Chapter 8

User interfaces are typically the most volatile aspect of web applications during development, so it is crucial to create flexible and extensible interfaces. This chapter shows you how to achieve that flexibility and extensibility by including common content. First we discuss standard JSP mechanisms—JSP includes and JSTL imports—that you can use to include common content in a JSF application. Next, we explore the use of the Apache Tiles package—which lets you encapsulate layout in addition to content, among other handy features—with JSF.

Common Layouts

Many popular web sites, such as `nytimes.com`, `java.sun.com`, or `amazon.com`, use a common layout for their web pages. For example, all three of the web sites listed above use a header-menu-content layout, as depicted in Figure 8–1.

You can use HTML frames to achieve the layout shown in Figure 8–1, but frames are undesirable for several reasons. For example, frames make it hard for users to bookmark pages. Frames also generate separate requests, which can be problematic for web applications. Including content, which is the focus of this chapter, is generally preferred over frames.

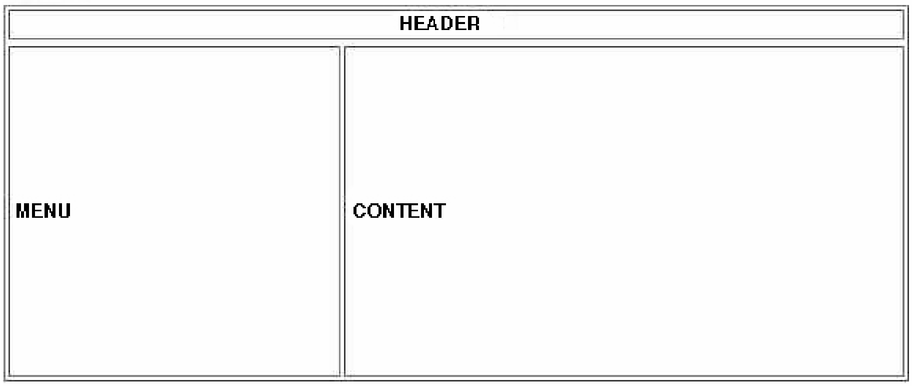


Figure 8–1 A typical web page layout

A Book Viewer and a Library

To illustrate implementing layouts, including common content, and using Tiles, we discuss two applications in this chapter: a book viewer and a library. Those applications are shown in Figure 8–2 and Figure 8–3, respectively.

The book viewer is intuitive. If you click a chapter link, that chapter is shown in the content region of the web page. The library is an extension of the book viewer that lets you view more than one book. You can select books from the menu at the top of the web page.

The book viewer addresses the following topics:

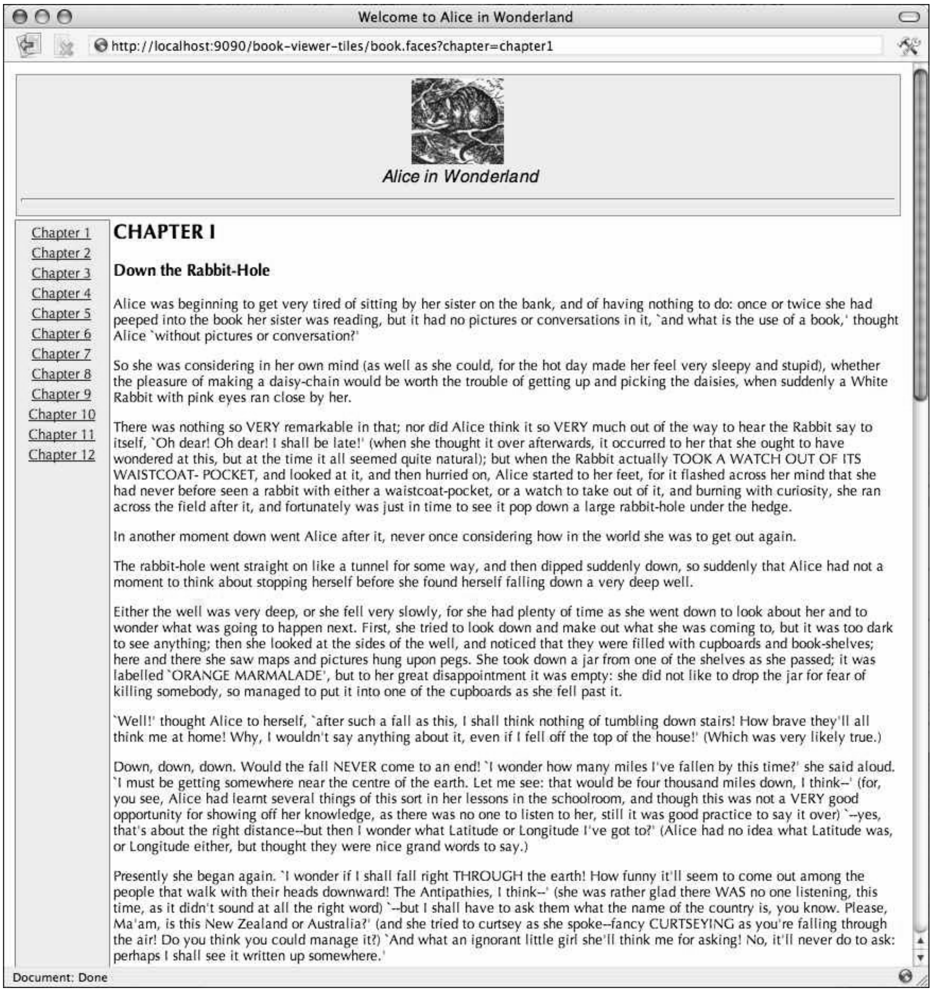
- "Monolithic JSF Pages" on page 320
- "Common Content Inclusion" on page 326
- "Looking at Tiles" on page 331
- "Parameterizing Tiles" on page 334
- "Extending Tiles" on page 335

The library illustrates these Tiles features:

- "Nested Tiles" on page 339
- "Tile Controllers" on page 341

Coverage of the book viewer begins in the next section, "The Book Viewer" on page 318. The library is discussed in "The Library" on page 339.

NOTE: For the examples in this chapter, we downloaded *Alice in Wonderland* and *Peter Pan* from the Project Gutenberg web site (`http://promo.net/pg/`), chopped them up into chapters, and converted them to HTML.



CHAPTER I

Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, `and what is the use of a book,' thought Alice `without pictures or conversation?'

So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so VERY remarkable in that; nor did Alice think it so VERY much out of the way to hear the Rabbit say to itself, `Oh dear! Oh dear! I shall be late!' (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually TOOK A WATCH OUT OF ITS WAISTCOAT- POCKET, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.

In another moment down went Alice after it, never once considering how in the world she was to get out again.

The rabbit-hole went straight on like a tunnel for some way, and then dipped suddenly down, so suddenly that Alice had not a moment to think about stopping herself before she found herself falling down a very deep well.

