Getting Started with Red Hat Developer Studio

Table of Contents

1 Getting Started with Red Hat Developer Studio

1.1. Technical Requirements

1.1.1. Java

Before installing Red Hat Developer Studio, make sure you have one of the following versions of Java installed:

- JDK 1.4.2 or higher

It must be the full JDK, not just the JRE (Java Runtime Environment). For more information about target operating environments and compatible Java versions, refer to the JBoss web site. Please note that the Eclipse documentation states that you only need a Java Runtime Environment for Eclipse, but usage of Red Hat Developer Studio requires a full JDK in order to compile Java classes and JSP pages.

1.1.2. Operating Systems

- Windows
- Linux

1.2. Installing

1.2.1. Installing from the downloaded version

- Download the appropriate installation file for your platform from www.exadel.com/web/portal/download.
- Run install.jar and follow the instructions presented by the installation wizard.

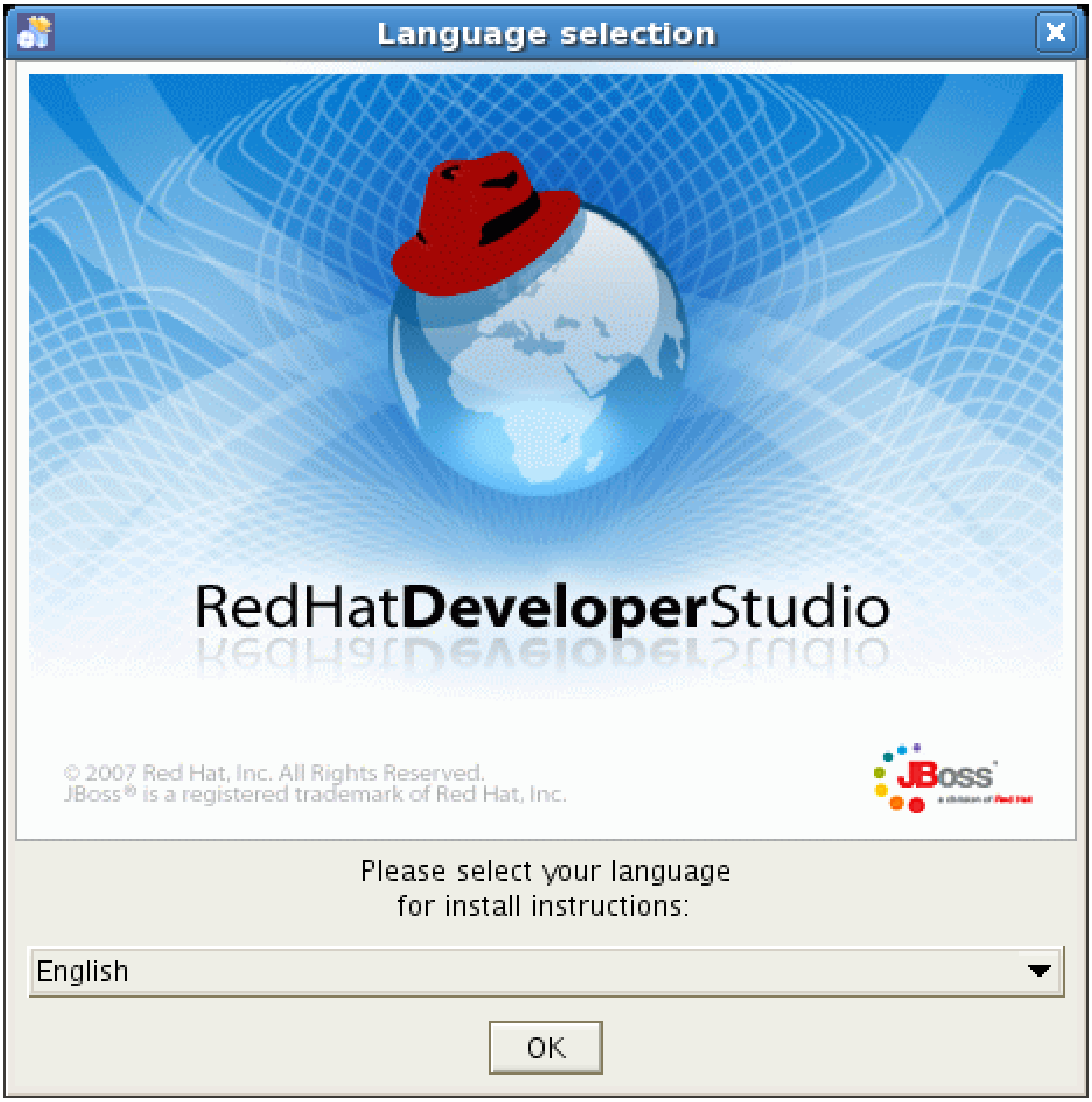
Language selection
RedHatDeveloperStudio
© 2007 Red Hat, Inc. All Rights Reserved.
JBoss® is a registered trademark of Red Hat, Inc.
JBoss
Please select your language
for install instructions:
English
OK

Figure 1.1. Red Hat Developer Studio Installation Wizard

After installation process you will have all required platforms to run Red Hat Developer Studio:

- Eclipse 3.2.2
- JBoss Application Server

Red Hat Developer Studio comes with the following plug-ins:

- WTP runtime 1.5
- GEF runtime 3.2

- JEM runtime 1.2
- EMF and SDO runtime 2.2
- XSD runtime 2.2
- Hibernate-tools-3.1.0 beta5 (without WTP, EMF, GEF, and other non-Hibernate plug-ins from Hibernate-tools-3.1.0 beta5)
- QuickImage 0.2.0
- Spring IDE 1.3.2
- Sun Glassfish 0.3

1.2.2. Installing through the update Site

TBD

1.3. Running for the First Time

- Launch Eclipse
- Select Window/Open Perspective.../Other...
- Select Red Hat Developer Studio from the perspective list

We recommend starting Eclipse with VM arguments to make more memory (heap) available for large-scale projects. Open the file <eclipse>\eclipse.ini and set the following:

eclipse -clean -vmargs -Xms512m -Xmx512m -XX:MaxPermSize=128m

This setting will allow the Java heap to grow to 512MB.

