the cluster.

Windows: wsadmin.bat -user <admin_user_id> -password <admin_password> -c "$AdminApp install wps.ear {-update -appname wps}"

Unix: wsadmin.sh -user <admin_user_id> -password <admin_password> -c "$AdminApp install wps.ear {-update -appname wps}"

where:

<admin_user_id> is the administrator's user ID
<admin_password> is the administrator's password

Note: Updates to the configuration of a WebSphere Portal cluster must occur on the Deployment Manager and resynchronized with the other nodes in the cluster.

If updates are made to individual nodes in the cluster, the updates will be lost when the master configuration on the Deployment Manager resynchronizes with the nodes again.

Step 2
Copy each of the portlet application WAR files to <wps_home>\installableApps on your server.

The following is a list of portlet application WAR files that you must copy:
HelloWorld.war

Step 3
Copy DeployPortlets.xml to <wps_home>\bin and execute the following XMLAccess command from the same location

Windows: xmlaccess -in DeployPortlets.xml -user <WpsAdminUser> -pwd <WpsAdminPassword>

Unix: xmlaccess.sh -in DeployPortlets.xml -user <WpsAdminUser> -pwd <WpsAdminPassword>

Archived

```
Step 4
   Copy DeployProject.xml to <wps_home>\bin and execute the following XMLAccess
command from the same location

       Windows: xmlaccess -in DeployProject.xml -user <WpsAdminUser> -pwd
<WpsAdminPassword>

       Unix: xmlaccess -in DeployProject.xml -user <WpsAdminUser> -pwd
<WpsAdminPassword>

Step 5
   To ensure that string changes in the project are correctly used by the
portal server, copy the following list of properties files from the 'nls'
directory
   to '<wps_home>/shared/app/nls', where <wps_home> is the installed location
of WebSphere Portal server.
       CSRes.properties
       CSRes_ar.properties
       CSRes_cs.properties
       CSRes_da.properties
       CSRes_de.properties
       CSRes_el.properties
       CSRes_en.properties
       CSRes_es.properties
       CSRes_fi.properties
       CSRes_fr.properties
       CSRes_hu.properties
       CSRes_it.properties
       CSRes_iw.properties
       CSRes_ja.properties
       CSRes_ko.properties
       CSRes_nl.properties
       CSRes_no.properties
       CSRes_pl.properties
       CSRes_pt.properties
       CSRes_pt_BR.properties
       CSRes_ro.properties
       CSRes_ru.properties
       CSRes_sv.properties
       CSRes_th.properties
       CSRes_tr.properties
       CSRes_uk.properties
       CSRes_zh.properties
       CSRes_zh_TW.properties

Step 6
   Browse to the portal server administration pages. Using the administrative
portlets, set access control as needed on the deployed configuration.
```

Archived

```
Note:
If the portal server is configured to use a HTTP port other than port 80,
append the parameter '-url <portal config url>' to the XMLAccess commands where
<portal config url> is of the form 'http://host:port/wps/config'.

For more information refer to the following topics in WebSphere Portal
InfoCenter:
   Administering your portal\XML configuration interface
   Designing your portal\Deploying customized themes and skins
   Designing your portal\Customizing the portal
```

4. To Verify that WebSphere_Portal and Server1 are running, open a command window, and change to directory **c:\WebSphere\AppServer\bin**
5. Type the command **serverStatus -all**

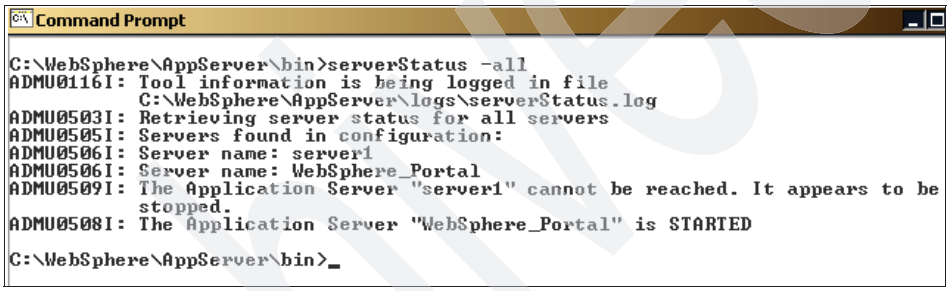


Figure 33-45 Verify Servers status

6. If any of the servers is stopped, enter **startServer** and the name of the server.

Example 33-2 Starting server1

```
c:\WebSphere\AppServer\bin>startServer server1
```

or

Example 33-3 Starting PortalServer

```
c:\WebSphere\AppServer\bin>startServer WebSphere_Portal
```

In case portal security is enabled also add to the command the suffix **-user user -password password**.

7. Once both servers are running, continue with the steps specified in the `DeployInstruction.txt` file. Copy the **wps.ear** file created by the Export wizard to **c:\WebSphere\AppServer\bin**

Important: It is recommendable that back up your Portal in case you want to return the original configuration.

