

Build error-free applications fast

Manage beans and error validation

Skill Level: Intermediate

[Jeff K. Wilson \(wilsonje@us.ibm.com\)](mailto:wilsonje@us.ibm.com)
E-business Architect

24 Sep 2004

The latest release of IBM WebSphere Studio and the Portal Toolkit plug-in provide new features for developing front-end applications using JavaServer Faces. These features enable developers to quickly and easily use visual rapid-application development tools and provide a rich set of interesting user interface components not easily created or maintained in the past. This tutorial demonstrates how a framework as flexible as JavaServer Faces and the tools provided by both WebSphere Studio and the Portal Toolkit make short order of integrating, testing and maintaining a portal-based front end.

Section 1. Before you start

About this tutorial

The latest release of IBM® WebSphere® Studio and the Portal Toolkit plug-in provide new features for developing front-end applications using JavaServer Faces (JSF). These new features enable developers to quickly and easily use visual rapid-application development tools and provide a rich set of interesting user interface components not easily created or maintained in the past.

In addition to a toolbox of user interface (UI) components, JSF developers have at their disposal a framework for both managing and validating data in a very stable and structured way. Because the open JSF standard defines this process, you can

build IDE tools that make the initial development a much simpler task. This is where the attention is focused in this tutorial -- taking advantage of the frameworks capabilities for holding and checking data and exploring the toolset to see how easy it is to implement these in your portal-based applications.

This tutorial provides a basic, but complete, understanding of both the framework in general and the process for developing complex applications that tie into back-end resources from within the WebSphere Portal environment.

Should I take this tutorial?

This tutorial is intended for WebSphere Portal developers who are interested in the latest JavaServer Faces tooling in WebSphere Studio.

Generally, developers should have a good understanding of Java development to get through this tutorial, though most of the steps use wizards and visual components. The few locations that require coding provide detailed explanations on what to type and what the code does. For the most part, how the application works behind the scenes is not as important as what the visual components do and how to use them within a portal-specific JSP.

Prerequisites

To complete the steps in this tutorial, you need the following software installed on your computer:

- IBM WebSphere Studio V5.1.2
- [IBM Portal Toolkit V5.0.2.2](#)
- [IBM DB2® Universal Database V8.1, or higher](#)
- [FacesPortletFiles.zip](#) contains sample code and JavaBeans.

Download the Portal Toolkit V5.0.2.2. Install WebSphere Studio, accepting all defaults, and run the Portal Toolkit installer, installing both the toolkit and the test environment.

Install DB2 accepting the defaults and install the Sample database from the First Steps window which should load on install. If the First Steps window does not load, select it from **Start Menu > Programs > IBM DB2**.

The FacesPortletFiles.zip file holds the JavaBeans you will use, as well as the code, in a file called CutandPasteCode.txt. It contains the code some of the steps have you to write. You can cut and paste snippets from here. Unzip FacesPortletFiles into

c:\temp -- it will create a directory called FacesPortletFiles in c:\temp.

Application overview

The application utilized in this tutorial accesses and updates employee data stored in a back-end repository. In a real-world development scenario, JavaServer Faces should provide a separate front-end layer that accesses some core components that, if designed well, do not rely on any specific style of front-end interface.

Typically, these core components would be JavaBeans, EJBs, or other components that hold all the business logic. These would likely be provided by other developers on your team who are skilled in Java technology but not necessarily in UI design. The components should also be flexible enough to be used, perhaps, by any type of front-end, whether it is a Swing-based rich client, Web service, or -- as you will do today -- a Web front end.

The application in this tutorial builds upon a couple of JavaBeans that are generic enough to be used in such applications and function basically as an interface to data stored in an IBM DB2 database.

The application loads those JavaBeans and layer rich components on JSPs that access these JavaBeans gaining access, in turn, to the data behind the scenes.

By the end, you will see how a framework as flexible as JavaServer Faces and the tools provided by both WebSphere Studio and the Portal Toolkit make short order of integrating and maintaining a portal-based front end.

Section 2. Set up the environment

Overview

Before you can get started building your application, you need to set up a few things. First, you need to build your database. The following steps detail how to create a new database and load the user information you will pull into your application.