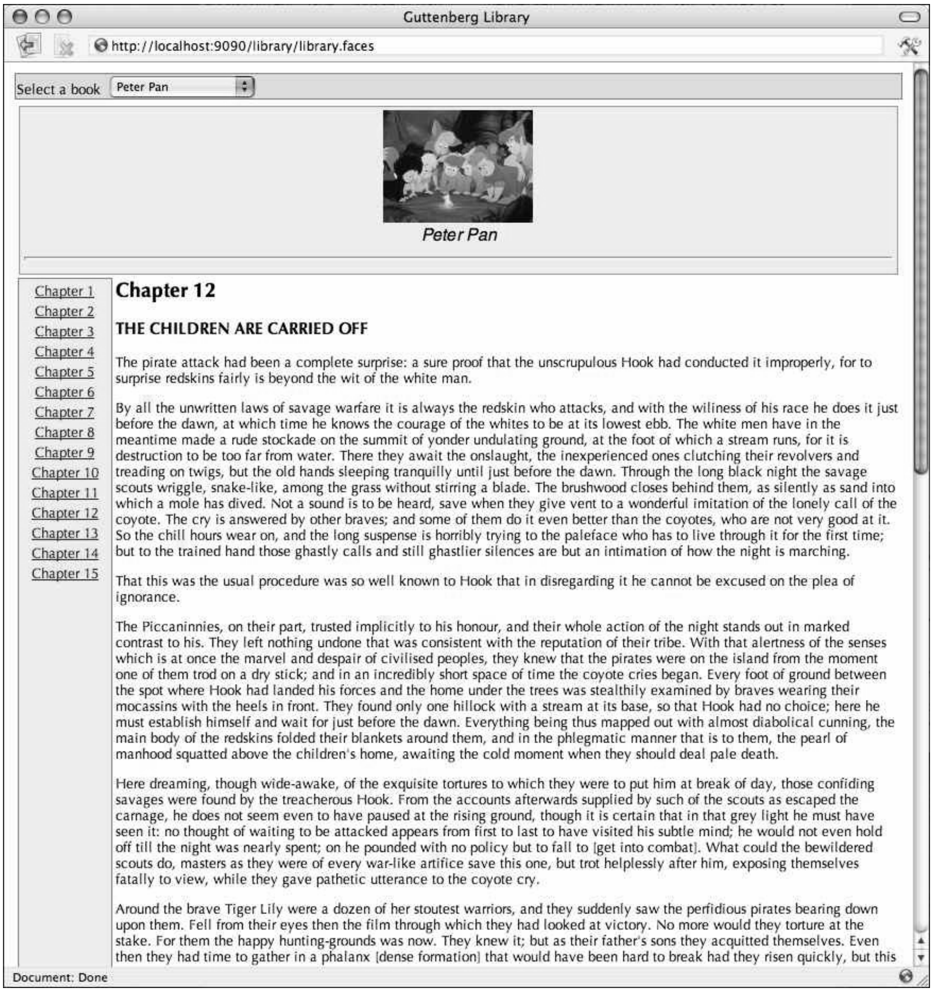
Either the well was very deep, or she fell very slowly, for she had plenty of time as she went down to look about her and to wonder what was going to happen next. First, she tried to look down and make out what she was coming to, but it was too dark to see anything; then she looked at the sides of the well, and noticed that they were filled with cupboards and book-shelves; here and there she saw maps and pictures hung upon pegs. She took down a jar from one of the shelves as she passed; it was labelled `ORANGE MARMALADE', but to her great disappointment it was empty: she did not like to drop the jar for fear of killing somebody, so managed to put it into one of the cupboards as she fell past it.

`Well!' thought Alice to herself, `after such a fall as this, I shall think nothing of tumbling down stairs! How brave they'll all think me at home! Why, I wouldn't say anything about it, even if I fell off the top of the house!' (Which was very likely true.)

Down, down, down. Would the fall NEVER come to an end! `I wonder how many miles I've fallen by this time?' she said aloud. `I must be getting somewhere near the centre of the earth. Let me see: that would be four thousand miles down, I think--' (for, you see, Alice had learnt several things of this sort in her lessons in the schoolroom, and though this was not a VERY good opportunity for showing off her knowledge, as there was no one to listen to her, still it was good practice to say it over) `--yes, that's about the right distance--but then I wonder what Latitude or Longitude I've got to?' (Alice had no idea what Latitude was, or Longitude either, but thought they were nice grand words to say.)

Presently she began again. `I wonder if I shall fall right THROUGH the earth! How funny it'll seem to come out among the people that walk with their heads downward! The Antipathies, I think--' (she was rather glad there WAS no one listening, this time, as it didn't sound at all the right word) `--but I shall have to ask them what the name of the country is, you know. Please, Ma'am, is this New Zealand or Australia?' (and she tried to curtsey as she spoke--fancy CURTSEYING as you're falling through the air! Do you think you could manage it?) `And what an ignorant little girl she'll think me for asking! No, it'll never do to ask: perhaps I shall see it written up somewhere.'

Figure 8–2 The book viewer



Chapter 1
Chapter 2
Chapter 3
Chapter 4
Chapter 5
Chapter 6
Chapter 7
Chapter 8
Chapter 9
Chapter 10
Chapter 11
Chapter 12
Chapter 13
Chapter 14
Chapter 15

Chapter 12

THE CHILDREN ARE CARRIED OFF

The pirate attack had been a complete surprise: a sure proof that the unscrupulous Hook had conducted it improperly, for to surprise redskins fairly is beyond the wit of the white man.

By all the unwritten laws of savage warfare it is always the redskin who attacks, and with the wiliness of his race he does it just before the dawn, at which time he knows the courage of the whites to be at its lowest ebb. The white men have in the meantime made a rude stockade on the summit of yonder undulating ground, at the foot of which a stream runs, for it is destruction to be too far from water. There they await the onslaught, the inexperienced ones clutching their revolvers and treading on twigs, but the old hands sleeping tranquilly until just before the dawn. Through the long black night the savage scouts wriggle, snake-like, among the grass without stirring a blade. The brushwood closes behind them, as silently as sand into which a mole has dived. Not a sound is to be heard, save when they give vent to a wonderful imitation of the lonely call of the coyote. The cry is answered by other braves; and some of them do it even better than the coyotes, who are not very good at it. So the chill hours wear on, and the long suspense is horribly trying to the paleface who has to live through it for the first time; but to the trained hand those ghastly calls and still ghastlier silences are but an intimation of how the night is marching.

That this was the usual procedure was so well known to Hook that in disregarding it he cannot be excused on the plea of ignorance.

The Piccaninnies, on their part, trusted implicitly to his honour, and their whole action of the night stands out in marked contrast to his. They left nothing undone that was consistent with the reputation of their tribe. With that alertness of the senses which is at once the marvel and despair of civilised peoples, they knew that the pirates were on the island from the moment one of them trod on a dry stick; and in an incredibly short space of time the coyote cries began. Every foot of ground between the spot where Hook had landed his forces and the home under the trees was stealthily examined by braves wearing their mocassins with the heels in front. They found only one hillock with a stream at its base, so that Hook had no choice; here he must establish himself and wait for just before the dawn. Everything being thus mapped out with almost diabolical cunning, the main body of the redskins folded their blankets around them, and in the phlegmatic manner that is to them, the pearl of manhood squatted above the children's home, awaiting the cold moment when they should deal pale death.

Here dreaming, though wide-awake, of the exquisite tortures to which they were to put him at break of day, those confiding savages were found by the treacherous Hook. From the accounts afterwards supplied by such of the scouts as escaped the carnage, he does not seem even to have paused at the rising ground, though it is certain that in that grey light he must have seen it: no thought of waiting to be attacked appears from first to last to have visited his subtle mind; he would not even hold off till the night was nearly spent; on he pounded with no policy but to fall to [get into combat]. What could the bewildered scouts do, masters as they were of every war-like artifice save this one, but trot helplessly after him, exposing themselves fatally to view, while they gave pathetic utterance to the coyote cry.

Around the brave Tiger Lily were a dozen of her stoutest warriors, and they suddenly saw the perfidious pirates bearing down upon them. Fell from their eyes then the film through which they had looked at victory. No more would they torture at the stake. For them the happy hunting-grounds was now. They knew it; but as their father's sons they acquitted themselves. Even then they had time to gather in a phalanx [dense formation] that would have been hard to break had they risen quickly, but this

Figure 8–3 The library

The Book Viewer

The book viewer is rather limited in scope. It supports only a single book, which is a managed bean that we define in the faces configuration file. The name of that bean is `book`.

The `book` bean has these properties:

- `titleKey`
- `image`

- numChapters
- chapterKeys

The titleKey property represents a key in a resource bundle for the book's title. In the book viewer's properties file we have the key/value pair titleKey=Alice in Wonderland. When we display the book's title, we use the titleKey property, like this:

```
<h:outputText value="#{msgs[book.titleKey]}"/>
```

The image property is a string. The application interprets that string as a URL and loads it in the book viewer's header like this:

```
<h:graphicImage url="#{book.image}"/>
```

The chapterKeys property is a read-only list of keys, one for each chapter. The book viewer populates the book viewer's menu with corresponding values from a resource bundle:

```
<h:dataTable value="#{book.chapterKeys}" var="chapterKey">
   <h:commandLink>
      <h:outputText value="#{msgs[chapterKey]}"/>
      ...
   </h:commandLink>
</h:dataTable>
```

The Book class uses the numChapters property to compute the chapter keys.

The implementation of the Book class is rather mundane. You can see it in Listing 8–3 on page 324. Here is how we define an instance of the Book class in faces-config.xml:

```
<faces-config>
   <!-- The book -->
      <managed-bean>
      <managed-bean-name>book</managed-bean-name>
      <managed-bean-class>com.corejsf.Book</managed-bean-class>
      <managed-bean-scope>request</managed-bean-scope>

      <managed-property>
         <property-name>titleKey</property-name>
         <value>aliceInWonderland</value>
      </managed-property>

      <managed-property>
         <property-name>image</property-name>
         <value>cheshire.jpg</value>
      </managed-property>
      <managed-property>
         <property-name>numChapters</property-name>
         <property-class>java.lang.Integer</property-class>
         <value>12</value>
      </managed-property>
   </managed-bean>
</faces-config>
```

There are many ways to implement page layout. In this section, we look at three options: a monolithic JSF page, inclusion of common content, and Tiles.

