DATA TABLES

Topics in This Chapter

Chapter 5

Classic web applications deal extensively in tabular data. In the days of old, HTML tables were preferred for that task, in addition to acting as page layout managers. That latter task has, for the most part, been subsequently rendered to CSS, but displaying tabular data is still big business.

This chapter discusses the `h:dataTable` tag, a capable but limited component that lets you manipulate tabular data.

NOTE: The `h:dataTable` tag represents a capable component/renderer pair. For example, you can easily display JSF components in table cells, add headers and footers to tables, and manipulate the look and feel of your tables with CSS classes. However, `h:dataTable` is missing some high-end features that you might expect out of the box. For example, if you want to sort table columns, you will have to write some code to do that. See "Sorting and Filtering" on page 203 for more details on how to do that.

The Data Table Tag—`h:dataTable`

The `h:dataTable` tag iterates over *data* to create an HTML *table*. Here is how you use it:

```
<h:dataTable value='#{items}' var='item'>
   <h:column>
```

```
      <%-- left column components --%>
      <h:output_text value='#{item.propertyName}'/>
   </h:column>

   <h:column>
      <%-- next column components --%>
      <h:output_text value='#{item.anotherPropertyName}'/>
   </h:column>

   <%-- add more columns, as desired --%>
</h:dataTable>
```

The `value` attribute represents the data over which `h:dataTable` iterates; that data must be one of the following:

- A Java object
- An array
- An instance of `java.util.List`
- An instance of `java.sql.ResultSet`
- An instance of `javax.servlet.jsp.jstl.sql.Result`
- An instance of `javax.faces.model.DataModel`

As `h:dataTable` iterates, it makes each item in the array, list, result set, etc., available within the body of the tag. The name of the item is specified with `h:dataTable`'s `var` attribute. In the preceding code fragment, each item (`item`) of a collection (`items`) is made available, in turn, as `h:dataTable` iterates through the collection. You use properties from the current item to populate columns for the current row.

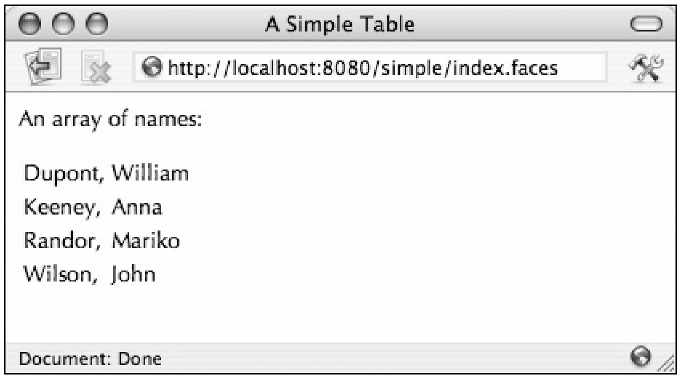
You can also specify any Java object for `h:dataTable`'s `value` attribute, although the usefulness of doing so is questionable. If that object is a scalar (meaning it is not a collection of some sort), `h:dataTable` iterates once, making the object available in the body of the tag.

The body of `h:dataTable` tags can contain only `h:column` tags; `h:dataTable` ignores all other component tags. Each column can contain an unlimited number of components in addition to optional header and footer components.

`h:dataTable` pairs a `UIData` component with a `Table` renderer. That combination provides robust table generation that includes support for CSS styles, database access, custom table models, and more. We start our `h:dataTable` exploration with a simple table.

A Simple Table

Figure 5–1 shows a table of names.



A Simple Table

http://localhost:8080/simple/index.faces

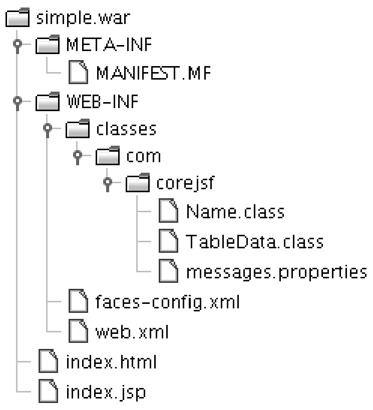
An array of names:

Dupont, William
Keeney, Anna
Randor, Mariko
Wilson, John

Document: Done

Figure 5–1 A simple table

The directory structure for the application shown in Figure 5–1 is shown in Figure 5–2. The application's JSF page is given in Listing 5–1.



```
simple.war
  META-INF
    MANIFEST.MF
  WEB-INF
    classes
      com
        corejsf
          Name.class
          TableData.class
          messages.properties
    faces-config.xml
    web.xml
  index.html
  index.jsp
```

Figure 5–2 The directory structure for the simple table

In Listing 5–1, we use `h:dataTable` to iterate over an array of names. The last name followed by a comma is placed in the left column and the first name is placed in the right column.

The array of names in this example is instantiated by a bean, which is managed by JSF. That bean is an instance of `com.corejsf.TableData`, which is shown in Listing 5–3. Listing 5–2 shows the `Name` class. The faces configuration file and message resource bundle are shown in Listing 5–4 and Listing 5–5, respectively.

Listing 5–1 simple/web/index.jsp

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

      <body>
         <h:outputText value="#{msgs.pageTitle}"/>
         <p>
         <h:form>
            <h:dataTable value="#{tableData.names}"
                           var="name">
               <h:column>
                  <h:outputText value="#{name.last}, "/>
               </h:column>

               <h:column>
                  <h:outputText value="#{name.first}"/>
               </h:column>
            </h:dataTable>
         </h:form>
      </body>
   </f:view>
</html>
```

Listing 5–2 simple/src/java/com/corejsf/Name.java

```
package com.corejsf;

public class Name {
   private String first;
   private String last;

   public Name(String first, String last) {
      this.first = first;
      this.last = last;
   }

```

Listing 5–2 simple/src/java/com/corejsf/Name.java (cont.)

```
   public void setFirst(String newValue) { first = newValue; }
   public String getFirst() { return first; }

   public void setLast(String newValue) { last = newValue; }
   public String getLast() { return last; }
}
```

Listing 5–3 simple/src/java/com/corejsf/TableData.java

```
package com.corejsf;

public class TableData {
   private static final Name[] names = new Name[] {
       new Name("William", "Dupont"),
       new Name("Anna", "Keeney"),
       new Name("Mariko", "Randor"),
       new Name("John", "Wilson")
   };

   public Name[] getNames() { return names;}
}
```

Listing 5–4 simple/web/WEB-INF/faces-config.xml

```
<?xml version="1.0"?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
   version="1.2">
   <application>
      <resource-bundle>
         <base-name>com.corejsf.messages</base-name>
         <var>msgs</var>
      </resource-bundle>
   </application>

   <managed-bean>
      <managed-bean-name>tableData</managed-bean-name>
      <managed-bean-class>com.corejsf.TableData</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>
</faces-config>
```

Listing 5–5 simple/src/java/com/corejsf/messages.properties

```
windowTitle=A Simple Table
pageTitle=An array of names:
```

The table in Figure 5–1 is intentionally vanilla. Throughout this chapter we will see how to add bells and whistles, such as CSS styles and column headers, to tables. But first we take a short tour of `h:dataTable` attributes.

CAUTION: `h:dataTable` data is row oriented—for example, the names in Listing 5–3 correspond to table rows, but the names say nothing about what is stored in each column—it is up to the page author to specify column content. Row-oriented data might be different from what you are used to; Swing table models, for example, keep track of what is in each row *and* column.

`h:dataTable` *Attributes*

`h:dataTable` attributes are listed in Table 5–1.

Table 5–1 Attributes for `h:dataTable`

Attribute	Description
`bgcolor`	Background color for the table
`border`	Width of the table's border
`captionClass` (JSF 1.2)	The CSS class for the table caption
`captionStyle` (JSF 1.2)	A CSS style for the table caption
`cellpadding`	Padding around table cells
`cellspacing`	Spacing between table cells
`columnClasses`	Comma-separated list of CSS classes for columns
`dir`	Text direction for text that does not inherit directionality; valid values: `LTR` (left to right) and `RTL` (right to left)
`first`	A zero-relative index of the first row shown in the table
`footerClass`	CSS class for the table footer

Table 5–1 Attributes for `h:dataTable` (cont.)

Attribute	Description
`frame`	Specification for sides of the frame surrounding the table; valid values: `none`, `above`, `below`, `hsides`, `vsides`, `lhs`, `rhs`, `box`, `border`
`headerClass`	CSS class for the table header
`rowClasses`	Comma-separated list of CSS classes for columns
`rows`	The number of rows displayed in the table, starting with the row specified with the `first` attribute; if you set this value to zero, all table rows will be displayed
`rules`	Specification for lines drawn between cells; valid values: `groups`, `rows`, `columns`, `all`
`summary`	Summary of the table's purpose and structure used for nonvisual feedback such as speech
`var`	The name of the variable created by the data table that represents the current item in the value
`binding`, `id`, `rendered`, `styleClass`, `value`	Basic
`lang`, `style`, `title`, `width`	HTML 4.0
`onclick`, `ondblclick`, `onkeydown`, `onkeypress`, `onkeyup`, `onmousedown`, `onmousemove`, `onmouseout`, `onmouseover`, `onmouseup`	DHTML events

The `binding` and `id` attributes are discussed in "IDs and Bindings" on page 98 of Chapter 4, and `rendered` attributes are discussed in "An Overview of the JSF HTML Tags" on page 94 of Chapter 4.

`h:dataTable` also comes with a full complement of DHTML event and HTML 4.0 pass-through attributes. You can read more about those attributes in Chapter 4.

The `first` attribute specifies a zero-relative index of the first visible row in the table. The `value` attribute points to the data over which `h:dataTable` iterates. At the start of each iteration, `h:dataTable` creates a request-scoped variable that you name with `h:dataTable`'s `var` attribute. Within the body of the `h:dataTable` tag, you can reference the current item with that name.

h:column *Attributes*

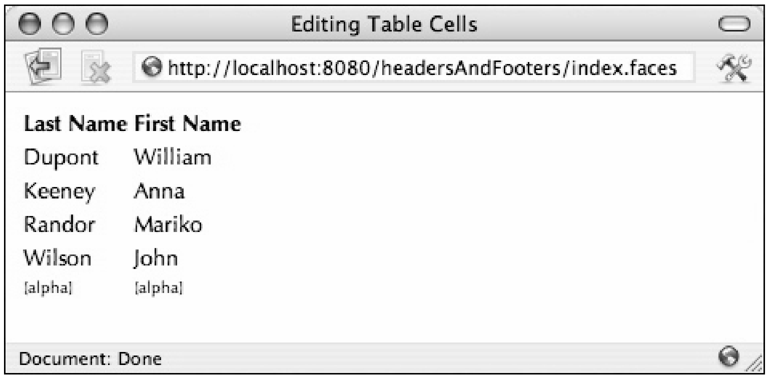
h:column attributes are listed in Table 5–2.

Table 5–2 Attributes for h:column

Attribute	Description
footerClass (JSF 1.2)	The CSS class for the column's footer
headerClass (JSF 1.2)	The CSS class for the column's header
binding, id, rendered, styleClass, value	Basic

Headers, Footers, and Captions

If you display a list of names as we did in "A Simple Table" on page 173, you need to distinguish last names from first names. You can do that with a column header, as shown in Figure 5–3.



Last Name	First Name
Dupont	William
Keeney	Anna
Randor	Mariko
Wilson	John
[alpha]	[alpha]

Figure 5–3 Specifying column headers and footers

Besides headers, the table columns in Figure 5–3 also contain footers that indicate the data type of their respective columns; in this case, both columns are [alpha], for alphanumeric.

Column headers and footers are specified with facets, as shown below:

```
<h:dataTable>
   ...
   <h:column headerClass="columnHeader"
             footerClass="columnFooter">
      <f:facet name="header">
```

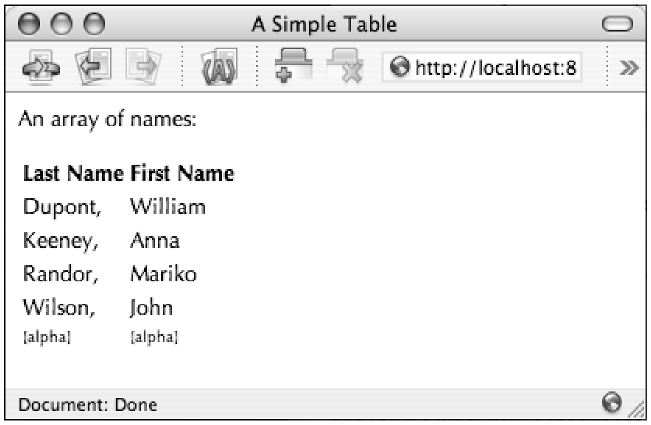
```
            <%-- header components go here --%>
         </f:facet>

         <%-- column components go here --%>

         <f:facet name="footer">
            <%-- footer components go here --%>
         </f:facet>
      </h:column>
      ...
   </h:dataTable>
```

h:dataTable places the components specified for the header and footer facets in the HTML table's header and footer, respectively. Notice that we use the h:column headerClass and footerClass attributes to specify CSS styles for column headers and footers, respectively. Those attributes are new for JSF 1.2.

JSF 1.2 adds table captions to h:dataTable. You add a caption facet to your h:dataTable tag, like this:

```
   <h:dataTable ...>
      <f:facet name="caption">
         An array of names:
      </f:facet>
   </h:dataTable>
```

If you add this facet to the table shown in Figure 5–3, this is what you will see:



An array of names:

Last Name	First Name
Dupont,	William
Keeney,	Anna
Randor,	Mariko
Wilson,	John
[alpha]	[alpha]

Figure 5–4 A table caption

You can use `captionStyle` and `captionClass` to specify a style or CSS class, respectively, for the caption:

```
<h:dataTable ... captionClass="caption">
   <f:facet name="caption">
      An array of names:
   </f:facet>
</h:dataTable>
```

In the preceding code snippet, we used some plain text for the facet, but like any facet, you can specify a JSF component instead.

The full code for the JSF page shown in Figure 5–4 is given in Listing 5–6. The application's resource bundle is shown in Listing 5–7. The directory structure for the application is identical to the one shown in Figure 5–2 on page 173.

You will notice we have used the `style` attribute for output components to format column headers and footers. See "HTML 4.0 Attributes" on page 101 of Chapter 4 for more information on the `style` attribute in general, and "Styles" on page 189 for more about CSS style classes and JSF tables.

Listing 5–6 headersAndFooters/web/index.jsp

```
<html>
   <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
   <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
   <f:view>
      <head>
         <link href="styles.css" rel="stylesheet" type="text/css"/>
         <title>
            <h:outputText value="#{msgs.windowTitle}"/>
         </title>
      </head>
      <body>
         <h:form>
            <h:dataTable value="#{tableData.names}"
               var="name"
               captionStyle="font-size: 0.95em; font-style:italic"
               style="width: 250px;">

               <f:facet name="caption">
                   <h:outputText value="An array of names:"/>
               </f:facet>

               <h:column headerClass="columnHeader"
                  footerClass="columnFooter">
```

Listing 5–6 headersAndFooters/web/index.jsp (cont.)

```
                <f:facet name="header">
                   <h:outputText value="#{msgs.lastnameColumn}"/>
                </f:facet>

                <h:outputText value="#{name.last}"/>

                <f:facet name="footer">
                   <h:outputText value="#{msgs.alphanumeric}"/>
                </f:facet>
             </h:column>

             <h:column headerClass="columnHeader"
                footerClass="columnFooter">
                <f:facet name="header">
                   <h:outputText value="#{msgs.firstnameColumn}"/>
                </f:facet>

                <h:outputText value="#{name.first}"/>

                <f:facet name="footer">
                   <h:outputText value="#{msgs.alphanumeric}"/>
                </f:facet>
             </h:column>
          </h:dataTable>
       </h:form>
    </body>
 </f:view>
</html>
```

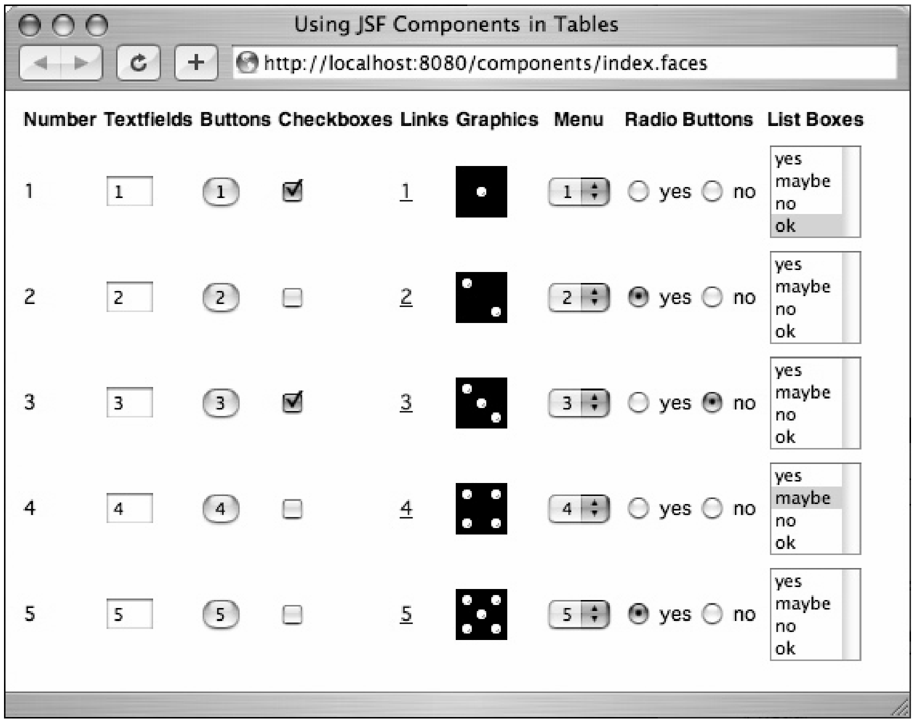
Listing 5–7 headersAndFooters/src/java/corejsf/messages.properties

```
windowTitle=Headers, Footers, and Captions
lastnameColumn=Last Name
firstnameColumn=First Name
editColumn=Edit
alphanumeric=[alpha]
```

TIP: To place multiple components in a table header or footer, you must group them in an `h:panelGroup` tag or place them in a container component with `h:panelGrid` or `h:dataTable`. If you place multiple components in a facet, only the first component will be displayed.

JSF Components

To this point, we have used only output components in table columns, but you can place any JSF component in a table cell. Figure 5–5 shows an application that uses a variety of components in a table.


Using JSF Components in Tables
http://localhost:8080/components/index.faces

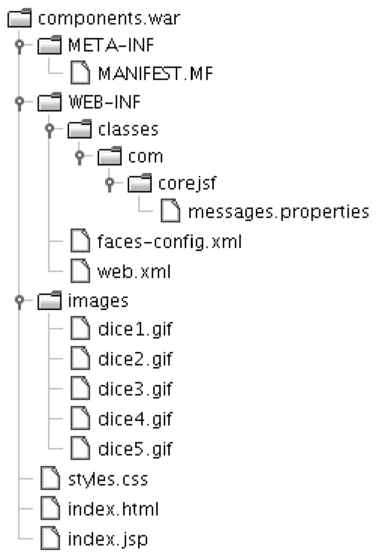
Number	Textfields	Buttons	Checkboxes	Links	Graphics	Menu	Radio Buttons	List Boxes
1	1	1	✓	1		1	yes no	yes maybe no [✓ ok]
2	2	2		2		2	[✓ yes] no	yes maybe no ok
3	3	3	✓	3		3	yes [✓ no]	yes maybe no ok
4	4	4		4		4	yes no	yes [✓ maybe] no ok
5	5	5		5		5	[✓ yes] no	yes maybe no ok

Figure 5–5 JSF components in table cells

`h:dataTable` iterates over data, so the table shown in Figure 5–5 provides a list of integers for that purpose. We use the current integer to configure components in the "Number", "Textfields", "Buttons", and "Menu" columns.

Components in a table are no different than components outside a table; you can manipulate them in any manner you desire, including conditional rendering with the `rendered` attribute, handling events, and the like.

The directory structure for the application shown in Figure 5–5 is shown in Figure 5–6. The JSF page, faces configuration file, and property resource bundle are given in Listings 5–8 through 5–10.