8. Update the WebSphere Portal EAR using the **wsadmin** command as shown:

```
wsadmin.bat -user wpsadmin -password wpsadmin -c "$AdminApp install
wps.ear {-update -appname wps}"
```

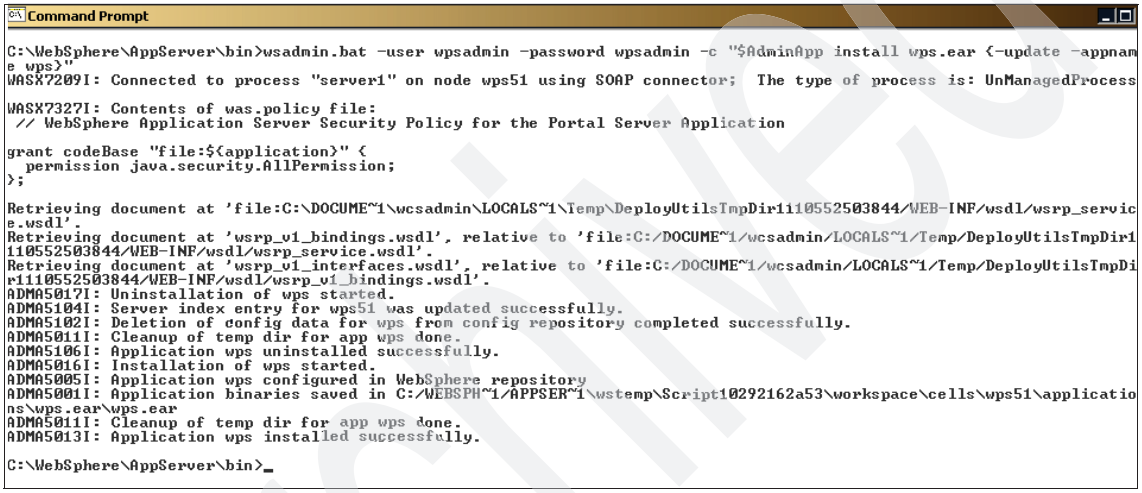


Figure 33-46 wsadmin command

9. Copy the **HelloWorld.war** portlet application WAR file to **c:\WebSphere\PortalServer\installableApps** on your server.
10. Copy **DeployPortlets.xml** and **DeployProject.xml** to **c:\WebSphere\ProtalServer\bin**
11. Execute the following XMLAccess command from **c:\WebSphere\ProtalServer\bin**

```
xmlaccess -in DeployPortlets.xml -user wpsadmin -pwd wpsadmin -url
http://wps51.itso.ral.ibm.com:9081/wps/config
```

Archived



Figure 33-47 Deploy portlets

12. Execute the following XMLAccess command from **c:\WebSphere\ProtalServer\bin**

```
xmlaccess -in DeployProject.xml -user wpsadmin -pwd wpsadmin -url
http://wps51.itso.ral.ibm.com:9081/wps/config
```

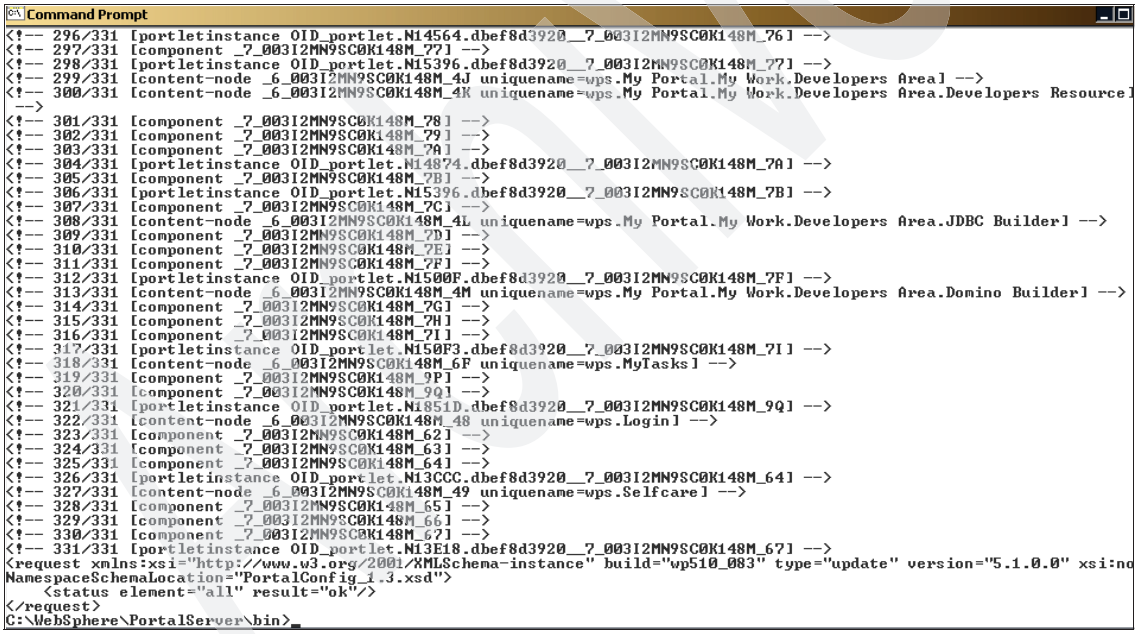


Figure 33-48 Deploy Project

13. Copy the properties files from the 'nls' directory to c:\WebSphere\PortalServer\shared\app\nls

14. Stop the server.

```
c:\WebSphere\AppServer\bin\stopServer WebSphere_Server
```

15. Start the server.

```
c:\WebSphere\AppServer\bin\startServer WebSphere_Server
```

16. In a Web browser enter the Portal Server url to open the site. For example, `http://wps51.itso.ral.ibm.com:9081/wps/portal`. The Welcome portal page will be displayed.