Also, you will create a new portal project in WebSphere Studio with the Portlet Toolkit wizards and finally set up and configure the server that you need to run the application in.

Setting up DB2

The first thing to set up is your database. From here, you will pull out employee data to display in your portlet.

1. Open a command prompt.
2. Enter `db2cmd` and press **Enter**.
3. In the DB2 CLP command window, enter `db2 create db acme`. After a short time, the system displays a message to let you know the database was successfully created.
4. Navigate to the script that loads your data by entering `cd c:\temp\FacesPortletFiles` and press **Enter**.
5. Connect to your new database by entering `db2 connect to acme` and press **Enter**.
6. To load the data into your new database, enter `db2 --tvf acme.ddl` and press **Enter**.

If everything was successful, the output of the various databases configuration calls will reflect so.

Setting up WebSphere Studio

The next step is to set up your project in WebSphere Studio where you will carry out your development tasks.

1. Open WebSphere Studio from the Start menu.
2. When prompted to choose a location for the workspace (the location on the file system where source files are kept), enter `C:\temp\FacesPortletTutorial` and click **OK**.
3. When WebSphere Studio finally loads, select from the main menu bar, **File > New > Other**.
4. On the New window, select **Portlet Development** in the left pane, **Portlet Project** on the right, and click **Next**.

5. In the Project name field of the New Portlet Project window, enter `EmployeePortlet`.
6. Select **Faces portlet** and click **Finish**.
7. If prompted to switch perspectives, click **Yes**.

Setting up the server environment

Eventually, you need to test your application in the Test Environment bundled in Studio. This server environment needs to hold your data source, user information of a user that can access the database, and finally, be associated with your new project.

1. From the main menu bar, select **File > New > Other**.
2. On the New window, select **Server** in the left pane, **Server and Server Configuration** in the right pane, and click **Next**.
3. Enter `Portal Test Environment` in the Server name field of the Create a New Server and Server Configuration window.
4. For the Server type, select **WebSphere Portal version 5.0** and then select **Test Environment**. Click **Finish**.

Defining a user to access the database

A data source is used to define connection information that an application uses to access databases. If you are not familiar with data sources but you use databases, you likely connect somehow in your code.

Abstracting connection information into a reusable and easily configurable spot makes managing your application's data connections much easier -- particularly if you need to reuse the same connection multiple times or if you need to edit the information once the application coding process is complete.

If you are developing software for resale, this is essential because you likely do not want to have predefined user names and passwords hard-coded in your binaries.

Data sources are defined on your server and are called from your code by the label associated with it. The user names, passwords, tables to access, and so on are defined outside of your application and easily updated without touching any code.

A data source must be associated with a user with the appropriate credentials to access the database. Before you can set up your data source, add the user that you will use with your data source. The user and the data source are defined in the server configuration. If your application was deployed to a live application server, you need to configure the setting on that server similarly.

1. From the main menu bar, select **Window > Open Perspective > Other**.
2. On the Select Perspective window, select **Server** and click **OK**.
3. In the Server Configuration view of the Server Perspective, double-click **Portal Test Environment** to open the configuration editor.
4. In the main Server Configuration editor, select the Security tab. This is where you add the user information necessary to access the database.
5. Click **Add** next to the JAAS Authentication Entries list.
6. In the Add JAAS Authentication Entry window, enter a user name and password that is already defined on your system (for this tutorial, use db2admin/password) and click **OK**.
Note: Using db2admin to access your database is only suggested if you honestly believe a hacker would never imagine any self-respecting db administrator would be so careless to use the default user. Just a tip.

Setting up the data source

Now that you have defined a user to access the database, create the data source to access the data used in your application.

1. Click the Data source tab to define your data source.
2. Click **Add** next to the JDBC provider list (this defines the driver to use for your data source).
3. On the Create a JDBC Provider window, select **IBM DB2** (clearly) from the Database type list.
4. Select **DB2 Legacy CLI-based Type 2 JDBC Driver** and click **Next**.
5. On the following page, enter the name `DB2 Driver`.