NOTE: We do not set the book's `chapterKeys` property in `faces-config.xml`. This is because the `Book` class creates that list of chapter keys for us. All we have to do is define the `numChapters` property.

Monolithic JSF Pages

A monolithic JSF page is perhaps the quickest way to implement the book viewer, shown in Figure 8–2. For example, here is a naive implementation:

```
<!-- A panel grid, which resides in a form, for the entire page --%>
<h:panelGrid columns="2" styleClass="book"
   columnClasses="menuColumn, chapterColumn">

   <!-- The header, containing an image, title, and horizontal rule --%>
   <f:facet name="header">
      <h:panelGrid columns="1" styleClass="bookHeader">
         <h:graphicImage value="#{book.image}"/>
         <h:outputText value="#{msgs[book.titleKey]}" styleClass='bookTitle'/>
         <hr>
      </h:panelGrid>
   </f:facet>

   <!-- Column 1 of the panel grid: The menu, which consists of chapter links --%>
   <h:dataTable value="#{book.chapterKeys}" var="chapterKey"
            styleClass="links" columnClasses="linksColumn">
      <h:column>
         <h:commandLink>
            <h:outputText value="#{msgs[chapterKey]}"/>
            <f:param name="chapter" value="#{chapterKey}"/>
         </h:commandLink>
      </h:column>
   </h:dataTable>

   <!-- Column 2 of the panel grid: The chapter content --%>
   <c:import url="${param.chapter}.html"/>
</h:panelGrid>
```

The book viewer is implemented with a panel grid with two columns. The header region is populated with an image, text, and HTML horizontal rule. Besides the header, the panel grid has only one row—the menu occupies the left column and the current chapter is displayed in the right column.

The menu is composed of chapter links. By default, `Book.getChapterKeys()` returns a list of strings that looks like this:

```
chapter1
chapter2
...
chapterN
```

`ChapterN` represents the last chapter in the book. In the book viewer's resource bundle, we define values for those keys:

```
chapter1=Chapter 1
chapter2=Chapter 2
...
```

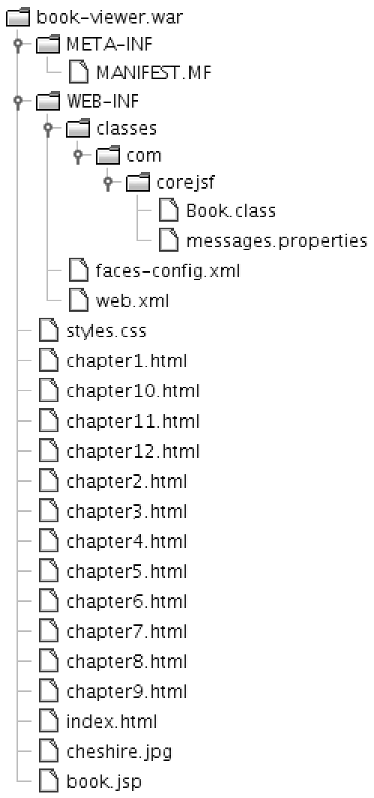
To create chapter links, we use `h:dataTable` to iterate over the book's chapter keys. For every chapter, we create a link whose text corresponds to the chapter key's value with this expression: `#{msgs[chapterKey]}`. So, for example, we wind up with "Chapter 1" … "Chapter 12" displayed in the menu when the number of chapters is 12.

The right column is reserved for chapter content. That content is included with JSTL's `c:import` tag.

The directory structure for the book viewer is shown in Figure 8–4. The monolithic JSF version of the book viewer is shown in Listing 8–1 through Listing 8–5.

NOTE: Notice the `f:param` tag inside `h:commandLink`. The JSF framework turns that parameter into a request parameter—named `chapter`—when the link is activated. When the page is reloaded, that request parameter is used to load the chapter's content, like this:

```
<c:import url="${param.chapter}"/>
```



```
book-viewer.war
  META-INF
    MANIFEST.MF
  WEB-INF
    classes
      com
        corejsf
          Book.class
          messages.properties
    faces-config.xml
    web.xml
  styles.css
  chapter1.html
  chapter10.html
  chapter11.html
  chapter12.html
  chapter2.html
  chapter3.html
  chapter4.html
  chapter5.html
  chapter6.html
  chapter7.html
  chapter8.html
  chapter9.html
  index.html
  cheshire.jpg
  book.jsp
```

Figure 8–4 The directory structure of the book viewer

Listing 8–1 book-viewer/web/book.jsp

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

   <f:view>
      <f:loadBundle basename="com.corejsf.messages" var="msgs"/>
      <head>
         <link href="styles.css" rel="stylesheet" type="text/css"/>
         <title><h:outputText value="#{msgs.bookWindowTitle}"/></title>
      </head>

      <body>
         <h:form>
```

Listing 8–1 book-viewer/web/book.jsp (cont.)

```
            <h:panelGrid columns="2" styleClass="book"
                   columnClasses="menuColumn, chapterColumn">
               <f:facet name="header">
                  <h:panelGrid columns="1" styleClass="bookHeader">
                     <h:graphicImage value="#{book.image}"/>
                     <h:outputText value="#{msgs[book.titleKey]}"
                              styleClass='bookTitle'/>
                     <hr/>
                  </h:panelGrid>
               </f:facet>

               <h:dataTable value="#{book.chapterKeys}" var="chapterKey"
                        styleClass="links" columnClasses="linksColumn">
                  <h:column>
                     <h:commandLink>
                        <h:outputText value="#{msgs[chapterKey]}"/>
                        <f:param name="chapter" value="#{chapterKey}"/>
                     </h:commandLink>
                  </h:column>
               </h:dataTable>

               <c:import url="${param.chapter}.html"/>
            </h:panelGrid>
         </h:form>
      </body>
   </f:view>
</html>
```

Listing 8–2 book-viewer/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">
   <managed-bean>
      <managed-bean-name>book</managed-bean-name>
      <managed-bean-class>com.corejsf.Book</managed-bean-class>
      <managed-bean-scope>request</managed-bean-scope>

```

Listing 8–2 book-viewer/web/WEB-INF/faces-config.xml (cont.)

```
      <managed-property>
         <property-name>titleKey</property-name>
         <value>aliceInWonderland</value>
      </managed-property>

      <managed-property>
         <property-name>image</property-name>
         <value>cheshire.jpg</value>
      </managed-property>

      <managed-property>
         <property-name>numChapters</property-name>
         <property-class>java.lang.Integer</property-class>
         <value>12</value>
      </managed-property>
   </managed-bean>
</faces-config>
```

Listing 8–3 book-viewer/src/java/com/corejsf/Book.java

```
package com.corejsf;

import java.util.LinkedList;
import java.util.List;

public class Book {
   private String titleKey;
   private String image;
   private int numChapters;
   private List<String> chapterKeys = null;

   // PROPERTY: titleKey
   public void setTitleKey(String titleKey) { this.titleKey = titleKey; }
   public String getTitleKey() { return titleKey; }

   // PROPERTY: image
   public void setImage(String image) { this.image = image; }
   public String getImage() { return image; }

   // PROPERTY: numChapters
   public void setNumChapters(int numChapters) { this.numChapters = numChapters;}
   public int getNumChapters() { return numChapters; }

```

Listing 8–3 book-viewer/src/java/com/corejsf/Book.java (cont.)

```
   // PROPERTY: chapterKeys
   public List<String> getChapterKeys() {
      if(chapterKeys == null) {
         chapterKeys = new LinkedList<String>();
         for(int i=1; i <= numChapters; ++i)
            chapterKeys.add("chapter" + i);
      }
      return chapterKeys;
   }
}
```

Listing 8–4 book-viewer/src/java/com/corejsf/messages.properties

```
bookWindowTitle=Welcome to Alice in Wonderland
aliceInWonderland=Alice in Wonderland

chapter1=Chapter 1
chapter2=Chapter 2
chapter3=Chapter 3
chapter4=Chapter 4
chapter5=Chapter 5
chapter6=Chapter 6
chapter7=Chapter 7
chapter8=Chapter 8
chapter9=Chapter 9
chapter10=Chapter 10
chapter11=Chapter 11
chapter12=Chapter 12
chapter13=Chapter 13
chapter14=Chapter 14
chapter15=Chapter 15
```

Listing 8–5 book-viewer/web/styles.css

```
.bookHeader {
   width: 100%;
   text-align: center;
   background-color: #eee;
   padding: 0 px;
   border: thin solid CornflowerBlue;
}
```

Listing 8–5 book-viewer/web/styles.css (cont.)

```
.bookTitle {
   text-align: center;
   font-style: italic;
   font-size: 1.3em;
   font-family: Helvetica;
}
.book {
   vertical-align: top;
   width: 100%;
   height: 100%;
}
.menuColumn {
   vertical-align: top;
   background-color: #eee;
   width: 100px;
   border: thin solid #777;
}
.chapterColumn {
   vertical-align: top;
   text-align: left;
   width: *;
}
```

Common Content Inclusion

A monolithic JSF page is a poor choice for the book viewer because the JSF page is difficult to modify. Also, realize that our monolithic JSF page represents two things: layout and content.