1.4. Upgrading

To upgrade, just uninstall your current version and install the new version.

- Make sure Eclipse is not running.

  Uninstall your current version of Exadel Studio. You can do one of the following:

- Run the Red Hat Developer Studio uninstaller
- Delete the file {EclipseHome}/links/com.exadel.studio.link

  Either method will uninstall Exadel Studio.

- Start Eclipse:

eclipse -clean

- Close Eclipse
- Install the new version of Red Hat Developer Studio.
- After installation, start Eclipse with the -clean option:
  eclipse -clean

This step is very important as it will clear the Eclipse cache of the old uninstalled plug-in.

Note:

It is recommended to uninstall Red Hat Developer along with Eclipse and install Red Hat Developer Studio into a new clean Eclipse

1.5. Uninstalling

Make sure Eclipse is not running.

Uninstall your current version of Red Hat Developer Studio. You can do one of the following:

- Run the Red Hat Developer Studio uninstaller
- Delete the file {EclipseHome}/links/com.exadel.studio.link

  Either method will uninstall Red Hat Developer Studio.

- Start Eclipse with the -clean option:

  eclipse -clean

This step is very important as it will clear the Eclipse cache of the uninstalled plug-in.

1.6. Subscription

TBD

1.7. Support

If you have comments or questions, you can send them to support@exadel.com [mailto:support@exadel.com] or Red Hat Developer Studio Forum [http://www.jboss.com/index.html?module=bb&op=viewforum&f=258].

When writing to support, please include the following information:

- Eclipse version

- Red Hat Developer Studio version
- Exact error message
- Exact steps you took to get the error

2

Getting Started Guide for Creating a JSF Application

2.1. Creating a Simple JSF Application

We are going to show you how to create a simple JSF application using the Red Hat Developer Studio plug-in for Eclipse. The completed application will ask a user to enter a name and click a button. The resulting new page will display the familiar message, "Hello <name>!" This document will show you how to create such an application from the beginning, along the way demonstrating some of the powerful features of Red Hat Developer Studio. You will design the JSF application and then run the application from inside Red Hat Developer Studio. We'll assume that you have already launched Eclipse with Red Hat Developer Studio installed and also that the Red Hat Developer Studio perspective is the current perspective. (If not, make it active by selecting Window/Open Perspective/Red Hat Developer Studio from the menu bar or by selecting Window/ Open Perspective/Other... from the menu bar and then selecting Red Hat Developer Studio from the Select Perspective dialog box.)

2.2. Setting Up the Project

We are first going to create a new project for the application.

1. Go to the menu bar and select File/New/Project... .
2. Select Red Hat Developer Studio/JSF Project in the New Project dialog box.
3. Click Next
4. Enter jsfHello as the project name.
5. Leave everything else as is, and click Finish.

2.3. The JSF Application Configuration File

A jsfHello node should appear in the upper-left Package Explorer view.

Figure 2.1.

6. Click the plus sign next to jsfHello to reveal the child nodes.
7. Click the plus sign next to WebContent under jsfHello.

8. Click the plus sign next to WEB-INF under WebContent.
9. Then double-click on the faces-config.xml node to display the JSF application configuration file editor.

2.4. Adding Navigation to the Application

In our simple application, the flow is defined as a single navigation rule connecting two views (presentation files). At this point, we will create the placeholders for the two JSP presentation files and then the navigation rule to connect them as views. Later, we will complete the coding for the JSP presentation files. With Red Hat Developer Studio, we can do all of this in the Diagram mode of the configuration file editor.

2.4.1. Adding Two Views (JSP Pages)

10. Right-click anywhere on the diagram and select New View... from the pop-up menu
11. In the dialog box, type pages/inputname as the value for From-view-id
12. Leave everything else as is
13. Click Finish.

    If you look in the Package Explorer view you should see a pages folder under WebContent. Opening it will reveal the JSP file you just created
14. Back on the diagram, right-click anywhere and select New View... from the popup menu
15. In the dialog box, type pages/greeting as the value for From-view-id
16. Leave everything else as is
17. Click Finish

2.4.1.1. Creating the Transition (Navigation Rule)

18. In the diagram, select the connection icon third from the top along the upper left side of the diagram

Figure 2.2.

to get an arrow cursor with a two-pronged plug at the arrow's bottom.

19. Click on the pages/inputname page icon and then click on the pages/greeting page icon

A transition should appear between the two icons.

Figure 2.3.

20. Select File/Save from the menu bar.

2.5. Adding a Managed Bean to the Application

To store data in the application, we will use a managed bean.

21. Click on the Tree tab at the bottom of the editing window
22. Select the Managed Beans node and then click the Add... button displayed along the right side of the editor window
23. Type in jsfHello.PersonBean for Class and personBean for Name. Leave Scope as is and Generate Source Code as is (checked)
24. Click Finish
25. personBean will now be selected and three sections of information, Managed Bean, Properties, and Advanced, will be displayed about it. Under the Properties section, click the Add... button
26. For Property-Name type in name. Leave everything else as is. (When Property- Class is not filled in, String is the assumed type.)
27. Click Finish
28. Select the personBean node in the tree

You should see this now:

Figure 2.4.

29. Select File/Save from the menu bar.

You have now registered the managed bean and created a stub-coded class file for it.

2.6. Editing the JSP View Files

Now we will finish editing the JSP files for our two "views" using Exadel's JSP Visual Page

2.6.1. inputname.jsp

30. Click on the Diagram tab for the configuration file editor
31. Open the editor for this first JSP file by double-clicking on the /pages/inputname. jsp icon

The Visual Page Editor will open in a screen split between source code along the top and a WYSIWIG view along the bottom:

Figure 2.5.

Some JSF code will already be in the file because we selected a template when creating the page.

32. Select the Visual tab, so we can work with the editor completely in its WYSIWYG mode

33. To the right of the editor, in the Exadel Palette, expand the JSF HTML palette folder by selecting it

Figure 2.6.

