The JSP Standard Tag Library is a collection of commonly used functions and tools invaluable to JSP developers to avoid re-creating the same functions on site after site. Sun has indicated that JSP development should be based around using tag libraries going forward, and will release JSP STL, as their official library. This book starts with an in-depth discussion of the JSP STL, then goes beyond the standard library to teach developers to create their own tags to further encapsulate the most common features of their specific applications. Along the way, readers will also learn to use tags to access data, process XML, handle expressions, and further customize pages for international visitors. Later chapters explain how readers can expand the Standard Tab Library by creating their own tags.
Table of Contents

Copyright
About the Author
Acknowledgments
Tell Us What You Think!
Introduction
  The Benefits of JSTL
  JSTL and JSP
  Who This Book Is For
  Structure and Organization of the Book
  The Forum Application
  Source Code and Updates

Chapter 1. Understanding JSP Custom Tags
  The Role of Tags in JSP
  Introducing JSTL
  Installing JSTL

Chapter 2. Programming the JSP Standard Tag Library
  Understanding Scoped Variables
  Accessing Application Data
  The Basics of Web Application Programming
  Summary

Chapter 3. Understanding Basic Tag Logic
  Exception Processing
  Using Conditionals
  A Chat Application
  Summary

Chapter 4. Using the Expression Language
  JSTL Expression Tags
  Using the EL Expression Language
  Using the RT Expression Language
  Summary

Chapter 5. Collections, Loops, and Iterators
  Understanding Collections
  The Iteration Tags
  Using Iterators
  Summary

Chapter 6. Formatting Data with Tags
  Formatting
Copyright

Copyright © 2003 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

Library of Congress Catalog Card Number: 2002106117

Printed in the United States of America

First Printing: September 2002

04 03 02 4 3 2 1

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer
About the Author

Jeff Heaton is a software designer for Reinsurance Group of America, Inc. (RGA). Jeff also teaches Java at St. Louis Community College at Meramec, and is a graduate student at Washington University in St. Louis. A Sun Certified Java Programmer specializing in Internet, socket-level/spidering, and artificial intelligence programming, Jeff is a member of the IEEE. He can be contacted by email at heatonj@heat-on.com or through his Web site at http://www.jeffheaton.com/.
Acknowledgments

First, I'd like to thank everyone at Sams Publishing who made this process productive and fun. Thanks go to my acquisitions editor, Todd Green, for believing in this project and helping to get it started. I'd also like to thank my editor, Elizabeth Welch, for helping to make the text easy to read and follow.

In addition, I want to thank Jenny Yao for providing the translation of our forum application to Chinese. And I am indebted to my friends Lisa Oliver and Marc Goldford for their encouragement during this project. I'd also like to mention Keith Turner for our many discussions of the proper design, format, and implementation of Java and UML. And thanks also go to my agent, Neil Salkind, for helping with this and other publishing projects.

Finally, I'd also like to acknowledge the great group of people I work with at Reinsurance Group of America Inc. (RGA). I've worked with many people at RGA, but in particular I'd like to mention my colleagues working on CybeRe, namely, Don Apperson, Kam Chan, Terri Dickman, Michael Gaffney, Lisa Hackman, Daniel Horn, Cindy Hsu, Gordon Huffman, Bryan Hunt, Larry Jin, Michel Lefebvre, Melinda Long, Scott Middelkamp, Yuko Miyao, Rick Nolle, Mark Phelps, Renu Singh, Lloyd Sloan, Sharon Stuckmeyer, Matt Tackes, John Tucker, Bala Vanukuru, Jonathan Westland, Thomas Wiginton, and anyone else I forgot.
Tell Us What You Think!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an executive editor for Sams Publishing, I welcome your comments. You can email or write me directly to let me know what you did or didn't like about this book as well as what we can do to make our books better.

Please note that I cannot help you with technical problems related to the topic of this book. We do have a User Services group, however, where I will forward specific technical questions related to the book.

When you write, please be sure to include this book's title and author as well as your name, email address, and phone number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@sams publishing.com

Mail:

Michael Stephens
Executive Editor
Sams Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

For more information about this book or another Sams title, visit our Web site at http://www.sams publishing.com/. Type the ISBN (excluding hyphens) or the title of a book in the Search field to find the page you're looking for.
Introduction

JavaServer Pages Standard Tag Library (JSTL) is one of the most significant additions to JSP since the introduction of JSP itself. JSP has traditionally been programmed using scriptlet code—Java source code that is inserted along with the HTML that normally makes up a Web page. JSTL seeks to use tags—similar to those that make up HTML—to replace JSP scriptlet programming. The purpose of this book is to teach you to use JSTL.

First, we must determine exactly what JSTL is. JSTL is a tag library and, as such, is programmed using tags. These tags are similar to the tags you use to program HTML—which is one of the key advantages of JSTL. This similarity is intended is to help alleviate the sharp contrast between the HTML intermingled with JSP scriptlet code. Because JSTL is tag-based, it flows much cleaner with the HTML code.
The Benefits of JSTL

There are several advantages to this tight integration of display and programming tags. First, JSTL is designed so that the Web programmer should be able to learn JSTL quickly and efficiently. JSTL is not just easier for humans to read and understand than JSP scriptlet programming; it is also easier for Web page layout programs to understand. Future Web layout programs will be able to recognize the JSTL tags and insert HTML formatting without disruption of the JSTL tags. Further, Web authoring tools will be able to interact with JSTL tags and perform basic programming.

One perfect example of how JSTL interaction with Web authoring tools can be of particular benefit is in the area of international programming. Traditionally, a multilingual Web application has stored individual text strings in a central location—a database or a data file. The problem is that the method used to display these international text strings is not standardized from one application to the next. Most Web applications use some sort of JSP scriptlet programming to retrieve and then display multilingual text. Unfortunately, Web authoring tools are completely oblivious to these different methods of international programming. JSTL provides us with a concise way to implement a multilingual application. Now it's possible to design Web authoring tools that are aware of the multilingual JSP tags and allows the native language of the page to be displayed just as easily as the foreign languages that the page will be translated into. This enables programmers designing multilingual Web sites to visually edit their Web pages in a variety of languages.
JSTL and JSP

JSTL is designed to work with JSP version 1.2 and the new JSP version 2.0. JSP 2.0 implements some very significant advantages over previous versions of JSP. One of the most important additions to JSP version 2.0 is the expression language, called EL, which is the way that Sun intends future expressions to be represented in JSP programming. JSTL supports this new expression language. For added flexibility, JSTL also supports the expression language typically used in JSP scriptlet programming in versions prior to JSP version 2.0. The EL expression language supports many advanced features.

The EL expression language allows you to access scoped variables. These variables are the primary means by which JSTL stores both temporary and sessionwide data. PCL expressions can be passed to most of the tags provided by the JSTL tag library.
Who This Book Is For

This book assumes that you have some experience as a JSP programmer. An in-depth knowledge of JSP is not required, but some familiarity with HTML forms, hyperlinks, URLs, and server-side programming is assumed. JSTL includes a set of SQL tags that allow the programmer to access a relational database. This book does not attempt to teach SQL, but rather shows you how to use JSTL to integrate with SQL.
Structure and Organization of the Book

This book is designed to be used both as a reference and as a tutorial for the JSTL tag library. The first five chapters discuss the core features of JSTL. If you're not familiar with JSTL, we recommend that you read Chapters 1 through 5 before reading the rest of the book. Chapters 1 through 3 explain core programming techniques that you will need for JSTL programming. Chapter 4 takes an in-depth look at the expression language supported by JSTL. Chapter 5 describes loops and iteration, the core feature of any programming language.

You can read the remaining chapters in any order; they describe individual tags that add specific features to JSTL. Chapter 6 describes how to format data. Chapter 7 examines how to access data with SQL. XML processing is covered in Chapter 8. Chapter 9 discusses Internet programming. Chapter 10 completes the coverage of JSTL and describes how to create an international program.

The final two chapters discuss topics beyond JSTL. Chapter 11 shows how you can create your own tag libraries compatible with JSTL. Chapter 12 demonstrates how to package and deploy your tag-based Web applications.
The Forum Application

One significant feature of this book is the forum application. This application is a Web-based bulletin board that allows users to register and then post messages to any of several forms. Throughout this book, the forum application is extended as you learn new features of JSTL.

We introduce the forum application in Chapter 7, where we implement it as a simple JSP Web application that uses the SQL tags. In the first version, the forum application is an English-only program. Chapter 10 extends this by making the application support additional languages.

The forum application now contains quite a few JSTL tags. JSTL is not meant to be the only technology you use to create a complex Web application; you should use your own custom tag libraries, which you create, to encapsulate your Web application's functionality. Chapter 11 shows you how to create a tag library designed to coexist with JSTL. This allows the forum application tag library to share data with JSTL tags.

At this point, you have a complete Web application that reflects the way in which JSTL should be used in conjunction with your own custom tag libraries. Chapter 12 then completes this process by showing you how to deploy this Web application.
Source Code and Updates

For updates to this book, and to download the source code and examples presented here, visit http://www.samspublishing.com/. From the home page, type this book's ISBN (0672324504) into the search window, and click Search to access information about the book and a direct link to the source code.
Chapter 1. Understanding JSP Custom Tags

JavaServer Pages Standard Tag Library (JSTL) is a standard tag library that can be used in JavaServer Pages (JSP). Sun introduced this exciting new technology to allow tag-based programming to exist more naturally with the already tag-based HTML. Because of this, Sun anticipates that JSTL will be an easier programming environment than the traditional scriptlet-based JSP programming. To understand JSTL, you must first understand what a tag library is.

Sun introduced JSTL as part of its Java Community Process (http://www.jcp.org/). Most Java Community Process technologies begin as a Java Specification Request (JSR). JSTL began as JSR-52 and JSR-152. JSTL has since been released as JSTL 1.0.

Tag libraries give the Java programmer the ability to write Java objects that will be executed as tags within the JSP code. Custom tags are interpreted by the Web server (or other servlet container) before they are transmitted to the Web browser. Tag libraries can output data of their own, as well as control the flow of HTML and scriptlet code in the JSP page. This allows content to be generated dynamically. Rather than always presenting the same content to the user, a JSP page dynamically creates pages to provide a more interactive experience for the user.

After tag libraries were introduced, many vendors began creating custom tag libraries that could be used with JSP. Many common uses for custom tags were discovered, such as data validation, internationalization, and program flow control. Each vendor implemented these "standard" functions differently.

JSTL is one of several programming methods for creating Web applications using Java. In addition to JSTL, the programmer can create custom libraries with custom tags and extend HTML.
The Role of Tags in JSP

JSP files that use scriptlet code are a mix of HTML tags and Java source code. One of the reasons JSTL and other custom tag libraries were introduced was to provide more consistency in JSP files. HTML is tag-based, whereas the scriptlet code appears as regular Java code. Tag-based programming allows the JSP page to remain completely tag-based. Further, these custom tag libraries have the ability to hide much of the Java code behind tags that can be inserted directly into the HTML. The result of using JSTL in JSP code is cleaner files that are much easier to modify and extend.

To illustrate the difference between the various programming technologies, let's look at the implementation of a simple example—the process of counting from 1 to 10—in each of these technologies. We begin with an HTML implementation.

Using HTML

HTML is static and cannot change. The look of a page is predefined before the user ever sees it. **Listing 1.1** shows how you would count to 10 using just a regular HTML page. Here, you can see that counting to 10 is nothing more than formatting the numbers 1 through 10.