Layout is implemented with an `h:panelGrid` tag, and content is represented by various JSF tags, such as `h:graphicImage`, `h:outputText`, `h:commandLink`, and the book chapters. Realize that *with a monolithic JSF page, we cannot reuse content or layout*.

In the next section, we concentrate on including content. In "Looking at Tiles" on page 331, we discuss including layout.

Content Inclusion in JSP-Based Applications

Instead of cramming a bunch of code into a monolithic JSF page, as we did in Listing 8–1 on page 322, it is better to include common content so you can reuse that content in other JSF pages. With JSP, you have three choices for including content:

- `<%@ include file="header.jsp"% >`
- `<jsp:include page="header.jsp"/>`
- `<c:import url="header.jsp"/>`

The first choice listed above—the JSP `include` directive—includes the specified file before the enclosing JSF page is compiled to a servlet. However, the `include` directive suffers from an important limitation: If the included file's content changes after the enclosing page was first processed, those changes are not reflected in the enclosing page. That means you must manually update the enclosing pages—whether the including pages changed or not—whenever included content changes.

The last two choices listed above include the content of a page at runtime and merge the included content with the including JSF page. Because the inclusion happens at runtime, changes to included pages are always reflected when the enclosing page is redisplayed. For that reason, `jsp:include` and `c:import` are usually preferred to the `include` directive.

The `c:import` tag works just like `jsp:include`, but it has more features—for example, `c:import` can import resources from another web application, whereas `jsp:include` cannot. Also, prior to JSP 2.0, you cannot use JSP expressions for `jsp:include` attributes, whereas you can with `c:import`. Remember that you must import the JSTL core tag library to use `c:import`.

Throughout this chapter, we use `c:import` for consistency. You can use either `jsp:include` or `c:import` to dynamically include content. If you do not need `c:import`'s extra features, then it is ever-so-slightly easier to use `jsp:include` because you do not need to import the JSTL core tag library.

JSF-Specific Considerations

Regardless of whether you include content with the `include` directive, `jsp:include`, or `c:import`, you must take into account two special considerations when you include content in a JavaServer Faces application:

1. You must wrap included JSF tags in an `f:subview` tag.
2. Included JSF tags cannot contain f:`view` tags.

The first rule applies to included content that contains JSF tags. For example, the book viewer should encapsulate header content in its own JSF page so that we can reuse that content:

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

```
<h:panelGrid columns="1" styleClass="header">
   <h:graphicImage value="books/book/cheshire.jpg"/>
   <h:outputText value="#{msgs.bookTitle}" styleClass="bookTitle"/>
   ...
</h:panelGrid>
```

Now we can include that content from the original JSF page:

```
<%-- This is from the original JSF page --%>
<f:view>
   ...
   <f:subview id="header">
      <c:import url="header.jsp"/>
   </f:subview>
   ...
</f:view>
```

You must assign an ID to each subview. The standard convention for including content is to name the subview after the imported JSF page.

JSF views, which are normally web pages, can contain an unlimited number of subviews. But there can be only one view. Because of that restriction, included JSF tags—which must be wrapped in a subview—cannot contain `f:view` tags.

CAUTION: The `book-viewer-include` application maps the Faces servlet to `*.faces`. That means you can start the application with this URL: `http://www.localhost:8080/book-viewer-include/book.faces`. The Faces servlet maps `books.faces` to `books.jsp`. However, you cannot use the `faces` suffix when you use `c:import`. If you use `c:import`, you must use the `jsp` suffix.

Content Inclusion in the Book Viewer

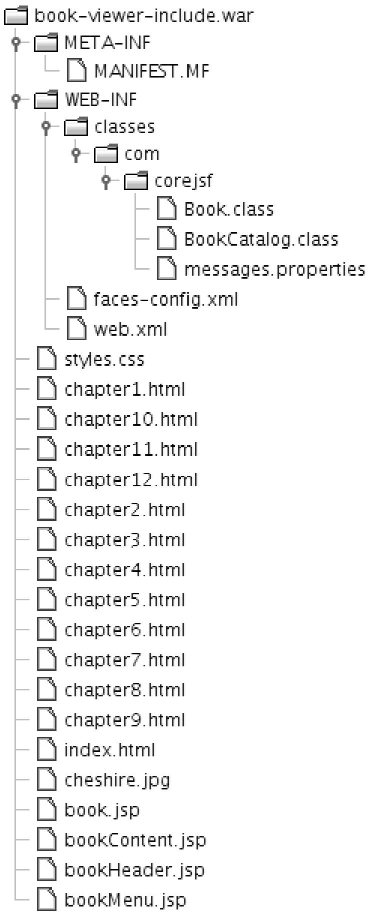
To include content in the book viewer, we split our monolithic JSF page into four files: the original JSF page, `/header.jsp`, `/menu.jsp`, and `/content.jsp`. We include the header, menu, and content in the original JSF page:

```
<h:panelGrid columns="2" styleClass="book"
   columnClasses="menuColumn, contentColumn">

   <f:facet name="header">
      <f:subview id="header">
         <c:import url="header.jsp"/>
      </f:subview>
   </f:facet>

   <f:subview id="menu">
```

```
        <c:import url="menu.jsp"/>
      </f:subview>

       <c:import url="content.jsp"/>
   </h:panelGrid>
   ...
```

This code is much cleaner than the original JSF page listed in Listing 8–1, so it is easier to understand, maintain, and modify. But more important, we are now free to reuse the header, menu, and content for other views.

The directory structure for the book viewer with includes example is shown in Figure 8–5. Listing 8–6 through Listing 8–9 show the JSF pages for the book, its header, menu, and content.



```
book-viewer-include.war
  META-INF
    MANIFEST.MF
  WEB-INF
    classes
      com
        corejsf
          Book.class
          BookCatalog.class
          messages.properties
    faces-config.xml
    web.xml
  styles.css
  chapter1.html
  chapter10.html
  chapter11.html
  chapter12.html
  chapter2.html
  chapter3.html
  chapter4.html
  chapter5.html
  chapter6.html
  chapter7.html
  chapter8.html
  chapter9.html
  index.html
  cheshire.jpg
  book.jsp
  bookContent.jsp
  bookHeader.jsp
  bookMenu.jsp
```

Figure 8–5 The directory structure of the book viewer with includes

Listing 8–6 book-viewer-include/web/book.jsp

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

   <f:view>
      <f:loadBundle basename="com.corejsf.messages" var="msgs"/>
      <head>
         <link href="styles.css" rel="stylesheet" type="text/css"/>
         <title><h:outputText value="#{msgs.bookWindowTitle}"/></title>
      </head>

      <body>
         <h:form>
            <h:panelGrid columns="2" styleClass="book"
                   columnClasses="menuColumn, chapterColumn">
               <f:facet name="header">
                  <f:subview id="header">
                     <c:import url="/bookHeader.jsp"/>
                  </f:subview>
               </f:facet>

               <f:subview id="menu">
                  <c:import url="/bookMenu.jsp"/>
               </f:subview>

               <c:import url="/bookContent.jsp"/>
            </h:panelGrid>
         </h:form>
      </body>
   </f:view>
</html>
```

Listing 8–7 book-viewer-include/web/bookHeader.jsp

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

<h:panelGrid columns="1" styleClass="bookHeader">
   <h:graphicImage value="#{book.image}"/>
   <h:outputText value="#{msgs[book.titleKey]}" styleClass="bookTitle"/>
   <hr>
</h:panelGrid>
```

Listing 8–8 book-viewer-include/web/bookMenu.jsp

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

<h:dataTable value="#{book.chapterKeys}" var="chapterKey"
        styleClass="links" columnClasses="linksColumn">
   <h:column>
      <h:commandLink>
         <h:outputText value="#{msgs[chapterKey]}"/>
         <f:param name="chapter" value="#{chapterKey}"/>
      </h:commandLink>
   </h:column>
</h:dataTable>
```

Listing 8–9 book-viewer-include/web/bookContent.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:import url="${param.chapter}.html"/>
```

Looking at Tiles

We have seen how to encapsulate and include content and how that strategy increases flexibility—it is much easier to reuse content if you include it rather than mixing it all in one file. Now that you can create user interfaces with pluggable content, you may be satisfied with that level of flexibility and reuse—but wait, there's more.