34. 34. Click on form within this folder, drag the cursor over to the editor, and drop it inside the red box in the editor

35. Another red box will appear inside the first red box.

36. Right-click on the innermost box and select <h:form> Attributes from the menu

37. In the value field next to id, type greeting and click on the Close button

38. Type Please enter name: inside the boxes

39. Select inputText within the JSF HTML palette folder and drag it into the innermost box in the editor after "Please enter name:"

40. In the attributes dialog, click in the value field next to the value attribute and click on the ... button

41. Then, select the Managed Beans/personBean/name node and click on the Ok button

42. Back in the attributes dialog, select the Advanced tab, type in name as the value for the id attribute, and then click on the Finish button

43. Select commandButton within the JSF HTML palette folder and drag it into the innermost box in the editor after the input box

44. In the attributes dialog, click in the value field next to the action attribute and click on the ... button

45. Then, select the View Actions/greeting node and click on the Ok button

46. Back in the attributes dialog box, type in Say Hello as the value for the value attribute ("Say Hello") and then click on the Finish button

The source coding should be something like this now:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

```
<html>
<head>
<title></title>
</head>
<body>
<f:view>
<h:form id="greeting">
<para>Please enter a name:</para>
<h:inputText id="name" value="#{personBean.name}"/>
<h:commandButton value=" Say Hello " action="greeting"/>
</h:form>
</f:view>
</body>
</html>
```

The editor should look like this:

Figure 2.7.

47. Save the file by selecting File/Save from the menu bar.

2.6.2. greeting.jsp

48. Click on the faces-config.xml tab to bring the diagram back
49. Open the editor for the second file by double-clicking on the /pages/greeting.jsp icon
50. Select the Visual tab, so we can work with the editor completely in its WYSIWYG mode
51. Type Hello (note space after hello) into the box
52. Select outputText within the JSF HTML palette folder and drag it into the innermost box in the editor after "Hello"
53. In the attributes dialog, click in value field next to the value attribute and click on the ... button
54. Then, select the Managed Beans/personBean/name node, click on the Ok button, and then click on the Finish button
55. Right after the output field, type an exclamation point (!)

The source coding should be something like this now:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
<head>
<title></title>
</head>
<body>
<f:view>
Hello <h:outputText value="#{personBean.name}"/>!
</f:view>
```

```
</body>
</html>
```

56. Save the file.

2.7. Creating the Start Page

You also need to create a start page as an entry point into the application.

57. In the Package Explorer view to the left, right-click jsfHello/WebContent and select New/JSP File
58. For Name type in index, for Template select JSPRedirect and click Finish.

A JSP editor will open up on the newly created file.

59. In the Source part of the split screen, type /pages/inputname.jsf in between the quotes for the page attribute

The source coding should look like this now:

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head></head>
<body>
<jsp:forward page="/pages/inputname.jsf" />
</body>
</html>
```

Note the .jsf extension for the file name. This is a mapping defined in the web.xml file for the project for invoking JavaServer Faces when you run the application.

60. Select File/Save from the menu bar.

2.8. Running the Application

Everything is now ready for running our application without having to leave Red Hat Developer Studio by using the Tomcat engine that comes with the Red Hat Developer Studio plug-in. For controlling Tomcat within Red Hat Developer Studio, the toolbar contains a special panel

Figure 2.8.

61. Start up Tomcat by clicking on the first icon from left. (If Tomcat is already running, stop it by clicking on the third icon from the left and then start it again. Remember, the JSF run-time requires restarting the servlet engine when any changes have been made.) After the messages in the Console tabbed view stop scrolling, Tomcat is available.

62. Click on the Red Hat run icon in the toolbar:

Figure 2.9.

This is the equivalent of launching the browser and typing http://localhost:8080/jsfHello into your browser. Our JSF application should now appear.

3 Getting Started Guide for Creating a Struts Application

We are going to show you how to create a simple Struts application using the Red Hat Developer Studio plug-in for Eclipse. The completed application will ask a user to enter a name and click a button. The resulting new page will display the familiar message, Hello <name>!

This document will show you how to create such an application from the beginning, along the way demonstrating some of the powerful features of Red Hat Developer Studio. You will design the application, generate stub code for the application, fill in the stub coding, compile the application, and run the application all from inside Red Hat Developer Studio.

Well assume that you have already launched Eclipse with Exadel Studio or Exadel Studio Pro installed and also that the Red Hat Developer Studio perspective is the current perspective. (If not, make it active by selecting Window/Open Perspective/Red Hat Developer Studio from the menu bar.)

3.1. Starting Up

We are first going to create a new project for the application.

1. Go to the menu bar and select File/New/Project... .
2. Select Exadel Studio/Struts/Struts Project in the New Project dialog box.
3. Click Next >.
4. Enter StrutsHello as the project name.
5. Leave everything else as is, and click Next >.
6. Click Next> again.
7. 7. Make sure that struts-bean.tld, struts-html.tld, and struts-logic.tld are checked in the list of included tag libraries and then click Finish.

A StrutsHello node should appear in the upper-left Package Explorer view.

8. Click the plus sign next to StrutsHello to reveal the child nodes.
9. Click the plus sign next to WebContent under StrutsHello.

10. Click the plus sign next to WEB-INF under WebContent.
11. Then, double-click on the struts-config.xml node to display a diagram of the Struts application configuration file in the editing area.

At this point, its empty except for the background grid lines.

3.2. Creating the Application Components

Now, we will design the application by creating the individual components as placeholders first. (We dont have to complete all of the details inside the components until afterwards.)

3.2.1. Creating JSP Page Placeholders

Next, lets create and place two JSP pages. We will not write any code for the files, but only create them as placeholders so that we can create links to them in the diagram. We will write the code a little bit later.

3.2.1.1. Creating the Page Placeholders

12. Bring the Web Projects view to the front of the Package Explorer view by selecting the Web Projects tab next to that tab.
13. Right-click the StrutsHello/WEB-ROOT (WebContent) folder in the Web Projects view and select New/ Folder... .
14. Enter pages for a folder name and click Finish.
15. We will keep our presentation files in this folder.
16. Right-click the pages folder and select New/File/JSP... .
17. 16. For Name type in inputname (the JSP extension will be automatically added to the file), for Template select StrutsForm, and then click on the Finish button.
18. Right-click the pages folder again and select New/File/JSP... .
19. For Name type in greeting , for Template leave as Blank, and then click on the Finish button.

Just leave these files as is for now.

3.2.1.2. Placing the Page Placeholders

Lets now place the two pages just created on the diagram.

