

Creating a JSR 168 portlet for use by diverse portals using Web Services for Remote Portlets

Part 1: Creating the portlet using multiple methodologies

Karl Bishop, Senior Software Engineer, IBM
Ron Lynn, Senior Software Engineer, IBM

October, 2005

© Copyright International Business Machines Corporation 2005. All rights reserved.

In this two part adventure, you walk the path of creating a portlet using the JSR 168 specification. You also see how to access a database from the portlet using SQL. With a twist of plot in part 2, you make this portlet available to other portals through Web Services for Remote Portlets (WSRP). Finally, you see how to access the portlet in a portal running under IBM® WebSphere® Portal Version 5.1 (hereafter called WebSphere Portal).

For every adventure or job that needs doing, certain tools come in handy; whether axe or rope, all have their uses. For our purpose, we use Rational® Software Architect Version 6.0 for portlet development. You could just as easily use Rational Application Developer V6.0. Or, if you prefer, you could use your favorite text editor. This tutorial includes instructions for using both the Rational toolset and command line tools.

The intention of the first part of this escapade is to illuminate the novice portlet developer in the use of the JSR 168 API. In the second part, we reveal the beauty of WSRP.

This guided adventure is for any stout adventurer with beginner level Java programming experience who might also have some familiarity with portlets and portals. We have kept the assumptions about portal development to a minimum.

Introduction	2
First steps upon the road.....	3
The Rational IDE path.....	4
Starting out	4
Building the portlet.....	9
The command line path.....	11
Starting out	11
Building the portlet.....	12
Installing the portlet: the road forks again	12
Using the WebSphere Portal administrator's GUI	12
Using xmlAccess.....	14
Bonus path: Using the Rational IDE.....	16
Deploying the portlet to a page	17
Finishing the portlet	19
Updating the portlet.....	24
Using the GUI	25
Using xmlAccess.....	27
Working with databases	27
Creating a sample database	27
Connecting to the database.....	27
Removing database access	28
Making a data source for the sample database.....	28
Accessing the Admin server.....	28
Setting up a data source.....	29
Creating the JDBC provider.....	30
Creating the data source	31
Conclusion.....	33
Resources	33
Download	33
About the authors	34

Introduction

Noble adventurer, we see you've made it past the abstract to the body of this little adventure. You are a stout and steadfast individual to have made it this far. It is our purpose to usher you through the creation of an SQL query portlet made from the raw ingredients of the JSR 168 API. The issuance of ad hoc SQL queries is a key ingredient in many a portlet development foray. We will lay out all the steps and code needed for this portlet in order to assure your success in this task.

After you have toiled in creation, you will exhibit your creation in two venues (part 2 of this series): the first being the host portal in which you install the portlet, and the second being the client of a WSRP facsimile of the portlet. We do hope you enjoy and find educational the following.

First steps upon the road

No matter how you choose to build your portlet, the end result will be the same. The environment in which you build should match your own personal taste in development.

Traditionally, we have developed portlets using emacs and command line tools for building, testing, and packaging. Within the last year or so, our group standardized on Rational Software Architect (hereafter called Architect) or Rational Application Developer (hereafter called Application Developer). The transition from command line to an Integrated Development Environment (IDE) wasn't always easy, but we must admit that the Rational toolset is very powerful, and we are very pleased with the results of our transition.

All that said, we do not want to focus on the tooling used, but rather on what we are building. We will give you the information needed to be successful for either environment. To that end, we pause here to consider the road ahead.

"Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that, the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
two roads diverged in a wood, and I --
I took the one less traveled by,
And that has made all the difference."

Robert Frost

You start the journey with a fork in the road. Unlike Robert Frost's traveler you have the opportunity to take both paths, should you so desire.

Fork A	Fork B
The first path leads to bright lights and	From the other path emanates the musty

shiny GUIs with all the bells and whistles which whisk you away to your destination. The Rational toolset path will get you there in style for a moderate investment in learning something new.	smell of aging paper manuals and leads one to the venerable, well worn, and comfortable command line , which, although less used today, can still get you to your destination.
---	--

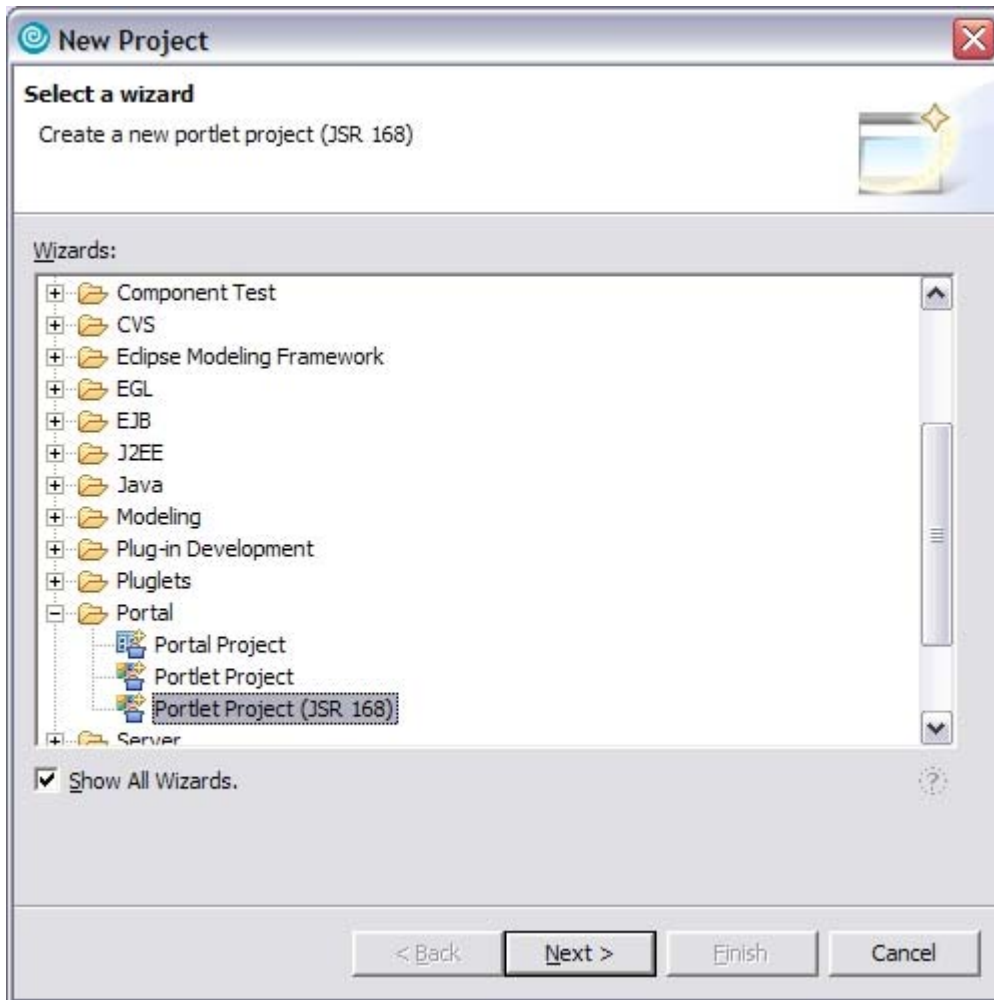
The Rational IDE path

Nothing like pulling a new toy from its crinkling packaging with the smell of fresh injection molded plastic. Well, as long as there aren't too many of those frustrating twist ties holding the toy in. For us, there were a few of those twist ties, but they were overcome with time and now we have made the Rational IDE a comfortable working environment. Since you have chosen to walk this path, we expect that you have some level of comfort in the Rational IDE (either with Architect or Application Developer). You should know how to change perspectives, and be able to get around enough to open files for editing and such.