In addition to *encapsulating content*, you can use Tiles to *encapsulate layout*. For the application shown in Figure 8–2 on page 317, encapsulating layout means making the layout code—the `h:panelGrid` and its contents listed in Listing 8–6 on page 330—available for reuse. As it stands in Listing 8–6, that layout code can only be used by the JSF page shown in Figure 8–2. If you implement JSF pages with identical layouts, you must *replicate that layout code for every page*.

With Tiles, you define a single layout that can be reused by multiple *tiles*, which are nothing more mysterious than imported JSP pages. *Tiles lets you implement layout code once and reuse it among many pages*.

But reusing layout is just the beginning of the Tiles bag of tricks. You can do more:

- Nest tiles
- Extend tiles

- Restrict tiles to users of a particular role
- Attach controllers (Java objects) to tiles that are invoked just before their tile is displayed

Those are the core features that Tiles offers in the pursuit of the ultimate flexibility in crafting web-based user interfaces.

Installing Tiles

To use Tiles, you need the standalone Tiles JAR file. That JAR file can be found in the source code for this book.

Once you have the Tiles JAR file, follow these steps to install Tiles in your application:

1. Copy the Tiles JAR file to your application's `WEB-INF/lib` directory.
2. Add the Tiles servlet to your deployment descriptor (`web.xml`). Use the `load-on-startup` element to ensure that the Tiles servlet is loaded when your application starts.

Your deployment descriptor should look similar to the following:

```
<?xml version="1.0"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
          http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
       version="2.4">
   ...
   <servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet>
      <servlet-name>Tiles Servlet</servlet-name>
      <servlet-class>org.apache.tiles.servlets.TilesServlet</servlet-class>
       <load-on-startup>2</load-on-startup>
   </servlet>
   ...
</web-app>
```

Using Tiles with the Book Viewer

Using Tiles with JSF is a three-step process:

1. Use `tiles:insert` to insert a tile definition in a JSF page.
2. Define the tile in your Tiles configuration file.
3. Implement the tile's layout.

For the book viewer, we start in `book.jsp`, where we insert a tile named `book`:

```
...
<%@ taglib uri="http://jakarta.apache.org/tiles" prefix="tiles" %>
...
<h:form>
   <tiles:insert definition="book" flush="false"/>
</h:form>
...
```

We define the `book` tile in `/WEB-INF/tiles.xml`:

```
<definition name="book" path="/headerMenuContentLayout.jsp">
    <put name="header" value="/bookHeader.jsp"/>
    <put name="menu" value="/bookMenu.jsp"/>
    <put name="content" value="/bookContent.jsp"/>
</definition>
```

The previous snippet of XML defines a tile. The tile's layout is specified with the `definition` element's `path` attribute. The tile attributes, specified with `put` elements, are used by the layout. That layout looks like this:

```
<%-- this is /headerMenuContentLayout.jsp --%>

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

<h:panelGrid columns="2" styleClass="gridClass"
   headerClass="headerClass"
   columnClasses="menuClass, contentClass">

   <f:facet name="header">
      <f:subview id="header">
         <tiles:insert attribute="header" flush="false"/>
      </f:subview>
   </f:facet>

   <f:subview id="menu">
      <tiles:insert attribute="menu" flush="false"/>
   </f:subview>
```

```
      <f:subview id="content">
         <tiles:insert attribute="content" flush="false"/>
      </f:subview>
   </h:panelGrid>
```

The `tiles:insert` tag dynamically includes content. That content is the value of the `attribute` tag of `tiles:insert`. For example, the preceding code inserts the `header` attribute. That attribute's value is `/bookHeader.jsp`, so `tiles:insert` dynamically includes that file.

Notice that we specified a `flush="false"` attribute for the `tiles:insert` tag. That is necessary for most modern servlet containers because those containers disallow buffer flushing inside custom tags. If your servlet container throws an exception stating that you cannot flush from a custom tag, then you know you have forgotten to specify that attribute, which is `true` by default.

What have we gained by using Tiles in this example? *We have encapsulated layout so that we can reuse it in other tiles, instead of replicating that layout code from one JSF page to another.* For example, you could reuse the book viewer's layout, implemented in `/headerMenuContentLayout.jsp`, for other pages in the application that have the same layout.

Parameterizing Tiles

There is one flaw to the layout listed in the previous section: It hardcodes CSS classes, namely `gridClass`, `headerClass`, `menuClass`, and `contentClass`. This means that every web page using the header-menu-content layout will have the same look and feel. It would be better if we could parameterize the CSS class names. That way, other tiles with a header-menu-content layout could define their own look and feel.

Next, we look at how we can do that. First, we add three attributes to the `book` tile:

```
<definition name="book" path="/headerMenuContentLayout.jsp">
   <put name="headerClass"  value="headerClass"/>
   <put name="menuClass" value="menuClass"/>
   <put name="contentClass" value="contentClass"/>

   <put name="header" value="/bookHeader.jsp"/>
   <put name="menu" value="/bookMenu.jsp"/>
   <put name="content" value="/bookContent.jsp"/>
</definition>
```

Then we use those attributes in the layout:

```
<%-- this is an excerpt of /headerMenuContentLayout.jsp --%>
...
<tiles:importAttribute scope="request"/>

<h:panelGrid columns="2" styleClass="#{gridClass}"
   headerClass="#{headerClass}"
   columnClasses="#{menuClass}, #{contentClass}">
   ...
</h:panelGrid>
```

Tile attributes, such as `headerClass`, `menuClass`, etc., in the preceding code, exist in *tiles scope*, which is inaccessible to JSF. To make our attributes accessible to the layout JSF page listed above, we use the `tiles:importAttribute` tag. That tag imports all tile attributes to the scope you specify with the `scope` attribute. In the preceding code, we imported them to request scope.

Now we can specify different CSS classes for other tiles:

```
<definition name="anotherTile" path="/headerMenuContentLayout.jsp">
   <put name="headerClass" value="aDifferentHeaderClass"/>
    ...
</definition>
```

NOTE: The `tiles:importAttribute` tag also lets you import one attribute at a time—for example: `<tiles:importAttribute name="headerClass" scope="..."/>`.

Extending Tiles

In "Parameterizing Tiles" on page 334 we defined a tile that looked like this:

```
<definition name="book" path="/headerMenuContentLayout.jsp">
   <put name="headerClass"  value="headerClass"/>
   <put name="menuClass" value="menuClass"/>
   <put name="contentClass" value="contentClass"/>

   <put name="header" value="/bookHeader.jsp"/>
   <put name="menu" value="/bookMenu.jsp"/>
   <put name="content" value="/bookContent.jsp"/>
</definition>
```

There are two distinct types of attributes in that tile: CSS classes and included content. Although the latter is specific to the `book` tile, the former can be used by tiles that represent something other than books. Because of that generality, we split the `book` tile into two:

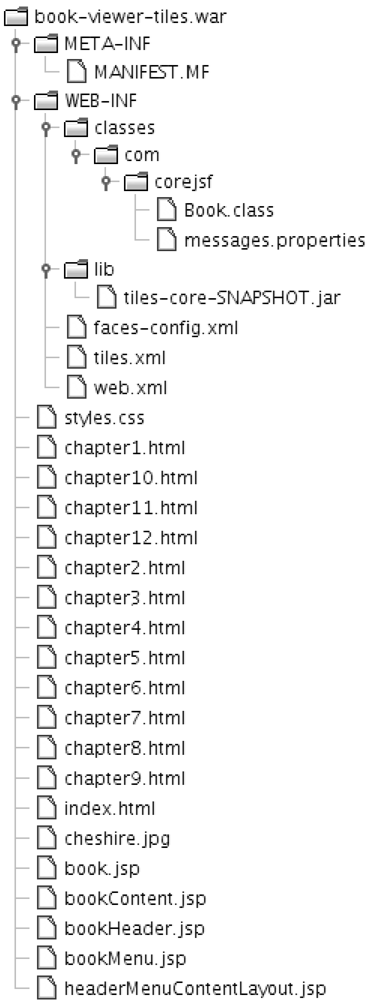
```
<definition name="header-menu-content" path="/headerMenuContentLayout.jsp">
   <put name="headerClass" value="headerClass"/>
   <put name="menuClass" value="menuClass"/>
   <put name="contentClass" value="contentClass"/>
</definition

<definition name="book" extends="header-menu-content">
   <put name="header" value="/bookHeader.jsp"/>
   <put name="menu" value="/bookMenu.jsp"/>
   <put name="content" value="/bookContent.jsp"/>
</definition>
```

Now the `book` tile *extends* the `header-menu-content` tile. When you extend a tile, you inherit its layout and attributes, much the same as Java subclasses inherit methods and variables from their base classes. Because we have split the original tile in two, the CSS class attributes are available for reuse by other tiles that extend the `header-menu-content` tile.