**Listing 1.1 HTML Counts to 10**

```
<html>
<head>
    <title>Count to 10 Example(using HTML)</title>
</head>
<body>
    1
    2
    3
    4
    5
</body>
```
Introducing JSTL

We now examine the actual structure of JSTL and how to use it. We begin by examining the Apache Taglibs project, which includes the reference implementation of JSTL. Then, we examine the components of JSTL itself.

The Apache Taglibs Project

The implementation of JSTL that we use in this book is a part of the Apache Software Foundation's Jakarta Taglibs project. The Taglibs project includes many tag libraries other than JSTL. All of these tag libraries can be used in JSP pages. You can find more information about the Taglibs project at http://jakarta.apache.org/taglibs/. You can use the tag libraries provided by the Taglibs project to add functionality to your JSP pages. However, keep in mind that some of the tag libraries available in the Taglibs project add features that are not currently supported by JSTL.

The Structure of JSTL

JSTL itself is made up of four smaller tag libraries. These tag libraries allow you to choose which components of JSTL you will use in your program. Table 1.1 summarizes the four components of JSTL.

Table 1.1. The Four Components of JSTL

<table>
<thead>
<tr>
<th>Component</th>
<th>URI</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td><a href="http://java.sun.com/jstl/core">http://java.sun.com/jstl/core</a></td>
<td>c</td>
</tr>
<tr>
<td>XML Processing</td>
<td><a href="http://java.sun.com/jstl/xml">http://java.sun.com/jstl/xml</a></td>
<td>x</td>
</tr>
<tr>
<td>I18N Formatting</td>
<td><a href="http://java.sun.com/jstl/fmt">http://java.sun.com/jstl/fmt</a></td>
<td>fmt</td>
</tr>
</tbody>
</table>
Installing JSTL

Before you can use the JSTL components, you must have a JSP 1.2-compliant Web server installed, such as Tomcat 4. Installation on other JSP Web servers should be similar. The source code we present in this book should work with any JSP Web server.

If you are not familiar with the installation of Apache Tomcat, you can refer to Appendix B, "Installing Tomcat and JSTL." In this appendix, we tell you how to acquire and install both Tomcat and JSTL.

Adding JSTL to Your Web Server

You must download and install JSTL into your Web server if you want to use the JSTL tags. The process for installing JSTL is similar to the process for installing any other tag library.

You can download JSTL from the Apache Taglibs Project site at http://jakarta.apache.org/taglibs/. From the Apache site, choose to download the Standard Tag Library. Again, the complete process for installing JSTL is covered in Appendix B.

Using JSTL in Your JSP Pages

After you have properly installed JSTL into your Web server, you can use it in any of your JSP pages. However, the JSTL libraries do not just become automatically available to each of your JSP pages. Any JSP pages that need to make use of one of the JSTL tag libraries must include an appropriate taglib directive at the top of the JSP page. As we've pointed out, the taglib directive used to access the Core taglib is as follows:

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

With this directive included near the top, this JSP page is now ready to use JSTL taglibs. It is also necessary to make the correct additions to the web.xml file and ensure that JSTL is properly installed, as we explain in Appendix B.

The following is a typical JSTL tag:

```jsp
<c:forEach var="i" begin="1" end="10" step="1">
```

Chapter 2. Programming the JSP Standard Tag Library

Every programming language has many fundamental constructs. One of the basic constructs inherent in every programming language is variables. Programming languages must provide variables to store and process data. This chapter begins by showing you how variables are stored and processed in JSTL.

Programs must also interact with the user. When a JSP page first loads all user input to that page, it sends the input as variables. In this chapter, we show you how to access and process this data. This will allow you to create HTML forms that are capable of sending their data to JSTL-based JSP pages.

Form processing is only one aspect of Web application development. Web application programmers must also manage state and provide continuity between their pages. Most Web programming environments accomplish this through the use of some sort of scoped variables. Scope refers to both how long a variable remains active and what parts of a program have access to the variable. Scope allows you to define certain variables that are active only for the current page, others that remain active while the user is logged in, and still other variables that are accessible to all users.

Scope is an inherent concept to JSTL variables; in fact, variables in JSTL are called scoped variables. As you learn in the next section, there are four levels to JSTL scope.

We conclude this chapter by showing you how to accomplish common Web application tasks with JSTL. All these actions are facilitated by variables.
Understanding Scoped Variables

Most JSTL tags operate on scoped variables. You create JSTL variables by using the set command. A typical set command is shown here:

\[
\text{Listing 2.1: The Main Scoped Sample Index (index.jsp)}
\]

\[
\begin{equation}
<	ext{c:set var="variableName" value="the value"/>}
\end{equation}
\]

This command assigns the value the value to the scoped variable, which is named variableName . If you are familiar with JSP scriptlet programming, you may be interested to know that the scoped variables in JSTL are stored by the same mechanism as JSP scriptlet scoped variables. Just as in JSP scriptlet programming, these variables are stored using the javax.servlet.jsp.PageContext class. Because of this, JSTL scoped variables are accessible to JSP using the PageContext class.

Because JSTL variables are called scoped variables, the concept of scope must enter the picture at some point. The programming term scope refers to the scope for which a variable is valid. In Java, the scope of an automatic, or local, variable is only within the function that initializes the variable. In the following code snippet, the variable i’s scope is only within the mymethod() method:

\[
\text{Listing 2.2: The myMethod Method}
\]

\[
\begin{equation}
\text{void mymethod()}
\end{equation}
\]

\[
\begin{equation}
\text{
int i;
for(i=0;i<10;i++)
{
System.out.println("Count:"+i);
}
}
\end{equation}
\]

However, JSTL does not use methods. The scope of a JSTL variable cannot be determined from its context in the same way a Java variable can. Therefore, you must specify the scope of every variable you create. If you fail to specify the scope for a variable, that variable's scope will default to page-level scope. The following tag will create a variable named variableName for session-level scope:

\[
\text{Listing 2.3: The scoped Variable Example (index.jsp)}
\]

\[
\begin{equation}
<	ext{c:set var="variableName" value="the value" scope="session"/>}
\end{equation}
\]

JSTL and scriptlet code support four different levels of scope—the same four levels of scope that JSP provides. These four levels are page scope, request scope, session scope, and application scope. Table 2.1 summarizes these scopes.

### Table 2.1. Variable Scopes

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Scope</td>
<td>A variable with this scope will only be available within a single page.</td>
</tr>
<tr>
<td>Request Scope</td>
<td>A variable with this scope will only be available within a single request.</td>
</tr>
<tr>
<td>Session Scope</td>
<td>A variable with this scope will only be available within a single session.</td>
</tr>
<tr>
<td>Application Scope</td>
<td>A variable with this scope will be available across all requests.</td>
</tr>
</tbody>
</table>
Accessing Application Data

Web applications must interact with the user, and to interact with the user, you must be able to access JSTL application data. Application data falls into two categories in JSTL. JSP makes two implicit variables available for your use.

The first category is page context data. Page context data lets you know general information about the request, such as the request method or how long the session has been running.

The second category, a collection named params, is made available to read the content posted from forms. Responding to parameter data is often a primary means of interaction between the Web application and the user.

Accessing Page Context Data

JSP includes a variable named pageContext that allows you to access certain contextual information about the page. To see what sort of information is available through the pageContext variable, let’s look at the example shown in Listing 2.4. This simple JSP page will list the important items stored in the page context.

Listing 2.4 Accessing Application Data

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

<html>
<head>
  <title>Page Data Example</title>
</head>

<body>
  <h3>&nbsp;</h3>
  <table border="1" width="539">
    <tr>
      <td colspan="2" width="529" bgcolor="#0000FF">
        <b>HTTP Request (pageContext.request.)</b>
      </td>
    </tr>
  </table>
</body>
```
The Basics of Web Application Programming

Now that you have a good understanding of JSTL variables and application data, you can start to create some simple Web applications. This section provides examples that illustrate common Web programming techniques.

Processing the Query String

The query string has always been a popular conduit for information between Web pages. The query string allows variable information to be embedded directly into a URL. For example, the URL http://www.jeffheaton.com/search.jsp?n=5 would pass the value 5 through the n variable to the script search.jsp. Anything containing a ? that follows the URL is considered part of the query string.

By separating the values with ampersands ( & ), you can send more than one value through the query string. To pass the value 10 for x and 20 for y, you would use the URL http://www.jeffheaton.com/search.jsp0&y=20.

These values can easily be accessed by the JSTL code. You will recall from the section "Accessing Application Data" earlier in this chapter that JSTL provides a variable called paramValues that allows you to access parameter arrays. There is also a variable called param that allows access to individual form and query string items. Note that neither param nor parmValues makes any distinction between form items and query string items. You would use the following tag to display a form or query string item named x:

```<c:out value="${param.x}"/>
```

Listing 2.7 shows a program that uses this technique. This simple example asks users what their favorite color is.

**Listing 2.7 Favorite Color (color.jsp)**

```<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
    <head>
        <title>Query String Example</title>
    </head>

    <body>Your favorite color is:
    <b>
        <c:out value="${param.color}"/>
    </b>
</body>
```
Summary

This chapter introduced you to the basics of scoped variables and how they are used with JSTL. In the next chapter, we show you how to use basic tag logic to perform if statements and other branching operations.
Chapter 3. Understanding Basic Tag Logic

The Core tag library of JSTL provides many rudimentary tags for performing basic flow control. Flow control refers to a program's ability to selectively execute portions of itself. Most programming languages achieve this flow control through if statements and other logical tags. A program's flow of execution can also be affected by errors.

Error handling is an important part of any computer program. JSP provides tags that let you perform basic error handling. By making use of JSTL's built-in exception-handling capabilities, you can create programs that are capable of handling common runtime errors.

In this chapter, we present a sample application that shows how to tie all these concepts together. A simple forms-based chat application will show you how to use scope and basic tag logic to allow two users to chat. Let's begin by examining JSTL's error-handling capabilities.
Exception Processing

Java programs handle errors through the use of exceptions. When a section of code is susceptible to throwing an exception, you can enclose that section of code in a try block. For example, when you're creating a new URL object, it is possible that an invalid URL will be provided to the object. If this happens, the URL object will throw a MalformedURLException. To properly handle this exception, the URL instantiation must be enclosed in a try block, and a catch must be provided that will catch the MalformedURLException. The following code illustrates this:

```java
try {
    URL = new URL("http://www.sams.com");
}
catch(MalformedURLException e) {
}
```

Exception handling allows your program to react to errors that are not a result of bad program design. For example, exceptions could be trapped when the user enters invalid information, a URL fails to load, or when some other noncritical error occurs. By trapping these exceptions, you design your program to handle these errors as they happen so they don't result in a server error. This ensures that your Web pages present a consistent and robust interface.

JSTL provides the `<c:catch>` tag to handle this case. Before we examine how to catch an exception in JSTL, let's see what causes an exception to be thrown.

Generating Exceptions

Many of the JSTL tags are capable of throwing exceptions, yet JSTL is designed to throw exceptions as little as possible. Many errors that would generate an exception in Java or other programming languages are ignored in JSTL. Consider division by zero. Nearly any programming language ever produced will return some sort of an error if a division is specified with zero in the denominator. Yet in JSTL, this case does not generate an exception. Listing 3.1 demonstrates this.

Listing 3.1 Division by Zero (exception.jsp)