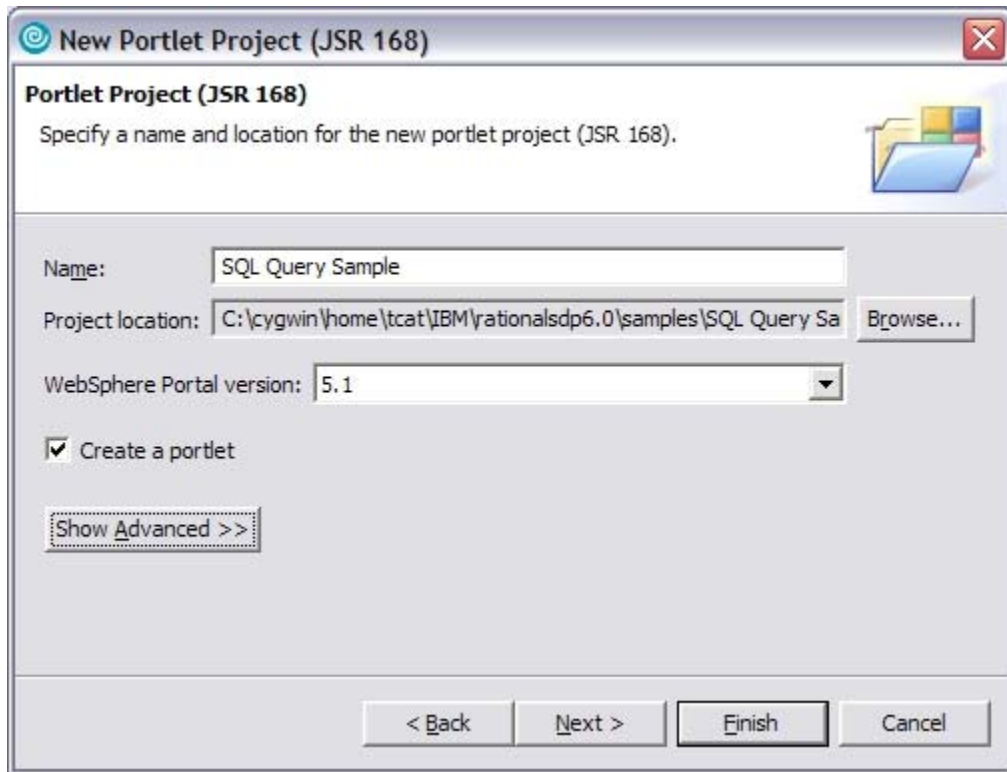
Starting out

You start with creating a new project in your chosen Rational IDE using the Portlet Project wizard for JSR 168 portlets. This wizard guides you through adding various options you could set for your portlet. It generates a skeleton of the portlet that you fill in. The neat part about this is that the skeleton is a working portlet, so that you can deploy it into a portal and play with it.

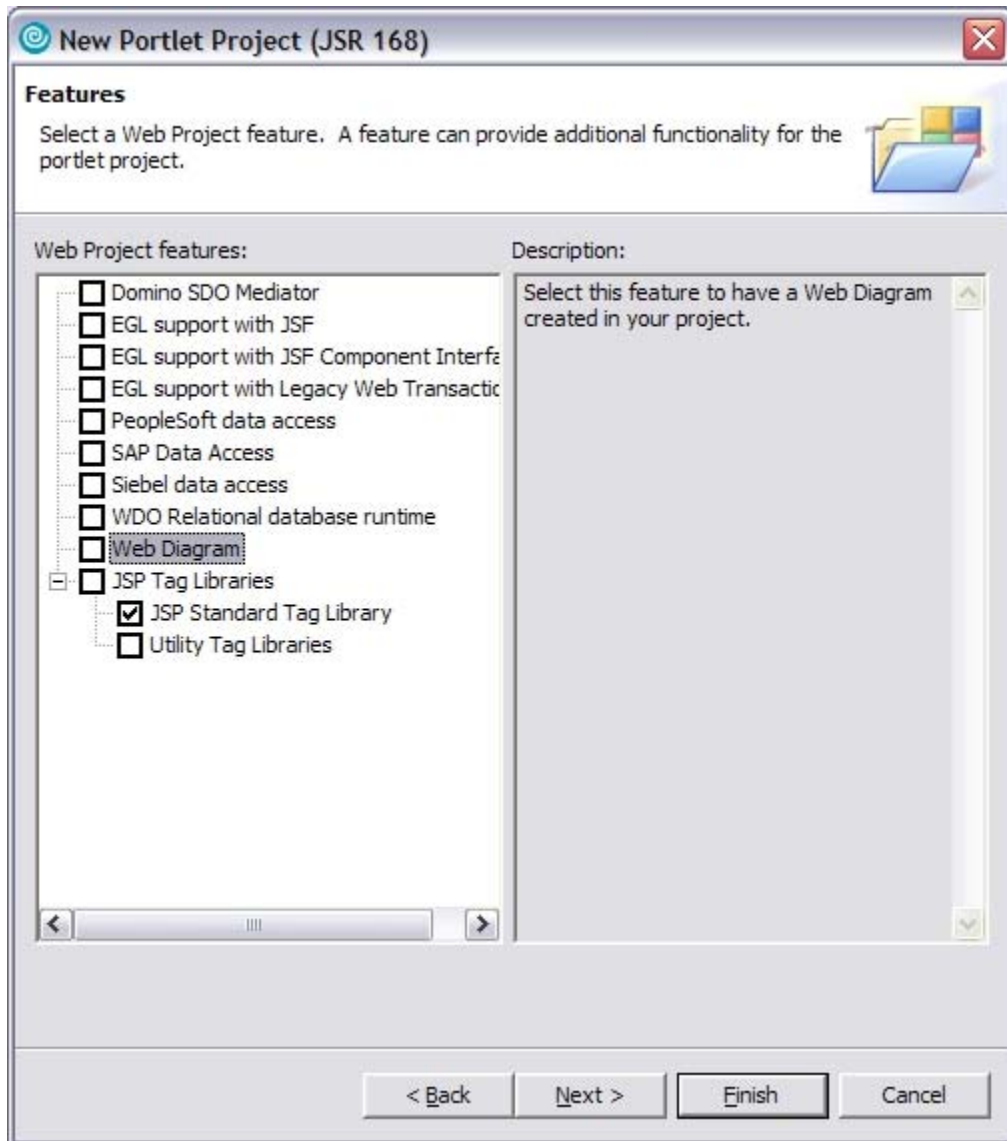
1. In your chosen Rational IDE, select **File => New => Project**.
2. Verify that the **Show All Wizards** checkbox is checked.
3. Expand the **Portal** folder.
4. Select **Portlet Project (JSR 168)**.



