Master the Tiles framework

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. Getting started 2
2. The Tiles framework and architecture 5
3. Tile layouts 8
4. Tile definitions 18
5. Advanced Tile topics 23
6. Lists 28
7. Advanced definition concepts 34
8. Wrap-up and resources 42

Section 1. Getting started

What is this tutorial about?

This tutorial describes how to use the Tiles framework to create reusable presentation components. (At its creation, the Tiles framework was originally called Components. The name was changed because "components" means too many different things, but the spirit of the original name remains.) Beyond site layouts, however, you can do much more with tiles. You can, for example, compartmentalize your presentation tier to better reuse layouts, HTML, and other visual components.

This tutorial strives to impart the basics of using the Tiles framework, then takes your knowledge up a notch. By the time you finish, you will be able to use the more advanced Tiles features to create reusable components.

Note: Throughout this tutorial, the terms *tile* and *page* are used interchangeably because any Web resource can be a tile. A tile layout represents a special type of tile you can use to place other tiles within it. A tile layout can be used as a tile in another tile layout.

Getting down to specifics, this tutorial:

- Defines the Tiles framework and architecture.
- Covers the Tiles architecture and how it integrates with Struts.
- Clarifies key Tiles concepts.
- Demonstrates how to build and use a tile layout as a site template.
- Demonstrates how to use tile definitions both in XML and JavaServer Pages (JSP) code.
- Defines tile scope and how to move objects in and out of tile scope.
- Works with attribute lists.
- Shows how to nest tiles.
- Demonstrates how to build and use a tile layout as a small visual component.

- Shows how to subclass a definition.
- Creates a controller for a tile.
- Demonstrates using a tile as an `ActionForward`.

Who should take this tutorial?

If you find yourself writing the same three lines of JSP code on every page, or you want to define complex template layouts easily, then you will benefit from this tutorial.

This tutorial assumes you have a solid understanding of Java programming, MVC (Model-View-Controller), Model 2, and JSP technology. While a good Struts background lets you get the most out of this tutorial, if you're adept at JSP programming, you should be able to follow most of what is going on.

Software requirements and code installation

To complete this tutorial, you will need:

- A 1.1, 1.2, or 2.0 JSP-compliant servlet/JSP container. *Apache Tomcat 3.x or higher* is an excellent choice. Note: The tutorial's examples were written with a JSP 1.2-compliant container.
- The Tiles framework. You can get the framework as part of the *Struts 1.1* download or standalone from the *Tiles Web site*.
- The source code. I've supplied two versions: one with jar files and one without jar files for those of us who are bandwidth impaired. Struts ships with a blank war file, `struts-blank.war` (under the `webapps` directory), which illustrates which configuration files and jar files you need, and where you typically put them. You'll use the same structure for the example code.

See Resources on page 43 for information on these materials and additional resources.

About the author

Rick Hightower enjoys working with Java technology, Ant, Struts, the IBM Emerging Technologies Toolkit (ETTK), and XDoclet. Rick currently serves as the CTO of *ArcMind Inc.*, a mentoring, consulting, and training company focusing on enterprise development. Rick, a regular IBM *developerWorks* contributor, has written more than 10 tutorials ranging from EJB (Enterprise JavaBeans) technology to Web services to XDoclet.

While working at eBlox, Rick and the eBlox team used Struts to build two frameworks and an ASP (application service provider) for online e-commerce stores. They started using Struts long before the 1.0 release. Rick recently helped put together a well-received course for Trivera Technologies on Struts that runs on Tomcat, Resin EE, WebSphere Studio Application Developer, and others.

Rick co-authored *Mastering Struts, 2nd edition* with James Goodwill (Wrox Press). Rick also co-wrote *Java Tools for Extreme Programming* (John Wiley & Sons, 2001), the best-selling software development book on Amazon.com for three months in 2002. It covers applying Ant, JUnit, Cactus, and more to J2EE (Java 2 Platform, Enterprise Edition) development. Rick also contributed two chapters to *Mastering Tomcat Development* (John Wiley & Sons, 2002), as well as many other publications.

Rick spoke at the 2003 JavaOne Developers Conference on EJB CMP/CMR and XDoclet, and at TheServerSide.com Software Symposium on J2EE development with XDoclet. Additionally, Rick has spoken at JDJEdge and WebServicesEdge. Moreover, Rick spoke about advanced Struts topics at the Complete Programmer Network software symposiums (in six different cities across the US).

When not traveling around the country teaching the Trivera Struts course, speaking at conferences about Struts, or doing Struts consulting, Rick enjoys drinking coffee at an all-night coffee shop, writing about Struts and other topics, and writing about himself in the third person.

Section 2. The Tiles framework and architecture

Tiles framework

The Tiles framework turns the `jsp:includes` concept inside out -- thus letting you more feasibly create reusable pages. With the Tiles framework, developers can build pages by assembling reusable tiles. You should think of tiles as visual components.

A tile layout is a special JSP page that allows tiles to be placed. Tile layouts dictate where the tiles will be laid out on the page. In many respects the tile layout resembles a template layout. In fact, if you have used Struts templates before, then you will note that the Tile frameworks is backwards compatible with the template custom tag library.

Clarification of terms

The terms presented in this tutorial may seem overwhelming at first, so before we discuss the Tiles framework in more detail, let's define some important terms.

Glossary of terms

Tiles The template framework for Struts to create reusable presentation components.

Page A Web resource included by a tile layout.

Tile The same as a page.

Region An area in a tile layout that inserts another tile. Regions have logical names like header, footer, and so on.

Tile layout A JSP page that describes where other pages should be positioned. Acting as a template, a tile layout defines regions where other tiles are inserted. A tile layout can be a tile to another tile layout.

Definition Defines parameters for calling a tile layout.

Tile layouts

In some ways, the tile layout works like a display function. To use a tile layout, call it with the `tiles:insert` tag. When you invoke the tile layout, you pass parameters.

The parameters become attributes to the tile layout; for example, the parameters are put into tile scope.

The parameters passed when invoking a tile can be other JSP pages or Web resources, which you can insert at predefined locations, called *regions*, in the layout. The parameters also consist of strings you can insert into the tile layout. In fact, you can pass many types of objects as parameters to the tile. The parameters become attributes in the tile scope available only to that tile.

The *tile scope* resembles a page scope in that the tile scope is less general than a request scope. The tile scope lets tile users pass arguments, called attributes, to the tile. The tile scope lets you pass variables available only to that tile layout or tile. Special custom tags let you copy attributes from tile scope to page, request, session, or application scope, or display the attribute as an included Web resource.

Default arguments

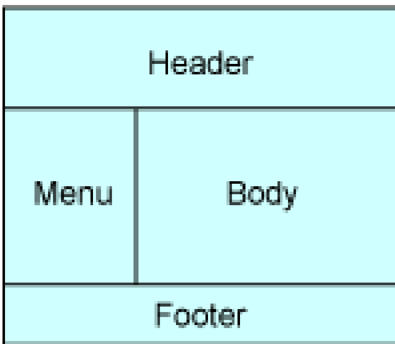
Some programming languages, like C++, Visual Basic, and Python, let you pass default arguments to functions and methods. To further extend the display function metaphor, the Tiles framework also lets you pass default arguments to a tile layout. To do so you must define a *tile definition.* Tile definitions let you define default parameters for tiles. Definitions can be defined in JSP code or XML.

A definition can extend other definitions in a way similar to how a class can extend another class. By using definitions and tile layouts, you can create reusable display components.