NOTE: Here is one more thing to consider about Tiles. Imagine the book viewer has been a huge success and Project Gutenberg has commissioned you to implement a library that can display all 6,000+ of their books. You define more than 6,000 tiles that reuse the same layout—one tile for each book—and present your finished product to the folks at Gutenberg. They think it's great, but they want you to add a footer to the bottom of every page. Since you have used Tiles, you only need to change the single layout used by all your tiles. Imagine the difficulty you would encounter making that change if you had replicated the layout code more than 6,000 times!

Figure 8–6 shows the directory structure for the "tileized" version of the book viewer. That directory structure is the same as the previous version of the book viewer, except that we have added a layout—`headerMenuContentLayout.jsp`—and the tiles definition file, `/WEB-INF/tiles.xml`.

Listing 8–10 through Listing 8–12 show the Tiles definition file, the book layout, and the JSF page that displays *Alice in Wonderland*. We left out the listings of the other files in the application because they are unchanged from the application discussed in "Content Inclusion in JSP-Based Applications" on page 326.



```
book-viewer-tiles.war
  META-INF
    MANIFEST.MF
  WEB-INF
    classes
      com
        corejsf
          Book.class
          messages.properties
    lib
      tiles-core-SNAPSHOT.jar
    faces-config.xml
    tiles.xml
    web.xml
  styles.css
  chapter1.html
  chapter10.html
  chapter11.html
  chapter12.html
  chapter2.html
  chapter3.html
  chapter4.html
  chapter5.html
  chapter6.html
  chapter7.html
  chapter8.html
  chapter9.html
  index.html
  cheshire.jpg
  book.jsp
  bookContent.jsp
  bookHeader.jsp
  bookMenu.jsp
  headerMenuContentLayout.jsp
```

Figure 8–6 Book viewer with extended tile directory structure

Listing 8–10 book-viewer-tiles/web/WEB-INF/tiles.xml

```
<!DOCTYPE tiles-definitions PUBLIC
   "-//Apache Software Foundation//DTD Tiles Configuration//EN"
   "http://jakarta.apache.org/struts/dtds/tiles-config.dtd">

<tiles-definitions>
   <definition name="book" path="/headerMenuContentLayout.jsp">
       <put name="gridClass"           value="headerMenuContent"/>
       <put name="headerClass"         value="header"/>
       <put name="menuColumnClass"     value="menuColumn"/>
       <put name="contentColumnClass"  value="contentColumn"/>

      <put name="header"  value="/bookHeader.jsp"/>
      <put name="menu"    value="/bookMenu.jsp"/>
      <put name="content" value="/bookContent.jsp"/>
   </definition>
</tiles-definitions>
```

Listing 8–11 book-viewer-tiles/web/headerMenuContentLayout.jsp

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

<tiles:importAttribute scope="request"/>

<h:panelGrid columns="2" styleClass="#{gridClass}"
      headerClass="#{headerClass}"
      columnClasses="#{menuColumnClass}, #{contentColumnClass}">
   <f:facet name="header">
      <f:subview id="header">
         <tiles:insert attribute="header" flush="false"/>
      </f:subview>
   </f:facet>

   <f:subview id="menu">
      <tiles:insert attribute="menu" flush="false"/>
   </f:subview>

   <f:subview id="content">
      <tiles:insert attribute="content" flush="false"/>
   </f:subview>
</h:panelGrid>
```

Listing 8–12 book-viewer-tiles/web/book.jsp

```
<html>
   <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
   <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
   <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
   <%@ taglib uri="http://jakarta.apache.org/tiles" prefix="tiles" %>

   <f:view>
      <f:loadBundle basename="com.corejsf.messages" var="msgs"/>
      <head>
         <link href="styles.css" rel="stylesheet" type="text/css"/>
         <title><h:outputText value="#{msgs.bookWindowTitle}"/></title>
      </head>

      <body>
         <f:subview id="book">
            <h:form>
               <tiles:insert definition="book" flush="false"/>
            </h:form>
         </f:subview>
      </body>
   </f:view>
</html>
```

The Library

In this section, we turn the book viewer into a library, as shown in Figure 8–7.

The library application shown in Figure 8–7 contains a menu at the top of the page that lets you select a book, either *Alice in Wonderland* or *Peter Pan*. The rest of the application works like the book viewer we have discussed throughout this chapter.

The library employs two Tiles techniques that are of interest to us: nesting tiles and using tile controllers.

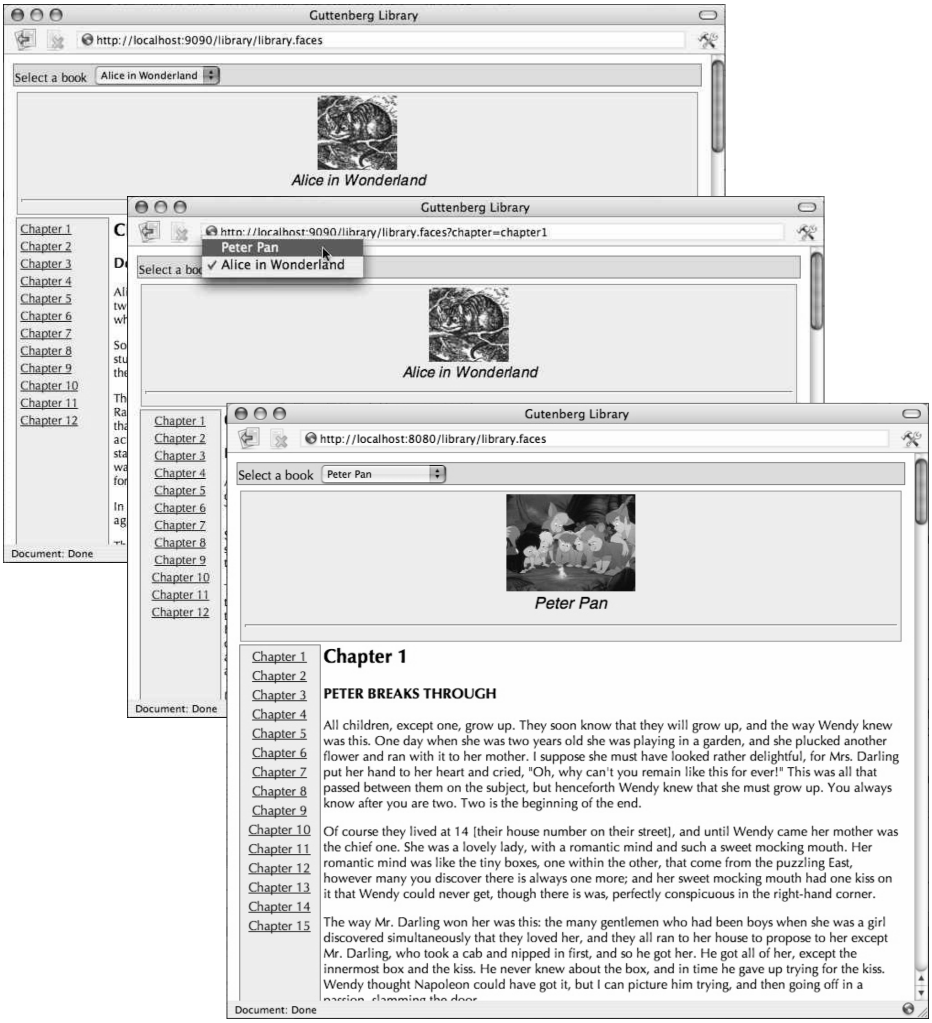
Nested Tiles

The library shown in Figure 8–7 contains a book viewer. So does the library tile:

```
<definition name="book">
   ...
</definition>

<definition name="library" path="/libraryLayout.jsp"
      controllerClass="com.corejsf.LibraryTileController">
```

