CREATOR BASICS

Topics in This Chapter

- Creator Window Layout
- Component Palette
- Source Editors/Code Completion
- Clips Palette
- Page Navigation Editor
- Application Outline Window
- Server Navigator Window
- Creator Help System

Chapter 2

Sun Java Studio Creator makes it easy to work with web applications from multiple points of view. This chapter explores some of Creator's basic capabilities, the different windows (views) and the way in which you use them to build your application. We show you how to manipulate your application through the drag-and-drop mechanism for placing components, setting attributes in the Properties window, controlling page flow with the Page Navigation editor, and selecting services from the Server Navigator window.

2.1 Examples Installation

We assume that you've successfully installed Creator. The best source of information on installing Creator is Sun's product information page at the following URL.

```
http://developers.sun.com/prodtech/javatools/jscreator/
```

Creator runs on a variety of platforms and can be configured with different application servers and JDBC database drivers. However, to run all our examples we've used the default application server (J2EE 1.4 Application Server) and PointBase database. Once you've configured Creator for your system, the examples you build in the text should run the same on your system.

Download Examples

You can download the examples for this book at

```
http://www.asgteach.com/download/index.htm
```

and select *Java Studio Creator Field Guide: Book Examples*. The examples are packed in a zip file. When you unzip the file, you'll see the **FieldGuide/Examples** directory and subdirectories for the various chapters and projects. As each chapter references the examples, you will be instructed on how to access the files.

You're now ready to start the tour of Creator.

2.2 Creator Views

Figure 2–1 shows Creator's initial window layout in its default configuration. When you first bring it up, no projects are open and Creator displays its Welcome window.

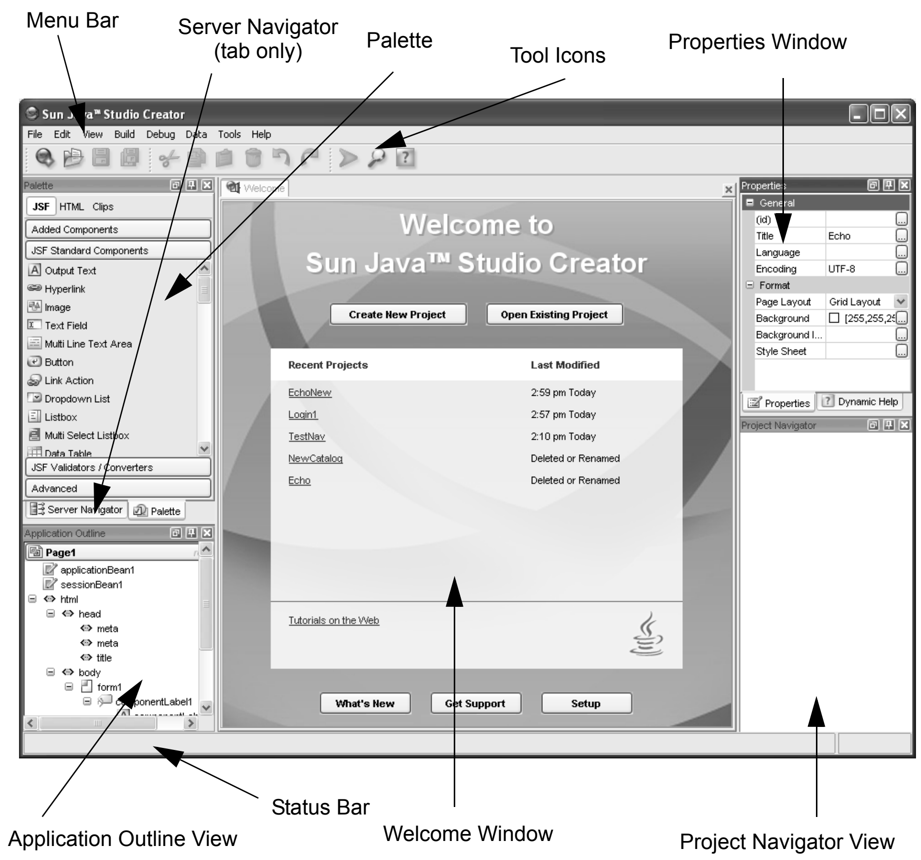
There are other windows besides those shown in the initial window layout. As you'll see, you can hide and display windows, as well as move them around. As we begin this tour of Creator, you'll probably want to run Creator along with the text.

Welcome

The Welcome window lets you to create new projects or work on existing ones. Figure 2–2 shows the Welcome window in more detail. It lists the projects you've worked on recently and offers selection buttons for opening existing projects or creating new projects.

To demonstrate Creator, let's use a project that we've already built with Creator. The project is in directory **FieldGuide/Examples/Projects/Login1**.

1. Select the Open an Existing Project button and browse to the **FieldGuide/Examples/Projects** directory for Java Creator Field Guide.
2. Select **Login1** (look for the projects icon) and double click. This opens the **Login1** project and displays the design canvas for the project's first page.



Recent Projects	Last Modified
EchoNew	2:59 pm Today
Login1	2:57 pm Today
TestNav	2:10 pm Today
NewCatalog	Deleted or Renamed
Echo	Deleted or Renamed

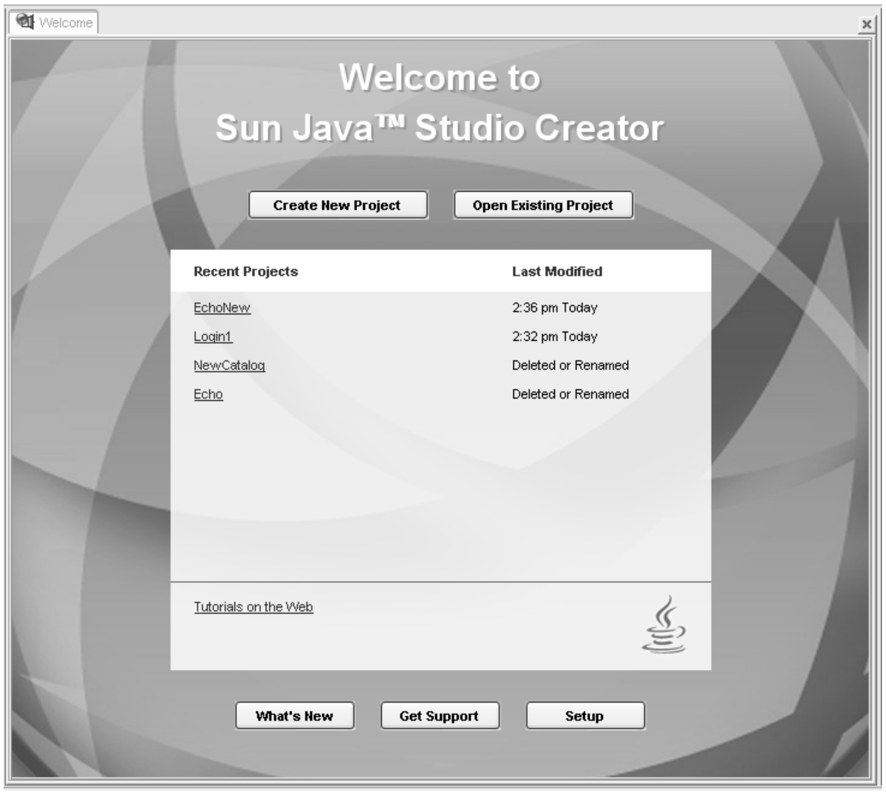
(id)	
Title	Echo
Language	
Encoding	UTF-8
Page Layout	Grid Layout
Background	[255,255,2
Background I...	
Style Sheet	

Figure 2–1 **Creator's initial window layout**

Design Canvas

Figure 2–3 shows a close-up of the design canvas window for project **Login1**. You see the design grid and the components we've placed on the canvas. As you select the individual components, their properties appear in the Properties window (not shown here).

Each project in Creator can have many files associated with it. Here, the design canvas window displays the design view of file **Page1.jsp**. You can have more than one of your project's files open at a time (currently, there's just one open). When you open other files, their file tab appears at the top of the editor pane. You use the File Tab to select other files.



Recent Projects	Last Modified
EchoNew	2:36 pm Today
Login1	2:32 pm Today
NewCatalog	Deleted or Renamed
Echo	Deleted or Renamed

Figure 2–2 **Creator's Welcome window**

The editor pane allows you to manipulate elements of your project. When the design canvas is showing, you can select components, validators, or converters from the palette. From the Server Navigator windows you can select nonvisual components, such as data sources or web services. In the design canvas, you see the components we've added to this page: a text field component, a secret field component, two component labels, two buttons, a message list component, and an output text component used as a page title.

Creator allows you to configure your display's workspace to suit the tasks you're working on. All the windows can be hidden when not needed (click the red X in a window's title bar to close it). To view the window again, select View from the menu bar and then the window name. You can dock Creator windows by selecting the pushpin in the window title bar. This action minimizes the window along the left or right side of the workspace. Make it visible again by

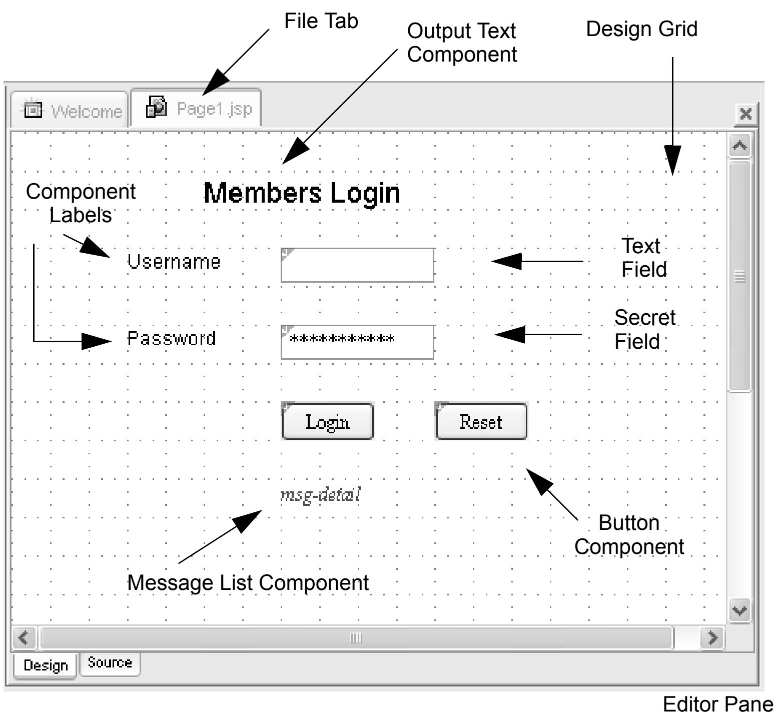


Figure **2–3 Creator's design canvas showing project Login1**

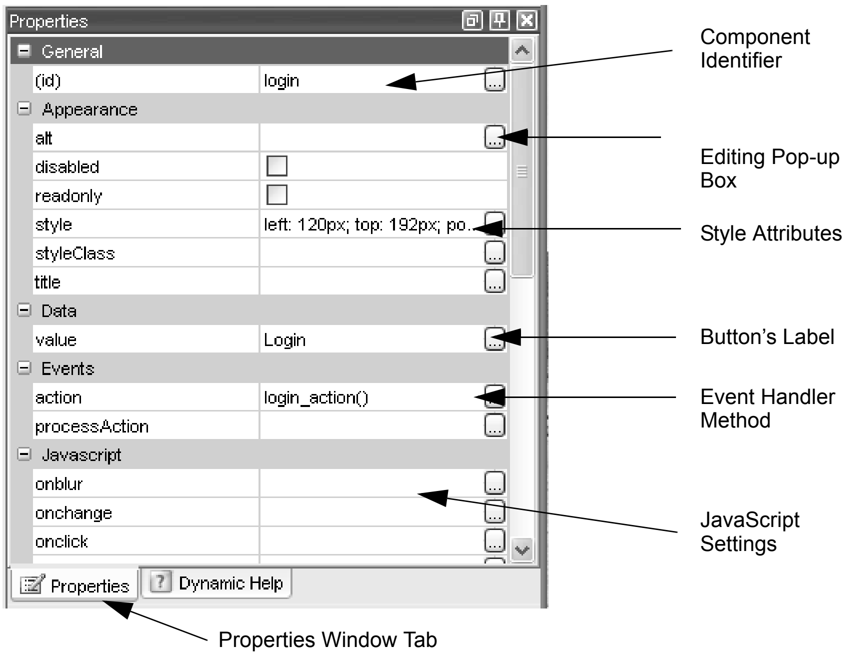
moving the cursor over its docked position. Undock it by toggling the pushpin icon.

Select the button labeled "Login" on the design canvas. This will bring up the button component's Properties window, as shown in Figure 2–4.

Properties

Creator lets you configure the components you use by manipulating their properties. When you change a component's properties, Creator automatically updates the JSF source for you. Here are the login button's properties.

The `id` attribute uniquely identifies the component on the page. Creator generates the name for you, but you may want to change it (as we have in this example) to more easily work with the generated code. You can use the `style` attribute to change its appearance. If you click in the editing box opposite `style`, you can see what it's set to. The `position` attribute reflects the component's position on the page. When you move the button component, Creator updates this for you.



General	
(id)	login
Appearance	
alt	
disabled	☐
readonly	☐
style	left: 120px; top: 192px; po...
styleClass	
title	
Data	
value	Login
Events	
action	login_action()
processAction	
Javascript	
onblur	
onchange	
onclick	

Figure 2–4 **Properties window for button component "Login"**

The `value` attribute under Data holds the button's label. Click on the Login button again in the design canvas and from the keyboard begin typing a new label. You'll see that Creator automatically selects the `value` attribute in the Properties window and enables editing as you begin typing. Finish editing with **<Enter>**.

Under Events, the `action` attribute is set to `login_action()`. This refers to an event handler method in the Java page bean that controls what happens when the user clicks the button.

Each component has a different list of attributes (although many attributes are the same). As you select other components in the design canvas, note that the list of attributes in the Properties window changes.

Palette

Figure 2–5 shows the JSF Standard Components window and JSF Validators/ Converters window. These are the palettes that let you select components and add them to your page. Page designers, for example, select a component and drag it to the design canvas, positioning it on the page. Once the component is

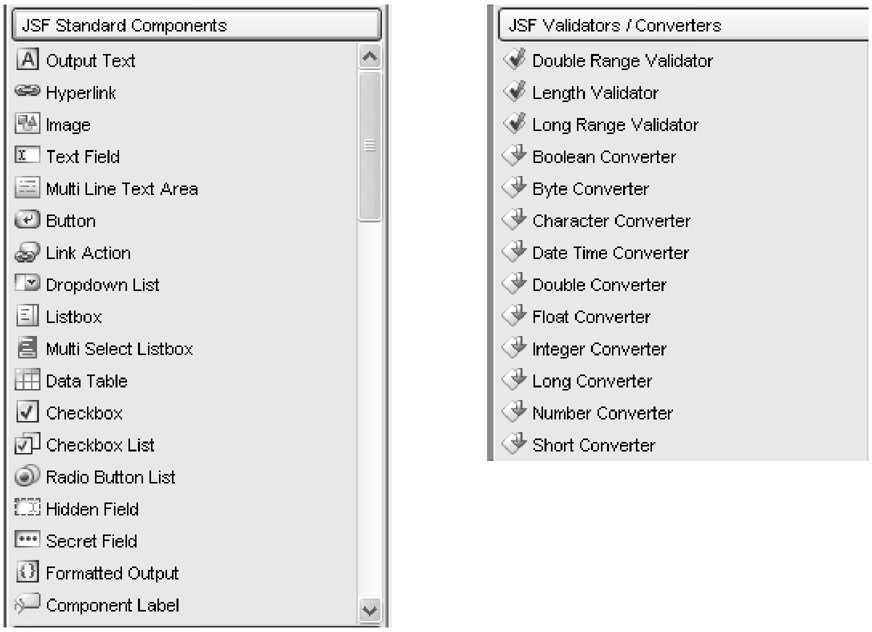


Figure 2–5 **JSF Standard Components and Validators/Converters windows**

on the page, you can configure it by changing the attributes in its Properties window.

Click on the bar labeled JSF Validators/Converters to see these components. You can select these just like the standard components. When you drag one to the canvas and drop it on top of a component, the validator or converter will bind to that component. (To test this, select the Length Validator and drop it on top of the `userName` text field component. You'll see a length validator `#{Page1.lengthValidator1.validate}` defined for the text field's `validator` attribute in the Properties window.)

Note that the components, validators, and converters all have icons next to them. Creator uses these icons consistently so you can easily spot what kind of component you're looking at. For example, reselect the JSF Standard Components bar in the palette and then select the Login button component on the design canvas. Now look at the Application Outline view. You'll see that the icon next to the button component in the JSF Standard Components palette matches the Login button in the Application Outline window.

Application Outline

Figure 2–6 is the Application Outline window for project **Login1**. (Its default placement is in the lower-left portion of the display.) The Application Outline window is handy because it shows both visual and nonvisual components for the page that's currently displayed in the design canvas. You can select other pages (here we have **LoginGood.jsp** and **LoginBad.jsp**) as well as the preconfigured managed beans, SessionBean1 and ApplicationBean1. These JavaBeans components are handy to hold your project's data that belong in either session or application scope, respectively. (We discuss scope issues for web application objects in "Scope of Web Applications" on page 128.)

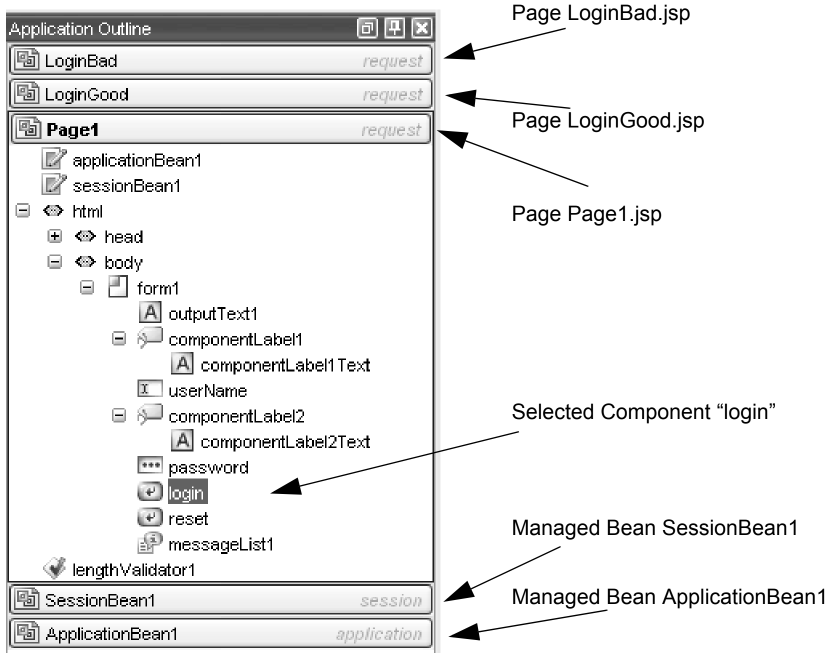


Figure 2–6 **Creator's Application Outline window for project Login1**

Some components are composite components, meaning that they contain nested elements. The Application Outline window will show this structure and allow you to expand and compress the display (using '+' and '-') as needed.

When you placed the length validator component on the `userName` text field, it appeared here as component `lengthValidator1`. Delete it by selecting it in the Application Outline view, then right-click and select Delete from the context menu. The validator should disappear from the Application Outline view.

Select the text field component `userName` and reset its `validator` attribute to null.

Creator Tip

Once you delete the `lengthValidator1` *component from your project, you must make sure that the validator attribute for component* `userName` *is really set to null. If it still references the validator, your project will build without any errors. However, the system will throw an exception because it cannot find the validator at run time.*

Now let's look at the Project Navigator window.

Project Navigator

Figure 2–7 shows the Project Navigator window for project **Login1**. Its default location is in the lower-right corner. Whereas the Application Outline view displays the components for individual pages and managed beans, the Project Navigator window displays your entire project. Project **Login1** contains three JSP pages: **Page1.jsp**, **LoginGood.jsp**, and **LoginBad.jsp**. Double-click on any one of them to pull it up in the design canvas. When the page opens, Creator displays a file name tab so you can easily switch among different files in the design canvas.

When you create your own projects, each page comes with a Java component "page bean." These are Java classes that conform to the JavaBeans structure we mention in Chapter 1 (see "JavaBeans Components" on page 13). To see the Java files in this project, expand the Java Sources folder (click on the '+'), then the **login1** folder. When you double-click on any of the Java files, Creator brings it up in the Java source editor. Without going to the editor, you can also see the Java classes, fields, constructors, and methods by expanding the '+' next to each level of the Java file.

The Project Navigator also lists the Resources node, which lives under the Web Pages node. The Resources node typically holds file `stylesheet.css` and any image files. The Library References node lists the libraries Creator needs to deploy your application. These are all of the JSF, Web Services, JDBC Rowset, and Exception Handler support classes. These class files (compiled Java classes) are stored in special archive files called JAR (Java Archive) files. You can see the name of the JAR files by expanding each of the nodes under Library References.[1] We show you how to add a Library Reference to your project in Chapter 5 (see "Add a Library Reference to Your Project" on page 133).

By default, the Project Navigator window shows you the Logical View of your project. Select the project name **Login1**, right-click, and choose Show FileSystem View from context menu, shown in Figure 2–8. The FileSystem View

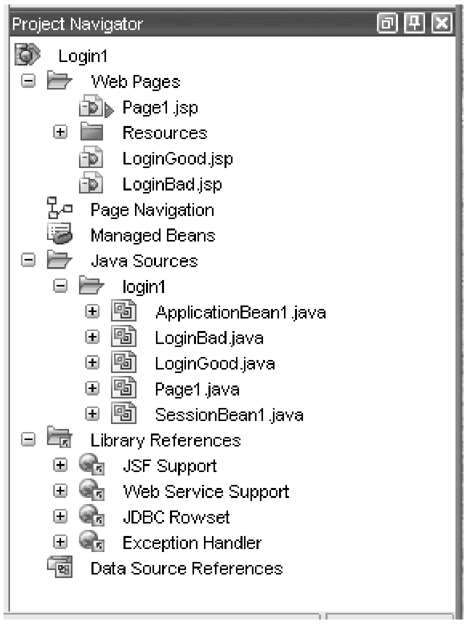


Figure 2–7 **Creator's Project Navigator window for project Login1**

shows all of the files in your project. This is handy for editing a configuration file that doesn't have a "visual editing interface." For example, expand the view until you see file **web.xml**, as shown below. Double-click this file. Creator brings it up in the editor pane. You can close the window again by selecting its File Tab at the top of the editor pane, right-click, and choose Close.

JSP Source Editor

With **Page1.jsp** in the design editor, bring up the JSP for this page by clicking the tab labeled Source at the bottom of the editor pane. Figure 2–9 shows Creator's JSP Source Editor.

1. For those of you who are curious about their contents, you can look at these jar files with the Java command-line `jar` command, Winzip or other compatible utility. With Winzip, be sure to specify `*.*` for all file types when opening an archive. The command that produces a file list of a jar archive is `jar tvf` *`lib.jar`*. The files are found in the directory structure of your project. (Look in `Login1/build/WEB-INF/lib` for the JAR files.)

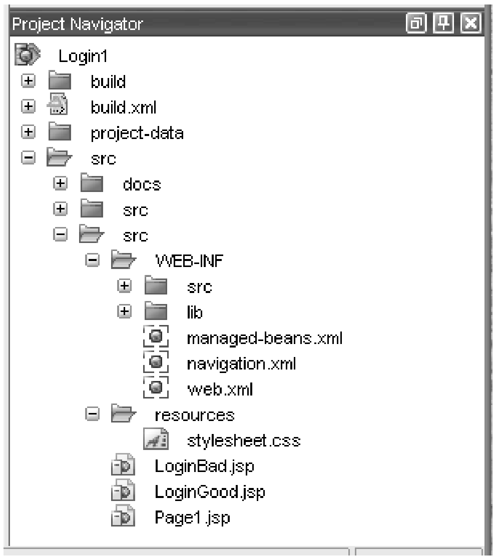


Figure 2–8 **The FileSystem View of the Project Navigator window.**

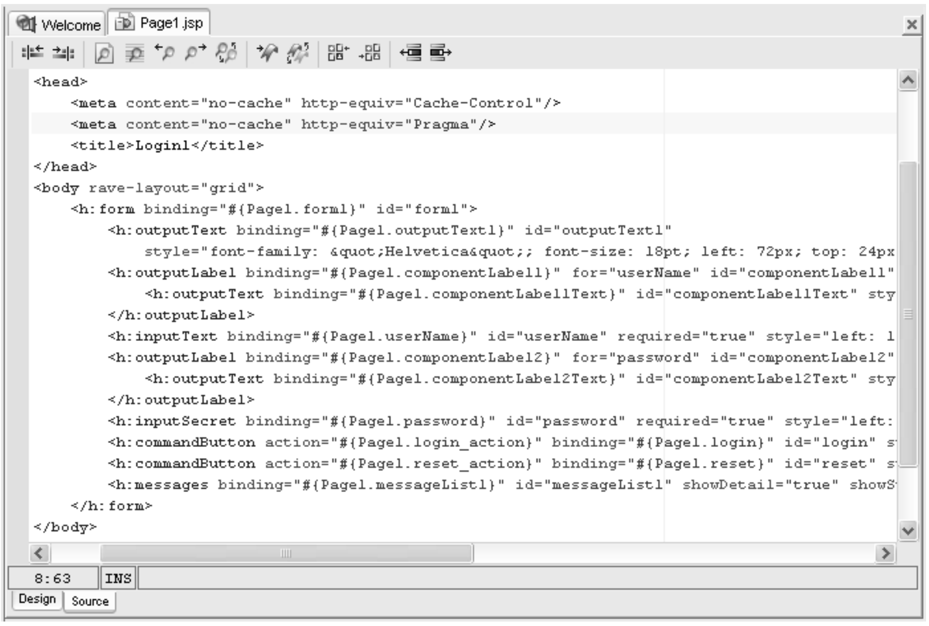
This is the JSP source that Creator generates for your page. Normally, you will not need to edit this page directly, but studying it is a good way to understand how JSF components work and how you can configure them by managing their properties. You'll see a close correspondence between the JSF tags and the components' properties as shown in the Properties window.

You'll also note the JSF Expression Language (EL) used to refer to methods and properties in the Java page bean. For example, the login button's `action` attribute is set to `#{Page1.login_action}`, which is a method in class **Page1.java**.

Let's look at the Java source for **Page1.java** now. Return to the design canvas for this page (select the tab labeled Design at the bottom of the editor pane). Right-click on the design canvas and select View Page1 Java Class from the menu.

Java Source Editor

You're looking at Creator's Java source editor for Java file **Page1.java**, the page bean for **Page1.jsp**. This Java file is a bean (conforming to a JavaBeans structure). Its properties consist of the components we placed on the page: each component corresponds to a private variable and has a getter and setter. This allows the JSF EL expression to access the properties of the page bean.



```
<head>
    <meta content="no-cache" http-equiv="Cache-Control"/>
    <meta content="no-cache" http-equiv="Pragma"/>
    <title>Login1</title>
</head>
<body rave-layout="grid">
    <h:form binding="#{Page1.form1}" id="form1">
        <h:outputText binding="#{Page1.outputText1}" id="outputText1"
            style="font-family: &quot;Helvetica&quot;; font-size: 18pt; left: 72px; top: 24px
        <h:outputLabel binding="#{Page1.componentLabel1}" for="userName" id="componentLabel1"
            <h:outputText binding="#{Page1.componentLabel1Text}" id="componentLabel1Text" sty
        </h:outputLabel>
        <h:inputText binding="#{Page1.userName}" id="userName" required="true" style="left: l
        <h:outputLabel binding="#{Page1.componentLabel2}" for="password" id="componentLabel2"
            <h:outputText binding="#{Page1.componentLabel2Text}" id="componentLabel2Text" sty
        </h:outputLabel>
        <h:inputSecret binding="#{Page1.password}" id="password" required="true" style="left:
        <h:commandButton action="#{Page1.login_action}" binding="#{Page1.login}" id="login" s
        <h:commandButton action="#{Page1.reset_action}" binding="#{Page1.reset}" id="reset" s
        <h:messages binding="#{Page1.messageList1}" id="messageList1" showDetail="true" showS
    </h:form>
</body>
```

Figure 2–9 **Page1.jsp Source window**

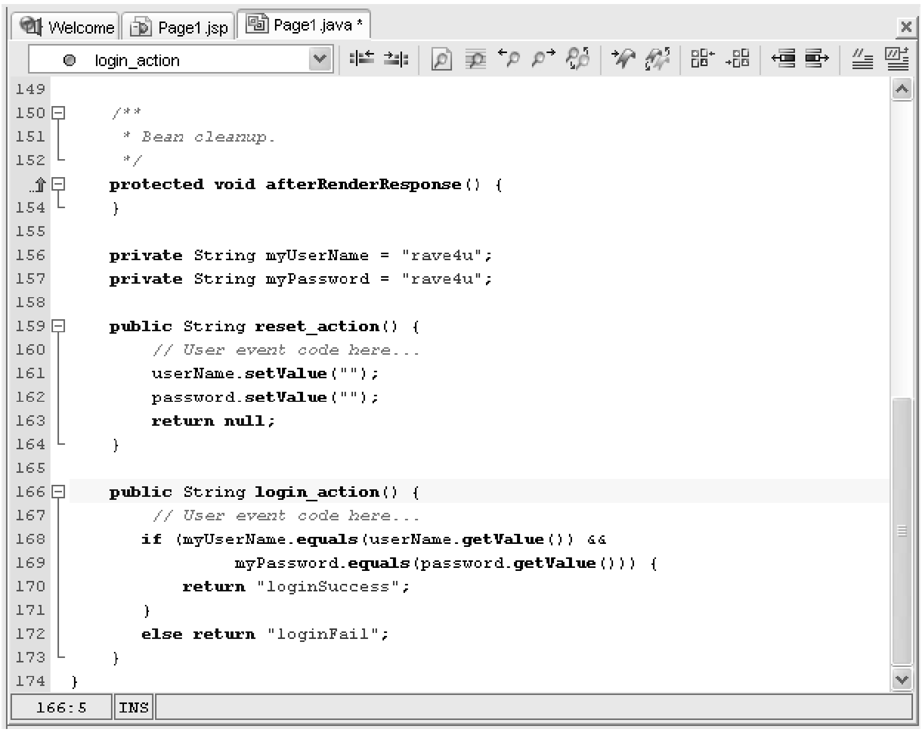
There's a dropdown menu at the top left of the window. You can use this menu to locate a field, method, or constructor in the file. Open the dropdown menu and select method `login_action`. Figure 2–10 shows this view.

There's much to learn about the Java source editor, so we'll just touch on a few tidbits now. First, we enabled line numbers here. To do this for your display, right-click in the margin (on the left-side of the editor window) and select Show Line Numbers. You see that method `login_action()` begins at line 166.

All of Creator's editors are based on NetBeans. The Editor Module is a full-featured source editor and provides code completion (we show an example shortly), a set of abbreviations, and fast import **<Alt-Shift-I>**.

To see the set of abbreviations, select Tools > Options from the menu bar. The Options dialog pops up. Under Options, select Editing > Editor Settings > Java Editor. On the right side of the display, click the small editing box next to Abbreviations. Creator pops up the window shown in Figure 2–11.

The window lists the abbreviations in effect for your Java editor. (You can edit, add, or remove any item.) For example, to add a `for` loop to your Java



```
    /**
     * Bean cleanup.
     */
    protected void afterRenderResponse() {
    }

    private String myUserName = "rave4u";
    private String myPassword = "rave4u";

    public String reset_action() {
        // User event code here...
        userName.setValue("");
        password.setValue("");
        return null;
    }

    public String login_action() {
        // User event code here...
       if (myUserName.equals(userName.getValue()) &&
               myPassword.equals(password.getValue())) {
           return "loginSuccess";
       }
       else return "loginFail";
    }
}
```

Figure 2–10 **Page1.java Java source editor**

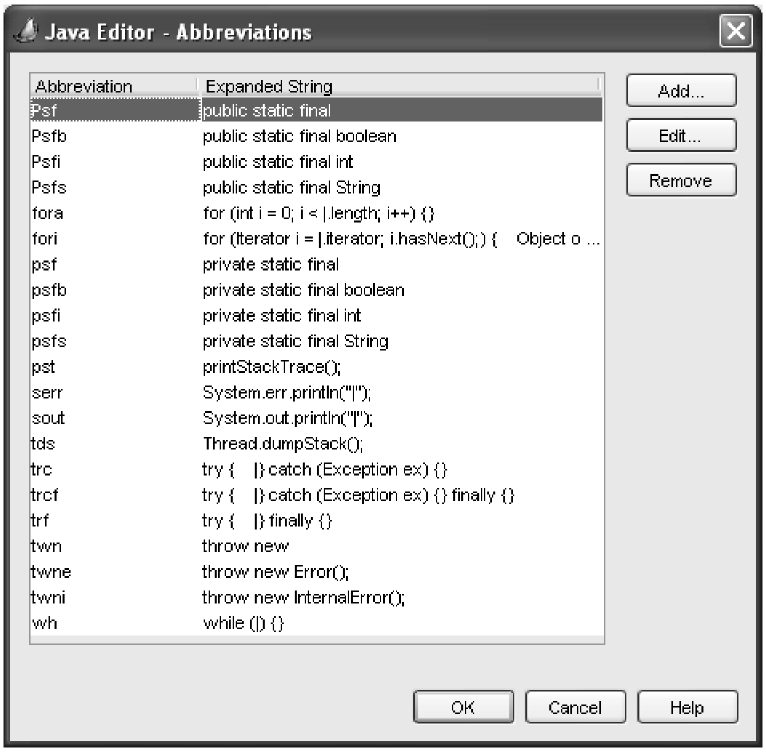
source file, type the sequence **fora** (*for array*) followed by **<Space>**. The editor adds

```
for (int i = 0; i < .length; i++) {
}
```

and places the cursor in front of `.length` so that you can add an array name. (`.length` refers to the length of the array object. This code snippet lets you easily loop through the elements of the array.)

The Java source editor also helps you with Java syntax and code completion. All Java keywords are bold, and variables and literal Strings have unique colors (you'll see that in your display; in the text here, it's different shades of gray).

When you add statements to your Java source code, the editor helps you by dynamically marking syntax errors (in red, of course). The editor also pops up

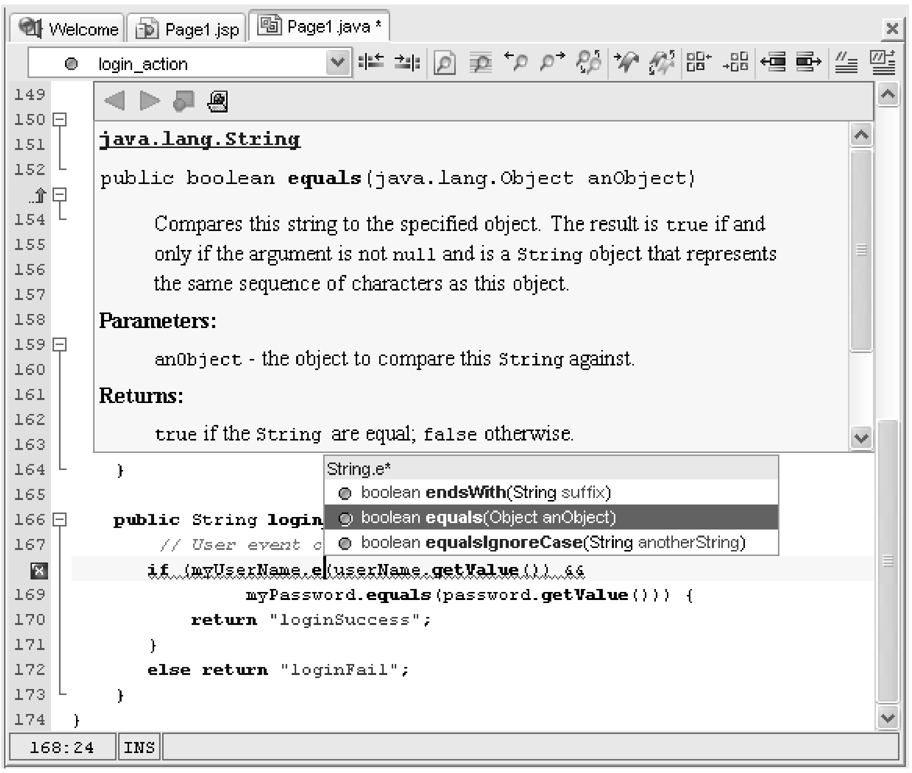


Abbreviation	Expanded String
Psf	public static final
Psfb	public static final boolean
Psfi	public static final int
Psfs	public static final String
fora	for (int i = 0; i < |.length; i++) {}
fori	for (Iterator i = |.iterator; i.hasNext();) { Object o ...
psf	private static final
psfb	private static final boolean
psfi	private static final int
psfs	private static final String
pst	printStackTrace();
serr	System.err.println("|");
sout	System.out.println("|");
tds	Thread.dumpStack();
trc	try { |} catch (Exception ex) {}
trcf	try { |} catch (Exception ex) {} finally {}
trf	try { |} finally {}
twn	throw new
twne	throw new Error();
twni	throw new InternalError();
wh	while (|) {}

Figure 2–11 **Java source editor list of abbreviations**

windows to help with code completion or package location for classes you need to reference (press **<Ctrl-Space>** to activate the code completion window). For example, Figure 2–12 shows the code completion mechanism as you start to type a method that begins with "e" for a String object (on line 168).

When you use the down-arrow to select method `equals()`, the help mechanism displays the Javadoc documentation about `equals()` for `java.lang.String`. With method `equals()` highlighted, press **<Enter>**. The code completion does more for you than you really want here. Delete all the characters after "`equals`" up to the parenthesis in front of "`userName`." The syntax errors should disappear. (To retrieve Javadoc documentation on any class in your source file, select it and press **<Ctrl-Shift-Space>**.)



java.lang.String

`public boolean equals(java.lang.Object anObject)`

Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.

Parameters:

`anObject` - the object to compare this `String` against.

Returns:

`true` if the `String` are equal; `false` otherwise.

```
String.e*
  boolean endsWith(String suffix)
  boolean equals(Object anObject)
  boolean equalsIgnoreCase(String anotherString)
```

```
    }

    public String login
        // User event c
        if (myUserName.e(userName.getValue()) &&
                myPassword.equals(password.getValue())) {
            return "loginSuccess";
        }
        else return "loginFail";
    }
}
```

```
168:24  INS
```

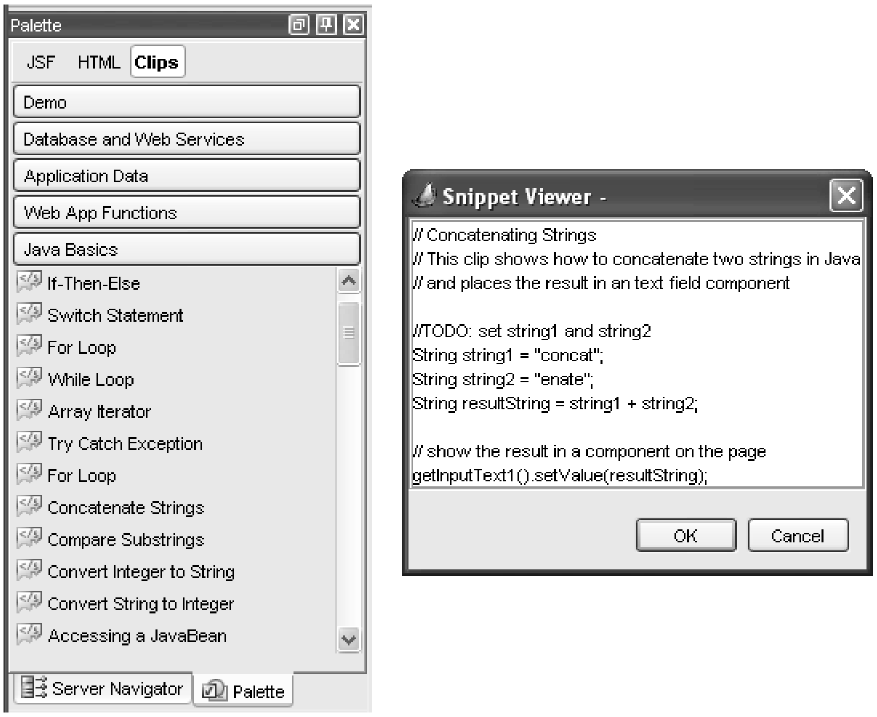
Figure 2–12 **Java source editor code completion**

Clips Palette

When the Java Source editor is displayed, Creator replaces the component palette with the Clips palette, as shown in Figure 2–13. Here we select the Java Basics Clips. Highlight clip Concatenate Strings. If you hold the cursor over the clip name, Creator displays a snippet window. You can drag and drop the clip directly into your Java source file. We do this when we build our first sample project in the next section.

To view or edit a clip, select it, right-click, and choose Edit. The second window in Figure 2–13 shows the Concatenate Strings clip.

The Clips palette is divided into categories to show sample code for that general topic. For example, if you click Application Data, you'll see a listing of



```
// Concatenating Strings
// This clip shows how to concatenate two strings in Java
// and places the result in an text field component

//TODO: set string1 and string2
String string1 = "concat";
String string2 = "enate";
String resultString = string1 + string2;

// show the result in a component on the page
getInputText1().setValue(resultString);
```

Figure 2–13 **Java Basics Clips Palette and Viewer**

clips that shows you how to access objects defined in your web application's different scopes.

Page Navigation Editor

Return to the Java Source window and look at method `login_action()`. You'll see that this method returns one of two Strings: either `"loginSuccess"` or `"loginFail"`. These Strings are returned to the action event handler, which then passes them to the navigation handler. The navigation handler manages page flow. Let's look at the Page Navigation editor now.

1. From the top of the Java source window, select the tab labeled **Page1.jsp**. This returns you to the design canvas for this page.
2. Now right-click in the design canvas and select Page Navigation from the context menu. Creator brings up the Page Navigation editor for project **Login1**, as shown in Figure 2–14.

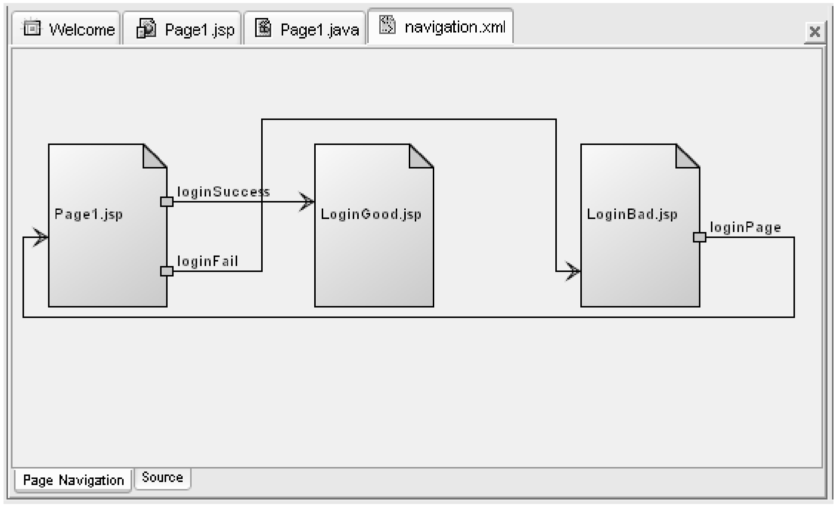


Figure 2–14 **Page navigation editor for project Login1**

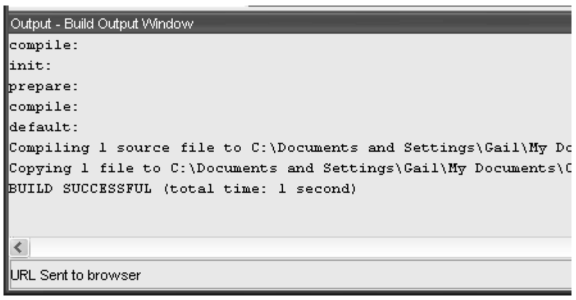
We mentioned earlier that there were three pages in this project. The Page Navigation editor displays these pages and indicates page flow logic with labeled arrows. The two labels originating from page **Page1.jsp** correspond to the return Strings we just looked at in the action method `login_action()`.

You learn how to specify Navigation in Chapter 4 (see "Page Navigation" on page 96). The Page Navigation editor is also a handy way to bring up any of the project's pages: just double-click inside the page. As you bring up each page, a corresponding file tab appears at the top of the editor pane. And once you've visited the Page Navigation editor, Creator displays a file tab called **navigation.xml** so you can easily return.

Before we explore our project any further, let's deploy and run the application. From the menu bar, select Build > Run Project. (Or, click the green chevron on the toolbar icon, which also builds and runs your project.)

Build Output Window

Figure 2–15 shows the build output window just as the application is finishing deployment. Creator uses the Ant build tool to control project builds. This Ant build process requires compiling Java source files and assembling the resources used by the project into an archive file called a WAR (Web Archive) file. Ant reads its instructions from a Creator-generated XML configuration file, called **build.xml**, in the project's directory structure.



```
compile:
init:
prepare:
compile:
default:
Compiling 1 source file to C:\Documents and Settings\Gail\My Do
Copying 1 file to C:\Documents and Settings\Gail\My Documents\C
BUILD SUCCESSFUL (total time: 1 second)
```

URL Sent to browser

Figure 2–15 **Build Output window for project Login1**

If problems occur during the build process, Creator displays messages in the Build Output window. A compilation error with the Java source is the type of error that causes the build to fail. When the build succeeds (the window will show `BUILD SUCCESSFUL`, as you see above), Creator tells the application server to deploy the application. If the application server is not running, Creator starts it for you. If errors occur in this step, messages come from the application server. Finally, it's possible for the deployment to be successful but for an exception to be shown in the browser's web page. A likely source of this type of error is a problem with JSF tags on the JSP page or a resource that is not available for the runtime class loader.

When the build/deployment process is complete, Creator brings up your browser with the correct URL. (The status window displays "Starting browser for `http://localhost:18080/login1`.") So, to run project **Login1** with the Sun J2EE 1.4 Application Server, Creator uses this web address.

```
http://localhost:18080/login1/
```

You use `localhost` if you're running the application server on your own machine; otherwise, use the Internet address or host name where the server is running. The port number `18080` is unique to Sun's J2EE application server. Another server will use a different port number here.

The Context Root is **/login1** for this application. The application server builds a directory structure for all its deployed applications, and the context root is the "base address" for all the resources that your application uses.

Figure 2–16 shows the **Login1** project deployed and running in a browser. Type in some values for Username and Password. If you leave the Username field empty, you'll get a validation error. The correct Username and Password is "rave4u" for both fields.

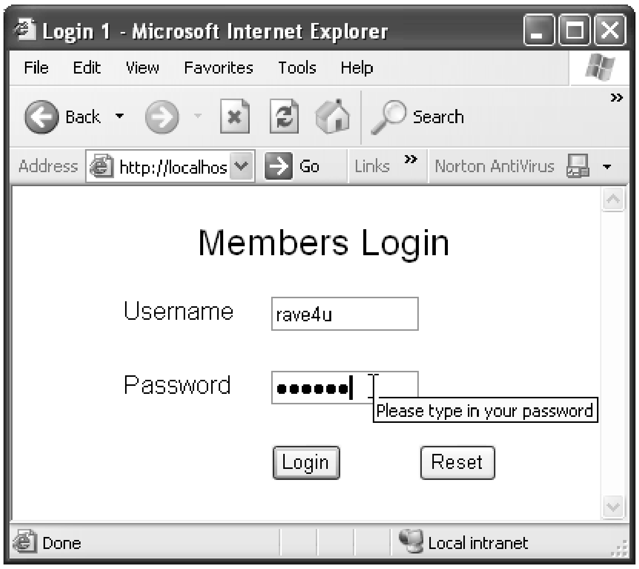


Figure 2–16 **Login page web application**

If you type in the correct values and click the Login button, the program displays page **LoginGood.jsp**. Incorrect values display **LoginBad.jsp**. You'll build project **Login1** from scratch in Chapter 4 ("Dynamic Navigation" on page 111).

It's time now to explore the Server Navigator window, located in the upper-left portion of your Creator display. Click the tab labeled Server Navigator.

Server Navigator

Figure 2–17 shows the Server Navigator window after you've deployed project **Login1**. Four categories of servers are listed here: Data Sources, Web Services, Deployment Server, and Database Server.

The Data Sources node is a JDBC database connection. The default database server is PointBase, but you can configure a different one. Creator comes configured with several sample databases, which are visible if you expand the Data Sources node.

Creator Tip

PointBase must be running. You can start it on Windows by clicking the Start button and selecting All Programs > Sun Microsystems > J2EE 1.4 SDK > Start PointBase. You will need to restart Creator.

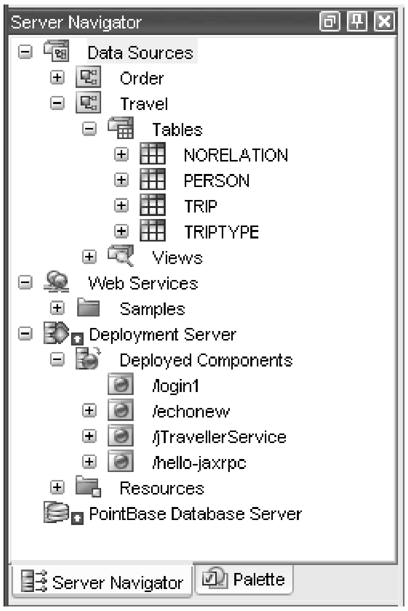


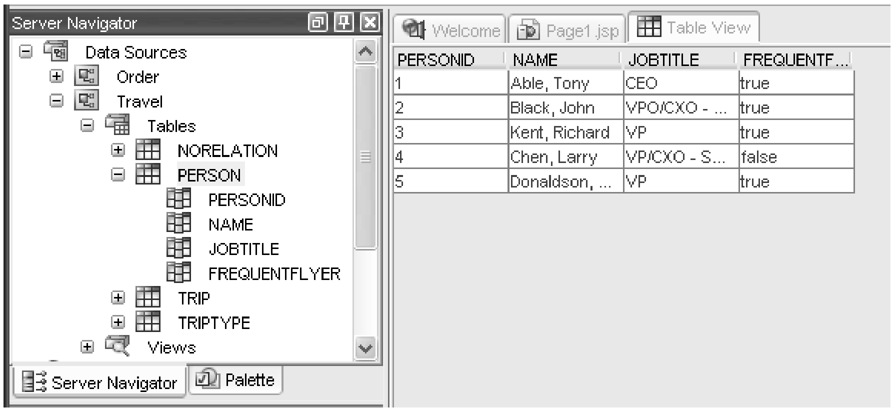
Figure 2–17 **Server Navigator window**

Now expand the Travel > Tables node to see the database tables as shown (there are four). As you select different tables, Creator displays their properties in the Properties window.

You can expand each table further and see the database table field names. If you double-click on the table name, Creator displays the data in the editor pane, as shown in Figure 2–18. You can close the table view by selecting the tab labeled Table View, right-click, and select Close from the context menu. We discuss creating web applications that access databases in Chapter 7 (see "Accessing Databases" on page 204).

The second server type is Web Services, which provides access to remote APIs from Creator applications. This requires the cooperation of several Java technologies, which we discussed in Chapter 1. When you install Creator, it includes a web services API to access Google. In Chapter 6 we show you how to create an application that uses the Google web service API.

The third server type is the Deployment Server, which is the application server. Sun's J2EE 1.4 application server is Creator's default deployment server, which manages the deployment of web applications that you create. To start or stop the server, select the Deployment Server node, right-click, and select Start/Stop Server. Creator pops up a window that allows you to initiate one of these actions, or you can simply close the window.



PERSONID	NAME	JOBTITLE	FREQUENTF...
1	Able, Tony	CEO	true
2	Black, John	VPO/CXO - ...	true
3	Kent, Richard	VP	true
4	Chen, Larry	VP/CXO - S...	false
5	Donaldson, ...	VP	true

Figure 2–18 **Selecting View Data under the Data Sources node**

If you expand the Deployment Server node, you'll see the Deployed Components node. Under this node are the deployed applications. The **/login1** application is listed (since you deployed it). You can undeploy an application by right-clicking its node and selecting Undeploy.

The J2EE application server also has administration access through your browser at

```
http://localhost:14848/asadmin/
```

You can get there directly from Creator by selecting the Deployment Server node, right-clicking, and selecting Show Admin Console. (The application server must be running for access to the administration port.) Use user name `admin` and password `adminadmin`.

Debugger Window

Creator has a debugger that lets you trace the call stack, track local variables, and set watches. You can see this window by selecting View > Debugger Window from the toolbar. To run your application in "debug mode," click on Debug > Debug Project from the menu bar. The application server has to stop and restart if it's not already in debug mode. In Chapter 9 we walk you through the debugger options, setting breakpoints, stepping through code, and other debugging activities.

Creator.book Page 42 Thursday, June 10, 2004 2:17 PM

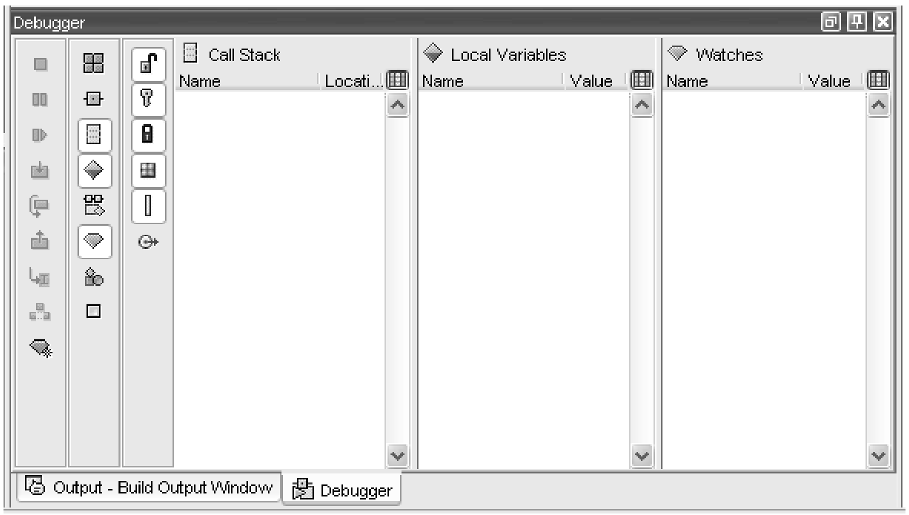


Figure 2–19 **Creator debugger window**

Creator Help System

The Creator Help System is probably the most useful window for readers new to Creator. This help system includes a search capability, contents, and an index. You access the help system by selecting Help > Help Contents in the menu bar. In Figure 2–20 we clicked the Index tab and selected entry “binding data to list components.” The relevant information appears in the adjacent window.

2.3 Sample Application

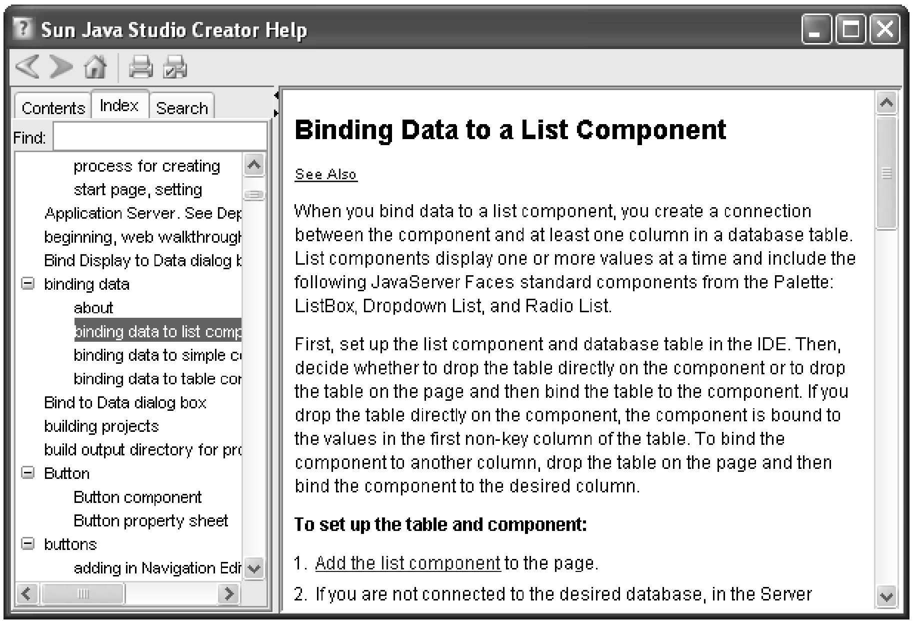
Now that you’re comfortable with Creator, let’s create a simple web application. Even though this application is simplistic, it hints at some of the power in Creator. Figure 2–22 on page 46 provides a preview of this web application.

Create a Project

Close project **Login1** if it’s open.

1. From the Welcome window, select button Create New Project.
2. Specify **Echo** for the Name, then click OK.

Creator.book Page 43 Thursday, June 10, 2004 2:17 PM



process for creating
start page, setting
Application Server. See Dep
beginning, web walkthrough
Bind Display to Data dialog b
binding data
about
binding data to list comp
binding data to simple co
binding data to table cor
Bind to Data dialog box
building projects
build output directory for pro
Button
Button component
Button property sheet
buttons
adding in Navigation Edi

Binding Data to a List Component

See Also

When you bind data to a list component, you create a connection between the component and at least one column in a database table. List components display one or more values at a time and include the following JavaServer Faces standard components from the Palette: ListBox, Dropdown List, and Radio List.

First, set up the list component and database table in the IDE. Then, decide whether to drop the table directly on the component or to drop the table on the page and then bind the table to the component. If you drop the table directly on the component, the component is bound to the values in the first non-key column of the table. To bind the component to another column, drop the table on the page and then bind the component to the desired column.

To set up the table and component:

1. Add the list component to the page.
2. If you are not connected to the desired database, in the Server

Figure 2–20 **Creator Help window**

Specify Title

Creator initializes a project for you and brings up the design canvas editor. Now you'll be able to add components and modify properties.

1. If the Properties window is not enabled for the page, select the page by clicking anywhere inside the design canvas.
2. In the Properties window under General, change the `Title` attribute to **Echo**. Finish with **<Enter>**.

Add Components

You'll add three components to the page: a component label, a text field, and an output text component. When you created project **Echo** and Creator brought up the design canvas, this made the JSF Standard Components palette visible.

1. From the JSF Standard Components palette, select Component Label and drag it over to the design canvas. Drop it onto the canvas, near the top on the left side. Don't resize it.

Creator Tip

If you resize the component, Creator generates static length and width attributes for your component. If you leave it unsized, the rendering mechanism will dynamically size the component to accommodate the text it needs to display.

2. Component label is a composite component; it contains an embedded (nested) output text component. The output text component displays the label. From the Application Outline view, select the component label's nested output text component, named `componentLabel1Text`.

Creator Tip

Creator provides an enhanced selection mechanism for composite components (such as component label). In the design canvas, select the component label and look in the Properties window to see which component is selected. Now click the component again and you'll see a different component's properties. For component label, you switch back and forth between the top-level, label component and the embedded output text component by clicking the component in the design canvas.

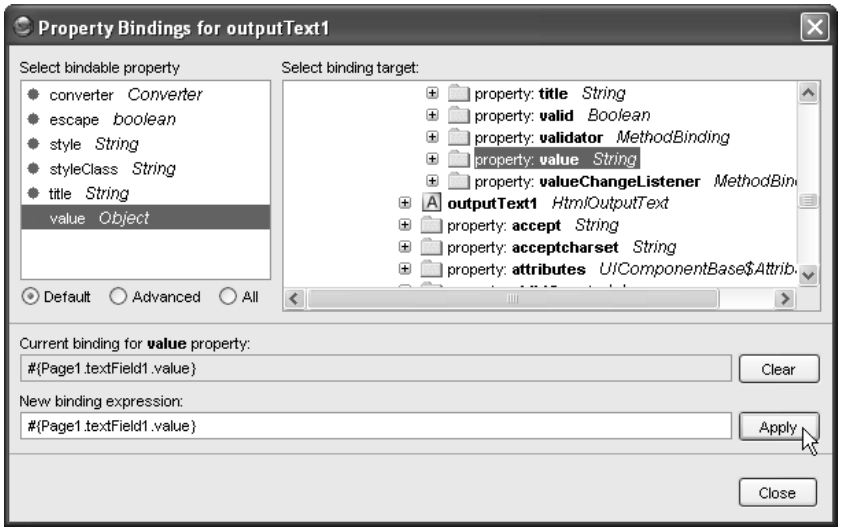
3. Now go to the Properties window and select the `value` attribute. Type the label **Type in some text**. Finish with **<Enter>**. The text you type will appear in the component.
4. From the JSF Standard Components palette, select component Text Field and place it on the design canvas underneath the component label you just added.
5. Make sure that it's selected, and in the Properties window, change its `value` attribute to **Echo Text**. Finish with **<Enter>**.
6. Still in the Properties window for the text field, change its `title` attribute to **The text you enter here will be echoed below**. Finish with **<Enter>**. You've just created a tooltip for this component.
7. Now select component label `componentLabel1` (the top-level component) from the Application Outline view.
8. In the Properties window, select the small editing square opposite the `for` attribute. Creator pops up a dialog for you to choose a component id. Select component `textField1` and click OK. This associates the label component with the text field component. You'll see (when you run the web application) that this affects GUI operations such as cursor movement and selection behavior. For example, selecting the label also selects the text field.
9. From the JSF Standard Components palette, select an Output Text component and place it under the text field. (Again, don't resize it.)

You've finished adding the components. Now you will use property binding to bind the text field component `value` property to the output text component `value` property. Here's how.

1. Select the output text component (`outputText1`), right-click, and choose Property Bindings from the context menu. Creator brings up the Property Bindings dialog. (See Figure 2–21.)
2. Under Select bindable property, choose **value Object**.
3. Under Select binding target, expand Page1 > html > body > form1 by clicking the '+' at each level.
4. Select component **textField1** (the text field component you added). Expand the component by clicking '+' on `textField1` and select **value String**. (The properties are listed in alphabetical order, so `value` is near the end.)
5. Click the Apply button. (If you don't click the Apply button, Creator doesn't set the property binding.) Under Current binding for **value** property, you should see the following JSF EL expression

```
#{Page1.textField1.value}
```

6. Click Close to finish.



Property Bindings for outputText1

Select bindable property
- converter *Converter*
- escape *boolean*
- style *String*
- styleClass *String*
- title *String*
- value *Object*

(●) Default ( ) Advanced ( ) All

Select binding target:
- property: **title** *String*
- property: **valid** *Boolean*
- property: **validator** *MethodBinding*
- property: **value** *String*
- property: **valueChangeListener** *MethodBin*
- **outputText1** *HtmlOutputText*
- property: **accept** *String*
- property: **acceptcharset** *String*
- property: **attributes** *UIComponentBase$Attrib*

Current binding for **value** property:
#{Page1.textField1.value}

New binding expression:
#{Page1.textField1.value}

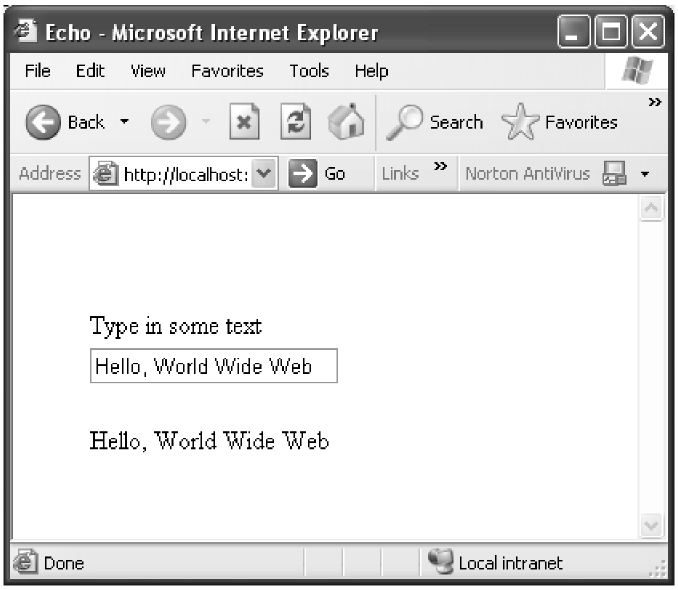
Figure 2–21 **Property Bindings dialog**

So, what did all this accomplish? You've just configured *property binding* on the output text component (id `outputText1`). This means that JSF gets the `value` attribute (the text that is displayed on the page) for `outputText1` *from* the text field's value (component `textField1`). This, in turn, means that whatever you type for input will be echoed in the output text's display.

Deploy and Run

You've finished creating the application. Now it's time to build, deploy, and run it. From the menu bar, select Build > Run Project. Creator builds the application, deploys it, and brings up a browser with the **Echo** web application running.

Figure 2–22 shows what the browser window displays after you type **Hello, World Wide Web** inside the text field followed by **<Enter>**.



Type in some text

Hello, World Wide Web

Hello, World Wide Web

Figure 2–22 **Web application Echo running in a browser**

Modify Project Echo

As promised, let's access the Clips palette and add some Java code to our project. First, we'll add an output text component.

1. Make sure the Creator design canvas is visible.

2. From the JSF Standard Components palette, select Output Text component and place it underneath component `outputText1`. This adds component `outputText2` to your page. (You should see `outputText2` when you look at the Application Outline view of the project.)
3. Make sure component `outputText2` is selected and in the Properties window click on the small editing box opposite attribute `value`.
4. Creator pops up a dialog to manipulate the component's value. Click button Reset to default. This removes any text from the design canvas for this component.

Now you will bring up the Java source editor to manipulate the Java page bean for this page.

1. Place the cursor anywhere in the design grid, right-click, and select View Page1 Java Class. This brings up the Java Source editor with file **Page1.java**.
2. Open the dropdown menu at the top of the editor pane and select the constructor, `Page1()`. Creator will place the cursor at the beginning of this method and highlight the line in yellow. (Note that some Creator-managed code is hidden—"folded"—to help keep your editor window uncluttered.)
3. Move the cursor to the end of the following line.

```
// Additional user provided initialization code
```

4. Click the mouse and start a new line by pressing **<Enter>**. Since you're in the Java Source editor, the Clips palette is visible to the left of the editor pane.
5. Select Java Basics > Concatenate Strings. Drag this clip to your Java source and drop it directly underneath the above comment line (where you just added a new line).

   Creator adds the following code to your Java source file.

```
// Concatenating Strings
// This clip shows how to concatenate two strings in Java
// and places the result in a text field component

//TODO: set string1 and string2
String string1 = "concat";
String string2 = "enate";
String resultString = string1 + string2;

// show the result in a component on the page
getInputText1().setValue(resultString);
```

Note that the last line is underlined (in red), since it produces a syntax error. Before we fix the error, let's align the newly added code.

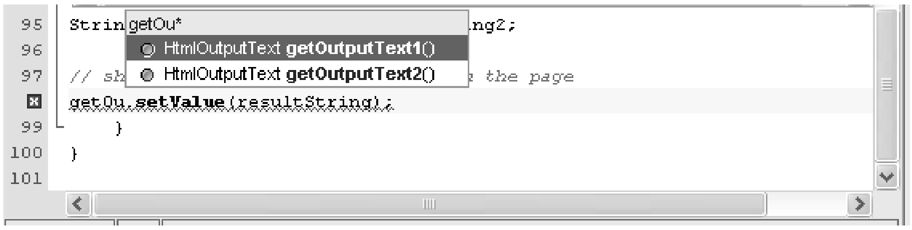
1. Select the code you just added by swiping all the lines with the mouse. The selected lines turn gray.
2. At the top of the editor pane, hold the cursor over the icons until you find the icon with the tooltip "Shift Line Right (Ctrl + T)."
3. Click the Shift Right icon until the selected code block is aligned with the line above it.

Let's fix the flagged error now. We need to use method `getOutputText2()` instead of method `getInputText1()`.

1. Change the underlined line above to the following.

```
getOutputText2().setValue(resultString);
```

(Remove `InputText1()` and start typing the replacement text. Then press **<Ctrl-Space>** and see the code completion box, as shown in Figure 2–23. Here we are given two choices. Use the down-arrow key to select method `getOutputText2()` and press **<Enter>**.)


getOu*
HtmlOutputText getOutputText1()
HtmlOutputText getOutputText2()

```
String                                  ng2;

// sh                                   the page
getOu.setValue(resultString);
    }
}
```

Figure 2–23 **Code completion helping out**

2. Redeploy and run web application **Echo**. You'll see that the String "concatenate" displays in the output text component on the page. We placed this code in Page1's *constructor*. This is where you provide initialization code that is executed before the page is displayed.

This completes our tour of Creator. The next chapter provides a detailed description of the JSF standard components, validators, and converters.

2.4 Key Point Summary

- Creator has multiple windows to give you different views of the project that you’re working on. The windows can be sized, docked and undocked, or hidden.
- From the main menu, select View and the desired window name to enable viewing.
- Use the Welcome Window to select a Project to open, or to create a new project.
- The design canvas allows you to manipulate components on a page and control their size and placement.
- Use the components palette to place a component on the design canvas.
- Use the converters/validators palette to select data converters and input validators for your project.
- The Properties window allows you to inspect and edit a component’s properties. Each element type displays a different list of properties.
- A component’s `value` attribute usually contains text that is displayed (such as labels on buttons or input and output fields). The `title` attribute gives you a tooltip for the component and the `style` attribute lets you change the font characteristics.
- You can apply Property Binding and “connect” the value of one component to another.
- The Application Outline window shows all of the elements on a page, including nonvisual components.
- The Project Navigator displays an entire project, including Java source files. Choose between the Logical view and the FileSystem view by right-clicking the project name.
- The JSP Source editor displays a page’s source. Most of the page includes JSF tags for components and their properties. As you make changes in the design canvas, Creator keeps the JSP source (as well as the Java source) synchronized with your changes.
- The Java Source editor displays the Java source for each “page bean,” the JavaBeans component that manipulates each page’s elements. You typically place event handler code or custom initialization code in the Java page bean.
- The Java Source editor includes a code completion mechanism that provides pop-up windows with possible method names (use **<Ctrl-Space>** to invoke) and Javadoc documentation for classes and objects in your program (use **<Ctrl-Shift-Space>** to invoke). It also includes a dynamic syntax analyzer to warn you about compilation errors before you compile.
- The Clips palette provides sample Java code to accomplish common programming tasks. The Clips are organized into categories based on function.

- The Page Navigation editor lets you specify page flow. It builds a navigation configuration file, **navigation.xml**.
- When you build your project, the Build Output window provides diagnostic feedback and completion status.
- The Server Navigator window displays the Data Sources, Web Services, Deployment Server, and Database Server nodes.
- You can start and stop the application server and undeploy running web applications from the Deployment Server node.
- You can view Table data by expanding the Data Sources node and selecting individual tables. Creator displays the data in the editor pane when you right-click the table name and select View Data.
- The Debugger Window displays several views that are helpful when you are debugging your project. You can monitor the call stack, local variables, and watches, for example.
- The Creator Help system provides a table of contents, an index, and a search mechanism to help you use Creator.