In some cases the class path variable might not be set up properly. Instead of editing the actual class path variable that might affect other applications, add a hard path to the driver libraries for the purposes of this tutorial.

6. Click **Add External JARS**.
7. Locate the db2java.zip library (likely in C:\Program Files\SQLLIB\java).
8. Finally, click **Finish**.
9. Now, back in the main Server Configuration editor, click **Add** next to the Data source defined... list.
10. On the Create a Data Source window, select **DB2 Legacy CLI-based Type 2 JDBC Driver** from the JDBC provider list, leave the default Version 5.0 radio button selected, and click **Next**.
11. On the Modify Data Source window, change the JNDI name to jdbc/Employees.
12. From the Component-managed authentication alias list, select the db2 user you added on the Security tab and click **Next**.
13. On the Create a Data Source page, select **databaseName** from the Resource Properties list and change the Value field from sample to ACMECO and click **Finish**.
14. Save (CTRL-S) and close the server configuration editor by clicking the x in the tab of the configuration editor.

Summary

At this point, you have the basic project ready to be built upon. In the upcoming sections, you will add components to your project and run it in the embedded server environment within WebSphere Studio. Because of the work you did setting up the data source, connecting to the database will be a snap.

Section 3. Build the project

Overview

Now that you have your basic project set up, you can start building your application. Part of this application was built by your imaginary core application developers. As JSF-based portlet developers, your task is to build a front end to the base application, which can include JavaBeans, EJBs, and other resources.

The theory is that those who have certain skill develop certain types of components. You might wear several hats in your team but, even still, abstracting out certain types of tasks into their own appropriate types of components is generally considered good form.

If you built your business logic -- your core application methods -- into the JSF or portlet specific components, you might have quite a task reworking the code should you decide to move your application to a Swing-based type of application.

So, in the spirit of a true, well-designed, J2EE design methodology, you import your core application packaged into a couple JavaBeans. In practice, you might have imported dozens of JavaBeans, JSPs, EJBs, and who knows what else.

Once you pulled in your core application, you can start manipulating how your application flow functions. It is in these components that you will add your JavaServer Faces logic.

Importing the core application

First things first: bring in the existing application components to use in your JSF portlet.

1. On the left side of WebSphere Studio, click the Portlet Perspective icon to switch to that perspective.
2. In the Project Navigator view of the Portlet Perspective, expand **EmployeePortlet > Java Resources**.
3. Right-click **pagecode** and select **Import** from the context menu.
4. On the Import window, select **File system** and click **Next**.
5. On the File system page of the Import window, click **Browse**, navigate to `C:\temp\FacesPortletFiles`, and click **OK**.
6. Check both **Employee.java** and **EmployeeCommandBean.java** in the

right panel of the window.

7. Make sure `EmployeePortlet/JavaSource/pagecode` is entered in the Into folder field and click **Finish**.

Note: `Employee.java` is a `JavaBean` that holds information about a given employee. When you log in, this bean is populated with your data and helps in the session.

`EmployeeCommandBean.java` is a class that manages the connections to the database. The two main methods on this bean retrieve the logged in user's employee data and runs an update call to the database should the employee decide to change any data.

In a more robust enterprise application, these methods might call `EJBs`, run scripts, send messages, or any of a number of other methods to manage the data. For the purposes of this tutorial, your `JSF` components will simply call these methods. As a front-end developer, you do not really care how the back-end folks decide to manage connections so long as the desired task gets done.

Adding front-end JSPs

Now you need a couple of more `JSPs` to manage the flow of your application. By simply creating the project, the first `JSP` was created for us. Use this as your login page. But now you need to add an error page for those who do not follow directions well and fail the login and, additionally, the desired successful login home page. On this page you will add a few interesting `JSF` widgets to experiment with what this technology has to offer.

1. In the Project Navigator view, expand **EmployeePortlet > WebContent**.
2. Right-click **WebContent** and select **New > Faces JSP File**.
3. In the File name field of the New Faces JSP File window, enter `error.jsp` and click **Finish**.
4. Right-click **WebContent** again and select **New > Faces JSF** file again to add another `JSP`.
5. In the File Name field, enter `home.jsp` and click **Finish**. Your two new files are displayed in the Project Navigator view.