```
components.war
  META-INF
    MANIFEST.MF
  WEB-INF
    classes
      com
        corejsf
          messages.properties
    faces-config.xml
    web.xml
  images
    dice1.gif
    dice2.gif
    dice3.gif
    dice4.gif
    dice5.gif
  styles.css
  index.html
  index.jsp
```

Figure 5–6 Directory structure for the components example

Listing 5–8 components/web/index.jsp

```
<html>
   <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
   <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
   <f:view>
      <head>
         <link href="styles.css" rel="stylesheet" type="text/css"/>
         <title>
            <h:outputText value="#{msgs.windowTitle}"/>
         </title>
      </head>
      <body style="background: #eee">
         <h:form>
            <h:dataTable value="#{numberList}" var="number">
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.numberHeader}"/>
                  </f:facet>
                  <h:outputText value="#{number}"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.textfieldHeader}"/>
                  </f:facet>
```

Listing 5–8 components/web/index.jsp (cont.)

```
            <h:inputText value="#{number}" size="3"/>
         </h:column>
         <h:column>
            <f:facet name="header">
               <h:outputText value="#{msgs.buttonHeader}"/>
            </f:facet>
            <h:commandButton value="#{number}"/>
         </h:column>
         <h:column>
            <f:facet name="header">
               <h:outputText value="#{msgs.checkboxHeader}"/>
            </f:facet>
            <h:selectBooleanCheckbox value="false"/>
         </h:column>
         <h:column>
            <f:facet name="header">
               <h:outputText value="#{msgs.linkHeader}"/>
            </f:facet>
            <h:commandLink>
               <h:outputText value="#{number}"/>
            </h:commandLink>
         </h:column>
         <h:column>
            <f:facet name="header">
               <h:outputText value="#{msgs.graphicHeader}"/>
            </f:facet>
            <h:graphicImage value="images/dice#{number}.gif"
               style="border: 0px"/>
         </h:column>
         <h:column>
            <f:facet name="header">
               <h:outputText value="#{msgs.menuHeader}"/>
            </f:facet>
            <h:selectOneMenu>
               <f:selectItem itemLabel="#{number}" itemValue="#{number}"/>
            </h:selectOneMenu>
         </h:column>
         <h:column>
            <f:facet name="header">
               <h:outputText value="#{msgs.radioHeader}"/>
            </f:facet>
            <h:selectOneRadio layout="LINE_DIRECTION" value="nextMonth">
               <f:selectItem itemValue="yes" itemLabel="yes"/>
               <f:selectItem itemValue="no" itemLabel="no" />
            </h:selectOneRadio>
         </h:column>
```

Listing 5–8 components/web/index.jsp (cont.)

```
            <h:column>
               <f:facet name="header">
                  <h:outputText value="#{msgs.listboxHeader}"/>
               </f:facet>
               <h:selectOneListbox size="3">
                  <f:selectItem itemValue="yes" itemLabel="yes"/>
                  <f:selectItem itemValue="maybe" itemLabel="maybe"/>
                  <f:selectItem itemValue="no" itemLabel="no" />
                  <f:selectItem itemValue="ok" itemLabel="ok" />
               </h:selectOneListbox>
            </h:column>
         </h:dataTable>
      </h:form>
   </body>
</f:view>
</html>
```

Listing 5–9 components/web/WEB-INF/faces-config.xml

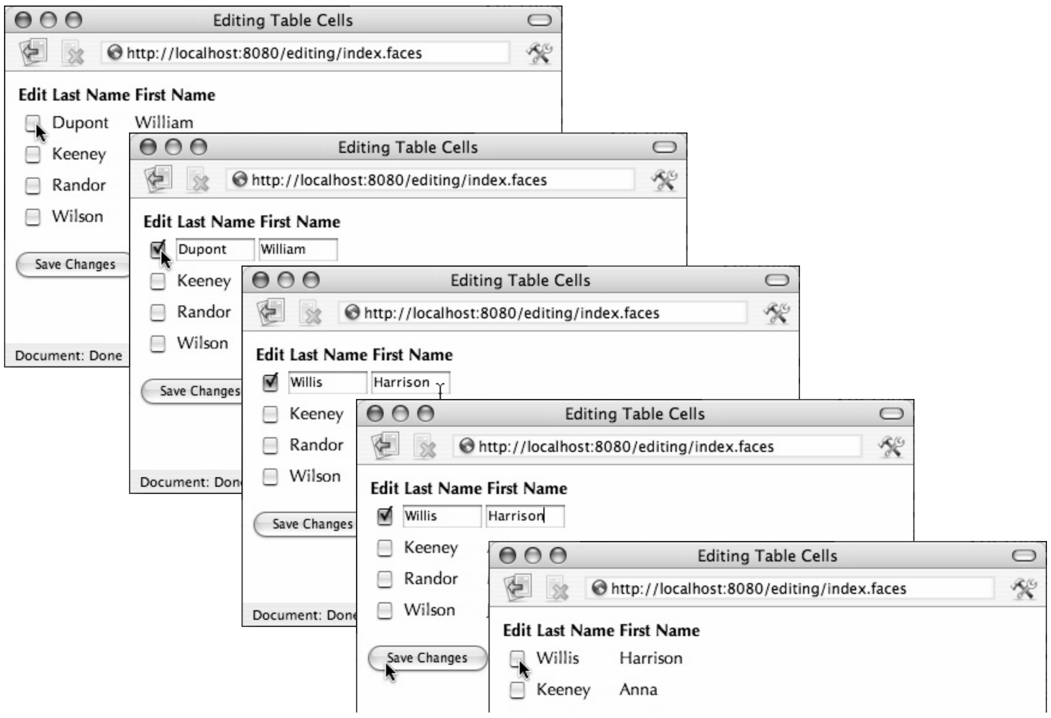
```
<?xml version="1.0"?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
   version="1.2">
   <application>
      <resource-bundle>
         <base-name>com.corejsf.messages</base-name>
         <var>msgs</var>
      </resource-bundle>
   </application>

   <managed-bean>
      <managed-bean-name>numberList</managed-bean-name>
      <managed-bean-class>java.util.ArrayList</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
      <list-entries>
         <value>1</value>
         <value>2</value>
         <value>3</value>
         <value>4</value>
         <value>5</value>
      </list-entries>
   </managed-bean>
</faces-config>
```

Listing 5–10 `components/src/java/com/corejsf/messages.properties`

```
windowTitle=Using JSF Components in Tables

numberHeader=Number
textfieldHeader=Textfields
buttonHeader=Buttons
checkboxHeader=Checkboxes
linkHeader=Links
menuHeader=Menu
graphicHeader=Graphics
radioHeader=Radio Buttons
listboxHeader=List Boxes
```

Editing Table Cells

To edit table cells, you provide an input component for the cell you want to edit. The application shown in Figure 5–7 allows editing of all its cells. You click a checkbox to edit a row and then click the “Save Changes” button to save your changes. From top to bottom, Figure 5–7 shows a cell being edited.



Editing Table Cells

http://localhost:8080/editing/index.faces

Edit	Last Name	First Name
☐	Dupont	William
☐	Keeney	
☐	Randor	
☐	Wilson	

Save Changes

Document: Done

Editing Table Cells

http://localhost:8080/editing/index.faces

Edit	Last Name	First Name
✓	Dupont	William
☐	Keeney	
☐	Randor	
☐	Wilson	

Save Changes

Editing Table Cells

http://localhost:8080/editing/index.faces

Edit	Last Name	First Name
✓	Willis	Harrison
☐	Keeney	
☐	Randor	
☐	Wilson	

Save Changes

Document: Done

Editing Table Cells

http://localhost:8080/editing/index.faces

Edit	Last Name	First Name
✓	Willis	Harrison
☐	Keeney	
☐	Randor	
☐	Wilson	

Save Changes

Editing Table Cells

http://localhost:8080/editing/index.faces

Edit	Last Name	First Name
☐	Willis	Harrison
☐	Keeney	Anna

Figure 5–7 Editing table cells

The table cells in Figure 5–7 use an input component when the cell is being edited and an output component when it is not. Here is how that is implemented:

```
...
<h:dataTable value='#{tableData.names}' var='name'>
   <!-- checkbox column -->
   <h:column>
      <f:facet name='header'>
         <h:output_text value='#{msgs.editColumn}'
            style='font-weight: bold'/>
      </f:facet>

      <h:selectBooleanCheckbox value='#{name.editable}'
         onclick='submit()'/>
   </h:column>

   <!-- last name column -->
   <h:column>
      ...
      <h:inputText value='#{name.last}'
         rendered='#{name.editable}'
         size='10'/>

      <h:outputText value='#{name.last}'
         rendered='#{not name.editable}'/>
   </h:column>
   ...
</h:dataTable>
<p>
<h:commandButton value="#{msgs.saveChangesButtonText}"/>
...
```

The preceding code fragment lists only the code for the checkbox and last name columns. The value of the checkbox corresponds to whether the current name is editable; if so, the checkbox is checked. Two components are specified for the last name column: an `h:inputText` and an `h:outputText`. If the name is editable, the input component is rendered. If the name is not editable, the output component is rendered.

The full listing for the JSF page shown in Figure 5–7 is given in Listing 5–11. The messages resource bundle for the application is shown in Listing 5–12. The directory structure for the application is the same as the one shown in Figure 5–2 on page 173.

Listing 5–11 editing/web/index.jsp

```
<html>
   <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
   <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
   <f:view>
      <head>
         <title>
            <h:outputText value="#{msgs.windowTitle}"/>
         </title>
      </head>
      <body>
         <h:form>
            <h:dataTable value="#{tableData.names}" var="name">
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.editColumn}"
                        style="font-weight: bold"/>
                  </f:facet>
                  <h:selectBooleanCheckbox value="#{name.editable}"
                     onclick="submit()"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.lastnameColumn}"
                        style="font-weight: bold"/>
                  </f:facet>
                  <h:inputText value="#{name.last}" rendered="#{name.editable}"
                     size="10"/>
                  <h:outputText value="#{name.last}"
                     rendered="#{not name.editable}"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.firstnameColumn}"
                        style="font-weight: bold"/>
                  </f:facet>
                  <h:inputText value="#{name.first}"
                     rendered="#{name.editable}" size="10"/>
                  <h:outputText value="#{name.first}"
                     rendered="#{not name.editable}"/>
               </h:column>
            </h:dataTable>
            <h:commandButton value="#{msgs.saveChangesButtonText}"/>
         </h:form>
      </body>
   </f:view>
</html>
```

Listing 5–12 `editing/src/java/com/corejsf/messages.properties`