20. Click on the struts-config.xml tab in the Editing area to bring the diagram to the front.
21. Click on the inputname.jsp page in the Web Projects view, drag it onto the diagram, and drop it.

22. Click on the greeting.jsp page in the Web Projects view, drag it onto the diagram, and drop it to the right of the /pages/inputname.jsp icon with some extra space.

You should now have two JSP pages in the diagram.

3.2.2. Creating an Action Mappings

Using a context menu on the diagram, we are next going to create an Action mapping.

23. Right-click between the two icons and select Add/Action
24. Enter the following values:

Table 3.1.

(GetNameForm is the name for a form bean that we will create later.)

25. Click Finish.

The /greeting action should appear in two places, in the diagram and also under the action-mappings node under the struts-config.xml node in the Outline view. Also, note the asterisk to the right of the name, struts-config.xml, in the Outline view showing that the file has been changed, but not saved to disk.

3.2.3. Creating a Link

Lets now create a link from the inputname.jsp page to the action.

26. On the left-hand side of the diagram in the column of icons, click on this icon:
27. 26. In the connect-the-components mode you are in now, click on the /pages/inputname.jsp icon in the diagram and then click on the /greeting action.

A link will be created from the page to the action.

3.2.4. Creating a Forward

Next, we are going to create a forward for the action.

28. On the left-hand side of the diagram in the column of icons, click on this icon, again:
29. Click on the /greeting action icon in the diagram and then click on the /pages/greeting.jsp icon.
30. Thats it. A link will be drawn from the actions new greeting forward to the greeting.jsp JSP page. Note that the forwards name will be set based on the name of the target JSP file name. If you dont like it, you can easily change it.

31. Select the Tree tab at the bottom of the editor window (between Diagram and Source).

32. Expand the struts-config.xml/action-mappings//greeting node and then select the greeting forward.

33. In the Properties Editor to the right, change the text to sayHello in the Name field.

34. Select the Diagram tab at the bottom of the editor window and see how the diagram is also updated to reflect the change.

3.2.5. Creating a Global Forward

One last component that we need to create in the diagram is a global forward.

35. Somewhere in the top-left corner of diagram, right-click and select Add/Global Forward....

36. Enter getName in the Name field.

37. Select the Change... button for Path.

38. In the Edit Path window, switch to the Pages tab.

39. Expand the StrutsHello/WEB-ROOT (WebContent)/pages node and then select the inputname.jsp page.

40. Click Ok.

41. Leave the rest of the fields blank and click Ok.

A forward object now appears on the diagram and also in the global-forwards folder in the Outline view.

42. Tidy up the diagram, by clicking and dragging around each icon, so that the diagram looks something like this:

3.2.6. Creating a Form Bean

One last thing that we need to do is to create a form bean.

43. Switch to the Tree viewer in the editor for the struts-config.xml file, by selecting the Tree tab at the bottom of the editor window.

44. Right-click struts-config.xml/form-beans and select Create Form Bean.

45. Enter GetNameForm in the name field and sample.GetNameForm for type.

46. Click Finish.

47. To save your changes to struts-config.xml, select File/Save from the menu bar.

Note the disappearance of the asterisk next to the name, struts-config.xml.

3.3. Generating Stub Coding

We are done with designing the application through the diagram. Now we need to write code for the action component. We also need to write an action class for the /greeting mapping along with a FormBean. To aid in the coding phase, Exadel Studio can generate Java class stubs for all of the components shown in the diagram.

48. Switch back to the diagram, by selecting the Diagram tab at the bottom of the editor window.

49. Right-click a blank space in the diagram and select Generate Java Code.

50. Leave everything as is in the dialog box and click Generate .

You should see a screen that says:

Generated classes: 2

Actions: 1

Form beans: 1

51. Click Finish.

The Java files will be generated in a JavaSource/sample folder that you can see in the Package Explorer view under the StrutsHello node. One Action stub and one FormBean stub will have been generated.

3.4. Coding the Various Files

We will now code both the Java stub classes just generated, the JSP files left in as placeholders from previous steps, and a new start JSP page we will have to create.

3.4.1. Java Stub Classes

52. To finish the two Java classes, switch to the Package Explorer view and expand the JavaSource/sample folder.

3.4.1.1. GetNameForm.java

53. Double-click GetNameForm.java for editing.

54. You are looking at a Java stub class that was generated by Exadel Studio. Now we are going to edit the file.

55. Add the following attributes at the beginning of the class:

private String name = "";

private String greetName = "";

56. Inside the reset method, delete the TO DO and throw lines and add:

this.name = "";

this.greetName = "";

57. Inside the validate method, delete the TO DO and throw lines and add:

ActionErrors errors = new ActionErrors();

return errors;

58. Right-click and select Source/Generate Getters and Setters... from the context menu.

59. In the dialog box, check the check boxes for name and greetName, select First method for Insertion point, and click on the OK button.

The final GetNameForm.java file should look like this:

```
package sample;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
public class GetNameForm extends org.apache.struts.action.ActionForm {
private String name = "";
private String greetName = "";
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public String getGreetName() {
return greetName;
}
public void setGreetName(String greetName) {
this.greetName = greetName;
}
public GetNameForm() {
}
public void reset(ActionMapping actionMapping, HttpServletRequest
request) {
this.name = "";
this.greetName = "";
}
public ActionErrors validate(ActionMapping actionMapping,
Exadel Studio: Getting Started Guide for Creating a Struts Application
page 7 of 11
HttpServletRequest request) {
ActionErrors errors = new ActionErrors();
return errors;
}
}
```

60. Save the file.

3.4.1.2. GreetingAction.java

61. Open GreetingAction.java for editing.
62. Inside the execute method, delete the TO DO and throw lines and add the following:

```
String name = ((GetNameForm)form).getName();
String greeting = "Hello, "+name+"!";
((GetNameForm)form).setGreetName(greeting);
return mapping.findForward(FORWARD_sayHello);
The final version of GreetingAction.java should look like this:
package sample;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
public class GreetingAction extends org.apache.struts.action.Action {
// Global Forwards
public static final String GLOBAL_FORWARD_getName = "getName";
// Local Forwards
public static final String FORWARD_sayHello = "sayHello";
public GreetingAction() {
}
public ActionForward execute(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws
Exception {
String name = ((GetNameForm)form).getName();
String greeting = "Hello, "+name+"!";
((GetNameForm)form).setGreetName(greeting);
return mapping.findForward(FORWARD_sayHello);
}
}
```

63. Save the file.
64. Close the editors for the two Java files.

The last thing left to do is to code the JSP files whose editors should still be open from having been created as placeholders.