5. Click the **Next** button.
6. If asked if you would like to enable the **Portal Development** capability, click **OK**.
7. Type the name of the project in the Name field: `SQL Query Sample`
The name is used to create a directory for your portlet project.
8. Click **Next**.



9. On the **Portlet Type** page of the Wizard, select the **Basic Portlet (JSR 168)** portlet type. This option creates base Java code and JSPs for the portlet.
10. Click **Next**.
11. On the **Features** page, select only the **JSP Standard Tag Library**. You might not use it, but if you do need it you will have it available.



12. Click **Next**.
13. On the **Portlet Settings** page, change the Package prefix and Class prefix elements in Code generation. Strictly speaking, you don't have to do this but, at the very least, we recommend you make the Package prefix something that is uniquely yours. In this case, we set the Package prefix to `com.ibm.portlets.sample` and the Class prefix to `SQLQuery`.

New Portlet Project (JSR 168)

Portlet Settings
Define the general settings of the portlet.

General

Portlet name: SQL Query Sample

Display name: SQL Query Sample portlet

Description:

Internationalization

Locale: en English

Display name: SQL Query Sample portlet

Description:

Code generation options

Change code generation options

Package prefix: com.ibm.portlets.sample

Class prefix: SQLQuery

< Back Next > Finish Cancel

14. Click **Next** or click **Finish**.

For this exercise, you do not need to change any of the other defaults or to enable any other options past this screen. If you choose to click through the rest of the wizard screens, you see other options and features that you can enable. For this tutorial, please take the defaults.

15. After you click **Finish**, the Rational IDE generates your starter project and code. If you are asked to change perspective to the Web perspective, click **Yes**.

After the project is created, it opens the `SQLQueryView.jsp` file for editing. You need to rewrite the `SQLQueryView.jsp` to look like the following. So, switch to the **Source** view of the JSP, and replace what is there with the code in Listing 1.

Listing 1. SQLQueryView.jsp revised coding

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

<%
SQLQuerySessionBean sessionBean = (SQLQuerySessionBean)
renderRequest.getPortletSession().getAttribute(SQLQuery.SESSION_BEAN);
String formText = sessionBean.getFormText();
%>

<h3 style="margin-bottom: 3px">SQL Query</h3>

<div style="margin: 12px; margin-bottom: 36px">
  <form method="POST" action="<portlet:actionURL/>">
    <label for="<%=SQLQuery.FORM_TEXT%>">Enter SQL query:
    </label>
    <br />
    <textarea rows="5" cols="40"
              name="<%=SQLQuery.FORM_TEXT%>"><%=formText%>
    </textarea>
    <br />
    <input name="<%=SQLQuery.FORM_SUBMIT%>" type="submit"
           value="Submit" />
  </form>
</div>
```

Now you have a JSP that will display a single text area and a button. The user types his or her query into the text area and then clicks the button to submit the query. At this point you are using the pre-created session bean and portlet code. You could build and run it and you will have a portlet which remembers the query and displays it back to the user in the text area.

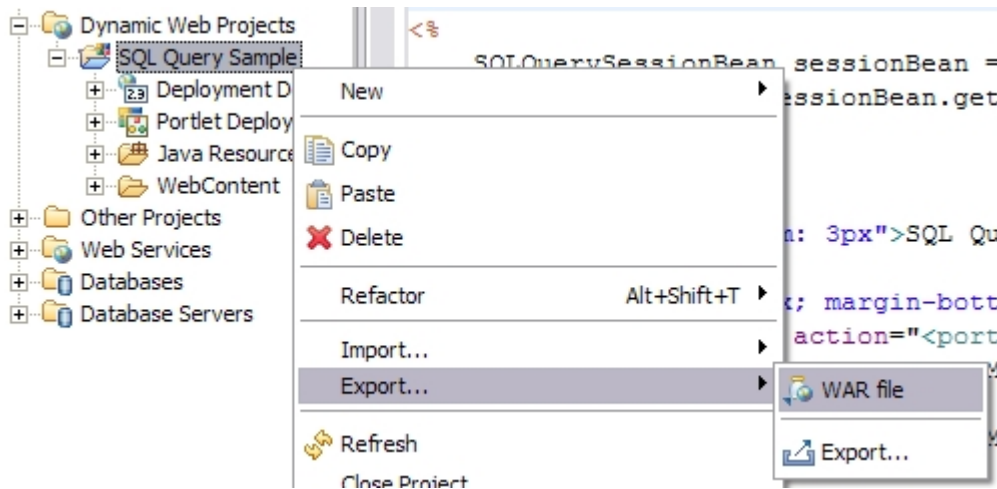
For new portlet developers, this is one of the most wonderful aspects of using the Rational tools; you don't have to start from a blank sheet of paper. Rather like a coloring book, you are given an outline which you can fill in. The outline itself is complete, but your embellishments make it so much more than how it started. You will finish filling in the portlet and building the results display a little later, but for now, let's learn how to build this skeleton portlet.

Building the portlet

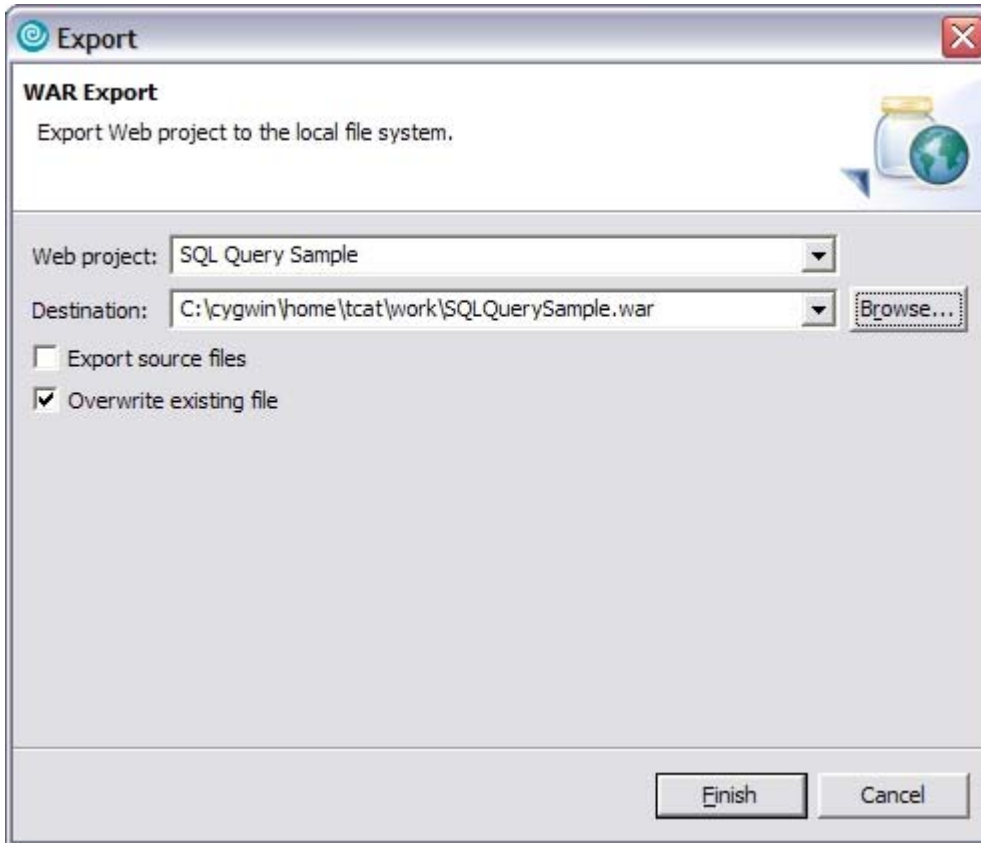
Compiling and packaging the portlet for deployment is a simple feat when you use the Rational environment. To review a basic portlet concept, portlets are packaged into Web Archive files, or WAR files, which are deployed into the portal, and then laid out on portal pages.

To build the portlet:

1. Switch to the Web perspective (if you're not already in it).
2. Right click the project under Dynamic Web Project, called **SQL Query Sample**.
3. Select **Export => WAR file**.



4. Select a Destination and name for the WAR. For example:
c:\cygwin\home\tcat\work\SQLQuerySample.war
5. Optionally, check the Overwrite existing file checkbox. If you don't do this you need to choose a unique name for your WAR file.
6. Click **Finish**.



You now have a deployable portlet! The next step is to deploy it so you can test it out. If you're on the Rational path, please skip ahead to [Installing the portlet](#).

The command line path

We do enjoy settling into our chairs, stretching our fingers out and putting them to well worn keys. It is a comfortable place to be and one that we are quite familiar with. To be able to build the sample portlet from the command line, you need a directory structure to put the code into, a JDK, a text editor, and a JAR file from your WebSphere Portal installation. We always try to use the same JDK that the portal is using. This keeps us from straying to the use of features in newer JDKs.

Starting out

1. [Download the code for this portlet](#) so you don't have to type everything.
2. Extract the contents, maintaining the directory structure to see what is where. You see the Java code in the JavaSource directory and JSPs in the WebContent directory, along with the portlet.xml and web.xml files.
3. Notice that there are two JAR files in the WebContent/WEB-INF/lib directory. These are for JSTL, in case you want to play with it. We don't use JSTL in this tutorial, but you would probably want to use it in a real application, instead of Java scriptlets in the JSP code.

Building the portlet

To build the portlet from the command line:

1. Get to the JavaSource directory, and compiling the source code using this command:

```
javac -cp <pathToJAR>/portlet.jar com/ibm/portlets/sample/*.java
```

2. Copy the class files to the corresponding directory under WebContent:

```
WebContent/WEB-INF/classes/com/ibm/portlets/sample
```

You could, of course, create a JAR file from the class files and put it into the WebContent/WEB-INF/lib directory instead. You could also use a script, Ant, or Maven to do these build steps.

3. Let's go ahead and build the WAR file. Change directory to the WebContent directory and create the WAR file.

```
cd WebContent
jar -cvf ../SQLQuerySample.war *
```

The result is a WAR file ready for deployment. Continue with the next step to install and deploy the portlet.

Installing the portlet: the road forks again

There are multiple ways to deploy portlets:

- [Using the WebSphere Portal GUI](#)
- [Using an xmlAccess script](#)
- [Using the Rational IDE tools](#)

We'll walk through them all, and you can choose which suits you and your environment.

Using the WebSphere Portal administrator's GUI

1. Log onto your portal as an administrator. This is most easily accomplished by going to `http://<portal.server.name>/wps/myportal`. Type the **User ID** and **Password**, then click the **Log in** button at the bottom. (The default portal administrator user ID is `wpsadmin`.)

Login Portlet

User ID:

Password:

Not registered? [Sign up](#)

2. Click the **Administration** link. So, where is this link? That is part of the adventure of portal. The **Administration** link placement is completely theme dependent. In the case of the default themes, it should be fairly obvious and is usually included as one of the page tabs. If you're working from a theme developed elsewhere and can't find it, ask your portal administrator. (Some sites hide the **Administration** link behind an icon.)
3. In Portal Administration, click **Portlet Management => Web Modules**.
4. Click the **Install** button.

Manage Web Modules

Search by: Search:

Web modules Click Install to install a Web module. Select a Web module to view portlets. Click Delete to remove the Web module from your portal or click Assign A work with the Web module.

Page 1 of 8

Name	Status
Managepages.war	
lwpeoplefinder.war	

5. Click **Browse** and locate the WAR file you created, or type in the complete path to it.

Installing a Web module, Step 1: Select WAR file.

 Click the Browse button to specify the location of the \ continue or the Cancel button to go back to the Web mo

Directory:

6. Click **Next**.

Manage Web Modules



Installing a Web module, Step 2: View WAR file contents.

 The selected WAR file contents are displayed below. Select the Finish button to install the WAR. Cancel button to go back to the Web module page.

Portlet applications	Portlets
com.ibm.portlets.sample.SQLQuery.778f228960	SQL Query Sample portlet

7. Verify that you've chosen to install the correct WAR.
8. Click **Finish** . You should see a success message like this:

Manage Web Modules

 EJPAQ1332I: Web module was successfully installed.

Using xmlAccess

To use xmlAccess to install the portlet, you need to write an xmlAccess script to install our portlet. (You can use the same script later in this tutorial to update the portlet.)

The xmlAccess script you need to use to install the portlet looks like the code in Listing 2.

Listing 2. xmlAccess script to install the portlet

```
<?xml version="1.0" encoding="UTF-8"?>
<request
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="PortalConfig_1.2.xsd"
  type="update"
  create-oids="true">
  <portal action="locate">
    <web-app action="update" active="true"
      uid="com.ibm.portlets.sample.SQLQuery.778f228960.webmod">
      <url>file:///tmp/SQLQuerySample.war</url>
      <servlet action="update" active="true"
        referenceid="SQL Query Sample.servlet"/>
      <portlet-app action="update" active="true"
        uid="com.ibm.portlets.sample.SQLQuery.778f228960">
        <portlet action="update" active="true"
          name="SQL Query Sample" />
      </portlet-app>
    </web-app>
  </portal>
</request>
```

We say “looks like” because we can’t predict what your unique identifiers (UIDs) will be. The unique identifier `com.ibm.portlets.sample.SQLQuery.778f228960`, appears in two places. The Portlet wizard places this unique identifier into the `portlet.xml` file for you. If you hand-coded `portlet.xml`, you need to map the **uid** attribute from the **portlet-app** element in that file to the **uid** attribute of the **portlet-app** and **web-app** elements in the `xmlAccess` script.

1. Next, change the **url** element of the script so it point it to the location of the WAR file you created earlier. For example, on our Linux machine, it is in `/tmp`. Specify the proper path in for your environment.
2. Once you have written this little script, save the file as `updateSQL.xml`.

Next, you run the script to install the portlet. To run the script, you need to know where your portal `bin` directory is installed. On our Linux machine it’s under `/opt/IBM/WebSphere/PortalServer/`. On a Windows machine it might be under `c:\WebSphere\PortalServer\`.