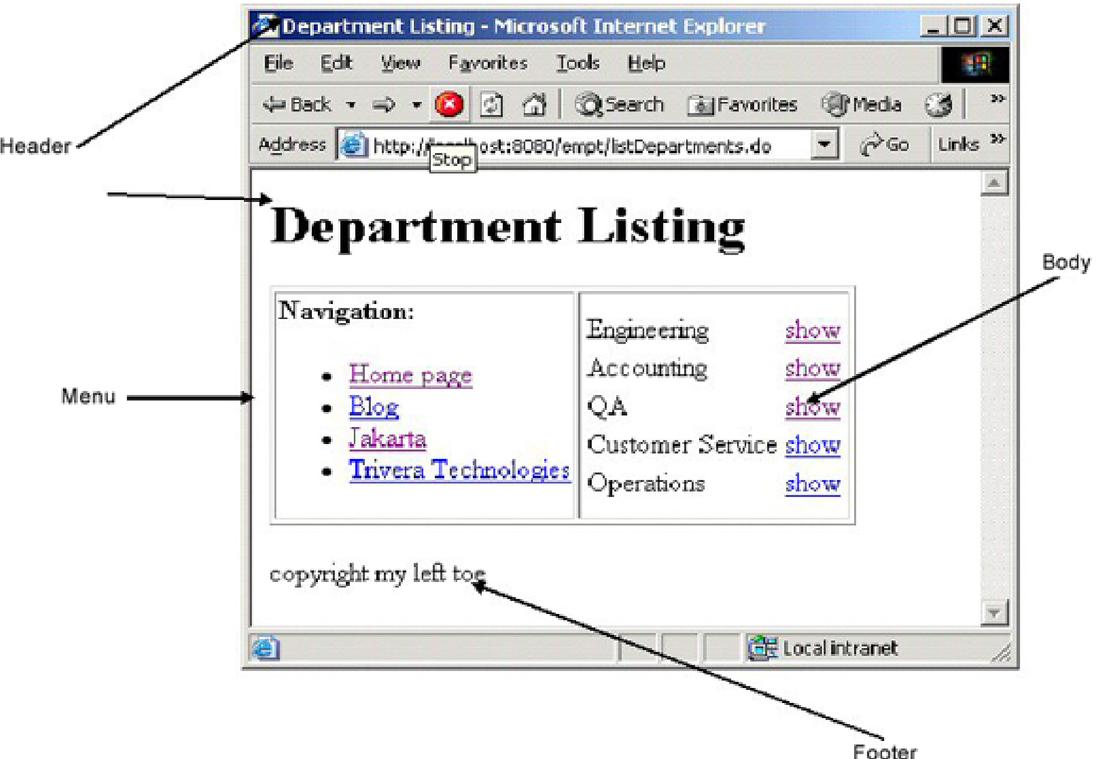
You can use Tiles with or without Struts. To use Tiles with Struts, you will use the Tiles taglibs that ship with Struts. In addition, the Tiles framework includes its own `RequestProcessor` to handle tile layouts as `ActionForward`s -- letting you forward to a tile definition instead of a JSP page. Tiles does that by overriding the `processActionForward` in its custom `RequestProcessor`.

A typical tile layout

A typical tile layout may define rectangular regions for the header, footer, menu, and body, as depicted in Figure 1.



The regions shown in Figure 1 may map to a Web site like that seen in Figure 2.



Notice that you can easily redefine reusable pieces of this application just by passing the correct parameters. For example, the employee listing might use the same header and footer but a different menu and body, yet still can use all of the general layout regions defined by the tile layout. That lets you reuse the same tile layout with different contents. Instead of including the HTML markup, you include the content in the markup.

Section 3. Tile layouts

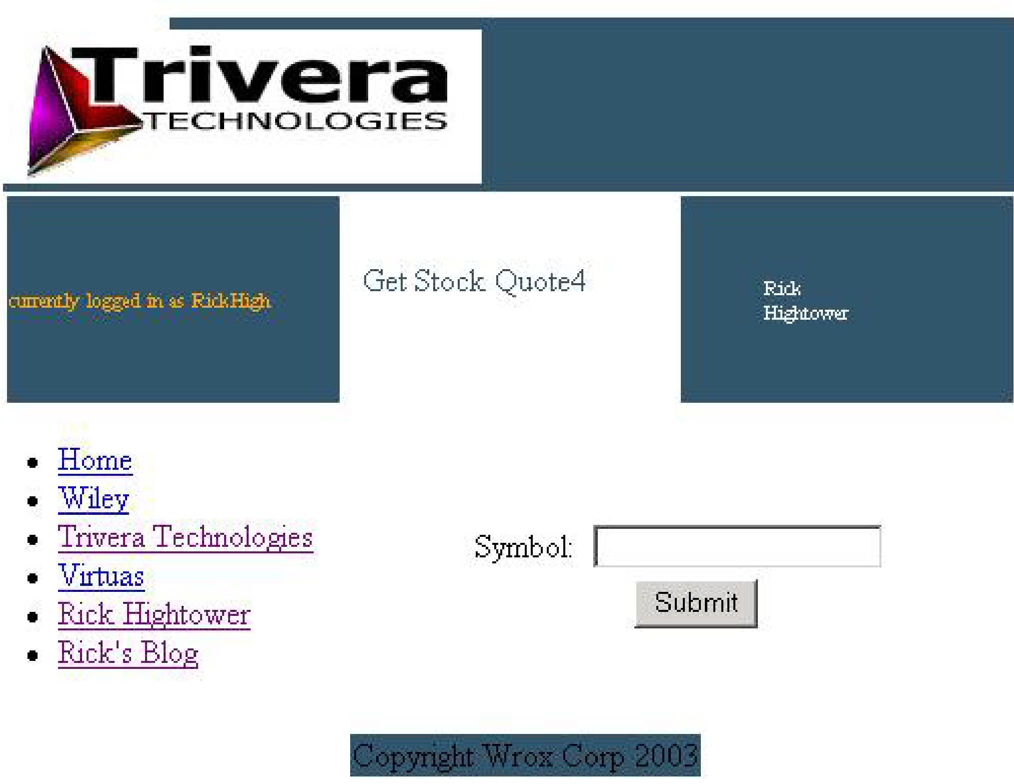
Build your first tile layout

Wouldn't it be nice for your site to reuse the same layout (implemented with an HTML table) and images without duplicating the same HTML code?

Tiles shines in creating a common look and feel for your site. That said, many developers don't realize that Tiles also shines at creating reusable components implemented in JSP.

If you find yourself repeating the same HTML code in multiple pages, then consider those pages candidates for a tile layout. Similarly, if you employ the same HTML or JSP tags in various places on various pages, then you're well positioned to use tiles to create a small visual component.

As a Tiles framework example, on the following pages you will refactor a simple stock quote application to utilize a tile layout, as shown in Figure 3.



Sample application

The simple sample application features a stock quote page with a form that takes a single parameter, a stock quote symbol (index.jsp). Another page displays the value of the stock quote (quote.jsp).

