

# Struts-based portal applications

## Model and develop them with WebSphere Studio

Skill Level: Introductory

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

09 Jul 2004

Struts is a very popular framework that adds a flexible control layer to building Web based applications using common standard technologies like servlets, JavaBeans components, resource bundles, and custom tag libraries. This tutorial provides a hands-on approach to developing Struts based portal applications using WebSphere Studio V5.1.2 and the Portal Toolkit V5.0.2.2.

## Section 1. Before you begin

### Overview

This tutorial provides a hands-on approach to developing Struts-based portal applications using WebSphere® Studio Application Developer V5.1.2 and the Portal Toolkit V5.0.2.2. These tools are available for both WebSphere Studio Site Developer, which comes with WebSphere Portal, and Application Developer. The Portal Toolkit functions the same way in both tools.

The popular Struts framework adds a flexible control layer for building Web-based applications using common standard technologies like servlets, JavaBeans components, resource bundles, and custom tag libraries. Primarily, Struts focuses on modeling navigation throughout an application; however, it also provides many other benefits, particularly with form validation and resource handling. In addition,

with Struts it's easy to implement custom tags that help you control how elements on JSPs function -- for instance, form input fields can manage values and submissions.

Struts functions by using a controller mechanism that manages all requests, comparing them to a deployment descriptor that maps necessary components. This conflicts with a similar WebSphere Portal concept, where the portal engine manages the control of incoming requests. As a workaround, WebSphere Portal provides a Struts Portal Framework bridge that manages the flow of requests between the portal environment and the Struts framework (among other things).

In the past, implementing this Struts Portal Framework in a portal application was mainly a manual and sometimes kludgy process that required you to always begin with an existing Struts-enabled portal application (never mind the philosophical "portlet or the egg" question). Thankfully, now the Portal Toolkit V5.0.2.2 provides some slick and rather complete tools to aid such projects.

## Who should take this tutorial?

This tutorial is aimed at portal developers who want to implement Struts into their portal or, I suppose, Struts developers who want to develop portlets.

This tutorial simply describes the process of implementing Struts in your portal; you really don't need to be an expert with either Struts or portlet development in order to take it. However, if you pay attention, you might learn a little about both in the process.

In short, it's for any Web developer who cares, and maybe a few that don't.

## Prerequisites

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

- [WebSphere Studio V5.1.2](#)
- [Portal Toolkit V5.0.2.2](#)
- [CutAndPasteCode.txt](#)

The CutAndPasteCode.txt file contains code snippets used in some of the steps throughout the tutorial. You can use this file to easily cut and paste code from (unless you prefer to test your spelling skills by typing it all by hand).

Download both WebSphere Studio (Site or Application) Developer V5.1.2 and 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.

---

## Section 2. Model the application with the Web diagram

### Overview

The Web diagram allows you to model your Struts application before generating any code. This makes it really easy to describe the relationships and the flow within the portal prior to coding. Additionally, the visual development process is a big benefit.

As you will see, once you define the relationship among the portal elements, the wizards that generate the code can make certain assumptions based on what you described in the diagram and do more of the coding for you.

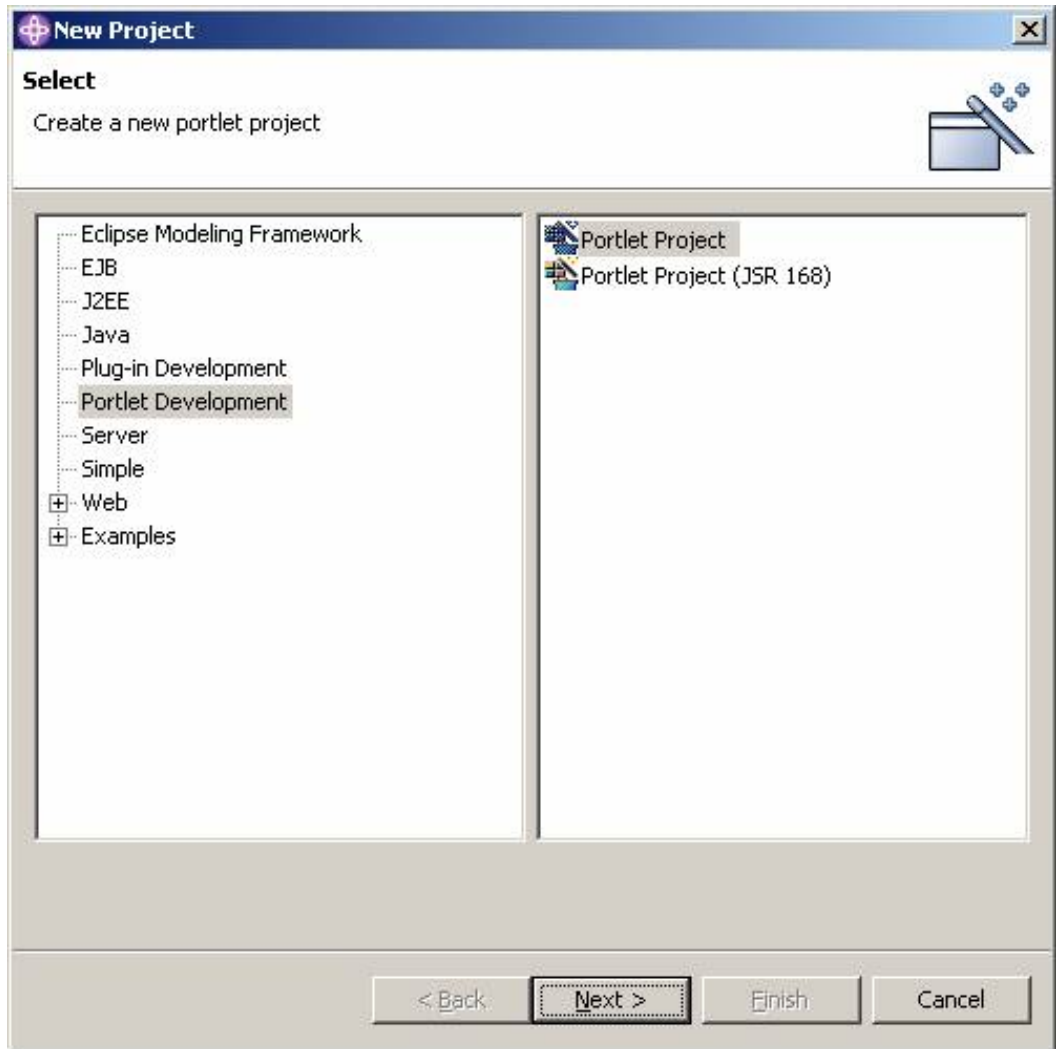
### Creating the portal application

Before you can do anything else, you need to create the portal application. Begin by opening WebSphere Studio and creating a simple Struts-based portal application using the included wizards. This will most importantly create the project structure, complete with the Struts APIs, custom JSP tag libraries, and everything you need to get started.

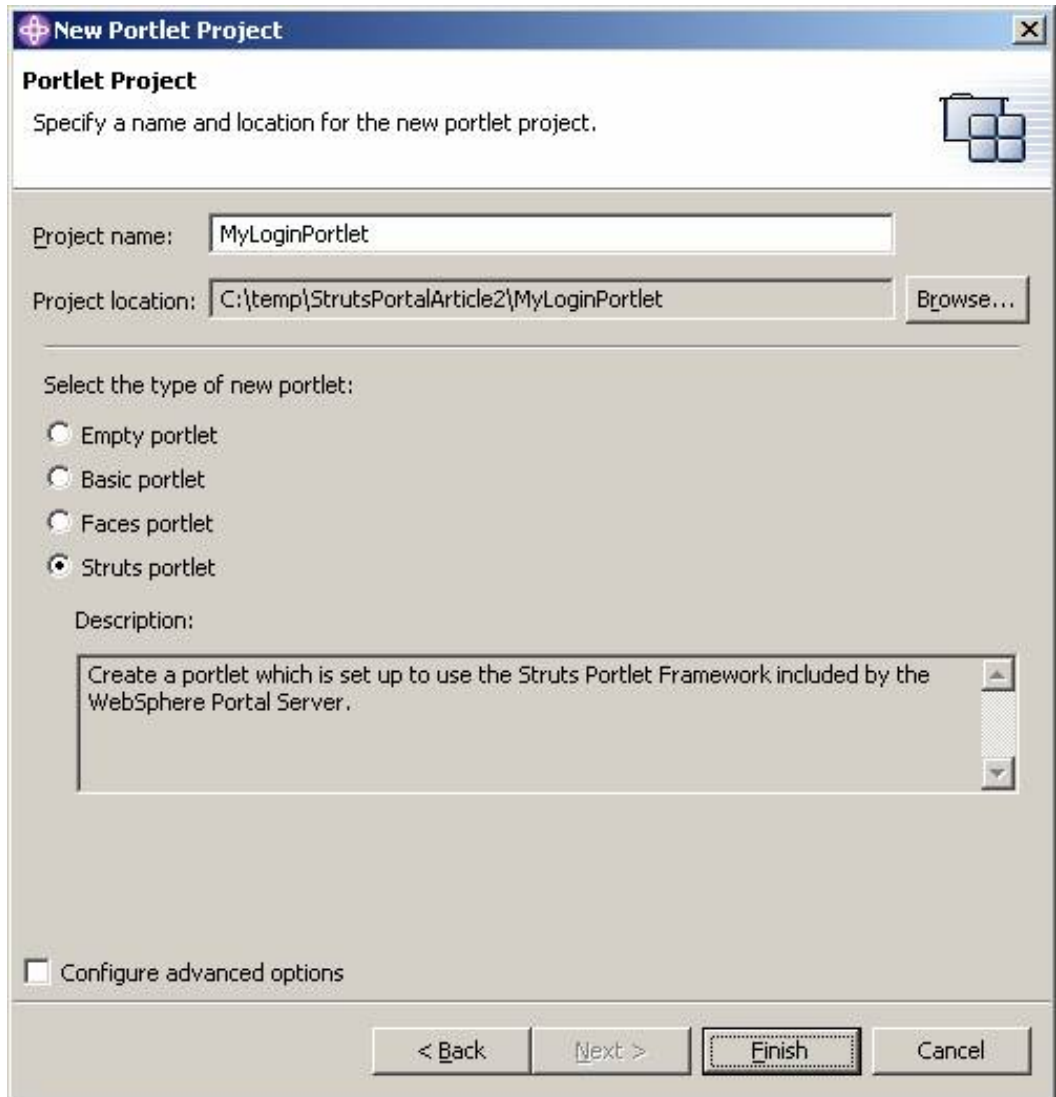
1. Open WebSphere Studio by choosing **Start > Programs > IBM WebSphere Studio > [Site or Application] Developer 5.1.2**.
2. Click **File > New > Project**.