```
# ./xmlaccess.sh -in <xmlAccess script name with path> -user wpsadmin -  
pwd <yourPassword> -url <WP server>:9081/wps/config
```

The script is a `.bat` file in Windows. If you copy the `xmlAccess` script into the `bin` directory, you can use this command to install the portlet:

```
# ./xmlaccess.sh -in updateSQL.xml -user wpsadmin -pwd i'mnottelling -  
url localhost:9081/wps/config
```

If this all works as planned, you see status messages scrolling by, and finally one that says:

```
<status element="all" result="ok"/>
```

So, what if things don’t go as planned? Start with reading the error message carefully. They can be somewhat cryptic, but they can help. When in doubt, the first place to look is the `xmlAccess` script we just wrote by hand:

- Are all the XML tags closed?
- Do the UIDs match what’s in the `portlet.xml`?
- Is the path to the WAR file correct?

It could be that the `xmlAccess` script is fine, but something is wrong with your WAR file. You could try installing the portlet [using the GUI](#) or [using the Rational interface](#). If that works, you have eliminated the WAR as being part of the problem.

Bonus path: Using the Rational IDE

Some may take the road less traveled, but when you really need to get somewhere fast, take the easiest route. For those of you using the Rational IDE, the third option for deploying portlets is to let the Rational tools do it for you.

To enable the Rational IDE to work its magic, you simply tell it the basics of the server environment you wish to deploy to, and then it's a matter of a few clicks to deployment nirvana.

First, right-click on your portlet project, and select **Deploy Portlet**.

The first time you travel this path, you need to create a deployment server definition.

1. Click on **New...** to create a server using the Wizard.

When we developed this exercise, the WebSphere Portal server was on our local Linux machine, so we selected `localhost` as the server's host name. For remote servers or custom installations, you need to adjust these settings to match your environment. There are many options to choose from when defining a deployment server (which might stray us off our immediate path). The Rational help has many more details for your reading enjoyment.

2. Define a New Server.
 - a. Host Name: `localhost`
 - b. Server Type: WebSphere Portal v5.1 for Import, Export, and Deploy
3. Click **Next**.
4. WebSphere Portal Settings. Define the settings for your environment. If you performed a standard WebSphere Portal installation, you can keep most of the defaults. Otherwise, you need to update the settings to match your system.
 - **Server HTTP port:** 9081
 - **Base URI:** `/wps`
 - **Default Page:** `/portal`
 - **Personalized home page:** `/myportal`
 - **Install location:** {Enter your Portal Server install location}
 - **WP Admin User ID:** {Enter your WebSphere Portal administrator's user ID}
 - **WP Admin Password:** {Enter your WebSphere Portal administrator's password}
5. Publishing settings:
 - **Transfer Method:**
For locally hosted server, select **Local Copy**.
For remote servers, select **FTP**.
 - **Web Application:** {Enter the location of the installed `wps.ear` file on your WebSphere Application Server installation}
For example:
`/opt/IBM/WebSphere/AppServer/installedApps/<node>/wps.ear`

- **Library:** {Enter the location of the portal's shared apps directory}
For example:
`/opt/IBM/WebSphere/PortalServer/shared/app`
6. Add Remove Projects.
Select the Portlet project EAR(s) or sub-projects that you want to be deployable on this server. For this example you simply select our single SQL project's EAR and add it to the Configured projects area.
 7. Click **Finish** to complete the server definition.

Now that your server is defined, you can deploy the portlet to it.

1. Select Server:
Portlet Project: Select the project you wish to deploy.
Server to use: Select the server you just defined (WebSphere Portal V5.1 for Import, Export, and Deploy @ localhost)
2. Portlets:
The only item you typically want to select on this page is the checkbox to automatically overwrite portlets. (Even if you don't set it, you are prompted if you want to overwrite the existing portlets anyway.) One other option to consider is if you want to simply update the potentially existing portlets, or if you want to Remove and Deploy them. Typically, you want to simply do an Update. The only reason we can think of for wanting to do a remove/deploy is if you had made changes to the init parameters and wanted to test them out. If you elect to do this, you need to re-add the portlets on the pages again.

A few seconds later, your portlets are deployed on the server. It's really quick and simple. Let's not even call this a path. Let's think of it more like a deployment moving side-walk, like you find at an airport.

Deploying the portlet to a page

There are two paths you could take to deploy a portlet on a page. Because you only need to deploy it once, and then you can just update the portlet, we present the graphical method. If you'd like to use xmlAccess for portlet deployment, then we suggest you feast your eyes upon the InfoCenter. See the section which explains xmlAccess and provides samples for you to devour.

1. If you are not already logged onto your default portal running under WebSphere Portal, log on now as an administrator: For example, log on here as wpsadmin:
`http://<portal.server.name>/wps/myportal`
2. In portal administration, click the **Administration** link.
3. Click **Portal User Interface => Manage Pages**.
4. Click the **My Portal** link.
5. Find the page you'd like to deploy your portlet to or create a new page.
For this exercise, you could use the **Welcome** page.

Manage Pages



Use the controls below to work with your pages. Browse or search for pages to work with. Click New to create new pages, labels and urls. Activate and deactivate pages, re-order, edit properties and layout, move pages, assign permissions and delete. For more information, click Help.


Search by: Title starts with Search: Search

Content Root > My Portal

Pages in My Portal Add, Edit, Delete, and Reorder pages

New page New label New url

Title	Unique name	Status
tcat test	6_0_1BP	Active

6. Click the **Edit Page Layout** icon (.
7. Click an **Add Portlets** button. There could be several on the page so pick one where you'd like the portlet to be displayed.
8. Find our portlet. The easiest way is to do that is:
 - a. Select **Search by: Title starts with**
 - b. In the **Search:** field, enter `SQL`, and then click the **Search** button. Of course, if you named your portlet something other than `SQL`, enter that name in the Search field instead.

Edit Layout

Search by: Title starts with Search: SQL Search

9. Click the **Select** checkbox next to the portlet you'd like to place on the page, and then click **OK**.

Edit Layout

Search by: Title starts with Search: SQL

Select	Portlet Title	Description	Unique name
<input checked="" type="checkbox"/>	SQL Query Sample portlet		

Page 1 of 1

OK Cancel

10. Click **Done**.
11. After the portlet is deployed to a page, you can go play with it!

The portlet doesn't do a whole lot at this point, but you do have a portlet that it working on the page. It stores a query in the session and puts it back in the text area for the user to

see. This is all the code necessary for user interaction. The next step is to finish the portlet by adding some code to implement the query and render the results.

Finishing the portlet

To finish this portlet, you add the query logic and some code to display results. We've included all the code in the [download](#). If you have not already downloaded this code, do that now. For those of you following the Rational path, you could replace what the wizard produced with the modules in the download, so that we're all on the same page in the rest of the discussion. What we've provided is complete, and it will save you some typing. It's pretty straight-forward, although we changed some of the default code that Portlet wizard produced to make it a little easier to follow. We have commented the code liberally to clarify its intention.

The display code shows only the first 10 rows of data returned from the Select statement. You might want to add pagination through the result set, code to provide an option to update the data, and provide a way of switching to a different data source. The purpose of this tutorial, however, is to walk through creating a complete application, without all the extras. So, we stick to creating a simple portlet that is complete enough to give you a feel for JSR 168, and one that will let you do something useful.

Start with updating the doView method of the portlet. You call a new method, runQuery, and pass in the RenderRequest and the SQLQuerySessionBean.

Listing 3. The doView Method

```
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
    SQLQuerySessionBean sessionBean = getSessionBean(request);
    if( null == sessionBean ) {
        response.getWriter().println(
            "<b>NO PORTLET SESSION YET</b>");
        return;
    }
    //issue the query
    runQuery( request, sessionBean );

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

We simplified the call to the request dispatcher somewhat. You don't have to do this; the generated code is fine. It saves a few lines of finger exercises for those who choose the non-GUI path for development.