Adding Managed Beans to the project

JSF has a concept of a Managed Bean to make it easy for JSP developers to access methods on JavaBeans. If you are comfortable with the notion of JSP tags to access JavaBeans held in the session or request, you will likely see how much simpler it is to drag and drop objects stored in this special JSF way.

1. If `EmployeePortletView.jsp` is not open, double-click it in the Project Navigator view.
2. With the `EmployeePortletView.jsp` file selected in the main editor, locate the Page Data view.
3. In the Page Data view, right-click **JSP scripting** and select **New > JavaBean**.
4. Make sure the Add new JavaBean radio button is selected.
5. In the Name field of the Add JavaBean window, enter `employee`.
6. Click the button next to the Class field.
7. Start typing `Employee` in the Choose a class field (notice that a list of available classes are updated as you type).
8. Select **Employee** and click **OK**.
9. Back on the Add JavaBean window, check **Make this JavaBean reusable**.
10. For the Scope, select **session** and click **Finish**.
11. In the Page Data view, right-click **JSP scripting again** and choose **New > JavaBean**.
12. In the Add JavaBean window, enter `empCB` for the Name field.
13. Click the Class button.
14. In the Class Selection window start typing `EmployeeCommandBean`.
15. Select **EmployeeCommandBean** and click **OK**.
16. Back on the Add JavaBean window, check **Make this JavaBean reusable**.

17. For the Scope, select **session** and click **Finish**.
18. Notice the two new JavaBeans listed in the Page Data view. The getter/setter methods are shown in this view. Any other method in either of these beans is accessible from your Java code as you will see, but the *fname*, *lname*, and so forth shown here represent getter/setter methods that can be directly dragged onto a JSP. When done so, the JSF framework manages calling the appropriate methods to either retrieve or set (in the case of an input field of a form) data back in the Employee class. Non-getter/setter methods are not displayed here, but can be seen in the outline view, through content assist, or just by viewing the code.

Creating the login process

First, build your login page.

1. Open EmployeePortletView.jsp.
2. In the main editor, edit the text to display **Please log in:** and press **Enter** twice to add a couple of line breaks.
3. In the Palette view, locate the Output component and drag and drop it onto your JSP below the text.
4. With the outputText component selected on the JSP, locate the Attributes view in the lower left corner of the WebSphere Studio.
5. Enter `username` in the Value field and press **Enter**.
6. From the Palette view, select the Input component and drag it onto your JSP next to the Username output text.
7. In the Attribute view, change the Id field of the input field to `username` and press **Enter**.
8. In the Page Data view, expand **JSP scripting**.
9. Right-click **requestScope** and select **Add Request Scope Variable** from the context menu.
10. In the Add a Request Scope Variable window, enter `username` into the Variable name field.

11. Enter `java.lang.String` into the Type field and click **OK**.
12. From the Page Data view, expand **requestScope** and drag *username* onto the input text box on the JSP.
This binds the request variable *username* to the value entered in the text box. When the text box changes (and is submitted), the request variable changes. Additionally, binding the *username* request variable to the text box automatically prepopulates the field with its value if you are returned to this page. You will see it in action in a moment.
13. Add a couple line breaks after the text box by pressing **Enter** a couple of times.
14. From the Palette view, drag **Command - Button** to the bottom of the JSP.
15. In the Attributes view, click the Format tab.
16. In the Label field, enter `Log In` and click **Enter**.

Handling navigation points

When the Log in button is clicked, processing of the form fires and eventually returns a navigation label. These navigation labels are associated with a given page. This is done so that if a form is submitted successfully, the processing might return a simple "success" -- and success is mapped to `pageA.jsp`, for example. If after some time the return page needs to be changed to `pageB.jsp`, the developer simply needs to edit `faces-config.xml`, the xml descriptor that lists the navigation points. No code needs to be changed.