17. Login to the portal by enter the user `wpsadmin` and password `wpsadmin`. The portal should look as Figure 33-49. Notice that the WebSphere page was created as part of the sample scenario described in Chapter 32, “Updating a portal layout” on page 935.

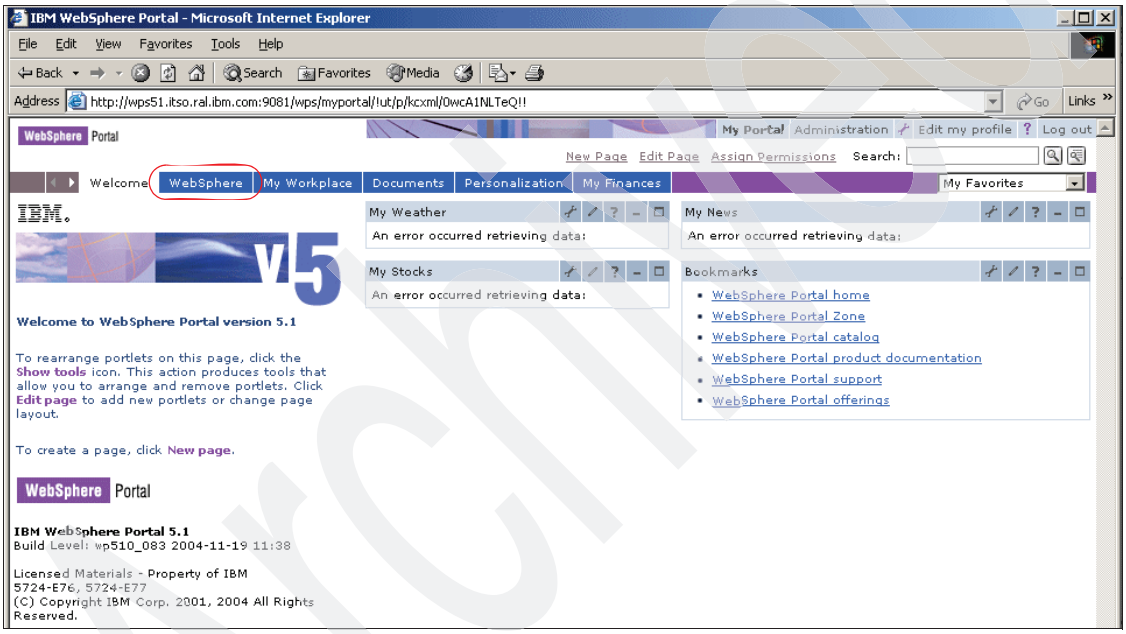


Figure 33-49 Portal page

18. Click **WebSphere** Page. The page will be shown with the `MyCorp` theme.

Archive

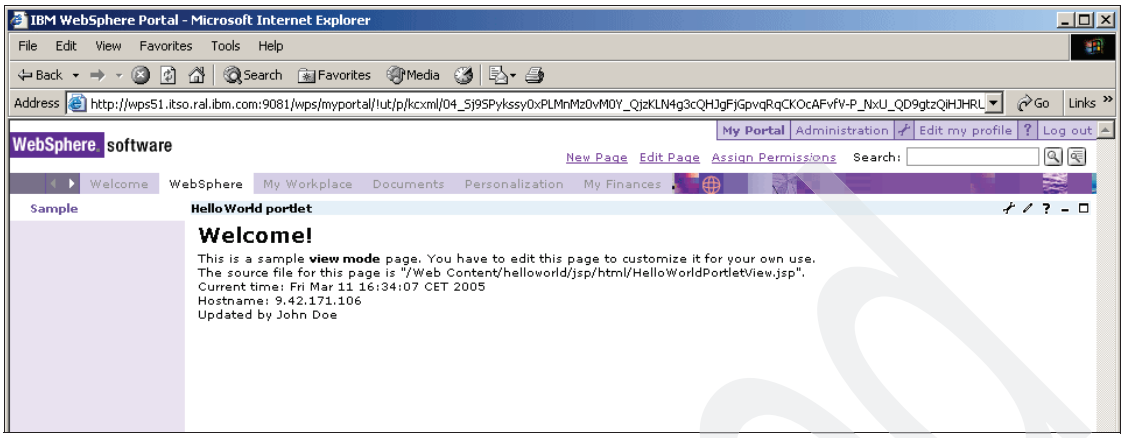


Figure 33-50 WebSphere page with MyCorp theme

Archived

Archived

A

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246681

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246681.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
sg246681.zip	Zipped code samples

© Copyright IBM Corp. 2005. All rights reserved.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 30 GB minimum
Operating System: Windows
Processor: 1 Ghz or higher
Memory: 1 GB or higher

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 1007. Note that some of the documents referenced here may be available in softcopy only.