Study the following two code listings. You will refactor them to use all manner of tile layouts.

index.jsp

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

<html>
  <head>
    <title>Stock Quote</title>
  </head>

  <body>
    <table width="500" border="0" cellspacing="0" cellpadding="0">
      <tr>
        <td> </td>
      </tr>
      <tr bgcolor="#36566E">
        <td height="68" width="48%">
          <div align="left">
            <img src="images/hp_logo_rickhightower.gif"
              width="220"
             height="74">
          </div>
        </td>
      </tr>
      <tr>
        <td> </td>
      </tr>
    </table>

    <html:form action="Lookup">
      <table width="45%" border="0">
        <tr>
          <td><bean:message key="app.symbol" />:</td>
          <td><html:text property="symbol" /></td>
        </tr>
        <tr>
          <td colspan="2" align="center"><html:submit /></td>
        </tr>
      </table>
    </html:form>

  </body>
</html>
```

quote.jsp

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

<html>
  <head>
    <title>Stock Quote</title>
  </head>
  <body>

    <table width="500" border="0" cellspacing="0" cellpadding="0">
      <tr>
        <td> </td>
      </tr>
      <tr bgcolor="#36566E">
        <td height="68" width="48%">
          <div align="left">
            <img src="images/hp_logo_rickhightower.gif" width="220" height="74">
          </div>
        </td>
      </tr>
      <tr>
        <td> </td>
      </tr>
      <tr>
        <td> </td>
      </tr>
      <tr>
        <td> </td>
      </tr>
      <tr>
        <td>
          <bean:message key="app.price" />: <%= request.getAttribute("PRICE") %>
        </td>
      </tr>
      <tr>
        <td> </td>
      </tr>
    </table>
  </body>
</html>
```

To learn how to use the Tiles framework, first you must write a tile layout, then you'll refactor the two example pages so they don't repeat so much HTML code.

Step-by-step tile layout

To create a tile layout you need to do the following:

1. Find the similarities between the two pages.

2. Create a new layout page.

3. Create two new content pages that contain just the differences between EmployeeListing.jsp and DeptListing.jsp.

4. Insert the tile layout in the page -- that is, have EmployeeListing.jsp and DeptListing.jsp insert the tile layout into their page, passing the content as a parameter and any other necessary parameters (like Title).

Because finding similarities between the two pages requires skill in HTML layout and Web site usability, it proves to be more of an art than a science. For some reason it helps if you have purple hair and body piercing. If you don't think so, just ask my good friend Boo.

This tutorial focuses on Struts, not those necessary skills in HTML layout and Web site usability. For that matter, you will not learn about body piercing or purple hair dye. In fact, the example's HTML layouts are deliberately simple so as not to detract from the Tiles framework.

Create a tile layout

Once you find the similarities between pages (the hard part), you can create the new layout page (the easy part). To create a tile layout, you must do the following:

- Import the tiles taglib into the JSP and any other taglibs you need with the taglib directive.

- Use string parameters to display things like the page title using the `tiles:getAsString` tag.

- Insert the tiles in the correct regions of the layout using the `tiles:insert` tag.

- Pass any needed parameters to the internal tiles using `tiles:put` -- a subtag of `tiles:insert`.

Import the tiles taglib into the JSP and any other taglibs you need as follows (siteLayout.jsp):

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

Note: To use the tiles taglib, don't forget to include the tag library in your `web.xml` file:

```
  <taglib>
    <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-tiles.tld
    </taglib-location>
  </taglib>
```

Next, use string parameters to display things like the page title. You not only want to change the page's content, you also want to change the title that appears in the browser. To do so, pass in a title that the tile layout will use:

```
<html>
  <head>
    <title>
        <tiles:getAsString name="title" ignore="true"/>
    </title>
  </head>
```

Notice you use the `tiles:getAsString` tag to display string parameters. Not only can you pass string parameters, you can also pass other pages to be inserted into this page. That assumes the calling JSP page passes a title to this tile layout; otherwise, the title will be blank.

Note: The `ignore` attribute:

The `ignore` attribute, if true, means ignore the attribute if missing. Otherwise, if `ignore` is false, the tiles framework will throw an exception and the page will not display if the parameter does not pass (false is the default).

To insert the content JSP, use the `tiles:insert` tag, which inserts any page or Web resource that the framework refers to as a tile. The tag effectively defines a region in the tile layout. Remember, the objective of the tile layout is to lay out the tiles into the layout. Here is an example of inserting a tile into the layout:

```
<tiles:insert attribute="content"/>
```

The example above is really simple. What if you want to insert a tile and pass it items in the current page scope? For example, it's possible to pass the title parameter (in tile scope) to header.jsp with the Tiles framework.

Call other tiles (passing attributes)

Any time you insert a tile, you can optionally pass it parameters. The parameters you pass the tile will be put into the tile's tile scope (referred to as tile attributes). For example, in addition to having the title display in the browser's title bar, you'd like the tile to appear in the page's header region.

The header.jsp file accomplishes that. Even though the title variable is in the tile layout page scope, it will not be in the scope of the tiles the tile layout inserts. Each tile and tile layout gets its own context -- that is, its own tile scope. Thus, you must pass the tile variable to the header tile as follows:

```
<tiles:insert attribute="header" ignore="true">
      <tiles:put name="title"
                 beanName="title" beanScope="tile"/>
</tiles:insert>
```

The `tiles:put` tag puts the title parameter in this tile layout scope into the header tile's scope. Then the header tile can use the parameter just as the tile layout did using the `tiles:getAsString` tag. The parameter name is the name of the attribute in the header's tile scope. The bean parameter is the name of the bean in the current scope (siteLayout.jsp). The beanScope is the scope where you look for this attribute (possible values are page, tile, request, session, and application). You can pass beans from any scope to the tile.

Complete listing of tile layout

Next, you'll see the complete new layout page (siteLayout.jsp) that quote.jsp and index.jsp will use:

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

<html>

  <head>
    <title>
        <tiles:getAsString name="title" ignore="true"/>
    </title>
  </head>

  <body>
```

```
    <table width="500" border="0" cellspacing="0" cellpadding="0">

      <tr bgcolor="#36566E">
        <td height="68" width="48%">
          <div align="left">
            <img src="images/hp_logo_rickhightower.gif"
                 width="220" height="74">
          </div>
        </td>
      </tr>

      <tr>
        <td height="68" width="2000">
            <tiles:insert attribute="header" ignore="true">
               <tiles:put name="title"
                          beanName="title" beanScope="tile"/>
            </tiles:insert>
         </td>
      </tr>
      <tr>
        <td>
            <div align="center">
            <tiles:insert attribute="content"/>
            </div>
        </td>
      </tr>
      <tr>
        <td>
            <tiles:insert attribute="footer" ignore="true"/>
        </td>
      </tr>

    </table>

  </body>
</html>
```

Please take a few moments to study the above listing. Notice how you insert tiles into various regions (header, footer, content), and how you employ an HTML layout to define the regions for tiles, thus defining the complete layout for your application.

Use a tile layout

Now that you've defined a tile layout that uses tiles, you need to use the layout. Both index.jsp and quote.jsp will use the same layout. While that seems like a lot of work for just two pages, for a real Web application you might use the same layout for 20 pages or more. This way, you won't have to repeat the HTML or include several JSP fragments in 20 locations.

Note: Why not just use `jsp:include`?

Including JSP fragments in the right location proves a fragile way to reuse HTML. Imagine 20 pages including the same five JSP fragments -- you'd have to repeat yourself 100 times.

To use a tile, do the following steps:

1. Import the tiles taglib with the `taglib` directive.
2. Use `tiles:insert` to insert the tile layout into the current page.
3. Use `tiles:put` to pass string parameters.
4. Use `tiles:put` to pass in parameter tiles.

By using a tile layout, you can externalize the entire HTML needed for the site's layout in one location, and then just insert it in each page. Take a look at the following example, which shows how to insert the tile layout into index.jsp:

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

<tiles:insert page="/siteLayout.jsp" flush="true">

    <tiles:put name="title" type="string"
                     value="Get Rick Hightower Stock Quote" />
    <tiles:put name="header" value="/header.jsp" />
    <tiles:put name="footer" value="/footer.jsp" />
    <tiles:put name="content" value="/indexContent.jsp"/>
</tiles:insert>
```

Now, when you want to do the same thing in quote.jsp, you merely change the content and the header.