A more useful example might be for a "home page" labeled home. All your pages might have a link that returns the label "home" that is mapped to `index.html`. After some time, the home page is changed to `index.jsp`. In this case, again only one file needs editing -- not every page of the site.

Because of the handy tools in WebSphere Studio, you do not need to edit the `faces-config.xml` file directly. You can simply use the UI in the navigation tab of the Attributes view. Much nicer than XML coding.

1. In the Attributes view, click the Navigation tab.
2. Click **Add** to add a navigation point.

3. On the Add Navigation Rule window, select **home.jsp** from the Page drop-down list.
4. For the Alias, enter `success`.
5. Leave This page only selected for Make this rule available option and click **OK**.
When you return success (as you will soon do) the JSF framework looks up "success" in the faces-config.xml file and sees that it really points to home.jsp.
6. On the Navigation tab of the Attributes view, click **Add**.
7. Select **error.jsp** from the Page drop-down list.
8. Set the Alias to **failure**.
Success and failure are typical standard practice labels used for form submissions, but there is no reason that you could not use any labels or as many labels as you wish. When processing the form, your various processing statements will determine which label to return.
9. Leave the radio button for This page only radio selected and click **OK**.
These links both are local links meaning that there may be "success" navigation labels set to different pages called from other forms. Let's add a couple of global links that any page can use.
10. In the Attributes view, click **Add** again.
11. Select **EmployeePortletView.jsp** from the Page drop-down list.
12. Enter `login` for the Alias.
13. Select the Globally radio button and click **OK**.
Now any page that returns a navigation label "login" will end up returning EmployeePortletView.jsp to the user.
14. Click **Add** one last time to add another navigation label.
15. On the Add Navigation Rule window, select **home.jsp** from the Page drop-down list.
16. Enter `home` for the Alias.

17. Select **Globally** and click **OK**.

Processing the form

The final thing to add to your login page is the processing of the form.

1. Right-click on the Log In button and select **Quick Edit**.
2. In the Quick Edit view at the bottom of WebSphere Studio, select **Command** in the left pane and then click anywhere in the right pane. When the Log In button is pressed, the command action will fire. Any processing of the form can be done in here. The return shown above is what indicates which navigation point is to be returned. You will add some processing to determine whether to return success or failure.

Your EmployeeCommandBean has a method that looks up a user from a database. Simply pass the username to this method. The method returns an employee -- the one you pass into it -- populated with data pulled from the database. If the employee ID is not blank, you know the user exists and you forward to "success". Otherwise, the user is bounced to the "failure" error page.

3. Replace all the code in the Quick Edit view with the following:

```
empCB = new EmployeeCommandBean();
employee = empCB.checkUser(getEmployee(), getUsername().getValue().toString());

if (employee.getEmpno().equals("")){
    return "failure";
} else {
    return "success";
}
```

Employee and empCB were defined and are accessible because you added the JavaBean to your Page Data view earlier -- you just need to initialize them to use them.

`checkUser()` is the method that looks up the username and returns true or false depending on if the user exists in the database.

`getEmployee()` is a method that will return the employee object you are storing in the session. You pass that into the `checkUser()` method and it will be populated with data from the database.

`getUsername().getValue().toString()` is the series of method calls to extract the value of the username text area you added to the JSP.

Both `getEmployee()` and `getUsername()` were created when you added components to your application.

4. Save (CTRL-S) and close `EmployeePortletView.jsp`.

Handling login errors

There are many ways you can handle errors with JSF. In your application, simply show an error page and provide the mechanism to log in again.

1. Open `error.jsp`.
2. In the Page Data view, expand **JSP scripting**.
3. Right-click **requestScope** and select **Add Request Scope Variable**.
4. On the Add Request Scope Variable window, enter `username` into the Variable name field.
5. Set the Type to `java.lang.String` and click **OK**.
Note: If a form was submitted and directed to this page and a request scope variable named `username` was set, this entry will hold this value. Recall the username text input field on the login page was bound to the request scope variable `username`.
6. In the main editor for your `error.jsp` file, change the text, "Place content here", to the following:

```
Sorry, the username was not found in the Employee
database.
```

Notice that there are two spaces between "Sorry, the username" and "was not found..." You are going to put a JSF component between those phrases.
7. Add two line breaks after the new text by pressing **Enter** twice.
8. From the Palette view, select the Output component and drag it onto the JSP between `Sorry, the username` and `was not found...`
9. From the Page Data view, expand **JSP scripting > requestScope** and drag and drop `username` onto the new `outputText` component on the JSP.