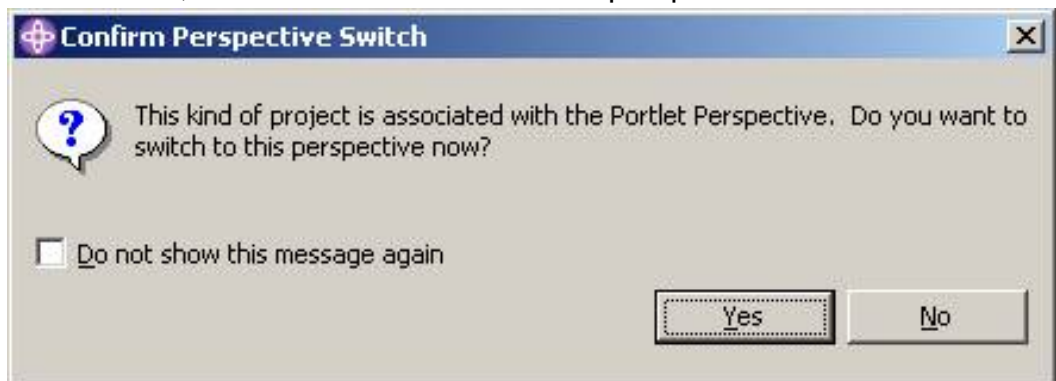
3. On the New Project window, select **Portlet Development** on the left and **Portlet Project** on the right.



4. Click **Next**.
5. Name your project `MyLoginPortlet`, click the Struts Portlet radio button, and click **Next**.



6. Click **Finish**, then click **Yes** to confirm the perspective switch.



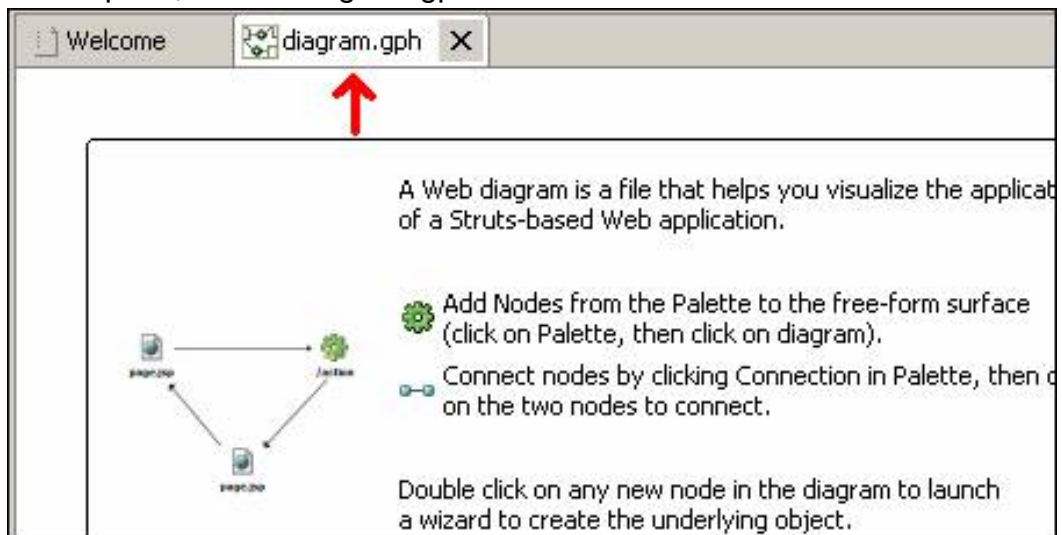
7. Click the X to close the portlet deployment descriptor.



## Adding Web pages to the diagram

Clearly, you need some Web pages on which to implement your Struts framework. Using the Web Diagram tool, you can model your application and connect the components, defining the flow from which you can generate your code. Model-driven development can be a very quick way to design applications without being bogged down with actual coding.

1. At this point, the file diagram.gph should be loaded in the main editor.



2. If the Palette view is not open in your IDE, select **Window > Show View > Palette** from the main menu.
3. From the Palette view on the left, click to expand **Struts Tools**.
4. Click the Web Page element.



5. Click somewhere within the diagram to drop the Web Page element.
6. Label that element `/index.jsp` (the `/` will disappear from the icon).



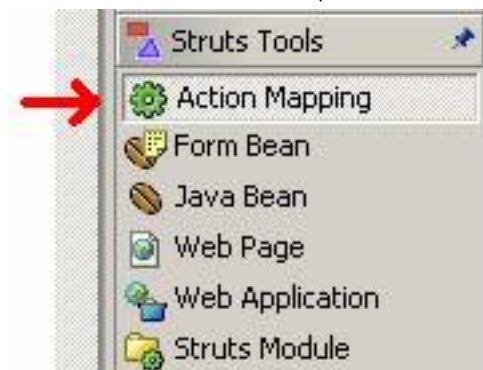
7. Repeat these steps to add two more Web Page elements, named `/success.jsp` and `/failure.jsp`, to your diagram.



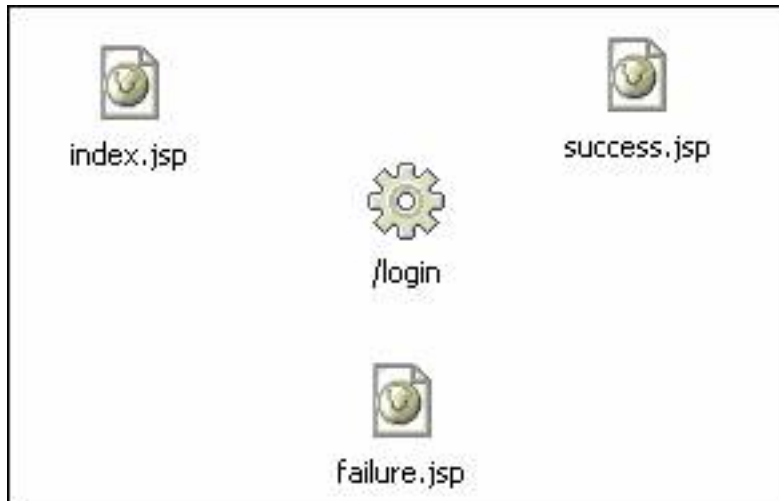
## Adding action mapping to the diagram

Defining action maps simply defines how requests are processed by the server. Again, using the Web Diagram tool to visually define the flow can be a very simple way to see how the user will interact with the application.

1. From the Palette view, select **Action Mapping**.



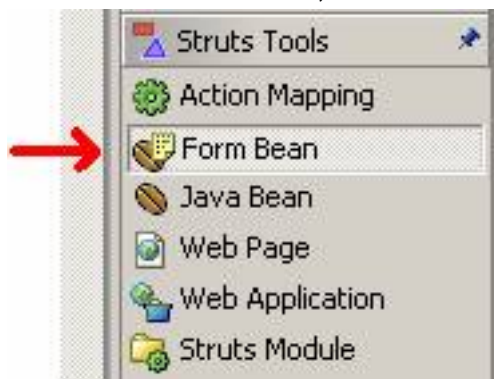
2. Click in the middle of the diagram to drop that element into the flow, then name it /login.



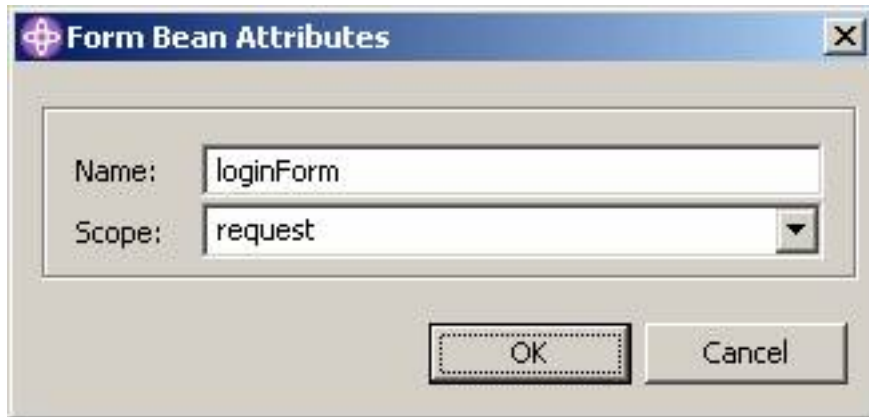
## Adding a form bean to the diagram

Form beans act as validation components in a Struts application. They contain properties that map to input parameters on HTML forms that can be checked against rules set in the bean. The Web Diagram tool builds the structure of the bean for you, and all you need to add are the rules that are automatically called when a form is processed. If a rule is broken -- that is, if input is not entered correctly -- you can kick the request out, throwing an error. This typically means pushing the user back to the form and displaying the error so that the parameter can be entered correctly.

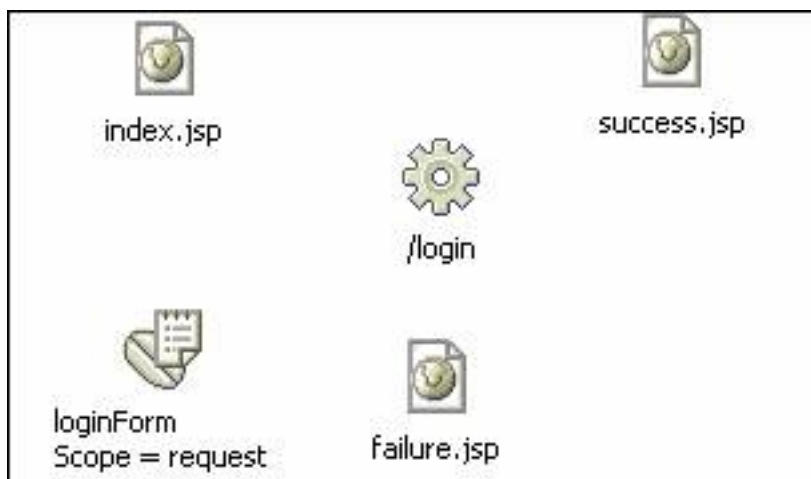
1. From the Palette view, select **Form Bean**.



2. Click in the diagram to drop the element.
3. On the Form Bean Attributes window, name the bean `loginForm` and select **request** as its scope.



4. Click **OK**.



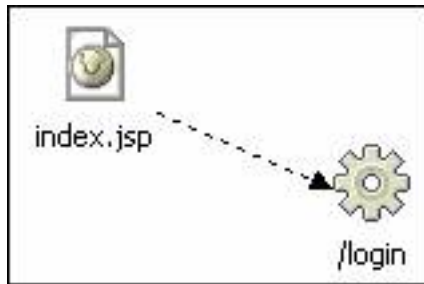
## Defining the connections and flow in the diagram

Add the connections defining the actual flow between the components you added to your application.

1. From the Palette view, select **Connection**.



2. Connect `index.jsp` to `login`. In the diagram, click `index.jsp` and then click `/login` to complete the connection.

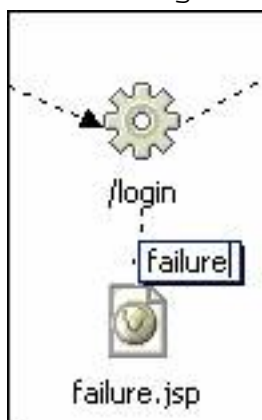


3. In the same way, connect login to success.jsp.
4. A label, <new>, appears. Name it success.

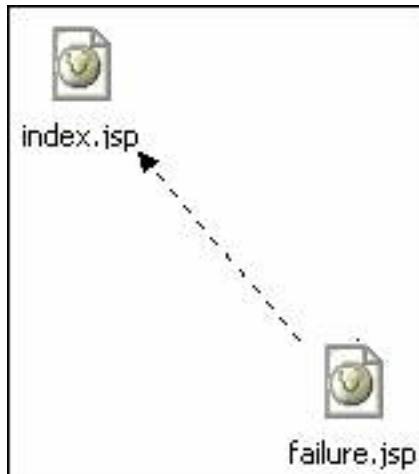


The success label will be used to refer to forwards to success.jsp.