```html
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
    <head>
        <title>Throw an Exception</title>
    </head>
    <body>
```

...
Using Conditionals

Conditional statements are an important part of any programming language. These statements allow you to selectively execute parts of the program based on certain conditions. JSTL provides several tags that allow this sort of conditional execution.

JSTL's implementation of conditionals is somewhat different from Java's. In Java, your two primary conditional constructs are the if/else statement and the switch/case statement. The `<c:if>` and `<c:choose>` tags provide many of the same features, yet there are some important differences compared to their Java counterparts. We highlight these differences in this section.

In previous chapters, we have seen how the `<c:if>` tag can be used to make basic decisions. We now take a more in-depth look at the `<c:if>` tag and other conditional tags provided by JSTL. Let's begin by examining the `<c:if>` tag.

Using the `<c:if>` Tag

Nearly every programming language contains an if statement, and JSTL is no exception. The JSTL if statement is implemented using the `<c:if>` tag. There are two forms of the JSTL if statement:

```
<!-- Syntax 1: Without body content -->
<c:if test="testCondition"
    var="varName"
    [scope="{page|request|session|application}"]/>
```

```
<!-- Syntax 2: With body content -->
<c:if test="testCondition"
    var="varName"
    [scope="{page|request|session|application}"]>
    body content
</c:if>
```

The attributes that are accepted by the `<c:if>` tag are as follows:
A Chat Application

The last example that we examine in this chapter is a simple chat application. This program shows how more than one browser session can be made to communicate. We use a simple example of a chat application in order to demonstrate some of the techniques shown in this chapter. Chapter 5, "Collections, Loops, and Iterators," will extend this simple chat program into a more complex example.

When you first access the chat page, you will be prompted to enter your login identity. This is the name by which other users will know you. This login can be nearly anything you like. Once you enter your name, you will be taken to the main chat page. (See Figures 3.2 and 3.3.)

Figure 3.2. Our first user chatting.

Figure 3.3. Our second user chatting.
Summary

In earlier chapters, you have seen expressions used in conjunction with the techniques being introduced. The next chapter will focus on expressions and show you some of the advanced features they provide.
Chapter 4. Using the Expression Language

Programming languages use expressions to manipulate application data. Expressions resemble algebraic formulas in that they often specify mathematical transformations involving the program's variables. For example, the expression a+10 tells the program to evaluate the expression 10 added to a. This does not change the value of a; rather, the computer simply computes this sum and then performs some action based on that sum.

JSTL supports expressions through the use of two expression languages. JSTL includes support for scriptlet-based expressions, called rtexprvalue expressions. JSTL also introduces Expression Language (EL), which greatly simplifies expression handling in JSTL. The designers of JSTL assume that most programmers will use EL expressions and use rtexprvalues for more advanced purposes. This chapter will focus primarily on EL expressions, although we cover expressions using rtexprvalues as well. We begin by examining the tags that JSTL makes available for expression handling.
JSTL Expression Tags

In JSTL, an expression is always wrapped between the characters ${ and }. For example, the expression for displaying the variable a with 10 added would be ${a+10}. This can easily be incorporated into many of the JSTL tags. The following code would use a <c:out> tag to display the expression a+10:

```<c:out value="${a+10}"/>
```

Eventually, JSP 2.0 will allow EL expressions to be inserted right into HTML code. For example, the following line in a JSP file would display the expression a+10:

```<p>The value of a+10 is ${a+10}</p>
```

While the expression language is a powerful feature of JSP 2.0, one of the design requirements for JSTL was compatibility with JSP 1.2. To accommodate JSP 1.2's lack of expression support, Sun added the <c:out> tag to provide a means of displaying a JSTL expression.

The <c:out> tag is not the only tag that JSTL provides that can work with expressions. In addition to the <c:out> tag, JSTL includes several tags that are designed to deal directly with expressions. In the following sections, we explain how to use each of these tags.

Using the <c:out> Tag

The <c:out> tag is used to display a JSTL expression and set the value of a JSTL scoped variable. There are two forms of the <c:out> tag:

- **Syntax 1: Without a body**
  ```
  <c:out value="value" [escapeXml="{true|false}"]
  [default="defaultValue"] />
  ```
- **Syntax 2: With a body**
  ```
  <c:out value="value" [escapeXml="{true|false}"]
  default value
  </c:out>
  ```

If you want to display a small amount of text, you should use the bodyless version of the <c:out> tag. This will cause your code to appear more concise because your text will be included in the tag itself. The other version of the <c:out> tag includes a body in order to allow a larger area of text to be displayed. This area can include multiple blank lines, text, and expressions.
Using the EL Expression Language

EL provides two methods for using expressions. We discuss the first method, the EL expression language, next, and cover the RT expression language in the section that follows.

The designers of JSTL assumed that most developers will use the EL expression language. An expression using EL must always be delimited by ${ and }.

Accessing Properties

To access the properties of objects with EL, you should use either the . or [] operator. The . operator is used when you know ahead of time exactly what property you want to access. Listing 4.7 shows both techniques.

Listing 4.7 Accessing Properties (prop.jsp)

```html
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
<head>
<title>Property Access</title>
</head>

<body>
<c:if test="${pageContext.request.method=='POST'}">
<c:set var="idx" value="name" />

param.name =
<c:out value="${param.name}" />

<br />

param[name] =
<c:out value="${param[idx]}" />

<br />
</c:if>

<br />
<form method="post">Please enter your name?
<input type="text" name="name" />

<input type="Submit" />
</form>
</body>
</html>
```
Using the RT Expression Language

JSTL also includes support for scriptlet type expressions through the RT expression language. These expressions are bounded between the <%= and %> characters. By using the RT expression language, you can access familiar constructs from scriptlet programming.

JSTL implements support for two scripting languages through the use of twin tag libraries. To access the EL version of the if tag, the tag <c:if> is used. The RT version of the if tag can be accessed by using the <c-rt:if> tag. The four components of JSTL are each duplicated to support the RT expression language. Table 4.3 shows four components that can use RT with JSTL. This is not defined in the JSTL 1.0 spec, but it is in the JSP 1.2 spec.

Table 4.3. The Four Components of RT

<table>
<thead>
<tr>
<th>Component</th>
<th>URI</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td><a href="http://java.sun.com/jstl/core-rt">http://java.sun.com/jstl/core-rt</a></td>
<td>c-rt</td>
</tr>
<tr>
<td>XML Processing</td>
<td><a href="http://java.sun.com/jstl/xml-rt">http://java.sun.com/jstl/xml-rt</a></td>
<td>x-rt</td>
</tr>
<tr>
<td>I18N Formatting</td>
<td><a href="http://java.sun.com/jstl/fmt-rt">http://java.sun.com/jstl/fmt-rt</a></td>
<td>fmt-rt</td>
</tr>
<tr>
<td>Relational DB Access (SQL)</td>
<td><a href="http://java.sun.com/jstl/sql-rt">http://java.sun.com/jstl/sql-rt</a></td>
<td>sql-rt</td>
</tr>
</tbody>
</table>

For example, to take advantage of the core RT tags, you would use the following command:
<%@ taglib uri="http://java.sun.com/jstl/core-rt" prefix="c-rt" %>

WARNING

Of course, you must also make sure that your web.xml file has entries for the RT libraries if you want to use them. Appendix B covers this.

The RT expression language allows you to use expressions exactly as you did in JSP scriptlet programming. For example, the following <c-rt:if> tag checks the request method:
<c-rt:if test="<%=request.getMethod().equals("POST" ) %>"/>
Summary

This chapter showed you how to use the expression language. The expression language allows you to modify scoped variables as it suits your program. JSTL can make use of both the new EL (to be included with JSP 1.3) and the older scriptlet-based expressions. The next chapter will explain how to use iterators so that you can work with larger sets of scoped variables.
Chapter 5. Collections, Loops, and Iterators

Collections have always been an important part of JSP programming. JSP pages often receive collections of objects, such as rows from a database that must be displayed to the HTML user. JSTL provides several tags that enable the JSP programmer to work with collections of objects. These tags allow a more consistent approach to handling collections than scriptlet JSP programming provides.

In addition to the ability to iterate over collections of objects, JSTL provides basic parsing into the iteration tags. Token iteration tags allow the programmer to iterate through strings that contain items separated by designated tokens. This allows a string to be tokenized and the results processed in a way that is appropriate for the page.

This chapter will introduce you to both iteration and tokenization. We begin by taking a closer look at the collections that can be used with these tags.
Understanding Collections

The term collection has a slightly different meaning in JSTL than it does in Java. In JSTL many different types of objects can function as a collection. In particular, the following types of classes often form collections in JSTL:

- Any class that implements the `java.util.Collection` interface
- Arrays of objects or primitive data types
- Any class that implements the `java.util.Iterator` interface
- Any class that implements the `java.util Enumeration` interface
- Any class that implements the `java.util.Map` interface
- Comma-delineated strings (`java.lang.String`)

All classes that are members of the Java collections API can be used with the JSTL iteration tags. These include `LinkedList`, `ArrayList`, `Vector`, `Stack`, `Set`, and `Map`. Additionally, string items that contain comma-delineated text can be used. Data is often collected using these collection types, and understanding how to access collections using JSTL is important.
The Iteration Tags

JSTL supports two tags that are used to iterate over collections. For iteration over the collections just discussed, you use the `<c:forEach>` tag. As we stated earlier, the `<c:forEach>` tag can iterate over comma-delineated strings. It is also possible to iterate over strings that are delineated by characters other than the comma. To iterate over such strings, you must use the `<c:forTokens>` tag. In this section, we discuss the `<c:forEach>` tag. In the next section, we will discuss the `<c:forTokens>` tag.

**The `<c:forEach>` Tag**

The `<c:forEach>` tag is used to iterate over a collection or to iterate a fixed number of times. There are two forms of the `<c:forEach>` tag:

```
// Syntax 1: Iterate over a collection of objects
<c:forEach var="varName" items="collection" [varStatus="varStatusName"] [begin="begin"] [end="end"] [step="step"]>
  body content
</c:forEach>
// Syntax 2: Iterate a fixed number of times
<c:forEach var="varName" [varStatus="varStatusName"]
  begin="begin" end="end" [step="step"]>
  body content
</c:forEach>
```

In syntax 1, you must provide the `<c:forEach>` tag with a collection to iterate over. You specify this collection by using the `items` attribute. Each item in the collection will be referenced by the variable specified in the `var` attribute of the `<c:forEach>` tag. This loop will continue until all of the items in the collection have been processed.

Syntax 2 does not rely on collections to set the boundaries that it will loop through. This example specifies a beginning, ending, and count for the loop. The attribute `begin` specifies where the loop should begin. The attribute `end` specifies where the loop should end. This example specifies a step, which is not required.
Using Iterators

Now that you have seen the `<c:forEach>` tag, let's look at a few examples of iterators. There are many common uses for iterators in JSP pages, and the following sections examine the most useful ones.

**Iterating over Strings**

As we mentioned earlier, the iterator tags can iterate over strings if the strings are comma delimited. **Listing 5.1** shows a program that iterates over a string of comma-separated values.

**Listing 5.1 Iteration and Strings (string.jsp)**

```html
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
<head>
    <title>String Collection Examples</title>
</head>
<body>
    <h3>String Collection Example</h3>
    <c:set var="str" value="Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday" />
    <b>Input String:</b>
    <c:out value="${str}" />
    