```
windowTitle=Editing Table Cells
lastnameColumn=Last Name
firstnameColumn=First Name
editColumn=Edit
alphanumeric=[alpha]
saveChangesButtonText=Save changes
```

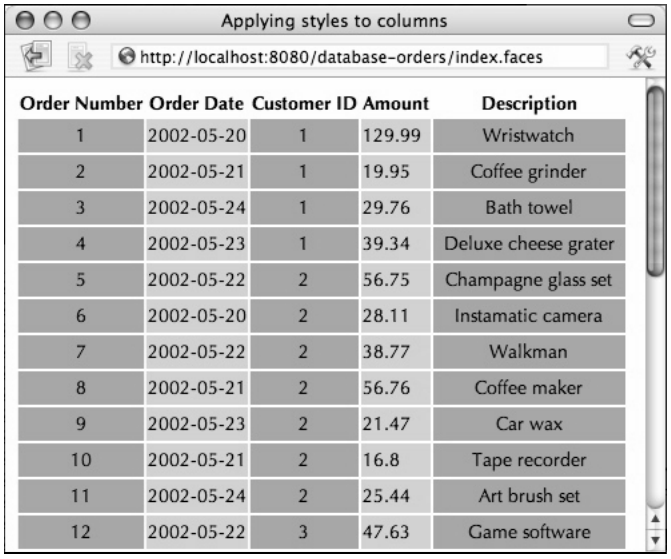
NOTE: Table cell editing, as illustrated in "Editing Table Cells" on page 186, works for all valid types of table data: Java objects, arrays, lists, result sets, and results. However, for database tables, the *result set* associated with a table must be *updatable* for the JSF implementation to update the database.

Styles

`h:dataTable` has attributes that specify CSS classes for the following:

- The table as a whole (`styleClass`)
- Column headers and footers (`headerClass` and `footerClass`)
- Individual columns (`columnClasses`)
- Individual rows (`rowClasses`)

The table shown in Figure 5–8 uses `styleClass`, `headerClass`, and `columnClasses`.


Applying styles to columns
http://localhost:8080/database-orders/index.faces

Order Number	Order Date	Customer ID	Amount	Description
1	2002-05-20	1	129.99	Wristwatch
2	2002-05-21	1	19.95	Coffee grinder
3	2002-05-24	1	29.76	Bath towel
4	2002-05-23	1	39.34	Deluxe cheese grater
5	2002-05-22	2	56.75	Champagne glass set
6	2002-05-20	2	28.11	Instamatic camera
7	2002-05-22	2	38.77	Walkman
8	2002-05-21	2	56.76	Coffee maker
9	2002-05-23	2	21.47	Car wax
10	2002-05-21	2	16.8	Tape recorder
11	2002-05-24	2	25.44	Art brush set
12	2002-05-22	3	47.63	Game software

Figure 5–8 Applying styles to columns and headers

NOTE: The `h:dataTable` `rowClasses` and `columnClasses` attributes are mutually exclusive. If you specify both, `columnClasses` has priority.

Styles by Column

Here is how the CSS classes in Figure 5–8 are specified:

```
<link href='styles.css' rel='stylesheet' type='text/css'/>
...
<h:dataTable value="#{order.all}" var="order"
   styleClass="orders"
   headerClass="ordersHeader"
   columnClasses="oddColumn,evenColumn">
```

Those CSS classes are listed below.

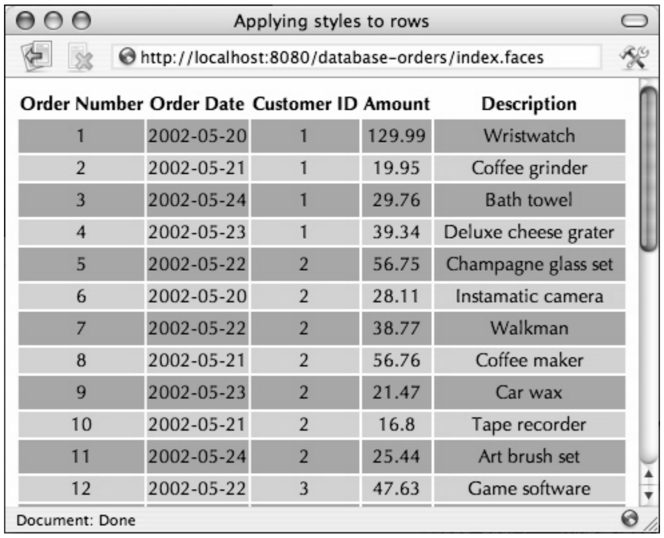
```
.orders {
   border: thin solid black;
}
.ordersHeader {
   text-align: center;
   font-style: italic;
   color: Snow;
   background: Teal;
}
.oddColumn {
   height: 25px;
   text-align: center;
   background: MediumTurquoise;
}
.evenColumn {
   text-align: center;
   background: PowderBlue;
}
```

We specified only two column classes, but notice that we have five columns. In this case, `h:dataTable` reuses the column classes, starting with the first. By specifying only the first two column classes, we can set the CSS classes for even and odd columns (1-based counting).

Styles by Row

You can use the `rowClasses` attribute to specify CSS classes by rows instead of columns, as illustrated in Figure 5–9. That data table is implemented like this:

```
<link href='styles.css' rel='stylesheet' type='text/css'/>
...
<h:dataTable value="#{order.all}" var="order"
   styleClass="orders"
   headerClass="ordersHeader"
   rowClasses="oddRow,evenRow">
```



Order Number	Order Date	Customer ID	Amount	Description
1	2002-05-20	1	129.99	Wristwatch
2	2002-05-21	1	19.95	Coffee grinder
3	2002-05-24	1	29.76	Bath towel
4	2002-05-23	1	39.34	Deluxe cheese grater
5	2002-05-22	2	56.75	Champagne glass set
6	2002-05-20	2	28.11	Instamatic camera
7	2002-05-22	2	38.77	Walkman
8	2002-05-21	2	56.76	Coffee maker
9	2002-05-23	2	21.47	Car wax
10	2002-05-21	2	16.8	Tape recorder
11	2002-05-24	2	25.44	Art brush set
12	2002-05-22	3	47.63	Game software

Figure 5–9 Applying styles to rows

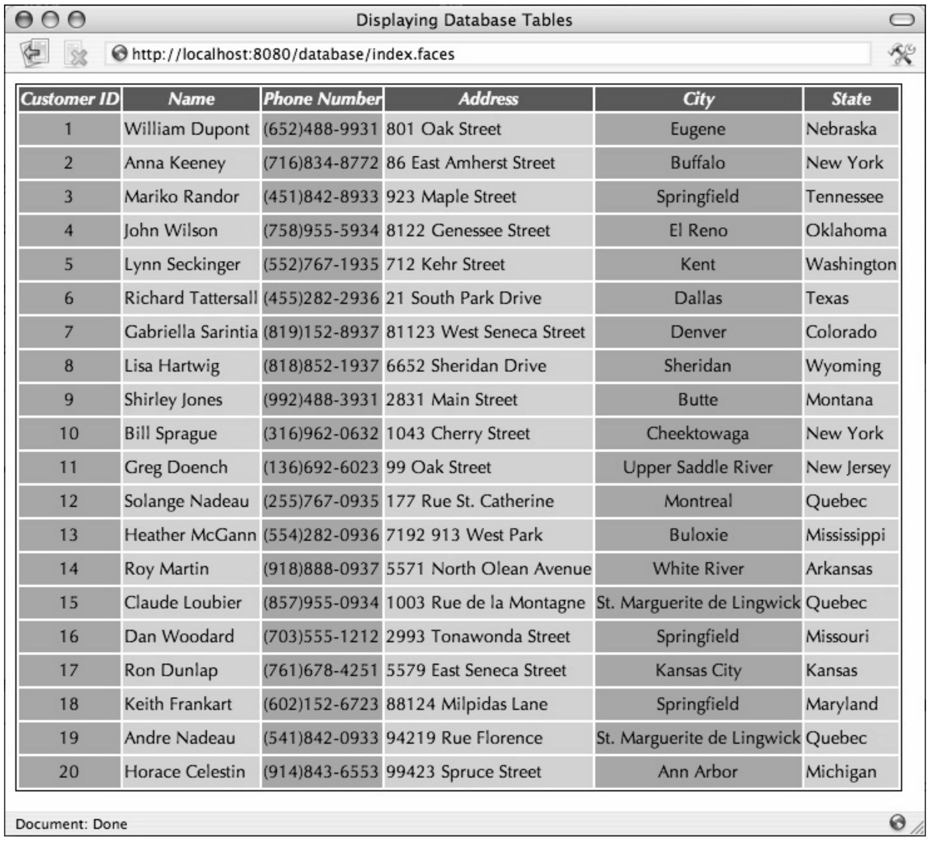
Like column classes, `h:dataTable` reuses row classes when the number of classes is less than the number of rows. In the preceding code fragment, we have taken advantage of this feature to specify CSS classes for even and odd rows.

CAUTION: We use color names, such as PowderBlue and Medium-Turquoise, in our style classes for the sake of illustration. You should prefer the equivalent hex constants because they are portable, whereas color names are not.

Database Tables

Databases store information in tables, so the JSF data table component is a good fit for showing data stored in a database. In this section, we show you how to display the results of a database query.

Figure 5–10 shows a JSF application that displays a database table.


Displaying Database Tables
http://localhost:8080/database/index.faces
Document: Done

Customer ID	Name	Phone Number	Address	City	State
1	William Dupont	(652)488-9931	801 Oak Street	Eugene	Nebraska
2	Anna Keeney	(716)834-8772	86 East Amherst Street	Buffalo	New York
3	Mariko Randor	(451)842-8933	923 Maple Street	Springfield	Tennessee
4	John Wilson	(758)955-5934	8122 Genessee Street	El Reno	Oklahoma
5	Lynn Seckinger	(552)767-1935	712 Kehr Street	Kent	Washington
6	Richard Tattersall	(455)282-2936	21 South Park Drive	Dallas	Texas
7	Gabriella Sarintia	(819)152-8937	81123 West Seneca Street	Denver	Colorado
8	Lisa Hartwig	(818)852-1937	6652 Sheridan Drive	Sheridan	Wyoming
9	Shirley Jones	(992)488-3931	2831 Main Street	Butte	Montana
10	Bill Sprague	(316)962-0632	1043 Cherry Street	Cheektowaga	New York
11	Greg Doench	(136)692-6023	99 Oak Street	Upper Saddle River	New Jersey
12	Solange Nadeau	(255)767-0935	177 Rue St. Catherine	Montreal	Quebec
13	Heather McGann	(554)282-0936	7192 913 West Park	Buloxie	Mississippi
14	Roy Martin	(918)888-0937	5571 North Olean Avenue	White River	Arkansas
15	Claude Loubier	(857)955-0934	1003 Rue de la Montagne	St. Marguerite de Lingwick	Quebec
16	Dan Woodard	(703)555-1212	2993 Tonawonda Street	Springfield	Missouri
17	Ron Dunlap	(761)678-4251	5579 East Seneca Street	Kansas City	Kansas
18	Keith Frankart	(602)152-6723	88124 Milpidas Lane	Springfield	Maryland
19	Andre Nadeau	(541)842-0933	94219 Rue Florence	St. Marguerite de Lingwick	Quebec
20	Horace Celestin	(914)843-6553	99423 Spruce Street	Ann Arbor	Michigan

Figure 5–10 Displaying database tables

The JSF page shown in Figure 5–10 uses `h:dataTable`, like this:

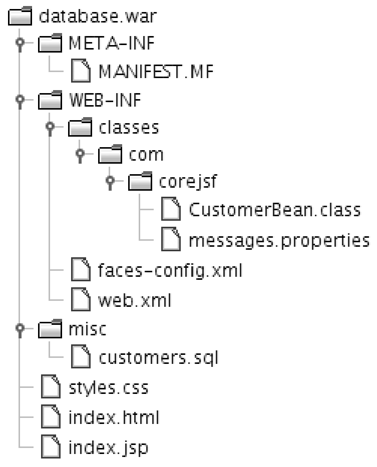
```
<h:dataTable value="#{customer.all}" var="currentCustomer"
   styleClass="customers"
   headerClass="customersHeader"
   columnClasses="custid,name">
      <h:column>
      <f:facet name="header">
         <h:outputText value="#{msgs.customerIdHeader}"/>
      </f:facet>
      <h:outputText value="#{currentCustomer.Cust_ID}"/>
   </h:column>
```

