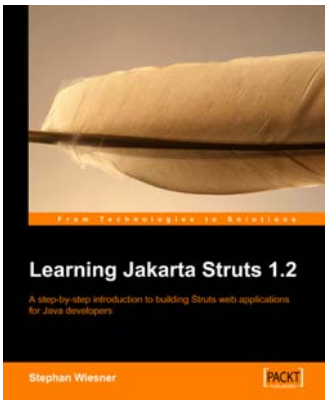
Learning Jakarta Struts 1.2:

A concise and practical tutorial

Stephan Wiesner



Chapter 4
"Internationalization and Taglibs"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter 4 "Internationalization and Taglibs"

A synopsis of the book's content

Information on where to buy this book

About the Author

Stephan Wiesner was born in October 1973 in Lüneburg, Germany. He graduated in business informatics in 2003. He currently lives and works in Lucerne, Swizerland, as a QS consultant and test manager.

4

Internationalization and Taglibs

In this chapter, we create the fundamental structure for the shop, illustrating key concepts of the Struts framework. We also look at configuring our web application to support multiple languages. Finally, the Custom Actions, or Tag Libraries, are introduced.

4.1 The Structure for the Shop

We constructed a simple Struts-capable website in Chapter 2. We can use that as a starting point here. Simply stop Tomcat, copy the `shop1` folder in `webapps`, and rename it as `shop2`.

4.2 Internationalization

Internationalization is a complex and involved subject. In brief, it refers to the automatic rendering of an application in the user's chosen language. It relates not only to the text itself, but also to numbers, date format, and currency values. Special symbols and alphabetical sorting in different languages bring interesting and unexpected problems with them.

4.2.1 Internationalization and Java

Java offers a variety of internationalization options for applications. The following two tutorials on Sun's website cover the subject quite well:

```
http://java.sun.com/j2ee/1.4/docs/tutorial/doc/
   WebI18N.html#wp76431
http://java.sun.com/docs/books/tutorial/i18n/resbundle/
   concept.html
```

Let's take a quick look at a very simple example:

Listing 4.1: Test.java

```
import java.util.Locale;
import java.text.NumberFormat;

public class Test
{
  public static void main(String args[])
  {
    String amountOut;
    NumberFormat numberFormatter;
    Double amount = new Double(345987.246);

    Locale en = new Locale("en", "US");
    Locale de = new Locale("de", "DE");

    // US output
    numberFormatter = NumberFormat.getNumberInstance(en);
    amountOut = numberFormatter.format(amount);
    System.out.println(amountOut + " "
                       +
NumberFormat.getCurrencyInstance(en).format(amount)
                       + " " + en.toString());

    // German output
    numberFormatter = NumberFormat.getNumberInstance(de);
    amountOut = numberFormatter.format(amount);
    System.out.println(amountOut + " "
                       +
NumberFormat.getCurrencyInstance(de).format(amount)
                       + " " + de.toString());
  }
}
```

This will give you the following output:

```
345,987.246 $345,987.25 en_US
345.987,246 345.987,25 Ç de_DE
```

The German currency should, of course, use the Euro symbol; a wrong character appears here as a consequence of the limited character set available in the DOS console.

Java allows us to define the text for several languages with the help of configuration files. A suitable language is chosen depending on the user's system language. A similar mechanism applies with Struts as well, but instead of the system language, it uses the language specified by the browser (see section 4.4, *Custom Actions*).

4.2.2 Internationalization and Struts

One of the limitations in Struts is that it does not offer any facility to accept input from languages that have special symbols. This is a particular issue for Asian languages where the browser has to provide support.

In Chapter 1 you saw how a page can be multilingual. How does Struts know where to look for the correct text for the language? The `WEB-INF/struts-config.xml` file contains the following entry:

```
<message-resources parameter="resources.application"/>
```

The first part of the `parameter` attribute value, `resources`, indicates the name of the subfolder within `WEB-INF/classes` that holds the language files. The second part, `application`, gives the filename "stem" of the language files (without the `.properties` extension and country code). A full list of country codes is at `http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt`.

So if you wish to store your language files somewhere else, you use this method to specify their location. Don't forget that you must reload the application context again whenever you adjust any configuration or language files.

You can organize your language keys within the language files in the following form:

```
classname.definition [.<further subdivision>]
```

You can group the keys according to how frequently you expect to use them. Have one row of single words, such as `general.street=street`.

Always make sure to define all the required keys, as it enables error messages to be generated correctly.