    <b>Iterating:</b>
    <c:forEach var="day" items="${str}">
        <c:out value="${day}" />
    </c:forEach>
</body>
</html>
```
Summary

In this chapter, we showed you how to use iterators. Iteration is one of the most important concepts in Web programming. As you progress through the remaining chapters in this book, you will use iterations to process other types of data, such as the results from an SQL database.

This chapter also showed that you can use iterators to perform basic string processing. In the next chapter, you will learn how to properly format data for display using JSTL’s formatting tag library.
Chapter 6. Formatting Data with Tags

JSTL provides several tags that greatly simplify the formatting and parsing of data. Tags are provided that allow you to format numbers, percents, dates, and currencies. By using these tags, you can fine-tune the output of your data, without the need for customized display programming.

Web applications must frequently process data entered by a user. Data from a user always arrives as a string. However, this string is often a number, date, percentage, or other data type. JSTL provides parsing tags that enable you to take the strings a user enters and parse them into other data types.

In this chapter, we examine both of these issues. We provide several examples that illustrate practical application of data formatting and parsing. We begin by examining the process of data formatting.
Formatting

Formatting and parsing, as defined by JSTL, are two related but quite different actions. JSTL considers formatting to be the process of changing the appearance of stored data. This data has already been validated to be correct, and most likely should not produce an error.

JSTL provides two tags for data formatting. The <fmt:formatNumber> tag is used to format numbers; the <fmt:formatDate> tag is used to format dates and times.

Using the <fmt:formatNumber> Tag

The <fmt:formatNumber> tag is used to format numbers, percentages, and currencies. There are two forms of this tag:

// Syntax 1: Without a body
<fmt:formatNumber value="numericValue"
    [type="{number|currency|percent}"]
    [pattern="customPattern"]
    [currencyCode="currencyCode"]
    [currencySymbol="currencySymbol"]
    [groupingUsed="{true|false}"]
    [maxIntegerDigits="maxIntegerDigits"]
    [minIntegerDigits="minIntegerDigits"]
    [maxFractionDigits="maxFractionDigits"]
    [minFractionDigits="minFractionDigits"]
    [var="varName"]
    [scope="{page|request|session|application}"]
    />

// Syntax 2: With a body to specify the numeric value to be formatted
<fmt:formatNumber [type="{number|currency|percent}"]
    [pattern="customPattern"]
    [currencyCode="currencyCode"]
    [currencySymbol="currencySymbol"]
    [groupingUsed="{true|false}"]
    [maxIntegerDigits="maxIntegerDigits"]
    [minIntegerDigits="minIntegerDigits"]
    [maxFractionDigits="maxFractionDigits"]
    [minFractionDigits="minFractionDigits"]
    [var="varName"]
    [scope="{page|request|session|application}"]
    >
    numeric value to be formatted
    </fmt:formatNumber>

The attributes accepted by the <fmt:formatNumber> tag include the following:
Parsing

In addition to the formatting tags provided by JSTL, a second set of tags is available for parsing. In JSTL, parsing refers to tags that are used to access data entered by a user. As a result, it is quite possible for errors to occur if the data that is to be parsed is not in the format that the JSTL tag expects. In this section, we show you how to use the JSTL parsing tags.

Using the <fmt:parseNumber> Tag

The <fmt:parseNumber> tag is used to parse numbers, percentages, and currencies. There are two forms of this tag:

```java
// Syntax 1: Without a body
<fmt:parseNumber value="numericValue"
    [type="{number|currency|percent}"]
    [pattern="customPattern"]
    [parseLocale="parseLocale"]
    [integerOnly="{true|false}"]
    [var="varName"]
    [scope="{page|request|session|application}"/>
// Syntax 2: With a body to specify the numeric value to be parsed
<fmt:parseNumber [type="{number|currency|percent}"]
    [pattern="customPattern"]
    [parseLocale="parseLocale"]
    [integerOnly="{true|false}"]
    [var="varName"]
    [scope="{page|request|session|application}"">
numeric value to be parsed
</fmt:parseNumber>
```

The attributes accepted by the <fmt:parseNumber> tag include the following:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>integerOnly</td>
<td>N</td>
<td>Specifies whether just the integer portion of the given value should be parsed. This attribute defaults to true.</td>
</tr>
<tr>
<td>parseLocale</td>
<td>N</td>
<td>Specifies the locale whose default formatting pattern (for dates and numbers) should be used.</td>
</tr>
</tbody>
</table>
Time Zones

Nearly every Web application must be concerned with time zones. It is very likely that the time zone that your Web server is in is different from the time zone of the user accessing your Web site. JSTL gives you several tags that allow you to set the time zones that date formatting tags deal with.

Setting Time Zones

There are two ways that you can set the time zone in a JSTL-based application. The first is by modifying the web.xml file for the Web site so that it specifies a default time zone. The second is to use the <fmt:timeZone> tag.

Using the <fmt:timeZone> Tag

It is also possible to set the time zone for a specific range of tags. The <fmt:timeZone> will specify the time zone that all tags within its body will use. This can allow you to customize the time zone on a per-user basis. Here is the format of the <fmt:timeZone> tag:

```
<fmt:timeZone value="timeZone"
  [var="varName"]
body content
</fmt:timeZone>
```

The <fmt:timeZone> tag accepts the following attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Y</td>
<td>Specifies the time zone that should be used by the tags enclosed in the body of this tag.</td>
</tr>
</tbody>
</table>

The body content of the <fmt:timeZone> tag specifies the tags that will be controlled by this time zone. For example, the following tags would use the U.S. Central time zone:

```
<fmt:timeZone value="CDT">
  <t:formatDate value="${now}" type="both" />
</fmt:timeZone>
```
Applying Date Formatting

Now that you know how to format dates and times in various ways, let's see how to apply these tags. **Listing 6.7** shows a program that displays a calendar.

**Listing 6.7 Displaying a Calendar (calendar.jsp)**

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

<html>
<body>
<form method="POST">
	<table border="1" cellpadding="0" cellspacing="0"
	style="border-collapse: collapse" bordercolor="#111111"
	width="62%" id="AutoNumber1">
	<tr>
		<td width="100%" colspan="2"bgcolor="#0000FF">
		<p align="center">
		</p>
		</td>
	</tr>
	<tr>
		<td width="47%">Enter a month (1-12)</td>
		<td width="53%">
		<input type="text" name="month" size="20" />
		</td>
	</tr>
	<tr>
		<td width="47%">Enter a year (i.e. 2002)</td>
		<td width="53%">
		<input type="text" name="year" size="20" />
		</td>
	</tr>
	<tr>
		<td width="100%" colspan="2">
		<p align="center">
		<input type="submit" value="Submit" name="submit" />
		</p>
		</td>
	</tr>
</table>
</form>
</body>
</html>
```
Summary

In this chapter, we saw how you can format and parse data by using JSTL tags. Up to this point, the only source of data that we have examined has been data entered by the user. In subsequent chapters, we examine data from other sources. In the next three chapters, we show you how to obtain data through XML, SQL, and the Internet.
Chapter 7. Accessing Data with SQL Tags

Data access is an important feature of any Web application. Most Web applications are driven by SQL-compatible databases. JSTL provides a complete set of tags that allows access to SQL databases.

You may be wondering why such tags are needed. Most multitier Web applications are designed to keep database access out of JSP pages completely. This allows JSP pages to concentrate on the presentation interface while a collection of Enterprise JavaBeans (EJB) handles the backend database access. The JSTL SQL tags are meant to give you a way to incorporate SQL access into smaller projects or prototypes that do not need the complexity introduced by a multitier system.

We begin by examining how to create a Web application using only JSTL tags. Chapter 11, "Creating Your Own Tag Libraries," will follow up by showing you how to move this functionality into your own programs.

This chapter will present a complete Web application that was written entirely in JSTL. The primary examples for the next several chapters in this book will build on this Web application. By the end of Chapter 12, you will have developed a multilingual JSTL database application that does all backend processing through a custom tag library. This is the model that Sun is promoting with JSP 1.3.
Introducing the Forum Example

Most Internet users are familiar with online forums. These systems go by many names—message boards, discussion threads, newsgroups, bulletin boards. Online forums allow users to post messages to an area so the messages can be viewed by other users. Users can post messages as well as responses to other people's messages. Figure 7.1 shows our forum application.

Figure 7.1. Our forum application.

As you can see, this Web application allows users to post messages. To accomplish this, the application requires several interrelated services:

* User security
* New user registration
Understanding JDBC Drivers

The SQL tags provided by JSTL make use of Java Database Connectivity (JDBC) drivers. Most commonly used databases provide a JDBC driver.

Sun also makes available a bridge driver that allows Java programs to make use of ODBC databases. Usually, you should try to find a suitable JDBC driver before you use the ODBC bridge driver. However, if no such driver exists, you may access the database using the ODBC bridge.

Connecting to a JDBC Data Source

To use the JSTL SQL tags, you have to identify a data source. Each of the JSTL SQL tags accepts a dataSource attribute that lets the SQL tag know what data source to use. Once the data source is specified, the JSTL tags will collaborate with that source to access the data requested.

JSTL provides three primary ways for setting up this collaboration. Let's take a look at each method.

Transparent Collaboration

For this approach, you must provide initialization code in the application logic of a server or other related Java code. For example, you could do this by using the application event listener of a server. The servlet must then store the application's default DataSource object in the javax.servlet.jsp.jstl.sql.DataSource application or session-scoped variable. This approach is advantageous because it makes the selection of the data source completely transparent to the page programmer. The JSP programmer does not need to specify a dataSource attribute to any of the JSTL tags being used, as shown here:

```html
<sql:query ...>
```
Using SQL Tags

So far, we have examined the <sql:setDataSource> tag. JSTL makes available many other tags to facilitate the programming of SQL data sources. Let's examine each of these tags, beginning with the <sql:query> tag.

Using the <sql:query> Tag

The <sql:query> tag is used to perform a query of the database. A query is a SQL command that causes data to return. Not all SQL commands function in this way. The SELECT SQL command is most commonly used with the <sql:query> tag. If you are going to execute a SQL statement that does not return data—for example, INSERT, UPDATE, or DELETE—you must use the <sql:update> tag. The <sql:query> tag takes three forms:

// Syntax 1: Without body content
<sql:query sql="sqlQuery"
var="varName"
[scope="{page|request|session|application}"]
[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]/>

// Syntax 2: With a body to specify query arguments
<sql:query sql="sqlQuery"
var="varName"
[scope="{page|request|session|application}"]
[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]>
<sql:param> actions
</sql:query>

// Syntax 3: With a body to specify query and optional query parameters
<sql:query var="varName"
[scope="{page|request|session|application}"]
[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]>
query
optional <sql:param> actions
</sql:query>
Implementing the Forum Example

You have now seen how the JSTL SQL tags work. You have also seen the basic screen and database structure of the forum application. We'll now show you how the forum application is implemented. This way, you can see all of the JSTL SQL tags in action.

The listings in the following sections make up part of the forum application. We will examine only those listings that demonstrate unique uses of the JSTL SQL tags. You can find the source code to the complete forum application on the Sams Publishing Web site.

Let's start at the very beginning, with logging into the system.

Using a Query to Log In

The first page that a user will encounter on the forum application is index.jsp. This file displays a form that asks users either to enter their user ID and password, or to click the Register button to create a new account. Listing 7.4 shows the source code to this page, and Figure 7.5 shows what the form looks like.
A General Query Engine

Let's look at one more SQL example. In every query that we've examined so far, we always knew exactly what columns were going to be returned. It is also possible to write your JSTL so that the results of a query will be displayed, regardless of what the names of the columns are. Our next example implements a general query form. Using this form, users can enter any SQL query that they like and see the results of the query displayed. Figure 7.8 shows this program executing a query.

Figure 7.8. A general query page.

This program works in the same way as many of the other pages we've examined. It presents a form that posts back to the general.jsp page. Listing 7.8 shows the source code.