3.4.2. JSP Pages

3.4.2.1. inputname.jsp

In this page, the user will enter any name and click the submit button. Then, the greeting action will be called through the form.

65. Click on the inputname.jsp tab in the Editing area to bring its editor forward.
66. In the Web Projects view, expand StrutsHello/Configuration/default/strutsconfig.xml/action-mappings and select /greeting.

67. Drag it and drop it between the quotes for the action attribute to the html:form element in the Source pane of the editor.
68. Then type this text on a new line just below this line:
69. Input name:
70. Select the Visual pane of the editor.
71. Then, in the Exadel Palette, expand the Struts Form library, select text, and drag it onto the box.
72. In the Insert Tag dialog box, type in name for property and select Finish.
73. In the Struts Form library in the Exadel Palette, select submit, and drag it to right after the the text box in the Visual pane of the editor.
74. Right-click the submit button and select <html:submit> Attributes from the context menu.
75. In the Attributes dialog box, select the value field and type in Say Hello! for its value.

After tidying the page source, the Editor window for the file should look something like this:

3.4.2.2. greeting.jsp

Next, we will fill in the result page.

76. Click on the greeting.jsp tab in the Editing area to bring its editor forward.
77. Type in the following code:

```
<html>
<head>
<title>Greeting</title>
</head>
<body>
<p>
</p>
</body>
</html>
```

To complete editing of this file, we will use macros from the Exadel Palette. This palette is a view that should be available to the right of the editing area.

78. Click on the Struts Common folder in the Exadel Palette to open it.
79. Position the cursor at the beginning of the greeting.jsp file in the Source pane and then click on bean taglib in the Exadel Palette.

This will insert the following line at the top of the file:

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

80. Click on the Struts Bean folder in the Exadel Palette to open it.

81. Position the cursor inside the p element.

82. Click on write in the Exadel Palette.

83. Type in GetNameForm for the name attribute and add a property attribute with greetName as its value.

The editor should should now look like this:

3.4.2.3. index.jsp

Finally, we will need to create and edit an index.jsp page. This page will use a Struts forward to simply redirect us to the getName global forward.

84. In the Web Projects view, right-click on StrutsHello/WEB-ROOT(WebContent)

    node and select New/File/JSP... .

85. Type index for Name and click on the Finish button.

86. On the Exadel Palette, select the Struts Common folder of macros by clicking on it in the palette.

87. Click on the logic taglib icon.

88. Press the Enter key in the editor to go to the next line.

89. Back on the palette, select the Struts Logic folder of macros.

90. Click on redirect.

91. Delete the ending tag, put a forward slash in front of the closing angle bracket, and type forward=getName in front of the slash.

The finished code for the page is shown below:

```
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<logic:redirect forward="getName"/>
```

92. To save all the edits to files, select File/Save All from the menu bar.

3.5. Compiling the Classes

Because this is the Eclipse environment, no explicit compilation step is required. By default, Eclipse compiles as you go.

3.6. Running the Application

Everything is now ready for running our applicationwithout having to leave Red Hat Developer Studio by using the Tomcat engine that comes with the Red Hat Developer Studio plug-in. For controlling Tomcat within Red Hat Developer Studio, the toolbar contains a panel.

93. Start up Tomcat by clicking on the first icon from left in this panel. (If Tomcat is already running, stop it by clicking on the third icon from the left and then start it again. Remember, the Struts run-time requires restarting the servlet engine when any changes have been made.)

94. After the messages in the Console tabbed view stop scrolling, Tomcat is available. At this point, right-click on the getName global forward in the struts-config.xml diagram view and select Run on Server.

The browser should appear with the application started.

4

Getting Started Struts Validation Examples

Validation of input is an important part of any Web application. All Apache Jakarta frameworks, including Struts, can use a common Jakarta Validation Framework for streamlining this aspect of Web application development. The Validation Framework allows the developer to define validation rules and then apply these rules on the client-side or the server-side.

Red Hat Developer Studio makes using the Validation Framework in Struts even easier through a specialized editor for the XML files that control validation in a project. In this document, we'll show you how this all works by creating some simple client-side validation and server-side validation examples.

4.1. Starting Point

The example assumes that you have already created our sample StrutsHello application from the Getting Started Guide for Creating a Struts Application. You should have the Red Hat Developer Studio perspective open on this StrutsHello project.

4.2. Defining the Validation Rule

In these steps you will set up the validation that can be used for either client-side or serverside validation. You need to enable validation as part of the project, define an error message, and tie it into the appropriate part of the application.

1. Right-click the plug-ins node under the StrutsHello/Configuration/default/ struts-config.xml node in the Web Projects view and select Create Special Plugin/Validators from the context menu.
2. Further down in the Web Projects view, right-click on the StrutsHello/ResourceBundles node and select New/ Properties File... from the context menu.
3. In the dialog box, click on the Browse... button next to the Folder field, expand the JavaSource folder in this next dialog box, select the sample subfolder, and click on the OK button.
4. Back in the first dialog box, type in applResources for the Name field and click on the Finish button.
5. Right-click the newly created file and select Add/Default Error Messages from the context menu.
6. Drag up the sample.applResources icon until you can drop it on the resources folder under struts-config.xml.
7. Select File/Save All from the menu bar.
8. Select validation.xml under the StrutsHello/Validation node and double-click it to open it with the Exadel Stu-

dio Validation Editor.

9. Expand the form-beans node under the StrutsHello/Configuration/default/ struts-config.xml node. Then, drag the form bean GetNameForm and drop it onto formset (default) in the Validation Editor.

10. In the Validation Editor, expand the formset node, right-click GetNameForm, and select Add Field... from the context menu.

11. Enter name for Property in the dialog box.

12. In the properties for the name field to the right of the "tree" for the validation.xml file, click on the Change... button next to the Depends entry field.