```
    <h:column>
       <f:facet name="header">
          <h:outputText value="#{msgs.nameHeader}"/>
       </f:facet>
          <h:outputText value="#{currentCustomer.Name}"/>
    </h:column>
    ...
 </h:dataTable>
```

The `customer` bean is a managed bean that knows how to connect to a database and perform a query of all customers in the database. The `CustomerBean.all` method performs that query.

The preceding JSF page accesses column data by referencing column names—for example, `#{customer.Cust_ID}` references the `Cust_ID` column.

The directory structure for the database example is shown in Figure 5–11. Listings for the application are given in Listing 5–13 through Listing 5–16.



```
database.war
  META-INF
    MANIFEST.MF
  WEB-INF
    classes
      com
        corejsf
          CustomerBean.class
          messages.properties
    faces-config.xml
    web.xml
  misc
    customers.sql
  styles.css
  index.html
  index.jsp
```

Figure 5–11 Directory structure for the database example

Listing 5–13 database/web/index.jsp

```
<html>
   <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
   <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
   <f:view>
      <head>
         <link href="styles.css" rel="stylesheet" type="text/css"/>
         <title>
            <h:outputText value="#{msgs.pageTitle}"/>
         </title>
      </head>
      <body>
         <h:form>
            <h:dataTable value="#{customer.all}" var="customer"
               styleClass="customers"
               headerClass="customersHeader" columnClasses="custid,name">
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.customerIdHeader}"/>
                  </f:facet>
                  <h:outputText value="#{customer.Cust_ID}"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.nameHeader}"/>
                  </f:facet>
                  <h:outputText value="#{customer.Name}"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.phoneHeader}"/>
                  </f:facet>
                  <h:outputText value="#{customer.Phone_Number}"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.addressHeader}"/>
                  </f:facet>
                  <h:outputText value="#{customer.Street_Address}"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.cityHeader}"/>
                  </f:facet>
                  <h:outputText value="#{customer.City}"/>
               </h:column>
```

Listing 5–13 database/web/index.jsp (cont.)

```
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.stateHeader}"/>
                  </f:facet>
                  <h:outputText value="#{customer.State}"/>
               </h:column>
            </h:dataTable>
         </h:form>
      </body>
   </f:view>
</html>
```

Listing 5–14 database/src/java/com/corejsf/CustomerBean.java

```
package com.corejsf;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.jsp.jstl.sql.Result;
import javax.servlet.jsp.jstl.sql.ResultSupport;
import javax.sql.DataSource;

public class CustomerBean {
   private Connection conn;

   public void open() throws SQLException, NamingException {
      if (conn != null) return;
      Context ctx = new InitialContext();
      DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/mydb");
      conn = ds.getConnection();
   }

   public Result getAll() throws SQLException, NamingException {
      try {
         open();
         Statement stmt = conn.createStatement();
         ResultSet result = stmt.executeQuery("SELECT * FROM Customers");
         return ResultSupport.toResult(result);
```

Listing 5–14 database/src/java/com/corejsf/CustomerBean.java (cont.)

```
      } finally {
         close();
      }
   }

   public void close() throws SQLException {
      if (conn == null) return;
      conn.close();
      conn = null;
   }
}
```

Listing 5–15 database/web/WEB-INF/web.xml

```
<?xml version="1.0"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   version="2.5">
   <servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>*.faces</url-pattern>
   </servlet-mapping>

   <welcome-file-list>
      <welcome-file>index.html</welcome-file>
   </welcome-file-list>

   <resource-ref>
      <res-ref-name>jdbc/mydb</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
   </resource-ref>
</web-app>
```

Listing 5–16 database/src/java/com/corejsf/messages.properties

```
pageTitle=Displaying Database Tables
customerIdHeader=Customer ID
nameHeader=Name
phoneHeader=Phone Number
addressHeader=Address
cityHeader=City
stateHeader=State
refreshFromDB=Read from database
```

JSTL Result Versus Result Sets

The value you specify for `h:dataTable` can be, among other things, an instance of `javax.servlet.jsp.jstl.Result` or an instance of `java.sql.ResultSet`—as was the case in "Database Tables" on page 191. `h:dataTable` wraps instances of those objects in instances of `ResultDataModel` and `ResultSetDataModel`, respectively. So how do the models differ? And which should you prefer?

If you have worked with result sets, you know they are fragile objects that require a good deal of programmatic control. The JSTL `Result` class is a bean that wraps a result set and implements that programmatic control for you; results are thus easier to deal with than are result sets.

On the other hand, wrapping a result set in a result involves some overhead in creating the `Result` object; your application may not be able to afford the performance penalty.

In the application discussed in "Database Tables" on page 191, we follow our own advice and return a JSTL `Result` object from the `CustomerBean.all` method.

Table Models

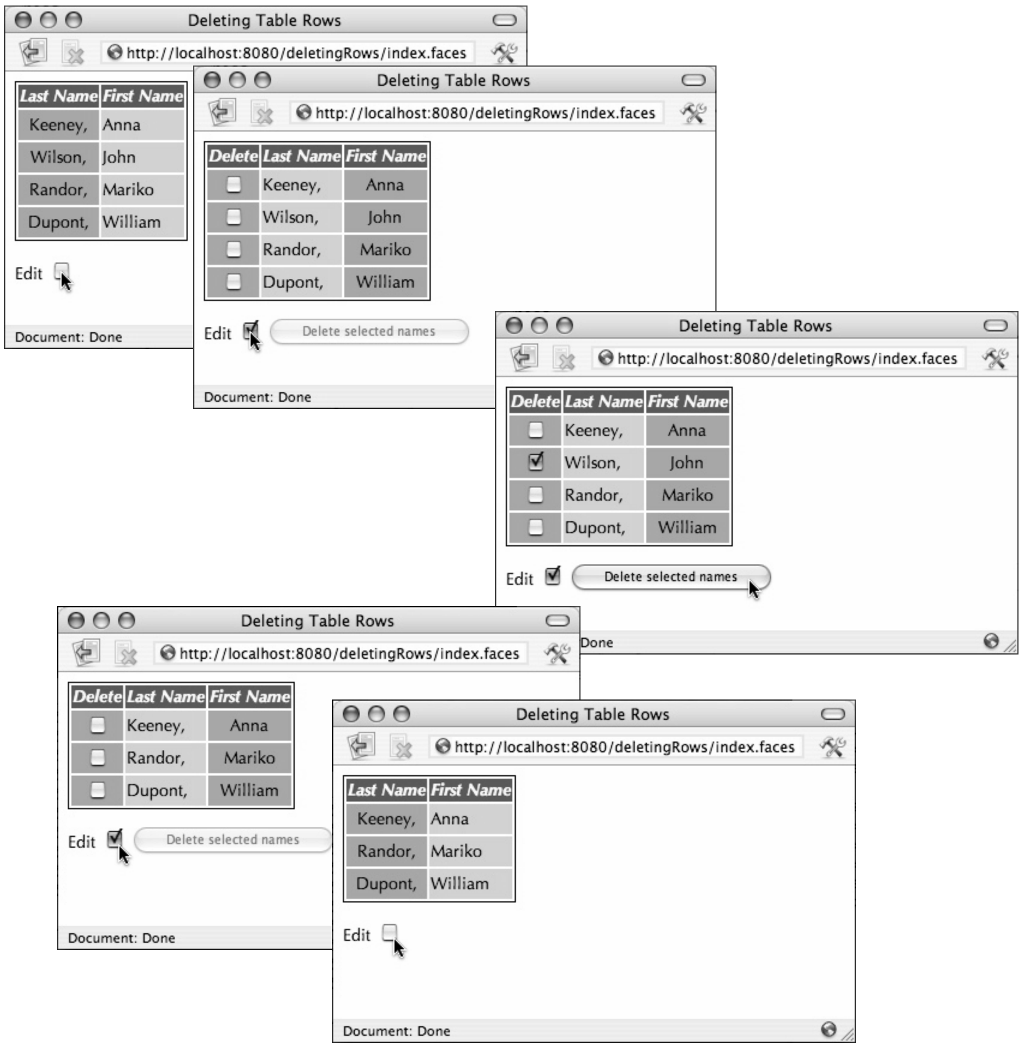
When you use a Java object, array, list, result set, or JSTL result object to represent table data, `h:dataTable` wraps those objects in a model that extends the `javax.faces.model.DataModel` class. All of those model classes, listed below, reside in the `javax.faces.model` package:

- `ArrayDataModel`
- `ListDataModel`
- `ResultDataModel`
- `ResultSetDataModel`
- `ScalarDataModel`

`h:dataTable` deals with the models listed above; it never directly accesses the object—array, list, etc.—you specify with the `value` attribute. You can, however, access those objects yourself with the `DataModel.getWrappedData` method. That method comes in handy for adding and removing table rows.

Editing Table Models

It is easy to add and delete table rows with two methods provided by all data models: `getWrappedData()` and `setWrappedData()`. Now we see how it works with an application, shown in Figure 5–12, that allows users to delete rows from a table.



Last Name	First Name
Keeney,	Anna
Wilson,	John
Randor,	Mariko
Dupont,	William

Delete	Last Name	First Name
☐	Keeney,	Anna
☑	Wilson,	John
☐	Randor,	Mariko
☐	Dupont,	William

Delete	Last Name	First Name
☐	Keeney,	Anna
☐	Randor,	Mariko
☐	Dupont,	William

Last Name	First Name
Keeney,	Anna
Randor,	Mariko
Dupont,	William

Figure 5–12 Deleting table rows

From top to bottom, Figure 5–12 shows the deletion of a single row. When a user activates the "Delete selected names" button, the JSF implementation invokes an action listener method that deletes the selected rows. That method looks like this:

```
public String deleteNames() {
   if (!getAnyNamesMarkedForDeletion())
      return null;

   Name[] currentNames = (Name[]) model.getWrappedData();
   Name[] newNames = new Name[currentNames.length -
      getNumberOfNamesMarkedForDeletion()];

   int i=0;
   for (Name name : currentNames) {
      if (!name.isMarkedForDeletion()) {
         newNames[i++] = name;
      }
   }
   model.setWrappedData(newNames);
   return null;
   }
}
```

The `deleteNames` method obtains a reference to the current set of names by calling the model's `getWrappedData` method. Then it creates a new array whose size is the size of the current array, minus the number of names that have been marked for deletion. Subsequently, the method adds each of the current names to the new array, leaving out names that were marked for deletion. Finally, the method calls the model's `setWrappedData` method to reset the model with the new array of names.

The JSF implementation invokes the `TableData.deleteNames` method when the "Delete selected names" button is activated. The `deleteNames` method obtains a reference to the current array of names by invoking the data model's `getWrappedData` method. The `deleteNames` method subsequently creates a new array—without the names marked for deletion—that it pushes to the model with the `setWrappedData` method.

Although the preceding example does not illustrate adding rows, it does illustrate the principle: Get the current object wrapped by the model with `getWrappedData()`, modify it (by adding or deleting rows), and then reset the wrapped object with `setWrappedData()`.

Figure 5–13 shows the directory structure for the application. Listing 5–17 through Listing 5–19 list the pertinent files from the application shown in Figure 5–12.

NOTE: Calling the model's `setWrappedData()` to reset the model's data is one way to delete rows. We could have also reset the model itself, like this:

```
model = new ArrayDataModel(newNames);
```

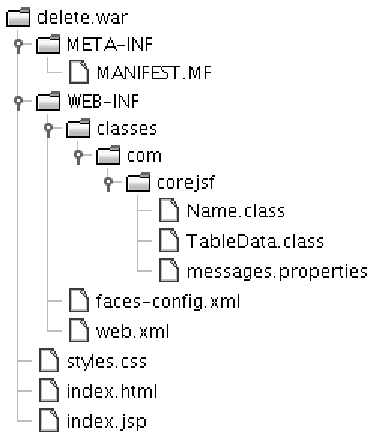


Figure 5–13 The directory structure for the delete example

Listing 5–17 delete/web/index.jsp

```
<html>
   <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
   <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
   <f:view>
      <head>
         <link href="styles.css" rel="stylesheet" type="text/css"/>
         <title>
            <h:outputText value="#{msgs.windowTitle}"/>
         </title>
      </head>
      <body>
         <h:form>
            <h:dataTable value="#{tableData.names}" var="name"
               styleClass="names" headerClass="namesHeader"
               columnClasses="last,first">
               <h:column rendered="#{tableData.editable}">
                  <f:facet name="header">
                     <h:outputText value="#{msgs.deleteColumnHeader}"/>
                  </f:facet>
```

Listing 5–17 delete/web/index.jsp (cont.)

```
                  <h:selectBooleanCheckbox value="#{name.markedForDeletion}"
                     onchange="submit()"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.lastColumnHeader}"/>
                  </f:facet>
                  <h:outputText value="#{name.last},"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.firstColumnHeader}"/>
                  </f:facet>
                  <h:outputText value="#{name.first}"/>
               </h:column>
            </h:dataTable>
            <h:outputText value="#{msgs.editPrompt}"/>
            <h:selectBooleanCheckbox onchange="submit()"
               value="#{tableData.editable}"/>
            <h:commandButton value="#{msgs.deleteButtonText}"
               rendered="#{tableData.editable}"
               action="#{tableData.deleteNames}"
               disabled="#{not tableData.anyNamesMarkedForDeletion}"/>
         </h:form>
      </body>
   </f:view>
</html>
```

Listing 5–18 delete/src/java/com/corejsf/Name.java

```
package com.corejsf;

public class Name {
   private String first;
   private String last;
   private boolean markedForDeletion = false;

   public Name(String first, String last) {
      this.first = first;
      this.last = last;
   }

   public void setFirst(String newValue) { first = newValue; }
   public String getFirst() { return first; }
```

Listing 5–18 delete/src/java/com/corejsf/Name.java (cont.)

```

   public void setLast(String newValue) { last = newValue; }
   public String getLast() { return last; }

   public boolean isMarkedForDeletion() { return markedForDeletion; }
   public void setMarkedForDeletion(boolean newValue) {
      markedForDeletion = newValue;
   }
}
```

Listing 5–19 delete/src/java/com/corejsf/TableData.java

```
package com.corejsf;

import javax.faces.model.DataModel;
import javax.faces.model.ArrayDataModel;

public class TableData {
   private boolean editable = false;
   private ArrayDataModel model = null;

   private static final Name[] names = {
      new Name("Anna", "Keeney"),
      new Name("John", "Wilson"),
      new Name("Mariko", "Randor"),
      new Name("William", "Dupont"),
   };

   public TableData() { model = new ArrayDataModel(names); }

   public DataModel getNames() { return model; }

   public boolean isEditable() { return editable; }
   public void setEditable(boolean newValue) { editable = newValue; }

   public String deleteNames() {
      if (!getAnyNamesMarkedForDeletion())
         return null;

      Name[] currentNames = (Name[]) model.getWrappedData();
      Name[] newNames = new Name[currentNames.length
         - getNumberOfNamesMarkedForDeletion()];
```

Listing 5–19 delete/src/java/com/corejsf/TableData.java (cont.)

```

      for(int i = 0, j = 0; i < currentNames.length; ++i) {
         Name name = (Name) currentNames[i];
         if (!name.isMarkedForDeletion()) {
            newNames[j++] = name;
         }
      }
      model.setWrappedData(newNames);
      return null;
   }

   public int getNumberOfNamesMarkedForDeletion() {
      Name[] currentNames = (Name[]) model.getWrappedData();
      int cnt = 0;

      for(int i = 0; i < currentNames.length; ++i) {
         Name name = (Name) currentNames[i];
         if (name.isMarkedForDeletion())
            ++cnt;
      }
      return cnt;
   }

   public boolean getAnyNamesMarkedForDeletion() {
      Name[] currentNames = (Name[]) model.getWrappedData();
      for(int i = 0; i < currentNames.length; ++i) {
         Name name = (Name) currentNames[i];
         if (name.isMarkedForDeletion())
            return true;
      }
      return false;
   }
}
```

We have seen how to perform simple manipulation of a data model. Sometimes, a little more sophistication is required—for example, when you sort or filter a model's data.

Sorting and Filtering

To sort or filter tables with `h:dataTable`, you need to implement a table model that decorates one of the table models listed on page 197. Figure 5–14 shows what it means to decorate a table model.

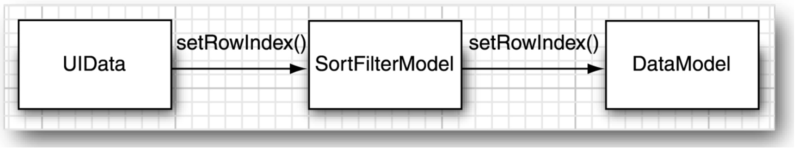


Figure 5–14 Data model filter

Instances of `UIData`—the component associated with `h:dataTable`—invoke methods on their model. When you decorate that model, your model intercepts those method calls. Your decorator model forwards method calls to the original model, except for the `setRowIndex` method, which returns a sorted index instead of the original model's index. Next, we see how that works.

Figure 5–15 shows the application discussed in "Editing Table Cells" on page 186, rewritten to support sortable table columns.

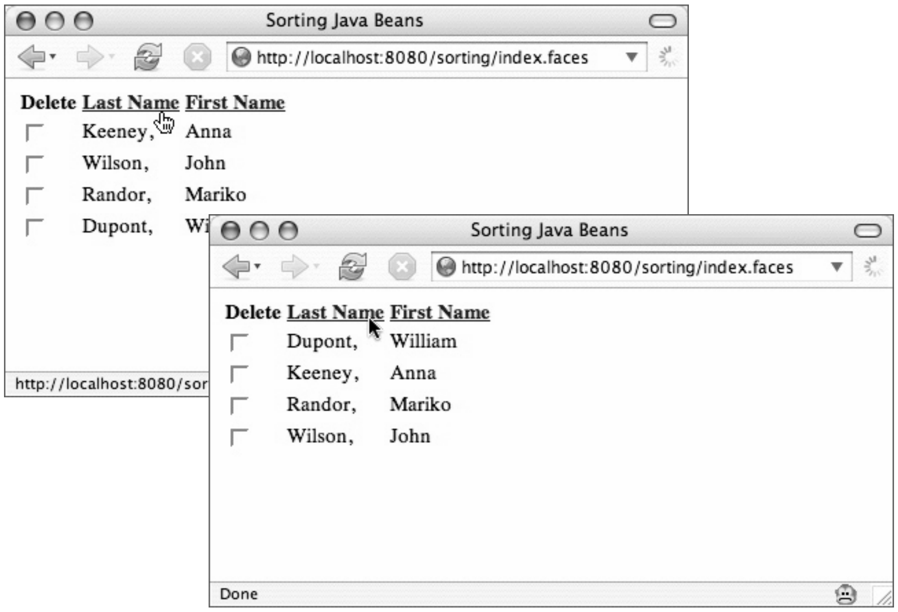

Sorting Java Beans
http://localhost:8080/sorting/index.faces
Delete Last Name First Name
Keeney, Anna
Wilson, John
Randor, Mariko
Dupont, Wi
http://localhost:8080/sor
Sorting Java Beans
http://localhost:8080/sorting/index.faces
Delete Last Name First Name
Dupont, William
Keeney, Anna
Randor, Mariko
Wilson, John
Done

Figure 5–15 Sorting table columns

The application shown in Figure 5–15 sorts table columns by decorating a table data model. First, we specify the `h:dataTable`'s `value` attribute, like this:

```
<h:dataTable value="#{tableData.names}" var="name" ...>
   ...
```

The TableData.names method returns a data model:

```
public class TableData {
   private DataModel filterModel= null;
   private static final Name[] names = {
      new Name("Anna", "Keeney"),
      new Name("John", "Wilson"),
      new Name("Mariko", "Randor"),
      new Name("William", "Dupont"),
   };

   public TableData() {
      ArrayDataModel model = new ArrayDataModel(names);
      filterModel = new SortFilterModel(model);
   }
   public DataModel getNames() {
      return filterModel;
   }
}
```

When the tableData object is created, it creates an ArrayDataModel instance, passing it the array of names. That is the *original* model. Then the TableData constructor wraps that model in a *sorting* model. When the getNames method is subsequently called to populate the data table, that method returns the sorting model. The sorting model is implemented like this:

```
public class SortFilterModel extends DataModel {
   private DataModel model;
   private Row[] rows;
   ...
   public SortFilterModel(DataModel model) {
      this.model = model;
      int rowCnt = model.getRowCount();
      if (rowCnt != -1) {
         rows = new Row[rowCnt];
         for (int i=0; i < rowCnt; ++i) {
         rows[i] = new Row(i);
      }
   }
   public void setRowIndex(int rowIndex) {
      if (rowIndex == -1 || rowIndex >= model.getRowCount()) {
      model.setRowIndex(rowIndex);
   }
   else {
      model.setRowIndex(rows[rowIndex].row);
      }
   }
```

```
      ...
   }
```

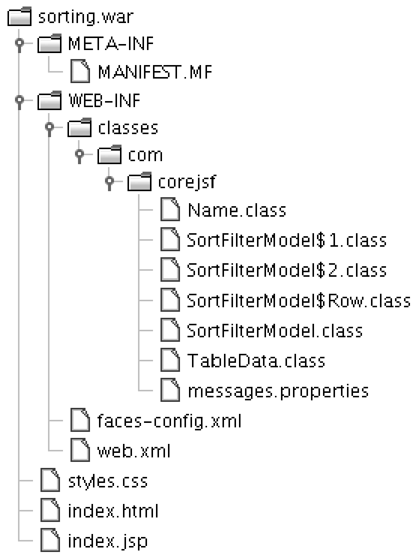
Notice that we create an array of indices that represent sorted indices. We return a sorted index from the `setRowIndex` method when the indicated index is in range.

So how does the sorting happen? The `SortFilterModel` class provides two methods, `sortByFirst()` and `sortByLast()`:

```
   public String sortByLast() {
      Arrays.sort(rows, byLast);
      return null;
   }

   public String sortByFirst() {
      Arrays.sort(rows, byFirst);
      return null;
   }
```

The `byLast` and `byFirst` variables are comparators. The former compares last names and the latter compares first names. You can see the implementation of the comparators in Listing 5–21 on page 208.

The directory structure for the sorting example is shown in Figure 5–16. Listing 5–20 through Listing 5–26 provide full listings of the application.



```
sorting.war
  META-INF
    MANIFEST.MF
  WEB-INF
    classes
      com
        corejsf
          Name.class
          SortFilterModel$1.class
          SortFilterModel$2.class
          SortFilterModel$Row.class
          SortFilterModel.class
          TableData.class
          messages.properties
    faces-config.xml
    web.xml
  styles.css
  index.html
  index.jsp
```

Figure 5–16 The directory structure for the sorting example

NOTE: The JSF 1.2 specification recommends that concrete `DataModel` classes provide at least two constructors: a no-argument constructor that calls `setWrappedData(null)` and a constructor that passes wrapped data to `setWrappedData()`. See Listing 5–21 on page 208 for an example of those constructors.

Listing 5–20 `sorting/web/index.jsp`

```
<html>
   <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
   <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
   <f:view>
      <head>
         <link href="site.css" rel="stylesheet" type="text/css"/>
         <title>
            <h:outputText value="#{msgs.windowTitle}"/>
         </title>
      </head>
      <body>
         <h:form>
            <h:dataTable value="#{tableData.names}" var="name"
               styleClass="names" headerClass="namesHeader"
               columnClasses="last,first">
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="#{msgs.deleteColumnHeader}"/>
                  </f:facet>
                  <h:selectBooleanCheckbox
                     value="#{name.markedForDeletion}"
                     onchange="submit()"/>
               </h:column>

               <h:column>
                  <f:facet name="header">
                     <h:commandLink action="#{tableData.names.sortByLast}">
                        <h:outputText value="#{msgs.lastColumnHeader}"/>
                     </h:commandLink>
                  </f:facet>
                  <h:outputText value="#{name.last}, "/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:commandLink action="#{tableData.names.sortByFirst}">
                        <h:outputText value="#{msgs.firstColumnHeader}"/>
```

Listing 5–20 sorting/web/index.jsp (cont.)

```
                     </h:commandLink>
                  </f:facet>
                  <h:outputText value="#{name.first}"/>
               </h:column>
            </h:dataTable>
            <h:commandButton value="#{msgs.deleteButtonText}"
               action="#{tableData.deleteNames}"
               rendered="#{tableData.anyNamesMarkedForDeletion}"/>
         </h:form>
      </body>
   </f:view>
</html>
```

Listing 5–21 sorting/src/java/com/corejsf/SortFilterModel.java

```
package com.corejsf;

import java.util.Arrays;
import java.util.Comparator;
import javax.faces.model.DataModel;
import javax.faces.model.DataModelEvent;
import javax.faces.model.DataModelListener;

public class SortFilterModel extends DataModel {
   private DataModel model;
   private Row[] rows;

   private static Comparator<Row> byLast = new
      Comparator<Row>() {
         public int compare(Row r1, Row r2) {
            Name n1 = (Name) r1.getData();
            Name n2 = (Name) r2.getData();
            return n1.getLast().compareTo(n2.getLast());
         }
      };

   private static Comparator<Row> byFirst = new
      Comparator<Row>() {
         public int compare(Row r1, Row r2) {
            Name n1 = (Name) r1.getData();
            Name n2 = (Name) r2.getData();
            return n1.getFirst().compareTo(n2.getFirst());
         }
      };
```

Listing 5–21 sorting/src/java/com/corejsf/SortFilterModel.java (cont.)

```java

   private class Row {
      private int row;
      public Row(int row) {
         this.row = row;
      }
      public Object getData() {
         int originalIndex = model.getRowIndex();
         model.setRowIndex(row);
         Object thisRowData = model.getRowData();
         model.setRowIndex(originalIndex);
         return thisRowData;
      }
   }

   public SortFilterModel() { // mandated by JSF spec
      this((Name[])null);
   }
   public SortFilterModel(Name[] names) { // recommended by JSF spec
      setWrappedData(names);
   }
   public SortFilterModel(DataModel model) {
      this.model = model;
      initializeRows();
   }

   public String sortByLast() {
      Arrays.sort(rows, byLast);
      return null;
   }

   public String sortByFirst() {
      Arrays.sort(rows, byFirst);
      return null;
   }

   public void setRowIndex(int rowIndex) {
      if(rowIndex == -1 || rowIndex >= model.getRowCount()) {
         model.setRowIndex(rowIndex);
      }
      else {
         model.setRowIndex(rows[rowIndex].row);
      }
   }
```

Listing 5–21 sorting/src/java/com/corejsf/SortFilterModel.java (cont.)

```java

   // The following methods delegate directly to the
   // decorated model

   public boolean isRowAvailable() {
      return model.isRowAvailable();
   }
   public int getRowCount() {
      return model.getRowCount();
   }
   public Object getRowData() {
      return model.getRowData();
   }
   public int getRowIndex() {
      return model.getRowIndex();
   }
   public Object getWrappedData() {
      return model.getWrappedData();
   }
   public void setWrappedData(Object data) {
      model.setWrappedData(data);
      initializeRows();
   }
   public void addDataModelListener(DataModelListener listener) {
      model.addDataModelListener(listener);
   }
   public DataModelListener[] getDataModelListeners() {
      return model.getDataModelListeners();
   }
   public void removeDataModelListener(DataModelListener listener) {
      model.removeDataModelListener(listener);
   }
   private void initializeRows() {
      int rowCnt = model.getRowCount();
      if(rowCnt != -1) {
         rows = new Row[rowCnt];
         for(int i=0; i < rowCnt; ++i) {
            rows[i] = new Row(i);
         }
      }
   }
}
```

Listing 5–22 sorting/src/java/com/corejsf/Name.java

```java
package com.corejsf;

public class Name {
   private String first;
   private String last;
   private boolean markedForDeletion = false;

   public Name(String first, String last) {
      this.first = first;
      this.last = last;
   }

   public void setFirst(String newValue) { first = newValue; }
   public String getFirst() { return first; }

   public void setLast(String newValue) { last = newValue; }
   public String getLast() { return last; }

   public boolean isMarkedForDeletion() { return markedForDeletion; }
   public void setMarkedForDeletion(boolean newValue) {
      markedForDeletion = newValue;
   }
}
```

Listing 5–23 sorting/src/java/com/corejsf/TableData.java

```java
package com.corejsf;

import javax.faces.model.DataModel;
import javax.faces.model.ArrayDataModel;

public class TableData {
   private DataModel filterModel = null;
   private static final Name[] names = {
      new Name("Anna", "Keeney"),
      new Name("John", "Wilson"),
      new Name("Mariko", "Randor"),
      new Name("William", "Dupont"),
   };

   public TableData() {
      filterModel = new SortFilterModel(new ArrayDataModel(names));
   }
```

Listing 5–23 sorting/src/java/com/corejsf/TableData.java (cont.)