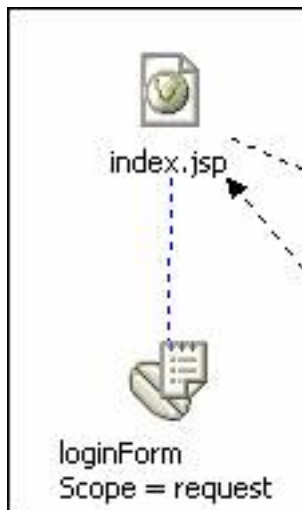
5. Connect login to failure.jsp. Label this connection failure.



6. Connect failure.jsp to index.jsp.



7. Connect `index.jsp` to `loginForm`.



## Adding documentation to the diagram

"Documentation?" I can hear you saying. "I code -- I don't need documentation!" Well, someone might need it eventually. Now is a good time to make note of why you added some of the components that you decided to add, and to describe what they are used for. You'll add documentation in the form of *notes*.

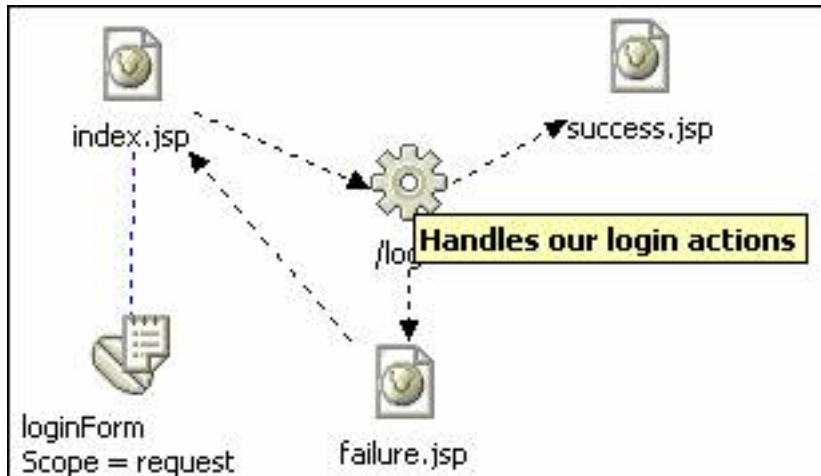
1. In the Palette view, select **Note**.



2. Click the **login** action map element.
3. Enter a descriptive message describing the action, as shown in the figure below.



4. Click **OK**.



5. Save the diagram (by pressing Ctrl-S) but do not close it.

## Section 3. Create a resource file

### Overview

A *resource file* abstracts textual data from an application. In particular, this makes it very easy to manage different languages in a single portal, or to manage consistent standards across an application. Resource files maintain a set of name/value pairs that can be accessed easily from JSPs. For example, you might have a strict requirement that submit buttons must be labeled "Submit." This is defined in the resource file, and the label is referenced from all the JSPs by its assigned name only. Not only does this ensure that you do not end up with one of your buttons labeled "Sumbit" by accident; it also means that if you decide to change the buttons to "Submit Query" across the board, there is only one location to edit.

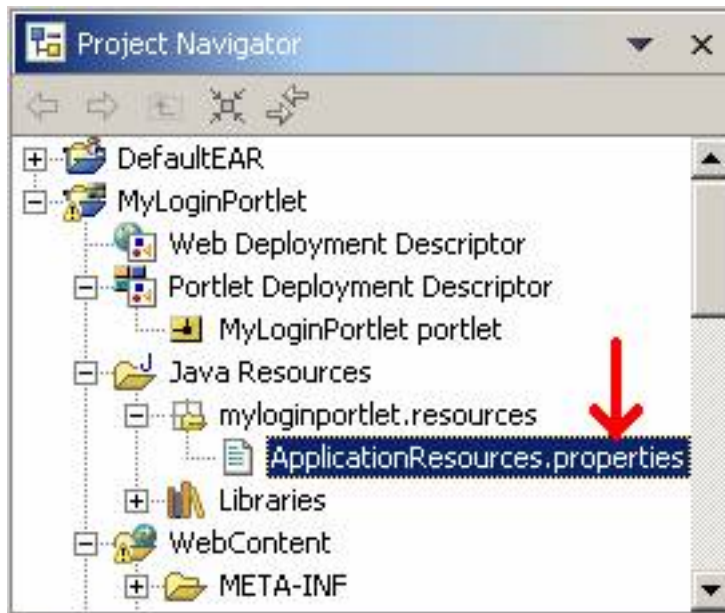
The resource file has many other more interesting uses, of course, such as header or footer text, titles of sections of the application or Web site, and so forth.

In this case, the most important use of this file is to provide a single location to manage error messages. The specific text returned from errors does not need to be nested in the code itself and therefore is far easier to manage in the long term (since I am sure you will use this particular login application for years to come).

### Editing the resource file

Open the resource file provided by the wizard to add some custom name/value pairs.

1. Open MyLoginPortlet/Java Resources/mystrutsportlet.resources/ApplicationResources.properties.



2. Uncomment the error lines by removing the # at the beginning of the error.header and error.footer lines.
3. Add two form field labels: form.login.username=Username and form.login.password=Password.
4. Now, add some good old scary red text to the errors. Locate the errors.header and errors.footer lines. Add `<font color="red">` to the header before the opening `<ul>` tag, and add `</font>` to the footer after the closing `</ul>` tag. The code should now look like this:

```
# Optional header and footer for <errors/> tag.
errors.header=<font color="red"><ul>
errors.footer=</ul></font>
form.login.username=Username
form.login.password=Password
```

5. Save and close the file.

## Section 4. Generate code from the diagram

### Overview

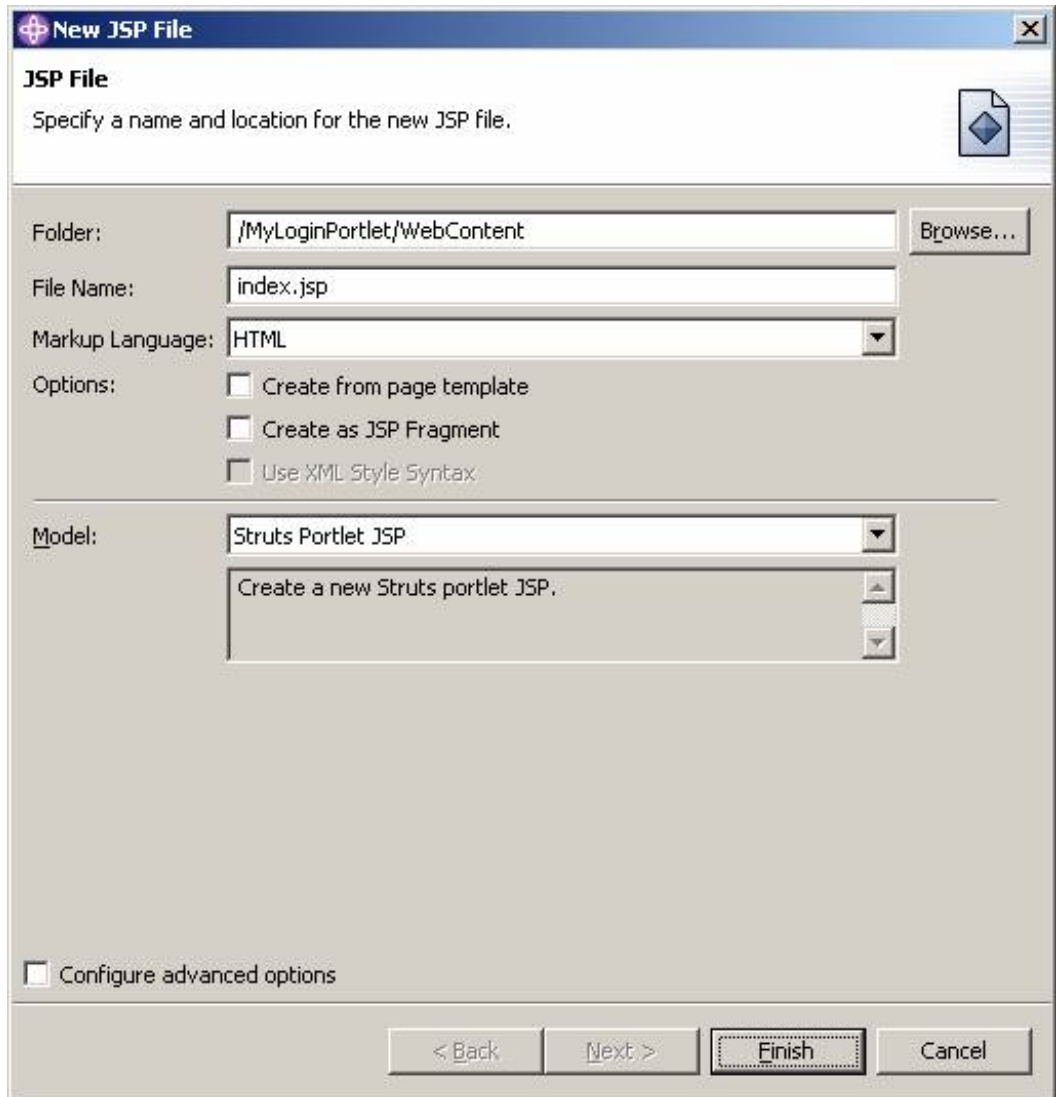
With your diagram and flow complete, you can now begin to export actual code from it. Much of the programming will be accomplished from the information you've already provided.

What you will find is that a functioning skeleton is generated to which you can add some custom code -- you can't let the wizards do everything for you, after all. But the code provides all the necessary nuts and bolts to run -- you just need to tie in the details.

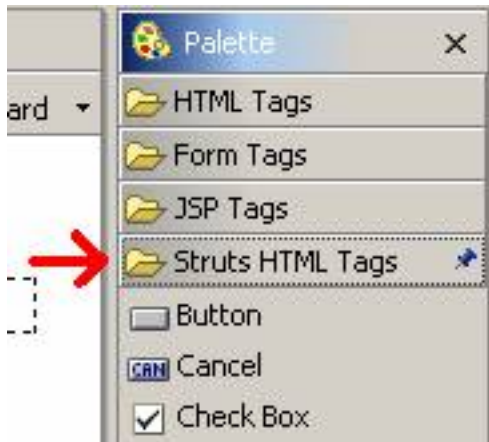
### Generating the initial login form

The Web Diagram tool makes it very simple to generate the files you modeled in your application. By simply double-clicking the components, the appropriate wizard launches to build the actual code represented by your model.