10. In the Attributes view, set the Properties field to `color:red` and press **Enter**.
11. To create a link back to the login page, from the Palette view, select **Command - Hyperlink**.
12. In the Quick Edit view, select **Command** in the left pane and click somewhere in the right pane.
13. In the right pane of the Quick Edit view, simply change the `return " ";` code to `return "login";`
Remember, you set the navigation label "login" to your login form on `EmployeePortletView.jsp`.
14. Click on the words "link label" in the JSP (note how the selection highlighting changes).
15. In the Attributes view, change Value field to `Try again` and press **Enter**.
16. Save (CTRL-S) and close `error.jsp`.

Managing successful login

The last thing to do is handle someone who successfully logs in.

1. Open `home.jsp`.
2. In the Page Data view, right-click **JSP scripting** and select **New > JavaBean**.
3. On the Add JavaBean window, select the Add existing reusable JavaBean radio button.
4. Hold the CTRL key and select both **employee** and **empCB** and click **Finish**.
5. Change the text `"Place content here".` to `"Welcome "` (include a trailing space because you will add a JSF component) and press **Enter** twice to add a couple of line breaks.
6. From the Palette view, drag and drop an Output component next to the word `Welcome` in the JSP.

7. In the Page Data view, expand **employee** and drag and drop **fname** onto the new outputText component.
8. From the Palette view, drag and drop a **Command - Hyperlink** next to the fname output text.
9. In the Quick Edit view, select **Command** from the left pane and click in the right pane.
10. Edit the code to return "login".
11. Click on the words **link label**.
12. In the Attributes view, set the Value field to **[switch user]**.

Creating a tabbed panel component

One of the big benefits of JSF is the multitude of UI components at your disposal. In the next couple of sections, you will add some of the more interesting ones. And, of course, with some typical style sheet and HTML design effort, you can make these look even snappier. But at least you'll see how to set it up.

1. From the Palette view, select the Panels -- Tabbed JSF component and drag and drop it on the JSP below the text.
2. On the Confirm resource copy window, click **OK**.
3. A two-tabbed set of panels is displayed in the main editor.
4. In the Attributes view, on the Basics tab, set the width to 500 pixels and press **Enter**.
5. Click on the Panel List tab.
6. Change Tab1 to `Everyone`.
7. Change Tab2 to `My Data`.
8. Click **Add** and change **NewTab** to **Update Me**.
9. Click on the My Data tab in the JSP.
10. From the Page Data view, select **employee** and drag and drop it onto the My Data panel.

11. On the Insert JavaBean window, change the labels to friendly, readable text and reorder the fields using the up and down arrows.
12. Leave the Display fields (read-only) radio button selected, the Control Types set to Output Field, and click **Finish**.
13. Click on the Update Me tab.
14. From the Page Data view, drag and drop the employee JavaBean again onto the Update Me panel.
15. This time, select the Updating fields radio button.
16. Uncheck **empno**.
17. Switch the Control type for salary to Output Field (sorry, wouldn't it be nice...).
18. Relabel the fields again and move the lname record up under *fname* and click **Finish**.
19. Select the Submit button.
20. In the Attributes view, click on the Format tab and enter `Update Me` in the Label field and press **Enter**.
21. Right-click the Update Me button and select **Quick Edit**.
22. In the Quick Edit view, select **Command** in the left panel and click inside the right panel.
23. Change the existing code to the following:

```
empCB = new EmployeeCommandBean();
empCB.updateUser(getEmployee());

return "home";
```

The `updateUser()` method on the command bean updates the database from information stored in an Employee object. Again, call the `getEmployee()` method to return the employee data you have been storing in the session. This particular form automatically updates the data in the bean first. All you need to do is put it into the predefined `updateUser()` method that persists the data down to the database. Returning "home" returns the user to `home.jsp` as defined in the page navigation from earlier. Pretty simple, eh?