If you have written portlets to the IBM API (versus the JSR 168 Portlet Specification), you might notice a difference in program flow. In the IBM API, you would normally run queries and such in the event processing phase (it's called the action phase in JSR 168 parlance) of the portlet—not in the render phase. In JSR 168 portlets, this isn't possible to do. So, we move this processing to the render phase. The idea is that such calculations could be done and optimized better during the render phase. So, if this were a long running query in a real application, you might want to cache the results for later retrieval. Likewise, if we were going to implement pagination, you might want to cache results to speed up subsequent calls to doView.

Ok, you with the scraggly beard mumbling in the back. We heard your question. For those of you who didn't hear it, he said, "Why can't we do this processing in the action phase?" Yes, we see those nods of agreement and hear the rattling of swords. It's a good question. The signature of the doView method and the processAction method take in different request objects. The doView method uses the RenderRequest and the processAction method uses the ActionRequest. Attributes set in one are not propagated to the other. So, what the API is whispering to us is that actions should change the state of the portlet. Things which you would store in the session or other longer-lived places should be processed in processAction. Anything needed to render the portlet, such as the results of a query, should happen in doView. Optimization should be applied where appropriate; for example caching query results. Make a little more sense now?

So, what does runQuery do? It simply runs the query that is stored in the session bean, and stores some of the results in the RenderRequest for the JSP to render. Nothing very fancy, but it is something that you may find yourself needing to do in some circumstances.

Listing 4. The runQuery method

```
private void runQuery( RenderRequest request,
                      SQLQuerySessionBean bean ) {
    //-- do a sanity check!
    String sql = bean.getFormText();
    if((null == sql) || "".equals(sql.trim())){
        return;
    }
    //-- Get our datasource
    setError(request, null);

    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {

        //-- get our query and validate it's a select
        if (!sql.toUpperCase().startsWith("SELECT ") ) {
            setError(request, "Invalid SQL Select statement: " + sql);
            return;
        }
    }
```

```

    //-- initialize our datasource
    if( null == datasource ){
        datasource = getDataSource(DATASOURCE_NAME);
    }

    //-- Run our Query if SQL is set.
    // Get connection
    con = datasource.getConnection();
    stmt = con.createStatement();
    rs = stmt.executeQuery( sql );
    setResults(request, rs);
} catch(NamingException ne) {
    setError(request, "DataSource NOT FOUND!");
    return;
} catch (SQLException se) {
    setError(request, "Error on SQL: " + se );
} catch (Exception e) {
    setError(request, "Unknown Error: " + e + "\n" );
} finally {
    try {
        if ( rs != null ){ rs.close(); }
        if ( stmt != null ){ stmt.close(); }
        if ( con != null ){ con.close(); }
    } catch ( Exception e ) {}
}
}
}

```

So, not much magic happening here. We test the parameters, and then grab the data source. We hard coded the name of the data source in the Java code in this example. Of course, this is not a best practice, and in real applications we would externalize the data source in a way that makes sense. For example, you could put it in a properties file, make it an editable property, make it user set-able, make a list of possible data sources with user friendly descriptions, and let the user select the data source. However, for the sake of simplicity in this wee sample, we just set it in the code:

```
public static final String DATASOURCE_NAME = "jdbc/MySampleDS";
```

When you're out there on your system playing with this portlet, you need to point the data source toward one of your own databases. If you don't know how to do this, see the section at end of this tutorial called [Working with databases](#). It provides a brief overview of creating a database, importing some data, and creating the data source in WebSphere Application Server. Listing 5 shows how to initialize the data source:

Listing 5. Initializing the data source

```
protected DataSource getDataSource(String dsname)
    throws NamingException{

    DataSource ds = null;
    Context context = new InitialContext();
    ds = (DataSource)context.lookup(dsname);
    return ds;
}

```

If you've worked with data sources before, you have probably seen this sort of code over and over: look up the data source and return it.

The next step is to implement the query and store the results using `setResults`. And, lastly we clean up the connection and such. We do try to trap some errors which will get shown to the user. The `setResults` method simply places the results into the `RenderRequest` for the JSP to pick up and use. The `setResults` method simply calls two other methods, `setHeaders` and `setRows`. The `setHeaders` method iterates through the results and creates a list to display to the user.

Listing 6. The `setHeaders` method

```
private void setHeaders(RenderRequest request, ResultSet rs)
    throws SQLException {

    ResultSetMetaData meta = rs.getMetaData();
    int columns = meta.getColumnCount();
    ArrayList headers = new ArrayList(columns+1);
    headers.add("&nbsp;");
    for(int i = 1; i <= columns; i++){
        headers.add(meta.getColumnName(i));
    }
    request.setAttribute("HEADERS", headers);
}
```

Yes, Larac, we could have used column labels instead of column names, and it would have been more user-friendly. We didn't; we used column names, and I'm sorry. (For those of you who are wondering, Larac is the UCD devil, I mean angel, who sits on my shoulder and usually talks sense into us. However, what she doesn't know is that there is method to our madness!) We used the column names so that users might have a better idea which columns they can use to limit their selections. A trivial detail, but as you can see we added a list of headers to the `RenderRequest` so that we can let the users know what columns they selected. Let's look at how we store the rows we've selected.

Listing 7. The `setRows` method stores the selected rows

```
private void setRows(RenderRequest request, ResultSet rs)
    throws SQLException {
    //generate 10 rows of data if that much exists
    ResultSetMetaData meta = rs.getMetaData();
    int columns = meta.getColumnCount();
    ArrayList rows = new ArrayList(10);
    int rowNumber = 1;
    while(rs.next() && (rowNumber <= 10)){
        ArrayList row = new ArrayList(columns+1);
        row.add(new Integer(rowNumber));
        for(int i = 1; i <= columns; i++){
            Object o = rs.getObject(i);
            if(null == o){
                row.add("null");
            }else{
                row.add(o);
            }
        }
    }
}
```

```

        rowNumber++;
        rows.add(row);
    }
    request.setAttribute("ROWS", rows);
}

```

We use the same idea here as we did for the headers. We created an ArrayList in which to store the rows, and each row is an ArrayList. We restrict the number of rows shown to ten, and then make the data structure available to the JSP as a RenderRequest attribute. Again, in a real application we would cache these results as an optimization so that we could page through them or let future users access them.

Let's finish off our pretty picture of Java code with a little bit of reality. If we happen upon errors, we set them in the RenderRequest, and render them with the JSP for the user to see.

Listing 8. Rendering errors

```

public void setError(RenderRequest request, String error) {
    if(null == error){
        //reset the error buffer
        request.removeAttribute("ERRORS");
        return;
    }
    StringBuffer errors = (StringBuffer)
        request.getAttribute("ERRORS");
    if(null == errors){
        //initialize the error buffer
        request.setAttribute("ERRORS", new StringBuffer(error));
    }else{
        errors.append("<br />\n");
        errors.append(error);
    }
}
}

```

This reasonably simple code strings the error messages together and put them into the RenderRequest for the JSP to handle. The JSP pulls the ERRORS string from the request and prints it out at the bottom of the page. We did not use JSTL (JSP Standard Template Libraries) in the JSP, although they would have made several things easier. Listing 9 shows only what's been added to the JSP; for the full version see the [download](#).