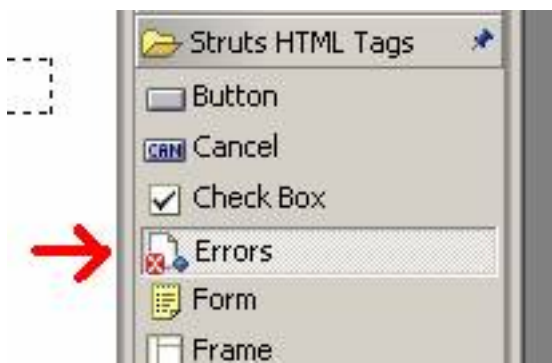
1. Double-click **index.jsp** to kick off the New JSP wizard.



2. Click **Finish**.
3. From the Palette view, click to expand **Struts HTML Tags**.



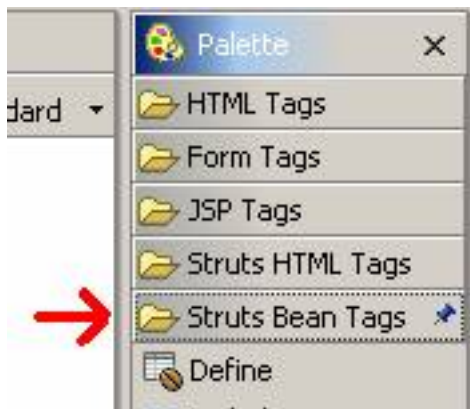
4. Click the **Errors** element.



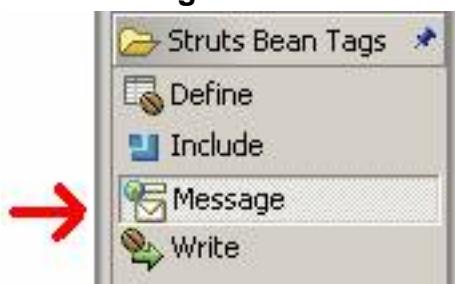
5. In the JSP, click between the `init` custom tag and the dotted box representing the form. A red bulleted list displays a list of errors that might appear.



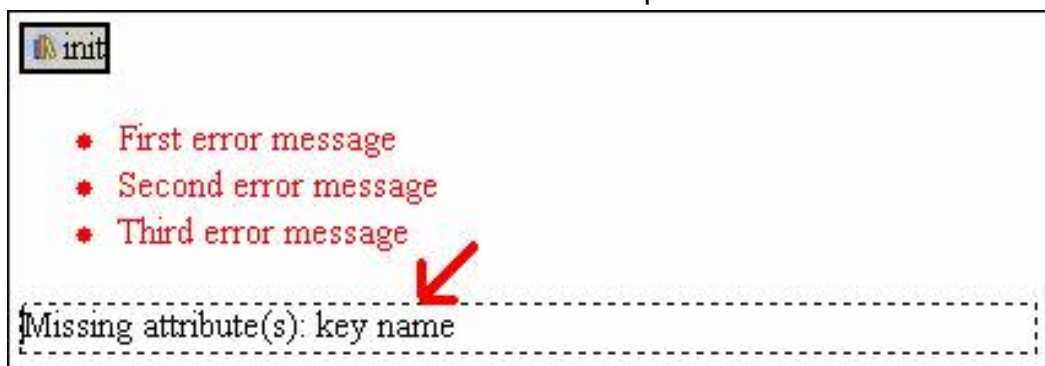
6. From the Palette view, click to expand **Struts Bean Tags**.



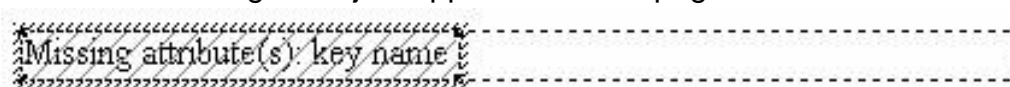
7. Click **Message**.



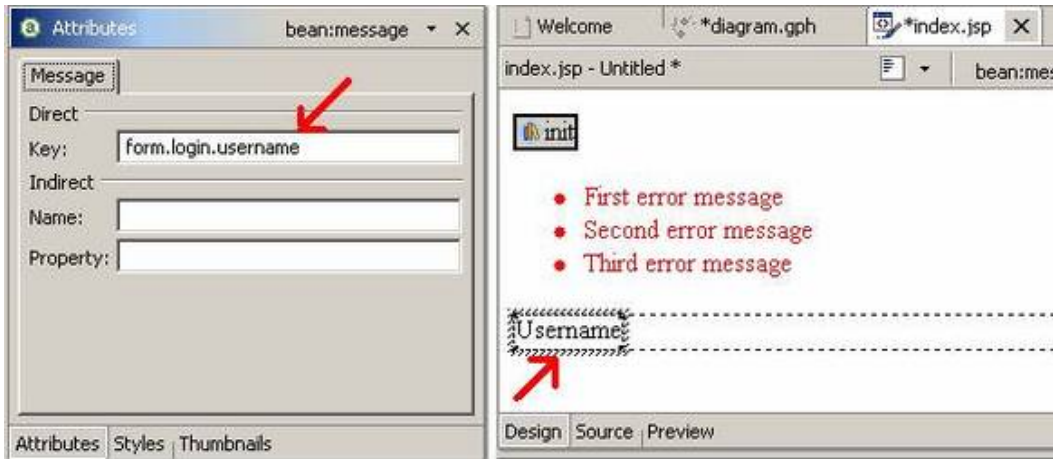
8. Click on the JSP within the dotted box that represents the form.



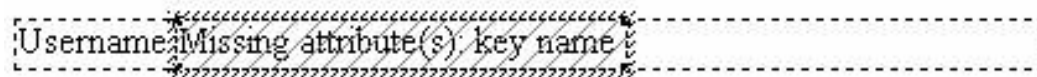
9. Select the message that just appeared on the page.



10. In the Attributes view, enter `form.login.username` for the Key field and press Enter.



11. Add another message next to Username.



12. Set the Key for that second message to form.login.password.



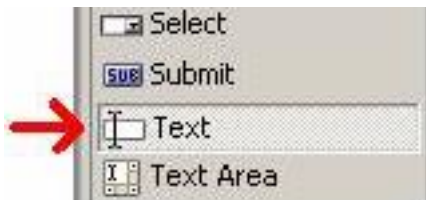
13. Place the cursor between Username and Password on the JSP and press Enter. This places a line break in the code.

14. Add two more line breaks after Password.

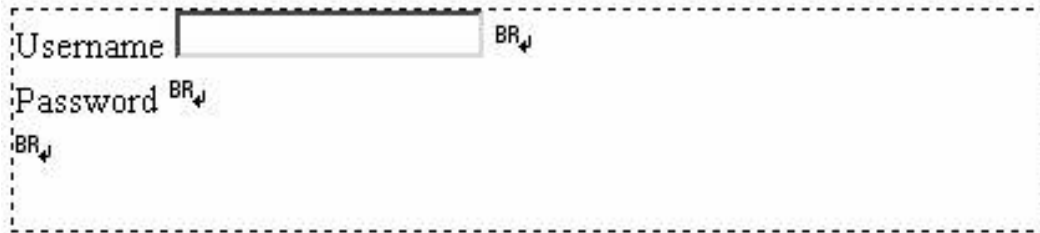


15. Back in the Palette view, click to expand **Struts HTML Tags**.

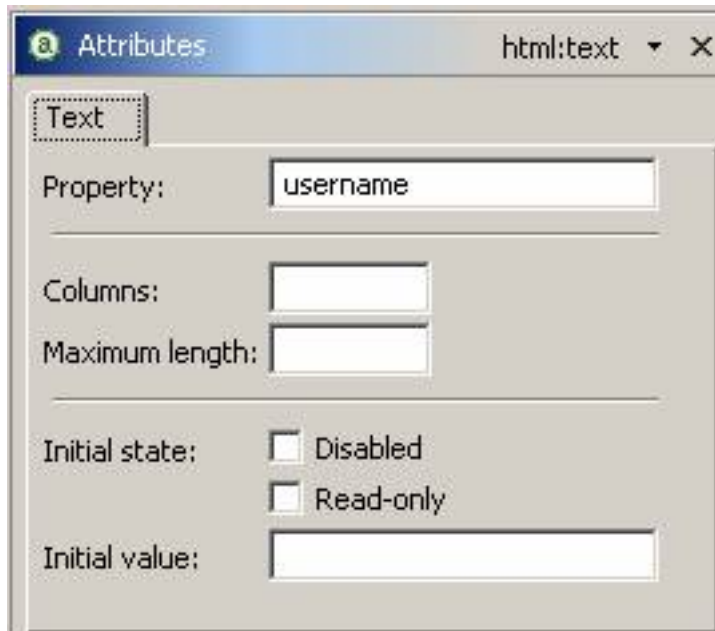
16. Click **Text**.



17. Click in the JSP to the right of the Username label to drop a text box.



18. Click the text box to select it.
19. In the Attributes view, set the Property field to `username`.



This sets the name of the text field to `username`.

20. From the Palette view, click to expand **Struts HTML Tags**.
21. Click the **Password** element.



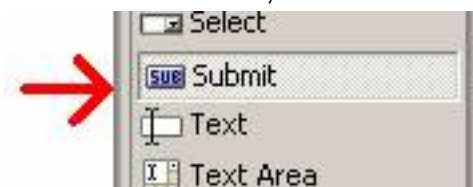
22. Click in the JSP to drop the element next to the Password label.

- 23. Select the text box.
- 24. In the Attributes view, set the Property field to `password`.



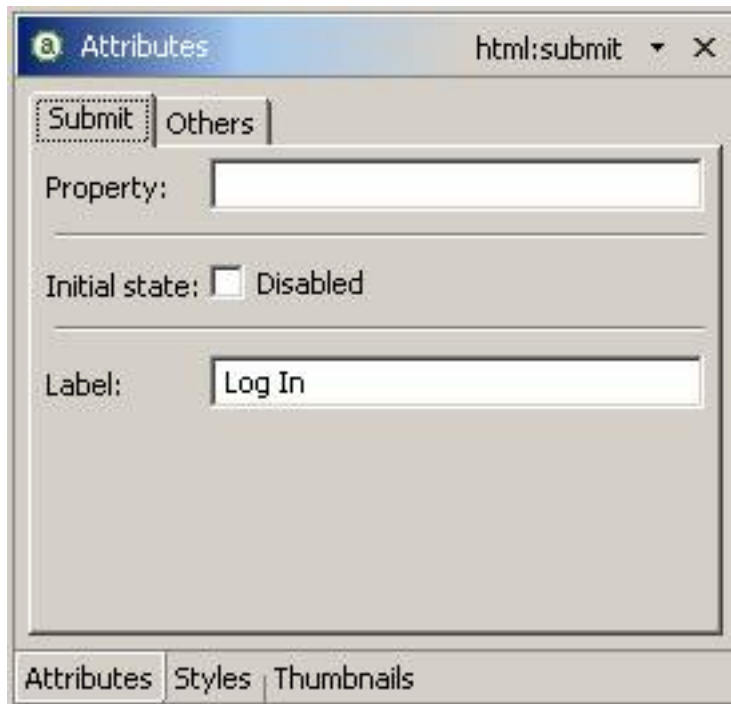
This sets the name of the text field to `password`.