```
    <put name="header" value="/bookSelector.jsp"/>
    <put name="book" value="book"/>
</definition>
```

Notice the value for the book attribute—it is a tile, not a JSP page. Using a tile name instead of a JSP page lets you nest tiles, as we did by nesting the book tile in the library.



Chapter 1

PETER BREAKS THROUGH

All children, except one, grow up. They soon know that they will grow up, and the way Wendy knew was this. One day when she was two years old she was playing in a garden, and she plucked another flower and ran with it to her mother. I suppose she must have looked rather delightful, for Mrs. Darling put her hand to her heart and cried, "Oh, why can't you remain like this for ever!" This was all that passed between them on the subject, but henceforth Wendy knew that she must grow up. You always know after you are two. Two is the beginning of the end.

Of course they lived at 14 [their house number on their street], and until Wendy came her mother was the chief one. She was a lovely lady, with a romantic mind and such a sweet mocking mouth. Her romantic mind was like the tiny boxes, one within the other, that come from the puzzling East, however many you discover there is always one more; and her sweet mocking mouth had one kiss on it that Wendy could never get, though there is was, perfectly conspicuous in the right-hand corner.

The way Mr. Darling won her was this: the many gentlemen who had been boys when she was a girl discovered simultaneously that they loved her, and they all ran to her house to propose to her except Mr. Darling, who took a cab and nipped in first, and so he got her. He got all of her, except the innermost box and the kiss. He never knew about the box, and in time he gave up trying for the kiss. Wendy thought Napoleon could have got it, but I can picture him trying, and then going off in a passion, slamming the door

Figure 8–7 Library implemented with JSF and tiles

Tile Controllers

In our book viewer application, we had one managed bean named `book` (see "The Book Viewer" on page 318 for more information about the `book` bean). The library, on the other hand, must be aware of more than one book.

In this section—with a sleight of hand—we show you how to support multiple books without having to change the book viewer. The book viewer will continue to manipulate a `book` bean, but that bean will no longer be a managed bean. Instead, it will be the book that was last selected in the library's pull-down menu at the top of the page.

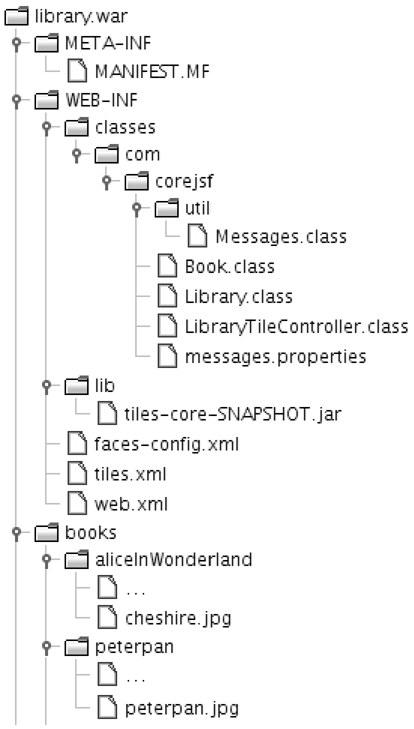
We accomplish that sleight of hand with a Tiles controller. Tiles lets you attach a Java object, called a *tile controller*, to a tile. That object's class must implement the `org.apache.struts.tiles.Controller` interface, which defines a single `perform` method. Tiles invokes that method just before it loads the controller's associated tile. Tile controllers have access to their tile's context, which lets the controller access the tile's attributes or create new attributes.

We attach a controller to the `library` tile. The controller looks for a `library` attribute in session scope. If the library is not there, the controller creates a library and stores it in session scope. The controller then consults the library's `selectedBook` property to see if a book has been selected. If so, the controller sets the value of the `book` session attribute to the selected book. If there is no selected book, the controller sets the `book` attribute to that for *Peter Pan*. Subsequently, when the `library` tile is loaded, the book viewer accesses the selected book. The controller is listed in Listing 8–20 on page 348.

Figure 8–8 shows the directory structure for the library application. For brevity, we left out the book HTML files.

The files shown in Figure 8–8 are shown in Listing 8–13 through Listing 8–28, with the exception of the HTML files. As you look through those listings, note the effort required to add a new book. All you have to do is modify the constructor in `Library.java`—see Listing 8–19 on page 346—to create your book and add it to the book map.

You could even implement the `Library` class so that it reads XML book definitions. That way, you could add books without any programming. Digesting XML is an easy task with Tiles's distant cousin, the Apache Commons Digester. See `http://jakarta.apache.org/commons/digester/` for more information about the Digester.



```
library.war
  META-INF
    MANIFEST.MF
  WEB-INF
    classes
      com
        corejsf
          util
            Messages.class
          Book.class
          Library.class
          LibraryTileController.class
          messages.properties
    lib
      tiles-core-SNAPSHOT.jar
    faces-config.xml
    tiles.xml
    web.xml
  books
    aliceInWonderland
      ...
      cheshire.jpg
    peterpan
      ...
      peterpan.jpg
```

Figure 8–8 Library directory structure

Listing 8–13 library/web/library.jsp

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

   <f:view>
      <f:loadBundle basename="com.corejsf.messages" var="msgs"/>
      <head>
         <link href="styles.css" rel="stylesheet" type="text/css"/>
         <title><h:outputText value="#{msgs.libraryWindowTitle}"/></title>
      </head>

```

Listing 8–13 library/web/library.jsp (cont.)

```
      <body>
         <f:subview id="library">
            <h:form>
               <tiles:insert definition="library" flush="false"/>
            </h:form>
         </f:subview>
      </body>
   </f:view>
</html>
```

Listing 8–14 library/web/WEB-INF/tiles.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

 <!DOCTYPE tiles-definitions PUBLIC
  "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
  "http://struts.apache.org/dtds/tiles-config_1_1.dtd">

<tiles-definitions>
   <definition name="menu-header-content" path="/headerMenuContentLayout.jsp">
      <put name="gridClass"           value="headerMenuContent"/>
      <put name="headerClass"         value="header"/>
      <put name="menuColumnClass"     value="menuColumn"/>
      <put name="contentColumnClass"  value="contentColumn"/>
   </definition>

   <definition name="book" extends="menu-header-content">
      <put name="header"  value="/bookHeader.jsp"/>
      <put name="menu"    value="/bookMenu.jsp"/>
      <put name="content" value="/bookContent.jsp"/>
   </definition>

   <definition name="library" path="/libraryLayout.jsp"
         controllerClass="com.corejsf.LibraryTileController">
      <put name="header" value="/bookSelector.jsp"/>
      <put name="book" value="book"/>
   </definition>
</tiles-definitions>
```

Listing 8–15 library/web/libraryLayout.jsp

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

<h:panelGrid columns="1" styleClass="book" headerClass="libraryHeader">
   <f:facet name="header">
      <f:subview id="header">
         <tiles:insert attribute="header" flush="false"/>
      </f:subview>
   </f:facet>

   <f:subview id="book">
      <tiles:insert attribute="book" flush="false"/>
   </f:subview>
</h:panelGrid>
```

Listing 8–16 library/web/bookSelector.jsp

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

<h:outputText value="#{msgs.selectABookPrompt}"/>

&nbsp;&nbsp;

<h:selectOneMenu onchange="submit()" value="#{library.book}"
      valueChangeListener="#{library.bookSelected}">
   <f:selectItems value="#{library.bookItems}"/>
</h:selectOneMenu>
```

Listing 8–17 library/src/java/com/corejsf/Library.java

```
package com.corejsf;

import java.util.*;
import javax.faces.model.SelectItem;
import javax.faces.event.ValueChangeEvent;

public class Library {
   private Map<String,Book> bookMap = new HashMap<String,Book>();
   private Book initialBook = null;
   private List bookItems = null;
```

Listing 8–17 library/src/java/com/corejsf/Library.java (cont.)

```
   private String book = null;
   private String selectedBook = null;

   public Library() {
      Book peterpan = new Book();
      Book aliceInWonderland = new Book();

      initialBook = peterpan;

      aliceInWonderland.setDirectory("books/aliceInWonderland");
      aliceInWonderland.setTitleKey("aliceInWonderland");
      aliceInWonderland.setImage("books/aliceInWonderland/cheshire.jpg");
      aliceInWonderland.setNumChapters(12);

      peterpan.setDirectory("books/peterpan");
      peterpan.setTitleKey("peterpan");
      peterpan.setImage("books/peterpan/peterpan.jpg");
      peterpan.setNumChapters(15);

      bookMap.put("aliceInWonderland", aliceInWonderland);
      bookMap.put("peterpan", peterpan);
   }
   public void setBook(String book) { this.book = book; }
   public String getBook() { return book; }

   public Map<String,Book> getBooks() {
      return bookMap;
   }
   public void bookSelected(ValueChangeEvent e) {
      selectedBook = (String) e.getNewValue();
   }
   public Book getSelectedBook() {
      return selectedBook != null ? bookMap.get(selectedBook) : initialBook;
   }
   public List getBookItems() {
      if(bookItems == null) {
         bookItems = new LinkedList();
         Iterator<Book> it = bookMap.values().iterator();
         while(it.hasNext()) {
            Book book = it.next();
            bookItems.add(new SelectItem(book.getTitleKey(),
                                             getBookTitle(book.getTitleKey())));
         }
      }
      return bookItems;
```

Listing 8–17 library/src/java/com/corejsf/Library.java (cont.)