You need to call the tile layout (the display function) with the insert tag. (Notice the `tiles:insert` tag to insert the tile layout into the current page):

```
<tiles:insert page="/siteLayout.jsp" flush="true">
```

The page attribute specifies the tile layout you defined in the last section. If the `flush` attribute is set to true, this tile (and page up to this point) will be written to the browser before the rest of the page (or the buffer fills and forces a flush).

To change the page title between quote.jsp and the header.jsp, use the subtag `tiles:put`:

```
     <tiles:put name="title" type="string"
                      value="Get Stock Quote" />
```

Notice how `tiles:put` passes string parameters to the tile layout. The `tiles:put`'s `name` attribute tag specifies the parameter name. The `tiles:put`'s `type` attribute specifies the parameter's type. Lastly, the `value` parameter is passed the `title` attribute's value. That lets you pass simple strings as parameters when the tile layout (display function) gets called with the `tiles:insert` tag. The parameters become tile layout attributes; that is, they get inserted into the tile scope of the tile layout.

Notice how you pass in three tiles as header, footer, and content parameters (header.jsp, footer.jsp, and indexContent.jsp):

```
     <tiles:put name="header" value="/header.jsp" />
     <tiles:put name="footer" value="/footer.jsp" />
     <tiles:put name="content" value="/indexContent.jsp"/>
```

The header.jsp page will be inserted in the tile layout's header region. The footer.jsp page will be inserted into the tile layout's footer region. The indexContent.jsp page will be inserted into the tile layout's content region. If you want to insert different content and title, simply change the content parameter's value.

Notice the form for index.jsp no longer resides in index.jsp. The form now resides in indexContent.jsp, listed as follows:

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

 <html:form action="Lookup">
   <table width="45%" border="0">
     <tr>
       <td><bean:message key="app.symbol" />:</td>
       <td><html:text property="symbol" /></td>
     </tr>
     <tr>
       <td colspan="2" align="center"><html:submit /></td>
     </tr>
   </table>
 </html:form>
```

In addition to specifying tiles as JSP pages, you can pass text in the body of the `tiles:put` tag as the tile. The quote.jsp page does just that:

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

 <tiles:insert page="/siteLayout.jsp" flush="true">
     <tiles:put name="title" type="string"
                      value="Rick Hightower Stock Quote" />
```

```
    <tiles:put name="header" value="/header.jsp" />
    <tiles:put name="footer" value="/footer.jsp" />
    <tiles:put name="content" type="string">
           <bean:message key="app.price"/>
           <%= request.getAttribute("PRICE") %>
    </tiles:put>
</tiles:insert>
```

Notice that the `tiles:put` tag's body contains the contents of quote.jsp. Everything else is defined by the layout and the same tiles you used in the last example. The advantage is that you reduce the number of JSP pages in the system. I've seen long debates about which approach works best, and I've concluded it depends on how much code will be in the body of the put.

Do you see a problem here? There is a rule: Don't repeat yourself (DRY), and you have broken it. Do you see how?

Section 4. Tile definitions

Create a definition

Unfortunately, both quote.jsp and index.jsp break the DRY rule in that they repeat defining the header and footer parameters. Because they both use the same parameter values, it would be nice not to repeat the same parameters for both pages.

Imagine a real application in which the tile layout included many more regions (say, eight) and many more pages used that tile layout. You'd find it a pain to repeat every parameter every time you wanted to use a tile layout. Since most pages will use the same header and footer, you'd benefit by defining them in one place instead of in each page.

Going back to the display function metaphor, remember in some respects the tile layout resembles a display function. You invoke the tile layout using `tiles:insert` and pass in parameters using `tiles:put`. The parameters are other JSP pages and strings that can be inserted into regions of the tile layout.

You now need the ability to define default parameters corresponding to the header and footer regions. The Tiles framework also lets you pass default arguments to a tile layout using definitions.

In this section, you'll learn to create and use definitions. Definitions define default parameters for tile layouts. You can define definitions in JSP code or XML. By the end of this section you will be able to do either.

Create and use a JSP definition

You'll find creating a definition with the JSP pages the easiest method, as it requires the least configuration.

To create a JSP definition do the following:

1. Import the tiles tag library using the `taglib` directive.
2. Ensure that the definition is defined only once using the `logic:notPresent` tag.
3. Define the definition with the `tiles:definition` tag passing the JSP page that defines the tile layout and the scope of the newly defined definition.

4. Define the default parameters with the `tiles:put` tag.

In the listing below, siteLayoutDefinition.jsp defines a definition that uses siteLayout.jsp as the tile layout and defines default parameters for the header and footer parameters (as well as others parameters):

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

<logic:notPresent name="siteLayoutDef" scope="application">
  <tiles:definition id="siteLayoutDef"
                    page="/siteLayout.jsp"
                    scope="application">
    <tiles:put name="title" type="string"
                     value="Rick Hightower Stock Quote System" />

    <tiles:put name="header" value="/header.jsp" />
    <tiles:put name="footer" value="/footer.jsp" />
    <tiles:put name="content" type="string">
       Content goes here
    </tiles:put>
  </tiles:definition>
</logic:notPresent>
```

The `tiles:definition` tag defines a JavaBean component of type `ComponentDefinition` (`org.apache.struts.tiles.ComponentDefinition`). `ComponentDefinition` has getter and setters for all of the attributes that you can pass it. The `logic:notPresent` tag ensures that the `ComponentDefinition` is only created once per application by checking to see if it's already in scope before defining it.

Note: Defaults can be evil.

Notice that you also define default parameters for content and title. That, however, is considered bad. Why? If someone forgets to use the title, they will get the default. Because the title should change for every page, you should not define a default for it. That way if someone forgets to pass the title, the tile layout will fail. In order for it to fail, you need to do two things:

- Don't define a default in the definition.
- Don't set `ignore` equal to true when defining the region in the tile layout with the `tiles:insert` tag.

Use a JSP tile definition

Using a tile definition resembles using a tile layout directly. The only differences: you will specify the definition instead of the tile layout JSP page, and you will pass in less parameters with `tiles:put`.

To use a tile definition, do the following:

1. Import the tiles tag library with the `taglib` directive.
2. Include the JSP page that defines the definition with `jsp:include`.
3. Use the `tiles:insert` tag, but specify the definition bean name and scope instead of the tile layout page.
4. Use the `tiles:put` attribute to specify title and content only (not header and footer parameters).

Below you'll find an example of using a tiles definition (index2.jsp):

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<jsp:include page="siteLayoutDefinition.jsp"/>

<tiles:insert beanName="siteLayoutDef" beanScope="application">
    <tiles:put name="title" type="string"
               value="Get Rick Hightower Stock Quote 2" />
    <tiles:put name="content" value="indexContent2.jsp"/>
</tiles:insert>
```

Notice that you specify the definition you defined in the last section with the `beanName` attribute `siteLayoutDef`. The `beanName` attribute value corresponds to the `id` attribute value of the bean definition in the last section. Please notice that you specify two parameters with `tiles:put` instead of four, which means less typing and less code to maintain -- the joys of DRY.

Do you see a problem here? You must create one JSP page for the definition (siteLayoutDefinition.jsp), one for the content (indexContent.jsp), one for index.jsp itself, one for the layout (siteLayout.jsp), one for the header, and one for the footer. Whew! Sum it up and you get six JSP pages instead of one (and this is a *simple* example). Granted you are getting reusability, but it appears you get this at the expense simplicity.

Another weird thing about this example is the definition itself. JSP pages are meant to express visual things in a document-centric fashion. However, the definition has nothing inherently visual. In fact, it is mostly just configuration. A layout may have several sets of definitions, so you'd find it a real hassle to have a JSP page for each. It would be nice to have the configuration data in one location, but how?