- 25. In the Palette view, select the **Submit** element.



- 26. Click on the form below the Password field.

27. Select the button that you just placed.
28. In the Attributes view, enter `Log In` in the Label field and press Enter.



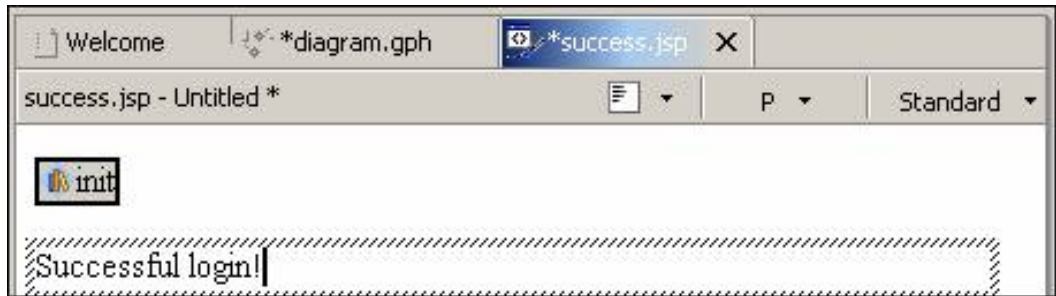
The JSP shows `Log In` on the button.

29. Save and close `index.jsp`.

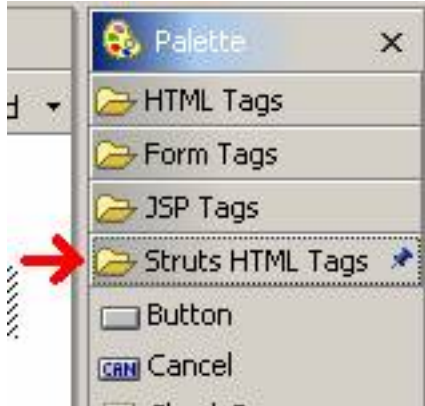
## Creating the successful login JSP

Next, generate the successful login page. Again, you will simply double-click the component in your diagram to launch the New JSP wizard, and then take things from there.

1. Back on the diagram, double-click the `success.jsp` icon.
2. Click **Finish** to generate the new JSP.
3. In the JSP, replace the text `Place content here.` with `Successful login!`



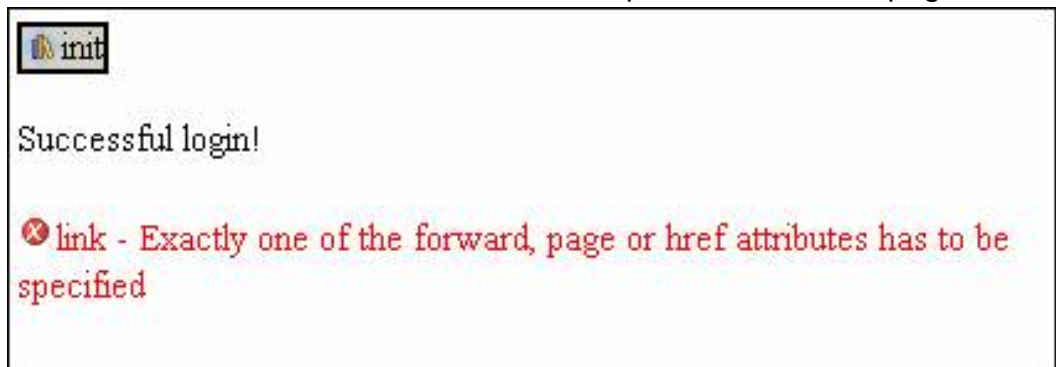
- From the Palette view on the right, click to expand **Struts HTML tags**.



- Click **Link** to select it.



- Click in the JSP below the added text to drop the link onto the page.



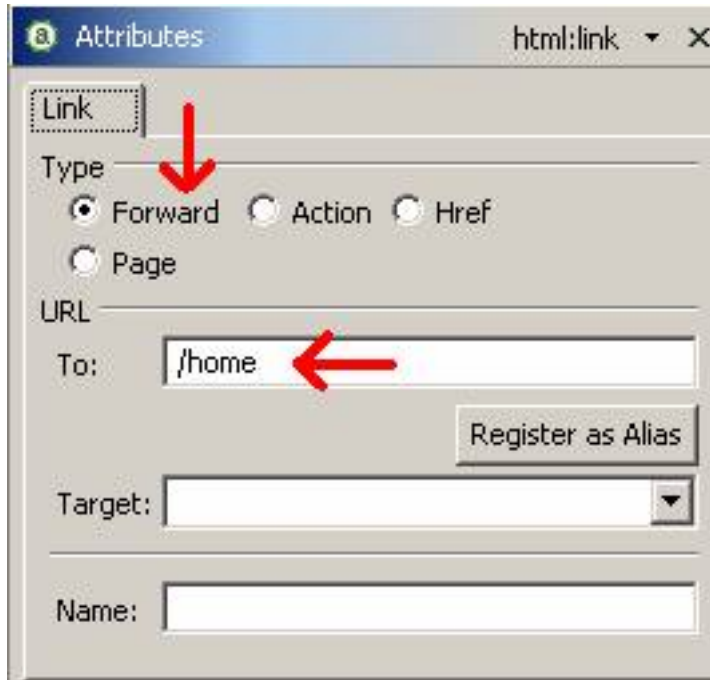
Oooh, look at the pretty red color!

- Select the link.

Successful login!

link - Exactly one of the forward, page or href attributes has to be specified

8. In the Attributes view, click the **Forward** radio button.
9. In the URL To field, enter /home and press Enter.



The screenshot shows the 'Attributes' dialog box for an HTML link. The 'Type' section has four radio buttons: 'Forward' (selected), 'Action', 'Href', and 'Page'. The 'URL To:' field contains '/home'. A red arrow points to the 'Forward' radio button, and another red arrow points to the '/home' text in the 'To:' field. There is also a 'Register as Alias' button and 'Target:' and 'Name:' fields.

10. The error in the JSP goes away, but you need to add the clickable text.
  - If you did not click elsewhere on the page, you will see the link still oddly selected by an outline around the cursor -- type Home.



- If you lost the link block before adding the clickable text and cannot seem to grab it again, click the Source tab and locate `<html:link forward="/home"></html:link>`. Enter Home between the opening and closing tags.



Save and close the file (you'll fix the warning in [Edit the Struts configuration file](#) ).

## Creating the failed login JSP

This JSP will be called if the action bean forwards to the target labeled *failure*. You may be wondering why the initial form and this `failure.jsp` file both manage errors. This is simply to illustrate that you can have two layers of tests for a submission or requested action.

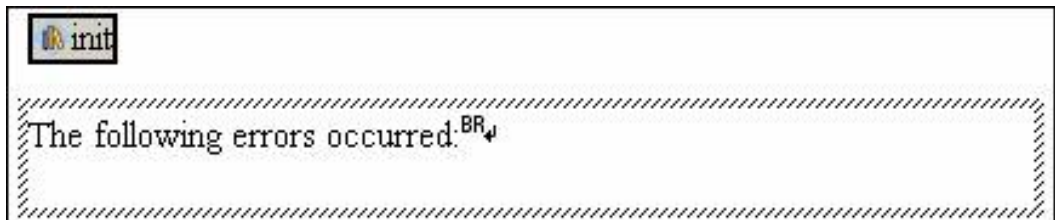
In the first layer, the form bean -- noting problems with the submission itself -- will kick the request out and push the user back to the initial form to try again. If the form submission is fine as far as the process of filling out the form is concerned, the action bean has an opportunity to determine the direction in which the user should be sent based on whatever checks you choose to include. In this case, if the test you set up fires, it will redirect to this target.

Here you will flag errors and define a failure page. As I mentioned earlier, this redirection could be for reasons other than errors; you could, for instance, redirect to state-specific pages based on a selected state on the initial form. It could also be based on criteria having nothing at all to do with the form information, like time of day, date, server resources, available products queried quickly from a database, or many other examples.

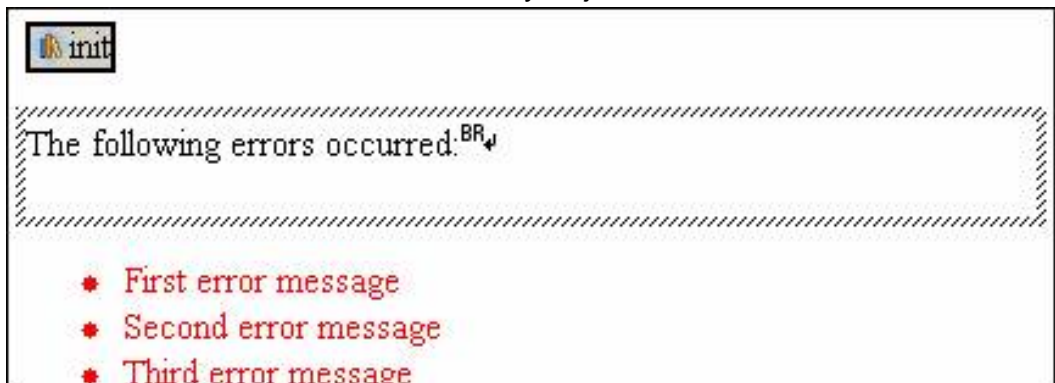
1. Double-click the `failure.jsp` icon and click **Finish** to generate the JSP. Make sure you are in the Design view.



2. Replace `Place content here.` with `The following errors occurred:` and press Enter to insert a line break.

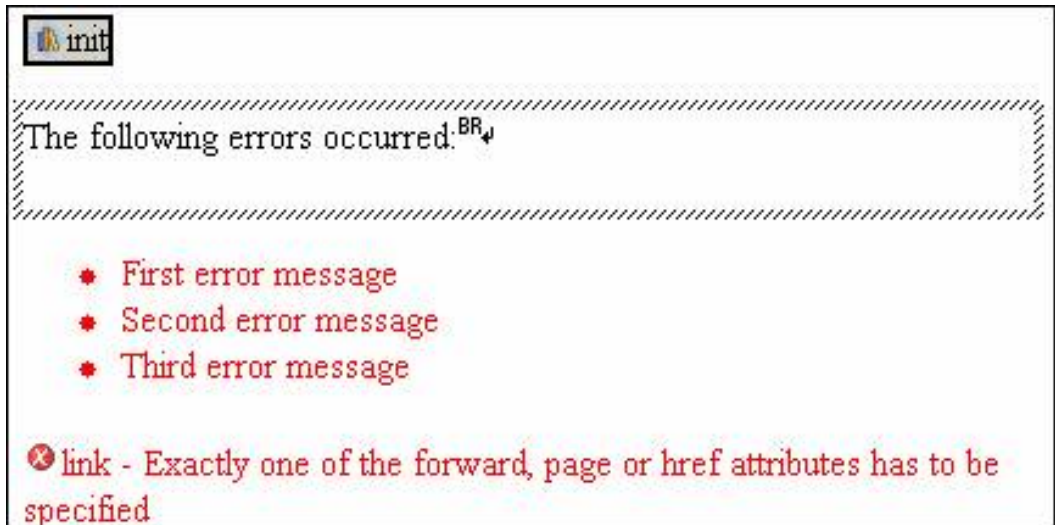


3. From the Palette view on the right, click to expand **Struts HTML tags**.
4. Click **Errors**.
5. Click in the JSP below the sentence you just added.