```
   public DataModel getNames() {
      return filterModel;
   }
   public String deleteNames() {
      if (!getAnyNamesMarkedForDeletion())
         return null;

      Name[] currentNames = (Name[]) filterModel.getWrappedData();
      Name[] newNames = new Name[currentNames.length
         - getNumberOfNamesMarkedForDeletion()];

      for(int i = 0, j = 0; i < currentNames.length; ++i) {
         Name name = (Name) currentNames[i];
         if (!name.isMarkedForDeletion()) {
            newNames[j++] = name;
         }
      }
      filterModel.setWrappedData(newNames);
      return null;
   }

   public int getNumberOfNamesMarkedForDeletion() {
      Name[] currentNames = (Name[]) filterModel.getWrappedData();
      int cnt = 0;

      for(int i = 0; i < currentNames.length; ++i) {
         Name name = (Name) currentNames[i];
         if (name.isMarkedForDeletion())
            ++cnt;
      }
      return cnt;
   }

   public boolean getAnyNamesMarkedForDeletion() {
      Name[] currentNames = (Name[]) filterModel.getWrappedData();
      for(int i = 0; i < currentNames.length; ++i) {
         Name name = (Name) currentNames[i];
         if (name.isMarkedForDeletion())
            return true;
      }
      return false;
   }
}
```

Listing 5–24 sorting/web/WEB-INF/faces-config.xml

```
<?xml version="1.0"?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
   version="1.2">
   <application>
      <resource-bundle>
         <base-name>com.corejsf.messages</base-name>
         <var>msgs</var>
      </resource-bundle>
   </application>

   <managed-bean>
      <managed-bean-name>tableData</managed-bean-name>
      <managed-bean-class>com.corejsf.TableData</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>
</faces-config>
```

Listing 5–25 sorting/web/styles.css

```
.names {
   border: thin solid black;
}
.namesHeader {
   text-align: center;
   font-style: italic;
   color: Snow;
   background: Teal;
}
.last {
   height: 25px;
   text-align: center;
   background: MediumTurquoise;
}
.first {
   text-align: left;
   background: PowderBlue;
}
.caption {
   font-size: 0.9em;
   font-style: italic;
}
```

Listing 5–26 sorting/src/java/com/corejsf/messages.properties

```
windowTitle=Sorting Java Beans
pageTitle=An array of names:
firstColumnHeader=First Name
lastColumnHeader=Last Name
deleteColumnHeader=Delete
deleteButtonText=Delete selected names
```

javax.faces.model.DataModel

- `int getRowCount()`
  Returns the total number of rows, if known; otherwise, it returns –1. The `ResultSetDataModel` always returns –1 from this method.
- `Object getRowData()`
  Returns the data associated with the current row.
- `boolean isRowAvailable()`
  Returns `true` if there is valid data at the current row index.
- `int getRowIndex()`
  Returns the index of the current row.
- `void setRowIndex(int index)`
  Sets the current row index and updates the scoped variable representing the current item in the collection (that variable is specified with the `var` attribute of `h:dataTable`).
- `void addDataModelListener(DataModelListener listener)`
  Adds a data model listener that is notified when the row index changes.
- `void removeDataModelListener(DataModelListener listener)`
  Removes a data model listener.
- `void setWrappedData(Object obj)`
  Sets the object that a data model wraps.
- `Object getWrappedData()`
  Returns a data model's wrapped data.

Scrolling Techniques

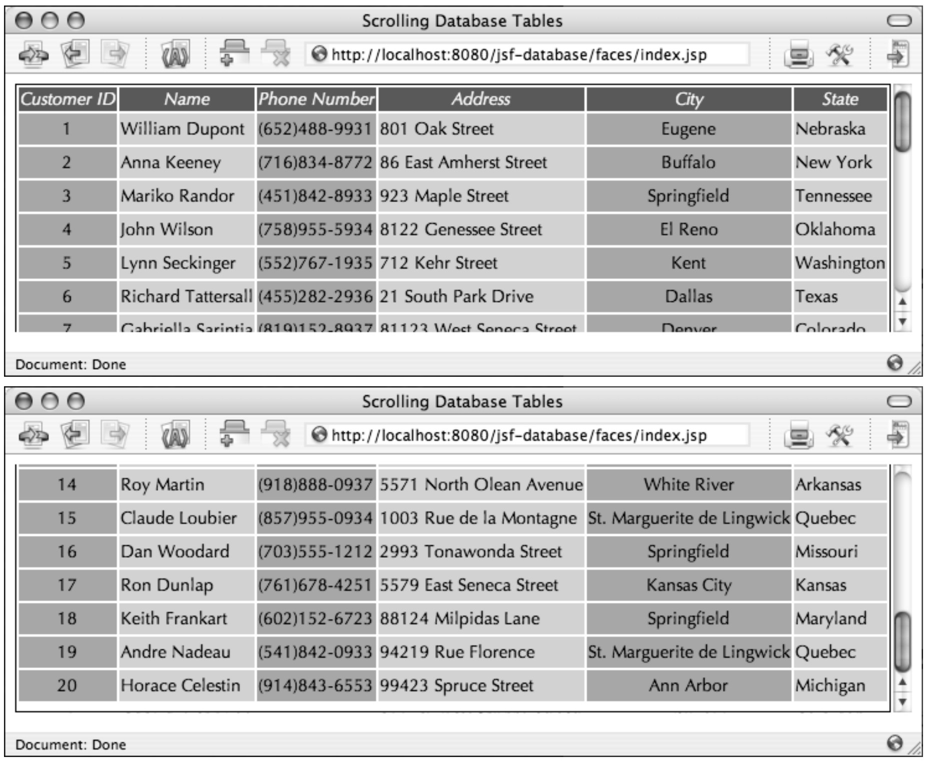
There are two ways to scroll through tables with lots of rows: with a scrollbar or with some other type of control that moves through the rows. We explore both techniques in this section.

Scrolling with a Scrollbar

Scrolling with a scrollbar is the simplest solution. Wrap your `h:dataTable` in an HTML `div`, like this:

```
<div style="overflow:auto; width:100%; height:200px;">
   <h:dataTable...>
      <h:column>
         ...
      </h:column>
      ...
   </h:dataTable>
</div>
```

The application shown in Figure 5–17 is identical to the application discussed in “Database Tables” on page 191, except that the data table is placed in a scrollable `div`, as shown above.


Scrolling Database Tables
http://localhost:8080/jsf-database/faces/index.jsp
Document: Done

Customer ID	Name	Phone Number	Address	City	State
1	William Dupont	(652)488-9931	801 Oak Street	Eugene	Nebraska
2	Anna Keeney	(716)834-8772	86 East Amherst Street	Buffalo	New York
3	Mariko Randor	(451)842-8933	923 Maple Street	Springfield	Tennessee
4	John Wilson	(758)955-5934	8122 Genessee Street	El Reno	Oklahoma
5	Lynn Seckinger	(552)767-1935	712 Kehr Street	Kent	Washington
6	Richard Tattersall	(455)282-2936	21 South Park Drive	Dallas	Texas
7	Gabriella Sarintia	(819)152-8937	81123 West Seneca Street	Denver	Colorado

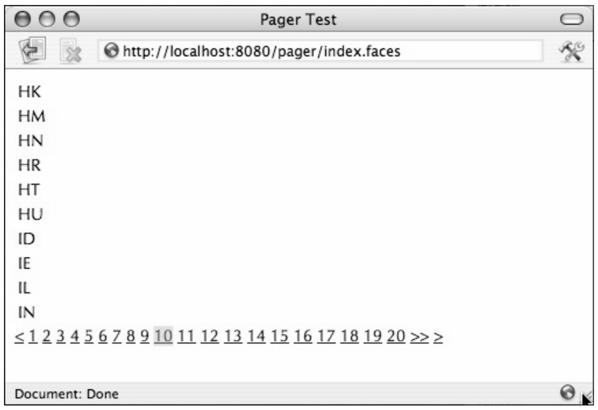
14	Roy Martin	(918)888-0937	5571 North Olean Avenue	White River	Arkansas
15	Claude Loubier	(857)955-0934	1003 Rue de la Montagne	St. Marguerite de Lingwick	Quebec
16	Dan Woodard	(703)555-1212	2993 Tonawonda Street	Springfield	Missouri
17	Ron Dunlap	(761)678-4251	5579 East Seneca Street	Kansas City	Kansas
18	Keith Frankart	(602)152-6723	88124 Milpidas Lane	Springfield	Maryland
19	Andre Nadeau	(541)842-0933	94219 Rue Florence	St. Marguerite de Lingwick	Quebec
20	Horace Celestin	(914)843-6553	99423 Spruce Street	Ann Arbor	Michigan

Figure 5–17   Scrolling a table with a scrollable `div`

Scrollbars are nice from a usability standpoint, but they can be expensive for large tables because all the table data is loaded at once. A less resource-intensive alternative is to scroll through tables with page widgets, an approach that requires only one page of data at a time.

Scrolling with Pager Widgets

Scrolling with pager widgets is more efficient than scrolling with a scrollable DIV, but it is also considerably more complex. In Chapter 13, we show you how to implement a pager widget that you can use with any table created with `h:dataTable` (see "How do I show a large data set, one page at a time?" on page 638 of Chapter 13). Figure 5–18 shows an example of that pager.



```
HK
HM
HN
HR
HT
HU
ID
IE
IL
IN
< 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 >> >
```

Figure 5–18 Scrolling with a JSF pager

The application shown in Figure 5–18 uses a data table that displays the ISO country codes for locales. We obtain that list by calling `java.util.Locale.getISOCountries()`, a `static` method that returns an array of strings.