```java
   }
   private String getBookTitle(String key) {
      return com.corejsf.util.Messages.
                       getString("com.corejsf.messages", key, null);
   }
}
```

Listing 8–18 library/web/bookSelector.jsp

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

<h:outputText value="#{msgs.selectABookPrompt}"/>

&nbsp;&nbsp;

<h:selectOneMenu onchange="submit()" value="#{library.book}"
      valueChangeListener="#{library.bookSelected}">
   <f:selectItems value="#{library.bookItems}"/>
</h:selectOneMenu>
```

Listing 8–19 library/src/java/com/corejsf/Library.java

```java
package com.corejsf;

import java.util.*;
import javax.faces.model.SelectItem;
import javax.faces.event.ValueChangeEvent;

public class Library {
   private Map<String,Book> bookMap = new HashMap<String,Book>();
   private Book initialBook = null;
   private List bookItems = null;
   private String book = null;
   private String selectedBook = null;

   public Library() {
      Book peterpan = new Book();
      Book aliceInWonderland = new Book();

      initialBook = peterpan;

```

Listing 8–19 library/src/java/com/corejsf/Library.java (cont.)

```
      aliceInWonderland.setDirectory("books/aliceInWonderland");
      aliceInWonderland.setTitleKey("aliceInWonderland");
      aliceInWonderland.setImage("books/aliceInWonderland/cheshire.jpg");
      aliceInWonderland.setNumChapters(12);

      peterpan.setDirectory("books/peterpan");
      peterpan.setTitleKey("peterpan");
      peterpan.setImage("books/peterpan/peterpan.jpg");
      peterpan.setNumChapters(15);

      bookMap.put("aliceInWonderland", aliceInWonderland);
      bookMap.put("peterpan", peterpan);
   }
   public void setBook(String book) { this.book = book; }
   public String getBook() { return book; }

   public Map<String,Book> getBooks() {
      return bookMap;
   }
   public void bookSelected(ValueChangeEvent e) {
      selectedBook = (String) e.getNewValue();
   }
   public Book getSelectedBook() {
      return selectedBook != null ? bookMap.get(selectedBook) : initialBook;
   }
   public List getBookItems() {
      if(bookItems == null) {
         bookItems = new LinkedList();
         Iterator<Book> it = bookMap.values().iterator();
         while(it.hasNext()) {
            Book book = it.next();
            bookItems.add(new SelectItem(book.getTitleKey(),
                                 getBookTitle(book.getTitleKey())));
         }
      }
      return bookItems;
   }
   private String getBookTitle(String key) {
      return com.corejsf.util.Messages.
                  getString("com.corejsf.messages", key, null);
   }
}
```

Listing 8–20 library/src/java/com/corejsf/LibraryTileController.java

```java
package com.corejsf;

import java.io.IOException;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.tiles.ComponentContext;
import org.apache.tiles.Controller;

public class LibraryTileController implements Controller {
   public void execute(ComponentContext tilesContext,
                  HttpServletRequest request,
                  HttpServletResponse response,
                  ServletContext context)
                  throws IOException, ServletException {
      HttpSession session = request.getSession();

      String chapter = (String) request.getParameter("chapter");
      session.setAttribute("chapter", chapter == null || "".equals(chapter) ?
                           "chapter1" : chapter);

      Library library = (Library) session.getAttribute("library");

      if(library == null) {
         library = new Library();
         session.setAttribute("library", library);
      }

      Book selectedBook = library.getSelectedBook();
      if(selectedBook != null) {
         session.setAttribute("book", selectedBook);
      }
   }
   public void perform(ComponentContext tilesContext,
                  HttpServletRequest request,
                  HttpServletResponse response,
                  ServletContext context)
                  throws IOException, ServletException {
      HttpSession session = request.getSession();
      execute(tilesContext, request, response, context);
   }
}
```

Listing 8–21 library/src/java/com/corejsf/Book.java

```java
package com.corejsf;

import java.util.LinkedList;
import java.util.List;

public class Book {
   private String titleKey;
   private String image;
   private String directory;
   private int numChapters;
   private List<String> chapterKeys = null;

   // PROPERTY: titleKey
   public void setTitleKey(String titleKey) { this.titleKey = titleKey; }
   public String getTitleKey() { return titleKey; }

   // PROPERTY: image
   public void setImage(String image) { this.image = image; }
   public String getImage() { return image; }

   // PROPERTY: numChapters
   public void setNumChapters(int numChapters) { this.numChapters = numChapters;}
   public int getNumChapters() { return numChapters; }

   // PROPERTY: directory
   public void setDirectory(String directory) { this.directory = directory;}
   public String getDirectory() { return directory; }

   // PROPERTY: chapterKeys
   public List<String> getChapterKeys() {
      if(chapterKeys == null) {
         chapterKeys = new LinkedList<String>();
         for(int i=1; i <= numChapters; ++i)
            chapterKeys.add("chapter" + i);
      }
      return chapterKeys;
   }
}
```

Listing 8–22 library/web/bookHeader.jsp

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

<h:panelGrid columns="1" styleClass="bookHeader">
   <h:graphicImage value="#{book.image}"/>
   <h:outputText value="#{msgs[book.titleKey]}" styleClass="bookTitle"/>
   <hr>
</h:panelGrid>
```

Listing 8–23 library/web/bookMenu.jsp

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

<h:dataTable value="#{book.chapterKeys}" var="chapterKey"
        styleClass="links" columnClasses="linksColumn">
   <h:column>
      <h:commandLink>
         <h:outputText value="#{msgs[chapterKey]}"/>
         <f:param name="chapter" value="#{chapterKey}"/>
      </h:commandLink>
   </h:column>
</h:dataTable>
```

Listing 8–24 library/web/bookContent.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:import url="${book.directory}/${chapter}.html"/>
```

Listing 8–25 library/web/styles.css

```
.library {
   vertical-align: top;
   width: 100%;
   height: 100%;
}
.libraryHeader {
   width: 100%;
   text-align: left;
```

Listing 8–25 library/web/styles.css (cont.)

```
   vertical-align: top;
   background-color: #ddd;
   font-weight: lighter;
   border: thin solid #777;
}
.bookHeader {
   width: 100%;
   text-align: center;
   background-color: #eee;
   border: thin solid CornflowerBlue;
}
.bookTitle {
   text-align: center;
   font-style: italic;
   font-size: 1.3em;
   font-family: Helvetica;
}
.menuColumn {
   vertical-align: top;
   background-color: #eee;
   border: thin solid #777;
}
.chapterColumn {
   vertical-align: top;
   text-align: left;
   width: *;
   padding: 3px;
}
.contentColumn {
   vertical-align: top;
   text-align: left;
   width: *;
}
.links {
   width: 85px;
   vertical-align: top;
   text-align: center;
}
.linksColumn {
   vertical-align: top;
   text-align: center;
}
```

Listing 8–26 library/web/WEB-INF/faces-config.xml

```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">
</faces-config>
```

Listing 8–27 library/web/WEB-INF/web.xml

```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>Tiles Servlet</servlet-name>
      <servlet-class>org.apache.tiles.servlets.TilesServlet</servlet-class>
      <init-param>
         <param-name>definitions-config</param-name>
         <param-value>/WEB-INF/tiles.xml</param-value>
      </init-param>
      <load-on-startup>2</load-on-startup>
   </servlet>

   <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>
</web-app>
```

Listing 8–28 library/src/java/com/corejsf/messages.properties

```
libraryWindowTitle=Gutenberg Library
aliceInWonderland=Alice in Wonderland
peterpan=Peter Pan
selectABookPrompt=Select a book

chapter1=Chapter 1
chapter2=Chapter 2
chapter3=Chapter 3
chapter4=Chapter 4
chapter5=Chapter 5
chapter6=Chapter 6
chapter7=Chapter 7
chapter8=Chapter 8
chapter9=Chapter 9
chapter10=Chapter 10
chapter11=Chapter 11
chapter12=Chapter 12
chapter13=Chapter 13
chapter14=Chapter 14
chapter15=Chapter 15
```