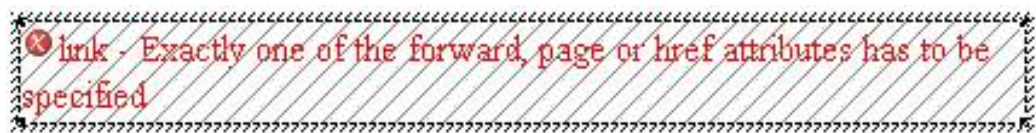
A red bulleted list appears.

6. Back in the Palette view, click **Link**.
7. Click in the JSP under the red bulleted list of errors.

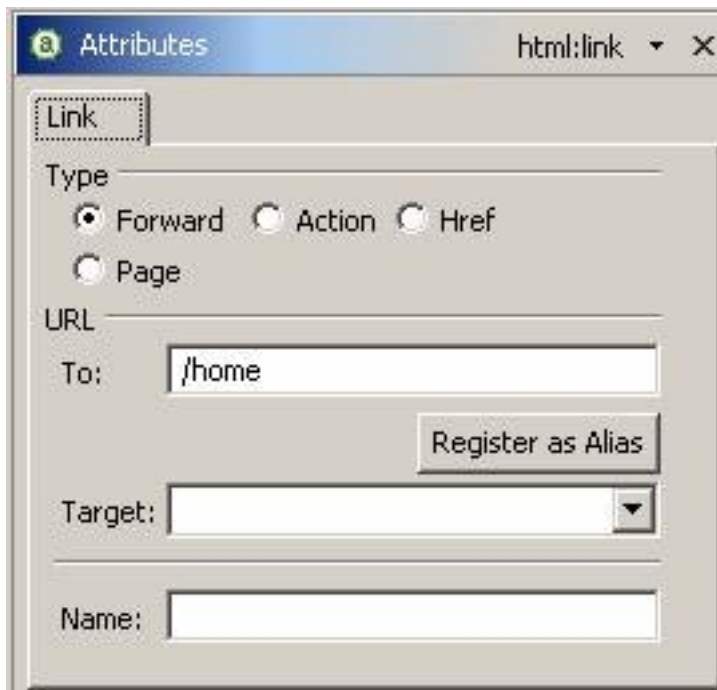


Oooh, more pretty red color!

8. Select the newly added link.

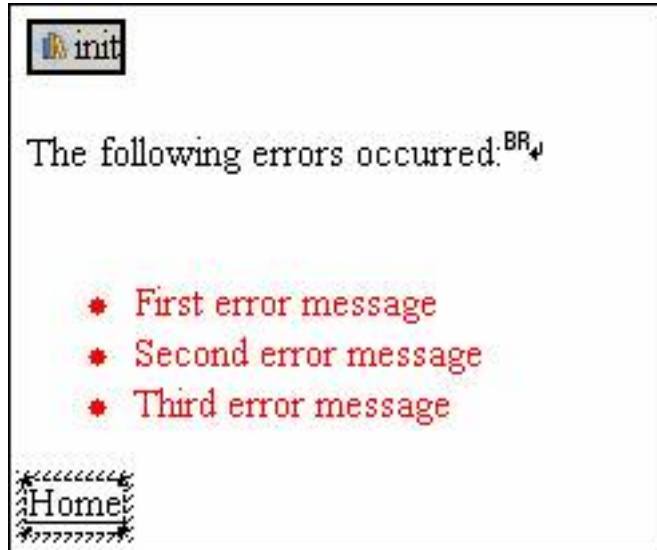


9. In the Attributes view, click the Forward radio button.
10. Enter /home in the **URL To** field.



11. The error in the JSP will go away, but you still need to add the clickable text:

- If you did not click elsewhere on the page, the link is still oddly selected by an outline around the cursor. Type `Home`.



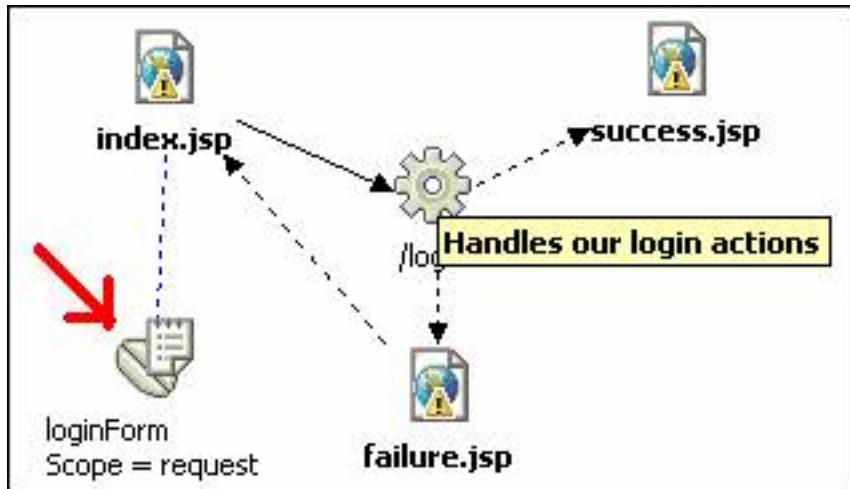
- If you lost the link block before adding the clickable text and cannot seem to grab it again, click the Source tab and locate `<html:link forward="/home"></html:link>`. Enter **Home** between the opening and closing tags.

12. Save and close the file.

## Creating the form bean

The `loginForm` is a simple Java(TM) class that maintains properties mapped to the fields that you want to watch -- in this case, the username and password fields of the `index.jsp` file. Because the relationship between `index.jsp` and `loginForm` was specified, the Struts framework sets properties on the bean from values entered on the form. Therefore, validation tests can be run based on those bean properties. If something fails, these properties are dumped back down to input fields originally mapped to the JSP -- and thankfully, handled by the framework, not you.

1. Double-click the `loginForm` form bean icon on the diagram.



2. On the New Form Bean window, accept the defaults and click **Next**.

**New Form-Bean**

Specify the information needed to define your new form-bean mapping.

Project name: MyLoginPortlet

Configuration File Name: /WEB-INF/struts-config.xml

Form Bean Name: loginForm

**Reference**

An existing ActionForm

Class: org.apache.struts.action.ActionForm

**Create**

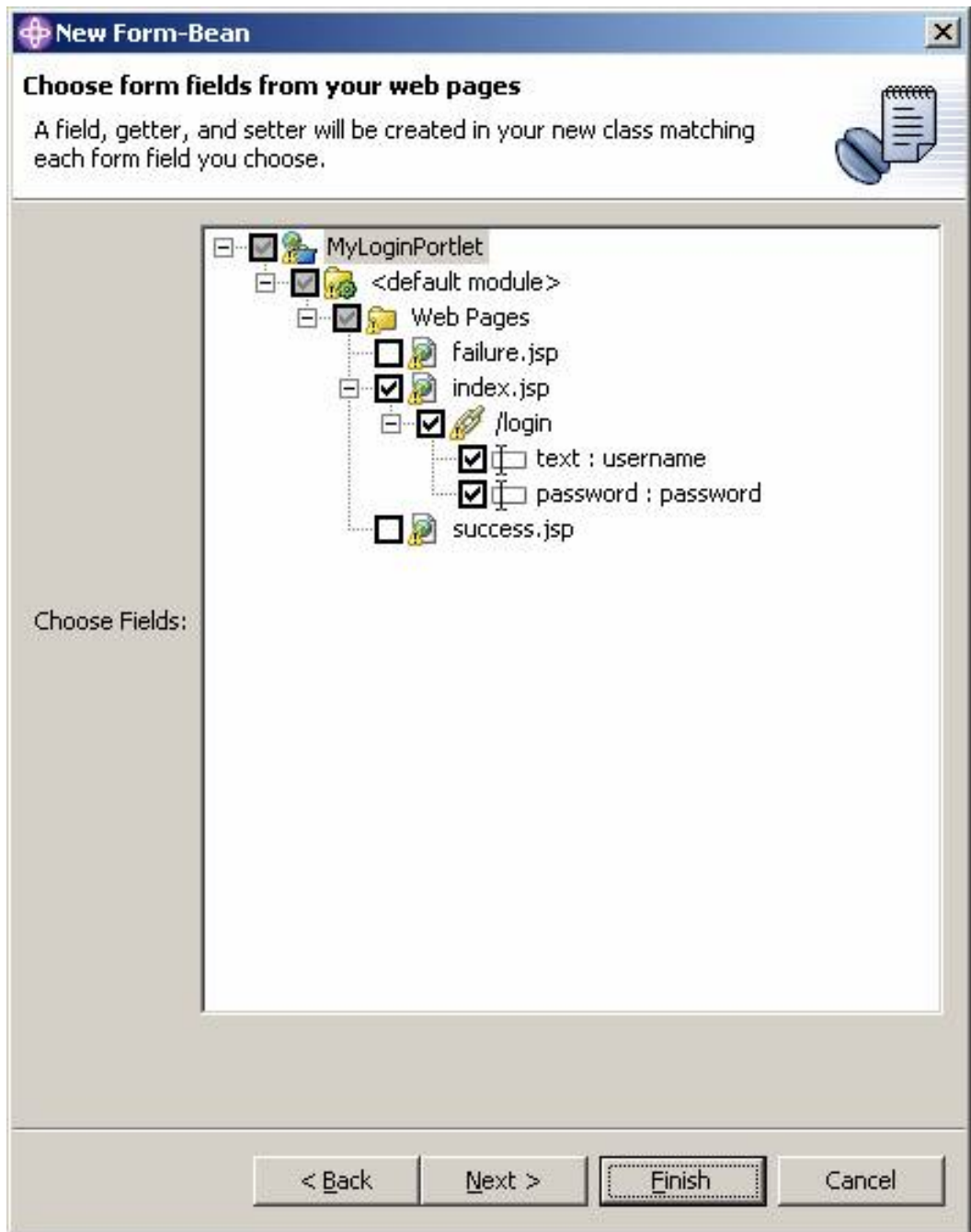
Create an ActionForm class or Struts dynaform using DynaActionForm

Model: Generic Form-Bean Mapping

Description: Generate a new form-bean mapping

< Back **Next >** Finish Cancel

3. Expand down to MyLoginPortlet/<default module>/Web Pages/index.jsp//login.
4. Check two form fields: username and password.




5. Click **Finish**.
6. Open the `ApplicationResources.properties` file again.

## Editing the resource file again

Add a few more name/value pairs to your resource file. Use these additions to display errors back to the end users.

1. Add the following three lines:

```
error.login.no_username=<li>Please enter a username</li>
error.login.no_password=<li>Please enter a password</li>
error.login.bad_password=<li>"password" is a bad password</li>
```



```
Welcome | *diagram.gph | LoginForm.java | ApplicationResources.properties X
# Optional header and footer for <errors/> tag.
errors.header=<font color="red"><ul>
errors.footer=</ul></font>
form.login.username=Username
form.login.password=Password
error.login.no_username=<li>Please enter a username</li>
error.login.no_password=<li>Please enter a password</li>
error.login.bad_password=<li>"password" is a bad password</li>
```

2. Save and close the file.

## Editing the generated form bean

Edit the form bean to add some custom validation rules. These rules are used to ensure that end users fill in their forms correctly.