- *Rational Application Developer V6 Programming Guide*, SG24-6449
- *IBM WebSphere Portal V5 A Guide for Portlet Application Development*, SG24-6076
- *IBM WebSphere Portal for Multiplatforms V5.1 Handbook*, SG24-6689
- *IBM WebSphere Everyplace® Access Version 4.3 Handbook for Developers,* SG24-7015-01
- *IBM WebSphere Everyplace Access V5 Handbook for Developers and Administrators Volume I: Installation and Administration,* SG24-6462
- *IBM WebSphere Everyplace Access V5 Handbook for Developers and Administrators Volume II: Application Development,* SG24-6463
- *IBM WebSphere Everyplace Access V5 Handbook for Developers and Administrators Volume III: E-Mail and Database Synchronization,* SG24-6676
- *IBM WebSphere Everyplace Access V5 Handbook for Developers and Administrators Volume IV: Advanced Topics,* SG24-6677
- *WebSphere Portal V5.0 Production Deployment and Operations Guide*, SG24-6391
- *IBM WebSphere Portal for Multiplatforms V5 Handbook*, SG24-6098
- *WebSphere Portal on z/OS®*, SG24-6992
- *Patterns: Portal Search Custom Design*, SG24-6881
- *Develop and Deploy a Secure Portal Solution Using WebSphere Portal V5 and Tivoli Access Manager V5.1*, SG24-6325
- *Deploying a Secure Portal Solution on Linux Using WebSphere Portal V5.0.2 and Tivoli Access Manager V5.1*, REDP-9121
- *WebSphere Portal Collaboration Security Handbook*, SG24-6438

© Copyright IBM Corp. 2005. All rights reserved.

- *WebSphere Portal Server and DB2 Information Integrator: A Synergistic Solution*, SG24-6433
- *Document Management Using WebSphere Portal V5.0.2 and DB2 Content Manager V8.2*, SG24-6349
- *Portal Application Design and Development Guidelines*, REDP-3829
- *IBM WebSphere Application Server V5.1 System Management and Configuration, WebSphere Handbook Series*, SG24-6195
- *IBM WebSphere V5.1 Performance, Scalability, and High Availability, WebSphere Handbook Series*, SG24-6198

Other publications

These publications are also relevant as further information sources:

- *IBM WebSphere Portal security solutions white paper, Integrating WebSphere Portal software with your security infrastructure*, G325-2090, available at:

  ftp://ftp.software.ibm.com/software/websphere/pdf/WS_Portal_Security_G325-2090-01.pdf
- *Troubleshooting Pickers in Collaborative Portlets*, Technote 1157249
- *Troubleshooting Automatic Detection of your Mail File with the Different Collaborative Portlets*, Technote 1157029

Online resources

These Web sites and URLs are also relevant as further information sources:

- IBM WebSphere Portal for Multiplatforms Version 5.1 Information Center

  http://publib.boulder.ibm.com/pvc/wp/510/ent/en/InfoCenter/index.html
- WebSphere Application Server V5.1 Information Center

  http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp
- WebSphere Portal product documentation

  http://www.ibm.com/developerworks/websphere/zones/portal/proddoc.html
- WebSphere Portal Catalog

  http://catalog.lotus.com/wps/portal/portal
- WebSphere Portal zone for developers

  http://www.ibm.com/websphere/developer/zones/portal

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

- IBM Support and downloads

  ibm.com/support

- IBM Global Services

  ibm.com/services

Archived

Archived

Index

A

Abstract and concrete portlet applications 155
accessibility 7
Accessing resource bundles in JSPs 382
Accessing resource bundles in portlets 381
Action Event Handling 145
Action events 50
ActionEvent 142, 208, 889
ActionEvent portlet 208
ActionForm 417
ActionListener 141–142
ActionServlet 417
actionURL 362
active credential 635
Add action request handler 649
Add configure mode 650
Add edit mode 650
Add form sample 649
Add portlets 952
addCookie 360
addDataHeader 360
addHeader 360
addIntHeader 360
Adjusting Portal resource bundles 391
administrator 253, 889, 891
Administrators 19
allows 339
Apache Jetspeed 9
applets 355
application components 450
architecture 9
Attribute storage summary 145
Authentication 5
Authorization 5

B

B2E 4
Basic Portlet 54
Basic portlet (JSR 168) 649
benefits 935
bidi 152, 364
Building a war file 171
Business to Business 4, 8
Business to Consumer 4, 8
Business to Employee 4, 8

C

c2a.tld 146
cache 337
calculator project 552
call center application 594
Canonical Portal URLs 35
Cascading Style Sheets 353
Cascading style sheets 366
cHTML 355
classes 18
Clear 985
Click-to-Action 837
Click-to-Action tag 594
Client Certificate Authentication 899
client portlet 606
Collaboration 5, 17
collections 18
Commons-Logging interface 444
Compile errors 902
components 436
concrete-portlet-app 337
config-param 339
configuration 936
Connection 435
Consume 892
Consumer portal 885–886
Consumer portals 889
containsHeader 360
Content Management 5, 16, 36
Content management 5
content.tld 147
ContentAccessService 618–619
Controller 174
cooperative broker 777
cooperative portlet 772
Cooperative Portlets 741
  beginEventPhase() 774
  broadcast 777
  broadcasting 771
  broadcasts 836

© Copyright IBM Corp. 2005. All rights reserved.