Note: Use `jsp:include` -- not `@page include` -- for definitions

If you have worked with tiles before, you may have seen examples that use the include directive (`@page include`) instead of the dynamic include action (`jsp:include`). I prefer `jsp:include` because the include directive happens at translation time, and, unless the page that includes it changes, the new JSP definition will not be redefined. Save yourself some development hassles and use the `jsp:include` action instead. The performance difference proves negligible (the directive is slightly faster), but the pain of dated JSP definitions boggles the mind.

Create and use an XML definition

XML definitions answer the problems with the explosion of non-visual JSPpages. Instead of defining one definition per JSPpage, you can define all definitions in one configuration file. Before you can start using XML definition, however, you need to use the corresponding Tiles plug-in for Struts:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
  <set-property property="definitions-config"
                value="/WEB-INF/tiles-defs.xml" />
  <set-property property="moduleAware" value="true" />
  <set-property property="definitions-parser-validate" value="true" />
</plug-in>
```

You need to add the above code to your struts configuration file. Notice that the `definition-config` property specifies the XML file that will contain the XML-based definitions. You also specify that the tiles engine is module-aware and that it validates the XML file.

Create an XML tile definition

Creating an XML definition becomes easy once you define the plug-in. You merely add another entry in the tiles definition file (`tiles-def.xml` for our example):

```
<tiles-definitions>

  <definition name="siteLayoutDef" path="/siteLayout.jsp">
    <put name="title" value="Rick Hightower Stock Quote System" />
    <put name="header" value="/header.jsp" />
    <put name="footer" value="/footer.jsp" />
    <put name="content" type="string">
```

```
        Content goes here
      </put>
    </definition>
 ...
```

The root element is `tiles-definition;` -- all tile definitions for this module will be defined inside the tiles-definition element.

The definition element specifies a tile definition. The definition defined above functionally equals the JSP version you defined earlier. Notice that the definition's attributes differ a little: You use `name` instead of `id` and `path` instead of `page`. (Irritating, isn't it?) If you know how to define a JSP-based definition, then defining an XML-based definition proves child's play, as they are nearly identical in form and function.

Use an XML tile definition

Now that you've defined the XML definition, you need to change quote.jsp and index.jsp to use it. Using the definition proves little different than before. The only difference is the attributes you pass the `tiles:insert` tag as follows (index3.jsp):

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<tiles:insert definition="siteLayoutDef">
    <tiles:put name="title" type="string"
               value="Get Rick Hightower Stock Quote 3" />
    <tiles:put name="content" value="indexContent3.jsp"/>
</tiles:insert>
```

Notice that now you use the definition attribute to specify the name of the definition defined in the tiles definition file (`tiles-def.xml`) instead of using `beanName` and `beanScope`. Also notice that you need not use `jsp:include` or `logic:notPresent` in the definition JSP.

Once you take the leap of faith and start using XML definitions instead of JSP definitions, life with tiles becomes a little easier. You write less code and maintain fewer tiny, non-visual JSP pages.

Section 5. Advanced Tile topics

Understand and use tile scope

Remember the tiles framework defines an additional scope called tile scope, similar to page scope. Like page scope, tile scope is more private than request scope. The tile scope lets the tiles user pass arguments (called parameters) to the tile. In essence, it makes the page callable like a display function.

Remember, `jsp:include` lets you call a page passing it request parameters (jsp:param). The `tiles:insert` tag resembles `jsp:include` but is more powerful. The `tiles:insert` tag lets you call a page passing it subpages (called tiles) and attributes. The tile scope essentially lets you pass variables available only to that tile layout.

You'll understand tile scope if you know how it is implemented. In my travels, I created a debug utility called listTileScope that lets me print out variables in tile scope, as seen in the code snippet below:

```
import org.apache.struts.taglib.tiles.ComponentConstants;
import org.apache.struts.tiles.ComponentContext;

public static void listTileScope(PageContext context)
                   throws JspException, IOException {

    JspWriter out = context.getOut();
    ComponentContext compContext =
            (ComponentContext)context.getAttribute(
                        ComponentConstants.COMPONENT_CONTEXT,
                        PageContext.REQUEST_SCOPE);

    out.println("--- TILE Attributes --- <br />");

    if (compContext!=null){

        Iterator iter = compContext.getAttributeNames();
        while(iter.hasNext()){
            String name = (String)iter.next();
            Object value = compContext.getAttribute(name);
            printNameValueType(name, value, out);
        }
    }else{
        out.println("---TILE Attributes NOT FOUND---<br />");
    }

    out.println("-------------------------- <br />");

}

private static void printNameValueType(
```

```
                              String name,
                              Object value,
                              JspWriter out)
                                      throws IOException{

     if (value !=null){

         out.println(
         name + " = " +  value +
         " type (" +
             value.getClass().getName()+ ") " +
         "<br /><br />");

     }else{
         out.println(name + " = " +  value +
         "<br /><br />");

     }
 }
```

Notice that the `ComponentContext` class implements the tile scope. The `ComponentContext` class is located in request scope under the `ComponentConstants.COMPONENT_CONTEXT` key. The tiles system ensures that each tile gets its own component context.

Nested tiles do not share the same tile scope as their parent (I learned this one the hard way). The current tile's tile scope is saved before displaying the nested tile. After the nested tile finishes, the parent tile scope is restored into the request. This magic is done in the `InsertTag (org.apache.struts.taglib.tiles.InsertTag)` class's nested class `InsertHandler`.

Use bean attributes as parameters

So far you have passed attributes into the tile layout that corresponds to subtiles or simple strings. You can pass in any bean type you want as attributes into the tile layout. Then use that attribute inside of the tile layout.

Let's say your application has an action that puts a `User` object into session scope, perhaps after the user logs into the system:

```
   public ActionForward execute(ActionMapping mapping,
     ActionForm form,
     HttpServletRequest request,
     HttpServletResponse response)

     throws IOException, ServletException {

     // Default target to success
     String target = new String("success");
```

```
    // If login successful.
    UserDomainObject user = new UserDomainObject();
    ...
    request.getSession().setAttribute("user", user);
    return (mapping.findForward(target));
  }
```

Next you pass that user to a tile you are inserting. In this example, you will use the tile you used inside your tile layout (siteLayout2.jsp):

```
<tiles:insert attribute="header" ignore="true">
    <tiles:put name="title" beanName="title" beanScope="tile"/>
    <tiles:put name="user" beanName="user"
                                        beanScope="session"/>
</tiles:insert>
```

The tile layout passes the `user` bean to the header tile by specifying a scope of `session` and a bean name of `user`. You can pass any bean from any JSP scope into the tile or tile layout using this technique, thus the tile scope becomes just another scope. That is not much different than before.