Creating a record list component

Now let's add a record list element. This record list component makes pulling data from a database as easy as it could possibly be -- including navigation components to easily get through the data.

1. Click on the Everyone tab.
2. From the Palette view, select **Relational Record List** and drag and drop it onto your Everyone tab.
3. On the Add WDO runtime? window, click **Yes**.
4. On the Add Relational Record List window, enter in the Name field, `employeeList` and click **Next**.
5. On the Record List Properties page, click the New button next to the Connection name field.
6. On the Select A Database page, accept the defaults for a new connection and click **Next**.
7. On the Database Connection page, set the Database field to **ACMECO**.
8. Add the *userid* and *password* of a user that can access the database.
9. Set the Database vendor type to `DB2 Universal Database V8.1`.
10. Set the Host to `localhost` and click **Finish**.
11. On the Record List Properties pages, select the `ACMECO.EMPLOYEES` table and click **Next**.
12. On the Column Selection and Other Tasks page, click the None button to deselect all the fields.
13. Check **USERNAME**, **FIRSTNME**, **LASTNAME**, **WORKDEPT**, and **SALARY**.
14. Click **Order results** under Tasks on the left side of the window.
15. Select **LASTNAME** and click the > button.
16. Select **FIRSTNME** and click the > button again and click **Close**.

17. Click Add another database table through a relationship under Advanced tasks.
18. On the Create Relationship window, select **ACMECO.DEPARTMENT** and click **Next**.
19. In the Multiplicity drop-down, select **FOREIGN KEY > PRIMARY KEY**.
20. In the Key columns mappings, click **Choose a column**.
21. Select **WORKDEPT: String** (you are defining how the EMPLOYEE table and DEPARTMENT table are joined -- the WORKDEPT field holds the DEPTNO id of the DEPARTMENT table).
22. On the Create Relationship window, click **Finish**.
23. On the Add Relationship Record List window, click **Next**.
24. Accept the defaults and click **Finish**.
25. From the Page Data view, drag and drop **employeeList** onto your Employee panel.
26. On the Configure Data Controls page, click **None** to deselect all check boxes.
27. Check **FIRSTNAME, USERNAME, LASTNAME, SALARY**, and **EMPLOYEES_DEPARTMENT.DEPTNAME**.
28. Rename the checked fields with more friendly names.
29. Move LASTNAME up to the top and click **Finish**.
30. Click within the LASTNAME column.
31. In the Attributes view, click on **h:column** in the upper-right corner and select **h:dataTable** to switch the selection in the main editor.
32. On the Basics tab of the Attributes view, click **Add header**.
33. From the Palettes view, drag and drop the Output component onto the new header space labeled **box1: Drag and Drop...**
34. In the Attributes view, with **outputText** selected in the main editor, enter `All AcmeCo Employees` in the Value field and press **Enter**.
35. Next to the Properties field, click the style editor button.

36. On the Fonts page of the Set Style Properties window, set the Size field to 14.
37. For the Font family, select **Arial** from the right column and click **Add** to move it to the left column.
38. For the Color field, click the eye dropper button and select the color of the tab behind the window.
39. Click **Font styles** in the left column.
40. Set the Weight to **Bold** and click **OK**.
41. Finally, select the **{SALARY}** output in the table.
42. In the Attributes view, enter a \$ to the beginning of the Value field.
43. Set the Properties field to **font-weight:bold**.

Handling record paging

Finally, let's add a mechanism to scroll through the list of employees because there will likely be more than you want to see on one page.

1. In the Attributes view, click the component listed in the upper right corner and select **h:dataTable** again.
2. Click on the Paging tab.
3. Enter 5 in the Items/page field and press **Enter**.
4. Click the Deluxe Pager button.
5. Save (CTRL-S) and close home.jsp.

Summary

And that should do it... As you can see, building a JavaServer Faces front end to an application is a simple matter of dragging and dropping components and configuring them through wizards and GUIs. Very little code is necessary for developing a well-designed application.