C2A wrapper 775
callback method 773
changedProperties() 772
Click-to-Action architecture 749
Click-to-Action event 772
Click-to-Action menus 778
combined scenario 779
Ctrl key 772
declarative approach 837
deployment descriptor 841
encodeProperty 745
encodeProperty tag 777
event phase 773
exchange capabilities 838
Import Resources from a WAR File 780
input parameters 772
Internationalization 844
internationalization 837
JSR 168 portlets 837
messaging communication 835
method invocations 849
output parameters 772
Overview 742
portlet code 842
Portlet Messaging 743
portlet.xml 844
programmatic approach 771, 774
Programming model 744
property broker 772, 836
PropertyBrokerService 843
PropertyListener 745
Register and publish properties 745
registerProperties() 774
resource bundle 844
Run the cooperative portlets 765
sample scenario 779
setProperties() 773
source 836
Source cooperative portlet 750
Steps to program a source cooperative portlet 744
Struts Actions 836
Struts portlets 835
target 836
Target cooperative portlet 761
Web Service Description Language 744
WebSphere Portal Property Broker 743
wires 837
WSDL 838
  WSDL file 844
Cooperative portlets 335, 741
core 365
Create the Service Factory 623
createReturnURI 147
createURI 148
credential slots 632
Credential Vault 335, 630, 641, 648
  administrative slot 633
  Credential slots 633
  Credentials objects 635
  Import a protected servlet application 644
  portlet private slot 633
  private keys 630
  shared slot 633
  SSL client certificates 630
  system slot 633
  user credentials 630
  Using passive credentials 667
  Vault segments 632
Credential Vault Portlet Service 631
credentials 630
CredentialVaultService 617
cryptography 900
CSS 353, 366
CSS file 424
current 937
custcmd 423
Custom developed portlets 41
Custom Services 620
Customizable portlets from a vendor 41

D

dataAttribute 148
Database service 12
dataLoop 149
Debug a portlet application 905
debugger 902
declarative approach 744
Default.jsp 969
define a parameter 599
defineObjects 361
Delegators 19
Demilitarized Zone 10
deployment concerns 115
deployment descriptor 336, 435
design 990
Design area 969

Archived

dir 364
Directory service 11
Directory services 5
doctype 336
Document Categories 36
documents 3
doView 643
drag and drop 595
Dynamic Web Projects 944

E

e-commerce 6
Edit mode 594
edit mode 446
Editing a theme 965
Editing Styles 976
Editors 19
e-learning 6
encodeNameSpace 150
encodeRedirectURL 359
encodeRedirectUrl 359
encodeURI 150
encodeURL 360
Enterprise portal 48
Event Handling 118
event handling 773
execute method 849
Export Wizard 990
extend.tld 147
extranet 10

F

Faces 594
Faces Portlet 54
Faces portlet (JSR 168) 553
Faces portlets
  broadcast 518
  Click-to-Action 602
  commandButton tag 521
  component classes 525
  components 518
  core tag library 518
  Create 594
  database access 518
  event 525
  event handlers 518
  Event Processing 518
  listener model 525
  listeners 518
  managed bean 524
  message tag 521
  navigation rules 521
  Render Response 517
  rendering model 525
  tag library 518
  validation model 525
  validators 518
  Web service client 607
  WSDL Binding Details 610
feature 935
Features page 554
First generation portals 3
first generation portals 3
flushBuffer 360
Flyweight pattern 21
fmt 365
Forms 369
Fourth generation portals 3
framework 51

G

generations of portal technology 3
getAttribute 359
getAttributeNames 359
getAuthType 359
getBufferSize 360
getCharacterEncoding 358, 360
getContentLength 358
getContentType 358
getcontextPath 359
getCookies 359
getDateHeader 359
getEncodeURL 355
getHeader 359
getHeaderNames 359
getHeaders 359
getInputStream 358, 618
getIntHeader 359
getLocale 359
getLocales 359
getMarkup 618
getMethod 359
getOutputStream 360
getParameter 359
getParameterMap 359
getParameterNames 359

Archived

getParameterValues 359
getPathInfo 358
getProtocol 358
getQueryString 358
getreader 358
getRealPath 358
getRemoteAddr 358
getRemoteHost 358
getRequestDispatcher 359
getRequestedSessionId 359
getRequestURI 358
getRequestURL 358
getScheme 359
getServerName 359
getServerPort 359
getServletPath 358
getSession 359
getURL 618
getWriter 360
Global Forwards 437

H

high availability 10
Highlights 31
Highlights in WebSphere Portal V5 31
Host integration 6
HTML form 208
HTTP Servers 11
HttpBasicAuth 643

I

IBM Portlet API 50
IBM Rational Application Developer 51
IBM Styles 370
IBM tags 364
IFRAME 354
images 18
include 618
include() method 356
InfoCenter 53, 893
infrastructure 15
init 147, 643
Insert 952
Internationalization 6–7, 551
Internet 10
Inter-Portlet Communication 118
isCommited 360
isRequestedSessionIdFromCookie 359
isRequestedSessionIdFromURL 359
isRequestedSessionIdFromUrl 359
isRequestedSessionIdValid 359
isSecure 359

J

J2EE technology 48
Java Community Process 12
Java Standard Tag Library 365
Java-based pluggable modules 49
JavaBean 594
JavaScript 355
JavaServer Faces portlets
  accessibility 512
  applications 511
  Apply request values 514
  behavior 512
  configuration file 514
  converters 515
  event handlers 515
  event-driven 513
  faces-config.xml 514
  internationalization 512
  Invoke Application 517
  Invoke application 514
  Java Beans 514
  life cycle 514
  main components 512
  model 511
  Model-View-Controller 512
  MVC 512
  presentation 512
  Process validators 514
  Render response 514
  Restore component tree 514
  reusable user interface 512
  server-side events 511
  technology 512
  Update model values 514
  validation errors 517
  validators 515
  Web container 513
JDBC 669, 703
  create a database connection 675
  DB Servers view 681, 713
  Importing the WAR file 692, 727
  Overview 690, 725, 962
Jetspeed implementation 13