Listing 9. The JSP code handles the display of data and errors

```

<%
ArrayList headers = (ArrayList)request.getAttribute("HEADERS");
ArrayList rows = (ArrayList)request.getAttribute("ROWS");
if((null != headers) || (null != rows)){
%>
<table cellpadding="0" cellspacing="3" border="1">
<%
    if(null != headers){
%>

```

```

<tr>
<%
        Iterator colIter = headers.iterator();
        while(colIter.hasNext()){
            %>
            <th><%= colIter.next().toString() %></th>
            <%
        }
    %>
</tr>
<%
    }
    %>
<%
    if(null != rows){
        Iterator iter = rows.iterator();
        while(iter.hasNext()){
            ArrayList row = (ArrayList)iter.next();
            %>
            <tr>
            <%
                Iterator colIter = row.iterator();
                while(colIter.hasNext()){
                    %>
                    <td><%= colIter.next().toString() %></td>
                    <%
                }
            %>
            </tr>
            <%
        }
    }
    %>
</table>
<%
}
%>
<%
StringBuffer errors = (StringBuffer)request.getAttribute("ERRORS");
if(null != errors){
    %><h3>Errors:</h3><%=errors%><%
}
%>
</div>

```

All we did was add this to the bottom of the original JSP. We think it's exactly what one would expect given the data structure we created in the Java code. We simply build a table of the results, and then print out any errors that might have occurred.

Updating the portlet

Updating the portlet is very similar to installing it, and in the case of the command line XML interface it's exactly the same. Updating makes a portlet deployed on a page unavailable to users while it is being updated, and invalidate the user's session for that

portlet. So, if you're doing this on a production system, you must be aware that your users will suffer a small inconvenience.

Using the GUI

Updating the portlet in the GUI is similar to installing the portlet in the GUI.

1. Log onto your portal as an administrator.
2. Click the **Administration** link.
3. Click **Portlet Management => Web Modules**.
4. Search for your installed WAR file.

Manage Web Modules

Search by: File name starts with Search: SQL Search

Web modules Click Install to install a Web module. Select a Web module to view its portlet applications and portlets. Click Delete to remove the Web module from your portal or click Assign Access to allow others to work with the Web module.

Install Consume

Name	Status
SQLQuerySample.war	
sql.war	

Page 1 of 1

5. Once you've found your portlet, click the update button ().
6. Use the **Browse** button to find your WAR file.
7. Click **Next**. The file is transferred to the server and the portlet.xml file parsed. When that's done you will see what's in the WAR.
8. Click **Finish**.

Manage Web Modules

Updating a Web module, Step 2: View WAR file contents.

The selected WAR file contents are displayed below. Select the Finish button to update the SQLQuerySample.war Web module with the unique identifier 1_0_1UH or the Cancel button to go back to the Web module page.

Portlet applications	Portlets
com.ibm.portlets.sample.SQLQuery.778f228960	SQL Query Sample portlet

Finish Cancel

9. You should see a success page.

Manage Web Modules



EJPAQ1333I: Web module SQLQuerySample.war was successfully updated.

10. Then, you can go play with your new SQL query portlet! Go ahead and try a couple of selects. For example, you could try something like this:

```
select * from cats
```

or

```
select * from digits
```

or

```
select id, value from digits where catsid=5
```

The following figure shows results displayed when you enter `select * from cats`.

The screenshot shows a web portlet titled "SQL Query Sample portlet". It features a section labeled "SQL Query" with a text input field containing the query "select * from cats" and a "Submit" button. Below the input is a table with the following data:

	ID	TIMESTAMP	TYPE	HANDLE
1	1	2005-10-07 11:39:24.919501	1	Cool Cat
2	2	2005-10-07 11:39:24.973312	1	Breeze Cat
3	3	2005-10-07 11:39:25.01446	1	Bad Cat
4	4	2005-10-07 11:39:25.05205	1	Cute Cat
5	5	2005-10-07 11:39:25.090046	1	Fluffy Cat
6	6	2005-10-07 11:39:25.126878	1	Wet Cat
7	7	2005-10-07 11:39:25.166498	1	Mouse Eater Cat
8	8	2005-10-07 11:39:25.203727	1	Scared E Cat
9	9	2005-10-07 11:39:25.240685	1	Silly Cat
10	10	2005-10-07 11:39:25.276485	1	Cat Man Do

Using xmlAccess

Updating the portlet with xmlAccess is a breeze. You simply need to rerun the xmlAccess install script that we described in [Installing the portlet, Using xmlAccess](#).

Working with databases

This section provides a quick overview of working with databases, in case this is a new experience for you. This discussion is about IBM DB2; your experience may vary if you're using a different database system. If in doubt, consult with your database administrator about your specific installation.

Creating a sample database

The first thing you need to do is to create the database. You must know the database instance user, or at least a user that has privileges to create databases. If DB2 has been set up in the default manner, then on Linux the instance user is *db2inst1*. Log on as that user or *su - db2inst1* to it. If you're on Windows, get to a command window.

Start=> All Programs=> IBM DB2=> Command Line Tools=> Command Window

No matter what your platform happens to be, the form of the database creation statement is the same:

```
db2 create database mysample
```

You should see a success message:

```
DB20000I  The CREATE DATABASE command completed successfully.
```

If you don't, well, you might not have the necessary privileges to create a database, or your DB2 environment might not be properly initialized.

Connecting to the database

The next step is to connect to the database so that you can create some tables.

```
db2 connect to mysample
```

You should see confirmation that you are connected to the database. For example:

```
Database Connection Information

Database server          = DB2/LINUX 8.2.1
SQL authorization ID    = DB2INST1
Local database alias    = MYSAMPLE
```

Finally, in the [download](#) is a file, *MySampleTable.sql*, which you can use to create the mysample database and insert a bit of data. You run this file using this command:

```
db2 +o -tvf MySampleTable.sql -z OutFile.txt
```

Two tables are created: one named *cats* and one named *digits*. At the end of `MySampleTable.sql`, a couple of `Select` statements verify that the tables were created and that the data was inserted into them properly. If you take a look at `OutFile.txt` you should see output from those selects at the end.

You could also run these `Select` statements from the command line. Go ahead and try them:

```
$ db2 "select * from cats"
```

or

```
$ db2 "select * from digits"
```

If this doesn't work, you have to go back and debug what went wrong. When everything looks good, reset the database connection.

```
$ db2 connect reset
```

Removing database access

Before we move on, *when you are finished* playing with this tutorial, portlet, and database you might want to clean up after yourself. To do that you need to issue:

```
$ db2 drop database mysample
```

You have to make sure there are not outstanding connections to it and such so if you've created a `datasource` to the database, you should remove the portlet, delete the `datasource`, and the drop the database.

Now that you have the basics, you can move on to making the database available to your code as a `datasource`.

Making a data source for the sample database

There are a couple of different ways to create a data source for your new database. We show you the graphical way through the Web interface.

Accessing the Admin server

Before you can look at the pretty Web interface, you need to start up the admin server, *server1*.