13. In the displayed double list, select required from the left list and then click Add->.

14. Click Ok.

15. Right-click name and select Add Arg... from the context menu.

16. In the Add Arg dialog box, click on the Change... button next to the Key field.

17. In the Key dialog box that appears now, click on the Add button.

18. Enter name.required in the Name field, and enter A person's name in the Value field.

19. Click Finish, then Ok, and then Ok again.

20. Select File/Save All from the menu bar.

4.3. Client-Side Validation

Client-side validation uses a scripting language (like JavaScript) running in the client browser

to actually do the validation. In a Struts application using the Validation Framework, however,

you don't actually have to do any of the script coding. The Validation Framework handles this.

To see how this works in our application, you'll just need to make a couple of modifications to one of the JSP files.

21. Double-click inputname.jsp under StrutsHello/WEB-ROOT (WebContent)/ pages to open it for editing.

22. Find the tag near the top and hit Return to make a new line under it.

23. In the Red Hat Palette view to the right, open the HTML folder and click on the javascript tag.

24. Back in the editor, just in front of the closing slash for this inserted tag, hit Ctrlspace and select formName from the prompting menu.

25. Over in the Web Projects view, select GetNameForm under the StrutsHello/Configuration/ default/ struts-config.xml/form-beans node, drag it, and drop it between the quotes in the editor.

26. Modify the >html:form< tag by inserting this attribute:

onsubmit="return validateGetNameForm(this)"

The file should now look like this:

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html:html>
<head>
<title>Input name</title>
<html:javascript formName="GetNameForm"/>
</head>
<body>
<html:form action="/greeting.do" onsubmit="return
<para>validateGetNameForm(this)"></para>
<table border="0" cellspacing="0" cellpadding="0">
<tr>
<td><b>Input name:</b></td>
</tr>
<tr>
<td>
<html:text property="name" />
<html:submit value=" Say Hello! " />
</td>
</tr>
</table>
</html:form>
</body>
</html:html>
```

27. Select File/Save from the menu bar.

28. 28. Start Tomcat by clicking on its icon (a right-pointing arrow) in the toolbar.

29. 29. Click on the Run icon in the toolbar.

Figure 4.1.

- 30. In the browser window, click on the Say Hello! button without having entered any name in the form.

A JavaScript error message should be displayed in an alert box.

4.4. Server-Side Validation

Server-side validation does the validation inside the application on the server. In a Struts application using the Validation Framework, you still don't have to do any of the actual validation coding. The Validation Framework handles this. You will, though, have to make a few changes to the JSP file you modified for client-side validation along with a change to an action and a few changes to the form bean class.

4.5. Editing the JSP File

30. Reopen inputname.jsp for editing.

31. Delete the onsubmit attribute in the >html:form< element that you put in for client-side validation.

32. Add an >html:errors/> tag after the >/html:form> tag.

The JSP file should now look like this:

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html:html>
<head>
<title>Input name</title>
<html:javascript formName="GetNameForm"/>
</head>
<body>
<html:form action="/greeting.do">
<table border="0" cellspacing="0" cellpadding="0">
<tr>
<td><b>Input name:</b></td>
</tr>
<tr>
<td>
<html:text property="name" />
<html:submit value=" Say Hello! " />
</td>
</tr>
</table>
</html:form>
<html:errors />
</body>
</html:html>
```

4.6. Editing the Action

33. In the Web Projects view, expand the node under the StrutsHello/Configuration/ default/ struts-config.xml/action-mappings node, right-click the /greeting action, and then select Properties... from the context menu.

34. In the Edit Properties window, insert the cursor into the value column for the input property and click on the ... button.

35. In the dialog box, make sure the Pages tab is selected, select StrutsHello/WEBROOT(WebContent)/pages/ inputname.jsp, click the Ok button, and then click on the Close button.

4.7. Editing the Form Bean

36. Right-click the /greeting action again and select Open Form-bean Source to open the GetNameForm.java file for editing.

37. Change the class that it extends to from: org.apache.struts.action.ActionForm to: org.apache.struts.validator.ValidatorForm

38. Comment out out the validate method.

The file should now look like this:

```
package sample;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
public class GetNameForm extends
org.apache.struts.validator.ValidatorForm {
private String name = "";
/**
* @return Returns the name.
*/
public String getName() {
return name;
}
/**
* @param name The name to set.
*/
public void setName(String name) {
this.name = name;
}
public GetNameForm () {
}
public void reset(ActionMapping actionMapping, HttpServletRequest
request) {
this.name = "";
}
// public ActionErrors validate(ActionMapping actionMapping,
HttpServletRequest request) {
// ActionErrors errors = new ActionErrors();
// return errors;
// }
}
```

Select File/Save All from the menu bar.

39. Reload the application into Tomcat by clicking on the Change Time Stamp icon (a finger pointing with a little star) in the toolbar.

40. Run the application.

41. In the browser window, click on the Say Hello! button without having entered any name in the form.

The error message should appear in a refreshed version of the form.

5 Getting Started Guide for JSF with Hibernate

In this guide, we will show you how to take a very simple ready-made JSF application and convert it to use a database with the help of Exadel Studio. After downloading, we will first

set up and run the application without persistence, and then with persistence.

The application itself is a simple JSF-based application that asks the user to enter a UserID. It tries to locate a record for the entered User ID (entered during the application session).

If the record is found, details are displayed. If the record is not found, you are asked to create this record. This application, of course, only runs as long as the Tomcat server is running. Once we stop the server all of the data is lost as all information is saved only in the application session context.

With the help of Exadel Studio, we will convert this application to use the lightweight hsqldb database (included with the downloaded project). We will use Exadel Studio special features for object-relational mapping for this conversion After the conversion, even if we restart the server, the data we entered will have been saved in a database and thus available to the application.

Before we start, we assume that you have Eclipse, and have installed Exadel Studio Pro with Tomcat server.

5.1. Installing the Project

We are first going to download and import this project (ormHibernate3-jsf) into Eclipse.

1. Download: http://webdownload.exadel.com/dirdownloads/ormhib/examples/ormHibernate3-jsf.zip
2. Unzip this file into your Eclipse workspace folder.
3. Launch Eclipse.
4. In Eclipse, select File/Import/JSF Project.
5. Click Next.
6. Browse to where the project was unzipped.
7. Find the web.xml file inside the WebContent folder in the WEB-INF folder, select it, and Click Finish.