The language files used above show the problem with the naming convention. It is often not clear how to name each part and one tends to choose the first thing that comes to mind. In this case you may end up with many placed under `Author`, for instance, even though they would be better placed in the `general` category. Such little details can be really beneficial as you use the files in a day-to-day context. It is much simpler to correct such mistakes immediately, rather than trying to remember illogical key conventions.

Managing the `properties` files

Many development environments can help manage the `properties` files. The Netbeans IDE, which comes with the Java SDK, shows keys sorted alphabetically and you can work with more than one language file at once. In long files, the alphabetical ordering turns out to be difficult to work with, and so you may prefer to use your own sorting and a simple text editor. Experiment with this a little bit!

One time-tested approach has been to separate the language files into smaller files and then consolidate them with `Ant`. This allows different developers to maintain individual files. The problem is to clearly define the categories for each file and stick to them since this approach can easily lead to omissions and repetitions.

4.3 The Welcome Page for the Shop

In Chapter 1, we wrote a very simple page to demonstrate the use of language files. It is reproduced below:

Listing 4.2: index.jsp

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

<html:html locale="true">          <!-- line 5 -->
<head>
  <html:base/>
  <title>
    <bean:message key="index.title"/>  <!--line 9 -->
  </title>
</head>

<body>
  <h2><bean:message key="index.title"/></h2>
  <% out.println(request.getHeader("User-Agent")); %>
</body>
</html:html>
```

Line 1: This is a JSP directive (indicated by the enclosing `<%@ . . . %>`) specifying the language used in any scriptlets on the page. As Java is the default language, this line is often omitted, but is necessary if you are using a mixture of languages (for example, Java and JavaScript using JRun instead of Tomcat).

Lines 2 and 3: These are directives that act somewhat like `import` statements in normal Java classes. We need them if we wish to use custom actions on the page (see the *Custom Actions* section).

Line 5: The `locale="true"` in this line indicates that multilingualism is to be applied (see the exercises at the end of the chapter).

Line 9: This uses a `bean` tag (see the *Custom Actions* section), which represents an object of the `org.apache.struts.taglib.bean.MessageTag` class. The `message` bean is called with a `key` attribute of `index.title`. Struts now searches for suitable language files for the language specified by the browser and inserts the corresponding text in place of the tag.

Visitors to our shop will be greeted with a welcome page in either their preferred language or the default language. Change `index.jsp` so that a company logo and a copyright notice are displayed:

Listing 4.3: index.jsp

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

<html:html locale="true">
<head>
<html:base/>
<title>
  <bean:message key="index.title"/>
</title>
</head>

<body>
  <h2><bean:message key="index.title"/></h2>
  <img src="pics/logo.jpg" alt="Logo" />
  <br />
  <br />
  <bean:message key="general.copyright"/>
</body>
</html:html>
```

You'll need to create a new `pics` folder for the logo directly under the main `shop2` folder. Choose a suitable image file to use and place it in the folder with the name `logo.jpg`. Relevant text for the `general.copyright` key must also be added to the language files.

Multilingual Images

The Struts `<html:img>` tag allows different languages to use different images, each with different `alt` text:

```
<html:img pageKey="button.clickMe.src"
altKey="button.clickMe.alt"/>
```

And then the following lines are required in the language files:

```
button.clickMe.src=/images/button_clickMe.jpg

button.clickMe.alt=Click Me
```

See `http://jakarta.apache.org/struts/userGuide/struts-html.html#img` for more details.

Exercise 1

Create a second logo for another language. Change `index.jsp` to display the logo in the user's language. Be brave: you can mix normal HTML and Struts!

4.4 Custom Actions

JSP supports **user-defined tags** or **Custom Actions** according to the tag extension mechanism defined by the JSP specification. Custom actions have access to all the information from the **Request** and **Response** objects, and can use variables in the session. A set of custom actions is called a **Tag Library** or **Taglib**.

As Struts is simply a set of custom actions, it makes sense to know how they are created.

Before going into details we'll run through a simple example. We'll create a small action that displays the date according to the user's local convention (refer to Chapter 12 for more on this topic).