Section 4. Run the application

Overview

Now that you have completed building your portlet, you can give it a test drive in the Portal Test Environment bundled in Studio. From here you can see if everything works -- making edits if necessary -- without the trouble of repackaging, deploying, and installing into a real live server.

You set up the environment earlier, so all you need to do is rebuild the project and run it. Rebuilding simply compiles some artifacts as well as validates the portal application as a whole. It might not be necessary for today's exercise, but it is generally good practice.

1. In the Project Navigator view, right-click **EmployeePortlet** and select **Rebuild**.

Running the test environment

To run the project in the Portal Test Environment, you simply need to launch it pointing it to the server you set up earlier. The tool will handle packaging and deploying for us. Let me forewarn you, it is not a speedy process.

1. In the Project Navigator view, right-click **EmployeePortlet** and select **Run on Server**.
2. On the Server selection window, select the Use an existing server radio button and click **Finish**.
3. Wait...
4. Wait...
5. Wait, some more...

Testing bad logins

Once the Portal Test Environment loads, you will see the portlet login form you created earlier.

1. In the Username field, enter `fred` (which is NOT in your database) and click **Log In**.
2. Hurray! You successfully failed the login...
3. Click **Try again**.

Testing good logins

Now, let's use a login that IS in your database.

1. From the login screen, enter `jamesj` and click **Log In**.
2. Click the paging buttons to test the employee list record list.

Editing managed data

Finally, let's test accessing and editing the data stored in your managed JavaBeans.

1. Click the My Data tab to test pulling and loading your data (or James Jefferson's, that is...)
Note: You can also navigate the tabs using the Back and Next buttons at the bottom of the panels.
2. Lastly, test the update capabilities -- click the Update Me tab.
3. Change the *username* to `wonderboy` and click **Update Me**.
To see if the database was updated, locate the record for JAMES JEFFERSON in the All AcmeCo Employees list -- this is data pulled straight from the database.
4. Yippie!

Summary

Clearly one can easily see the benefits of having an embedded server for testing, debugging, and profiling. In your example, you completed the application before running the testing. In a larger development scenario, developers will likely test components right along the way.

Not only does this provide a mechanism for testing simply IF an application works as designed, but also works well as a design step to determine how the end result looks when run. If one had to repackage, deploy, install, etc., every time one wanted to try a different style font, this might hinder development speeds.

Section 5. Wrap up

Summary

There are many areas that you can take your application beyond what you experienced in today's exercise. What you did accomplish, however, was to see the basic structure of the JavaServer Faces solution included in WebSphere Studio.

By approaching application development with JSF as a truly front-end slice of your full application, your UI developers have an easy time generating a rich and useful user experience while relieving your back-end developers from concerns of how a user interacts with the core components.

And finally, of course, adding an embedded testing environment makes debugging, experimenting, and testing a much smoother process for all involved.

Resources

Learn

- [Developing Web applications using RAD tools, IBM extended JSF components, and WebSphere Studio V.5.1.2](#)
- [Struts-based portal applications: Model and develop them with WebSphere Studio](#)
- [Developing JSF Applications using WebSphere Studio V5.1.1 -- Part 1](#)
- [Developing JavaServer Faces portlets using WebSphere Studio and the IBM Portal Toolkit -- Part 2](#)
- Stay current with [developerWorks technical events and Webcasts](#).

Get products and technologies

- [IBM Portal Toolkit V5.0.2.2](#)
- [IBM DB2 Universal Database V8.1, or higher](#)
- [FacesPortletFiles.zip](#)
- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.

Discuss

- [Participate in the discussion forum for this content.](#)

About the author

Jeff K. Wilson

Jeff Wilson, a self-proclaimed dot-com refugee, has for the past three years been an e-business architect for the DragonSlayers, IBM's developer relations technical consulting team in Austin, Texas. It is their goal to excite, evangelize, educate, and enable developers on the latest tools and technologies available. Jeff welcomes any and all questions, comments, recipes, insider stock tips, cash, prizes, and any good juicy gossip. He can be reached at wilsonje@us.ibm.com.