Listing 7.8 A General Query (general.jsp)

```jsp
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/core-rt" prefix="c-rt" %>
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
<sql:setDataSource var="dataSource" driver="org.gjt.mm.mysql.Driver" url="jdbc:mysql://localhost/forum?user=forumuser" />
```

Summary

In this chapter, you learned how to use SQL tags from JSTL. Using SQL tags allows you to quickly construct JSTL-only applications. For production applications, you will likely isolate data access away from the JSP files; Chapter 11 will show you how to do this.

Not all data is obtained using SQL. In the next chapter, we show you how to access XML data. We explain how you can use the SQL tags discussed in this chapter to both import and export XML data.
Chapter 8. Accessing Data with XML Tags

Considerable data is now available in the form of Extensible Markup Language (XML). Web pages often need to access XML data and display it. JSTL provides a range of XML tags that allow you to perform a variety of operations on XML data.

The JSTL tags enable you to iterate over XML data and perform comparisons on XML data using XPath expressions. You can also access individual elements of data within an XML document using XPath. In this way, you can customize the display of your XML using many of the JSTL tags you are already familiar with.

In addition, the JSTL tag library allows you to process data using Extensible Stylesheet Language templates (XSLT). By creating an XSL template, you can transform your XML data into HTML output, or even another XML document.

The XML tag library consists of three logical groups. The core tags perform the basic parsing and access to individual elements. Flow-control XML tags allow you to iterate over element collections and perform logical operations based on XPath expressions. Transformation operations allow you to use XSLT documents to reformat XML documents. We examine all three categories in this chapter. But first, let's look at XPath, a standard way of specifying sections of an XML document. The JSTL XML tag libraries make extensive use of XPath.
Understanding XML

An XML document is a hierarchical file consisting of nodes, attributes, and node values. Listing 8.1 shows a typical XML file. The examples presented in this chapter use this XML file.

Listing 8.1 A Typical XML File (students.xml)

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<students>
  <student id="1">
    <name>
      <first>John</first>
      <last>Smith</last>
      <middle>T</middle>
    </name>
    <grade>
      <points>88</points>
      <letter>B</letter>
    </grade>
  </student>
  <student id="2">
    <name>
      <first>James</first>
      <last>Smith</last>
      <middle>K</middle>
    </name>
    <grade>
      <points>92</points>
      <letter>A</letter>
    </grade>
  </student>
  <student id="3">
    <name>
      <first>Kelly</first>
      <last>Lane</last>
      <middle>A</middle>
    </name>
    <grade>
      <points>72</points>
      <letter>C</letter>
    </grade>
  </student>
</students>
```

XML files must always have only one top-level node. As you can see in Listing 8.1, the top-level node for this file is the `<students>` tag. Inside the `<students>` node are additional nodes, named `<student>`, for each student. Each of the `<student>` nodes has an attribute
Understanding XPath

XPath is a sort of query language for XML. On an abstract level, XPath resembles SQL. Like SQL, XPath is a language for accessing data. XPath expressions are not executed by themselves; XPath is always used in conjunction with another programming language. Generally, XPath is used with XSLT, but other languages are now taking advantage of XPath expressions. JSTL is one of these languages.

Complete coverage of XPath is beyond the scope of this book; we provide only a brief introduction to XPath. For more information on XPath, you should refer to the World Wide Web Consortium (W3C) site at http://www.w3.org/TR/xpath.

XPath expressions are similar to paths used to reference files. For example, the XPath expression doc/students/student/name/first would access the first student's first name. The $doc portion of the expression is provided by JSTL, and represents the actual XML document as the starting point. There are many other more complex forms of XSLT, some of which we examine in later chapters of this book.
Understanding XML Core Tags

Several core tags are provided by the JSTL XML tag library. These tags perform basic operations required by the other tags. The `<x:parse>` tag is used to parse XML data. When the `<x:parse>` tag is called, a variable is specified in which the parsed XML document will be stored. For example, consider the following code:

```xml
<!-- parse an XML document -->
<c:import url="http://www.site.com/students.xml" var="xml"/>
<x:parse source="${xml}" var="doc"/>
<!-- display using XPath expressions -->
<x:out select="${doc/name}"/>
<!-- set a scoped variable -->
<x:set var="name" scope="request" select="${doc/students/student/name/first}"/>
```

This code begins by accessing the file [http://www.site.com/students.xml](http://www.site.com/students.xml). This file is loaded into the variable `doc` using the `<c:import>` tag. The `<c:import>` tag allows the contents of a URL to be downloaded into a scoped variable. This tag will be covered in greater detail in Chapter 9, “Accessing Internet Resources with JSTL.”

The contents of the downloaded XML file are then parsed using the `<x:parse>` tag. The resulting document is stored in the scoped variable `doc`.

Now that the document has been parsed, we can display some of the values. We do this by using XPath expressions. These expressions are specified as the select attribute that is passed to the `<x:out>` and `<x:set>` tags. The XML document is accessed by specifying its scoped variable as part of the XPath expression using the form `$doc/students/student/name/first`.

You will find that many of the JSTL XML tags use this form. You simply specify a select attribute that holds an XPath expression that you want to be evaluated.

Now that we have seen how the core XML tags work in general, let's examine each of these tags in detail. We begin with the `<x:parse>` tag.

Using the `<x:parse>` Tag

The `<x:parse>` tag parses XML data so that it may be accessed using the other XML JSTL tags.
Using XML Flow-Control Tags

The XML flow-control tags enable your program to make decisions based on the results of XSL queries. The XML flow-control tags resemble some of the JSTL core tags.

The <c:if> tag allows the core tag library to perform logic based on what's entered or stored in scoped variables. The XML tag library adds the tag <x:if>, which enables your program to conditionally execute other tags based on the result of an XPath expression. The <c:if> tag is not the only tag that has a counterpart in the XML tag library; the tags <c:otherwise>, <c:choose>, and <c:when> all have the XML counterparts <x:otherwise>, <x:choose>, and <x:when>.

Using the <x:forEach> Tag

The <x:forEach> tag works much like the <c:forEach> tag. Both tags allow you to iterate over a collection of values. As we discussed earlier, an XPath expression can return a collection of nodes, and the <x:forEach> tag allows you to iterate over those nodes. There is one form of the <x:forEach> tag:

```<x:forEach [var="varName"] select="XpathExpression">
body content
</x:forEach>```

The <x:forEach> tag accepts the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>N</td>
<td>An XPath expression that you want to iterate over.</td>
</tr>
<tr>
<td>var</td>
<td>Y</td>
<td>Specifies the scoped variable that will hold each iteration.</td>
</tr>
</tbody>
</table>

The value placed in the variable—which is specified by the var attribute—contains all the values stored in the node. You can use <x:out> and <x:set> with relative XPaths that will obtain data from each member of the collection.
Transforming XML with XSLT

Using the flow-control and core XML tags of JSTL, you can transform XML documents into nicely formatted HTML documents. JSTL supports another standard that makes it even easier to format XML. Using XSLT, you can specify the exact format in which you would like your XML document to be formatted.

A complete discussion of XSLT is beyond the scope of this book. If you are interested in learning XSLT, visit [http://www.w3.org/TR/xslt](http://www.w3.org/TR/xslt). We also recommend Sams Teach Yourself XSLT in 21 Days (Sams, 2002, ISBN 0672323184).

We limit our discussion of XSLT to how XSLT is integrated with JSTL. JSTL allows you to specify two input files and receive the output. The input files are an XML data file and an XSLT formatting file. You specify the output in the XSLT formatting file. A properly written XSLT formatting file can output data in nearly any text format. The two most common forms are HTML and XML. XSLT is often used to generate HTML output from XML data. In addition, XSLT is used to translate between two incompatible XML formats.

The JSTL XML tags provide two tags that are used to work with XSLT files. The primary tag is the `<x:transform>` tag, which actually does the transformation. The second tag, `<x:param>`, allows you to specify parameters that will be passed to the XSLT script. These script parameters let you customize the operation of your XSLT script file.

Listing 8.1 shows the XML file that we've used for all the examples in this chapter. Let's now see how to use XSLT with this sample file. To do this, we must use an XSLT file designed to work with the students.xml file. We've shown this XSLT file in Listing 8.7.

Listing 8.7 A Sample XSLT File (transform.xsl)
```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="students">
        <html>
            <head>
                <title>XSLT Transform</title>
            </head>
            <body>
                <table border="1">
                    <tr>
                        <th>First</th>
                        <th>Last</th>
                        <th>Points</th>
                        <th>Letter</th>
                    </tr>
                    <xsl:for-each select="student">
                        <tr>
                            <td bgcolor="{$bgColor}">
                                <xsl:value-of select="name/first"/>
                                <xsl:value-of select="name/last"/>
                                <xsl:value-of select="grade/points"/>
                                <xsl:value-of select="grade/letter"/>
                            </td>
                        </tr>
                    </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```
Summary

In this chapter, you learned how to access XML data with the XML tag library provided by JSTL. The JSTL XML tag library makes extensive use of XPath, a standard query language developed by the W3C that enables you to query XML documents. JSTL uses XPath to specify individual elements that you want to access, collections you want to iterate over, and expressions you want to use for Boolean comparisons.

Now you know how to process XML. As you learned in this chapter, XML is often obtained from files residing on the Web server. XML can also come from completely external sources. In the next chapter, we show you how to access external data sources. Often that data is XML, and can be processed using the tags we discussed in this chapter.
Chapter 9. Accessing Internet Resources with JSTL

In the previous chapter, you learned how to work with XML data, but we did not focus on where this data comes from. Much data is available on the Internet, in the form of XML and HTML.

The tags that we discuss in this chapter allow you to access information that is available on the Internet. Usually, this data is in XML, but it does not have to be. The JSTL tag library can easily access any form of text data stored on the Internet.

Accessing resources on the Internet requires the use of a URL. JSTL provides tags that let you easily format URLs for a variety of purposes. You can format URLs that include parameters for accessing external resources outside your Web page.

In addition to accessing external resources, you may want to access resources stored on your own Web site. You can use the URL tags to create URL strings that specify URLs within your Web site. This allows portions of your Web site to send parameters to other parts of your site. We begin this discussion by showing you the JSTL tags that enable you to access information on the Internet and manage URLs.
The URL-Related Tags

JSTL makes available tags that allow you to construct URLs, import data from external sources, and redirect users to new Web pages. The \(<\texttt{c:url}>\) and \(<\texttt{c:param}>\) tags enable you to construct URLs. Using the \(<\texttt{c:import}>\) tag, your program can access data from external sources, and using \(<\texttt{c:redirect}>\), you can redirect your user to new Web pages. Let's examine all four of these tags, beginning with the \(<\texttt{c:import}>\) tag.

**Using the \(<\texttt{c:import}>\) Tag**

The \(<\texttt{c:import}>\) tag lets you import the contents of another Web page into your page. This data can be displayed directly to your page or it can be copied to a variable. There are two forms of the \(<\texttt{c:import}>\) tag:

**Syntax 1:** Resource content inlined or exported as a String object

```
<\texttt{c:import url="url" [context="context"]}
  [var="varName"] [scope="{page|request|session|application}"]
  [charEncoding="charEncoding"]>
  optional body content for \(<\texttt{c:param}>\) subtags
</\texttt{c:import}>
```

**Syntax 2:** Resource content exported as a Reader object