Archived

JSF 335
JSF portlet project 555
JSP editor 439
JSP tags 353, 360, 966
JSPs 353
JSR 168 355, 557
JSR 168 API
  administrator 253
  Deployment descriptors 251
  life cycle 251
  Listeners 251
  local variables 254
  multiple threads 254
  Objects 251
  parameters 252
  portlet deployment descriptor 252
  Portlet Modes 251
  Portlet windows 252
  portletPreferences object 252
  PortletRequest 254
  PortletResponse 254
  web.xml 252
  Window States 251
JSR 168 compliant portlets 844
JSR 168 portlets 618, 843, 849
JSTL 365

L

Label 984
labels 945
Layout 18
layout 18
Lightweight Third Party Authentication 900
Listeners 145
Load balancing 11
Local Test Environment 986
locale 365
log 152
logging 444
logging facility 444
LTPA 900
LTPA token 900

M

Managed Bean 551
Managers 19
menu option 952
menu.tld 147
message 51
Message events 51
Messaging 335
Mode 23
Model 174
Model View Control (MVC) architecture 23
Model-View-Controller 21, 174, 208
ModeModifier 144
Modes 50
multiple devices 9
multiple sources 48
MVC 23
  Model, View and Control 23
MyFirstStruts 431

N

name 364
namespace 363
National Language Support
  Accessing resource bundles in JSPs 382
  Accessing resource bundles in portlets 381
  Adjusting Portal resource bundles 391
  Creating Resource Bundles in WebSphere Studio 376
  NLS administration 386
  NLS best practices 392
  Portal NLS administration 390
  Portlet NLS administration 386
  Sample scenario 393
  Setting NLS titles 390
  Translating Resource Bundles 379
  Translating whole resources 383
  Working with characters 392
navigation 551, 945
next-generation desktop 5
NLS administration 373, 386
NLS best practices 373, 392

O

OCS 13
online access 892
Open Content Syndication 12
Open standards 12
Organization for the Advancement of Structured Information Standards 12
original 936
Overview 1
  Aggregation Module 15

Archived

Authentication Server 15
Authorization 33
Click-To-Action 39
Client to remote application 42
comparison of V4.x permission with V5.x roles 19
credentials 9
Customer Relationship Management 41
database structures 16
Document Categories and Summaries 36
e-Business needs 4
Enabling for Communities 32
Enterprise Resource Planning 41
Event Broker 32
events 9
evolution process 3
First generation portals 3
Fourth generation portals 3
General Infrastructure 32
generations of portal technology 3
high availability 10
J2EE platform 4
J2EE Security 33
Jetspeed implementation 9
LDAP 16
Member Subsystem 33
model-view-controller 23
Open Source Portal 9
Overview 4
Page content 14
page structure 14
page structure. 14
Permissions 19
Portal concepts 18
Portal Document Manager 36
Portal engine 14–15
Portal Install 32
Portal Servlet 15
portal technology 4
Portlet container 15
portlet container 26
Portlet events and messaging 26
portlets 8, 14
Presentation services 14
profile information 9
Property Broker 32
remote content 9
Reverse Proxy Security Server 11
Roles 19
Search 35
Second generation portals 3
Security services 16
services 16
Site Analysis 17
Skins 20
SSO Functionality 33
Struts Portlet Framework 37
Themes 20
Third generation portals 3
Transcoding 37
Transcoding Technology 16
User and Group Management 16
WebSphere Portal 15–16

P

Page 18
Page Aggregation 118
Page Designer 966
page designer 418
page layout 558, 595
Page transformation 16
pages 945
palette 434–435
Palette view 434
param 364
Parameter summary 168
passive credentials 636
patterns 1
PCs to PDAs 48
person.tld 147
Personalization 6, 16
Pervasive computing 6
PlaceBarInclude.jsp 973
Portal access control 34
portal administrator 20
Portal Configuration 984, 986
Portal configuration 945
Portal Designer 952
Portal Document Manager 36
Portal Framework 7–8
portal layout 935
Portal NLS administration 390
Portal page 48
portal page 48
Portal project
  Adding portlets 950
  Apply new themes and skins 962

Change 936
configuration 936
Create 936
Deploy 939
Edit a new theme 962
Edit styles 962
Editing a theme 965
Export 939
Host name 939
Import 936, 939
Import a portlet 950
Label 946
Layout 936, 945
look and feel 962
Modify 936
Navigation 945
new label 946
new themes 961
page layout 949
portal server 940
portlets 952
publish 962
scenario 936
server connection 937
Test 936
wizard 937
portal project 962
Portal services 16
Portal technology 1–2
portal themes 961
Portal Toolkit 40
Portal Tools 52
Portals
First generation portals 3
Second generation portals 3
Third generation portals 3
portals 1, 3
Portlet 18, 49, 121
portlet 337
Portlet API 115, 120
Abstract and concrete portlet applications 155
Action Event Handling 145
ActionEvent 142
ActionListener 142
Attribute storage summary 145
Building a war file 171
Client 125
Configure 119
Control 24–25
Create the Service Factory 623
Custom Services 620
Define the Service 620
destroy(PortletConfig config) 137
destroyConcrete(PortletSettings settings) 136
Edit 119
Event Handling 118
Help 119
Hierarchy 120
Implement the Service 621
init(PortletConfig config) 135
initConcrete(PortletSettings settings) 135
Inter-Portlet Communication 118
Listeners 145
login(PortletRequest request) 136
logout(PortletSession session) 136
Mode 23
Model 24–25
Model View Control architecture 23
ModeModifier 144
Page Aggregation 118
Parameter summary 168
Portlet 121
Portlet deployment 155
Portlet JSPs 146
Portlet life cycle 115
Portlet lifecycle 134
Portlet messaging 145
Portlet MVC architecture 24
Portlet MVC Sample 25
Portlet Services 146
Portlet Tag Library 147
portlet terms 22
Portlet window 22
portlet.xml 160
PortletAdapter 121
PortletApplicationSettings object 129
PortletApplicationSettingsAttributesListener 141
PortletConfig object 126
PortletContext object 127
PortletData object 130
PortletException 132
PortletLog object 131
PortletPageListener 138
PortletRequest 121
PortletResponse 123
PortletSession object 124
PortletSessionListener 139