The ormHibernate3-jsf project should appear in the Package Explorer with a standard Web application structure. As we mentioned before, this is a JSF application. To see the JSF configuration file, browse to WebContent/WEB-INF/faces-config.xml.

In the JavaSource folder, you will find the following Java source files. We will briefly explain these files and then run the application.

Table 5.1. JavaSource Folder

Class Name	Description
demo/Address.java	holds the user address
demo/User.java	holds user information
demo/GetUserIdBean.java	holds user Id and determines navigation (JSF Backing Bean)
demo/UserFormBean.java	holds new user input (JSF Backing Bean)
demo.bundle/Messages.properties	holds label messages used in the application

5.2. Running Without a Database

We are ready to run this project in a Web browser and see how it looks. We don't need to compile these classes, because Eclipse did it for us when we imported the project.

8. Start Tomcat.
9. Click on the running-man-and-blue-butterfly icon from the toolbar.
10. Go ahead and play with the application.

Initially users don't exist, so entering any ID will prompt you to enter user details. Once you have saved a user, you can go back to the main page by clicking on the Back to Login Page link. If you then enter that user's id again, the application will locate and display the user's details.

5.3. Converting for Use With a Database

Now we are ready to convert this application to use with a database with the help of ExadelStudio. To convert the application for use with a database, we need to set things up in three different places:

- The Application
- The Database
- The Application Server

5.4. Setting up the Application

Setting up the Application

Let's start by using Red Hat Developer Studio with the application project. First, we create the object/relational mapping from our simple object model to a database schema after adding Hibernate capabilities to our project.

11. Right-click on ormHibernate3-jsf in the Package Explorer view and select Exadel
12. Studio/Add Hibernate Capability... from the context menu.
13. Click on Yes in the the dialog box with Add Hibernate Jars selected.
14. In the Configuration Wizard, click twice in the Value field for dialect and select org.hibernate.dialect.HSQLDialect from the pop-up menu.
15. Click Finish
16. Select Object to Schema for the Mapping Approach.

We are only interested in saving the User class in a database, so we are going to create a mapping for the User class to a database table.

17. In the Persistent Classes Wizard dialog that appears next, click on the SelectClasses.
18. Leave all other values as they are in the next dialog box and click Finish.

5.5. Edit the Hibernate Configuration

Afterwards, the Hibernate configuration file, hibernate.cfg.xml, will appear in an editor window. So, let's adjust this file first.

19. Replace these two lines:

```
<property
name="hibernate.connection.driver_class">org.hsqldb.jdbcDriver</
property>
<property
name="hibernate.connection.url">jdbc:hsqldb:hsql:[hostname]</
property>
```

With this single line:

```

<property name="hibernate.connection.datasource">java:comp/env/
jdbc/kickstart</property>
```

(Make sure this is one line in the file and not two lines as displayed because of wordwrapping.)

Your file should now look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.datasource">java:comp/env/jdbc/
kickstart</property>
Exadel Studio 3.0
page 4 of 10
<property name="hibernate.dialect">org.hibernate.dialect.HSQLDialect</
property>
<mapping resource="demo/User.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

20. Save the file.

5.6. Edit the Mapping File

Next, we need to make on slight change to the mapping file, User.hbm.xml.

21. In the ORM Explorer view, reveal the ormHibernate3-jsf/JavaSource/hibernate.cfg.xml/demo/User -> user node, right-click it, and select Open Mapping from the context menu.

22. In the editor that opens up for the mapping file, just change the class attribute for generator to a value of assigned and you're done with this file.

Here is what the edited User.hbm.xml file should now look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
<para>"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"></para>
<hibernate-mapping package="demo">
<class name="User" table="user" optimistic-lock="none">
<id name="id" type="string" unsaved-value="null" column="id">
<generator class="assigned"/>
</id>
<property name="firstName" type="string" column="first_name"/>
<property name="lastName" type="string" column="last_name"/>
<property name="email" type="string" column="email"/>
<component name="address" update="true" insert="true" class="demo.Address">
<property name="city" type="string" column="address_city"/>
<property name="state" type="string" column="address_state"/>
<property name="street" type="string" column="address_street"/>
<property name="zip" type="string" column="address_zip"/>
</component>
</class>
</hibernate-mapping>
```

23. Save the file.

5.7. Add a General Class for Incorporating Hibernate

Next, we will need to create a special Java class for incorprating Hibernate into our application.

24. Switch to the Package Explorer view and create the class, HibernateHelper.java, in JavaSource/demo with this content and save it.

```
package demo;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateHelper {
/**
* Reference to SessionFactory.
*/
private static SessionFactory sf;
public static final ThreadLocal session = new ThreadLocal();
public static synchronized void init() {
if (sf != null) return;
System.out.println("Initializing Hibernate");
try {
Configuration cfg = new Configuration().configure();
sf = cfg.buildSessionFactory();
} catch (Exception he) {
System.err.println("Unable to create session factory from
configuration");
he.printStackTrace();
throw new RuntimeException("Unable to create session factory from
configuration", he);
}
System.out.println("Hibernate initialized");
}
/**
* Return the SessionFactory.
* @return The SessionFactory for this application session
*/
public static SessionFactory sessionFactory() {
if (sf == null) init();
return sf;
}
public static void destroy() {
if (sf != null) {
try {
sf.close();
} catch (HibernateException he) {
he.printStackTrace();
}
}
sf = null;
System.out.println("Hibernate resources released");
}
/**
* Closes an hibernate {@link Session}, releasing its resources.
* @throws HibernateException if an hibernate error occurs
*/
public static void closeSession() throws HibernateException {
Session s = (Session)session.get();
session.set(null);
if (s != null) {
s.close();
Exadel Studio Pro
page 6 of 9
}
}
/**
* Returns an hibernate {@link Session} from the session factory.
* @return an hibernate {@link Session}
* @throws HibernateException if an error occurs
*/
public static Session openSession() throws HibernateException {
if (sf == null) init();
Session s = (Session)session.get();
if (s == null) {
s = sf.openSession();
session.set(s);
}
return (s);
}
}
```

5.8. Edit the Two Bean Classes

We also need to modify the two bean classes in our application to "Hibernate-ize" them.

25. Modify the GetUserIdBean.java class in JavaSource/demo by adding these imports:

```
import org.hibernate.Session;
import javax.faces.application.FacesMessage;
```

26. Then, replace the action() method with this code and save.