```
<\texttt{c:import url="url" [context="context"]}
  varReader="varReaderName"
  [charEncoding="charEncoding"]>
  body content where varReader is consumed by another action
</\texttt{c:import}>
```

The \(<\texttt{c:import}>\) tag accepts these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>charEncoding</td>
<td>N</td>
<td>Allows you to specify the character encoding (for example, UTF16).</td>
</tr>
<tr>
<td>context</td>
<td>N</td>
<td>Specifies the base URL that will be used to resolve a relative URL given by the url attribute.</td>
</tr>
</tbody>
</table>
Understanding Internet Resources

Numerous sources of XML information are available on the Internet. Many are subscription-based and require you to make payments to their provider. There are also many free XML resources that you can take advantage of on the Internet.

One benefit of using Internet resources is that many times they allow two platforms to work together. Microsoft Passport is an example of Java working with a subscription-based XML service available on the Internet. Passport attempts to solve the problem of identifying users. Often, a user will maintain accounts with several different subscription-based Web sites that the user frequents. This requires the user to register with each individual Web site.

Microsoft Passport attempts to alleviate this burden by having the user register just once, and then provides a universal registration to every site that the user frequents. This universal login is provided by the Microsoft Passport Service.

In order for a typical Web site to maintain such a relationship with Microsoft Passport, the site must exchange information with Microsoft. To accomplish this, XML is used. This is an example of how a Web site might use the Internet resource tags provided by JSTL.

There are also many free sources of XML data that you can integrate into your application. One such service is RSS, which we use for many of this chapter's examples. Let's take a look.

Introducing RSS

Rich Site Summary (RSS) is an XML standard for making the headlines available through a news Web site available to the other Web sites. This standard was introduced by Netscape.
JSTL: JSP Standard Tag Library Kick Start
By Jeff Heaton

Table of Contents

Chapter 9. Accessing Internet Resources with JSTL

Summary

In this chapter, we explained how to use the URL-related tags provided by JSTL. We can use these tags to construct URLs that include parameters that enable communication to take place across Web pages. In the next chapter, we introduce I18N programming in JSTL.
Chapter 10. Understanding JSTL Internationalization

Java was designed to support internationalization. All strings in Java are stored internally as Unicode strings, which normally are transparent to the programmer. However, when your program has to support a language such as Japanese or Chinese, you must use Unicode.

One of the most common reasons for making a program international is the need to display the program's data in a variety of languages. JSTL provides several tags that make it easy to create a multilingual program. In this chapter, we show you how to make our forum example from Chapter 7 work with multiple languages.
The I18N-Related Tags

To create a multilingual JSTL Web application, you must isolate all language-dependent parts of the program to one area. The language-dependent parts are the strings that your program displays.

To make our forum example multilingual, we must go through the individual source files and find all the strings that are currently in the English language. We then have to move these strings to a resource bundle.

A resource bundle is a collection of strings in one particular language, such as English. When you want your program to support languages other than English, you simply translate the resource bundle to another language. These resource bundles are then made available to the Web application. Your program can dynamically choose between resource bundles and change the language that people are using as they work with the application.

A resource bundle is something that exists in standard Java as well as JSTL. In fact, you may have already worked with resource bundles. Later in this chapter, we describe how to create them, but let's begin by explaining the tags that you can use to work with resource bundles. The first tag that we'll look at is the <fmt:setLocale> tag, which enables you to select the locale that your Web application is being used from.

Using the <fmt:setLocale> Tag

The <fmt:setLocale> command is used to set the locale of a Web session. It lets you set the locale for one user or for the entire Web application. Usually, it is preferable to have the Web server maintain individual locales for each of the logged-in users. The locale determines how resource bundles are chosen and how other formatting operations act. There is one form of
Resource Bundles

In this section, we show you how to create a resource bundle. You create resource bundles for JSTL the same way you do for ordinary Java applications.

Programmers create resource bundles for a variety of languages. For someone who is already familiar with English, it is quite easy to create a resource bundle for English or one of the other Romance languages. Non-Romance languages, such as the Asian languages, present some additional challenges, however.

What Is a Resource Bundle?

First, we'll describe a resource bundle and show you how to construct one. The method that we focus on involves creating compiled Java classes that extend the java.util.ListResourceBundle class. You must compile these class files and make them available to the classpath of your Web application.

A Java resource bundle file contains a series of key-to-string mappings. The following demonstrates a simple Java resource bundle:

```java
public class Example extends ListResourceBundle {
    public Object[][] getContents() {
        return contents;
    }
    static final Object[][] contents = {
        {"count.one", "One"},
        {"count.two", "Two"},
        {"count.three", "Three"},
    }
}
```

If you compiled this resource bundle and made it available to the classpath, you could use the following JSTL tags to display the three numbers:

```xml
<fmt:setLocale value="en"/>
<fmt:setBundle basename="com.heaton.bundles.Example" var="lang" scope="session"/>
<fmt:message key="count.one" bundle="${lang}"/><br/>
<fmt:message key="count.two" bundle="${lang}"/><br/>
<fmt:message key="count.three" bundle="${lang}"/><br/>
```
A Multilingual Forum Application

You should now have the necessary resource bundles for our forum application. Let's see how to convert the forum example shown in Chapter 7 to a multilingual application. The majority of the work involves changing all the English text to `<fmt:message>` tags. We won't show you every converted JSP file; rather, we'll just examine the index.jsp file in order to demonstrate how to use the `<fmt:message>` tags.

The index page for our forum application now allows you to easily switch between the three languages. The default language, English, is shown in Figure 10.3.

Figure 10.3. Our multilingual form in English.

Simply clicking on the Spanish link will cause the site to redisplay in Spanish. Figure 10.4 shows this transformation.

Figure 10.4. The forum in Spanish.
Summary

In this chapter, we saw how you can add languages to a Web application. We extended the sample program from Chapter 7 to support Spanish and Chinese. Our forum application is now a multilingual Web application.

In the next chapter, we make one final change to our forum application. We take the many SQL commands that exist in the application and isolate them to a custom tag library. You'll learn how you can use your own custom tag libraries, in conjunction with JSTL, to create Web applications.
Chapter 11. Creating Your Own Tag Libraries

In this chapter, we show you how to create custom tag libraries. We focus particularly on creating tag libraries that work well with the JSTL tag library. By designing your tag libraries so that they can deal with JSTL scoped variables and the JSP 1.3 expression language (EL), you will be able to create tag libraries that easily integrate with JSTL code.

Tag libraries allow you to take procedures that would be lengthy to write using just JSTL and encapsulate them in a tag. One perfect example of this is the JSTL SQL tags. It is generally not a good idea to put SQL database code in JSP pages. Database code is an implementation detail that should be separated from the presentation-oriented JSP pages. The example we present in this chapter shows you how to take our forum example from Chapter 10, "Understanding JSTL Internationalization," and isolate all the database logic behind a custom tag library. We begin with a discussion of how we should structure the tag library.
Developing Custom Tag Libraries

While our goal in this chapter is to develop a custom tag library that encapsulates the database management of our forum application, this is not to say that we will completely replace the use of JSTL. Rather, we want to design a number of custom tags that are capable of processing forum information. We will still use JSTL for formatting, iteration, flow, and other basic presentation-level issues.

It is important to correctly spread the application across the necessary tags. This requires some design to determine the best mix of classes and tags that will be used by your JSP pages. In the next section, we examine the tags that we created for our forum application.

Tags Used by the Forum Application

The forum application's tag library consists of 16 tags. These tags are summarized in Table 11.1.

Table 11.1. Custom Tags Used by the Forum Application

<table>
<thead>
<tr>
<th>Tag Name</th>
<th>Attributes</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>createForum</td>
<td>code, name,</td>
<td>Creates a new forum.</td>
</tr>
<tr>
<td></td>
<td>sequence</td>
<td></td>
</tr>
<tr>
<td>deleteForum</td>
<td>code</td>
<td>Deletes an existing forum.</td>
</tr>
<tr>
<td>deleteMessage</td>
<td>code, number</td>
<td>Deletes a forum message.</td>
</tr>
<tr>
<td>deleteUser</td>
<td>id</td>
<td>Deletes a user.</td>
</tr>
<tr>
<td>editForum</td>
<td>code, name,</td>
<td>Edits a forum.</td>
</tr>
<tr>
<td></td>
<td>sequence</td>
<td></td>
</tr>
</tbody>
</table>
The Components of a Tag Library

You have now seen the basic structure of the forum application's tag library. We must now implement this tag library and modify the JSP files to use the new tag library rather than the SQL tags we used in Chapter 10.

In essence, a tag library is a JAR and an XML-formatted tag library descriptor (TLD) configuration file. Installing a tag library consists of three steps:

1. Copy the tag library descriptor file (*.tld) to the WEB-INF directory for your Web application.

2. Modify the web.xml file for your Web application so that it includes the TLD file that you copied in step 1.

3. Copy the tag library's JAR file to the /lib directory of your Web application so that your Web server can access it.

Let's now examine each of these steps in detail. In particular, we focus on the structure of the JAR and TLD files so that you will be able to create your own tag libraries that you can use to supplement JSTL with your own company-specific procedures.

The Tag Library Descriptor File (TLD)

The forum application's tag library stores its TLD information in a file called forum.tld. This file is quite lengthy, and will not be included here in its entirety. It contains definitions for each of the 16 tags that make up the forum tag library.

The TLD File Header

The forum.tld file begins with basic header information:
Summary

In this chapter, we showed you how to create your own tag libraries. We described the steps necessary to allow your tags to process EL-based expressions. It is important to design your tag libraries to be compatible with the JSTL tag libraries. That way, you won't have to design the common formatting and iteration tags that make up the JSTL tag library.

In the next (and final) chapter, you'll learn how to tie everything together. We explain how to deploy Web applications that consist of a mixture of custom tag libraries, JSTL, and JSP code. These components can all be combined into one common file called a Web Archive (WAR) file.

Debugging is an important aspect of Web development. In the next chapter, we also examine how to debug JSTL tag libraries. We show you how to output trace information, read error messages, and use the JDK 1.4 logging API with Tomcat.
Chapter 12. Debugging and Deploying Tag Libraries

Up to this point, you have learned how to use the JSTL tag libraries and how to create your own tag libraries. There are two other important factors that you must also consider when developing a Web application.

Debugging is always a considerable part of the development process. Programs rarely work as designed on the first try. In this chapter, we show you how to read error messages generated by the Web server. You will also learn how to use an integrated development environment (IDE) to debug your Web applications.

Packaging and deployment of your Web applications is another important factor. Web applications are often deployed onto multiple computers. Properly packaging your application makes this deployment process easy. In this chapter, we also show you how to deploy your Web applications.
Debugging

Debugging is a critical process for any programming endeavor. JSTL, like any other programming language, is susceptible to programming errors. To effectively program JSTL and custom tag libraries, you must know how to combat these errors. In this section, we show you debugging techniques you can use to debug errors you're likely to encounter when using JSTL and your own custom tag libraries.

Various types of errors can be returned from a JSTL page. The error reports that JSTL displays are designed to provide information that will lead you to the cause of the problem.

Error reports returned from a JSP server may look like a complex jumble of text and numbers, but you must learn how to read them to determine the cause of the error. Several types of errors are common with JSTL pages; they range from general exceptions to JSTL-specific parsing errors. We begin by examining a general exception.

General Exceptions

An application displays a general exception whenever it executes a JSP page that causes a Java exception to be thrown. More than one type of exception can be thrown to a JSP page. Any exception that can be thrown by Java can cause a JSP page to terminate if the exception is not caught.

General exceptions have many causes, and we won't attempt to list them all. In this section, we examine the exception that is thrown when the JSTL processor encounters beginning and ending tags that do not match. This can often happen when you forget an ending tag. When such an error occurs, you'll see output similar to Figure 12.1.

Figure 12.1. A general JSTL exception.
Debugging with an IDE

An integrated development environment (IDE) is a GUI application that allows you to develop a Java application using a convenient editor. Though using an IDE makes it convenient to develop your application, one of the chief benefits to using an IDE is debugging. You have likely already used an IDE to debug your Java applications. It is also possible to use an IDE to debug a custom tag library, such as the tag library used in Chapter 11.

In this section, we show you how to use an IDE to debug a custom tag library. Using the IDE, you can easily debug your application as you step through your routines and observe the values of variables as your program runs. This can be invaluable for debugging your program and learning why it does what it does.

Commercial IDEs include Visual Cafe, JBuilder, and Visual Age. Many IDEs are available for free. For example, Sun's Forte includes a community edition that you can download for free Forte at http://java.sun.com/.

In addition, the Eclipse IDE is a completely open source project being developed by IBM. Next, we show you how to install this IDE to use with Tomcat.

Introducing Eclipse

Eclipse is an advanced and very popular IDE for the Java platform. There is some speculation as to where the name for Eclipse came from. Some say that it IBM meant to imply that they will "eclipse" Sun Microsystems in the Java tools market. Whatever the meaning behind the name, it is a valuable tool.

You can download Eclipse from the IBM Web site at http://www.eclipse.org/. The Eclipse IDE can be used with many platforms. If you're using Windows or Linux, you can download a binary version that is ready for install.

When you download Eclipse, you'll notice that the program is in one large Zip archive. Inside this archive is a single folder, named eclipse. You should copy the eclipse folder to a convenient location on your computer. Inside the eclipse archive is a single executable file named eclipse.exe. Double-click this file to start the Eclipse system.
Deploying Web Applications

Installing and packaging your Web application is a critical step. Your application will not be successful if it is not easy to install. Web applications that make use of JSTL and your own custom tag libraries are distributed in exactly the same way as other Web applications. In this section, you'll learn how to create a single archive that stores all of your Web application's files.

Understanding WAR Files

A Web Archive, or WAR file, is a common means of distributing Web applications. A WAR file resembles a Java JAR file: Both have exactly the same format and are based on the Zip file format. A WAR file allows you to package all of the files that are necessary to run your Web application. This includes the JSP pages, the JAR files, and other configuration files. This file is then placed in the Web applications directory (webapps) of a JSP Web server, where it can be executed.

Although a WAR file allows you to store many of the components of your Web application, it cannot store everything. Programs that are completely external to your Web application are not stored in the WAR file. For example, the database in your database connection must be configured separately from the WAR file. The forum application uses a MySQL database to store its data.

While you are not able to store the database as part of the WAR file, the configuration settings used to connect to that database are probably stored in other configuration files. These configuration files will be stored in the WAR file.

Let's look at an example, using the forum application we created in Chapter 11. The forum application is just a directory off the ROOT directory of Tomcat. We'll move this directory to a WAR file. That way, we can distribute the forum application as one single file component that we can place on a Web server.

Creating a WAR file is relatively simple. If you know how to create a JAR file, you already know how to create a WAR file. The procedure for creating a WAR file is identical to that for creating a JAR file. The only difference is that the final output file has a .war extension rather than .jar.
Summary

In this chapter, we showed you how to debug and deploy Web applications that make use of JSTL and custom tag libraries. We can package Web applications as WAR files that can be easily integrated with a Web server such as Tomcat. WAR files are a common way to distribute Web applications.

Throughout this book, you learned how to use the JSTL tag library. You are now ready to design and implement Web applications that use the JSTL tags. You have also learned how to develop your own custom tag libraries that are compatible with JSTL. This allows you to create advanced Web applications and encapsulate their unique functionality within custom tag libraries. You can then create advanced Web applications using your own custom tag libraries and using JSTL to tie the entire application together.
Appendix A. JSTL Reference

This appendix contains an alphabetical listing of the JSTL tags. The tags are grouped according to the four tag libraries that make up JSTL. We present each tag, along with the attributes used with that tag.

JSTL supports the notion of twin tag libraries. The tags shown here are from the more commonly used EL library. These tags are capable of handling expressions of the following form:

\[
\texttt{<c:set var="a" value="\{100*100\}"/>}
\]

In addition to the EL expression language, JSTL includes a second set of tags that are able to process RT expressions. To specify the RT twin library, you must append the suffix `-rt` to the end of the tag library name. For example, the tag `<c:set>` becomes `<c-rt:set>`. The following tag uses the RT expression language:

\[
\texttt{<c-rt:set var="a" value="\%=100*100\%"/>}
\]

In general, you should use the EL expression language whenever possible. The older RT expression language is included primarily for compatibility with older code.
The Core Tags

The core tags perform operations that form the foundation of the JSTL tag library. Using the core tags, you can set scoped variables, iterate over collections, and handle exceptions.

The <c:catch> Tag

The <c:catch> tag is used to allow JSP pages to handle exceptions that might be raised by any of the code contained in the body of the <c:catch> tag.