To use this `user` bean in header.jsp, copy it from tile scope into a scope that other beans understand. You can do this using the `tiles:useAttribute` tag. The `tiles:useAttribute` tag is analogous to the `jsp:useBean` action, except that it works only with tile scope (header2.jsp):

```
<tiles:useAttribute id="user"
                    name="user"
                    classname="rickhightower.UserDomainObject"
                    />
```

Thus `tiles:useAttribute` copies the user object from tile scope into page scope. Once the bean is defined, you can start using it as you would with any bean defined in page scope:

```
              <bean:write name="user" property="userName"/>
```

Next, let's see the complete listing for the new header2.jsp file:

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

<tiles:useAttribute id="user"
                    name="user"
                    classname="rickhightower.UserDomainObject"
                    />
```

```
      <td width="33%" bgcolor="#36566E">

         <div align='left'>
         <font size="1" color="orange">

               currently logged in as
              <bean:write name="user" property="userName"/>
         </font>

         </div>

       </td>


      <td width="33%">

          <font color="#36566E">
              <tiles:getAsString name="title" ignore="true"/>
          </font>

      </td>
      <td width="33%" bgcolor="#36566E">

         <div align='left'>

         <font size="1" color="white">

             <blockquote>

              <bean:write name="user" property="firstName"/>

               <br />

              <bean:write name="user" property="lastName"/>

               <br />

              </blockquote>

         </font>

         </div>



      </td>

 </tr>

 </table>

 </center>
```

As you can see, the header now displays information about the current user logged into the site -- a powerful feature. You can create tiles that specialize in displaying domain

objects, and then reuse those tiles in many parts of our application. Considering that, it's easy to see why they originally considered calling the Tiles framework components: You can, in effect, create display components. Unlike custom tags (pre JSP 2.0), these components are all created in JSP pages.

Section 6. Lists

Understand lists

You frequently must pass more than one parameter. For example, you may want to pass a list of parameters to display links in the navigation region of the tile layout.

Going back to the Employee Listing example, you may have a Web application that displays a company's divisions, departments, and employees. When in the employee listing view, employeeListing.jsp will be in the content region of the tile layout and the current division's department links will be in the navigation region. When the user clicks on a department link, a new listing of that department's employees will appear. When in the department view, deptListing.jsp will be in the content region of the tile layout, and the company's list of division links will be in the navigation region. When the user clicks on a division link, a new listing of departments will appear. Thus, each page (employeeListing.jsp and deptListing.jsp) will pass in a new list of links. You can accomplish that with `putList`.

Use putList in XML

Tiles lets users pass in links using the `putList` subelement -- usable in XML and JSP definitions or in calls to a definition or tile layout from a JSP.

Let's say you want a standard set of navigational links for your site layout. You can specify such links with the `putList` subelement in your tiles configuration file as follows (`tiles-def.xml`):

```
<definition name="siteLayoutDef3" path="/siteLayout3.jsp">
  <put name="title" value="Rick Hightower Stock Quote System" />
  <put name="header" value="/header2.jsp" />
  <put name="footer" value="/footer.jsp" />
  <put name="content" type="string">
     Content goes here
  </put>

  <putList name="items" >
    <item value="Home"
          link="/index.html"  />
    <item value="Wiley"

          link="http://www.wiley.com"  />
    <item value="Trivera Technologies"
          link="http://www.triveratech.com/"  />
    <item value="Virtuas"
```

```
            link="http://www.virtuas.com/"  />
      <item value="Rick Hightower"
            link="http://www.rickhightower.com"  />
      <item value="Rick's Blog"
            link="http://rickhightower.blogspot.com/"  />
    </putList>
  </definition>
```

The `putList` element lets you specify a list of items associated with links. In the above listing, `putList` defines six links.

The `items` list (`java.util.List`) gets put into tile scope. The name `items` is set with the `name` attribute of the `putList` element.

The `item` element defines a link by putting an instance of `org.apache.struts.tiles.beans.MenuItem` in the list. The value attribute corresponds to the label on the link, while the link refers to the link's URL.

Icon and tool tip

The `item` element also has elements for specifying the tool tip and the icon for the link. You can learn more about the `item` element and `putList` by looking at the DTD (`tiles-config_1_1.dtd`), found in the Struts source.

Use the list in the tile layout

To use this list of links, you must modify the tile layout as follows (siteLayout3.jsp):

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

<tiles:importAttribute />

<html>
  <head>
    <logic:present name="title">
        <title>
        <tiles:getAsString name="title" ignore="true"/>
        </title>
    </logic:present>
  </head>

  <body>
    <table width="500" border="0" cellspacing="0" cellpadding="0">

      <tr bgcolor="#36566E">
        <td height="68" width="70%">
```

```
          <div align="left">
   <img src="images/hp_logo_rickhightower.gif" width="220" height="74">
          </div>
        </td>
      </tr>

      <tr>
        <td height="68" width="2000">
            <tiles:insert attribute="header" ignore="true">
               <tiles:put name="title"
                          beanName="title" beanScope="tile"/>
               <tiles:put name="user"
                          beanName="user" beanScope="session"/>
            </tiles:insert>
         </td>

      </tr>

      <table>
      <tr>
        <td width="50%">
        <ul>
        <logic:iterate id="item" name="items"
                type="org.apache.struts.tiles.beans.MenuItem" >
           <li>

           <bean:define id="link" name="item" property="link"
                        type="java.lang.String"/>

           <logic:match name="link"
                        location="start" value="/" >
                  <html:link page="<%=link%>" >
                       <bean:write name="item"
                                   property="value"/>
                    </html:link>
               </logic:match>
           <logic:notMatch name="link"
                           location="start" value="/" >
                   <html:link href="<%=link%>">
                       <bean:write name="item"
                                   property="value"/>
                    </html:link>
               </logic:notMatch>

           </li>
            </logic:iterate>
        </ul>
        </td>
        <td width="50%">
            <div align="center">
            <tiles:insert attribute="content"/>
            </div>
        </td>
      </tr>
      </table>

      <tr>
        <td>
```

```
            <tiles:insert attribute="footer" ignore="true"/>
        </td>
      </tr>

    </table>

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

Pay particular attention to the code section that iterates over the list:

```
        <ul>
         <logic:iterate id="item" name="items"
                 type="org.apache.struts.tiles.beans.MenuItem" >
            <li>

            <bean:define id="link" name="item" property="link"
                         type="java.lang.String"/>

            <logic:match name="link"
                         location="start" value="/" >
                    <html:link page="<%=link%>" >
                        <bean:write name="item"
                                  property="value"/>
                     </html:link>
                </logic:match>
            <logic:notMatch name="link"
                            location="start" value="/" >
                    <html:link href="<%=link%>">
                        <bean:write name="item"
                                  property="value"/>
                     </html:link>
                </logic:notMatch>

            </li>
             </logic:iterate>
         </ul>
```

Later you will simplify this.

Use tiles:importAttribute

The `tiles:importAttribute` tag imports the attributes in tile scope into page scope. It resembles the `tiles:useAttrribute` tag, but it's closer to a shotgun than a scalpel. It is lazy, dirty, and cheap; I use it all the time (what does this say about me?). This effectively copies the items list from tile scope into page scope.

Note: `tiles:importAttribute` copies into any scope you specify.

By default, `tiles:importAttribute` copies all of the attributes into page scope. You can copy the attributes into other scopes as well by using the scope attribute.

Once the items list is in page scope, you can access it with the standard Struts tags as follows (siteLayout3.jsp):

```
 <logic:iterate id="item" name="items"
    type="org.apache.struts.tiles.beans.MenuItem" >

  ...
</logic:iterate>
```

Notice the logic you implement with the `logic` tag for displaying the link. You check to see whether the link begins with "/" to determine if the link is relative. If the link *is* relative, use the `html:link` tag's `page` attribute. Otherwise, use the `html:link` tag's `href` attribute if the link refers to an absolute URL as follows (siteLayout3.jsp):

```
<bean:define id="link" name="item" property="link"
             type="java.lang.String"/>

<logic:match name="link"
             location="start" value="/" >
        <html:link page="<%=link%>" >
            <bean:write name="item"
                       property="value"/>
        </html:link>
 </logic:match>
<logic:notMatch name="link"
                location="start" value="/" >
        <html:link href="<%=link%>">
            <bean:write name="item"
                       property="value"/>
         </html:link>
 </logic:notMatch>
```

As you can imagine, you may want to use this bit of display logic to display menu items in more than one location. That is, you may want to reuse it outside of this page's scope. In a later section you'll see how to do this by nesting one tile layout into another tile layout.

Use putList in JSP

In addition to adding items to the list in the tile definition, you can add items to the list in the JSP using the `tiles:putList` element and its `tiles:add` subelement (index6.jsp):

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

<%@ page import="org.apache.struts.tiles.beans.SimpleMenuItem" %>
<tiles:insert definition="siteLayoutDef4">
    <tiles:put name="title" type="string"
               value="Get Rick Hightower Stock Quote6" />
    <tiles:put name="content" value="indexContent5.jsp"/>

    <tiles:putList name="items" >
        <jsp:useBean id="item1" class="SimpleMenuItem"/>
        <jsp:setProperty name="item1" property="link"
                         value="/index.html"/>
        <jsp:setProperty name="item1" property="value"
                         value="Home" />
        <tiles:add beanName="item1"/>
    </tiles:putList>
</tiles:insert>
```

The above listing uses `jsp:useBean` to create an instance of `SimpleMenuItem`. Then it uses `jsp:setProperty` to set the link and value properties of the `SimpleMenuItem` bean. Lastly, it adds this bean to the list using `tiles:add`.