Custom actions are usually developed and deployed in JAR files. In our case though, we will be using the class files compiled from our source code as we write it, so create a subdirectory called `mytags` under `WEB-INF/classes` to hold the tag files. Create a new Java file in this directory and name it `CountryDate.java`. Enter the following code:

Listing 4.4: CountryDate.java

```
package mytags;
import javax.servlet.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

public class CountryDate
       extends javax.servlet.jsp.tagext.TagSupport
{
  protected PageContext pageContext;

  public void setPageContext(PageContext pageContext)
  {
    this.pageContext = pageContext;
  }

  public int doStartTag() throws JspException
  {
    Date today = new Date();
    ServletRequest request =
         (ServletRequest)pageContext.getRequest();
    Locale user = request.getLocale();
    DateFormat dateFormatter =
         DateFormat.getDateInstance(DateFormat.DEFAULT, user);

    String now = dateFormatter.format(today);
    try
    {
      pageContext.getOut().write(now);
    }
    catch(java.io.IOException ex)
    {
      throw new JspException(ex.getMessage());
    }
```

```
        return EVAL_BODY_INCLUDE;
      }
    }
```

The `CountryDate` class extends the interface `javax.servlet.jsp.tagext.TagSupport`. This ensures that Tomcat will automatically recognize it and call `setPageContext()` passing in environment information (`request`, `session`, etc.).

The `doStartTag()` method is also defined by the interface. It is called whenever Tomcat encounters the action embedded in a JSP. In this code, this method gets the user's locale from the browser request, and uses it to create a `DateFormat` object based on the default date format for that locale. Finally, we get today's date—suitably formulated using the `format()` method of the `DateFormat` object—and write it to the current output stream. This is usually the response page, which the user sees.

If you try to compile the class now, you will get an error message saying:

```
C:\Tomcat\webapps\Shop2\WEB-INF\classes>javac mytags/*.java
mytags/CountryDate.java:3: package javax.servlet.jsp does not exist
import javax.servlet.jsp.*;
^
```

We see this because the JSP and servlet packages, `servlet-api.jar` and `jsp-api.jar`, are not part of the standard Java packages and so must be added to the classpath.

Adding the JSP and servlet packages to the classpath

You can either add `servlet-api.jar` and `jsp-api.jar` (found within Tomcat's `common\lib` directory) to your `classpath` environment variable, or you can specify them with the `-classpath` switch every time you compile with `javac`.

A good idea is to make a batch file (a text file with a `.bat` extension) in your `classes` directory containing the following two lines:

```
javac -classpath "%CATALINA_HOME%\common\lib\jsp-api.jar";
 "%CATALINA_HOME%\common\lib\servlet-api.jar" mytags/*.java

pause
```

As a batch file, you can simply double click it from Windows explorer to compile all the Java files in the `mytags` subdirectory. Note that if you added `jsp-api.jar` and `servlet-api.jar` to your classpath, you can omit the `classpath` switch shown above.

In order to use this custom action in a JSP, it must be declared and reported to Tomcat.

The `web.xml` file, which we got when we copied the `struts-blank` template, contains various `<taglib>` elements. These reference the TLD configuration files that define custom actions. Insert the following lines after the other Taglib entries in the `web.xml` for `shop2`:

```
<taglib>
  <taglib-uri>
    http://www.stephanwiesner.de/struts
  </taglib-uri>
  <taglib-location>
    /WEB-INF/taglib.tld
  </taglib-location>
</taglib>
```

The `<taglib-uri>` element is a way to identify a particular Taglib uniquely, allowing two different Taglibs to contain a custom action with the same name but still be used at the same time. It doesn't have to be a valid URL; a company (or individual) URL is often used because the company can be fairly sure no one else will use that to identify their own Taglib. So naturally, you can replace my name with your name, but remember that the name you choose should be consistent in every place where it's used. The `<taglib-location>` element specifies the name and path of the Taglib configuration file, here `/WEB-INF/taglib.tld`. Not surprisingly, you'll need to create this file as well. It's an XML file and should have the following content:

Listing 4.5: taglib.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.// DTD JSP Tag
Library 1.1//EN"
 "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>shop</shortname>
  <uri>http://www.stephanwiesner.de/struts</uri>
  <info>Taglibs for the Struts book</info>

  <tag>
    <name>countryDate</name>
    <tagclass>mytags.CountryDate</tagclass>
    <info>Displays a formulated Date</info>
  </tag>
</taglib>
```

Make sure that the `mytags.CountryDate` package specification in the `<tagclass>` element is correct. The `<info>` element contains some descriptive text. The `<name>` element defines the name that JSPs must use when they invoke our custom actions.

Now, restart the application context (or Tomcat) and we can use our new custom action in a JSP:

Listing 4.6: index.jsp

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