```xml
<c:catch var="e">
  ... Program code that may throw an exception
  ...
</c:catch>
```

The <c:catch> tag has one attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>Y</td>
<td>Specifies the scoped variable that should receive the exception.</td>
</tr>
</tbody>
</table>

The <c:choose> Tag

The <c:choose> tag is used to form the body of a mutual exclusion. The tags <c:when> and <c:otherwise> should be embedded inside the <c:choose> tag. When the <c:choose> block is reached, only one of the <c:when> tag blocks will be executed. If none of the <c:when> tags evaluates to true, then the <c:otherwise> tag, if present, will be executed. The <c:choose> block works much like the switch, case, and default statements in the Java programming language. The <c:choose> tag has no attributes.
The I18N Tags

Java contains extensive support for internationalization, or I18N. Multilingual programming is supported by allowing strings to be stored in language-specific Java resource bundles. These bundles can be exchanged as language needs vary. JSTL also includes formatting tags that allow you to properly format numbers, dates, and currencies for specific locales.

The <fmt:bundle> Tag

The <fmt:bundle> tag is used to load the specified bundle and use it as the default bundle for any <fmt:message> tags that fall inside the <fmt:bundle> block. If you want to load a bundle to a scoped variable, you should use the <fmt:setBundle> tag.

```xml
<fmt:bundle basename="basename" prefix="prefix"/>
</fmt:setBundle>
```

The <fmt:bundle> tag has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>basename</td>
<td>Y</td>
<td>Specifies the base name of the resource bundle that is to be loaded.</td>
</tr>
<tr>
<td>prefix</td>
<td>N</td>
<td>Specifies a prefix that will be added to every key attribute specified in the <a href="">fmt:message</a> tags.</td>
</tr>
</tbody>
</table>

The <fmt:formatDate> Tag
Appendix A. JSTL Reference

The Relational Database Tags (SQL)

The relational database tags allow a Web application to access a SQL database. You must have a JDBC driver and know the correct URL to use to access this database. You can use the SQL tags to perform queries, updates, and use transactions.

The SQL tags are intended primarily for rapid prototyping or very small Web applications. For most nontrivial Web applications, we suggest that you isolate your database access into tag libraries of your own. Chapter 11 shows how to do this.

The `<sql:dateParam>` Tag

If you are going to specify a parameter to the `<sql:query>` or `<sql:update>` tags and that parameter is a date, you must use the `<sql:dateParam>` tag. If the parameter is not a date, then use the `<sql:param>` tag.

```xml
<sql:dateParam value="value" type="[timestamp|time|date]" />
```

The `<sql:dateParam>` tag has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Y</td>
<td>The value to be inserted for this parameter.</td>
</tr>
<tr>
<td>type</td>
<td>N</td>
<td>Must be timestamp, time, or date. The default is timestamp.</td>
</tr>
</tbody>
</table>

The `<sql:param>` Tag
The XML Tags

XML is becoming a common form of data encountered on the Internet. JSTL makes available several XML tags that allow XML documents to be formatted and processed. Most of the XML tags use XPath expressions to specify the operation you want to perform. XPath is covered in Chapter 8.

The <x:choose> Tag

The <x:choose> tag is used to specify a mutual exclusion block. Inside the <x:choose> tag's body, <x:when> and <x:otherwise> tags reside. The first <x:choose> tag that is satisfied will execute the tags inside the <x:choose> tag's body. If no <x:choose> tags are satisfied, then the <x:otherwise> tag, if present, will be executed. The <x:choose> tag has no attributes.

```xml
<x:choose>
  body content  (<x:when> and <x:otherwise> subtags)
</x:choose>
```

The <x:forEach> Tag

The <x:forEach> tag allows you to iterate through child nodes of an XML document. The <x:forEach> tag works much like the <c:forEach> tag, in that the body is executed for each item in the collection. As the loop progresses, each item is copied to the var attribute.