In the above example, you add a `SimpleMenuItem`, which subclasses the `MenuItem` that the tile layout uses. However, you can add any bean type.

Note: Adding any bean type in XML

To add any bean type in the tiles XML definition, use the `putList`'s subelement bean. The bean element takes an `id` and `classtype`. For simple types, you can use `putList`'s `add` subelement as well. See the tiles configuration DTD (`tiles-config_1_1.dtd`) for more information.

Section 7. Advanced definition concepts

Extend definitions

Several JSP pages frequently use the same defaults parameters. Yet other pages use the same tile layout but use different tile parameters. Instead of defining a completely different definition, one definition can extend another definition. The `extends` attribute lets one definition extend another.

Here's an example:

```
<definition name="siteLayoutDef3" path="/siteLayout3.jsp">
  <put name="title" value="Rick Hightower Stock Quote System" />
  <put name="header" value="/header2.jsp" />
  <put name="footer" value="/footer.jsp" />
  <put name="content" type="string">
     Content goes here
  </put>

  <putList name="items" >
    <item value="Home"
          link="/index.html"  />
    <item value="Wiley"
          link="http://www.wiley.com"  />
    <item value="Trivera Technologies"
          link="http://www.triveratech.com/"  />
    <item value="Virtuas"
          link="http://www.virtuas.com/"  />
    <item value="Rick Hightower"
          link="http://www.rickhightower.com"  />
    <item value="Rick's Blog"
          link="http://rickhightower.blogspot.com/"  />
  </putList>
</definition>

<definition name="siteLayoutDef4" extends="siteLayoutDef3">
  <put name="title" value="Rick Hightower Quote Sub System" />
  <putList name="items" >
    <item value="Home"
          link="/index.html"  />
    <item value="Wiley"
          link="http://www.wiley.com"  />
    <item value="Trivera Technologies"
          link="http://www.triveratech.com/"  />
    <item value="Virtuas"
          link="http://www.virtuas.com/"  />
  </putList>
 </definition>

<definition name="siteLayoutDef5"  extends="siteLayoutDef4">
  <putList name="items" >
  </putList>
```

```
</definition>

<definition name="siteLayoutDef6" path="/siteLayout4.jsp"
            extends="siteLayoutDef4">
</definition>
```

Notice that `siteLayoutDef4` extends `siteLayoutDef3`, overrides the values of title, and defines a shorter navigation list. It inherits all the other parameters from `siteLayoutDef4` that it overrode -- that is, header, footer, and content. Furthermore, notice that `siteLayoutDef5` extends `siteLayout4`, except it blanks out the list of items. A definition inherits all of the attributes of its super definition and all of its super definition's super definitions, ad infinitum.

In addition to overriding attributes, you can change the tile layout JSP. Look at `siteLayoutDef6` as it extends `siteLayoutDef5` and specifies a new tile layout (siteLayout4.jsp).

Nest tiles

One tile layout can insert another tile layout, and so on. In fact, you can create tile layouts so small that they are not really templates per se. Instead, they become small visual components more similar to custom tags instead of page templates.

Remember the logic you implemented for displaying a link. You checked to see if the link begins with "/" to see if the link is relative or not and then displayed it correctly. If you wanted to use that same routine in several places in your application, you would create a visual component.

Create a visual component

A visual component is just another tile layout. Whether a tile layout is a visual component instead of a template just depends on your viewpoint (the eye of the beholder). The following tile layout defines a visual component for displaying a link (linkLayout.jsp):

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

   <tiles:useAttribute id="item"
          name="item"
          classname="org.apache.struts.tiles.beans.MenuItem"
```

```
                    />

    <bean:define id="link" name="item" property="link"
                 type="java.lang.String"/>

    <logic:match name="link" location="start" value="/" >
        <html:link page="<%=link%>" >
            <bean:write name="item" property="value"/>
             </html:link>
    </logic:match>
    <logic:notMatch name="link" location="start" value="/" >
        <html:link href="<%=link%>">
            <bean:write name="item" property="value"/>
             </html:link>
    </logic:notMatch>
```

This approach, compared to JSP Custom tags, lets you use other custom tags. Also, it's a document-centric JSP compared to a Java class like a Custom tag, which makes using HTML tags and such easier.

Note: JSP 2.0 tag files

You can realize many of the advantages from tile layouts with JSP tag files available in JSP 2.0 and later. If you use an older JSP version that does not support tag files, then you can start using this technique now. However, as you'll soon see, the Tiles framework, in my opinion, better separates the controller from the view.

Use a visual component

Once you define the visual component, you should create a definition for it, as follows:

```
<definition name="linkLayoutDef" path="/linkLayout.jsp">
</definition>
```

Now that you've defined the definition, you can use this visual component on any page using the `tiles:insert tag`. You even use this visual component inside of another tile. The following code example uses this visual component inside the tile layout you define earlier (siteLayout4.jsp).

```
    <td width="50%">
         <ul>
         <logic:iterate id="item" name="items"
                type="org.apache.struts.tiles.beans.MenuItem" >
            <li>

                  <tiles:insert definition="linkLayoutDef">
                    <tiles:put name="item"
                               beanName="item"
```

```
                                  beanScope="page"/>
                  </tiles:insert>

            </li>
             </logic:iterate>
         </ul>
         </td>
 ...
```

The above code iterates over the list of items, then calls `tiles:insert`, passing the current item to the visual component (`linkLayoutDef`) to display. The visual component knows how to display a domain object (a menu item). Any time you see yourself repeating the same JSP code lines over and over again, consider writing a visual component using a tile layout.

Use a tile as a parameter to another tile

The previous example explicitly called the visual component you defined. What if the tile layout you use varies based on several factors (whether the user is logged in or not, whether the user is in a certain role, which part of the site you are on)? In that situation, it would be nice to pass the tile as a parameter.

You can do so using the `put` element as follows (`tiles-def.xml`):

```
   <definition name="link.layout.def" path="/linkLayout.jsp">
   </definition>

   <definition name="siteLayoutDef7" path="/siteLayout5.jsp" extends="siteLayoutDef4">
     <put name="title" value="Rick Hightower Quote System 9" />
     <putList name="items" >
     </putList>
     <put name="linkDisplay" value="link.layout.def"/>
   </definition>
```

Notice that `siteLayoutDef7`'s `linkDisplay` attribute's value equals `link.layout.def`. Now inside your tile layout (siteLayout5.jsp) you specify the `linkDisplay` attribute instead of specifically calling a particular tile layout definition:

```
        <ul>
         <logic:iterate id="item" name="items"
                type="org.apache.struts.tiles.beans.MenuItem">
            <li>

             <tiles:insert attribute="linkDisplay">
                    <tiles:put name="item"
                               beanName="item"
                               beanScope="page"/>
```

```
            </tiles:insert>

           </li>
            </logic:iterate>
        </ul>
```

Thus your site layout does not know which visual component it is using. You can programmatically switch how pieces of the layout display by switching which visual component the site layout uses.

Tile controller

If you feel you are putting too much Java code into your tile layout or you have to put the same Java code into every action that forwards to a page that uses a particular tile layout, then you should use a tile controller. You can specify a controller class that gets called before a tile is inserted using the `controllerClass` attribute:

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

<tiles:insert definition="siteLayoutDef5"
              controllerClass="rickhightower.SimpleController">
    <tiles:put name="content" value="indexContent5.jsp" />
</tiles:insert>
```

The controller class resembles an action. In the controller, you can map model objects into scope so that the tile can display the items.