Archived

PortletSettings object 128
PortletSettingsAttributesListener 141
PortletTitleListener 137
PortletURI 133, 142
PortletWindow object 132
PropertyListener 145
Real Estate 117
Register the Service 623
Security 119
service(PortletRequest request, PortletResponse response) 136
Servlets vs. Portlets 117
State 22
Test the service 624
UID Guidelines 171
UnavailableException 132
User object 133
View 24–25, 119
web.xml 157
Web.xml and Portlet.xml relationship 170
What is a portlet 116
What is a Portlet Application 116
WindowListener 140
Portlet application 18, 116
Portlet applications 49
Portlet debugging 901
Fix compile errors 902
Portlet deployment 155
portlet development 593
Portlet events 50
Portlet life cycle 115, 134
Portlet lifecycle 134
Portlet messaging 145, 226, 743
Portlet messaging versus cooperative portlets 743
Portlet modes
Configure 50
Edit 50
Help 50
View 50
portlet modes 254
Portlet MVC architecture 24
Portlet MVC sample 25
Portlet NLS Administration 386
Portlet NLS administration 386
Portlet Project 594
Portlet Project (JSR 168) 648
Portlet Project (JSR168) 552
portlet services 617
Portlet solution patterns 41
Portlet states 50
Maximized 50
Minimized 50
Normal 50
portlet terms 22
Portlet to remote application 42
Portlet to Web application 43
Portlet window 22
Portlet Wiring Tool 844
portlet wiring tool 778
portlet.tld 146
portlet.xml 160
PortletAction 208
portlet-app 336
PortletApplicationSettings object 129
PortletApplicationSettingsAttributesListener 141
portletConfig 361
PortletConfig object 126
PortletContext object 127
PortletData object 130
PortletException 132
PortletLog object 131
portletMode 362–363
portlet-name 337
PortletPageListener 138
PortletRequest 121
PortletResponse 123
Portlets 21
Portlet 121
Portlet API 120
Portlet deployment 155
Portlet JSPs 146
Portlet life cycle 134
Portlet MVC architecture 24
Portlet MVC Sample 25
Portlet Services 146
portlet terms 22
Portlet window 22
portlet.xml 160
PortletAdapter 121
PortletApplicationSettings object 129
PortletApplicationSettingsAttributesListener 141
PortletConfig object 126
PortletContext object 127
PortletData object 130
PortletException 132
PortletLog object 131
PortletPageListener 138

Archived

PortletRequest 121
PortletResponse 123
PortletSession object 124
PortletSessionListener 139
PortletSettings object 128
PortletSettingsAttributesListener 141
PortletTitleListener 137
PortletURI 133, 142
PortletWindow object 132
Servlets versus Portlets 117
Web.xml and Portlet.xml relationship 170
What is a Portlet Application? 116
WindowListener 140
portlets 421
PortletSession object 124
PortletSessionListener 139
PortletSettings 128
PortletSettingsAttributeListener 141
PortletSettingsAttributesListener 141
PortletTitleListener 137
PortletURI 133, 142
Privileged Users 19
processAction 643
Producer 889
Producer portal 886
Producer portals 889
product installation 53
programmatic approach 745
properties 18
properties view 983
PropertyBrokerService 617

R

RAD 962
rates 18
Rational Application Developer 51, 430, 988
recommendations 935
Redbooks Web site 1007
Contact us xxii
Register 892
Register the Service 623
registration 892
relational database 689, 724
remote content 9
Remote Portlets 899
Remote portlets
Consumer 886
Local portlets 886
Markup 888
portals 889
Portlet Management 888
Producer 886
Registration 888
sample scenario 885
Service Description 887
standard portlets 886
Users 886
withdraw a portlet 889
WSDL 887
removeAttribute 359
render action 356
render method 356
renderRequest 361
renderResponse 361
RenderResponse object 355
renderURL 361
reset 360
resetBuffer 360
Resource bundles 373
Resources 171
Resume 913
retrieve credentials 641
Reverse proxy 11
Rich Site Summary 12
RSS 12
Run on Server 443
runtime errors 902

S

Samples Gallery 54
Search 6, 16
search and taxonomy 7
search functionality 35
Second generation portals 3
secure 362–363
secure servlet 644
SecureCredentialSlot 649
SecureServletEAR 645
Security 899
security options 899
security server 11
sendError 360
sendRedirect 360
server configuration 943
Servers 11
Service Provider Interface 34