1. Log in to your WebSphere Application Server machine.
2. Get to the application server `bin` directory. On our Linux machine it is:
`/opt/IBM/WebSphere/AppServer/bin`

3. Make sure you log in as someone who can start and stop application servers. In the case of Linux, by default the *root* user is who you need to be.

4. In the `bin` directory, run the command to start the server:

```
./startServer.sh server1
```

5. You should see a line eventually that tells you that the server is open for e-business.

```
ADMU3000I: Server server1 open for e-business; process id is 27279
```

6. Now, you can open a Web browser to the admin server Web interface:

```
http://<yourServer>:9090/admin
```

7. Log in through the login screen.

Setting up a data source

Now you can begin setting up a data source.

1. Create the J2C Authentication. Look at the navigation bar on the left and traverse the following:

Security => JAAS Configuration => J2C Authentication Data

2. Click **New**.

3. Put in the Alias information including the Alias, User ID, and password.

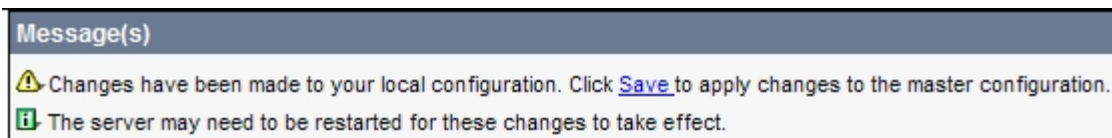
[J2C Authentication Data Entries >](#)

New

Specifies a list of userid and password for use by Java 2 Conn

Configuration	
General Properties	
Alias	* SampleAlias
User ID	* db2inst1
Password	* *****
Description	
Apply OK Reset Cancel	

4. Click **OK**.
5. Click the **Save** link at the top.



6. Click the **Save** button.

Creating the JDBC provider

The next step is to create the JDBC provider.

1. Create the MySample JDBC provider. Look at the navigation bar on the left and traverse the following:
Resources => JDBC Providers
2. Click the **New** button
3. In the JDBC Providers drop down, select **DB2 Universal JDBC Driver Provider**
4. Click the OK button
5. Change the name from DB2 Universal JDBC Driver Provider to MySample JDBC Driver Provider

6. Change the following path variables to the correct path if they are not set up in your environment:

```
{DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar  
{UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar  
{DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar
```

to this (you will want to verify the path for your environment)

```
/opt/IBM/db2/V8.1/java/db2jcc.jar  
{UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar  
{DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar
```

7. Click the **OK** button.

Creating the data source

Now, you can create the data source.

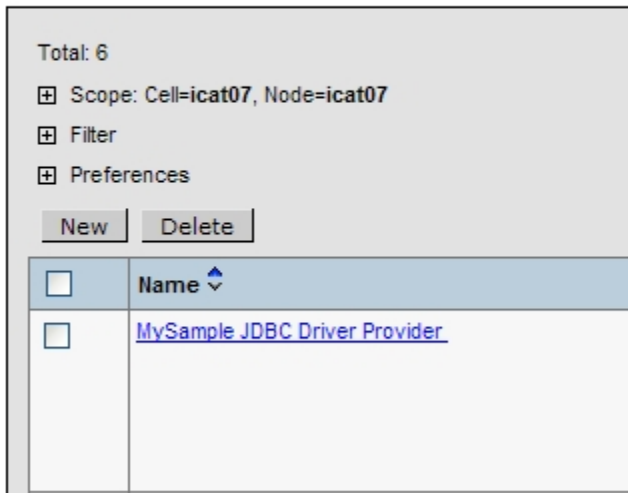
1. Click the **MySample JDBC Driver Provider** link in the list of JDBC providers.
2. Click the **Data Sources** link at the bottom of the provider information.
3. Click the **New** button.
4. Change the name from DB2 Universal JDBC Driver DataSource to MySample JDBC Driver DataSource.
5. Change JNDI Name to jdbc/MySampleDS.
6. Select the <servername>/SampleAlias from the **Component-managed Authentication Alias** dropdown box.
7. Click the **OK** button.

Set up the datasource.

1. Click the **MySample JDBC Driver DataSource** link

JDBC Providers

JDBC providers are used by the installed applications to access



2. Click the **Custom Properties** link at the bottom.
3. Click the **databaseName** link.
4. Set the databaseName to `mysample`.
5. Click the **OK** button.
6. Click the **serverName** link.
7. Set the serverName to the database servername.
8. Click the **OK** button.
9. Click the **portNumber** link.
10. Set the portNumber to `50001`.
To find the port number for your machine, run this command on the database server:
`cat /etc/services |grep db2c_db2inst1`
11. Click the **OK** button.
12. Click the **Save** link at the top.
13. Click the **Save** button.

Finally, test the connection.

1. Look at the navigation bar on the left and traverse the following:
**Resources => JDBC Providers => MySample JDBC Driver Provider
=> Data Sources**
2. Click the checkbox next to **MySample JDBC Driver DataSource**.
3. Click the **Test Connection** button. If you did everything correctly you see a message which indicates a successful test of the connection.



If not, go back and double-check all your work.

With all this work done you will probably need to restart your portal server to read in the new data source information. After that, you should be able to get a data source for your new sample database in the portlet code.

Conclusion

This brings us to the end of this part of our journey together. Let's reflect a moment on where we've been. We started at a fork in the road where we chose a development environment. Then, we developed a simple portlet that accesses a database backend to let users run SQL Select statements. We followed that up with the deployment of that portlet into a portal using a couple of different techniques, and that brings us to where we are now.

In the second part of this series, we continue the journey. You learn how to deploy the portlet we developed here using the Web Services for Remote Portlets (WSRP) protocol. Keep an eye out for it here on the developerWorks Portal Zone! Until then, remember these words from Yogi Berra, "When you arrive at a fork in the road, take it."

Resources

developerWorks WebSphere Portal zone

<http://www.ibm.com/developerworks/websphere/zones/portal/>

WebSphere Portal product documentation

<http://www.ibm.com/developerworks/websphere/zones/portal/proddoc.html>

Rational Application Developer V6 trial software

http://www.ibm.com/developerworks/downloads/r/rad/?S_TACT=105AGX10&S_CMP=ART

Rational Application Developer for WebSphere Software technical resources

<http://www.ibm.com/developerworks/rational/products/rad/>

Download

To get the download file, see the cover page for this tutorial, at:

https://www6.software.ibm.com/developerworks/education/websphere/0510_lynn/0510_lynn.html

The download includes:

SQLQuerySample.war	Source and deployable WAR file packaging of the Portlet.
SQLQuerySample.zip	Zip file packaging of the portlet source code.
MySampleTable.sql	DDL for the creation and population of a simple database.

About the authors



Karl Bishop is a Senior Software Engineer on the IBM® Web Enablement and Support team. He works from the wilds of North Carolina. Karl work on various internal and external portal based application, as well as being a strong proponent of Linux based solutions. You can reach Karl at kfbishop@us.ibm.com.



Ron Lynn is a Senior Software Engineer on the IBM® Web Enablement and support team. He works from a small farm in the San Joaquin Valley of central California. Ron is currently working on internal portal projects that focus on giving more visibility to IBM's partner applications. He has written and spoken on portlet development and other topics numerous times and is co-author of Programming Portlets. You can reach Ron at tcatt@us.ibm.com.