To write a tile controller, you must do the following:

1. Create a class that implements `org.apache.struts.tiles.Controller`.
2. Implement the `perform()` method.
3. In the `perform()` method, do something with the model and map the results into scope so the tile can use it.

The following listing shows a way to implement a controller (`rickhightower.SimpleController`):

```
package rickhightower;

import java.io.IOException;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.tiles.ComponentContext;
import org.apache.struts.tiles.Controller;


import org.apache.struts.tiles.beans.MenuItem;
import org.apache.struts.tiles.beans.SimpleMenuItem;

import java.util.ArrayList;
import java.util.List;

/**
 * @author rhightower
 */
public class SimpleController implements Controller{


    private MenuItem createMenuItem(String label, String link){
            SimpleMenuItem item = new SimpleMenuItem();
            item.setLink(link);
            item.setValue(label);
            return item;
    }

    private List getLinks(){
            List list = new ArrayList();

            list.add(createMenuItem("Home",
                  "/index.html"));

            list.add(createMenuItem("Rick's",
            "http://www.rickhightower.com"));

            list.add(createMenuItem("Trivera",
                  "http://www.triveratech.com"));

            return list;

    }
      /* (non-Javadoc)
       *
      */
      public void perform(ComponentContext context,
                          HttpServletRequest request,
                          HttpServletResponse response,
                          ServletContext servletContext)
                        throws ServletException, IOException {
            List items = (List)getLinks();
            context.putAttribute("items",items);
      }
}
```

Notice that the `perform()` method gets passed the component context. The component context holds the tile scope's attributes. Putting things in the component context puts them in the tile scope. In this simple example, you call `getLinks`, which returns a simple `MenuItems` list you map into tile scope. A real world example would

likely talk to the model -- perhaps a facade that communicates with a database to look up links specific to the type of user logged into the system.

Note: Using an action as the controller.

You can instead use an action as the controller for the tile. To do this, you specify the path of the action with the `controllerUrl` attribute.

Use a tile definition as an ActionForward

You may be unaware that when you installed the Tiles plug-in, it installed a custom request processor that extends the way Struts handles `ActionForward`. Thus, you can forward to a tile definition instead of a JSP page.

Let's say you have a definition defined like so:

```
<definition name="main.index" extends="siteLayoutDef7">

  <put name="content" value="/indexContent.jsp"/>
</definition>
```

In your struts configuration file, you can define a forward that forwards to the `main.index` definition instead of specifying a JSP page:

```
<action
              path="/Lookup"
              type="rickhightower.SimpleLookupAction"
              name="lookupForm"
              input="/index.jsp">

        <forward name="success" path="/quote.jsp"/>
        <!-- forward name="failure" path="/index.jsp"/ -->
        <forward name="failure" path="main.index" />
</action>
```

Being able to forward to a definition proves a powerful tool for eliminating extraneous logic from your JSPs. For example, if a user was logged in as a manager instead of a regular user, you could forward that user to a definition that defined special parameter tiles that only a manager could use.

The manager's definition could extend the regular user's definition. They could even use the same tile layout if the tile layout used the `insert` tag with the `ignore` attribute. The action would select the correct forward. You would not need to use a `logic:*` tag at all.

Getting logic out of your JSP and into the controller is a step in the right direction and is

so much easier with the Tiles framework.

Section 8. Wrap-up and resources

Summary

If you are new to the Tiles framework and have worked through this tutorial, then you are well on your way. In a relatively small timeframe, we covered:

- The Tiles framework and architecture
- How to build and use a tile layout as a site template
- How to use tile definitions both in XML and JSP
- How to move objects in and out of tile scope
- How to work with attributes lists
- How to nest tiles
- How to build and use tile layouts as small visual components
- How to subclass a definition
- How to create a controller for a tile
- How to use a tile as an `ActionForward`

The Tiles framework makes creating reusable pages and visual components easier. Developers can build Web applications by assembling reusable tiles. You can use tiles as templates or as visual components.

In some respects, the tile layout is like a display function. First you pass tile layout parameters to use. The parameters can be simple strings, beans, or tiles. The parameters become attributes to the tile and get stored in the tile's tile scope. For its part, the tile scope resembles page scope, and is less general than request scope. The tile scope lets the tile's user pass arguments (called attributes) to the tile.

Definitions let you define default parameters for tiles. Definitions can be defined in JSP or XML. Definitions can extend other definitions similarly to how a class can extend another class. Moreover, definitions can override parts of the definition it is extending.

The Tiles framework includes its own `RequestProcessor` to handle tile layouts as `ActionForward`s. Thus you can forward to a tile definition instead of a JSP if you install the Tiles plug-in.

If you are using Struts but not Tiles, then you are not fully benefiting from Struts and likely repeat yourself unnecessarily. The Tiles framework makes creating reusable site layouts and visual components feasible.

Resources

- Download the source code used in this tutorial. Two versions are available: one with jar files and one without jar files.

- If you want to use Tiles with Struts, download *Struts 1.1*, which includes the Tiles framework.

- If you want to use Tiles standalone, visit the *Tiles Web site* for downloads and other useful information.

- Download the latest version of Apache Tomcat at the *official Tomcat Web site*.

- Tomcat 5 has some pretty nifty additions. This article by Sing Li (*developerWorks*, March 2003) covers one of them, *servlet filtering*, in detail.

- *developerWorks* hosts the following articles on Struts/Tiles:
  - Malcolm Davis's "*Struts, an open-source MVC implementation* " (February 2001) introduces Struts and shows how to manage complexity in large Web sites.
  - Wellie Chao's "*Struts and Tiles aid component-based development* " (June 2002) combines both Struts and Tiles to construct Web applications.
  - In "*Struttin' your stuff with WebSphere Studio Application Developer, Part 2: Tiles*," (November 2002) David Carew shows how to use the Tiles templating framework with Struts.
  - "*Integrating Struts, Tiles, and JavaServer Faces*" (September 2003) combines the power of all three technologies into one powerful package.
  - "*Architect Struts applications for Web services*" (April 2003) brings the power of MVC to Web services.
  - David Carew's "*Go-ForIt Chronicles, Part 19: Struttin' your stuff with WebSphere Studio*" (September 2002) tutorial will get you started building Struts-based applications using the WebSphere Studio IDE.
  - The author, Rick Hightower, also co-authored *Mastering Jakarta Struts, 2nd edition* with James Goodwill (Wrox Press).
  - Chuck Cavaness, author of *Programming Jakarta Struts*, excerpts this *four-part series on Tiles* from his book.

- *JavaWorld* also offers a look at Tiles in "*UI design with Struts and Tiles*" by Prakash Malani.

- You'll find hundreds of articles about every aspect of Java programming in the IBM *developerWorks* *Java technology zone*.

Feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. We'd also like to hear about other tutorial topics you'd like to see covered.

For questions about the content of this tutorial, contact the author, Richard Hightower, at *rick_m_hightower@hotmail.com*.

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.