```
{
public String action()
throws Exception
UserFormBean ufb;
User user;
String actionResult = "inputuser"; // new user by default
Map sessionMap =
FacesContext.getCurrentInstance().getExternalContext().getSessionMap();
if (sessionMap != null) {
ufb = new UserFormBean();
ufb.setId(id);
try {
Session hSession = HibernateHelper.openSession();
user = (User)hSession.get(User.class, id);
HibernateHelper.closeSession();
} catch (Exception e) {
FacesContext context = FacesContext.getCurrentInstance();
context.addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_ERROR,e.toString(), null));
return "failed";
}
if (user==null || !user.getId().equals(id)) {
user = new User();
user.setId(id);
sessionMap.put("user", user);
} else {
// fill UserFormBean with user information
loadUser(ufb,user);
actionResult="greeting";
}
sessionMap.put("UserFormBean", ufb);
}
return actionResult;
}
private void loadUser(UserFormBean userForm,User user) {
userForm.setId(user.getId());
userForm.setFirstName(user.getFirstName());
userForm.setLastName(user.getLastName());
userForm.setEmail(user.getEmail());
userForm.setStreet(user.getAddress().getStreet());
userForm.setCity(user.getAddress().getCity());
userForm.setState(user.getAddress().getState());
userForm.setZip(user.getAddress().getZip());
}
```

27. Modify the UserFormBean.java class in JavaSource/demo by adding these imports:

```
import org.hibernate.Session;
import org.hibernate.Transaction;
```

28. Then, replace the save() method with this code and save the class.

```
public String save() throws Exception {
Map sessionMap =
FacesContext.getCurrentInstance().getExternalContext().getSessionMap();
if (sessionMap != null) {
UserFormBean ufb=(UserFormBean)sessionMap.get("UserFormBean");
try {
Session hSession = HibernateHelper.openSession();
Transaction tran = hSession.beginTransaction();
User user = (User)hSession.get(User.class, id);
if (user == null) {
user = new User();
user.setId(id);
hSession.save(user);
}
saveUser(ufb, user);
tran.commit();
HibernateHelper.closeSession();
} catch (Exception e) {
return "failed";
}
}
return "greeting";
}
```

5.9. Setting up the Database

To set up the database end, we need to use Red Hat Developer Studio and our HSQL database engine to create a database table corresponding to the class we are trying to persist and make the database available.

5.10. Creating the Database Table

Let's first create the script for our database table in Red Hat Developer Studio.

29. In the ORM Explorer view, right-click on JavaSource/hibernate.cfg.xml and select "Generate DDL Wizard".

30. Select HSQL as the Dialect and leave Location as is.

31. Click Finish.

A DDL file called schema.sql will be created in the root of the project and will be opened in an editor window.

5.11. Making the Database Available for the Application

The databse server, HSQLDB, is provided with the project. It's located in the ormHibernate3-jsf/hsqldb folder.

32. Start the database server: .../ormHibernate3-jsf/hsqldb/bin/server.bat

33. In a separate window, start the admin tool: .../ormHibernate3-jsf/hsqldb/bin/dbadmin.bat

This will launch a small GUI application, HSQL Database Manager.

34. Leave all values as they are, only change URL: to the following: jdbc:hsqldb:hsql://localhost

35. Click OK.

36. Select File/Open Script... from the menu bar of HSQL Database Manager.

37. Find and open the the DDL file we just created.

38. Click Execute back in the main screen of the Database Manager.

39. 38. Select View/Refresh Tree from the menu bar.

The User database should now appear in the expand/collapse tree to the left.

40. Select File/Exit from the menu bar.

41. Stop the database server: .../ormHibernate3-jsf/hsqldb/bin/shutdown.bat

5.12. Setting up the Application Server

Finally, we need to set up the application server before we can run the database-enabled application in a Web browser. To do this, we'll need to modify the application context in the- Tomcat server.xml file.

42. Stop the Tomcat server, if it's running.

43. Locate the server.xml file in the Package Explorer view under the Tomcat Server node under Servers.

44. Double-click the file to open an editor on it.

45. Find the Context tag for your application in the file. It will have a path attribute with a value of /ormHibernate3-jsf.

You'll need to convert this "empty" XML element into one with beginning and ending tags, so we can insert the special resource tags for Tomcat to run this application with a database.

46. Delete the closing slash at the end of the Context tag.

47. Insert a blank line after the tag and then start another line.

48. On this line, insert a closing tag:

```
</Context>
```

49. On the blank line between the starting and ending tags, add the following resource definition coding.

```
<Resource name="jdbc/kickstart" scope="Shareable"
type="javax.sql.DataSource"/>
<ResourceParams name="jdbc/kickstart">
<parameter>
<name>factory</name>
<value>org.apache.commons.dbcp.BasicDataSourceFactory</
value>
</parameter>
<parameter>
<name>url</name>
<value>jdbc:hsqldb:hsql://localhost</value>
</parameter>
<parameter>
<name>driverClassName</name>
<value>org.hsqldb.jdbcDriver</value>
</parameter>
<parameter>
<name>username</name>
<value>sa</value>
</parameter>
<parameter>
<name>password</name>
<value></value>
</parameter>
<parameter>
<name>maxWait</name>
<value>3000</value>
</parameter>
<parameter>
<name>maxIdle</name>
<value>100</value>
</parameter>
<parameter>
<name>maxActive</name>
<value>10</value>
</parameter>
</ResourceParams>
```

50. Finally, copy .../ormHibernate3-jsf/hsqldb/lib/hsqldb.jar to your Tomcat .../common/lib folder.

5.13. Running Our New Application

51. Start the database server: .../ormHibernate3-jsf/hsqldb/bin/server.bat
52. Start the Tomcat server.
53. Run the application.

Play with the application. Restart Tomcat and the database server. If you run the application again and enter a user that you already saved, the application should retrieve it from the database and display its details.