1. In the LoginForm.java file, locate the `validate()` method.

```
public ActionErrors validate(
    ActionMapping mapping,
    HttpServletRequest request) {

    ActionErrors errors = new ActionErrors();
    // Validate the fields in your form, adding
    // adding each error to this.errors as found, e.g.

    // if ((field == null) || (field.length() == 0)) {
    //     errors.add("field", new org.apache.struts.action.Acti
    // }
    return errors;
}
```

2. Uncomment the `if` statement.

```
if ((field == null) || (field.length() == 0)) {
    errors.add("field", new org.apache.struts.action.ActionError("error.login.no_username"));
}
```

- Now you need to change the conditional to test a username, not the sample field. Change the reference to `field==null` to `getUsername()==null`, and the reference `field.length()==0` to `getUsername().length()==0`.

```
if ((getUsername() == null) || (getUsername().length() == 0)) {
```

- In the line `errors.add("field", ...)`, change `field` to `username`.

- Change `error.field.required` to `error.login.no_username`.

```
errors.add("username", new org.apache.struts.action.ActionError("error.login.no_username"));
```

- Copy the `if` statement and make the following changes to test for the entered password:

- Change the conditional to test `getPassword`, not `getUsername`.
- Change the reference to `getUsername()==null` to `getPassword()==null`.
- Change the reference `getUsername().length()==0` to `getPassword().length()==0`.
- In the line `errors.add("username", ...)`, change `username` to `password`.
- Change `error.login.no_username` to `error.login.no_password`.

The `if` statement code should look like the following:

```
if ((getUsername() == null) || (getUsername().length() == 0)) {
    errors.add("username", new org.apache.struts.action.ActionError("error.login.no_username"));
}
if ((getPassword() == null) || (getPassword().length() == 0)) {
    errors.add("password", new org.apache.struts.action.ActionError("error.login.no_password"));
}
```

- Save and close the `LoginForm.java` file

## Generating the action mappings and action bean

Now you need to generate the action bean.

- Back on the diagram, double-click the action labeled **/login**.

- For the Form Bean Name, select **loginForm**.
- For the Form Bean Scope, select **request**.

**New Action Mapping**

Specify the information needed to define your new action mapping.

Project name: MyLoginPortlet

Configuration File Name: /WEB-INF/struts-config.xml

Action Mapping Path: /login

Name	Path	Context relative?
success	/success.jsp	false
failure	/failure.jsp	false

Forwards:

Form Bean Name: loginForm

Form Bean Scope: request

**Reference**

An existing Action class

Class: com.ibm.wps.struts.action.StrutsAction

**Create**

Create an Action class

Model: Struts Portlet Framework Action Mapping

Description: Generate a new Struts Portlet Framework action mapping

< Back   Next >   **Finish**   Cancel

- Click **Finish**.

5. Notice that one of the broken links went away now that there is an action labeled /login.
6. Locate the try-catch block.

```
try {  
  
    // do something here  
  
} catch (Exception e) {  
  
    // Report the error using the appropriate name and ID.  
    errors.add("name", new ActionError("id"));  
  
}
```

7. In the try section, add the following if statement:

```
try {  
  
    // do something here  
    if (loginForm.getPassword().equals("password"))  
        errors.add("login", new ActionError("error.login.bad_password"));  
  
} catch (Exception e) {  
  
    // Report the error using the appropriate name and ID.  
    errors.add("name", new ActionError("id"));  
  
}
```

This tests to see if the word password is entered for the password.

8. Locate the if (!errors.isEmpty()) { statement.

```
// If a message is required, save the specified key(s)  
// into the request for use by the <struts:errors> tag.  
  
if (!errors.isEmpty()) {  
}  
  
// Write logic determining how the user should be forwarded.  
forward = mapping.findForward("success");  
forward = mapping.findForward("failure");  
  
// Finish with  
return (forward);
```

9. Add saveErrors(request, errors); to save the errors collected.
10. Move the forward = mapping.findForward("failure"); listed below into this if clause (make sure that it is deleted from its original

location).

11. Add an `else` block and move the `forward = mapping.findForward("success");` line into it. The complete code should now look like this:

```
// If a message is required, save the specified key(s)
// into the request for use by the <struts:errors> tag.

if (!errors.isEmpty()) {
    saveErrors(request, errors);
    forward = mapping.findForward("failure");
} else {
    forward = mapping.findForward("success");
}
// Write logic determining how the user should be forwarded.

// Finish with
return forward;
```

12. Save and close the action bean `LoginAction.java`.
- 

## Section 5. Edit the Struts configuration file

### Overview

The file `struts-config.xml` holds all the specific mapping information and is consulted at each request. This file specifies details of the action being invoked: who is supposed to be calling the action, the form beans associated with the initiating form or JSP, action targets and forwards, and so on. For the most part, the wizards have already added the necessary information to this file for you. However, you do need to make a couple of small additions.

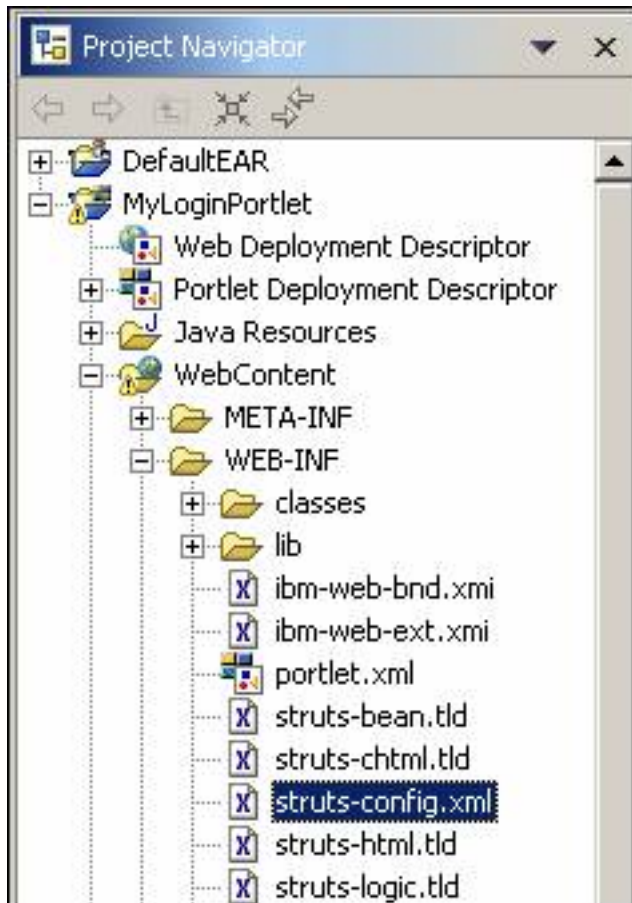
First, you need to specify the JSP that is going to initiate the action `/login`; you also need to specify what `/home` means, since you've created links that reference this target. Simply put, you need to map a location to `/home`.

### Editing `struts-config.xml`

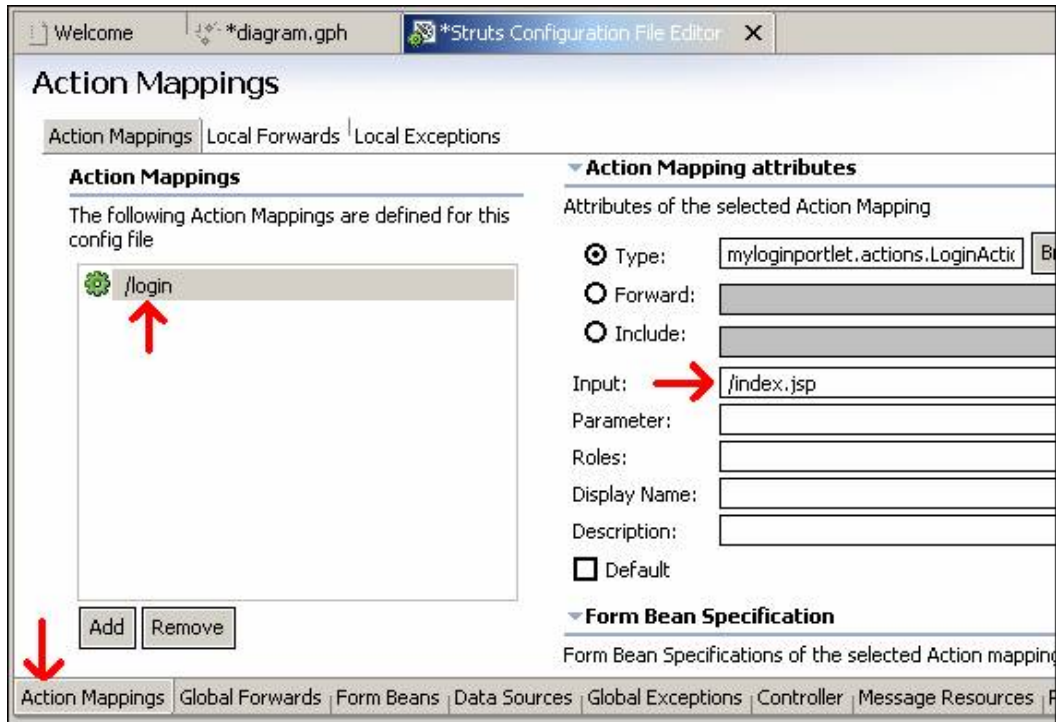
`struts-config.xml` defines all your configurations and is referenced on each server request (when a form is submitted, for example). Much of what you've already

generated has updated this file for you, but you need to manually add a forward here.