```xml
<x:forEach [var="varName"] select="XpathExpression">
  body content
</x:forEach>
```

The <x:forEach> tag has the following attributes:
Appendix B. Installing JSTL and Tomcat

In this appendix, we show you how to install Tomcat and JSTL for use with this book. We focus on the Windows installation, but you can use these instructions with other operating systems as well. For specific information on how to install JSTL in a Unix environment, refer to Appendix D, "Unix Installation Notes."

To compile and execute the examples in this book, you must install several components: a JDK version, Tomcat, and JSTL. Once these three components are installed, you are ready to develop JSTL Web applications and use the examples in this book.

Some of our examples also use a database connection. To use these examples, you must also install a database, such as MySQL. We cover installing MySQL in Appendix C.
To run any of the examples in this book, you have to install the JDK. We tested our examples with JDK 1.4, so we suggest you use JDK 1.4 or higher for JSTL.

You can download the JDK from the Sun Web site at http://java.sun.com/j2se. Make sure that you download the full SDK, not just a runtime environment (JRE). The file you download will be named j2sdk-1_4_0_01-windows-i586.exe, or something similar. You should execute this file to begin the process of installing the JDK.

Executing the downloaded file will launch an Install Shield installation program. This program makes the process of installing the JDK relatively easy. Once you’ve installed the JDK, the installation program exits and you are ready to install Tomcat.
Installing Tomcat

JSTL requires JSP pages to execute. To execute JSP pages, you must have a JSP-compatible Web server installed on your system. JSTL should work with any JSP 1.2 or higher Web server. The Web server we use in this book is the Apache Tomcat Web server. In this section, we show you how to install Tomcat.

The examples in this book were tested with Apache Tomcat v4.0.3. We recommend that you use at least this version of Tomcat. Before you install Tomcat, it is absolutely essential that you have the JDK installed. Refer to the previous section for more information on installing the JDK.

You can download the Tomcat Web server from the Apache Web site at http://jakarta.apache.org/tomcat/. When you begin the download, you'll notice that several versions are available. If you are using Windows, you can choose between a light (LE) and a full version. Both versions are meant to function exactly the same. The only difference is that the LE version does not include some of the JAR files normally included with JDK 1.4, such as for XML processing. The result is that if you are using JDK 1.4, you can choose the LE version and not have to download JAR files that you already have. The safest route is usually just to install the full (non-LE) version.

Once you download Tomcat, you should have a file named Jakarta-tomcat-4.0.3.exe. This file, when executed, functions as the installer for the Tomcat Web server. To begin the installation, execute Jakarta-tomcat-4.0.3.exe. Once the Tomcat installation program begins, you see a window that looks like the one shown in Figure B.1.

Figure B.1. Installing Apache Tomcat.
Installing the Book Examples

Now that you have Tomcat properly installed, you are ready to install the sample programs from this book. You can download these programs from [http://www.samspublishing.com/](http://www.samspublishing.com/). The examples are packaged as one Web directory that you can copy to the Tomcat Web server. This directory contains both the examples and JSTL 1.0. This is the easiest way to install the examples. However, there are times when you may want to directly install each component. Later in this appendix, we explain how to install JSTL without the book examples.

Tomcat stores all of the Web files it needs in a directory named webapps. When you download the examples that come with this book, you'll notice that the Zip archive includes a directory named webapps, which contains all of the book examples. Simply overwrite your Tomcat Web server's webapps directory with this one from Sams. Figure B.5 shows the directory structure.

**Figure B.5. The Tomcat directories.**
The Classpath and Search Path

It is usually a good idea to make the BIN directory for the JDK a part of the system search path. Doing this will allow you to type the command javac from any directory and be able to compile a program. You also have to modify your classpath so that certain Tomcat JAR files are available to the Java compiler. You must do this in order to execute the examples from Chapter 11.

Both the classpath and system search path are defined in the same area. If you are using Windows XP, Windows 2000 Pro, or Windows NT, you set them in the System window in Control Panel, as shown in Figure B.6.

Figure B.6. The System window.

You have to modify the system path to include the path of your JDK's BIN directory. The system path is stored in the environment variable named PATH. Your JDK BIN directory is most likely C:\j2sdk1.4.0\bin or something similar. Add this path to your PATH environment variable.
Appendix B. Installing JSTL and Tomcat

Installing JSTL without the Examples

If you follow the instructions in the previous section, you will correctly install JSTL and the book examples. In this section, we explain how to install JSTL apart from the book's examples. If you want to install JSTL into a server environment, this is the procedure you follow.

The exact directory that JSTL will install into varies somewhat, depending on the version. In the 1.0 version, there is a jstl-1.0 directory, and in 1.0.1 it is called standard-1.0.1. Inside this directory is a lib directory that contains several files. Table B.1 documents the purpose of each of these files.

Table B.1. The JSTL Files

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>c.tld</td>
<td>The core tag library (EL version)</td>
</tr>
<tr>
<td>c-rt.tld</td>
<td>The core tag library (RT version)</td>
</tr>
<tr>
<td>dom.jar</td>
<td></td>
</tr>
<tr>
<td>fmt.tld</td>
<td>The I18N tag library (EL version)</td>
</tr>
<tr>
<td>fmt-rt.tld</td>
<td>The I18N tag library (RT version)</td>
</tr>
<tr>
<td>jaxp-api.jar</td>
<td>Required JAR file for JSTL</td>
</tr>
<tr>
<td>jaxen-full.jar</td>
<td>Required JAR file for JSTL</td>
</tr>
<tr>
<td>jdbc2_0-stdext.jar</td>
<td>Required JAR file for JSTL</td>
</tr>
<tr>
<td>jstl.jar</td>
<td>Required JAR file for JSTL</td>
</tr>
<tr>
<td>sax.jar</td>
<td>Required JAR file for JSTL</td>
</tr>
<tr>
<td>saxpath.jar</td>
<td>Required JAR file for JSTL</td>
</tr>
<tr>
<td>sql.tld</td>
<td>Database tag library (EL version)</td>
</tr>
<tr>
<td>sql-rt.tld</td>
<td>Database tag library (RT version)</td>
</tr>
<tr>
<td>standard.jar</td>
<td>Required JAR file for JSTL</td>
</tr>
<tr>
<td>xalan.jar</td>
<td>Required JAR file for JSTL</td>
</tr>
<tr>
<td>xerxesImpl.jar</td>
<td>Required JAR file for JSTL</td>
</tr>
<tr>
<td>x.tld</td>
<td>XML tag library (EL version)</td>
</tr>
<tr>
<td>x-rt.tld</td>
<td>XML tag library (RT version)</td>
</tr>
</tbody>
</table>

To properly install JSTL for your Web application, you must copy these files to specific locations. The two JAR files (standard.jar and jstl.jar) must be copied to your Webapp's library directory. This ensures that these files will be part of the classpath. Under the Windows operating system, your lib directory for Tomcat will be C:\Program Files\Apache Tomcat 4.0\webapps\ROOT\WEB-INF\lib.

You copy the tag definition files, which have a .tld extension, to your WEB-INF directory. This is typically a child directory of the Web root directory for your Web application. The Web root directory is the directory that holds all your JSP and HTML files. Under the Windows operating system, Tomcat's Web root directory is typically C:\Program Files\Apache Tomcat 4.0\webapps\ROOT. Tomcat's WEB-INF directory is typically C:\Program Files\Apache Tomcat 4.0\webapps\ROOT\WEB-INF.

The last step is to let the Web server know where the TLD files are stored. You do this by modifying the web.xml file, which should be located in your WEB-INF directory. Listing B.1 shows a web.xml file that uses all four JSTL tag libraries.

Listing B.1 A web.xml That Uses All Four JSTL Taglibs

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <taglib>
        <taglib-uri>http://jakarta.apache.org/taglibs/core</taglib-uri>
        <taglib-location>/WEB-INF/c.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>http://jakarta.apache.org/taglibs/sql</taglib-uri>
        <taglib-location>/WEB-INF/sql.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>http://jakarta.apache.org/taglibs/fmt</taglib-uri>
        <taglib-location>/WEB-INF/fmt.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>http://jakarta.apache.org/taglibs/xml</taglib-uri>
        <taglib-location>/WEB-INF/x.tld</taglib-location>
    </taglib>
</web-app>
```
Appendix C. Installing MySQL

To create useful Web applications, you must use some sort of a database. In this appendix, we show you how to install a MySQL database. MySQL is a popular database that is released under a GNU Public License (GPL). There is no charge for using MySQL.

Some of the applications we present in this book require that you use a relational database. Several chapters use a forum application that demonstrates JSTL features. The forum application requires a database to maintain its user lists and message areas. In this appendix, you'll learn how to create a MySQL database that is ready to work with the forum applications we present in this book.
Obtaining and Installing MySQL

You must download a copy of MySQL to install it to your computer system. Prebuilt binary versions of MySQL are available for all popular operating systems. To download a copy of MySQL, go to [http://www.mysql.com/](http://www.mysql.com/). From this Web site, you can access a download section that allows you to choose among several versions of MySQL.

Notice that there are two main versions of MySQL. A full-featured version, named MySQL-Max, is the one you should download. The regular version of MySQL has no support for some of the more advanced features. For this book, we assume that you are using the Max version; we used MySQLMax 3.23.

Once you download the Windows version of MySQL, you should have a file named something like mysql-3.23.48-win.zip. (This is a zipped file. If you are unable to unzip files, refer to the site [http://www.winzip.com/](http://www.winzip.com/).) Once you unzip the MySQL file, you'll find a standard installation program. Run the program named setup to install MySQL to your system. This process is shown in Figure C.1.

![MySQL Server and Clients 3.23.49 Setup](image)

**Figure C.1. Installing MySQL.**
Setting Up MySQL

Now that you've installed your MySQL database system, you must be able to launch your server. If you're using Windows XP, Windows 2000 Pro, or Windows NT, we suggest that you configure MySQL to run as a service. If you're using an older version of Windows, refer to the MySQL documentation.

If you're using Unix, you do not need to worry about services. MySQL for Unix installs to be ran as a daemon process. For more information about using Unix, refer to Appendix D, "Unix Installation Notes."

By configuring MySQL to work as a service, you are guaranteed that the MySQL database system will always be running. Services run even when no user is currently logged into the system. To set MySQL to run as a service, you have to start a command prompt window. From this window, change the current directory to the BIN directory of MySQL.

Earlier, we set up the path so that you can execute the MySQL commands from anywhere. When you're installing the service, it is important that you do not install these commands from anywhere. To install MySQL to run as a service, you must change to the c:\mysql\bin directory. To do this, issue the following command in the command prompt window:

cd \mysql\bin

Now that you are in the BIN directory, you have to instruct MySQL to install itself as a service:

mysql-max-nt –install

At this point, MySQL is set up to run as a service. The next time that you restart your computer, MySQL will run in the background and you can easily access it. To start the service without having to restart your computer, use this command:

net start mysql

Now that the service is running, you can connect to your MySQL database for the first time. The following mysql command connects you to the MySQL instance you just installed (as the root user):

mysql –u root
Creating the Forum Example

Now that you've installed MySQL and created a forum database, you must install the database. You do this using a script file provided at the Sams Web site. This script file, which is explained in greater detail in Chapter 7, contains all of the SQL necessary to create your forum database. The script is designed to be called against either a blank database or a database that already has a forum. If you run this script against an existing forum database, it will reset the database back to its original state.

To run this script, you must be at a command prompt. Make sure you are in a directory that contains createforum.sql. From the command prompt, issue this command:
mysql –u forumuser <createforum.sql

Once this command executes, you will have the tables necessary to test the forum application. To ensure that you installed everything correctly, you should perform a query against the t_users table. To do this, enter MySQL with the following command:
mysql –u forumuser

Now that you are inside MySQL, you must set the correct database:
use forum;

At this point, you can perform a basic SQL SELECT statement, such as the following:
select * from t_users;

If all of these commands execute properly, your screen should look like the one shown in Figure C.5.

Figure C.5. Setting up the forum database.
Installing a JDBC Driver

For Java to use a database, you have to have a JDBC driver. Several drivers are available that you can use. The driver that we use in this book is the MM driver, which you can download from http://mmmysql.sourceforge.net/. If you followed the instructions in Appendix B, you won't need to install the MM driver. The examples download contains the MM driver already properly installed.

The MM driver is distributed as a JAR file named mm.mysql-2.0.14-you-must-unjar-me.jar. This is not an ordinary JAR, and it cannot be used directly by Java. The author of MM is simply using JAR as an archive format, similar to Zip. The easiest way to extract from this archive is to rename it to a .zip file and use a Zip tool.

Once you access the contents of this archive, locate the MM JAR file, which should be named mm.mysql-2.0.12-bin.jar or something similar. You must make this file accessible to the classpath. For the purposes of this book, this involves placing the file in the directory C:\Program Files\Apache Tomcat 4.0\webapps\ROOT\WEB-INF\lib.

Now that you have the MM driver installed, you are ready to use MySQL from JDBC. To access any driver from JDBC, you must know the driver name and the URL of the database. The driver name for the MM driver is org.gjt.mm.mysql.Driver, and the URL is jdbc:mysql://localhost/forum?user=forumuser. To use this information in JSTL, use the following <sql:setDataSource> tag:

```jsp
<sql:setDataSource var="dataSource" 
    driver="org.gjt.mm.mysql.Driver"
    url="jdbc:mysql://localhost/forum?user=forumuser"
    scope="session" />
```

See Chapters 7 and 11 for more information about using JSTL.
Appendix D. Unix Installation Notes

This book contains many examples that illustrate the use of JSTL. Appendixes B and C explain how to set up your environment for the Windows platform. The instructions in these two appendixes generally apply to using JSTL with Unix as well.

We developed our examples on a system that runs Red Hat Linux 7.2. This was our primary development environment, so all of the code in this book is well tested under Linux.

Linux versions are available for Tomcat, MySQL, and the JDK. You should install all of these programs to their standard locations. You must also modify your profile file so that the classpath and system path are set up properly.

Under Unix, the procedure for changing your profile file varies greatly, depending on which shell you are running. Listing D.1 shows a simple profile for the BASH shell, which is the default shell for Red Hat Linux.

Listing D.1 A Sample .bash_profile File

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:/usr/java/j2sdk1.4.0/bin
JAVA_HOME=/usr/java/j2sdk1.4.0/
CLASSPATH=.:/usr/jar/mm.mysql-2.0.12-bin.jar:/var/tomcat4/common/lib/
```