```
<%@ taglib uri="http://www.stephanwiesner.de/struts" prefix="sw" %>

<html:html locale="true">
<head>
  <html:base/>
  <title><bean:message key="index.title"/></title>
</head>

<body>
  <h2><bean:message key="index.title"/></h2>
  <img src="pics/logo.jpg" alt="Logo"/>
  <br />
  <br />
  <bean:message key="general.copyright"/>
  <br />
  <sw:countryDate />
</body>
</html:html>
```

Notice the third `taglib` page directive (enclosed in `<%@ . . . %>`) that imports our custom actions:

```
<%@ taglib uri="http://www.stephanwiesner.de/struts" prefix="sw" %>
```

The namespace given in the `uri` attribute must match exactly with the `<uri>` and `<taglib-uri>` elements in the configuration files. The `prefix` attribute specifies the prefix that will be used in the JSP page code to identify custom actions associated with this URL. So, to use the `countryDate` custom action, we use the following tag:

```
<sw:countryDate/>
```

All this fuss with naming might seem unnecessarily verbose and tiresome. However, imagine another Taglib written by someone else, which also defines a custom action named `countryDate`, but which has a quite different purpose to ours. Because they are contained in packages that have different URLs, we could use both on the same page without any ambiguity:

```
.
.
.
<%@ taglib uri="http://www.stephanwiesner.de/struts" prefix="sw" %>
<%@ taglib uri="http://www.java.com/jsp" prefix="sun" %>
.
.
.
<sw:countryDate /> is not the same as <sun:countryDate />
```

As you see, custom actions bear many similarities to normal HTML tags and the idea is that they can be used by web designers who know only HTML.

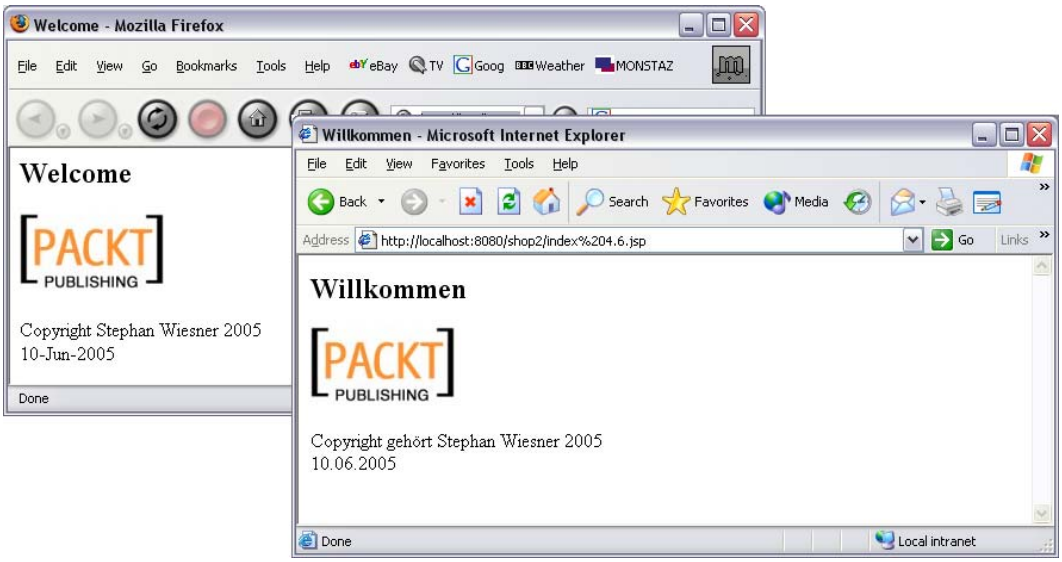


Figure 4.1: The `countryDate` custom action produces a different date format for different locales.

Like JavaBeans, custom actions make for simpler usage of frequently repeated functionality, without the need for Java code in scriplets to be embedded in the HTML. For further information, check out Hans Bergsten's *JavaServer Pages*, from O'Reilly (2000, ISBN 1-56592-746-X) or `http://www.javaworld.com/javaworld/jw-08-2000/jw-0811-jsptags.html`.

4.5 struts-bean.tld

Now that you have a basic knowledge of Taglibs, we are ready to move deeper into Struts. Open up the file `WEB-INF/struts-bean.tld` in your preferred editor and search for `message`. You will find an entry, on or around line 149, which defines the action that you have just used:

```
<bean:message key="index.title"/>
```

The `bean:message` tag can take up to five parameterized arguments, numbered 0 to 4. We can assign a key and a scope to it as well.

It is instructive to have a quick skim through the documentation and see how each is used. You'll find it at the following website: `http://127.0.0.1:8080/struts-documentation/userGuide/struts-bean.html`.

TagUtils

Struts comes with the very useful `org.apache.struts.taglib.Tag-Utils` class. This class contains a range of helper functions that you will find handy when building taglibs.

Exercise 2

What happens when you leave out `locale=true` in a JSP? What if you set `locale=false`?

What happens if a key of a message bean has no corresponding entry in the language file?

Enhance the welcome page. Change the logo and copyright info and write a short multilingual introduction.

How would you implement one more language (for example, Spanish)? How complex would it be?

Could you allow the user to choose the language explicitly, and if so how? For example, could you offer a button on the page?

Think about how you could provide more extensive text in multiple languages, such as a help page? The standard language files do not provide much help here, because they can only handle short texts efficiently. (Hint: Take another look at the first custom action example.)

Why shouldn't you work on the file `WEB-INF/struts-bean.tld`? Think about what would happen if you did add your own settings to that, and then a new version of Struts was released.

Summary

This chapter provided a short introduction to Taglibs and custom actions, and took a first glance at the inner life of Struts. Our shop now has a welcome page, although it's pretty sparse and not so attractive yet!

Learning Jakarta Struts 1.2: A concise and practical tutorial

> *It's a long journey via instructions,*
> *Short and effective via examples.*
>
> Seneca

This book offers step-by-step instructions on Jakarta Struts Framework as well as on building a web application based on Open Source-Software. As the book progresses, we will develop a Web shop: the classic Book Store. This Web application will help us to integrate the individual components of Struts together into complete and extensive software.

The crucial point lies in the practical application and not so much in theoretical background. This book will first demonstrate the process of developing an application followed by a short explanation. You can then experiment with the given code.

What This Book Covers

Each chapter begins with a short introduction and ends with in-depth questions and a short summary. Each chapter builds up on the previous one. Those readers who are already familiar with Struts can skip individual chapters, but they should at least skim through those chapters, so they won't lose the plot and it will also help them to keep up with the web application development part of the book.

Here is an overview for the content of the each chapter:

Chapter 1 presents an overview of Struts in compact form.

Chapter 2 explains the installation for the Struts and shows the first, very easy example. The aim is to start working and experimenting with a fully running system. If you have worked with Struts before, you can skip this chapter.

Chapter 3 introduces the concept the book is based on—the web application called the "Book Store". This will be built, expanded and adapted step-by-step in the following chapters. The underlying design will be introduced in this chapter.

Chapter 4 discusses Internationalization—the possibility to adapt the text of a web site for different languages. Here are some further information and tricks: Custom Actions are explained and a particular Custom Action will be developed.

Chapter 5 emphasizes the importance of correct logging and use of configuration data for software quality. This chapter presents different implementation approaches.

Chapter 6 discusses forms. Handling forms is a standard problem. Error handling in particular generally requires the same mechanisms. Struts offers various possibilities for simplifying this task. Furthermore, this chapter gives insight on how to access databases.

Chapter 7 introduces Logic Tags which allow us to manage conditional generation of a web page via HTML tags. Logical statements such as Loops and If statements are explained in this chapter.

Chapter 8 discusses error handling, the central element of any application. A well-designed shop will always display an error message; after all it's about the customer's convenience.

Chapter 9 discusses the controller and templates. The basic process for the controller in the Model 2-concept is manually implemented and subsequently explained for Struts. Templates (Tiles) allow modular design of the web site and easy alteration.

Chapter 10 is quite an extensive chapter where the elements of the shop are further expanded and merged together as one package. Authorization is implemented, a shopping cart is built and the order process completed.

Chapter 11 offers interesting additional information on Struts independent of our online shop.

Chapter 12 discusses JSTL—frequently called a successor of Struts. Here is a short overview of how these two techniques can be combined together.

Chapter 13 gives useful tools and enhancements for Struts as well as practical solutions for frequently encountered problems.

Appendix A is the Solutions section. Here you will find answers to several exercises. Do yourself a favor and look into this section only after you have solved the problem yourself!

Appendix B contains a Glossary explaining important concepts.

Appendix C contains Literature where you will find some more book tips.

Where to buy this book

You can buy *Learning Jakarta Struts 1.2* from the Packt Publishing website: `http://www.packtpub.com/struts/book`. This book carries at least a 10% discount on the website as well as free shipping to the US, UK, Europe, Australia & New Zealand.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com