1. In the Project Navigator view, expand MyLoginPortlet/WebContent/WEB-INF and open struts-config.xml.

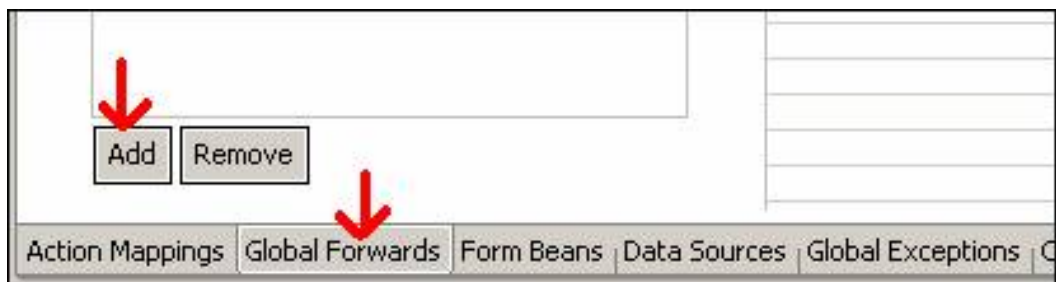


2. On the Action Mappings page, select **/login** and enter `/index.jsp` for the Input field.

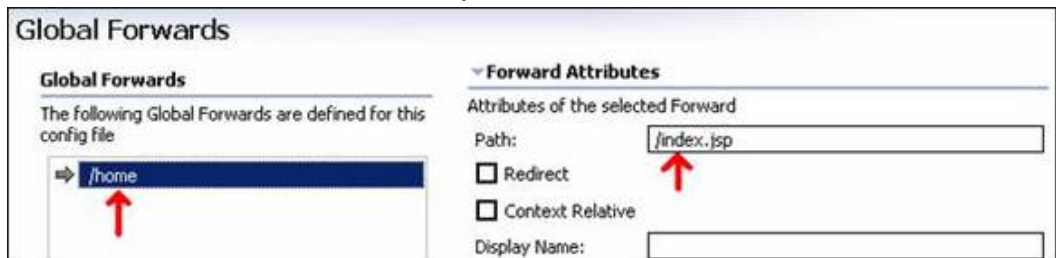


3. Click the Global Forwards tab.

4. Click **Add**.



5. Change the name of the added forward from <no name> to /home, and then enter /index.jsp for the path.



6. Save and close the file.

---

## Section 6. Test the application

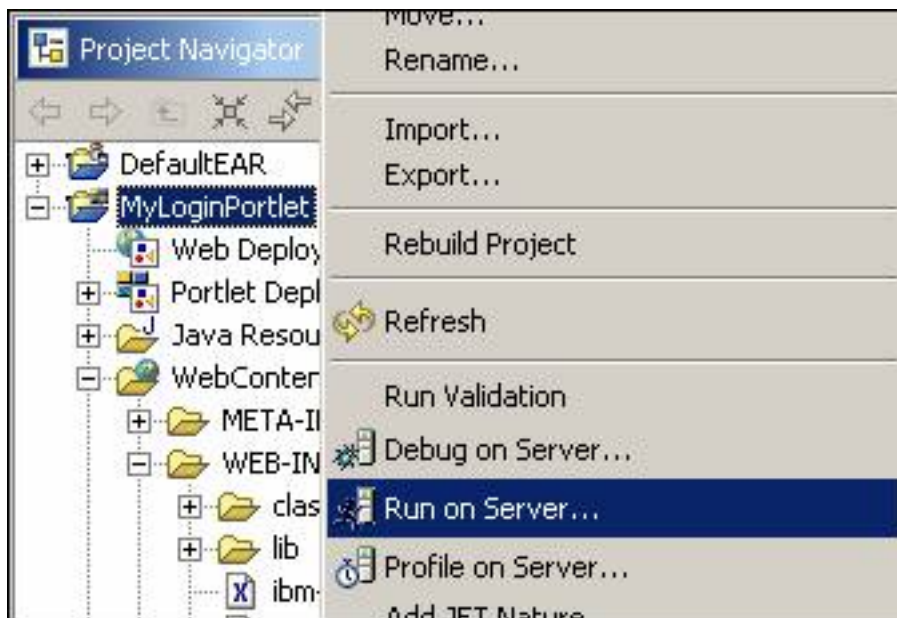
### Overview

Now that you have created your application, the last thing you need to do before declaring it ready for production is test it -- always a good idea. Luckily, you have the handy-dandy Portal Test Environment available. This is an easy-to-use tool that is at your fingertips because you need to do very little to deploy it -- simply select **Run on Server** and off it goes.

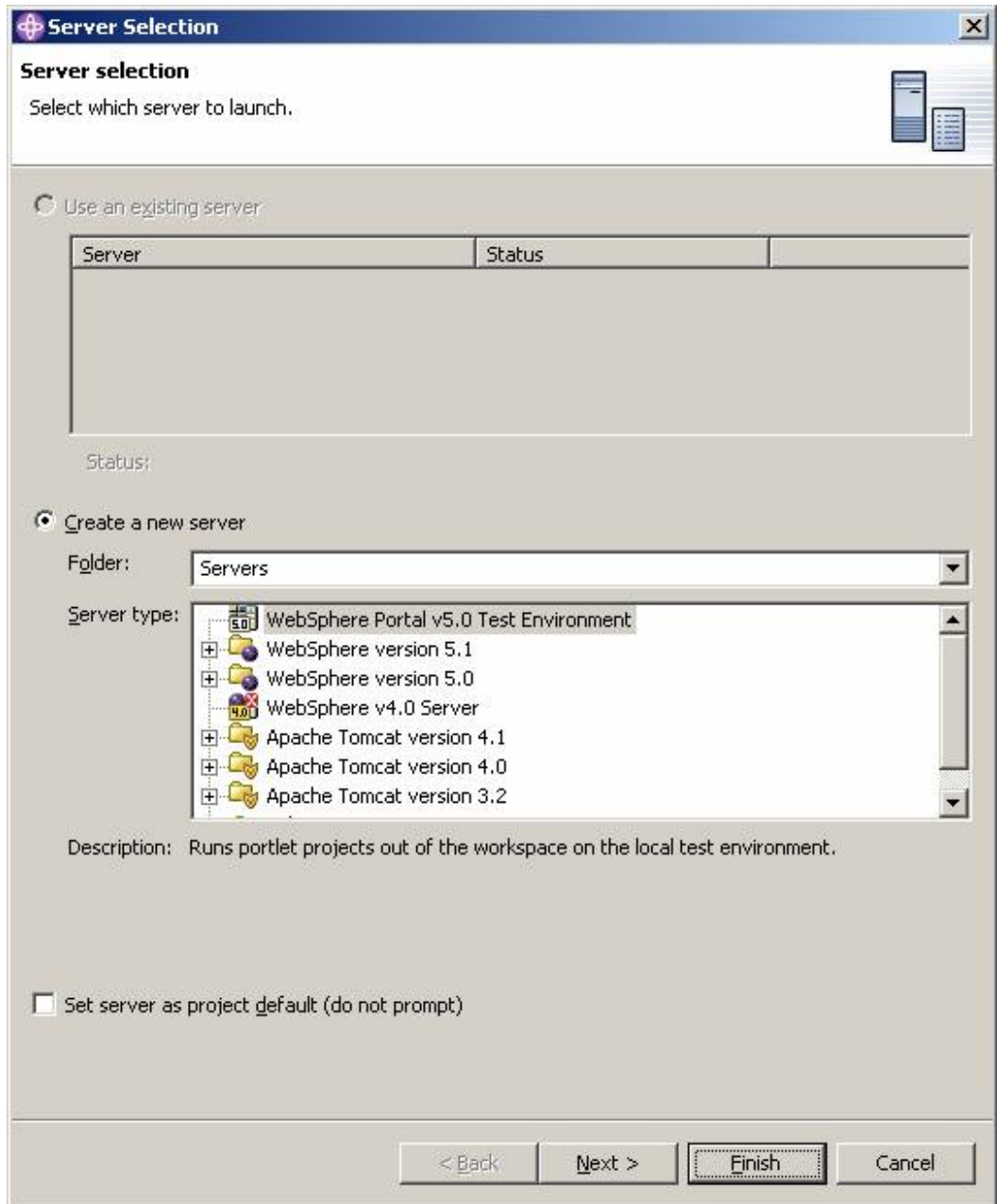
### Testing the application

Let's give the application a run and see if it all works.

1. In the Project Navigator view, right-click **MyLoginPortlet** and select **Run on Server**.



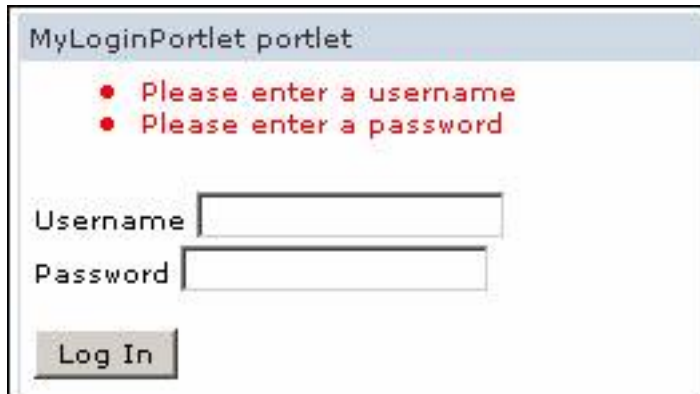
2. If you are asked to save the diagram, click **OK**.
3. On the Server Selection window, click **WebSphere Portal v5.0 Test Environment Server Type** and then click **Finish**.



Now wait, wait, and wait some more. It takes a while to load everything -- eventually, your portlet will load in WebSphere Studio's embedded browser.

Isn't it pretty? Now you can test it!

1. Click **Log In** without entering a username and password.




MyLoginPortlet portlet

- Please enter a username
- Please enter a password

Username

Password

2. Enter a username and *no* password and then click **Log In**.



MyLoginPortlet portlet

- Please enter a password

Username

Password

As you've noticed with these two inputs, the form bean and Struts input fields manage the testing and pre-population of the fields -- you didn't need to do those yourself with special JavaScript or JSP Java code.

3. Enter *password* into the password field and click **Log In**.



MyLoginPortlet portlet

The following errors occurred:

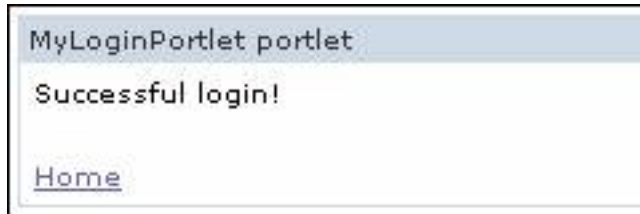
- "password" is a bad password

[Home](#)

Note that the action class ran an extra set of checks -- in this case, to make sure that a password of `password` wasn't submitted. Since you did in fact submit that password, your special login error page (labeled as `failure`) was returned. In theory, you could have several different kinds of checks that might forward to several different JSPs, rather than just success and failure pages. You can have 50 different pages as the targets of the action test based on the U.S. state entered in a shipping

address form. You can have male and female pages for a medical application, age groups, language preferences, and many other examples.

4. Click **Home**.
5. Enter a username and password (other than `password`) and click **Log In**.



Whoopie! It worked!

---

## Section 7. Summary

In this tutorial, you took a look at the tip of the iceberg -- but still the core -- of implementing a Struts-based framework into a portal application. You obviously could do a lot more interesting work, as you would have noticed by all the other components you could have added. Additionally, you could have built a system that was far more complex -- as most real-world systems would end up being.

What was covered here described preparing the application, developing the primary components, and setting up some simple flow examples. Any Struts-based portal application would begin from this point and certainly could become far more complex as needed.

# Resources

## Learn

- "[Developing and deploying a Struts application as a WebSphere Portal V5 Portlet](#)," Tim Hanis, Jim Bonanno, and Lisa Tomita (*developerWorks*, January 2004) is a great explanation of the Struts Portal Framework.
- Check out the [official Struts site](#) at the Apache Project.
- "[Migrating a Struts application to WebSphere Portal](#)," Colin Yu (*IBM WebSphere Developer Technical Journal*, March 2004) explains how to move an existing Struts application into a WebSphere framework.
- Stay current with [developerWorks technical events and webcasts](#).

## Get products and technologies

- Download [Portal Toolkit V.5.0.2.2](#)
- [CutAndPasteCode.txt](#)
- 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.