Archived

ServletResponse 123
Servlets 353
servlets 421
Servlets versus portlets 117
Servlets vs. Portlets 117
Set Style Properties 971
setAttribute 359
setBufferSize 360
setCharacterEncoding 358
setContentLength 360
setContentType 360
setDataHeader 360
setHeader 360
setIntHeader 360
setLocale 360
setProperties 775
setStatus 360
Setting NLS titles 390
settingsAttribute 149
settingsLoop 149
simple action Strings 761
Single Sign-On 630
site analytics 7
Site usage 6
Skin 985–986
Skin Properties 986
Skins 963
skins 963
sql 366
SSL 11, 899
SSL support 11
SSO 900
Standard MVC architecture 23
State 22
store credentials 641
Struts 335, 429, 836
Struts Action 420
struts action 837
Struts applications 430
  Struts Portlet Framework 38
Struts portlet configuration 430
Struts Portlet Framework 54
Struts portlets
  Action phase 420
  ActionForm 416
  actions 421
  ActionServlet 417
  application development 419
  business logic 416
  components 417
  configuration file 417
  controller 416
  CSS file 424
  execute method 423
  external applications 416
  forwards 422
  framework 415
  IBM Portlet API 420
  Initialization 420
  Internationalization 418
  JSP tags 418
  JSR 168 API 420
  legacy systems 416
  life cycle 418
  Logic 418
  look and feel 424
  mappings 423
  Markup support 424
  model 416
  modular applications 416
  portlet communication 420
  portlet framework 415
  portlet life cycle 418
  processing 422
  Render phase 420
  render phase 422
  rendering 421
  Tag libraries 418
  tags 424
  view 416
struts_source 420
struts-config.xml 436
style sheet 976
stylesheet.css 560
Stylesheets 423
supports 338
Switch Active Document 969
symbols 966
Syndicated content 12

T

TAM 11
target portlet application 782
Target server 645
techniques 901
technology 3
Test Environment 52, 648

Archived

Test the service 624
TestPortalServer 962, 984, 986
text 152
The Portlet Wiring Tool 40
theme 963, 965, 983–984
themes and skins 989
Third generation portals 3
Title 947
Tooling 17
top 420
Transcoding technology 37
Translating Resource Bundles 379
Translating whole resources 373, 383
translation 850
Tutorials Gallery 55

U

UID Guidelines 171
URIAction 148
URIParameter 148
URL 35
URL Generation, Processing and Mappings 35
URL Mappings 35
user 891
User object 133
User-friendly Portal URLs 35
Users 19
Using resource bundles 373

V

validators 551
value 364
value change events 551
var 362–363
vault segment 632
View 174
View mode 643

W

WAR file 844
Web browser 988
Web deployment descriptors 426
Web Diagram 434
Web perspective 644
Web service 606
Web Service client 857
Web Service Description Language 744
Web Services 17, 857, 887
  JavaBean 859
  Sample scenario 858
  wizard 858
  WSDL 858
Web Services Client
  Create a Web Services client portlet 874
Web Services Description Language 887
Web Services for Remote Portals 12
Web Services for Remote Portlets 885
web.xml 157, 337, 435
Web-based content 8
WEB-INF 433
WebSEAL 11
WebSphere 1
WebSphere Portal 18, 54, 899, 990
WebSphere Portal Property Broker 743
WebSphere Portal V5.1 885
What is a portlet 116
What is a Portlet Application 116
Window events 51
WindowListener 140
windowState 361, 363
wizard 430
wizards 434
WML 355
workbench 52, 988
Working with characters 373, 392
WpsStrutsPortlet 433
WSRP 366, 885, 899
WSRP services 887
WSRP styles 366

X

xml 365
XML feed 49

Archived

Archived

Redbooks

IBM Rational Application Developer V6 Portlet Application Development and Portal Tools

(1.5" spine)
1.5"<-> 1.998"
789 <->1051 pages

Archive

IBM Rational Application Developer V6 Portlet Application Development and Portal Tools



Learn how to develop MVC, Struts and JavaServer Faces portlet applications

Learn about IBM Portlet API and the JSR 168 standard API

Access Web Services and secure Web applications

This IBM Redbook provides an overview and hands-on scenarios to help you design, develop and implement portlet applications using Rational Application Developer V6.0 and the provided Portal Tools. The sample scenarios included in this redbook target Business-to-Employee (B2E) enterprise applications, but most of the scenarios presented will also apply to Business-to-Consumer (B2C) applications.

You will find step-by-step examples and scenarios showing ways to integrate your enterprise applications into an IBM WebSphere Portal environment using the WebSphere Portal APIs provided by the Portal Tools to develop portlets. You will also learn how to extend your portlet capabilities to use advanced functions such as cooperative portlets, internationalization, action events, using the Credential Vault to enable Single Sign-On, Web Services, remote portlets, portal design and portlet debugging capabilities. Elements of the Portlet API and the standard JSR168 API are described and sample code is provided. The scenarios included in this redbook can be used to learn about portlet programming and as a basis for your own portlet applications. You will also find scenarios describing recommended ways to develop portlets and portlet applications that follow the MVC design pattern, the Struts framework and JavaServer Faces technology.

Basic knowledge of Java technologies such as servlets, JavaBeans, EJBs, JavaServer Pages (JSPs), as well as of XML applications and the terminology used in Web publishing, is assumed.

SG24-6681-00 ISBN 073849352X

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks