

BEYOND THE BOOK

Topics in This Supplement

- Global Issues
- Look and Feel Enhancements
- Discrepancies in Project Build Instructions
- Additional Project Examples



Appendix A

Beyond the Book is a work in progress. We first published this document in August of 2004 to keep pace with updates released by the Creator Team. In January 2005 the Creator Team released Update 6, a major release entitled Reef Shark. In this document, we'll point out any discrepancies between how Creator works and the instructions we've given you to build the projects. The good news is that the discrepancies are few and minor. We'll also point out enhancements and better approaches that affect the instructions as we've learned more about the product.

Future postings will show examples that illustrate Creator features not discussed in this book.

Like all major products, Sun Java Studio Creator will go through improvements as it expands its functionality. We intend to revise Java Studio Creator Field Guide to keep pace with Creator's releases. We'll use this supplement to explore new features as new versions of the product are released.

A.1 Global Issues

Creator Updates

An update to Sun Java Studio Creator (Update 6, Reef Shark) has been released (January 2005), and is available from the Creator Update Center. To install the update, select Tools > Update Center. Several dialogs will step you through the

process. To read about the bug fixes, see the Sun Java Studio Creator web site at <http://developers.sun.com/prodtech/javatools/jscreator/index.jsp>.

Look and Feel

The Look and Feel of the Creator IDE has been enhanced to ease the developer's design time chores. We'll cover some of these improvements. Many of them result in slight modifications to instructions we've outlined in the Creator Field Guide Book.

Design View

There are a few cosmetic changes in the design view. When the design canvas is displayed, the tabs at the bottom of the editor pane are labeled "Visual Design" and "JSP Source." To bring up the Java page bean, right-click inside the editor pane and select context menu "Edit Page1 Java Source" instead of "View Page1 Java Class."

Message Components

When you place either a message list or an inline message component on a page, Creator sets attribute `showDetail` to false and `showSummary` to true. In general, you do not need to change the default settings of these components, including attribute `style` or `styleClass`. (In all of our project build instructions, we have you change the font-style to italic. We recommend that you leave it unaltered.)

Furthermore, you can use a keyboard short-cut to set the `for` attribute for the inline message component. When you place an inline message component on the design canvas, Creator displays the text

`Ctrl+Shift` and drag to set "for" property

Press **<Ctrl-Shift>** and drag the cursor to the component that you'd like to link to (see Figure A-1). Creator links the inline message component to the target component by setting its `for` property visually. You can use **<Ctrl-Shift>** and drag to set the `for` property visually with component labels as well.

We recommend that you use message list components on your web pages. Any conversion or validation errors produce error messages which are only visible if you've placed one of the message components on the page. The message list component also allows you to provide feedback for the user using the page bean methods listed in Table A.1.

Each of these methods creates a `FacesMessage` with the summary text provided by the `String` argument. If you include a component argument, the mes-

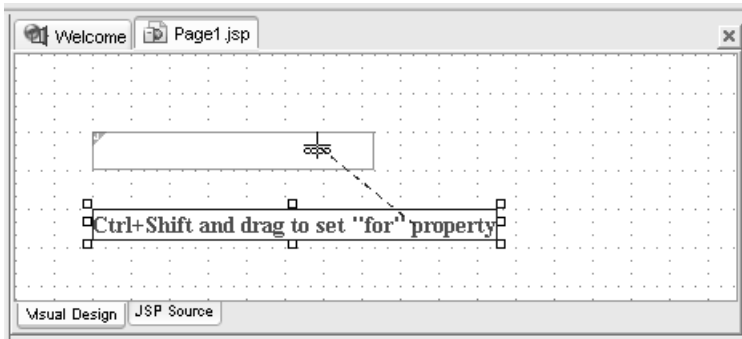


Figure A-1 Setting the “for” property visually

Table A.1 Methods for displaying text in a message component

	least severe
<code>void info(String)</code>	
<code>void info(UIComponent, String)</code>	
<code>void warn(String)</code>	
<code>void warn(UIComponent, String)</code>	
<code>void error(String)</code>	
<code>void error(UIComponent, String)</code>	
<code>void fatal(String)</code>	
<code>void fatal(UIComponent, String)</code>	
	most severe

sage is associated with that component. Otherwise, the method creates a global `FacesMessage`. Note that each method’s name determines its severity level.

Application Outline and Default Properties

The Application Outline view displays a page’s components. It includes the component’s icon, its id (its unique name as defined in the page bean), and its *default property value*. The default property is a component’s “main” property. For output text, text field, and button components for example, this is the value property. By displaying the default property, you can more easily identify which component to select in the Application Outline view, especially when many of the same components are placed on a page. Figure A-2 shows the Application Outline view from project `MusicInsert`. You can see that the

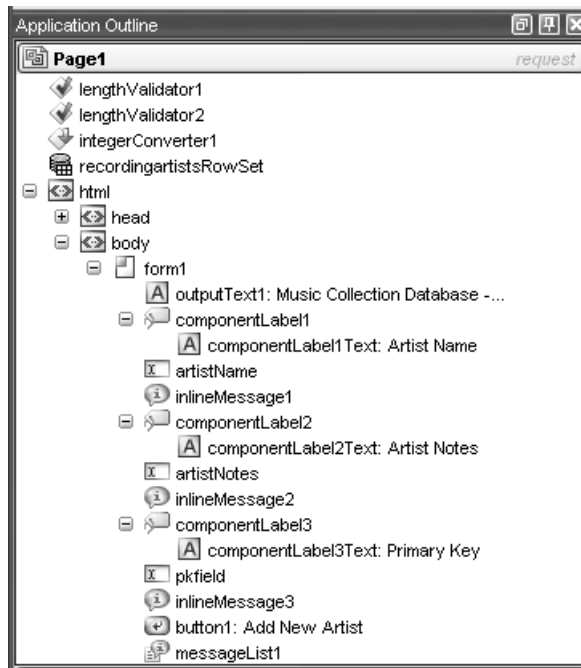


Figure A–2 Application Outline view for project MusicInsert

component label’s embedded output text components each display their value making them easy to distinguish from each other.

StyleClass Dialog

In the example projects described in the Creator Field Guide, we use an output text component to hold text for a page’s heading. After you place the output text component on the page, you are told to set its style attributes to include font-style Helvetica and a larger font-size. A better approach is to use a modified **stylesheet.css** file and set the component’s `styleClass` using a newly defined style rule. We’ve rebuilt all the projects in the download examples file using this approach. Here’s how to do this.

1. Once you have created a project, select **stylesheet.css** under Resources in the Project Navigator window.
2. Double-click and Creator opens the style sheet in the editor pane.
3. Copy and paste the contents of file **FieldGuide/Examples/Resources/stylesheet.css** and replace the default contents in the editor.
4. Save the changes to your project by selecting File > Save All from the top menu.

The modified style sheet includes the following rule for `body`. This makes all text that appears inside the HTML `body` tag appear in Helvetica. Now you don't have to modify each component's style attribute to include `font-family: Helvetica`; it will appear in Helvetica automatically.

```
body {  
  background-color: white;  
  color: black;  
  font-family: Helvetica;  
}
```

The second addition includes the following rule for style class `heading`.

```
.heading {  
  font-size: 200%;  
}
```

When you place an output text on a page and you want to use it as a page heading, select the small editing box opposite attribute `styleClass`. Creator pops up a styleClass selection dialog. Select styleClass `heading`, click on the right arrow (`>`), and finish with OK. The design canvas reflects the new style rules. You may apply more than one style class. Figure A-3 shows the styleClass selection dialog.

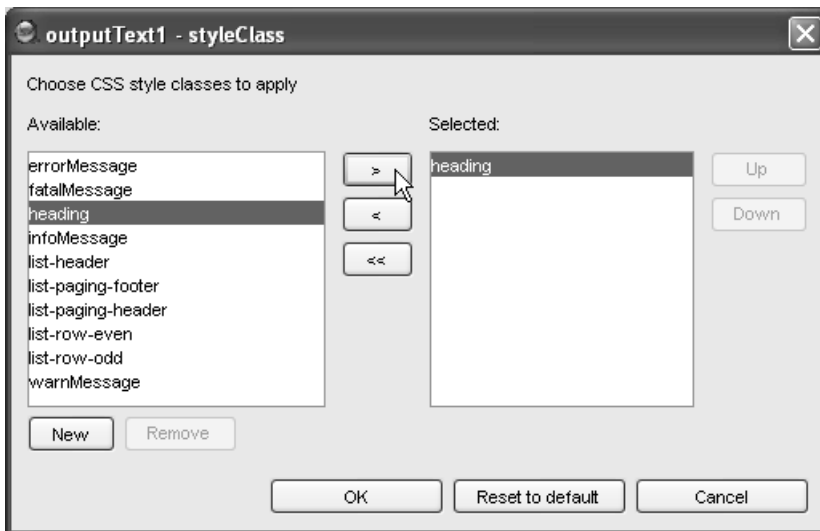


Figure A-3 Style class selection dialog

When you're designing your own web pages, you'll want to achieve a uniform look using style rules defined in a style sheet file. We discuss this in more detail in *Beyond the Book 2* ("Using Attribute styleClass" on page 4).

Message Bundles

In Chapter 8 we display text for German and Spanish versions of several projects. The text for these projects are contained in **.properties** files, which consist of key-value pairs. We've converted these files to use ASCII only text using the Java utility `native2ascii`. This utility is included in the Java SDK toolkit you installed with Creator. Utility `native2ascii` converts special symbols such as ñ or ö to unicode values (`\uxxxx`, where x is a hexadecimal digit).

Cleaning Projects

When Creator builds a project, it creates a directory structure under the build directory in your project's directory tree. Cleaning a project clears the build directory and is a way to "start over" when compiling class files and copying configuration XML files and other resources for deployment. Clean Project also undeploys the web application. This is handy for conserving your computer's resources.

To clean a project, select the project name (the top node) in the Project Navigator window, right-click, and select Clean Project. Alternatively, select Build > Clean Project from the top menu bar.

We recommend cleaning a project when you want to archive or port your application to another system (a cleaned project takes much less space). We also recommend cleaning a project before performing Save Project As.

Certain components will not render correctly at design time after cleaning a project. For example, suppose your page includes a data table component that is bound to a JavaBeans component that has been added to your project as a Java class. When you build the project, Creator generates the class file for you, making the custom JavaBeans component available in the Property Bindings dialog. After cleaning the project, the class file is removed and the component that binds to this object cannot be rendered and is displayed in red. To be able to render the component at design time, build the project then close and reopen it in the IDE.

Custom JavaBeans components that are stored in JAR files as a library reference will always be available to the IDE, even when you clean the project.

Library Reference

When you create a Library Reference, Creator stores the absolute pathname to the library's JAR file. If you port the project to another system, make sure you also copy the JAR file and install it at the same path location. Or, if the JAR file pathname changes, you can recreate the library reference.

When you download the examples projects for this book, projects Login2, Login3, and Color1 contain a library reference to **C:\FieldGuide\Examples\asg.jar**. If you install the FieldGuide zip file at another location, you'll have to recreate the library reference for these projects before you can deploy them. (You may need to close the project and reopen it in order for the design canvas to display correctly.) This is not an issue if you're building the projects from scratch.

A.2 Discrepancy Notes

We've organized these notes by chapter, project name, and page number so you can easily refer to the list.

Chapter 2

Project Login1

Page 27. When you drop a length validator onto the text field, `lengthValidator1` appears in the text field's `validator` attribute in the Properties window (not `#{Page1.lengthValidator1.validate}`).

Page 29. (Creator Tip.) When you delete a validator from your project, Creator clears the text field's `validator` attribute for you.

Page 30. The tab is labeled JSP Source.

Page 31. Figure A-4 shows the FileSystem View of the Project Navigator window for project Login1 (some of the directory and file name labels are different than the structure shown in Figure 2-8 on page 31).

Page 31. Select the tab labeled Visual Design to return to the design canvas. Right-click on the design canvas and select Edit Page1 Java Source (the menu text has changed).

Page 37. (Figure 2-14) The file tab label for the Page Navigation editor is **Page Navigation**, not **navigation.xml**.

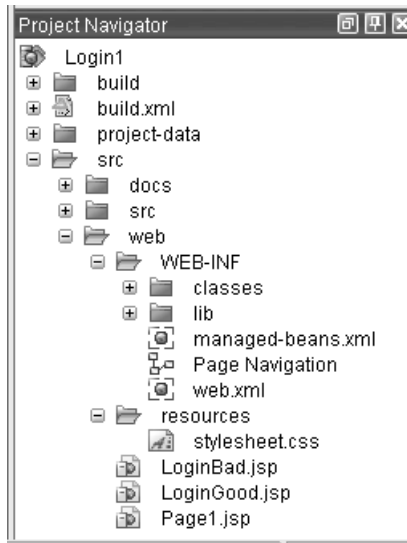


Figure A–4 The FileSystem View of the Project Navigator window.

Project Echo

Page 44. (Step 8.) Use **<Ctrl-Shift>** and drag the mouse from the component label to component `textField1`. This sets the component label's for property to the text field.

Page 47. (Steps 3 & 4 at the top of the page.) It is not necessary to modify attribute value.

Chapter 4

Project Navigate1

Page 103. (Middle of page.) The file tab is labeled **Page Navigation**, not **navigation.xml**.

Page 105. (Step 3.) *Clarification:* Return to the design canvas, select the button Members Login, and right-click. Select Edit Event Handler > action. Creator

brings up **Page1.java** in to Java source editor. Change the return statement to

```
return "userLogin";
```

Chapter 5

Project Payment1

Page 150. (Step 3.) The snippet initializes property amount to 100,000, not 10,000 (100,000 is correct).

Page 151. (Configuration file managed-beans.xml.) Creator generates the following `<managed-bean>` element for the `LoanBean` managed session bean instead of what's shown in the book:

```
<managed-bean>
  <managed-bean-name>
    asg$bean_examples$LoanBean
  </managed-bean-name>
  <managed-bean-class>
    asg.bean_examples.LoanBean
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Page 153. (Figure 5–9) Figure A–5 shows the design canvas for project Payment1.

Page 155. (Step 2.) Creator displays `doubleRangeValidator1` in the Properties window for the validator attribute for component `loanAmount`.

Page 156. (Steps 3–5.) To specify property binding for component `loanAmount`, choose `amount` under node **asg/bean_examples/LoanBean** in the Property Bindings dialog. The following binding expression appears in the New binding expression window.

```
#{asg$bean_examples$LoanBean.amount}
```

Page 157. (Table 5.3.) To specify property binding for component `interestRate`, choose `rate` under node **asg/bean_examples/LoanBean** in the Prop-

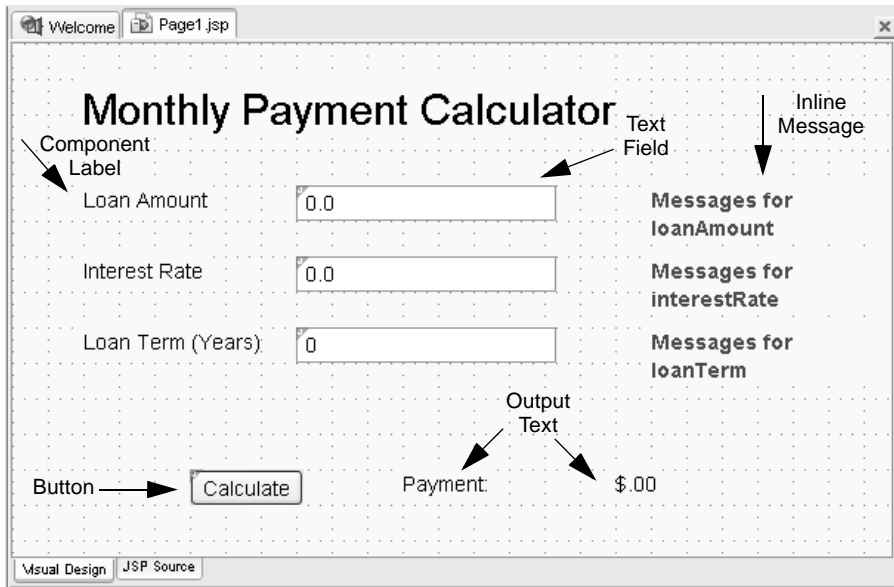


Figure A-5 Design canvas showing placement of components for project Payment1

erty Bindings dialog. The following binding expression appears in the New binding expression window.

```
{asg$bean_examples$LoanBean.rate}
```

Page 158. (Table 5.4.) To specify property binding for component `loanTerm`, choose `years` under node `asg/bean_examples/LoanBean` in the Property Bindings dialog. The following binding expression appears in the New binding expression window.

```
{asg$bean_examples$LoanBean.years}
```

Page 159. (Table 5.5.) To specify property binding for component `cost`, choose `payment` under node `asg/bean_examples/LoanBean` in the Property Bindings dialog. The following binding expression appears in the New binding expression window.

```
{asg$bean_examples$LoanBean.payment}
```

Chapter 6

Project Google1

Page 167. (Step 3.) If you select tab URL, you are prevented from selecting radio button Copy. Selecting radio button Link works fine. If you'd like to use a file reference instead, select tab File and browse to **FieldGuide/Examples/WebServices/images** in the book's download directory. Select file **Logo_40wht.gif**. Creator will copy the file into your project's Resources directory.

Project Google2

Page 182. (Step 4.) Creator displays `lengthValidator1` in the Properties window for the `validator` attribute for component `searchString`.

Project Google4

Page 194. (Step 4, top of page.) In the Properties window, the `image` attribute for a button component is under Appearance, not Advanced.

Page 194. (Step 3, bottom of page.) In the Properties window, the `image` attribute for a button component is under Appearance, not Advanced.

Page 195. (Step 1.) The package name under node Java Sources is **google1**, not **google4**.

Page 196. (Listing 6.4.) The package name for class `SessionBean1` is `google1`, not `google4`.

Chapter 7

Project Music1

Page 219. (Figure 7-5.) The Table Layout dialog displays **tracksRowSet** in the Get data from: window.

Page 225. (Important Creator Tips.) The method for reporting warnings is `warn()`, not `warning()`.

Project Music2

Page 232. (Figure 7–13.) The Table Layout dialog displays **recordingartists-RowSet** in the Get data from: window.

Project MusicUpdate

Page 235. (Steps 3-5.) It is not necessary to add the output text component `sqlMessage`. Instead, we add a message list component for reporting error and informational messages to the user (see Step 4 on Page 239).

Page 237. (Step 3.) Change the default component type to Text Field (editable).

Page 237. (Figure 7–15.) The Table Layout dialog displays **musiccategories-RowSet** in the Get data from: window. The Component type: displays Text Field (editable).

Page 239. (Modify the RowSet's Attribute.) It is not necessary to modify the rowset's concurrency attribute since you are modifying the database through the data table component. Creator automatically sets the `dataTableModel`'s attributes for you.

Project MusicInsert

Page 243. (Step 1.) In the Properties window for component `artistName`, the `validator` attribute is set to `lengthValidator1`.

Page 244. (Step 1, towards bottom of page.) In the Properties window for component `artistNotes`, the `validator` attribute is set to `lengthValidator2`.

Project MusicDelete

Page 253. (Step 3.) It is not necessary to add the output text component `sqlMessage`. Instead, we add a message list component for reporting error and informational messages to the user (see Step 8 on Page 255).

Chapter 8

Project Login3

Page 268. (Steps 2-3.) You can add a `<f:loadBundle/>` tag to your page's JSF source by dragging the `<f:loadBundle>` tag from the palette. Click the tab

labeled “Advanced,” select the `<f:loadBundle>` tag, and drop onto the page in the design view. Now click the JSP Source at the bottom. You’ll see that Creator has added the following tag.

```
<f:loadBundle>Load Bundle</f:loadBundle>
```

Edit the JSP source so that the `<f:loadBundle>` tag is as follows:

```
<f:loadBundle basename="asg.messages.login3" var="messages"/>
```

Page 270. (Steps 2 and 3.) The label’s text is replaced by the italicized word *Text*.

Page 278. (Step 3.) The component name is `password`, not `passWord`.

Page 278. (Step 4.) Component `password` is already set to `required=false` (unchecked).

Page 279. (Step 2.) Expand nodes `src > web > WEB-INF`.

Project Color1

Page 288. (Steps 2-3.) Add the `<f:loadBundle/>` tag to your page’s JSF source by dragging the `<f:loadBundle>` tag from the palette. Click the tab labeled “Advanced,” select the `<f:loadBundle>` tag, and drop onto the page in the design view. Now click the JSP Source at the bottom. You’ll see that Creator has added the following tag.

```
<f:loadBundle>Load Bundle</f:loadBundle>
```

Edit the JSP source so that the `<f:loadBundle>` tag is as follows:

```
<f:loadBundle basename="asg.messages.color1" var="messages"/>
```

Chapter 9

Project Payment1 (Debugging)

Page 311. (Step 7.) Right-click the same line number as the `getPayment()` method in Step 5 (line 107 in Figure 9–3).

Page 313. (Creator Tip.) Click the '+' on any field, then click the rounded box by the value to modify.

Page 319. (Steps 4 and 5.) If `java.util` and `MissingResourceException` do not appear in the dropdown list, type them in.

Page 322. (Creator Tip.) Right-click on Deployment Server in the Server Navigator window. Select View Server Log to see the server log file.

Page 324. (Top of page.) Right-click on Deployment Server in the Server Navigator window. Select View Server Log to see the server log file. The output from the server log file on your system may be slightly different.

A.3 Data Table 1

The rest of this document shows you how to use a data table component to display an array of Strings (Data Table 1), an array of JavaBeans components (Data Table 2), and the search result from the Google Web Services example (Google3Alt and Google4Alt).

Data Table Component

The data table component is one of the more complex JSF components in Creator's components palette. We provide an overview of this component assuming that you will bind it to a database rowset. However, the data table component can be bound to Java data structures such as arrays or an instance of `java.util.List`. Because Creator provides all of the binding work for you when you use a database rowset, it is not readily clear how to use the data table component with an array or a List object.

In this section, we're going to explore using the data table component with arrays. We'll follow the style of the Creator Field Guide and give you step by step instructions for building these projects yourself. You can also download the projects (they're included in Creator download zip file).

Create a New Project, Set Title and Style Sheet

This is a simple web application that uses a data table component to display a list of names. You'll store the list of names as Strings in an array. You can keep the list of arrays in application scope, since you build it once and then simply access it in a read-only mode. This means that the data are suitable to store in an application-wide shareable bucket, `ApplicationBean1`. (Recall that Creator preconfigures `ApplicationBean1` for you as a managed bean in application

scope.) For further information on the scope choices for web applications, see “Scope of Web Applications” on page 128 of the Creator Field Guide.

1. From Creator’s Welcome Page, select button Create New Project. Specify project name **DataTable1**. The selected type is Default J2EE Web Application. Click OK.
2. Creator comes up in the design view of the editor pane. Click anywhere in the middle of the design canvas.
3. Select **Title** in the Properties window. Change the title to **Data Table Example1** and finish by pressing **<Enter>**.
4. In the Project Navigator window, open node Resources and double-click **stylesheet.css**. Creator opens the default style sheet in the editor pane.
5. Copy and paste the contents of file **FieldGuide/Examples/Resources/stylesheet.css**, replacing the default style sheet. The new style sheet sets all text (in body) to `font-family Helvetica` and adds a style rule called `.heading`.
6. Select File > Save All from the top menu bar (or click the double-floppy Save All icon on the icon tool bar) to save the modifications made to the project thus far.

Add an Output Text Component

You’ll need an output text component to put a heading on the page.

1. From the JSF Standard Component palette, select Output Text component and drag it to the top of the page.
2. When you drop it onto the design canvas, it remains selected and you can begin typing its value. Type in **Data Table Example**. Finish with **<Enter>**. (Don’t resize the component; Creator will stretch it to fit the text.)
3. Under Appearances in the Properties window, click the small editing box opposite attribute `styleClass`. Creator pops up a style class selection dialog as shown in Figure A-6. Select style class **heading**, click the **>** button, and finish with OK.

The output text should now appear with its new style characteristics on the design canvas.

Add a String[] Property to ApplicationBean1

You’re going to create an array of Strings to store the names and add it to managed bean `ApplicationBean1` as a *property*. This will enable JSF to automatically instantiate it when it instantiates `ApplicationBean1`. It will also make it available to the GUI components as an `ApplicationBean1` property.

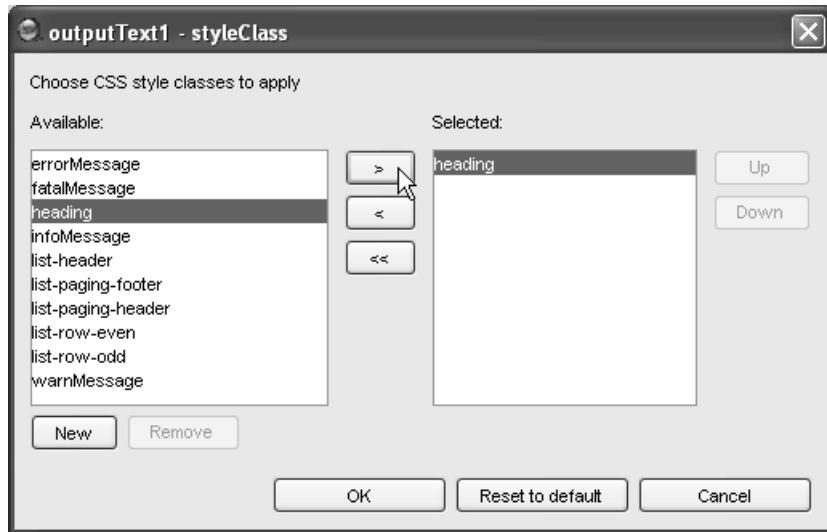


Figure A-6 Style class selection dialog

1. In the Project Navigator window, expand the Java Sources folder and then the project's default package folder. You'll see the Java page bean source for your project, **Page1.java**.
2. You'll also see "template" beans **ApplicationBean1.java** and **SessionBean1.java** for beans at application and session scope, respectively.
3. Right-click the file node **ApplicationBean1.java** and select Add > Property. This pops up the New Property Pattern dialog (see Figure A-7).
4. Fill in the dialog as follows. Under Name specify **names**, under Type specify **String[]**, and under Mode, select **Read Only**.



Creator Tip

Name and Type are case sensitive. Also make sure that type is `String[]` (which designates an array of Strings). You'll have to type in `String[]` since it is not in the dropdown list.

5. Make sure that options Generate Field and Generate Return Statement are checked. Click OK to add property names to ApplicationBean1.
6. Still in the Project Navigator window, double-click the file node **ApplicationBean1.java**. This brings up the file in the Java source editor.

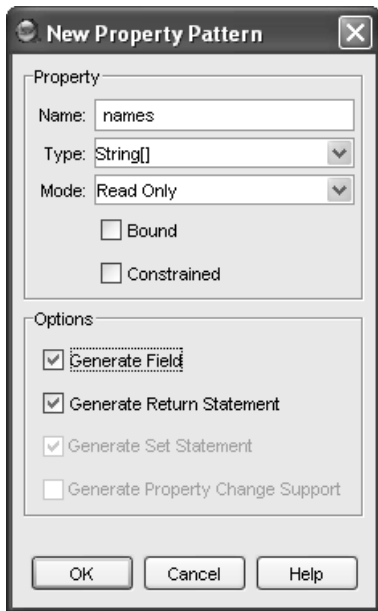


Figure A-7 New Property Pattern dialog

Page down to the end of the file. Here is the getter method Creator generated.

```
/**
 * Getter for property names.
 * @return Value of property names.
 */

public String[] getNames() {
    return this.names;
}
```

7. Now expand the folded code entitle Creator-managed Component Definition. Here is the instance variable Creator declared.

```
/**
 * Holds value of property names.
 */
private String[] names;
```

Now you'll add Java code that instantiates (with operator `new`) the names object and fills the array with Strings.

1. In the Java source editor (you're still editing file **ApplicationBean1.java**), add instantiation with operator `new` for property names inside the `ApplicationBean1()` constructor. Copy and paste from your Creator book's **FieldGuide/Examples/Beyond/snippets/DataTable1_init.txt** and add the code after the comment, as shown. The added code is bold.

```
// Additional user provided initialization code  
names = new String[] {  
    new String("Abigail"),  
    new String("Benjamin"),  
    new String("Carlton"),  
    new String("Denise"),  
    new String("Everett")  
};  
}
```

2. Save these modifications by selecting File > Save All from the top menu bar (or click the double-floppy Save All icon on the icon tool bar).

It's a good idea to check the status of your project as you make changes and add components and properties. Check that the `names` property appears as a Bean Pattern under `ApplicationBean1`, as follows.

1. In the Project Navigator window, expand node `ApplicationBean1`, class `ApplicationBean1`, and Bean Patterns.
2. There should be one bean pattern listed, `names`.
3. Select `names` under Bean Pattern and look at the Properties window. You'll see that its type is listed as `String[]`, it is read only, and its getter is `getNames()`.

Add a Data Table Component

It's time to add the data table component to your web page and bind it to the String array property you just created.

1. Return to the design canvas by selecting the tab labeled **Page1.jsp** at the top of the editing pane.
2. From the JSF Standard Components palette, select Data Table and drop it onto the design canvas under the output text component you added earlier.

3. Creator builds a data table component and configures it with 3 columns, 4 rows, and headers as shown in Figure A–8.

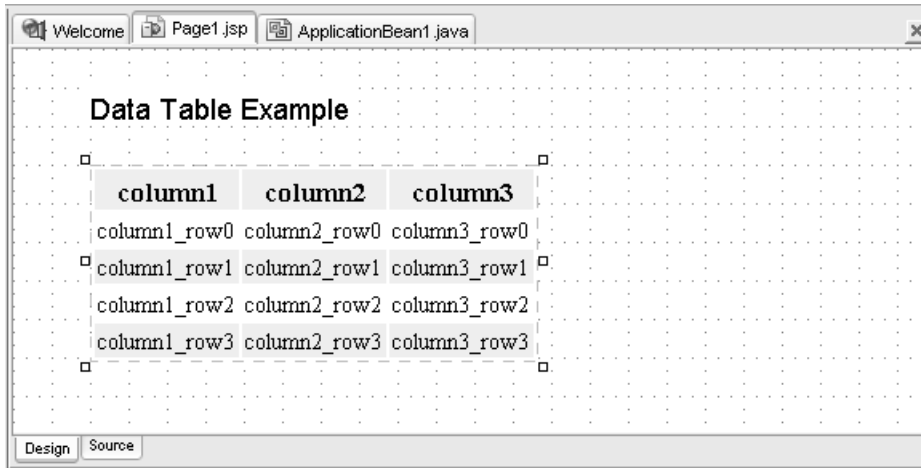


Figure A–8 Default configuration for data table component

4. In the Properties window for the data table component, set attribute bgcolor to **lightcyan**, border to **3**, cellspacing to **3**, and cellpadding to **3**.
5. Make sure that the data table component is selected, right-click, and select Property Bindings. Creator pops up the Property Bindings dialog as shown in Figure A–9.
6. From the window Select bindable property choose **value Object**. (It should already be selected.)
7. From the window Select binding target, expand node ApplicationBean1 and select **names**.
8. Click Apply. Under Current binding for value property you should see the reference expression `#{ApplicationBean1.names}`.
9. Click Close. The data table component still has 3 columns defined, but you only see the headings.

Now you're going to modify the table layout for the data table component.

1. The data table component should still be selected. Right-click and select Table Layout.
2. Creator brings up the Table Layout dialog. Note that the field labeled Get data from contains property names from ApplicationBean1.
3. Remove column2 and column3 by selecting them and clicking the left arrow (<).

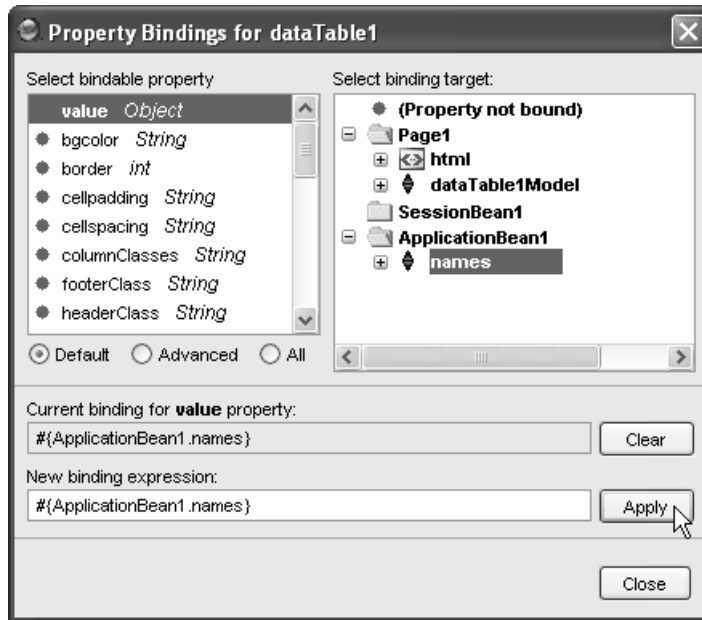


Figure A–9 Property Bindings dialog

4. Click Apply. The design canvas should now show a single column only.
5. Still in the Table Layout dialog under Displayed, click column1, as shown in Figure A–10.
6. For Header text, specify **Names**.
7. For Value, specify `{currentRow}`.
8. Click Apply followed by OK. The heading in the data table component should now say Names.

You're now ready to deploy and run the application. But first, let's look at some of the data table's nested components to see how Creator configured them.

1. In the design canvas, select the data table component (check to make sure you've selected component `dataTable1` by looking at the Application Outline view).
2. Scroll down the Properties window and look for category Data. You'll see that attribute value is set to `{ApplicationBean1.names}` and attribute var is set to `currentRow`.

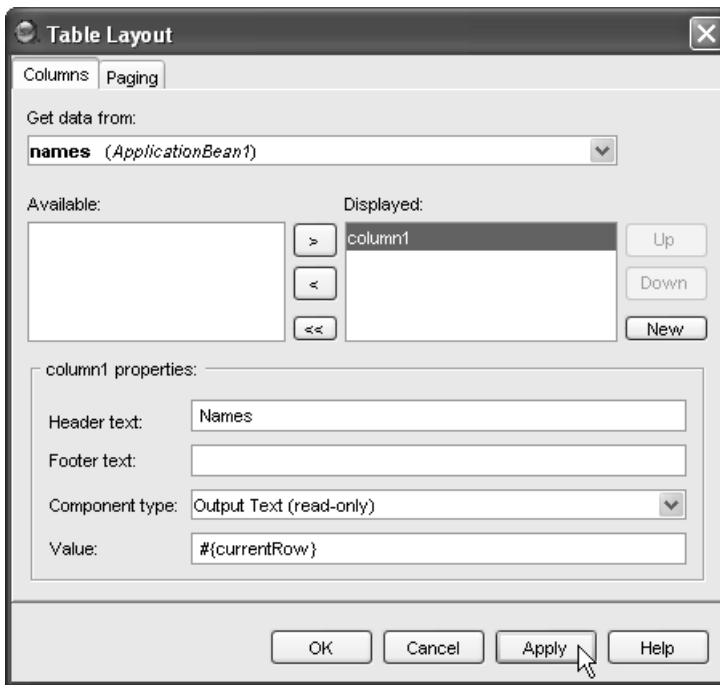


Figure A-10 Configuring the data table component with the Table Layout dialog

Attribute `value` is the object that the data table is bound to. When you obtain data from a data base rowset, this will be set to the appropriate `RowSet` object (such as `tracksrowset`). When the data comes from a Java object, this will be set to the appropriate bean or property name. Here, we set it to the `names` array, which is a property of `ApplicationBean1`.

Attribute `var` is set to `currentRow`. This attribute provides a name that Creator uses in reference expressions when it iterates through a rowset object, an array, or a `List` object. If you bind the data table to a simple array as we have done, then `currentRow` refers to each object in the array.

If the array contains objects which have properties, you can use `currentRow` to access these properties. We'll show you how to do this in the next example.

1. Now click on the data table component until component `outputText3` is selected in the Application Outline view.
2. In the Properties window, you'll see that its `value` attribute is set to the header "Names." When you specify the Header text (as shown in Figure A-10), Creator uses an output text component to store the text.
3. In the Application Outline view, select component `outputText2`.

4. In the Properties window, attribute `value` is set to reference expression `#{currentRow}`. Because the data table component is bound to a simple array of Strings, expression `#{currentRow}` evaluates to each String object in the array when JSF iterates through to fill the data table.

Deploy and Run

Deploy and run the application. Select the green chevron on the tool bar or select **Build > Run Project** from the main menu. Figure A-11 shows the application running with the data table filled with the names array you put in `ApplicationBean1`.

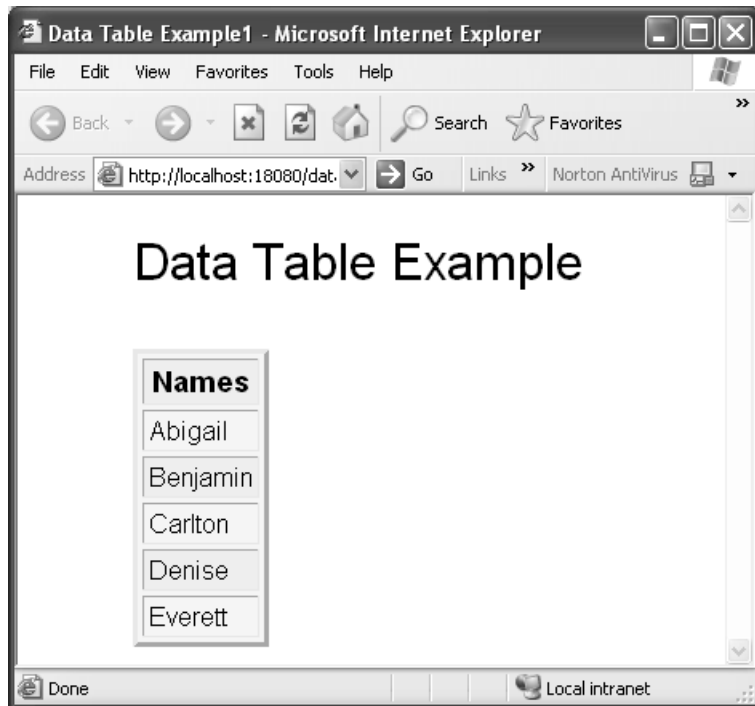


Figure A-11 Display an array of Strings in a data table component

A.4 Data Table 2

In the previous example, you learned how to bind a data table component to a simple array of Strings and display the array's elements in the rows of the table. In this example you will bind a data table component to an array, but the array consists of a JavaBeans component called Months that contains properties. You will then bind properties of bean Months to the data table's columns. Let's begin.

Create a New Project, Set Title and Style Sheet

1. From Creator's Welcome Page, select button Create New Project. Specify project name **DataTable2**. The selected type is Default J2EE Web Application. Click OK.
2. Creator comes up in the design view of the editor pane. Click anywhere in the middle of the design canvas.
3. Select **Title** in the Properties window. Change the title to **Data Table Example2** and finish by pressing **<Enter>**.
4. In the Project Navigator window, open node Resources and double-click **stylesheet.css**. Creator opens the default style sheet in the editor pane.
5. Copy and paste the contents of file **FieldGuide/Examples/Resources/stylesheet.css**, replacing the default style sheet. The new style sheet sets all text (in body) to font-family Helvetica and adds a style rule called `.heading`.
6. Select File > Save All from the top menu bar (or click the double-floppy Save All icon on the icon tool bar) to save the modifications made to the project thus far.

Add an Output Text Component

You'll need an output text component to put a heading on the page.

1. From the JSF Standard Component palette, select Output Text component and drag it to the top of the page.
2. When you drop it onto the design canvas, it remains selected and you can begin typing its value. Type in **Table of Months**. Finish with **<Enter>**. (Don't resize the component; Creator will stretch it to fit the text.)
3. Under Appearances in the Properties window, click the small editing box opposite attribute `styleClass`. Creator pops up a style class selection dialog. Select style class **heading**, click the **>** button, and finish with OK.

The output text should now appear with its new style characteristics on the design canvas.

Create Java Source for Months.java

You'll now create a JavaBeans component Months. (To read up on the importance of JavaBeans components with Creator, see "What Is a Bean?" on page 125 in the Creator Field Guide.)

1. In the Project Navigator window, expand node Java Sources and the default package name, **datatable2**.
2. Select **datatable2**, right-click, and select New > Java class. Creator pops up the New Wizard - Java Class dialog.
3. Specify **Months** for the class name and click Finish. Creator generates file **Months.java** under the default package name.

Now you'll add three String properties to the Months JavaBeans component.

1. Right-click the file node **Months.java** and select Add > Property. This pops up the New Property Pattern dialog (see Figure A-12).



Figure A-12 New Property Pattern dialog

2. Fill in the dialog as follows. Under Name specify **english**, under Type specify **String**, and under Mode, select **Read/Write**.

3. Make sure that options Generate Field, Generate Return Statement, and Generate Set Statement are checked. Click OK to add property `english` to `Months`.
4. Repeat steps 1-3 above and add property `spanish` and property `german` (both Strings).
5. The file **Months.java** should be visible in the Java source editor. If it's not, from the Project Navigator window, double-click the file node **Months.java**.
6. Find the constructor and replace it with the code in **FieldGuide/Examples/Beyond/snippets/DataTable2_months_const.txt**, as shown. The replaced text is in bold.

```

/** Creates a new instance of Months */
public Months(String english, String spanish, String german) {
    this.english = english;
    this.spanish = spanish;
    this.german = german;
}

```

Add a Months[] Property to ApplicationBean1

Just as you added an array of Strings as a property to `ApplicationBean1` in the previous project example, you'll now add an array of `Months` (`Months[]`) to `ApplicationBean1`.

1. In the Project Navigator window, expand the Java Sources folder and then the project's default package folder.
2. You'll see "template" beans **ApplicationBean1.java** and **SessionBean1.java** for beans at application and session scope, respectively.
3. Right-click the file node **ApplicationBean1.java** and select Add > Property. This pops up the New Property Pattern dialog.
4. Fill in the dialog as follows. Under Name specify **monthTable**, under Type specify **Months[]**, and under Mode, select **Read Only**.
5. Make sure that options Generate Field and Generate Return Statement are checked. Click OK to add property `monthTable` to `ApplicationBean1`.

Now you'll add Java code that instantiates (with operator `new`) the `monthTable` object and fills the array with `Months` in file **ApplicationBean1.java**.

1. In the Project Navigator window, double-click file **ApplicationBean1.java**.
2. Add instantiation with operator `new` for property `monthTable` inside the `ApplicationBean1()` constructor. Copy and paste from your Creator book's

FieldGuide/Examples/Beyond/snippets/DataTable2_monthTable_init.txt and add the code after the comment, as shown. The added code is bold.

```
// Additional user provided initialization code
monthTable = new Months[] {
    new Months("January", "enero", "Januar"),
    new Months("February", "febrero", "Februar"),
    new Months("March", "marzo", "März"),
    new Months("April", "abril", "April"),
    new Months("May", "mayo", "Mai"),
    new Months("June", "junio", "Juni"),
    new Months("July", "julio", "Juli"),
    new Months("August", "agosto", "August"),
    new Months("September", "septiembre", "September"),
    new Months("October", "octubre", "Oktober"),
    new Months("November", "noviembre", "November"),
    new Months("December", "diciembre", "Dezember")
};
```

3. Save these modifications by selecting File > Save All from the top menu bar (or click the double-floppy Save All icon on the icon tool bar).

Now let's check that the `monthTable` property appears as a Bean Pattern under `ApplicationBean1`, as follows.

1. In the Project Navigator window, expand node `ApplicationBean1`, class `ApplicationBean1`, and Bean Patterns.
2. There should be one bean pattern listed, `monthTable`.
3. Select `monthTable` under Bean Pattern and look at the Properties window. You'll see that its type is listed as `Months []`, it is read only, and its getter is `getMonthTable()`.



Creator Tip

In order for the IDE to correctly "see" the Months class, you must build the Months.class file. To do this, build the project. From the top menu, select Build > Build Project (deploying the project is not necessary). Now close and reopen the project in Creator. The property monthTable should now be visible in the Property Binding dialog, which you'll use in the next section.

Add a Data Table Component

It's time to add the data table component to your web page and bind it to the `Months` array property you just created.

1. Return to the design canvas by selecting the tab labeled **Page1.jsp** at the top of the editing pane.
2. From the JSF Standard Components palette, select Data Table and drop it onto the design canvas under the output text component you added earlier.
3. In the Properties window for the data table component, set attribute bgcolor to **lightcyan**, border to **3**, cellpadding to **3**, and cellspacing to **3**.
4. Make sure that the data table component is selected, right-click, and select Property Bindings. Creator pops up the Property Bindings dialog as shown in Figure A-13.¹

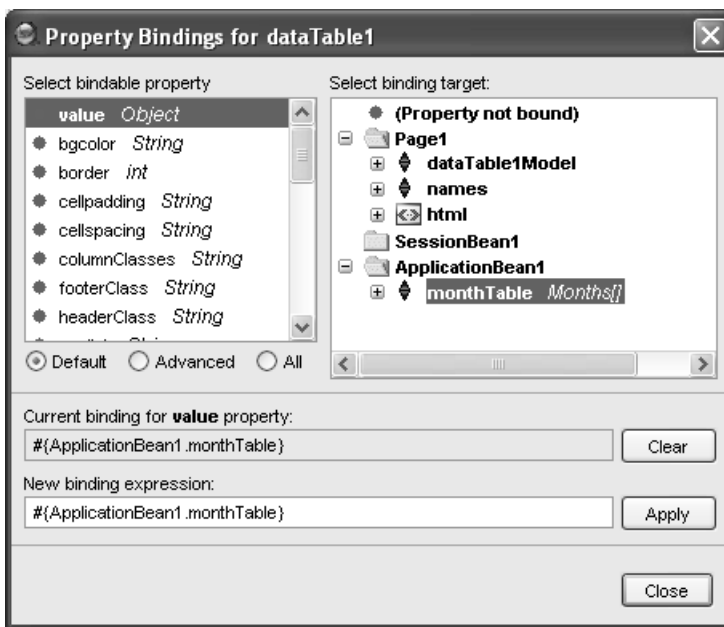


Figure A-13 Property Bindings dialog

5. From the window Select bindable property choose **value Object**. (It should already be selected.)
6. From the window Select binding target, expand node ApplicationBean1 and select **monthTable**.

1. If monthTable does not appear as a property under ApplicationBean1 in the Property Bindings dialog, build the project (use Build > Build Project from the top menu). Now close and reopen the project in the IDE. The monthTable property should now be visible in the Property Bindings dialog.

Appendix A Beyond the Book

7. Click Apply. Under Current binding for value property you should see the reference expression `#{ApplicationBean1.monthTable}`.
8. Click Close. The data table component still has 3 columns defined, but you only see the headings.

Now you're going to modify the table layout for the data table component.

1. The data table component should still be selected. Right-click and select Table Layout.
2. Creator brings up the Table Layout dialog. Note that the field labeled Get data from contains property `monthTable` from `ApplicationBean1`.
3. In the Table Layout dialog under Displayed, click `column1`, as shown in Figure A-14.

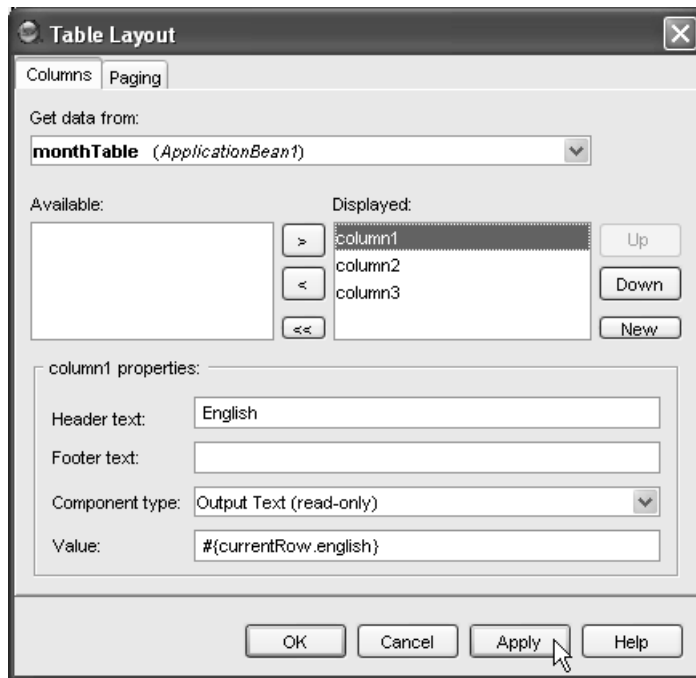


Figure A-14 Configuring the data table component with the Table Layout dialog

4. For Header text, specify **English**.
5. For Value, specify `#{currentRow.english}`.
6. Click Apply. The heading for the first column in the data table component on the design canvas should reflect the new text.

7. Repeat these steps for column2 and column3. For column2, use Header text **Spanish** and specify `#{currentRow.spanish}` for the Value.
8. For column3 use Header text **German** and specify `#{currentRow.german}` for the Value.
9. Click Apply and OK. (If you don't click Apply, the changes you made are discarded. Make sure the changes appear on the design canvas.)

The data table component on the design canvas consists of three columns with the header text. There are no rows displayed. Again, you'll want to look at the Properties window for the data table component and the nested output text components.

For the data table component, attribute value is set to `#{ApplicationBean1.monthTable}`, which refers to property `monthTable` in the default managed bean for application scope. Attribute var is set to `currentRow`, which is the name you use to access any properties in JavaBeans component `monthTable`.

For the nested output text components, (select `outputText2`, for example), its value is set to `#{currentRow.english}`, which specifies property `english` of the bound data object for `dataTable1`.

Deploy and Run

Deploy and run the application. Select the green chevron on the tool bar or select **Build > Run Project** from the main menu. Figure A-15 shows the application running with the data table filled with the names array you put in `ApplicationBean1`.

Creator Tip

*If you open project `DataTable2` from the Example Projects in the zip file, the data table component will be rendered in red. This is because the project was "cleaned" to save space, removing the `Months.class` file. To properly render the data table component, build the project. From the top menu, select **Build > Build Project** (deploying the project is not necessary). Now close and reopen the project in Creator. The data table component will now be correctly displayed in the design canvas window.*



A.5 Google 3 (Alternate)

In the previous two examples you built web applications that displayed data from arrays using a data table component. You bound the `value` attribute of the data table component to the array. In the first example, the (single-column)



The screenshot shows a Microsoft Internet Explorer browser window titled "Data Table Example2 - Microsoft Internet Explorer". The address bar shows "http://localhost:18080/dat.". The main content area displays a table titled "Table of Months". The table has three columns: "English", "Spanish", and "German". The rows list the months from January to December in each language.

English	Spanish	German
January	enero	Januar
February	febrero	Februar
March	marzo	März
April	abril	April
May	mayo	Mai
June	junio	Juni
July	julio	Juli
August	agosto	August
September	septiembre	September
October	octubre	Oktober
November	noviembre	November
December	diciembre	Dezember

Figure A-15 Display an array of Months in a data table component

data table displayed each element in the array of Strings. In the second example, you used an array of JavaBeans components, Months. Each bean in the array contains three properties and you bound these to each of the three columns in the data table component. To access properties in the array, you used the `var` attribute, which Creator preconfigures to `currentRow`.

Web applications Google3 (see “HTML with Output Text” on page 187) and Google4 (see “Displaying Multiple Pages” on page 192) use an output text component to display the results from the Google web search by building an HTML table programmatically. This works nicely and is a good way to go when you want programmatic control over the table’s contents. However, you can also use the JSF data table component and bind various embedded components to the Google search results returned in the `ResultElement` array (see “`GoogleSearchResult` public methods” on page 177 and “`ResultElement` public methods” on page 178). In this section, you’ll build an alternate Google3 web application that uses a data table component and property bindings to display the Google search results instead of an output text component built with HTML tags in the event handler.

Copy a Project

Let’s start with project **Google3** and copy it to create a new project called **Google3Alt**. This step is optional. If you don’t want to copy the project, skip this section and make modifications directly to project **Google3**.

1. Bring up project **Google3** in Creator. (Either open it from your Projects directory, or use the project from the book’s download, **FieldGuide/Examples/Projects/Google3**.)
2. Click anywhere inside the design canvas.
3. From the File menu, select Save Project As and provide the new name **Google3Alt**.
4. Deploy and run the application. Depending on the search criteria you provide, you should see an output page with up to 10 results, as shown in Figure 6–6 on page 191 of the Creator Field Guide text.²

Modify the Project

In project **Google3**, you built the results into an HTML table programmatically in method `buildTable()`. Instead, you’ll now build the results by binding the `ResultElement` array to a JSF data table component. First, you’ll make modifications to the page design. Then you’ll modify the Java code in **Page1.java**. Finally, you’ll add components to the data table and complete the property bindings.

1. Make sure that **Page1.jsp** is displayed in the design canvas.

2. Make sure you include your Google Web API’s License Key as the first argument of the Google search method. Otherwise, the application will throw an exception.

2. Select output text component `result` (either on the design canvas or in the Application Outline view). Right-click and select Delete. Creator deletes this component from the page.
3. Select inline message component `inlineMessage1`. Right-click and select Delete to delete this component.
4. From the JSF Standard Components palette, select Message List component and drag it to the design canvas. Drop it to the right of the text field component `searchString`. You'll use a message list component instead of the inline message component so that JSF will display error messages from multiple sources. Creator displays **List of all message summaries** in a bold, red font on the design canvas.

Now you'll make some modifications to the Java source page bean, **Page1.java**. There will be some syntax errors in this file since you deleted some components that are referenced in the code. You'll be either replacing or deleting this code.

1. Bring up **Page1.java** in the Java source editor. In the design canvas for **Page1.jsp**, click the mouse anywhere in the canvas background. Right-click and select Edit Page1 Java Source. Creator displays **Page1.java** in the Java source editor.
2. Remove method `buildTable()` from the source.
3. Remove the following declaration for variable `mySearchResult`.

```
private GoogleSearchResult mySearchResult;
```

(This will create some syntax errors which you'll fix when you make `mySearchResult` a property.)

4. In method `beforeProcessValidations()`, remove the following statement.

```
result.setValue("");
```

5. Replace method `search_action()` with the following code. Copy and paste file **FieldGuide/Examples/Beyond/snippets/google3Alt_search_action.txt**

from your Creator download directory. (Remember to supply your Google license key for the first argument.)

```
public String search_action() {
    // TODO Replace with your code
    try {
        mySearchResult =
            googleSearchServiceClient1.
                googlesearchportDoGoogleSearch(
                    "Your Google Key Here",

                    (String) searchString.getValue(),
                    0,10,true,"",true,"lang_en","","");
        timeCount.setValue("Search Time = " +
            mySearchResult.getSearchTime()

            + " Number of results returned = "
            + mySearchResult.getEstimatedTotalResultsCount());
        ResultElement[] myResults =
            mySearchResult.getResultElements();
    }

    catch (Exception e) {
        log("Remote Connect Failure", e);
        mySearchResult = null;
        error("Remote Site Failure" + e.getMessage());
    }
    return null;
}
```

You'll note that there are some subtle changes to method `search_action()`. First, the call to method `buildTable()` has been eliminated because you will use property bindings to display the search results. Second, the catch handler posts an error message with method `error()`. JSF will display this error message in the message list component that you just added. Third, variable `mySearchResult` is set to null in the catch handler. We set it to null so that the data table component won't be displayed when there are no results. (This is accomplished with the data table's `rendered` attribute, which we discuss shortly.)

Add Property `GoogleSearchResult` to Page1

In order to bind the results from the Google search to the data table component, you must make the `GoogleSearchResult` variable a property.

1. In the Project Navigator window, expand the Java Sources > `google1` nodes.

Appendix A Beyond the Book

2. Right-click on **Page1.java** and select Add > Property. Creator pops up the New Property Pattern dialog, as shown in Figure A-16.

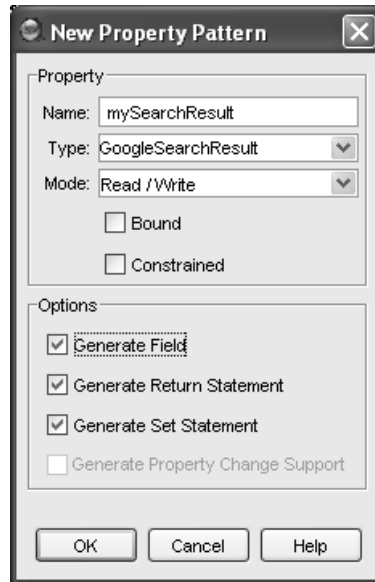


Figure A-16 Adding property **mySearchResult** to **Page1.java**

3. For Name, use **mySearchResult**, for Type, use **GoogleSearchResult**, and for Mode select the default **Read/Write**. Remember that Name and Type are case sensitive. Click OK.
4. Now return to **Page1.java** in the Java source editor. (If you're not already there, select the tab labeled **Page1.java** at the top of the editor pane. Creator brings up **Page1.java** in the Java source editor.)
5. Note that near the end of the file Creator added the setter and getter methods for property **mySearchResult** and the private declaration is inside the folded code entitled Creator-managed Component Definition.
6. Add the following initialization statement in the **Page1** constructor (after the comment as indicated). This initializes **mySearchResult** to null so that JSF will not display the data table component when the page first comes up in the browser.

```
// Additional user provided initialization code  
mySearchResult = null;
```

7. Save the changes you've made so far by selecting File > Save All from the top menu.

Add a Data Table Component

You'll use Creator's data table component to display the Google search results.

1. Select the tab labeled **Page1.jsp** at the top of the editor pane to bring up **Page1.jsp** in the design canvas.
2. From the JSF Standard Components palette, select component Data Table and drop it onto the page below the output text component `timeCount`. Align it on the left with the output text component. Creator displays a data table component with a default configuration of three columns and four rows.
3. Make sure that the data table component is selected (use the Application Outline view since the data table component contains nested column and output text components). Right-click and select Table Layout.
4. Select `column2` and remove it by clicking the left arrow (<).
5. Select `column3` and remove it by clicking the left arrow (<).
6. Click Apply (this will adjust the data table component on the design canvas).
7. Still in the Table Layout dialog, select `column1`. In the field labeled Header text, supply the following text (all on a single line).

```
Search Results ({Page1.mySearchResult.startIndex} to  
{Page1.mySearchResult.endIndex})
```

Click Apply and OK. This expression combines literal text with a JSF EL expression for the start index and end index values of property `mySearchResult`. (See Table 6.2 on page 177 for a description of methods `getEndIndex()` and `getStartIndex()` which return these property values.)

Configure the Data Table Component

One of the properties of the `GoogleSearchResult` object is a `ResultElement` array. Each `ResultElement` object contains a URL, a title, and a snippet. To display these, you'll use a hyperlink component for the URL and title and an output text component for the snippet. You'll also use a separate output text component to hold an HTML `
` element to start the snippet on its own line.

1. Make sure that the data table component is selected, right-click, and select Property Bindings. Creator pops up the Property Bindings dialog, as shown in Figure A-17.

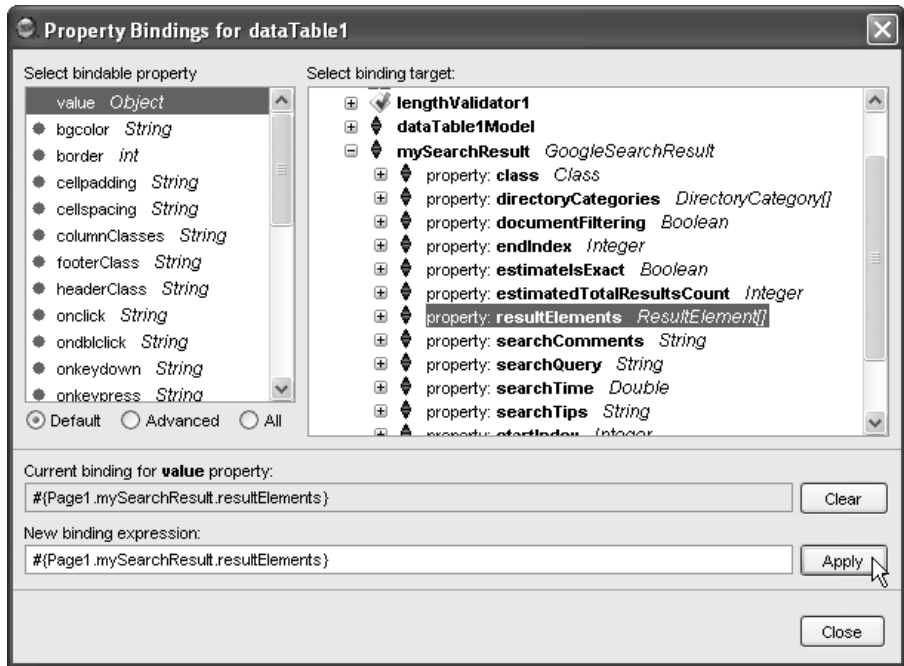


Figure A-17 Property Bindings dialog for dataTable1

- For Select bindable property, select **value Object** (this should already be selected).
- For Select binding target, expand **Page1** and **mySearchResult**. Choose property **resultElements**. Click Apply. Under Current binding for **value** property you should see the following expression.

```
{Page1.mySearchResult.resultElements}
```

- Click Close.
- Make sure that the data table component is still selected.
- In the Properties window, set attribute **bgcolor** to **lightcyan**, **border** to **3**, **cellpadding** to **3**, and **cellspacing** to **3**. (These are optional, cosmetic enhancements.)

Add Components to the Data Table's Column

Once you've bound the data table's `value` attribute to a property, you can reference properties within this property using the data table's `var` attribute `currentRow`. Thus `#{currentRow.URL}` is property URL, `#{currentRow.title}` is property title, and `#{currentRow.snippet}` is property snippet.

1. From the JSF Standard Components palette, select component `Hyperlink` and drop it on top of the `column1` component under `dataTable1` in the Application Outline view. This adds a hyperlink component and a nested output text component.

Creator Tip

Make sure that you drop the hyperlink component (and subsequent components that you'll add) on top of the `column1` component. If the component ends up in the wrong order, you can drag it on top of the `column1` component, which adds it to the end. Use this to reorder the components if necessary.



2. Make sure that the hyperlink component is selected. In the Properties window, change its `id` to `hyperlink_url` and its `value` to `#{currentRow.URL}`.
3. Now select the nested output text component, `hyperlink1Text`. In the Properties window, change its `id` to `title1` and its `value` to `#{currentRow.title}`. Make sure that attribute `escape` is *unchecked* (false).
4. From the JSF Standard Components palette, select component `Output Text` and drop it on top of component `column1` in the Application Outline view.
5. In the Properties window, change its `id` to `htm_formatter` and its `value` to `
`. Make sure that attribute `escape` is *unchecked* (false).
6. From the JSF Standard Components palette, select component `Output Text` and drop it on top of component `column1` in the Application Outline view.
7. In the Properties window, change its `id` to `snippet` and its `value` to `#{currentRow.snippet}`. Make sure that attribute `escape` is *unchecked* (false).
8. When you added header text to the column, Creator generated an output text component to hold the text. In the Application Outline view, select this component (its `value` attribute consists of the compound JSF expression including Search Results). Change its `id` attribute to `table_header`.

There is an unused output text component that Creator generated for you by default. Follow these steps to delete it.

1. Select the output text component `outputText1` nested under `column1` in the Application Outline view.

2. In the Properties window, click on the small editing square opposite attribute `value`. Creator pops up a dialog to edit the `value` attribute.
3. Select option Reset to default (at the bottom of the dialog).
4. Make sure this component is still selected. Right-click and choose Delete from the context menu. *Deleting this component is optional.*

Attribute Rendered

All JSF components that can be displayed have an attribute `rendered`, which is a boolean value. Select the data table component and in the Properties window you'll note that attribute `rendered` (under Advanced) is *checked* (true). If you deploy and run the application as it is, the initial page is rendered with an empty table and just the header displayed. A better solution is to render the data table component *only if* there are results to display.

1. Uncheck attribute `rendered` in the Properties window of the data table component. This generates the tag `rendered="false"` in the JSP source.
2. Select the tab labeled JSP Source at the bottom of the editor pane. This brings up **Page1.jsp** in the JSP editor.
3. Find the tags for the data table component and change the tag for attribute `rendered` to the following.

```
rendered="#{not empty Page1.mySearchResult}"
```

This sets attribute `rendered` to true if property `mySearchResult` is not empty. When you add the Java statement to initialize `mysearchResult` to null (and also to set it to null in the catch handler), you are able to use the JSF EL expression `not empty` with the data table's `rendered` attribute to control its display.

4. Click the tab labeled Visual Design to return to the design canvas.

Deploy and Run

It's time to deploy and run the project. Select Build > Run Project from the main menu (or select the green chevron from the icon tool bar). Any validation errors or exceptions generated from the remote web services site will appear in the message list component. Figure A-18 shows the application after returning from a search request with the results displayed in the data table component.

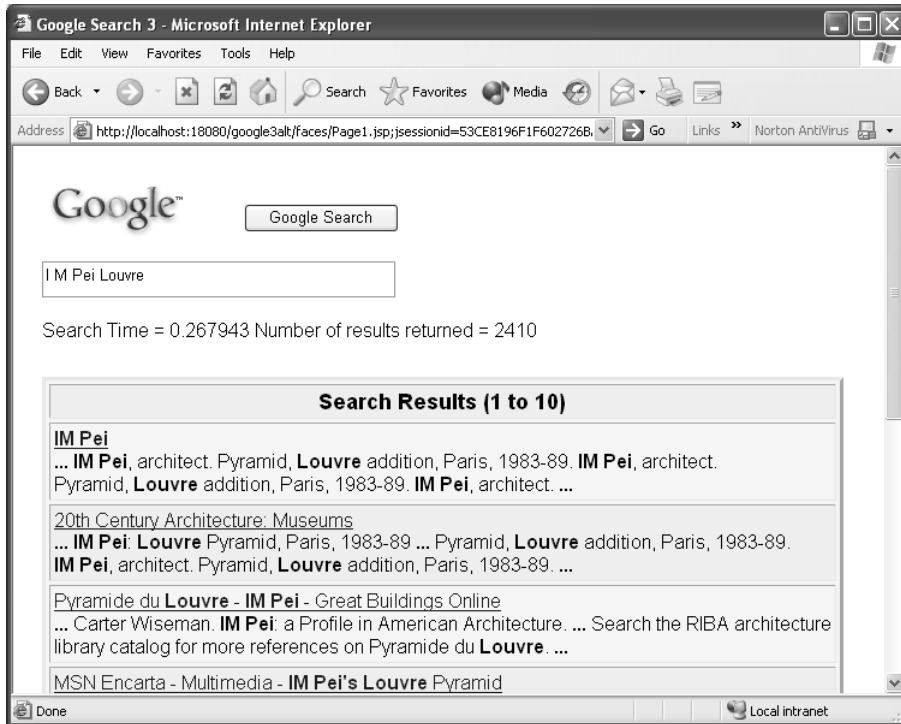


Figure A-18 Google Web Search application using JSF data table component

A.6 Google 4 (Alternate)

This section follows the Creator Field Guide text (see “Displaying Multiple Pages” on page 192), which adds two button controls with “arrow” images to display multiple pages of results from a Google search request. There’s only slight changes from the text, which the reader is encouraged to review. The mechanism for saving the index controls across multiple page requests is unchanged. The index controls are saved as properties in the preconfigured session bean.

Copy a Project

Copy **Google3Alt** and create a new project called **Google4Alt**. This step is optional. If you don't want to copy the project, simply skip this section and continue making modifications to the previous project.

1. Bring up project **Google3Alt** in Creator, if it's not already opened.
2. Click anywhere inside the design canvas.
3. From the File menu, select Save Project as . . . and provide the new name **Google4Alt**. You'll make changes to the **Google4Alt** project.
4. Change the project's title to **Google Search 4**.

Before you add components to project **Google4Alt**, look at Figure A-19, the design canvas for this project. You can see the placement of the buttons, the output text component, and the data table component. Also, the web services component, `googleSearchServiceClient1`, appears in the nonvisual tray at the bottom.

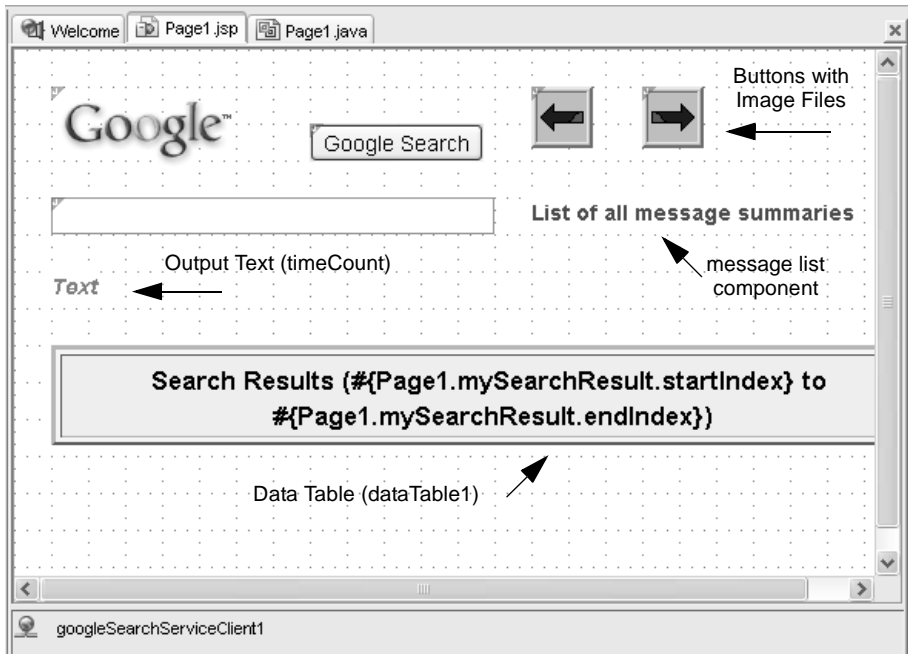


Figure A-19 Design canvas showing component layout for project Google4

Add Button and Image File

1. From the JSF Standard Components palette, select Button and drop it onto the design canvas. Place it to the right of the Google Search button as shown in Figure A-19. Change its `id` attribute to **prev1** by selecting `id` in the Properties window.

Creator Tip

You're changing the standard `id` that Creator uses because it's easier to keep track of the components in the Java page bean file. By using meaningful `id` names (such as `prev1` and `next1`), the associated action methods that Creator generates will have meaningful names, too.



2. In the Properties window, set attribute `title` to **Get the previous set of results** for its tooltip.
3. In the Properties window under Appearance, click the small editing box opposite attribute `image`. Creator pops up an image selection dialog.
4. Select the File tab and browse to directory **FieldGuide/Examples/Beyond/images** in this book's download directory. Select file **arrow.blue.left.gif**. Click OK. Creator copies the image file to the Resources folder and the left-arrow image should now appear on the design canvas.
5. With the button still selected, click the small editing box next to the `value` attribute. When an editing box pops up, click the Reset to default button. This removes the `value` attribute from the generated JSF tags for the `prev1` button.

Here's the JSP code that Creator generates for the `prev1` button with the `image` attribute set to the `.gif` file in the project's resources folder.

```
<h:commandButton binding="#{Page1.prev1}" id="prev1"
image="resources/arrow.blue.left.gif"
style="left: 48px; top: 144px; position: absolute"
title="Get the previous set of results"/>
```

A Second Button and Image File

Follow the same procedure to add a second button and a second image to the page.

1. From the JSF Standard Components palette, select Button and drop it onto the design canvas. Place it next to the left arrow button you just added. Change its `id` attribute to **next1** by selecting `id` in the Properties window.

2. In the Properties window, set attribute `title` to **Get the next set of results** for its tooltip.
3. In the Properties window under Appearance, click the small editing box opposite attribute `image`. Creator pops up an image selection dialog.
4. Select the File tab and browse to directory **FieldGuide/Examples/Beyond/images** in this book's download directory. Select file **arrow.blue.right.gif**. Click OK. Creator copies the image file to the Resources folder and the right-arrow image should now appear on the design canvas.
5. With the button still selected, click the small editing box next to the `value` attribute. When an editing box pops up, click the Reset to default button.

Deploy and Run

You might want to experiment with the placement of these newly added graphic components. Deploy and run project **Google4Alt**. (The arrow buttons won't do anything useful, but you should be able to display the 10 results as before.) Check the placement of the components and adjust them if necessary.



Creator Tip

Actually, the arrow buttons clear the page (why?). To see the search results again, click the Google Search button. The page is cleared because clicking an arrow button generates an action event, which initiates the JSF page life cycle process (see Figure 6–5 on page 185). You haven't specified any action, but the system proceeds through the different life cycle phases anyway. Before reaching the Process Validations phase, the system executes `beforeProcessValidations()`, which clears the page. Also, the data table component is not displayed since its rendered attribute is false.

Add SessionBean1 Properties

You will soon add control variables to the **Page1.java** file. These values keep track of the current index and other controls you need for displaying more than the first page of results. To maintain these values across page requests, you add them to the `SessionBean1` managed bean as *properties*. This automatically puts them in session scope. Here are the steps to add properties to `SessionBean1`.



Creator Tip

Creator allows you to interactively add properties to its managed beans (`SessionBean1` for example). However, in this case, it's easier to simply copy and paste the code directly into the Java source file.

1. From the Project Navigator window, select Java Sources > google1.
2. Double-click file **SessionBean1.java**. This brings it up in the Java source editor.
3. Add instance variables with their setters and getters for the properties referenced in **Page1.java**. Copy and paste from file **FieldGuide/Examples/Beyond/snippets/google4Alt_sessionbean.txt**. (Add the code after the `SessionBean1()` constructor as shown. The added code is bold.)

Listing A.1 SessionBean1.java

```
package google1;
import com.sun.jsfcl.app.*;

public class SessionBean1 extends AbstractSessionBean {
    public SessionBean1() {
        . . . code omitted . . .
    }

    // State variables saved in session scope
    private Integer startIndex = new Integer(0);
    public Integer getStartIndex() { return startIndex; }
    public void setStartIndex(Integer si) { startIndex = si; }

    private Integer currentCount = new Integer(0);
    public Integer getCurrentCount() { return currentCount; }
    public void setCurrentCount(Integer cc) {
        currentCount = cc; }

    private Integer totalCount = new Integer(0);
    public Integer getTotalCount() { return totalCount; }
    public void setTotalCount(Integer tc) { totalCount = tc; }
    . . .
}
```

Specify the Action Code

When a user clicks the right-arrow button, the web application should display the next 10 results from the Google search. Conversely, clicking the left-arrow button displays the previous 10 results. Because you'll make similar calls to the Google search API, you should place this code in its own method. Furthermore, you need to keep track of the current start index and previous index so that you know how to access the results data when the user clicks right arrow or left arrow.

1. To return to the design canvas for Page1, click the tab labeled **Page1.jsp** at the top of the editor pane.
2. Select the right-arrow button, `next1`, and double-click.
3. Creator brings up the Java page bean file, **Page1.java**, and displays the generated event handler, `next1_action()`.
4. To keep track of the index variables and the result count information that Google returns, you'll need integer control variables. Scroll up the **Page1.java** file until you find the Creator-generated method `afterRenderResponse()`. Immediately after this method (shown below), add the following declarations. Copy and paste file **FieldGuide/Examples/Beyond/snippets/google4Alt_variables.txt**.

```
protected void afterRenderResponse() {
}

private int startIndex = 0;
private int prevIndex = 0;
private int currentCount = 0;
private int totalCount = 0;
```

The `startIndex`, `currentCount`, and `totalCount` integer variables are saved and restored in session scope for the action handlers. To do this, you'll use methods `saveState()` and `restoreState()` and the `SessionBean1` object that Creator generates for you from **SessionBean1.java**.

Let's look at the `saveState()` method first. This method calls setters to store `startIndex`, `currentCount`, and `totalCount` as equivalently named properties in the `SessionBean1` object.

Listing A.2 Method `saveState()`

```
private void saveState() {
    getSessionBean1().setStartIndex(new Integer(startIndex));
    getSessionBean1().setCurrentCount(
        new Integer(currentCount));
    getSessionBean1().setTotalCount(new Integer(totalCount));
}
```

Note that `getSessionBean1()` is what you use to access the session object in your page bean. You can use `SessionBean1` to store your own data as needed in session scope.³

-
3. To store session data, be sure to use `Integer` (one of Java's wrapper classes) instead of an `int` primitive type. `Integer` allows property binding between components and JavaBeans properties.

Next, let's look at `getState()`. This method calls getters to retrieve the equivalently named properties from the `SessionBean1` object.

Listing A.3 Method `getState()`

```
private void getState() {
    startIndex = getSessionBean1().getStartIndex().intValue();
    currentCount =
        getSessionBean1().getCurrentCount().intValue();
    totalCount = getSessionBean1().getTotalCount().intValue();
}
```

1. Add the `saveState()` and `getState()` methods immediately after the variable declarations you just added in **Page1.java**. Copy and paste from **FieldGuide/Examples/Beyond/snippets/google4Alt_save_restore.txt**.
2. Most of the code that resides in the `search_action()` event handler can be pulled out and placed in a method that all three action event handlers will call. Let's call this new method `doSearch()`. To create the `doSearch()` method, use **FieldGuide/Examples/Beyond/snippets/google4Alt_doSearch.txt** and place it directly before the `search_action()` method (near the end of the Java page bean file).

Listing A.4 Method `doSearch()`

```
public void doSearch(int start) {
    try {

        mySearchResult =
            googleSearchServiceClient1.
                googlesearchportDoGoogleSearch(
                    "Your Google Key Here",
                    (String)searchString.getValue(),
                    start,10,true,"",true,"lang_en","","");

        totalCount =
            mySearchResult.getEstimatedTotalResultsCount();

        timeCount.setValue("Search Time = " +
            mySearchResult.getSearchTime()
            + " Number of results returned = "
            + totalCount);
    }
}
```

Listing A.4 Method doSearch() *(continued)*

```

        ResultElement[] myResults =
            mySearchResult.getResultElements();
        currentCount = myResults.length;
    }

    catch (Exception e) {
        log("Remote Connect Failure", e);
        mySearchResult = null;
        error("Remote Site Failure" + e.getMessage());
    }
}

```

(Remember, Your Google Key Here needs to be replaced by the licensed key supplied by Google when you register to access their service.)

3. The `search_action()` method is now simpler, since all that's required is to reset the index control variables and call `doSearch()`. Copy and paste from file **FieldGuide/Examples/Beyond/snippets/google4Alt_search.txt** to *replace* this method. Here's the new code. Note the call to `saveState()` to save the index variables in the session object before returning.

Listing A.5 Method search_action()

```

public String search_action() {
    // TODO Replace with your code
    startIndex = 0;

    prevIndex = 0;
    doSearch(startIndex);
    saveState();
    return null;
}

```

Clicking the right-arrow button returns the next set of results from Google. To effect this return, update the start parameter (see Table 6.1 on page 176) of the `doGoogleSearch()` method. Also note that the code in the action handler `next1_action()` is similar to the code in the above `search_action()`. You just need to check for upper limits in the index control variables.

4. Add code to the `next1_action()` event handler. Copy and paste from file **FieldGuide/Examples/Beyond/snippets/google4Alt_next1.txt**. Here's the code. Note that in this method, it's necessary to call `getState()` to set the index variables from the session object before using them. Likewise, `saveState()`

saves the index variables in the session object before returning. (The added code is bold.)

Listing A.6 Method next1_action()

```
public String next1_action() {
    // TODO Replace with your code
    getState();
    prevIndex = startIndex;
    startIndex = startIndex + currentCount;

    if (startIndex >= totalCount || startIndex >= 1000) {
        startIndex = prevIndex;
        prevIndex -= currentCount;
    }

    doSearch(startIndex);
    saveState();
    return null;
}
```

Now let's add a `prev1_action()` method to handle action events associated with the left-arrow button.

1. Return to the design canvas by selecting the tab labeled **Page1.jsp** above the editor pane.
2. Select the left arrow button `prev1` and double click. This creates the event handler method `prev1_action()` in the Java page bean file for you and places the cursor at the beginning of the method.
3. Add the following code to the empty `prev1_action()` method. Copy and paste from file **FieldGuide/Examples/Beyond/snippets/google4Alt_prev1.txt**. (The added code is bold.)

Listing A.7 Method prev1_action()

```
public String prev1_action() {
    // TODO Replace with your code
    getState();
    prevIndex = startIndex - currentCount;
    startIndex = prevIndex;

    if (startIndex <= 0) {
        startIndex = 0;
        prevIndex = 0;
    }
}
```

Listing A.7 Method `prev1_action()` (*continued*)

```

doSearch(startIndex);
saveState();
return null;
}

```

You'll note that the structure of `prev1_action()` is similar to the `next1_action()` method.

Deploy and Run

Deploy and run project **Google4Alt**. You should be able to page through multiple result sets by using the arrow graphics "right" and "left." Figure A-20 shows the third page of a result set.

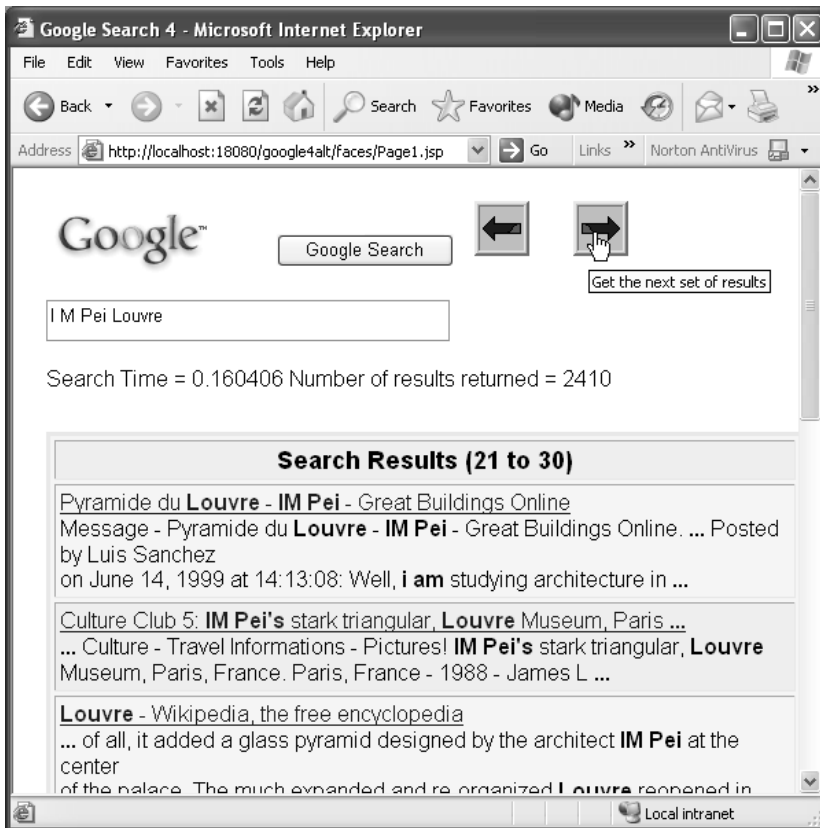


Figure A-20 The Google Web Search application displaying the third page of results